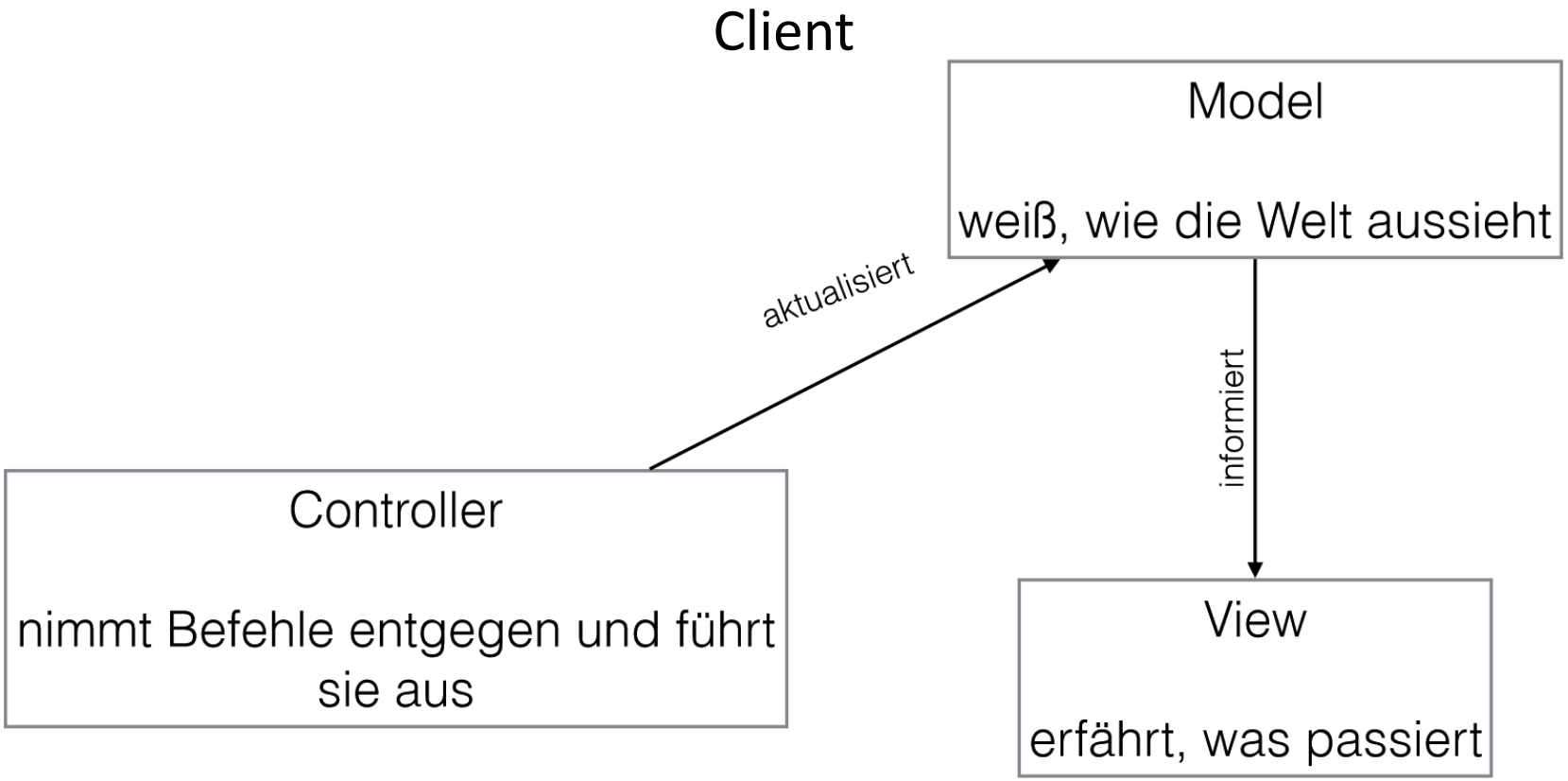
Programmierung einer KI

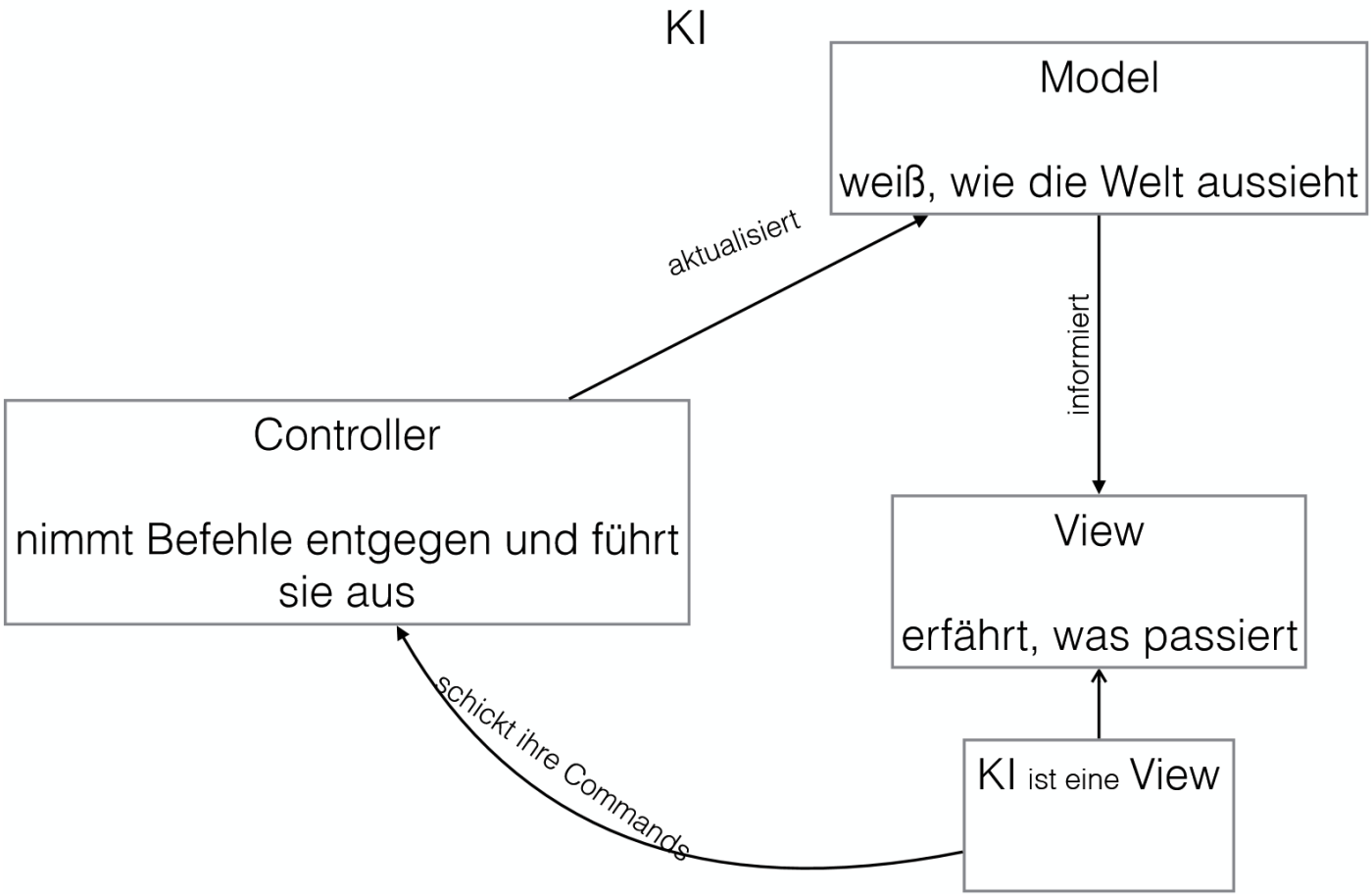
SoPra 2017

Jenny Hotzkow

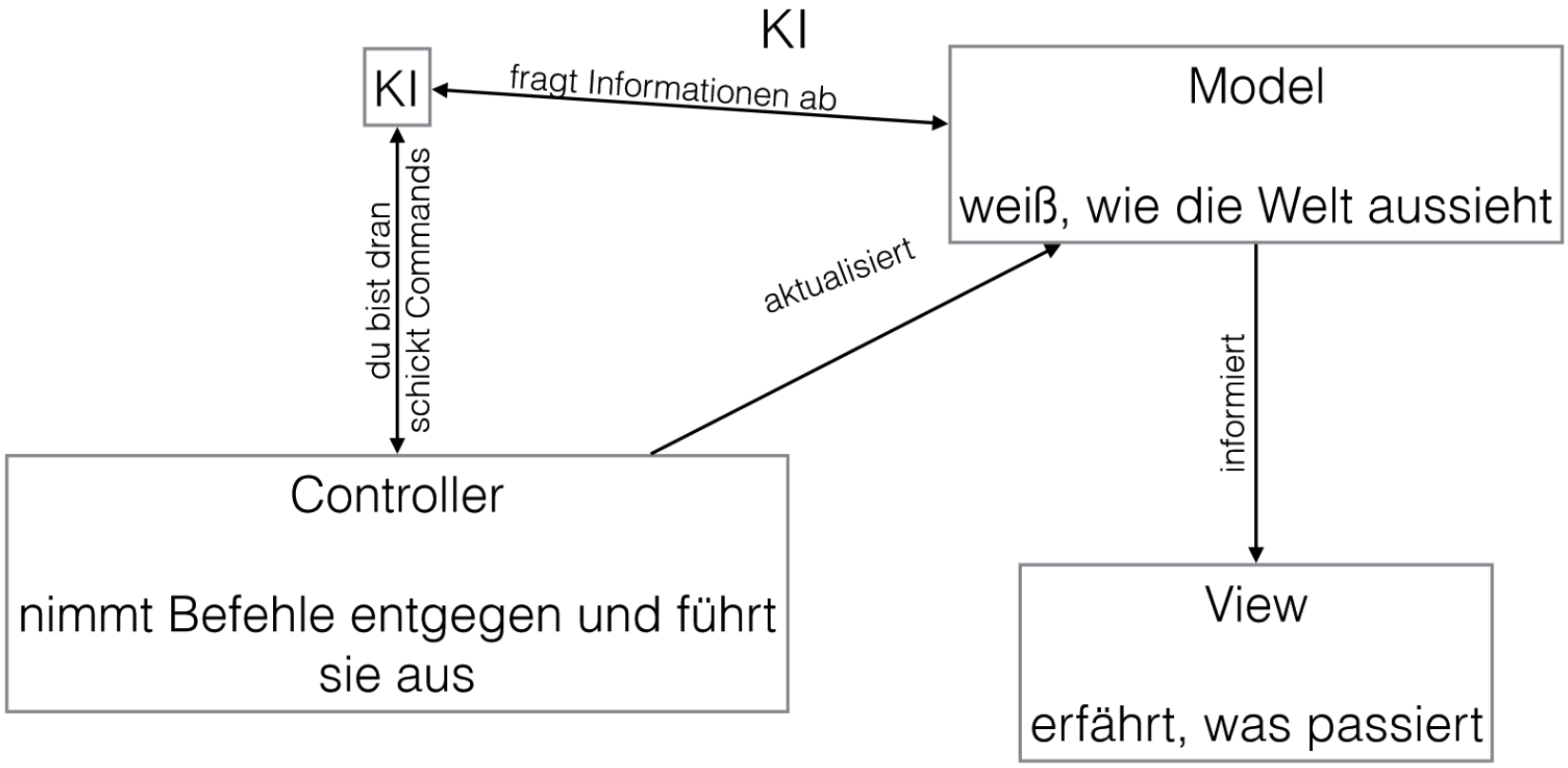
Model – View – Controller



KI Integration in die Softwarearchitektur (1)



KI Integration in die Softwarearchitektur (2)



Command-Pattern

```
// Bestimmung möglicher Aktionen
List<Command> commands =
getPossibleCommands(model, getActorId());

// eine aussuchen
int pos = 0;
Command command = commands.get(pos);
// check Validität (inklusive Crashes)
if(command.isSuitable()){
        send(command);  // und an den Server senden
}
```

Anforderungen an das Model

- Information wann ich dran bin
- Information über die Spielwelt:
  - die Karte, um zu bestimmen wo die KI fahren kann
  - wo sich die anderen Fahrzeuge befinden (bewegliche Hindernisse)
  - wo sich Oma befindet (berechenbares Hindernis)
- Möglichkeit die Commands an den Controller zu übergeben
- Möglichkeit Commands “ausprobieren” zu können

Wie werden Commands simuliert?

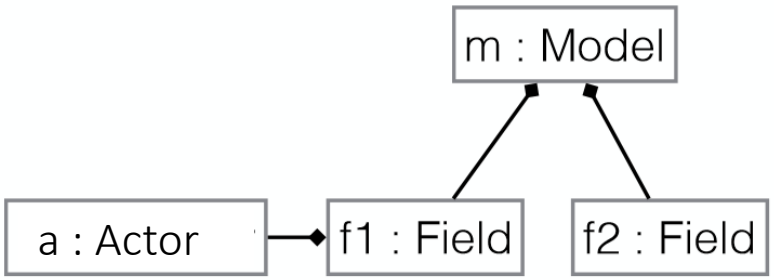
- Einen Zug “ausprobieren”, ohne das Model zu ändern
- Die KI muss wissen, wie die Welt nach dem Zug aussieht, um die Heuristik für diesen Zustand berechnen zu können

Simulieren:

- durch Berechnung der möglichen Command Auswirkungen, ohne sie auf das Model anzuwenden
- den Zug auf einer Kopie des Models ausführe
- im Command-Pattern ein “unexecute()” implementieren

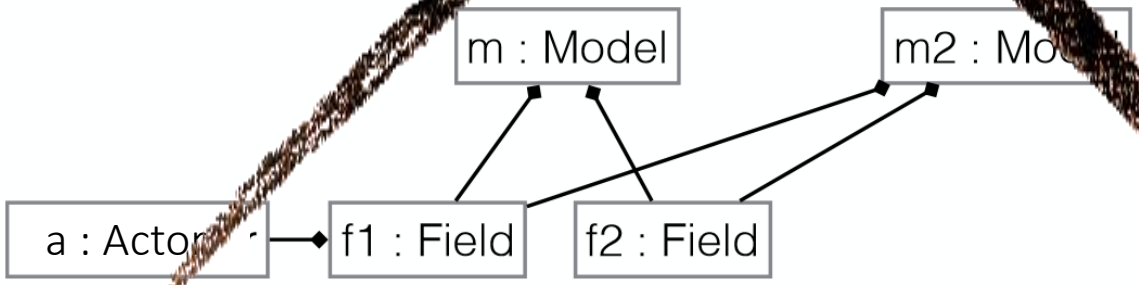
Kopieren des Model

```
Model copy(Model m) {
  Model m2 = new Model();
  m2.f1 = m.f1;
  m2.f2 = m.f2;
  return m2;
}
```



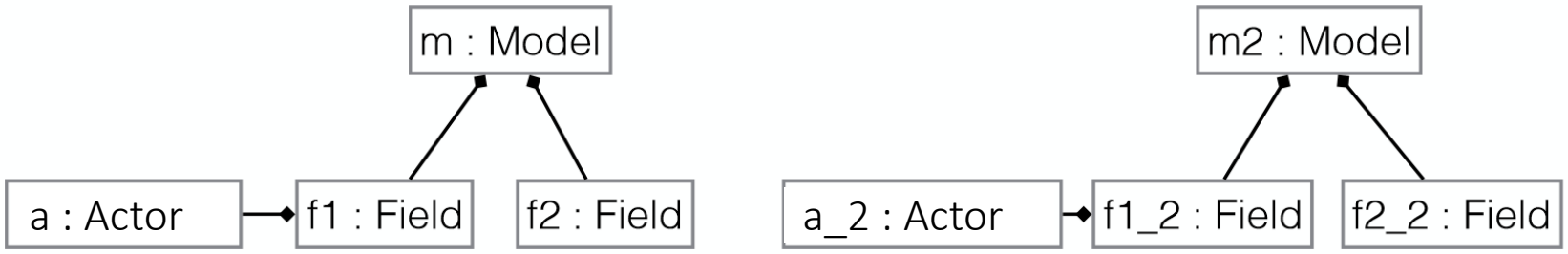
Kopieren des Model

```
Model copy(Model m) {
  Model m2 = new Model();
  m2.f1 = m.f1;
  m2.f2 = m.f2;
  return m2;
}
```

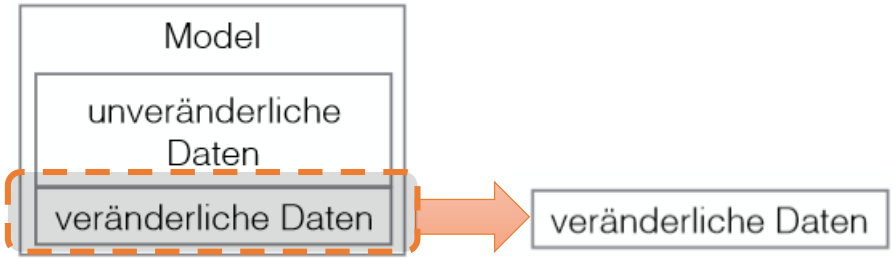


Kopieren des Models

```
Model copy(Model m) {
  Model m2 = new Model();
  m2.f1 = copy(m.f1);
  m2.f2 = copy(m.f2);
  return m2;
}
```



Memento-Pattern



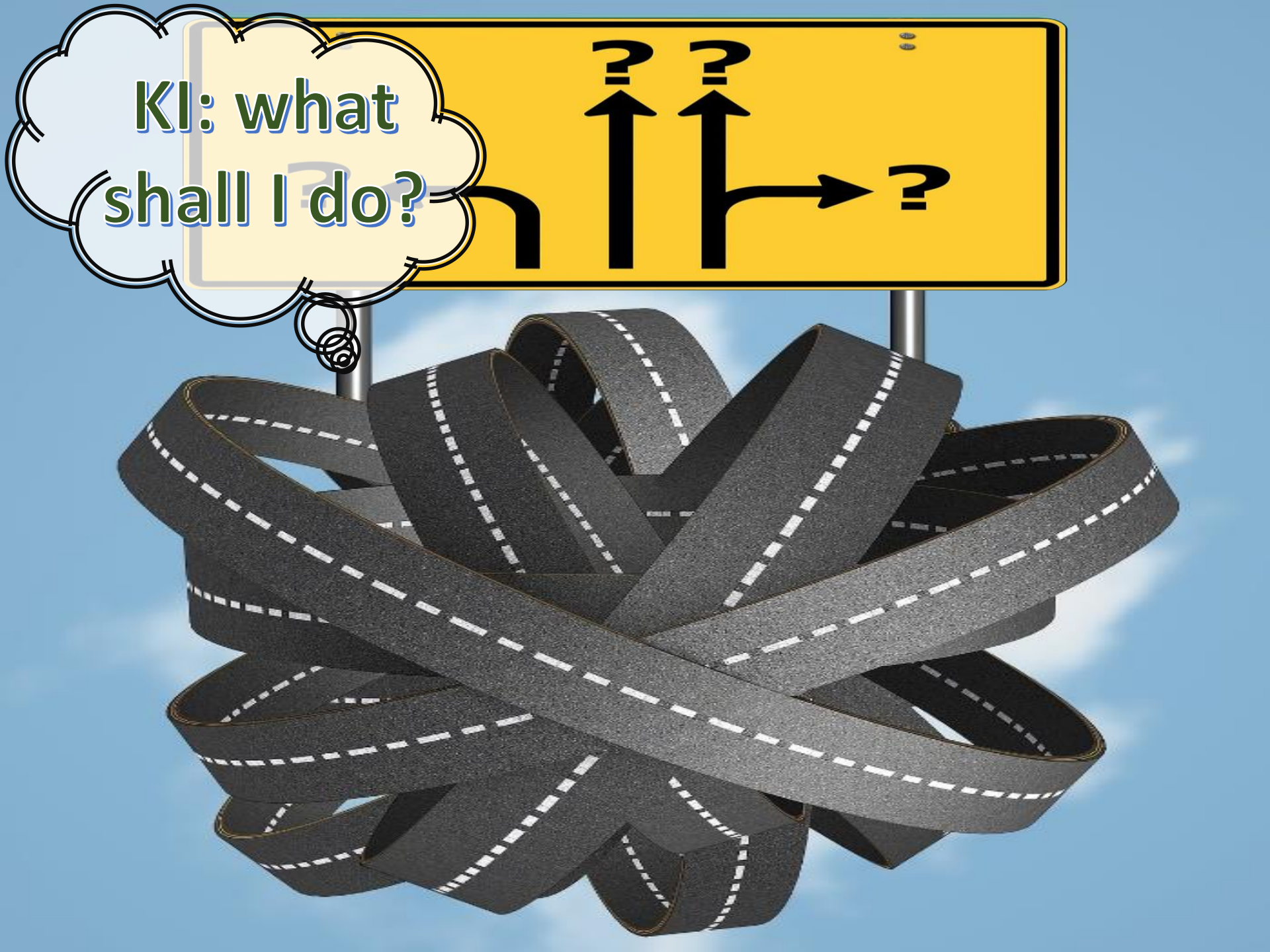
- Kopiere die ***veränderlichen Daten*** (Backup)
- Simulation ändert das Model
- Das Model kann zurückgesetzt werden, indem die veränderlichen Daten mit dem Backup ersetzt werden

Vorteile:

- Leichter zu implementieren als „unexecute()“ (weniger Code!)
- Weniger Daten zum Kopieren als bei voller Kopie

Nachteile:

- Der Entwurf entfernt sich vom intuitiven Objektmodell


KI: what shall I do?

Minimal KI => Schrittweise Verfeinerung

☑***Annahme:*** Wir kennen bereits unsere nächsten Zielkoordinaten in der richtigen Reihenfolge

Minimal KI => Schrittweise Verfeinerung

☑***Annahme:*** Wir kennen bereits unsere nächsten Zielkoordinaten in der richtigen Reihenfolge

☑***Annahme***: Es gibt keine beweglichen Hindernisse (Oma)

Minimal KI => Schrittweise Verfeinerung

☑***Annahme:*** Wir kennen bereits unsere nächsten Zielkoordinaten in der richtigen Reihenfolge

☑***Annahme***: Es gibt keine beweglichen Hindernisse (Oma)

☑***Annahme***: Es gibt keine Öl-, Nagel- oder Sandfelder

Minimal KI => Schrittweise Verfeinerung

☑***Annahme:*** Wir kennen bereits unsere nächsten Zielkoordinaten in der richtigen Reihenfolge

☑Annahme: Es gibt keine beweglichen Hindernisse (Oma)

☑Annahme: Es gibt keine Öl-, Nagel- oder Sandfelder

(1) Eine KI die von einer Zielkoordinate zur nächsten fahren kann

(2) Schrittweise Reduktion der Annahmen, um eine voll Funktionsfähige KI zu erhalten

(3) Nutzung von Pfadsuche oder Checkpoint-Optimierung um die Zielpunkte zu bestimmen

(4) Spezialisierung auf 2 von den 3 Challenges

(1) Bestimmung eines Command für Zielkoordinate(x, y)

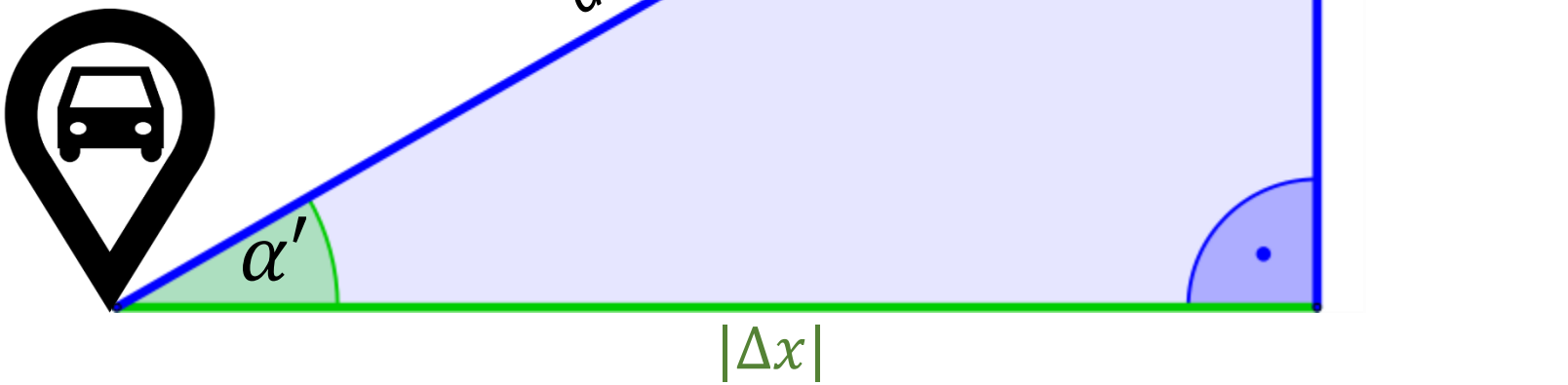
Gesucht: γ,a

1. Berechnung des Bewegungswinkel α' in rad (das ist noch nicht α_{neu})

Kosinussatz:

$$sin\alpha = \frac{Gegenkathete}{Hypotenuse}$$

$$\alpha' = arcsin\frac{|\Delta y|}{distance}$$

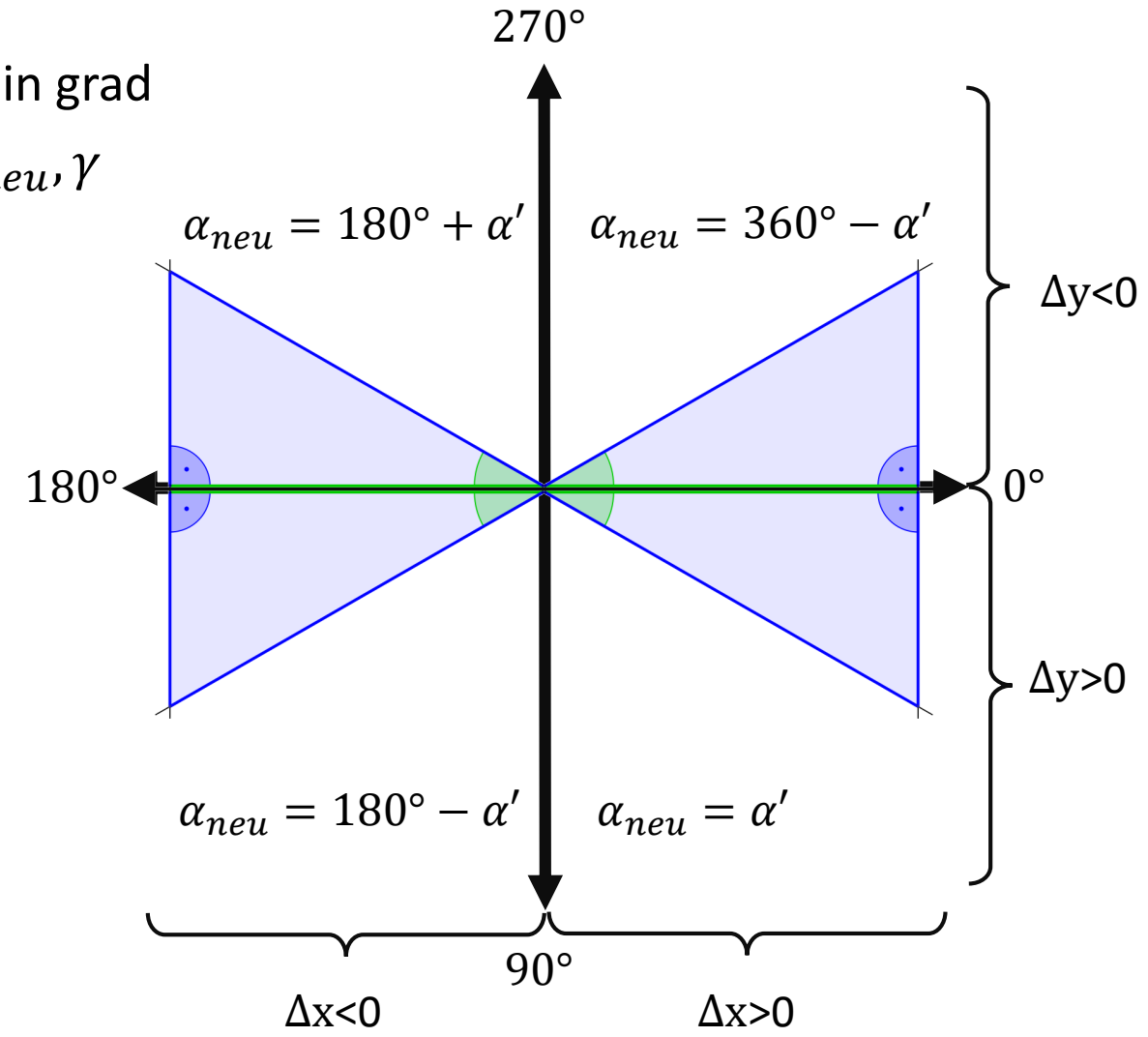


$\Delta y = y - y_0$
$\Delta x = x - x_0$

(1) Berechnung von α_{neu} und damit γ

2. Berechnung α' in grad
3. Berechnung α_{neu}, γ

$$\Delta y = y - y_0$$
$$\Delta x = x - x_0$$



(1) Bestimmung der Beschleunigung a

$$a = \frac{distance - v_{alt} + \mu_l \cdot v_{alt} \cdot t^2}{(0.5 - 0.4 \cdot \mu_r) t^2}$$

- Anhand der berechneten Parameter γ und a, muss die KI das Command bestimmen:
  - Move: $a = 0, \gamma = 0$
  - Turn: $\gamma \neq 0$
  - Accelerate $a > 0$
  - Brake $a < 0$
- Unter Umständen ist eine Kombination von Commands notwendig!

(2) Vermeidung von Hindernissen

Es gibt viele (einfache) Möglichkeiten.
Seien Sie kreativ!

(3) Bestimmung der Zielpunkte

- Optimierung eines Pfades aus Checkpoints
- Path Following steering behavior: Lenke ein, wenn das Fahrzeug von der Straße (oder anderen befahrbaren Feldern) abkommen würde
- Bestimme eine Fahrlinie auf der Strecke und wähle Zielpunkte auf dieser Linie in Abhängigkeit der pro Zug zurückgelegten Strecke
- Berechnung optimaler Zielpunkte in Bezug auf die spezifische Challenge und den damit verbundenen optimalen Commands

(4) Challenge Spezialisierung

Ökonomisch:

- Frühes Einlenken benötigt weniger Turns
- Move ist Teuer!

Zeitrennen:

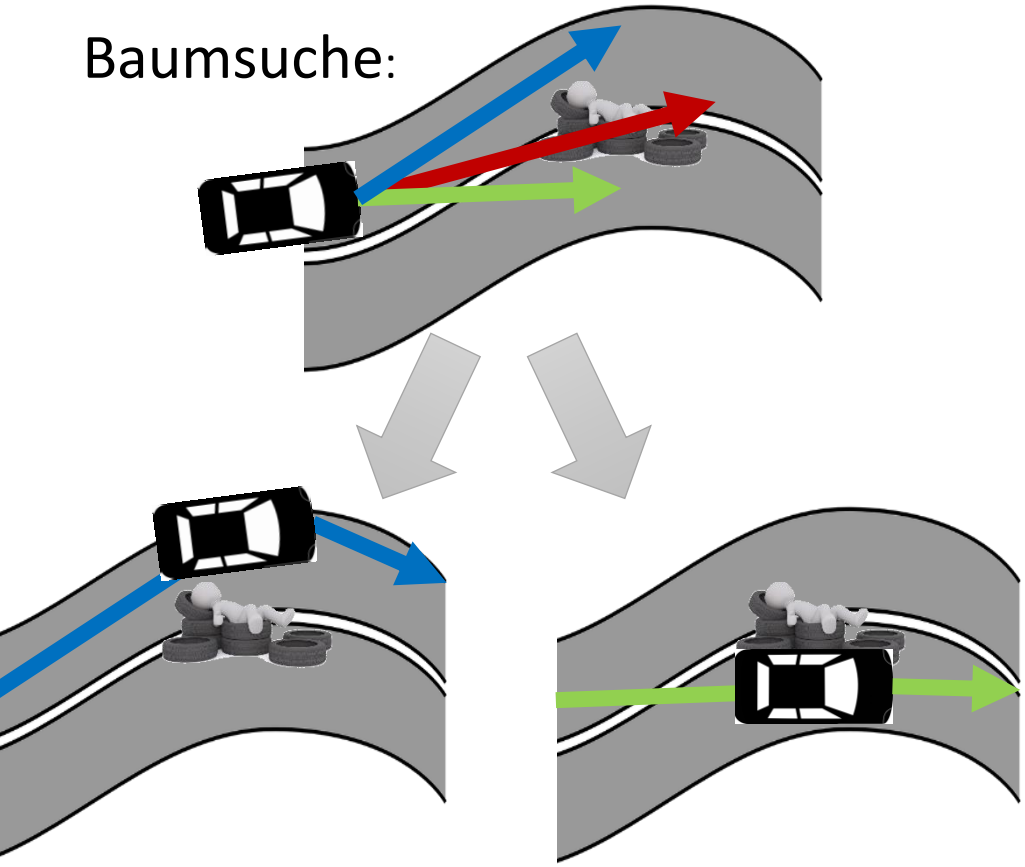
- Frühes Einlenken benötigt weniger Turns
- Anhalten sollte vermieden werden!

Kollisionsvermeidung:

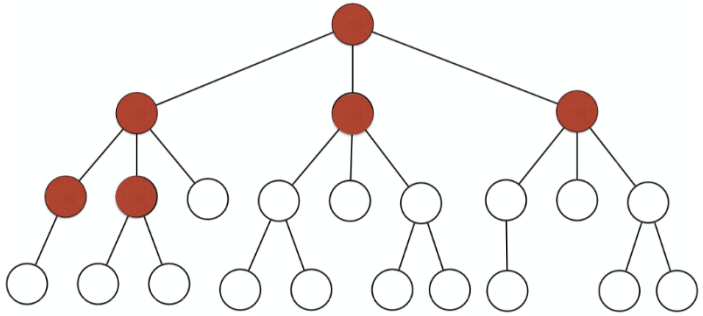
- Pfadberechnung oder Breiten/Tiefensuche möglicher Commands

KI Entscheidungsfindung

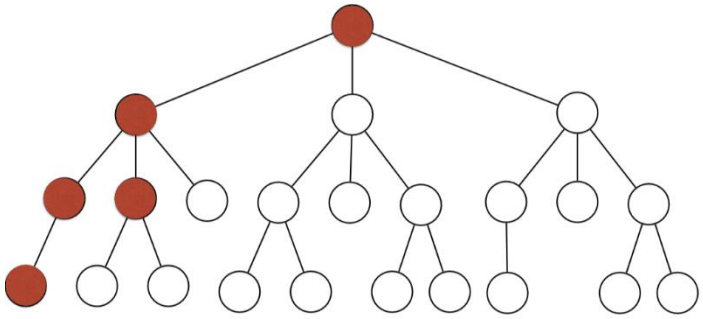
Baumsuche:



Breitensuche:



Tiefensuche:



KI Entscheidungsfindung

Regelbasiert:

- z.B. Move, wenn möglich Endposition ist auf einem Straßenfeld
- sonst Turn links/rechts
- …

Baumsuche:

„wie gut ist ein Zustand Bewertung“ (Heuristic):

- Distanz
- Anzahl von Commands
- je nach Challengetype leicht variierend

Breitensuche:

- z.B. wenn Turn notwendig ist, untersuche mögliche Alternativen für optimale Endposition

Tiefensuche:

- z.B. zum Konstruieren der Pfade von erreichbaren Checkpoints/Zielpunkten

Machine Learning:

- Lerne aus vorangegangenen Zügen (Runden) um die Strategie zu optimieren

Baumsuche Algorithmus

- Queue q
- lege den aktuellen Zustand in q
- solange noch Zustände in q sind:
  - nehme den nächsten Zustand aus q
  - erzeuge alle Kind-Zustände
  - lege die Kind-Zustände in q
- wenn keine Zustände mehr in q sind:
  - der beste Zustand ist der, wo ich hin will

Mehrere Heuristiken kombinieren

Gewichtete Summe:

$$h_c(m) = \alpha_1 h_1(m) + \alpha_2 h_2(m) + ... + \alpha_n h_n(m)$$

Steuern, welche Heuristik am Wichtigsten ist

z.B.

Für Ökonomisches Fahren ist der Energieverbrauch entscheidend, während im Zeitrennen die Distanz zum Ziel im Vordergrund steht.

In Beiden Fällen sollte die Anzahl von Turn Commands minimiert werden.

Heuristiken normalisieren

- Heuristiken zwischen 0 und 1 bringen:

$$h'(m) = \frac{h(m)}{\max h}$$

- Heuristiken auf eine gemeinsame Einheit bringen:
  - statt: Entfernung zum Zielfeld

    Wie viele Commands brauche ich noch da hin?
  - statt: verbleibendes Tankvolumen

    Wie viele (welche) Commands kann ich noch ausführen?

Achtung: Zeit ist Limitiert (Timeout)

- Suchbereich einschränken:
  Wenn ich ein Kind mit maximalem/minimalem Wert gesehen habe, muss ich die anderen Kinder nicht mehr angucken

- Suchtiefe beschränken:

  Beispiel: Ich gucke 5 Zustände in die Zukunft

- Heuristik mit anderen Informationen:
  - Anzahl von Wegpunkten/Strecke zwischen Wegpunkten zum Ziel
  - Voraussichtlicher Gesamtverbrauch zum Erreichen der Wegpunkte

- Machine Learning:
  - Sammeln Sie möglichst viele Zustände einschließlich Ergebnisse
  - Lassen Sie ein neuronales Netz (oder einen anderen Classifier) lernen, ob nach einem bestimmten Zustand ein Sieg kommt

Optionen für eine KI

Regelbasiert:

- Gute Kontrolle über KI-Verhalten
- Schlechte Anpassung an unerwartete Situationen

Heuristische Baumsuche:

- KI-Verhalten hängt stark von der Heuristik ab
- sehr erfolgreich (Schach, Mühle, …)

Machine Learning:

- Keine Kontrolle über KI-Verhalten (quasi keine Möglichkeit zum debuggen)
- Extrem erfolgreich