

# Gegner-KI

**Zustandsautomaten  
Behaviour Trees  
Entscheidungstheorie  
Spieltheorie**

# Gegner-KI

Wie intelligent ist intelligent?

Beispiele für “intelligentes” Verhalten von Computerspielern?

Begriffe:

Computerspieler	Die “Intelligenz” hinter dem Spiel, u.U. nicht sichtbar
Gegner, Mob, Boss	Einzelne Instanzen “autonomer”, präsenter “Objekte”

# 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

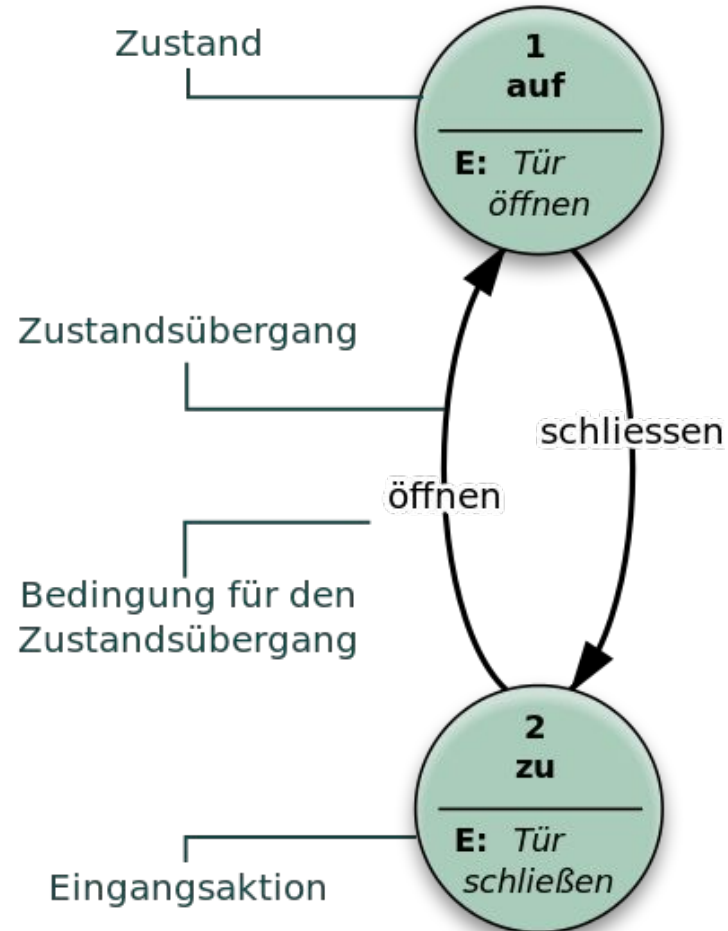
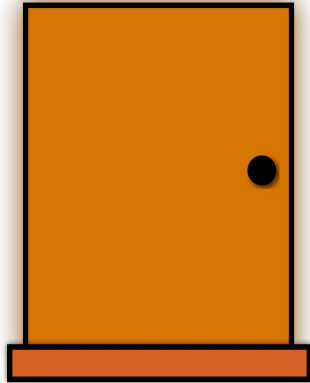
Bewertung auf Basis bekannter Informationen und Schätzungen der anderen Beteiligten

# Endliche Automaten

- Formale Beschreibung eines Algorithmus
- Endliche Anzahl möglicher Zustände
- Übergänge zwischen Zuständen
- Verzweigungen, Komplexitätsbestimmung
- auch: Zustandsautomat (Finite State Machine)
- hier:
  - ◆ Münze einwerfen
  - ◆ Wahlschalter drücken
  - ◆ Dose entnehmen

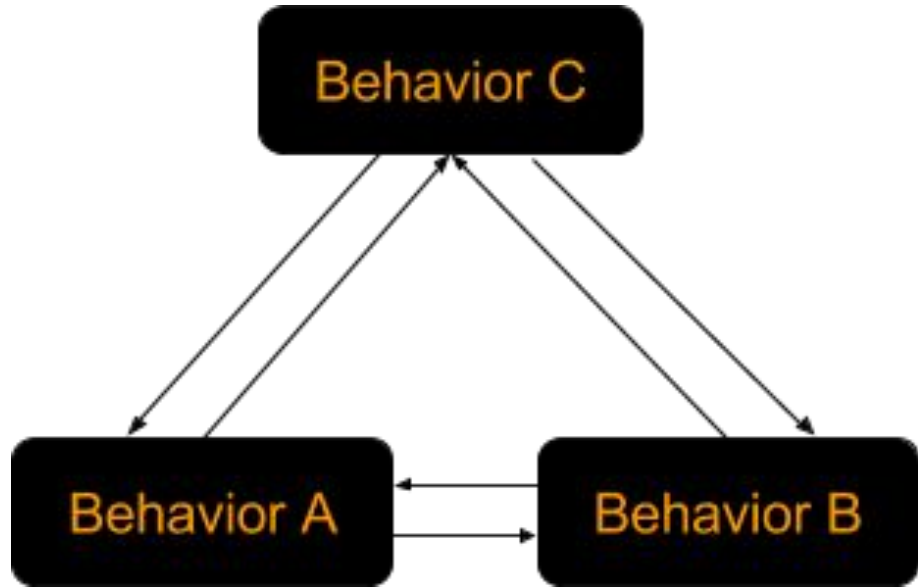


# Beispiel



# Zustandsautomaten

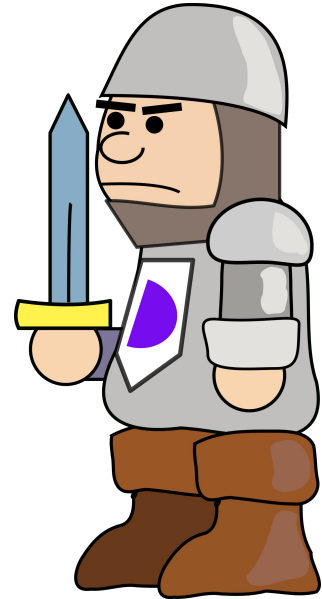
- Einfacher Aufbau
  - ◆ Zustände definieren
  - ◆ Übergänge definieren
- Gut geeignet für einfache KI
- Problem: Wird schnell unübersichtlich
- Unity: Playmaker



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

# Gruppenübung

Entwerfen Sie einen Zustandsautomaten, der das Verhalten eines Wächters in einer Burg beschreibt.

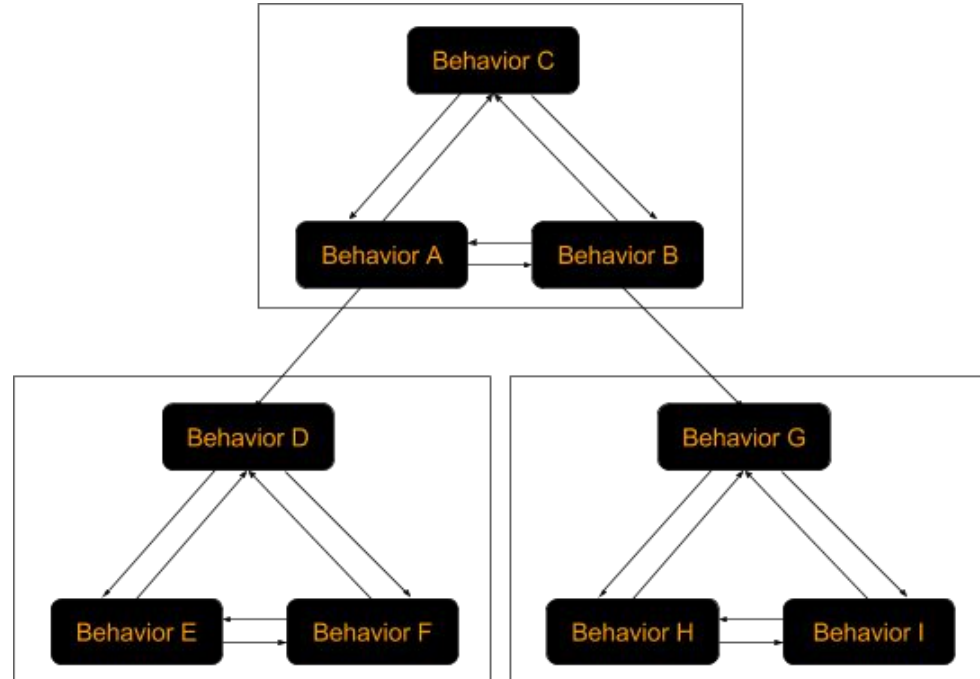


# Organisation komplexerer Automaten

→ Hierarchische Struktur

→ Beispiel:

- ◆ Behavior A: Schlafen
- ◆ Behavior B: Studieren
- ◆ Behavior C: Freizeit



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

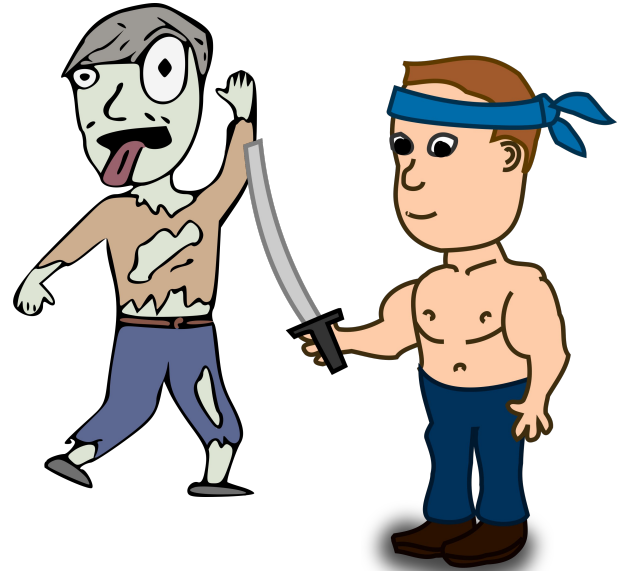


# Gruppenübung

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

Beispiele für NPCs:

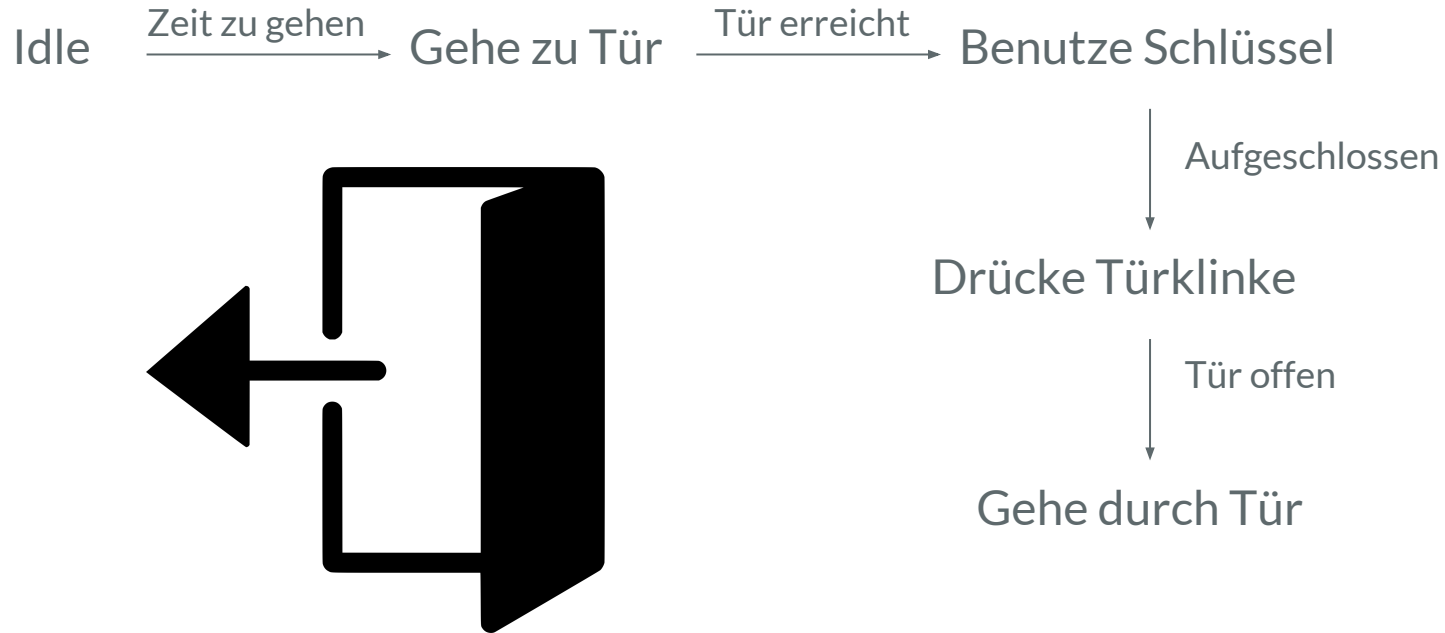
- Überlebender
- Infizierter
- Zombie, Monster



# Gruppenübung

- Wie strukturieren Sie Ihre Hierarchie?
- Wie gehen Sie mit “Handlungsabläufen” um?
  - Schleifen
  - Komplexität
- Wie einfach lässt sich ein “Überlebender” in einen “Infizierten” überführen?
  - Modifizierbarkeit

# Erster Schritt auf dem Weg zur Arbeit



# Handlungsabläufe

Hierarchische FSM



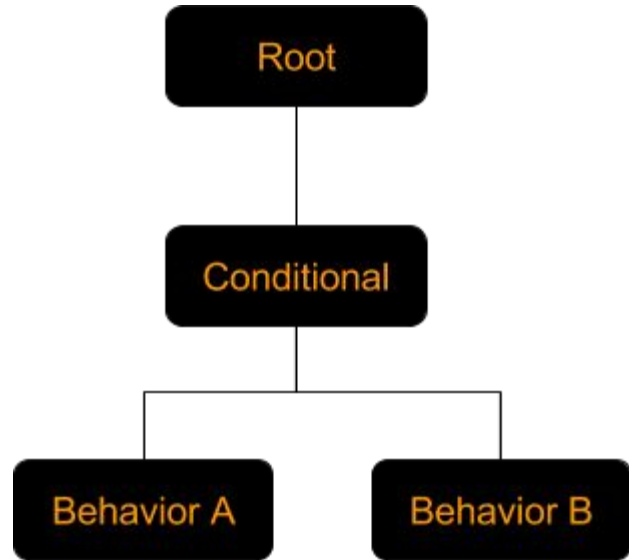
Behavior Tree

```
class Behaviortree:
    def __init__(self):
        t = threading.Thread(target=self.taskmain)
        t.start()
    def taskmain():
        self.gehe_durch_tuer()
        time.sleep(1)
        self.schliesse_tuer()
    def gehe_durch_tuer(self):
        self.benutze_schluessel()
        self.druecke_tuerklinke()
    def schliesse_tuer(self):
        pass
    def benutze_schluessel(self):
        pass
    def druecke_tuerklinke(self):
        pass
```

(Quelle: Wikipedia)

# Behavior Trees

- Aufbau nicht ganz so intuitiv
- Ablaufrichtung
- Sequenzen, Bedingungen, Repeater ...
- Gut geeignet für komplexere Gegner-KI
- Unity: Behavior Designer

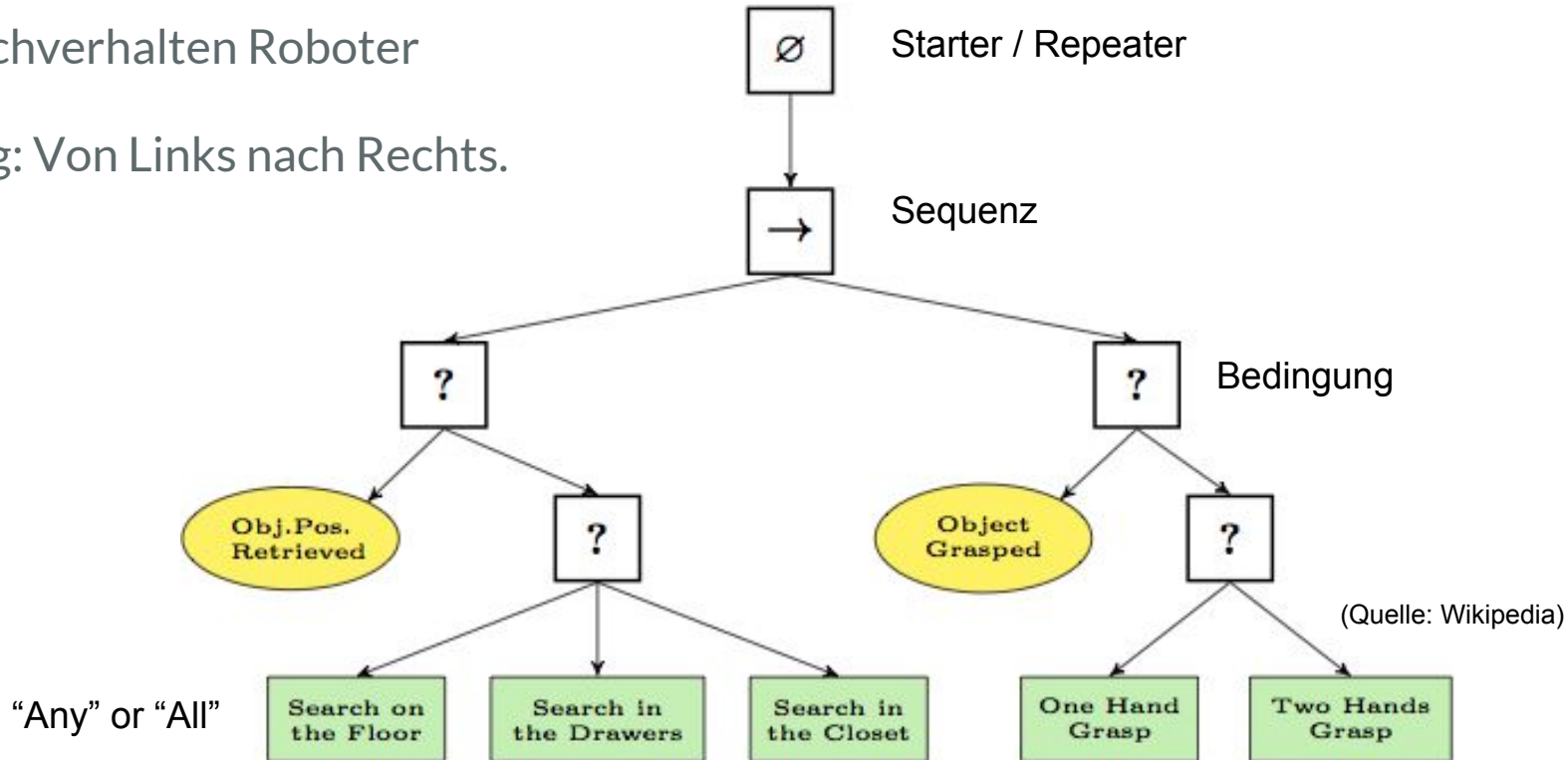


(Quelle: Rasmussen, J: Are Behavior Trees a Thing of the Past?, gamasutra.com, 27.04.2016)

# Elemente eines Behavior Trees

Beispiel: Suchverhalten Roboter

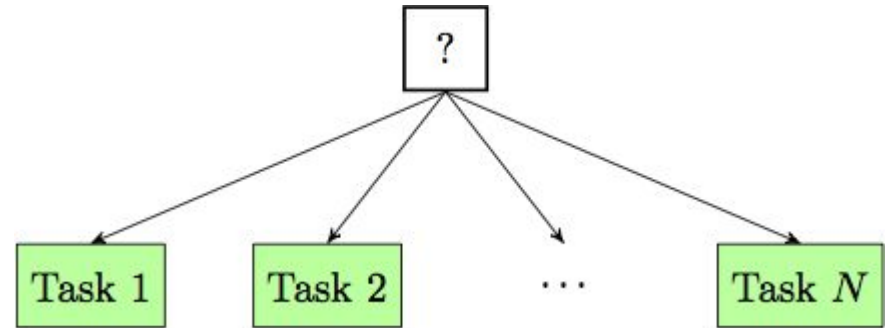
Abarbeitung: Von Links nach Rechts.



# Bedingung (Selector)

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

```
1 for i from 1 to n do
2   childstatus ← Tick(child(i))
3   if childstatus = running
4     return running
5   else if childstatus = success
6     return success
7 end
8 return failure
```

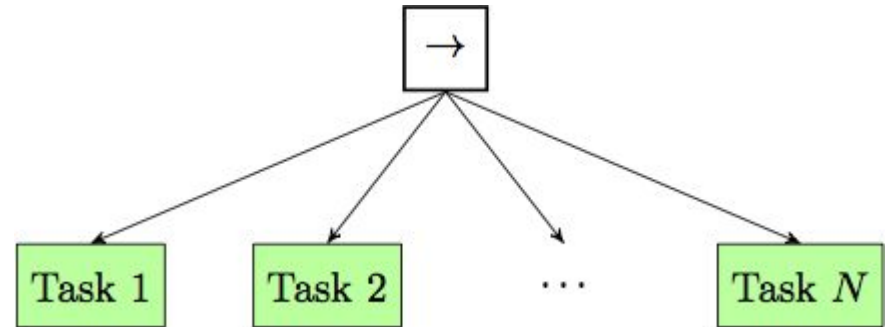


(Quelle: Wikipedia)

# Sequenz

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

```
1 for i from 1 to n do
2   childstatus ← Tick(child(i))
3   if childstatus = running
4     return running
5   else if childstatus = failure
6     return failure
7 end
8 return success
```

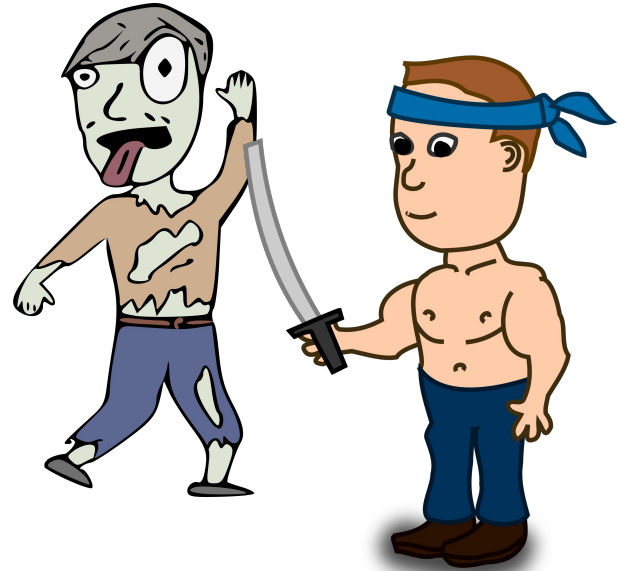




# Gruppenübung

Setzen Sie die Hierarchical FSM für Ihren Survival-Spiel NPC als Behavior Tree um.

→ Gibt es Vorteile? Nachteile?



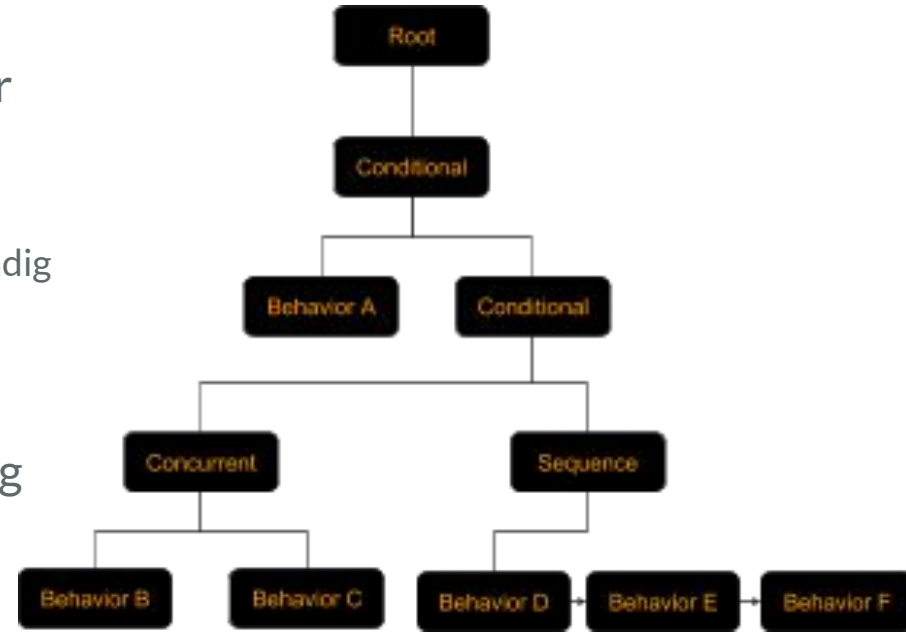
# Vor- und Nachteile

## Behavior Trees

- Subjekt-zentriert, intuitiv modellierbar
- Verhaltensweisen können sich ändern
  - bei Behavior Trees: einfach
  - bei Zustandsautomaten: einfach bis aufwendig

## Zustandsautomaten

- Ablauf-zentriert, formale Beschreibung
- Strukturierte, verzweigte Abläufe
- Gut geeignet für einfache Gegner-KI



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

# Beispiel

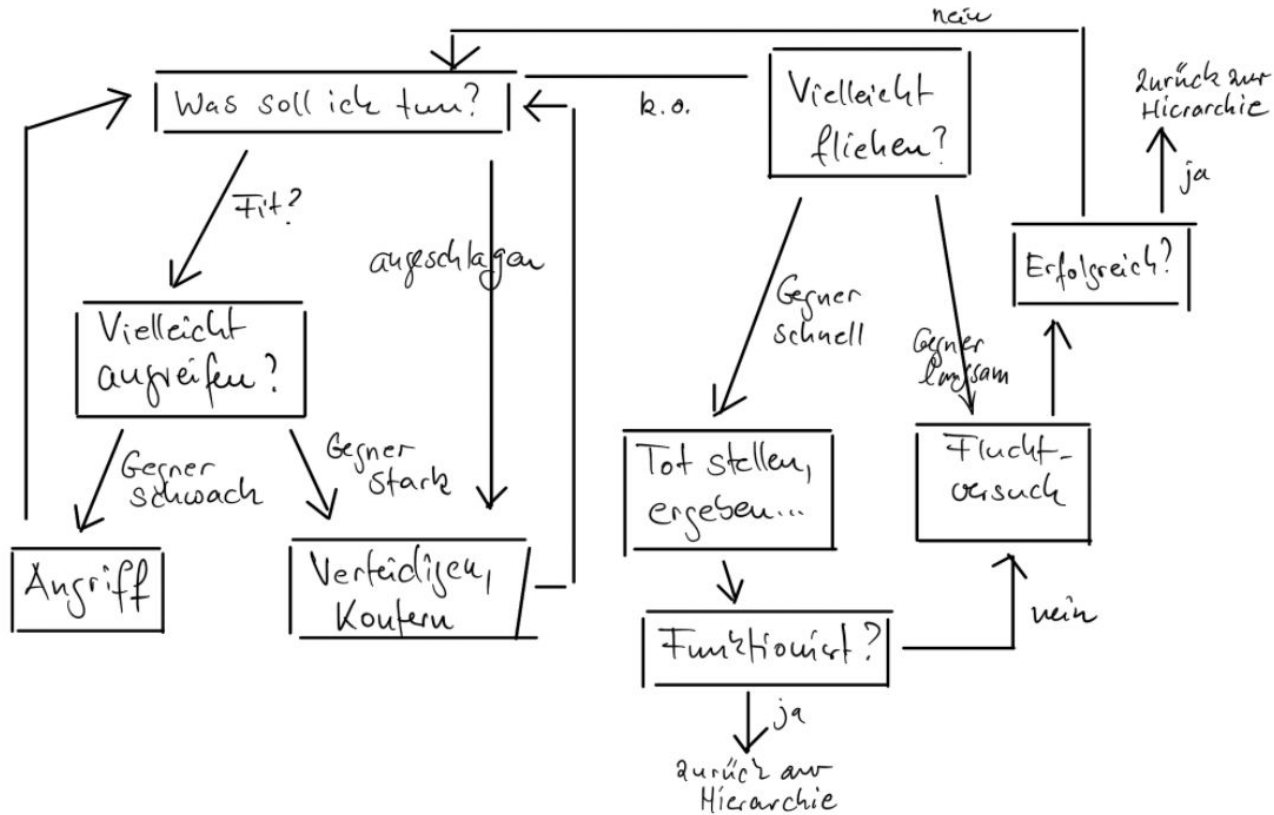
Kampfsystem, “Kampfhandlungen”

Grundsätzliche Frage: FSM oder BT?

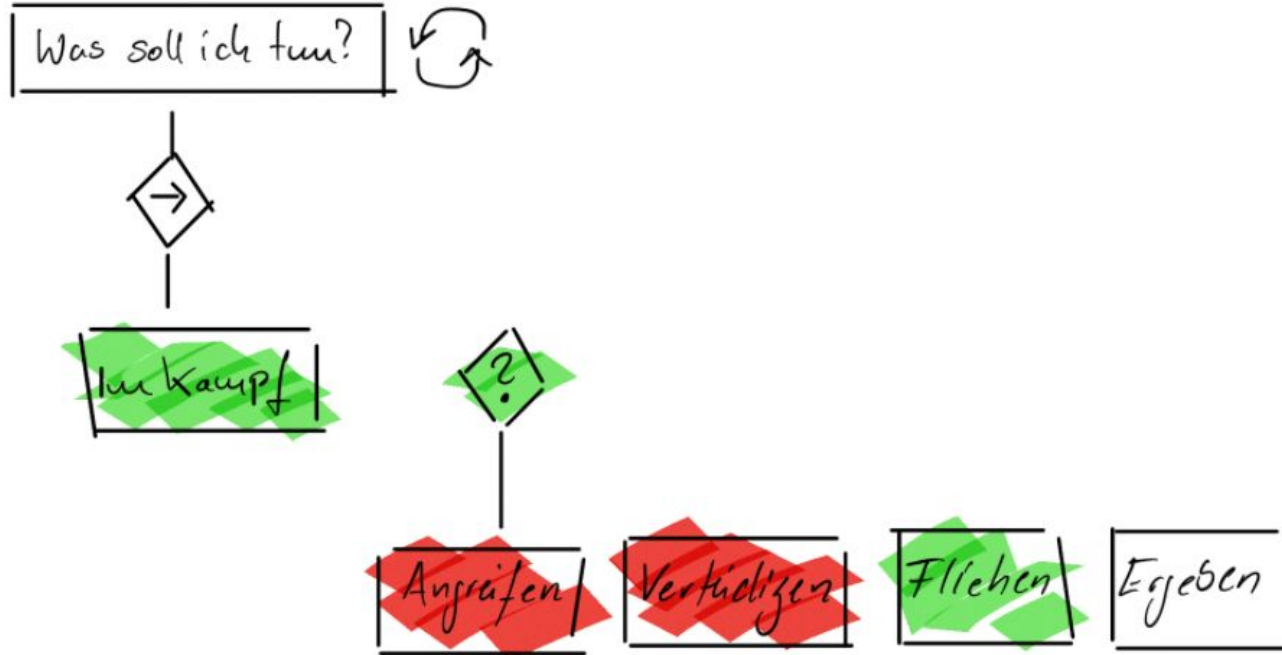
Checkliste:

- Komplexität (Knotenzahl)
- Liegt eine Abfolge vor? Gibt es eine Priorisierung?
- Transitions vs. Selektoren und Sequenzen
- Wäre eine Kombination von beiden nützlich?

# FSM: Kampfhandlung

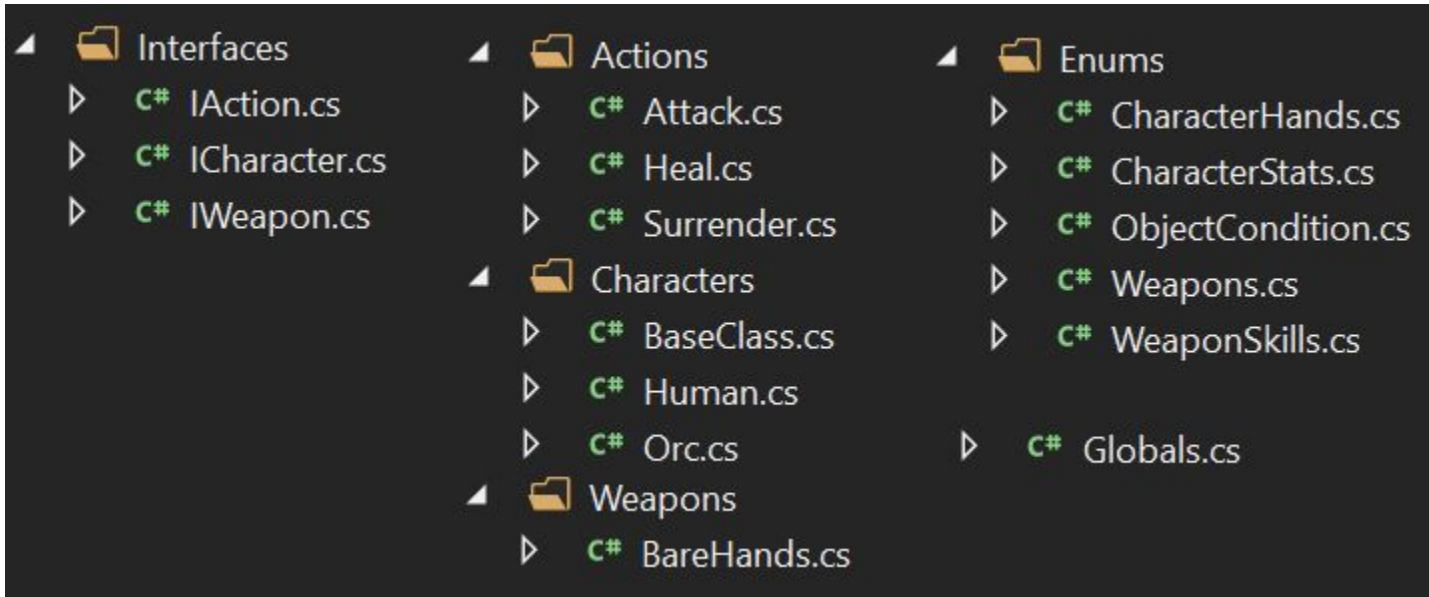


# BT: Kampfhandlung



# Implementierung in C#

## Projektstruktur



# Actions

6 Verweise | 0 Änderungen | 0 Autoren, 0 Änderungen

```
public interface IAction
```

```
{
```

7 Verweise | 0 Änderungen | 0 Autoren, 0 Änderungen

```
    string Description { get; }
```

6 Verweise | 0 Änderungen | 0 Autoren, 0 Änderungen

```
    int UpperLimit { get; }
```

8 Verweise | 0 Änderungen | 0 Autoren, 0 Änderungen

```
    int LowerLimit { get; }
```

4 Verweise | 0 Änderungen | 0 Autoren, 0 Änderungen

```
    bool CheckCondition(Interfaces.ICharacter player);
```

4 Verweise | 0 Änderungen | 0 Autoren, 0 Änderungen

```
    bool DoAction(Interfaces.ICharacter player, Interfaces.ICharacter otherPlayer = null);
```

```
}
```

# Characters

16 Verweise | 0 Änderungen | 0 Autoren, 0 Änderungen

```
public interface ICharacter
```

```
{
```

9 Verweise | 0 Änderungen | 0 Autoren, 0 Änderungen

```
    string Name { get; }
```

11 Verweise | 0 Änderungen | 0 Autoren, 0 Änderungen

```
    int[] Stats { get; set; }
```

9 Verweise | 0 Änderungen | 0 Autoren, 0 Änderungen

```
    int[] MaxStats { get; set; }
```

11 Verweise | 0 Änderungen | 0 Autoren, 0 Änderungen

```
    List<IWeapon> WeaponInventory { get; set; }
```

5 Verweise | 0 Änderungen | 0 Autoren, 0 Änderungen

```
    List<IWeapon> WeaponEquiped { get; set; }
```

7 Verweise | 0 Änderungen | 0 Autoren, 0 Änderungen

```
    int[] WeaponSkills { get; set; }
```

6 Verweise | 0 Änderungen | 0 Autoren, 0 Änderungen

```
    List<IAction> ActionsBattle { get; set; }
```

4 Verweise | 0 Änderungen | 0 Autoren, 0 Änderungen

```
    ICharacter Target { get; set; }
```

7 Verweise | 0 Änderungen | 0 Autoren, 0 Änderungen

```
    bool IsActiv { get; set; }
```

4 Verweise | 0 Änderungen | 0 Autoren, 0 Änderungen

```
    bool IsAlive();
```

```
}
```



# Weapons

9 Verweise | 0 Änderungen | 0 Autoren, 0 Änderungen

```
public interface IWeapon
```

```
{
```

2 Verweise | 0 Änderungen | 0 Autoren, 0 Änderungen

```
    string WeaponName { get; }
```

4 Verweise | 0 Änderungen | 0 Autoren, 0 Änderungen

```
    Enums.Weapons WeaponType { get; }
```

4 Verweise | 0 Änderungen | 0 Autoren, 0 Änderungen

```
    int MaxDamage { get; set; }
```

4 Verweise | 0 Änderungen | 0 Autoren, 0 Änderungen

```
    int MinDamage { get; set; }
```

4 Verweise | 0 Änderungen | 0 Autoren, 0 Änderungen

```
    int StaminaCost { get; set; }
```

2 Verweise | 0 Änderungen | 0 Autoren, 0 Änderungen

```
    Enums.ObjectCondition ObjectCondition { get; set; }
```

2 Verweise | 0 Änderungen | 0 Autoren, 0 Änderungen

```
    bool IsTwoHanded { get; }
```

2 Verweise | 0 Änderungen | 0 Autoren, 0 Änderungen

```
    int Attack(int skill);
```

2 Verweise | 0 Änderungen | 0 Autoren, 0 Änderungen

```
    bool Defend(int skill, int malus);
```

2 Verweise | 0 Änderungen | 0 Autoren, 0 Änderungen

```
    int DealDamage();
```

1-Verweis | 0 Änderungen | 0 Autoren, 0 Änderungen

```
    int Exhaust();
```

```
}
```

# Ablauf

```
foreach (var player in players)
{
    foreach (var action in player.ActionsBattle)
    {
        if (!action.CheckCondition(player)) continue;

        Console.WriteLine(player.Name + " is " + action.Description);
        var success = action.DoAction(player, player.Target);
        if (success)
        {
            Console.WriteLine(player.Name + " was successful");
            break;
        }
        Console.WriteLine(player.Name + " failed");
        break;
    }
}
```