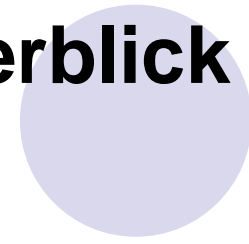
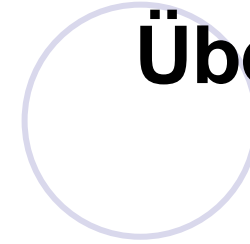
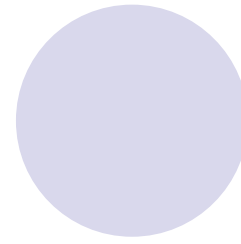
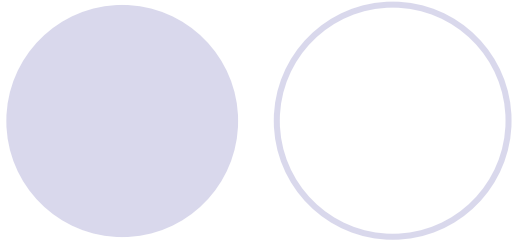




**Seminar SS07**  
**Spiele KI**

**Thema: Hindernisnavigation**

Sebastian Hammes mi2269



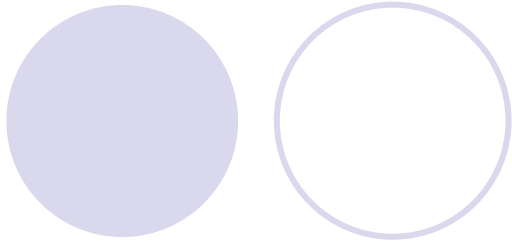
# Überblick

- Hindernisarten
- Wo findet man Hindernisnavigation
  - Mensch
  - KI
- Hindernisnavigation anhand von
  - Maps
  - Graphen
    - Goal-based Path Planning und Execution
- Zusammenfassung



# Was ist Hindernisnavigation

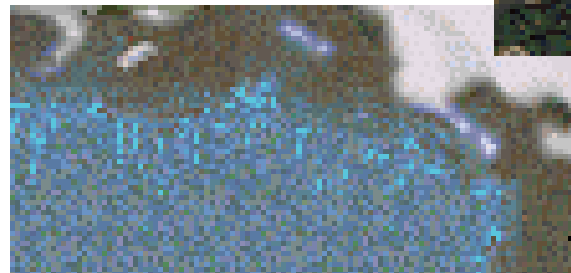
- Hindernisnavigation ist nicht das reine Umlaufen von Hindernissen
- Vielmehr auch das entfernen von Hindernissen
- Eine Art Hindernisnavigation findet bereits beim Pathfinding statt, z.B. A\*
- Es gibt 3 Arten von Hindernisnavigation
  - Navigation um unbewegliche Hindernisse
  - Navigation durch unbewegliche/entfernbbare Hindernisse
  - Navigation um bewegliche Hindernisse



# Hindernisarten

## unbewegliche Hindernisse

- Berge
- Flüsse
- Felsen
- Mauern
- Gebäude
- ...
- spielabhängig

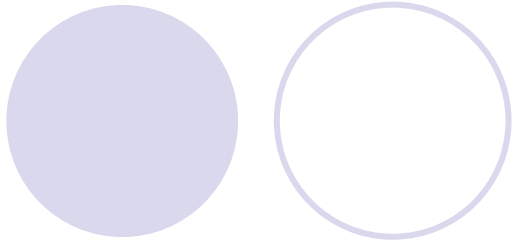


# Hindernisarten

## unbewegliche, beseitigbare Hindernisse

- Bäume
- Mauern
- Gebäude
- Brücken
- Mienen
- Türen
- ...
- spielabhängig





# Hindernisarten

## bewegliche Hindernisse

- Eigene Einheiten
- Fremde Einheiten
- Aufzüge
- Steine
- ...
- spielabhängig






# Wo findet man Hindernisnavigation

## Generell

- überall
- im alltäglichen Leben
- Beispiel Studenten in der FH Wedel
- Übertragen auf Spiele
  - Beim menschlichen Spieler
  - Beim NPC




# Wo findet man Hindernisnavigation

## Der menschliche Spieler

- Hindernisnavigation in First-Person-Spielen
  - In diesen Spielen muss der menschliche Spieler alle Arten von Hindernisnavigation selber übernehmen
- Hindernisnavigation in Strategiespielen
  - Vom menschlichen Spieler übernommene Hindernisnavigation
    - Unbewegliche/beseitigbare Hindernisse
  - Vom Computer übernommene Hindernisnavigation
    - Unbewegliche und bewegliche Hindernisse





# Wo findet man Hindernisnavigation

## Der Software-Agent Definition

„Als Software-Agent bezeichnet man ein Computerprogramm, das weitgehend unabhängig von Benutzereingriffen arbeitet, es löst Aktionen aufgrund eigener Initiative aus (proaktiv), reagiert auf Änderung der Umgebung (reaktiv), es kommuniziert mit anderen Agenten und lernt aufgrund zuvor getätigter Entscheidungen bzw. Beobachtungen.“

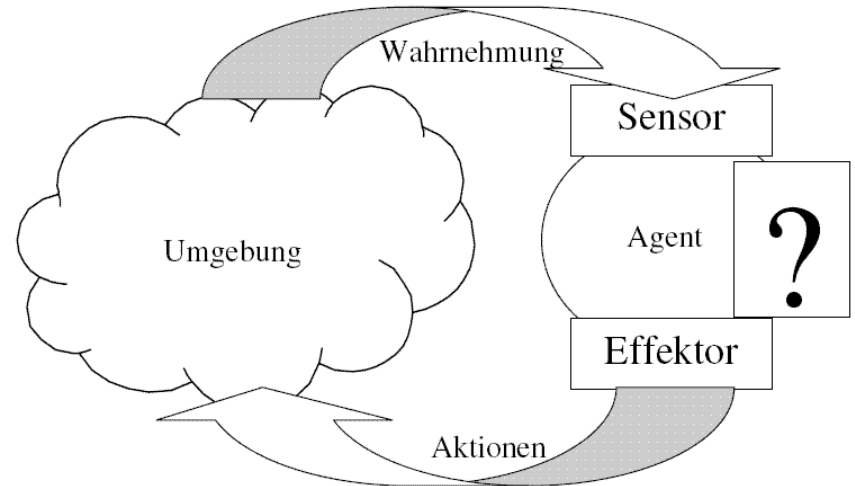
Wikipedia

# Wo findet man Hindernisnavigation

## Der Software-Agent

### Eigenschaften

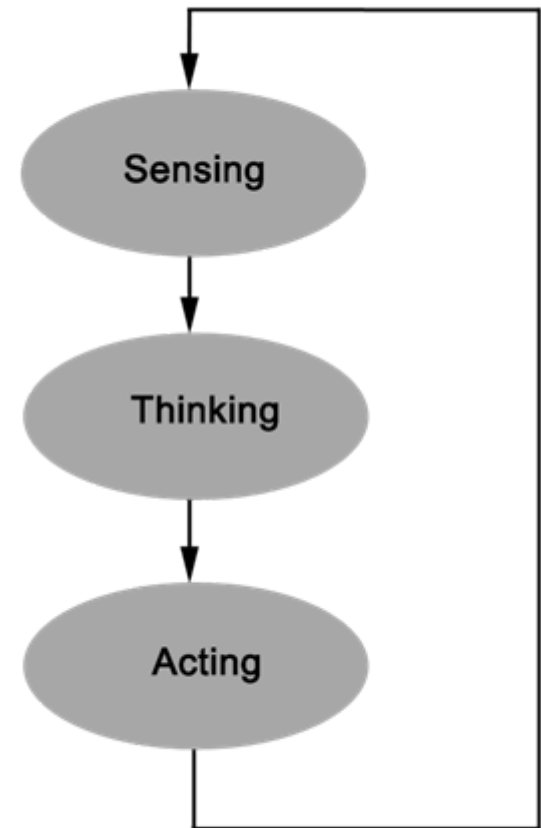
- Komplexes Programm
- Trifft Entscheidungen
- Wissensbasis
- Wahrnehmung von Ereignissen
- Regeln von Ereignis → Handlung
- Komplexer Agent
  - Ziele, Intentionen, Emotionen, Wünsche
  - Arbeitet Plan aus → Zielorientiert
  - Flexibel für eintretende Ereignisse



# Wo findet man Hindernisnavigation

## Der Software-Agent von innen

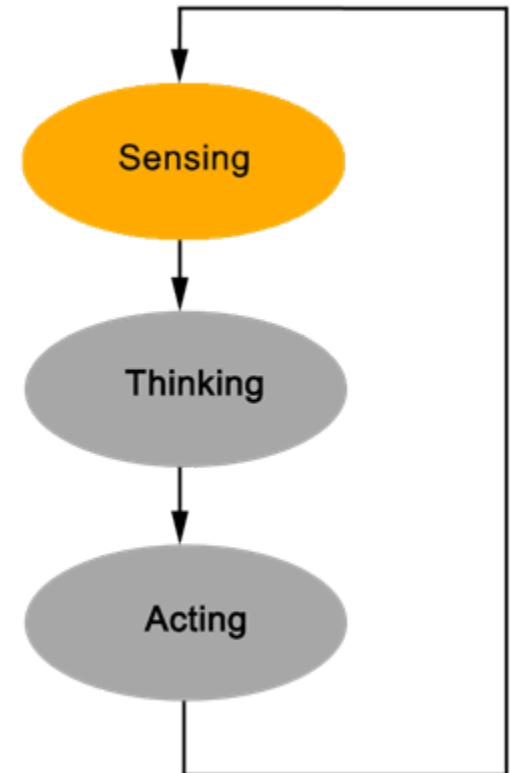
- Hauptbestandteile
  - Sensing
  - Thinking
  - Acting
- Parallelen zum menschlichen Spieler
- Unendliche Schleife
- Kann sehr komplex sein



# Wo findet man Hindernisnavigation

## Der Software-Agent von innen

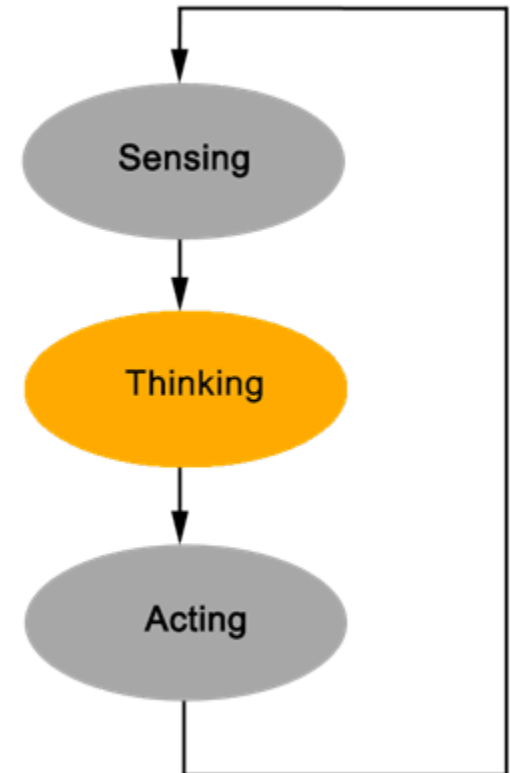
- Informationen über aktuellen Zustand ermitteln
- Informationen sind im Spiel immer präsent
  - NPC darf nicht alle Informationen kennen (cheating)
- Die “Sinne”
  - Visuelle Informationen
  - Auditive Informationen



# Wo findet man Hindernisnavigation

## Der Software-Agent von innen

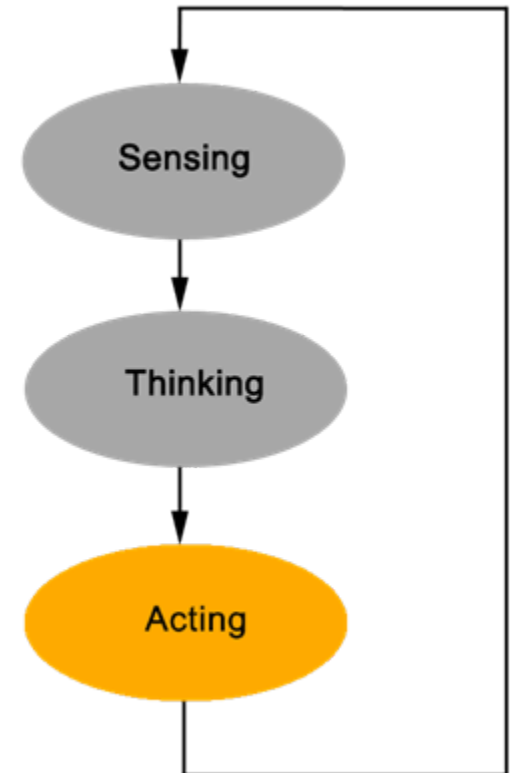
- Informationen verarbeiten
- Aufbau eines Plans
- Qualität abhängig von Komplexität des Software-Agenten
- Eigentliche Intelligenz
- Verschiedene Verfahrenstechniken
  - Regelbasiert
  - Suchbasiert
- Grundlage für die Hindernisnavigation




# Wo findet man Hindernisnavigation

## Der Software-Agent von innen

- Ausführung der Aktionen die im Thinking geplant wurden
- Einzige für den Spieler sichtbare Verhalten des NPC
- Gängige Aktionen
  - Positionsveränderung
  - Bewegung
  - Schießen
  - Gegenstand aufnehmen
- Je nach Weltdarstellung realistisch





# Wo findet man Hindernisnavigation

## Der Software-Agent

### Hindernisnavigation

- Thinking ermittelt einen Plan
- Die Hindernisnavigation setzt diesen um
- Acting führt diesen dann schließlich aus
- Hindernisnavigation ist also zwischen dem Thinking und dem Acting angesiedelt
- Greift auf gleiche Algorithmen zurück, wie beim menschlichen Spieler



# Arten der Welt Darstellung

- Vielzahl von Darstellungsarten
- 2 Arten der Welt Darstellung
  - Maps
  - Graphen
- Welt Darstellung hat Einfluss auf den Pathfindingalgorithmus
- Eine gute Welt Darstellung ist Grundvoraussetzung für eine gute Hindernisnavigation



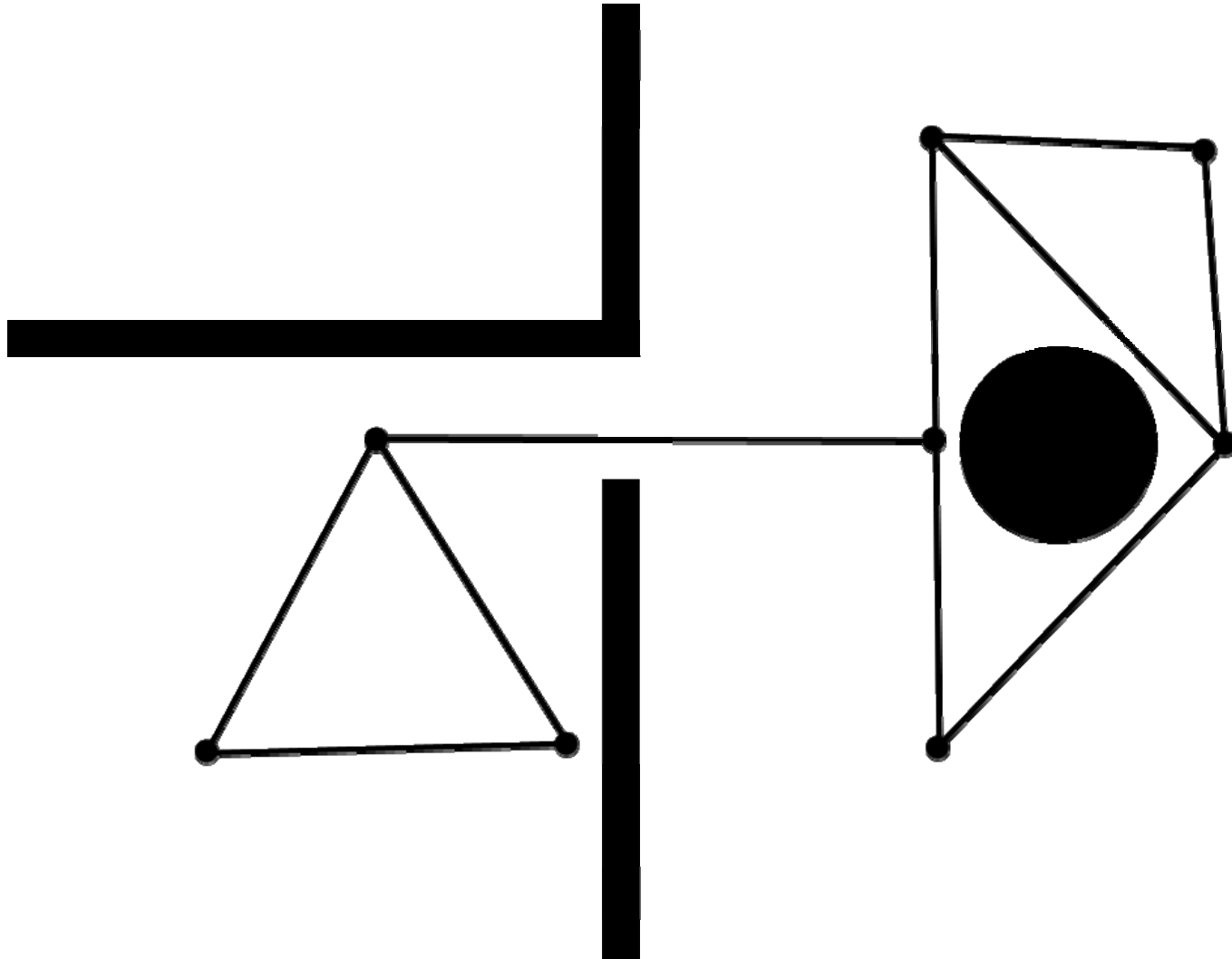
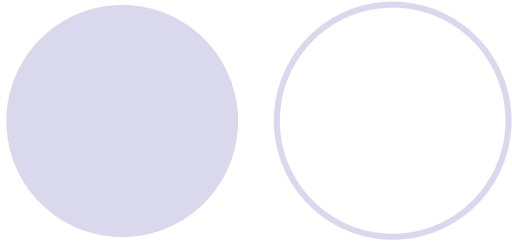
# Arten der Welt Darstellung

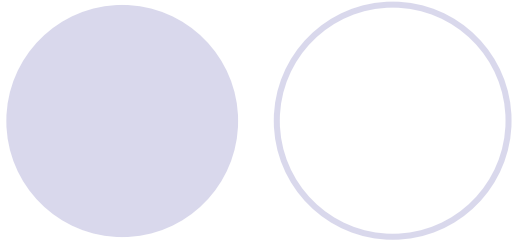
## Maps



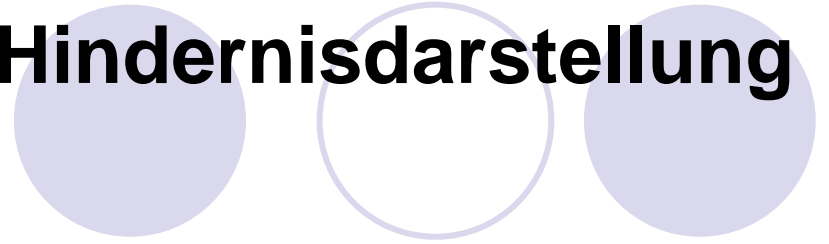
# Arten der Weltdarstellung

## Graphen





# Hindernisdarstellung



- bei den verschiedenen Welt darstellungsarten werden auch die Hindernisse verschieden dargestellt
- Hindernisdarstellung hat Einfluss auf die Hindernisnavigation
- Hindernisnavigation kann nur so gut sein, wie dies die Welt darstellung zulässt
- Ungenaue Welt darstellung bedeutet ungenaue Navigation
  - Dies kann zu unrealistischen Effekten führen

# Maps

## Hindernisdarstellung

- einzelne Zellen der Map werden markiert
- Hindernisdarstellung ist nur sehr ungenau
- Markierung ist abhängig vom Hindernis

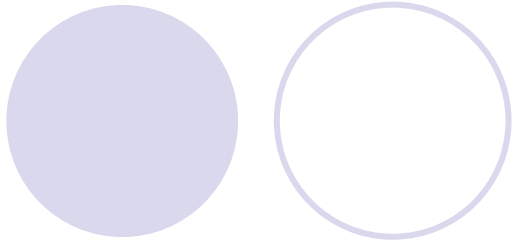


# Maps

## Hindernisdarstellung unbewegliche Hindernisse

- Füllen ganze Zellen aus
- Große Hindernisse belegen mehrere zusammenhängende Zellen
- Meist nur schlechte Realitätsannäherung
- Zellen dürfen nicht einfach nur als Hindernis markiert werden



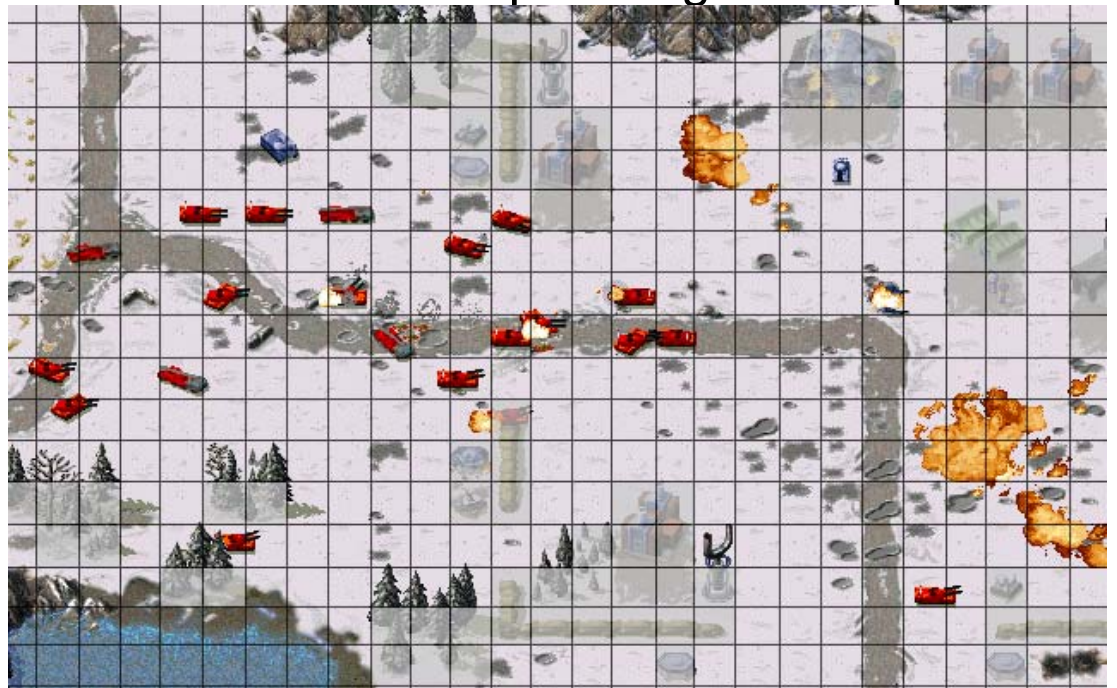


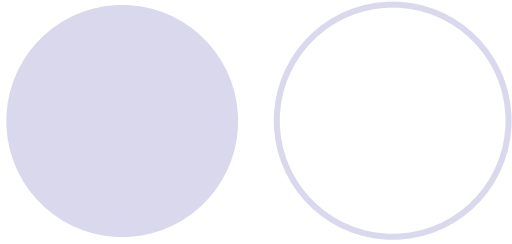
# Maps

## Hindernisdarstellung

unbewegliche, beseitigbare Hindernisse

- Einfache Darstellung möglich
- Belegen genauso Platz wie andere unbewegliche Hindernisse
- Jedoch zusätzlich Informationen notwendig
- auch hier gilt, dass die Hindernisse unterscheidbar bleiben müssen
- Bei Beseitigung dieser Hindernisse ist die Anpassung der Map notwendig





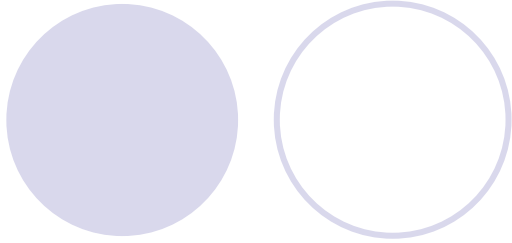
# Maps

## Hindernisdarstellung

### bewegliche Hindernisse

- werden genauso wie andere Hindernisse dargestellt indem eine Zelle als Hindernis markiert wird
- Die Ausmaße der Einheit decken meist nur schlecht das Raster ab
- hierbei gilt auch, dass nicht jedes bewegliche Hindernis für jede Einheit ein Hindernis darstellt
- Bei beweglichen Hindernissen muss ständig die Map aktualisiert werden



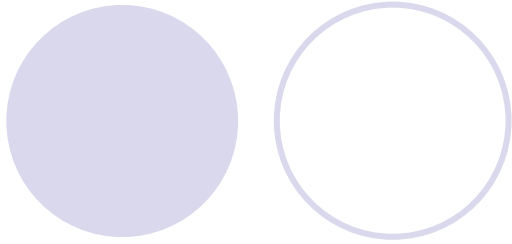


# Maps

## Hindernisnavigation unbewegliche Hindernisse

- Navigation bei unbeweglichen Hindernissen übernimmt dabei komplett der Pathfindingalgorithmus
- Nicht jedes unbewegliche Hindernis ist für jede Einheit ein Hindernis
- Flüsse und Berge sind z.B. für Flugzeuge keine Hindernisse, jedoch für Landeinheiten
- Flugzeuge besitzen keine unbeweglichen Hindernisse
  - Sie müssen nur die Flughöhe anpassen



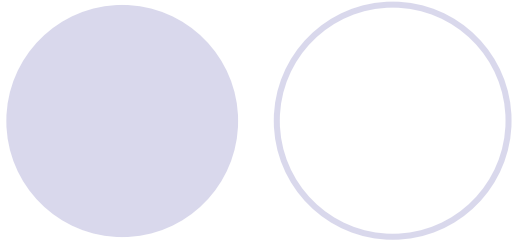


# Maps

## Hindernisnavigation

unbewegliche, beseitigbare Hindernisse

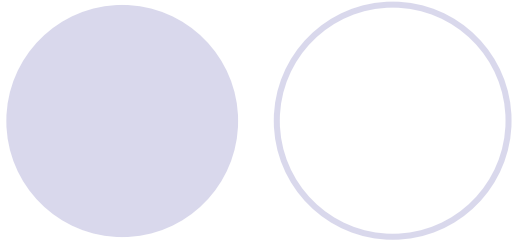
- nicht jedes Hindernis kann von jeder Einheit beseitigt werden
- verschiedene Arten von Einheiten können verschiedene Hindernisse zerstören
- Pathfindingalgorithmus muss also diese Art von Hindernissen besonders berücksichtigen
- Zum Abschätzen des Zerstörungsaufwandes muss eine Kostenfunktion bereitstehen



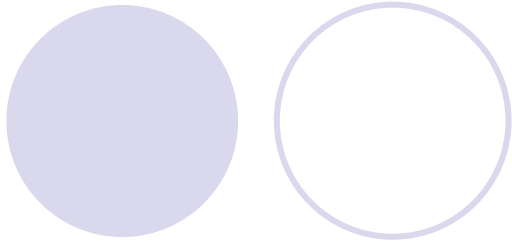
# Maps

## Hindernisnavigation bewegliche Hindernisse

- müssen gesondert berücksichtigt werden
- z.B.
  - gegnerische Einheiten werden als unbeweglich/beseitigbar eingestuft
  - eigene Einheiten werden als beweglich eingestuft
- Einheiten müssen also untereinander „kommunikativ“ agieren
- Das bloße Pathfinding reicht nicht aus



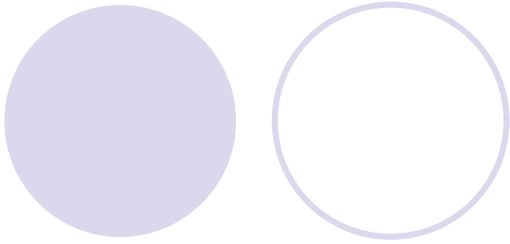
- Einfache Welt Darstellung
- Hindernisse können einfach durch Zellen definiert werden
- Hindernisse brauchen Flags zur Unterscheidung
- Anpassung der Map zur Laufzeit notwendig
- Wirkt sehr unrealistisch
- Hindernisnavigation wird zum Teil dem Pathfindingalgorithmus übergeben
- Hindernisnavigation bei unbeweglichen, beseitigbaren Hindernissen stellt sich sehr leicht da



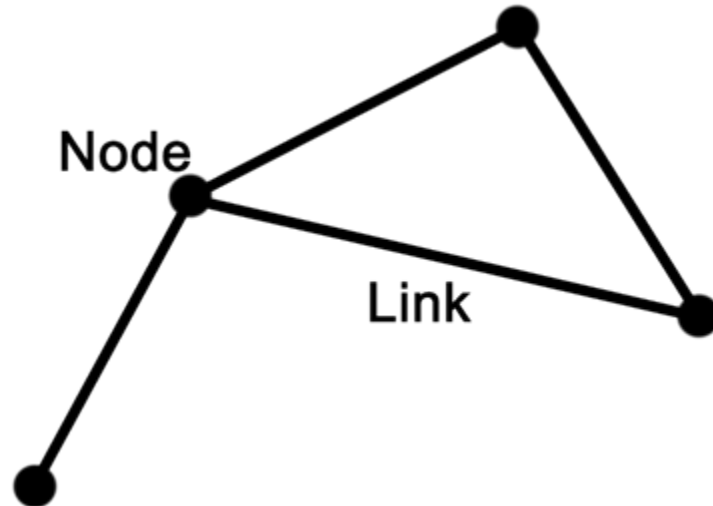
# Graphen

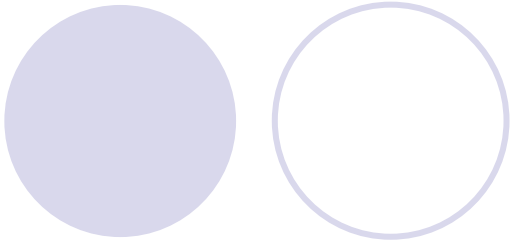
## Hindernisdarstellung

- Graph-Daten generieren → suche
- Einsatz vor allem in 3D-Spielen
- Sehr effizient in der CPU- und Memorynutzung
- Die Errichtung des Graphen
  - Benötigt zusätzlich Zeit und Arbeit
  - Recht schwer zu realisieren



- Nodemap
- Besteht aus Nodes und Links
- Path ist ein Weg aus hintereinander liegenden, verbundenen Nodes
- Links sind richtungsunabhängig



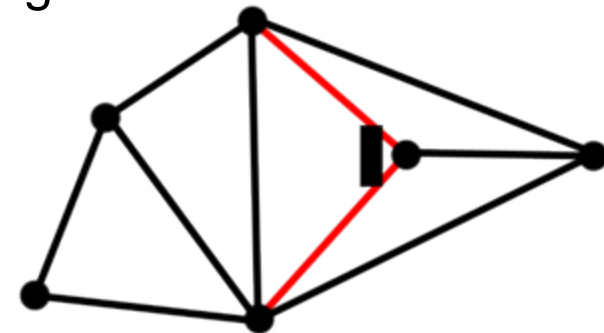


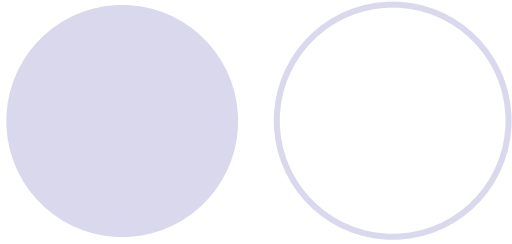
# Graphen

## Hindernisdarstellung

### Berechnung des Graphen

- automatisch generieren lassen
- manuell generieren
  - Level-Designer läuft Spielfeld ab
  - Setzen von Nodes, wenn eine bestimmte Distanz zu den benachbarten Nodes erreicht
  - 4-8 Meter soll gut funktionieren
  - Links werden automatisch gesetzt
    - Line-of-sight-testing
  - Nachbesserung der Nodes und Links vom Level-Designer
  - Wichtige Punkte müssen besonders sorgfältig berechnet werden
- Graph muss für jede Szene veränderbar sein



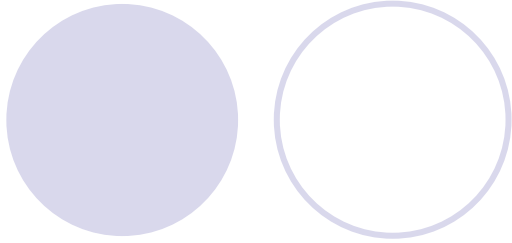


# Graphen

## Hindernisdarstellung

### Kommentierung der Nodemap

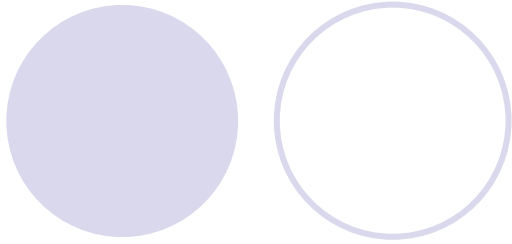
- wichtig für die Navigation
- Nutzen der Kommentierung
  - Markierung dient gescripteten Events
  - Markierung von strategisch guten Positionen
  - Markierung von Schaltern, Aufzügen
  - Bestimmung des Zustandes des Charakters
- Links besitzen Informationen zum durchqueren des Gebietes (z.B. Tür)
  - Link besitzt Information, ob man ihn durchqueren kann oder nicht
  - Türdrücker muss per Name verlinkt sein
  - Automatiktüren müssen nicht benannt werden
- Flags für gefährliche Gebiete (Klippen)
- Flags für das Springen entlang des Links
- Entlastet die Kollisionserkennung



# Graphen Hindernisdarstellung Probleme für die Hindernisnavigation

- Nodes beschreiben nur Punkte im Raum
  - Keine Informationen über den Freiraum um ein Node
- Links beschreiben nur direkte schmale Wege zwischen zwei Nodes
  - Keine Informationen über den Freiraum um einen Link
- Ansprüche an die Hindernisnavigation
  - Muss natürlich und effizient aussehen
- Aber wie?



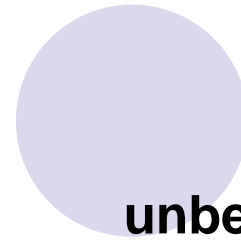
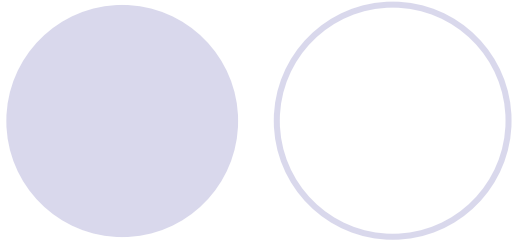


# Graphen

## Hindernisdarstellung

### Lösung

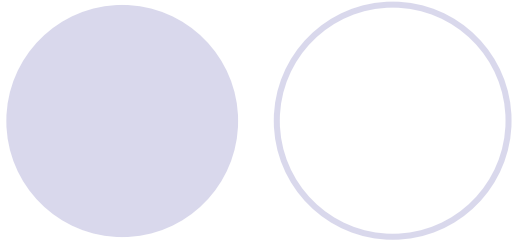
- der Graph muss um Informationen erweitert werden
- Definition von Freiraum um ein Node
  - Erweiterung der Nodes um eine Radiusangabe
- Definition von Freiraum um einen Link
  - Dafür sind keine weiteren Angaben nötig
  - Die Information der Radii reicht dafür bereits aus
- Sehr effiziente Lösung hinsichtlich der Datenmengen, um Freiraum um Nodes und Links zu speichern
- Radii sind 2D (horizontal)
- Wenn 3D benötigt, werden daraus Zylinder



# Graphen Hindernisarten

unbewegliche Hindernisse

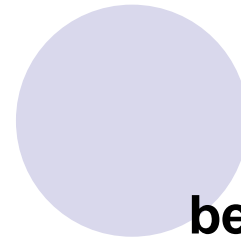
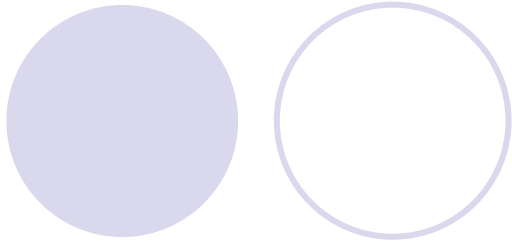
- können relativ gut abgebildet werden
  - abhängig von der Qualität des Graphen
- die Informationsmenge um Hindernisse zu speichern ist ressourcensparend
- Radii um Nodes definieren gleichzeitig Hindernisse
- Können realistisch von einem NPC umlaufen werden



# Graphen Hindernisarten

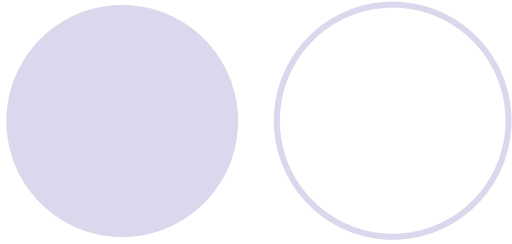
unbewegliche, beseitigbare Hindernisse

- Werden in einem Graph mit Links berücksichtigt, die eine Bedingung enthalten, um diesen Link folgen zu können
- Beispiel Tür
  - Erst muss ein Schalter gedrückt werden, um das Hindernis Tür zu beseitigen
  - Statusspeicherung, ob eine Tür passierbar ist, wird mit Flags realisiert



# Graphen Hindernisarten bewegliche Hindernisse

- Graphen dienen nur zur Darstellung der Weltgeometrie
- Bewegliche Hindernisse werden nicht berücksichtigt
- Dazu dient das System der Bounding-Boxes/-Spheres
- Ein ständiges Anpassen des Graphen ist unmöglich
- Zudem sind Computergenerierte Graphen meist schlecht zu gebrauchen



# Graphen

## Bezeichnung des „Irrrens“

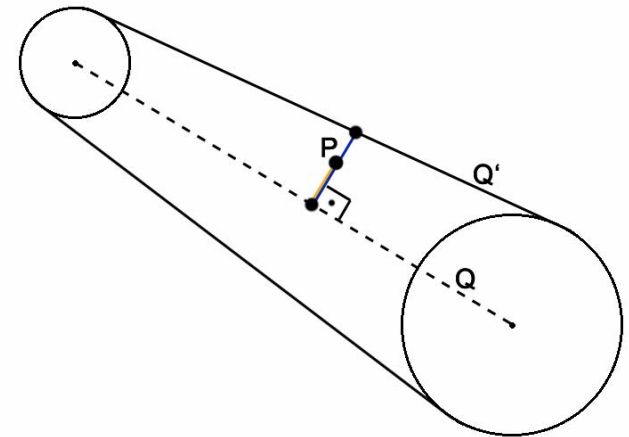
- Bezeichnung für die Distanz zum ursprünglichen Link
- anhand eines für die Hindernisnavigation modifizierten Graphen kann ein NPC jederzeit feststellen, wie weit er bereits vom ursprünglichen Pfad abgekommen ist

- Berechnung

- Berechne Abstand von P zu Q = a
- Berechne Abstand von Q zu Q' = b

- Ergebnis

- $a < b$ : NPC befindet sich innerhalb des Freiraums
- $a > b$ : NPC befindet sich außerhalb des Freiraums





# Graphen Goal-based Path Planning und Execution

- stellt das Problem als objektorientiert da
- Code für die Navigation auf dem Graphen wird einfacher zu schreiben, lesen, warten und debuggen
- Zum Planen wird grundsätzlich ein Pathfindingalgorithmus benötigt
- Der gefundene Pfad kann dann durch den NPC ausgeführt werden
- Dient der Erreichung des geplanten Ziels

## Goal-based Path Planning und Execution

### Eigenschaften von Goals (für unsere Zwecke)

- Flag zur Bestimmung ob Erfolg oder Fehler
- update() – Funktion
- Goals können selbst Subgoals als Liste von Goals erzeugen
- Subgoals werden in Listenreihenfolge nach und nach ausgeführt
- Bei Erfolg wird das Subgoal aus der Liste gelöscht
- Bei Misserfolg wird der Fehler dem Parentgoal gemeldet
  - Das Parentgoal entscheidet, ob die Subgoals neu geplant werden oder ob es den Fehler weiter nach oben gibt
- Goals in Form von Links können Kosten besitzen
  - Einen Umweg laufen geht schneller, als eine Zugbrücke herunter zu lassen



# Graphen

## Goal-based Path Planning und Execution

### Goal-Arten

- Goal\_GotoPosition
- Goal\_GotoNode
- Goal\_FollowLink



The slide features a decorative header with five light purple circles. The second circle from the left contains the text 'Goal-based Path Planning' in bold black font. The fourth circle contains the text 'Graphen' in bold black font. The fifth circle contains the text 'und Execution' in bold black font. Below these circles, the text 'Goal\_GotoPosition' is written in bold black font.

# Goal-based Path Planning und Execution

## Goal\_GotoPosition

- Highest Level Goal
- Position kann ein beliebiger Punkt auf der Weltgeometrie sein
- Position muss nicht ein definierter Node aus dem Graphen sein
- Kann als Subgoal Goal\_GotoNode generieren
- Dient zum Erreichen von Positionen, die nicht direkt durch den Graphen abgebildet sind

```
Goal_GotoPosition(Position) {  
    ... /* Subgoals */ ...  
}
```

# Graphen

## Goal-based Path Planning und Execution

### Goal\_GotoNode

- Dient ausschließlich zum Erreichen eines definierten Nodes
- benutzt einen Pathfindingalgorithmus, z.B. A\*
  - GenerateGoalsFromPath()
- Kann wiederum Subgoals erstellen
- Dient auch zur Fehlererkennung
- Viele Goal\_FollowLinks können durch ein Goal\_GotoNode ersetzt werden

```
Goal_GotoNode(Node) {  
    ... /* Subgoals */ ...  
}
```

# Graphen

## Goal-based Path Planning und Execution

### Goal\_FollowLink

- beschreibt den Übergang von einem Node zu einem anderen
- verschiedene Typen von Links
  - Sprünge über Abgründe
  - Sprünge über Mauern
  - Weg entlang einer Klippe
  - Weg durch eine Tür
- Abhängig von der Art des Links wird die jeweilige Funktion aufgerufen
- Dient auch zur Fehlererkennung
  - Beim Herunterfallen einer Klippe
  - Beim Abkommen vom rechten Weg

```
Goal_FollowLink(Link) {  
    ... /* Subgoals */ ...  
}
```

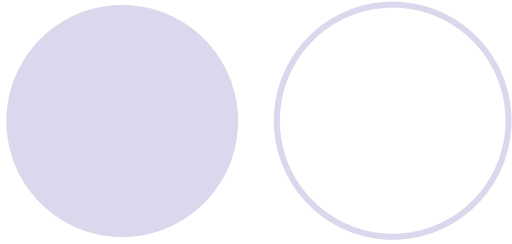


# Graphen

## Goal-based Path Planning und Execution

### Sprung über einen Abgrund/Mauer

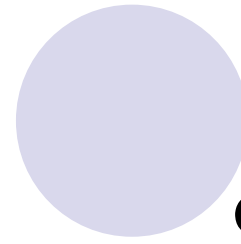
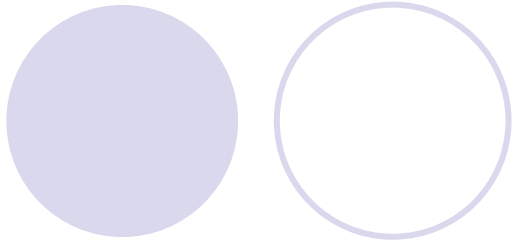
- bei jedem update() wird ein linecheck gegen die Weltgeometrie durchgeführt
- Abgrund
  - wenn der linecheck fehlschlägt
    - Klippe muss in der nähe sein
    - Charakter startet den Sprung (jump())
- Mauer
  - Linecheck bis Kollision mit Mauer
  - Charakter startet den Sprung (jump())



# Graphen

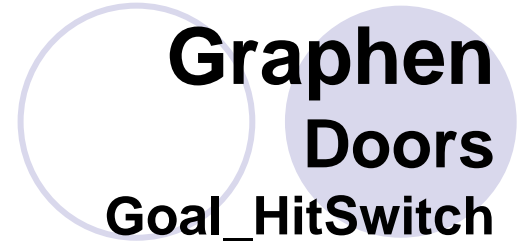
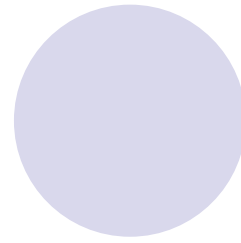
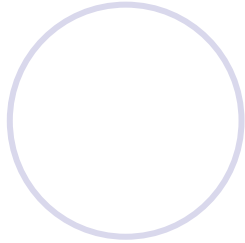
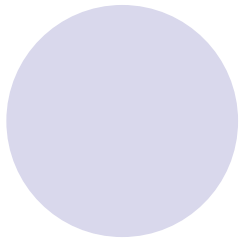
## Door und Elevators

- Links können auch Türen und Aufzüge darstellen
- `GenerateGoalsFromPath()` muss diese anders berücksichtigen
- Daher werden die `Goal_FollowLinks` rückwärts aufgebaut
- Bei Erreichen einer Tür wird ein `Goal_GoThroughDoor` erzeugt
- Der restliche Pfad wird dann von diesem Goal erzeugt



**Graphen  
Doors**  
Goal\_GoThroughDoor

- Suchen nach Schalter für die Tür
- Dazu wird die Liste mit verbundenen Komponenten bemüht
- Wenn Schalter gefunden, Goal kann seine Arbeit starten
- Goal unterteilt diese Aufgabe in 3 Subgoals
  - Das Drücken des Schalters
  - Das Bewegen des Charakters vor die Tür
  - Das Durchlaufen der Tür



- Unterteilung in 2 Subgoals
- übernimmt folgende Arbeiten
  - Berechnung des Pfads bis zum Schalter
  - das Positionieren des Charakters vor den Schalter
  - das Drücken des Schalters



# Graphen Zusammenfassung

- Prinzip besteht aus benannten Nodes und Links
- Auf dieser Grundlage wird ein Goal-based Path erzeugt
- Graphen sind sehr flexibel
- Voraussetzungen
  - Gute Editingtools und Level-Designer der Graph erstellt
- Je besser der Graph, desto besser das Ergebnis
- Code wird übersichtlicher
- Das System von Subgoals macht das ganze einfach wartbar
- Goals gehören zur Technik des hierarchischen Pathfindings



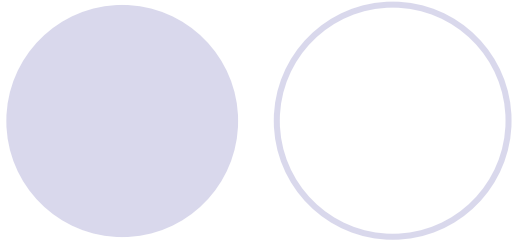


# Graphen

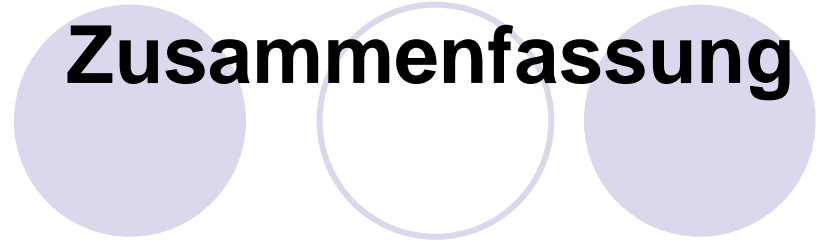
## Goal-based Path Planning, Following und Execution

### Beispiel

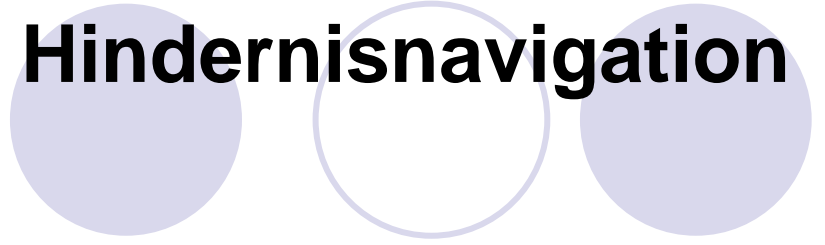
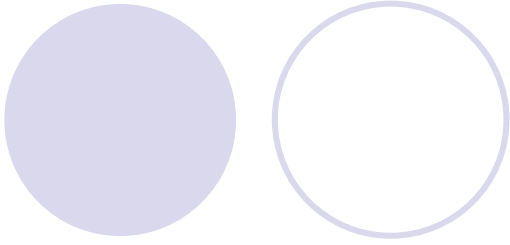
- An der Tafel



# Zusammenfassung



- Es gibt verschiedene Welt Darstellungen für verschiedene Genres
- Hindernisnavigation ist nicht das bloße Umlaufen von Hindernissen
- Hindernisnavigation braucht einen Pathfindingalgorithmus
- Ja nach Komplexität des Software-Agenten wird Hindernisnavigation eingesetzt
- Hindernisnavigation ist die Erweiterung des Pathfindings



- Fragen?