hochschule aschaffenburg
university of applied sciences

Game Mechanics

Repetitorium – Teil 1

Workflow Spielentwicklung

- Design Document
- Whiteboxing
- Playtesting

Aufgabe: Design Document für Diablo Clone

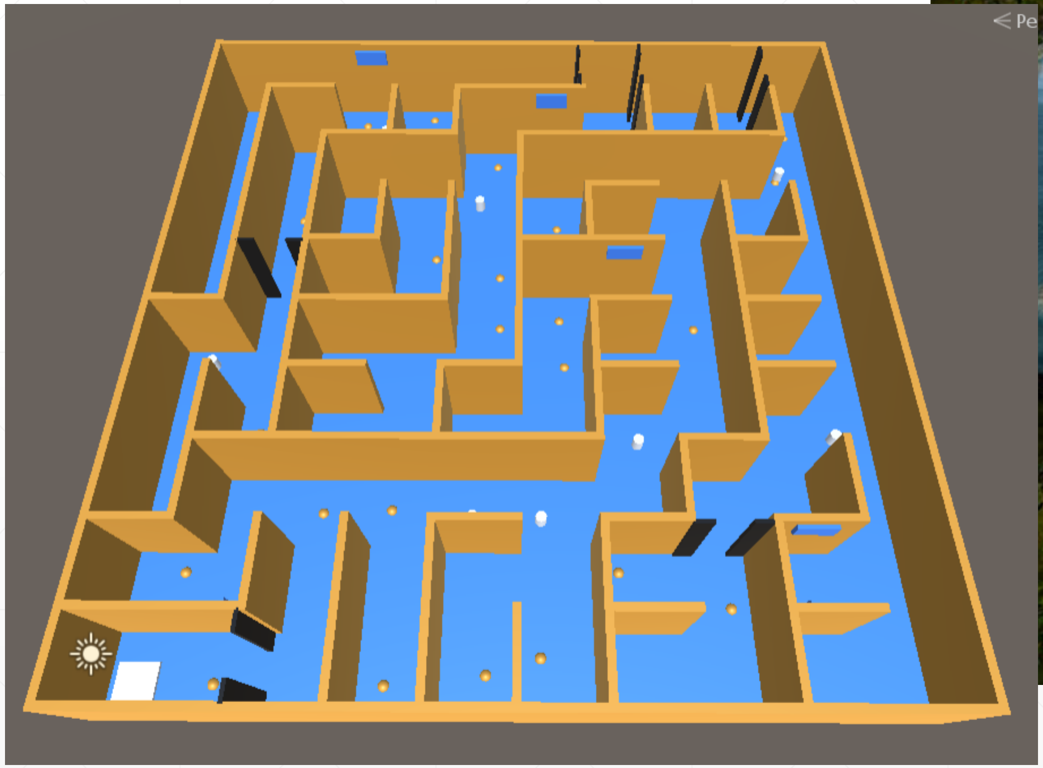
- Grundidee in 3 – 5 Sätzen
- Spielraum
  - Stichpunkte
  - Orte (Funktionen)
  - Resultierende Spielmechaniken
- Spielzeit
- Objekte: NPCs, Gegenstände …
- Attribute: Stats, Erfolg/Misserfolg, …
- Statusänderungen: Anzeige, Geheimnisse …

(Quelle: Wikipedia)

- Labyrinth Generierung

Generatives Design

Procedural Level Design



Binary Tree Maze Generator

- Stateless
- Strong bias: Gegenüberliegende Ränder
- Startpunkt (0/0)
- Pro Zelle entweder Durchgang Rechts oder Durchgang Oben

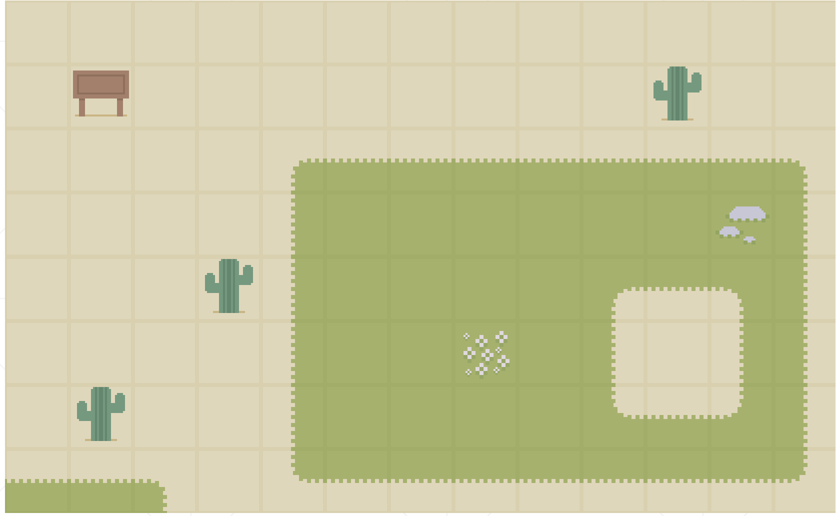
Recursive Backtracker Maze Generator

- Startpunkt beliebig
- Kein bias
- States („Visited“)
- Durchgang zu zufälligem, nicht besuchtem Nachbarn
- Von dort weiter (Rekursiv), bis keine unbesuchten Nachbarn
- Dann zurück, und zurück bis zum Startpunkt

Procedural Level Design

Tilemaps

- 2D
- Einfache Design Tools
- Procedural
  - Gelände
  - Assets
  - Monster
  - Quests
  - …



(Quelle: unity.com)

```
if (Zufall > Wahrscheinlichkeit[Asset.Baum]) {
    place(Baum, xpos, ypos);
}
```

Aufgabe

- Entwerfen Sie ein perfektes Labyrinth mit 5x5 Feldern.

Gegner KI – Teil 1

- Zustandsautomaten
- Behavior Trees

Intelligenz implementieren

Zustandsautomaten

Abfolge von Zuständen (States) und Übergängen (Transitions)

auch: Finite State Machine, FSM

Behaviour Tree

Organisation von Zuständen (Behaviors) in einer Baumstruktur

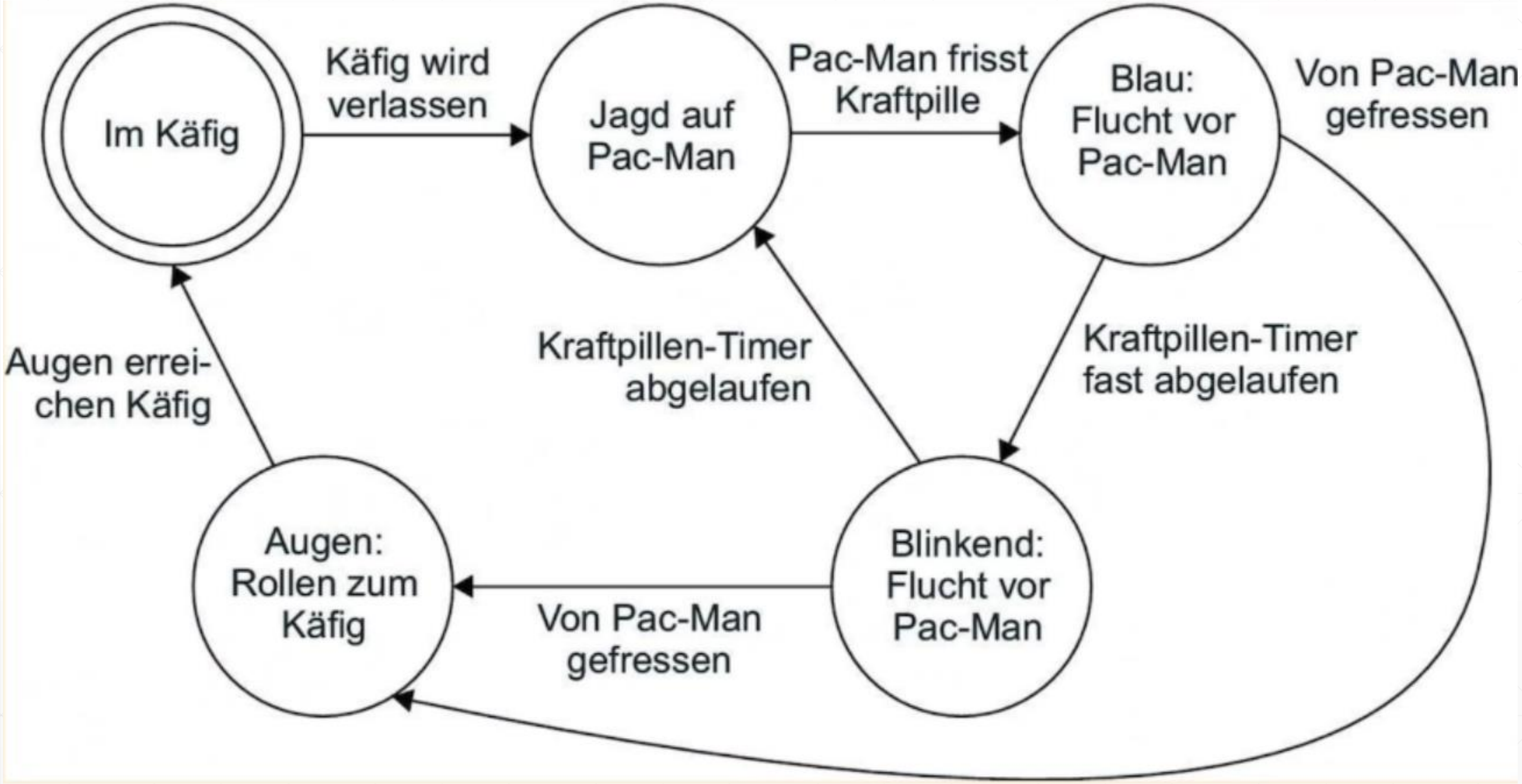
Entscheidungstheorie

Bewertung der Spielsituation auf Basis bekannter Informationen

Spieltheorie

Bewertung auf Basis bekannter Informationen und Schätzungen der anderen Beteiligten

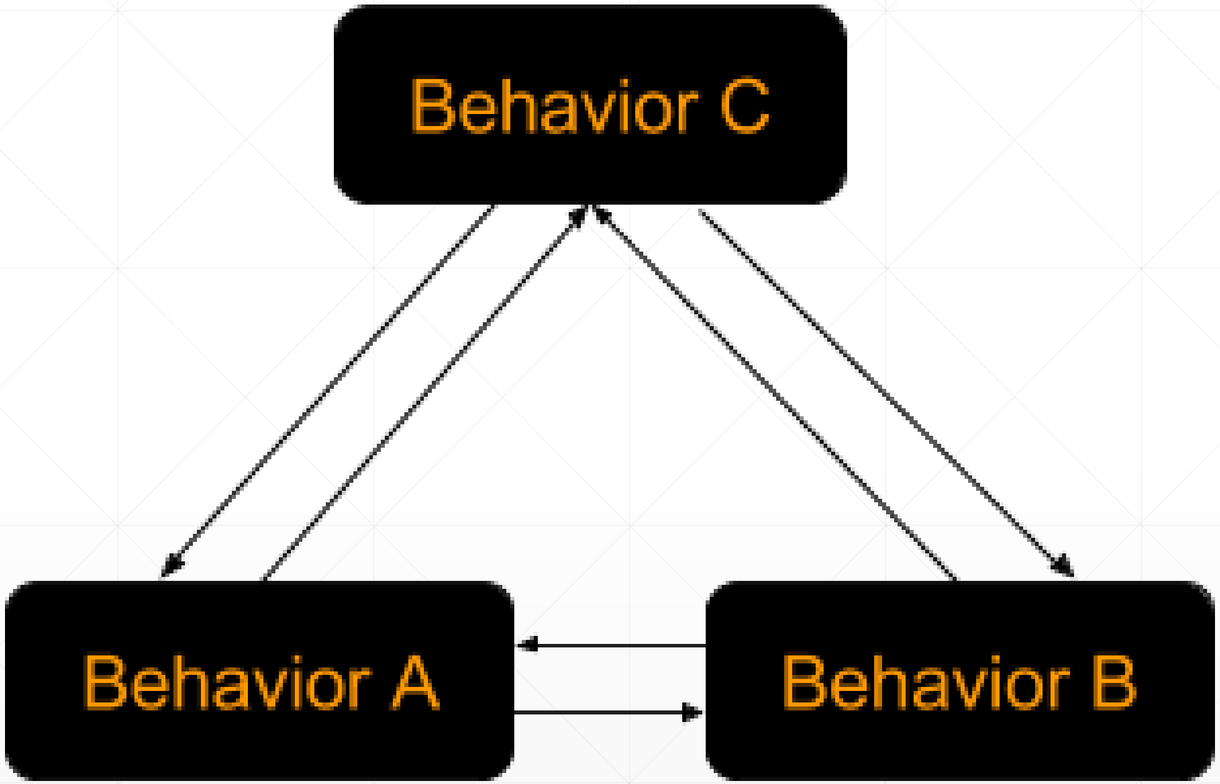
Zustandsautomaten



(Quelle: Shell, Die Kunst des Game Designs)

Zustandsautomaten

- Einfacher Aufbau
  - Zustände definieren
  - Übergänge definieren
- Gut geeignet für einfache KI
- Problem: Wird schnell unübersichtlich
- Unity: Playmaker



(Quelle: Rasmussen, J: Are Behavior Trees a Thing of the Past?, gamasutra.com, 27.04.2016)

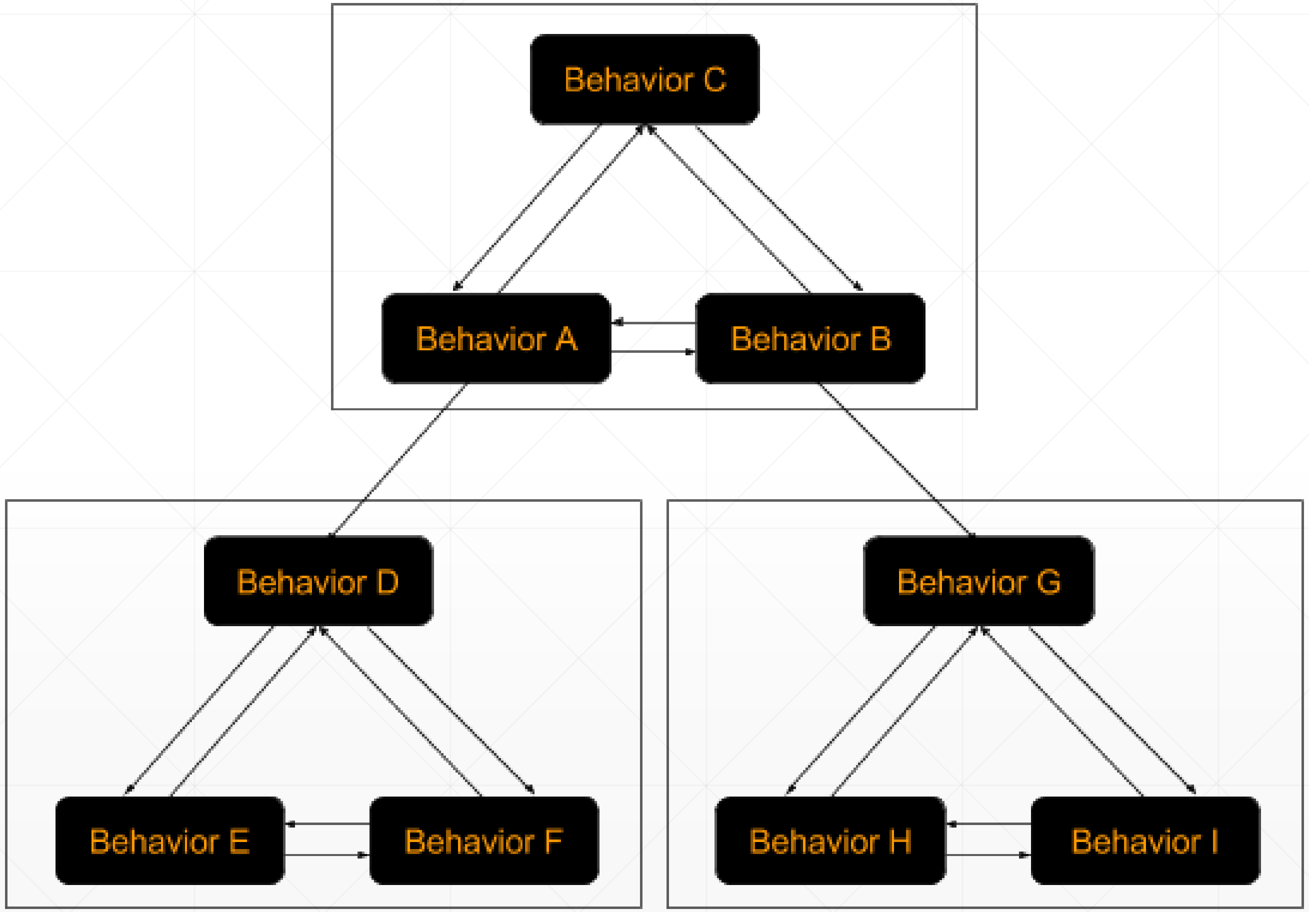
Aufgabe

- Nachteile von Zustandsautomaten?
- Möglichkeiten, diese Nachteile zu umgehen (mit Zustandsautomaten)?

Organisation komplexerer Automaten

- ➔ Hierarchische Struktur

- ➔ Beispiel:
  - ◆ Behavior A: Schlafen
  - ◆ Behavior B: Studieren
  - ◆ Behavior C: Freitzeit



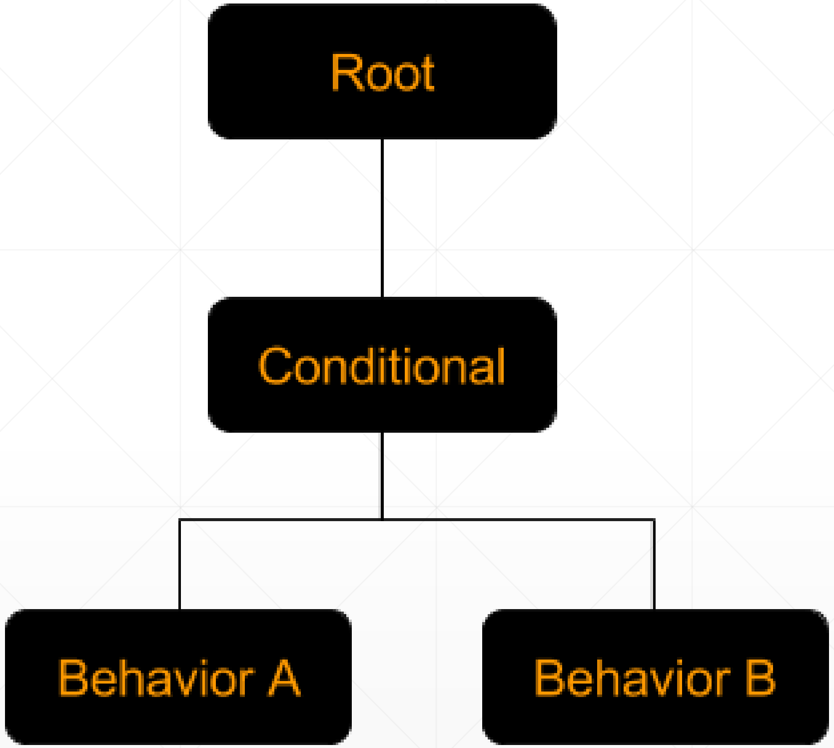
(Quelle: Rasmussen, J: Are Behavior Trees a Thing of the Past?, gamasutra.com, 27.04.2016)

Behavior Trees

- ➔ Aufbau nicht ganz so intuitiv
- ➔ Ablaufrichtung
- ➔ Sequenzen, Bedingungen, Repeater …

- ➔ Gut geeignet für komplexere Gegner-KI

- ➔ Unity: Behavior Designer

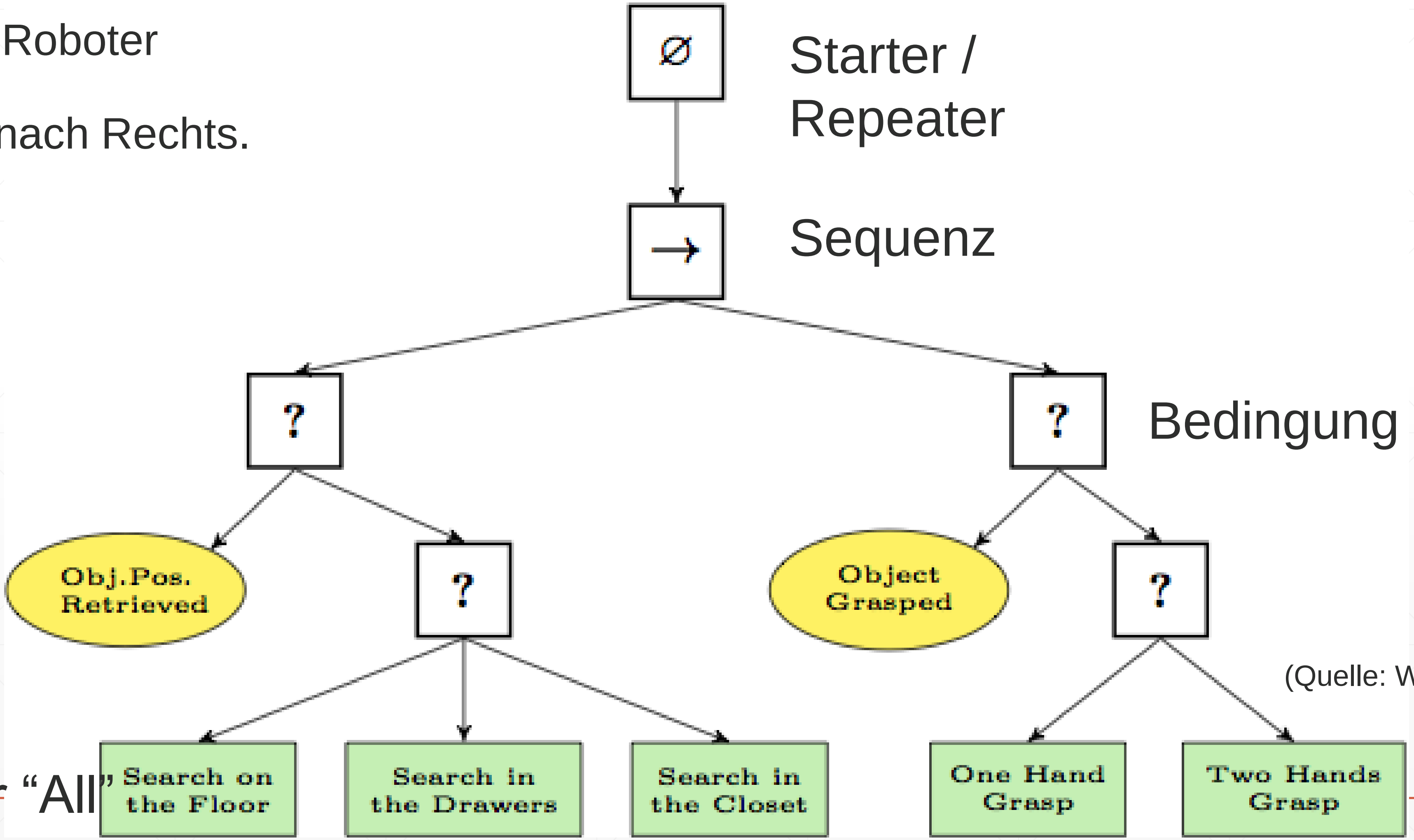


(Quelle: Rasmussen, J: Are Behavior Trees a Thing of the Past?, gamasutra.com, 27.04.2016)

Elemente eines Behavior Trees

Beispiel: Suchverhalten Roboter

Abarbeitung: Von Links nach Rechts.



Starter / Repeater

Sequenz

Bedingung

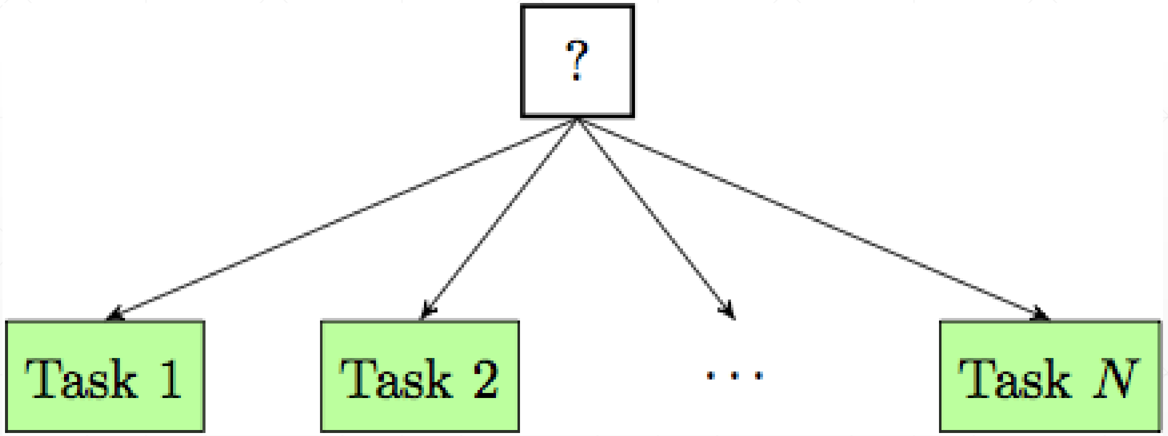
(Quelle: Wikipedia)

“Any” or “All”

Bedingung (Selector)

- ➔ Von Links nach Rechts
- ➔ Erster erfolgreicher Task
- ➔ Dann: Selector = true

```
for i from 1 to n do
    childstatus ← Tick(child(i))
    if childstatus = running
        return running
    else if childstatus = success
        return success
end
return failure
```

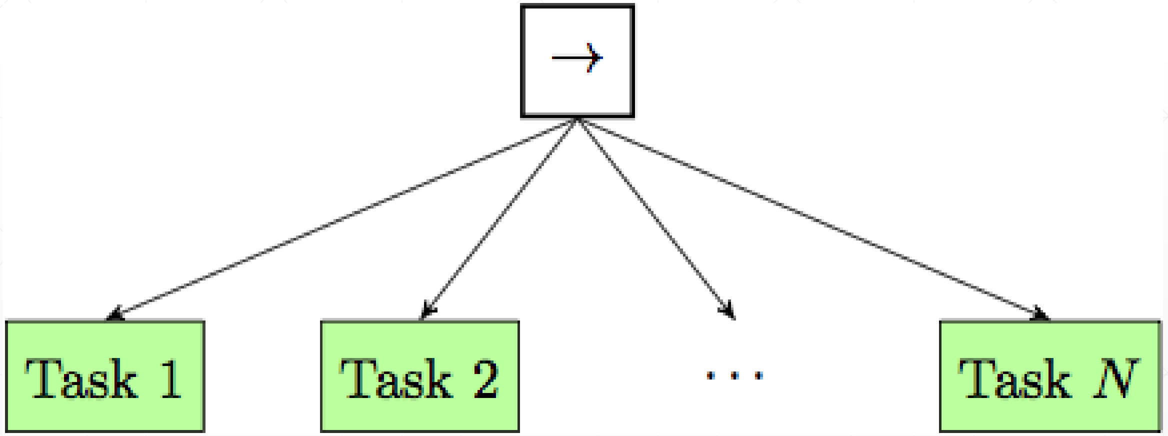


(Quelle: Wikipedia)

Sequenz

- ➔ Von Links nach Rechts
- ➔ Nicht erfolgreicher Task, dann Sequenz = false
- ➔ Alle Tasks erfolgreich, dann Sequenz = true

```
for i from 1 to n do
    childstatus ← Tick(child(i))
    if childstatus = running
        return running
    else if childstatus = failure
        return failure
end
return success
```

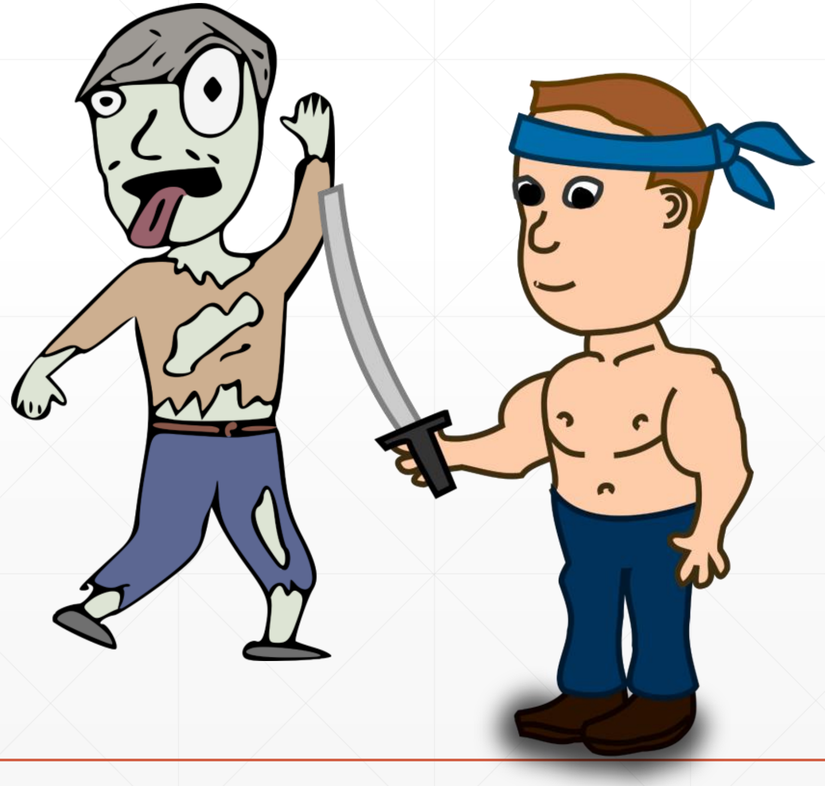


Aufgabe

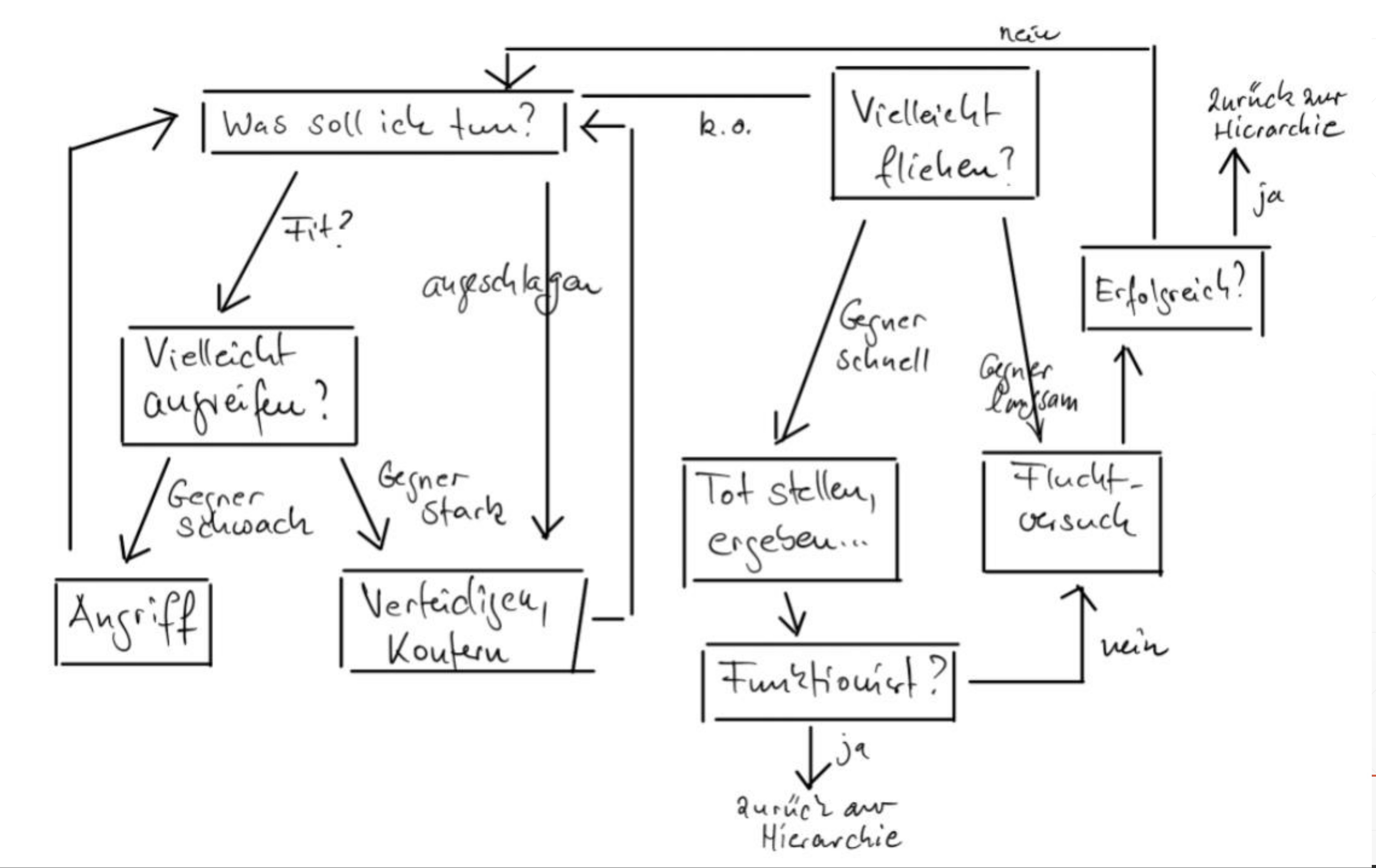
Entwerfen Sie eine Hierarchical FSM und einen Behavior Tree für einen NPC, der ein Survival-Spiel bevölkern soll.

Beispiele für NPCs:

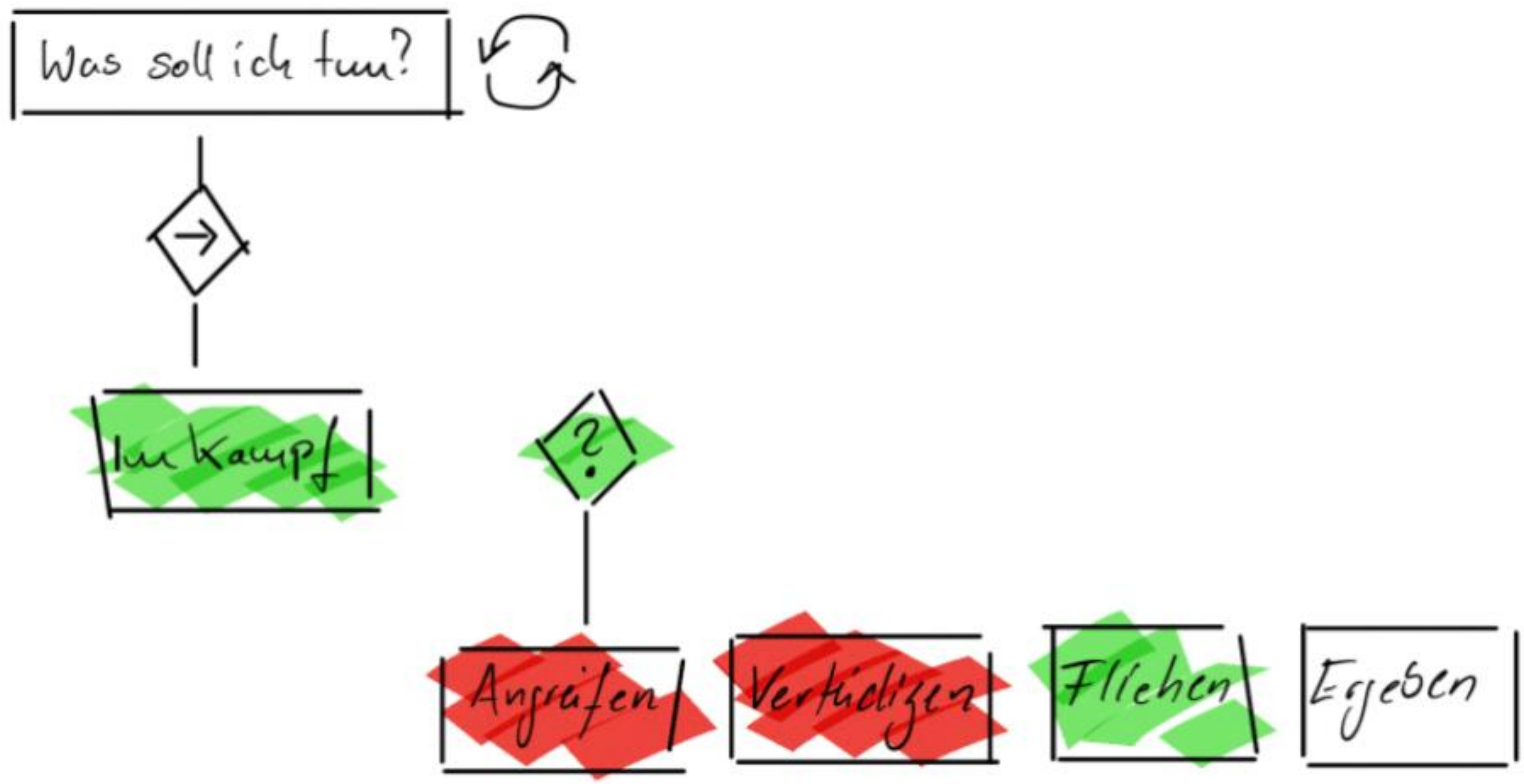
- Überlebender
- Infizierter
- Zombie, Monster



FSM: Kampfhandung



BT: Kampfhandlung



Gegner KI – Teil 2

- Entscheidungstheorie
- Spieltheorie

Entscheidung unter Ungewissheit

D.h. p(i) = ?

<u>Entscheidungsregeln</u>

- MaxiMin $$\phi(a_i) = \max_j\left(\min_i(u_{ij})\right)$$
- MaxiMax $$\phi(a_i) = \max_j\left(\max_i(u_{ij})\right)$$
- Hurwicz Regel $$\phi(a_i) = \max_j\left(\alpha \cdot \max_i(u_{ij}) + (1-\alpha) \cdot \min_i(u_{ij})\right)$$
  $$\alpha \in [0,1]$$
  $$\alpha = 1 \rightarrow \text{ starker Optimist; } \alpha = 0 \rightarrow \text{ starker Pessimist}$$

MaxiMin

		Gegner		
		stark	*normal*	*schwach*
Aktion	*Angreifen*	**-2**	**1**	**2**
	Abwarten	**-1**	**0**	**1**
	Fliehen	**0**	**-1**	**-2**

MaxiMin
„risikoavers“

Min = -2

Min = -1

Min = -2

Max = -1

Abwarten

MaxiMax

		Gegner		
		stark	*normal*	*schwach*
Aktion	*Angreifen*	**-2**	**1**	**2**
	Abwarten	**-1**	**0**	**1**
	Fliehen	**0**	**-1**	**-2**

MaxiMax
„risikoaffin“

Max = 2

Max = 1

Max = 0

Max = 2

Angreifen

Hurwicz

		Gegner		
		stark	*normal*	*schwach*
Aktion	*Angreifen*	-2	1	2
	Abwarten	-1	0	1
	Fliehen	0	-1	-2

MaxiMin „risikoavers“	**MaxiMax** „risikoaffin“	**Alpha = 0,5**
Min = -2	Min = 2	→ 0
Min = -1	Min = 1	→ 0
Min = -2	Min = 0	→ -1
Max = -1	Max = 2	Max = -1
Abwarten	**Angreifen**	**Angreifen od. Abwarten**

Beispiel: „Vor dem Kampf“

Schätzung (kontinuierlich)		Gegner		
		stark (p=50%)	*normal (p=30%)*	*schwach (p=20%)*
Aktion	*Angreifen*	**-2**	**1**	**2**
	Abwarten	**-1**	**0**	**1**
	Fliehen	**1**	**-1**	**-2**

Entscheidungsmatrix (Vorgabe)

Entscheidung unter Unsicherheit (Risiko)

μ-σ-Regel

$$\phi(a_i) = \max\left(\Phi\left(\mu_i, \sigma_i\right)\right)$$

wobei:

$$\Phi(\mu_i, \sigma_i) = \mu_i + \alpha \cdot \sigma_i = \text{Präferenzfunktion}$$

α repräsentiert Typ des Entscheiders:

$\alpha > 0 \longrightarrow$ Entscheider ist risikofreudig

$\alpha < 0 \longrightarrow$ Entscheider ist risikoavers

$\alpha = 0 \longrightarrow$ Entscheider ist risikoneutral

Entscheidung unter Risiko

		Gegner					
		stark (p=50%)	*normal (p=30%)*	*schwach (p=20%)*	μ	σ	**Alpha = 1**
Aktion	*Angreifen*	-2	1	2	-0,3	1,7	→ 1,4
	Abwarten	-1	0	1	-0,3	0,8	→ 0,5
	Fliehen	1	-1	-2	-0,2	1,2	→ 01,0

Max = 1,4

Angreifen

Rechenhilfe

1. $$\mu = \sum_i p_i x_i$$

2. $$\sigma^2 = \sum_i p_i \left(x_i^2 - \mu^2\right)$$

Spieltheorie

- Beispiel: Der Game Designer plant, welche Verhaltensweisen optimal sind.
  - Mehrere Spieler spielen
  - Auszahlungen bekannt

Übung

		Spieler 2	
		links	*rechts*
Spieler 1	*oben*	5; 4	0; 1
	mittig	4; 2	2; 7
	unten	3; 6	3; 1

Beispiel

		Spieler 2: links	Spieler 2: rechts
Spieler 1	oben	5; 4	1; 1
	mitte	4; 2	2; 7
	unten	3; 6	3; 1

Wiederholung: Optimalitätskonzept

Zentrales Optimalitätskonzept der Spieltheorie:

Eine Strategiekombination ist dann optimal, wenn keiner der Spieler seine Strategie nachträglich noch ändern wollen würde, nachdem er erfahren hat, welche Strategien die anderen Spieler gewählt haben.

D.h. die Strategien aller Spieler sind jeweils beste Antworten aufeinander.

Bezeichnung:

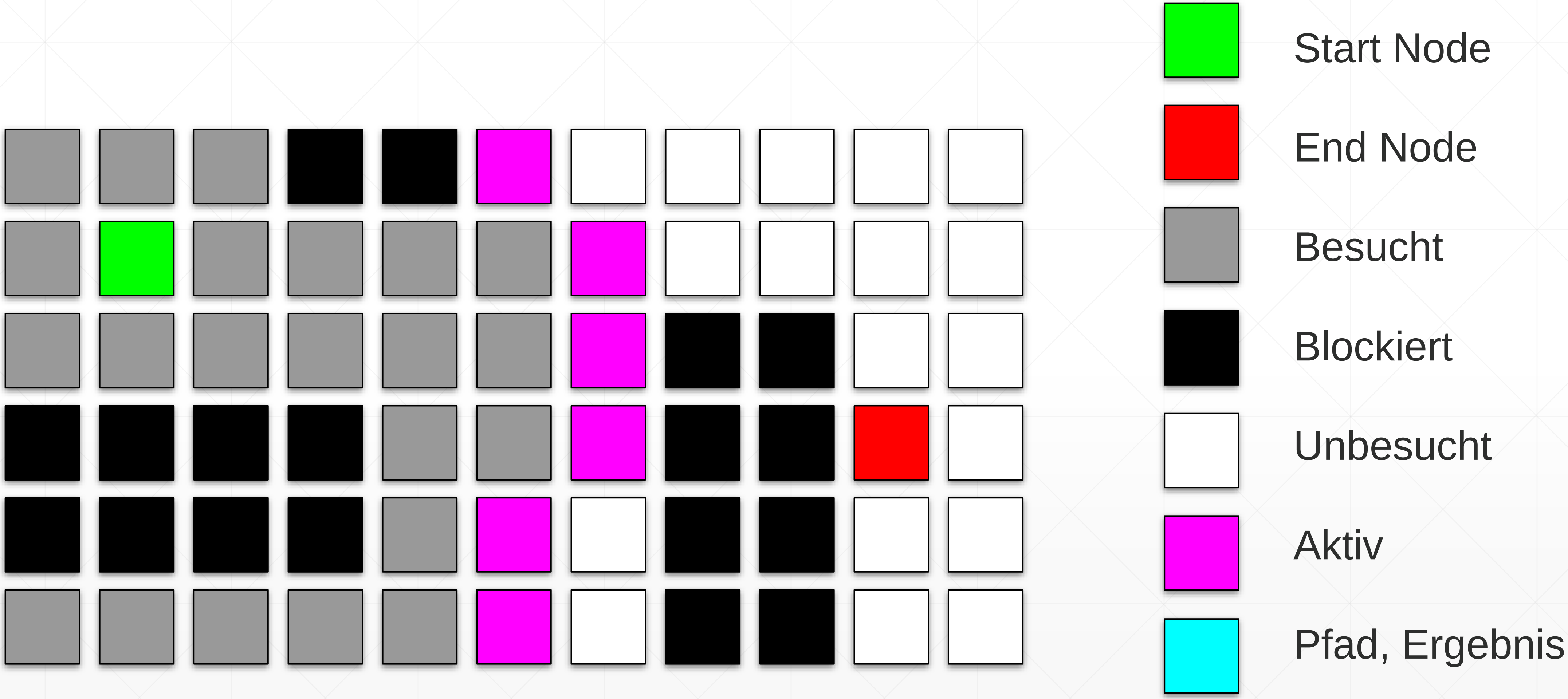
Eine solche Strategiekombination wird auch als Nash-Gleichgewicht bezeichnet.

Wegfindung

1. Breitensuche (Breadth First Search)

- Queue: FIFO
- Findet eine Lösung, solange Ziel-Knoten erreichbar (bspw. Perfekte Labyrinthe)

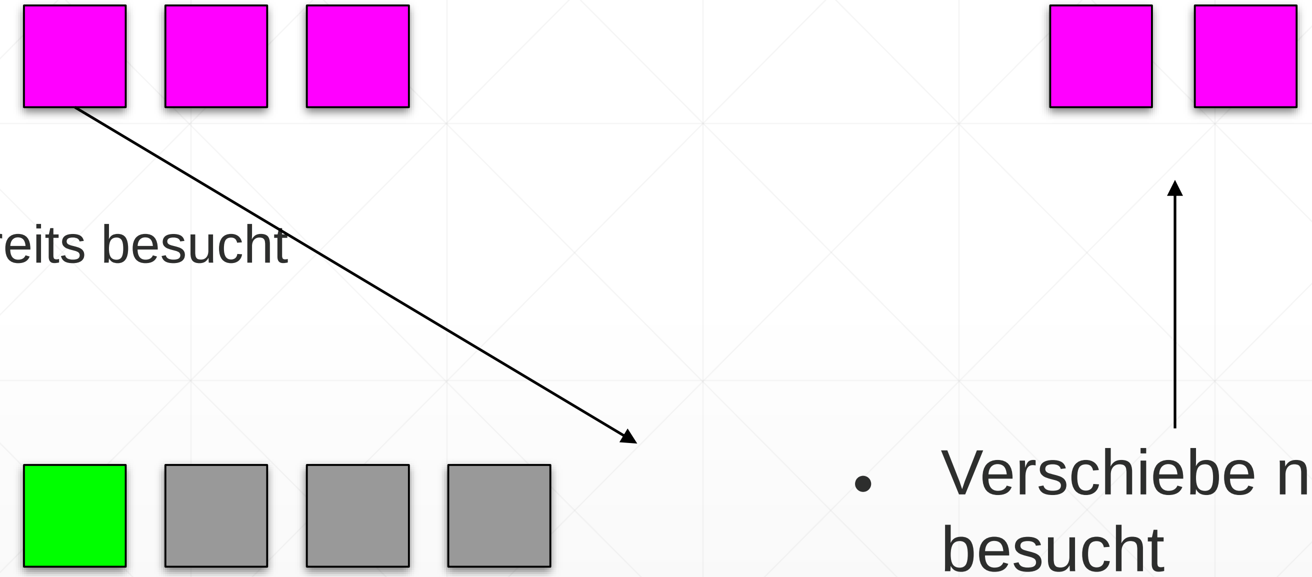
Bezeichnungen



Propagieren aktiver Knoten

Queue

Bereits besucht



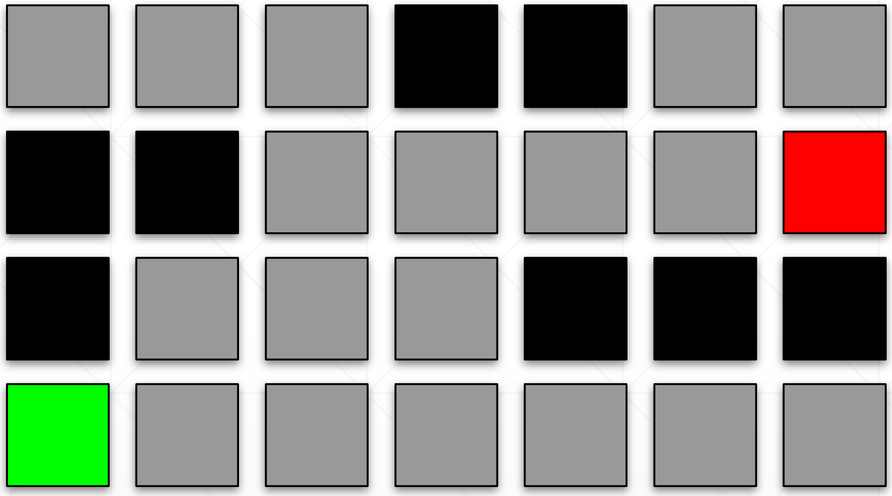
- Merke Vorgängerknoten
- Prüfe, ob Ziel erreicht

- Verschiebe nach bereits besucht
- Suche nächste Nachbarn
- Setze nächste Nachbarn in Queue

Weitere Algorithmen

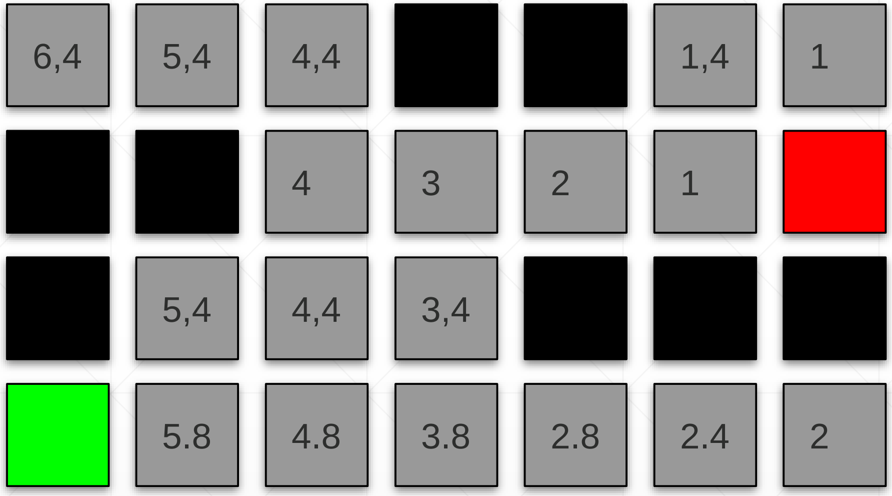
- Dijkstra
- Greedy Best First Search
- A*

Aufgabe (Greedy Best First Search)



Bestimmen Sie das Benchmark

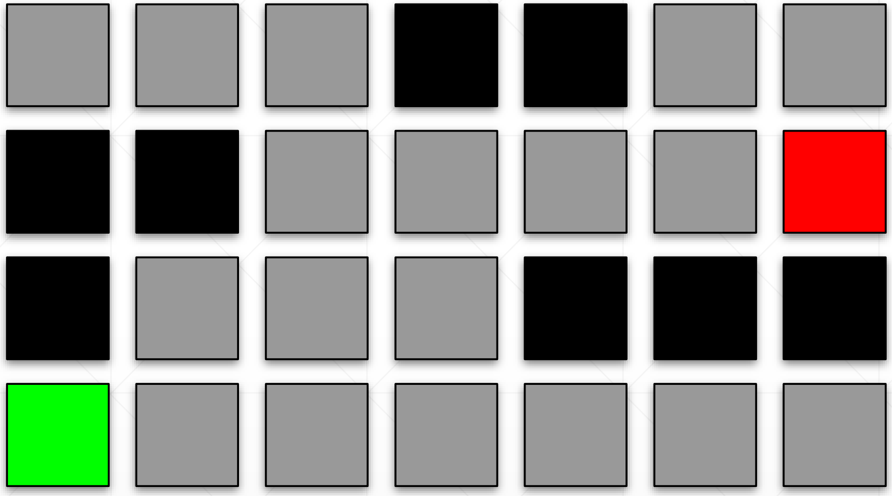
Aufgabe (Greedy Best First Search)



Luftlinie!

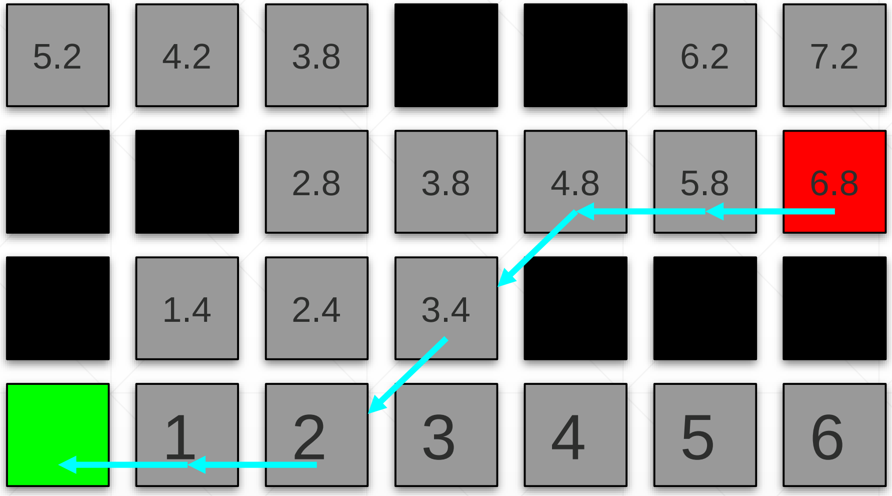
= Schätzung

Aufgabe (Dijkstra)



Bestimmen Sie das Benchmark

Aufgabe (Dijkstra)



Bestimmen Sie das Benchmark