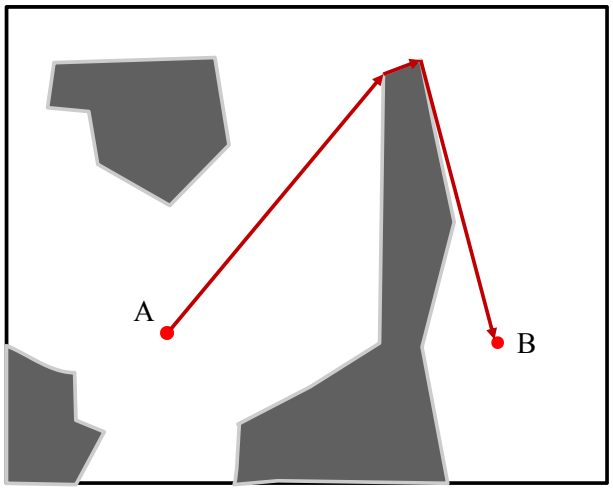
AI Engine und AI Dienste

- Aktionsgenerierung und AI Steuerungen benötigen Basisoperationen, die bestimmte Aufgaben effizient und generisch erledigen:
  - Finden von Ein- und Ausgängen
  - Finden kürzester Wege
  - antagonistisches Verhalten
  - Schwarmverhalten
  - ..
- AI Engine: Sammlung von Diensten die für die Umsetzung der KIs nützlich sind

Wegewahl in offenen Umgebungen

- offene Umgebung: 2D Raum ($\subseteq IR^2$)
- MOBs dürfen sich frei bewegen.
- Hindernisse versperren direkte Verbindungen
- Darstellung der Hindernisse durch:
  - ***Polygone***
  - Pixeldarstellungen
  - beliebige geometrische Flächen (Kreise, Ellipse,..)

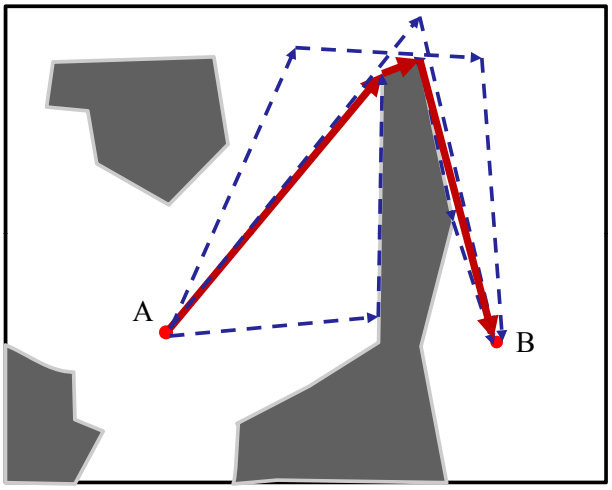


Lösung für Polygon-Darstellung:

- Ableiten eines Graphen für ein Karte auf dem die kürzesten Wege verlaufen (Sichtbarkeitsgraphen)
- Integrieren von Start und Ziel
- Verwendung von Wegewahl-Algorithmen wie Dijkstra, A* oder IDA*

Sichbarkeitsgraphen

- Finden des kürzesten Pfads in einer offenen Umgebung ist eine Suche in einem unendlichen Suchraum
- **Lösung**: Einschränken des Suchraums durch folgende Eigenschaften von optimalen Pfaden:
  - **Die Wegpunkte jedes kürzesten Pfades sind entweder Start, Ziel oder Ecken der Hindernis-Polygone.**
  - **Pfade dürfen die Polygone nicht schneiden**



- Der kürzeste Pfad in der offenen Umgebung *U* ist auch Teil des Sichtbarkeitsgraphen $G_U(V,E)$.

Sichbarkeitsgraphen

Umgebung: *U*

- Menge von Polygonen $U=(P_1,..,P_n)$ **(Hindernisse)**
- Polygon P: Planarer zyklischer Graph: $P = (V_P,E_P)$

Sichtbarkeitsgraph: $G_U(V,E)$

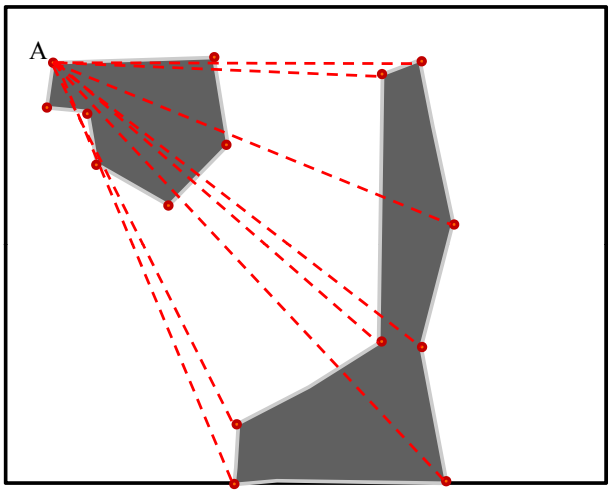
- *Knoten:* Ecken der Polygone P = *{V1,..,Vl)* in *U:* $V_U = \bigcup_{P \in U} V_P$
- *Kanten:* Alle Kanten der Polygone mit allen Kanten zwischen Knoten aus unterschiedlichen Polygonen, die keine Polygonkanten schneiden.

$$E_U = \bigcup_{P \in U} V_P \cup \{(x,y) \mid x \in P_i \wedge y \in P_j \wedge i \neq j \wedge \forall_{P \in U} \forall_{e \in E_P} : (x,y) \cap e = \{\}\}$$

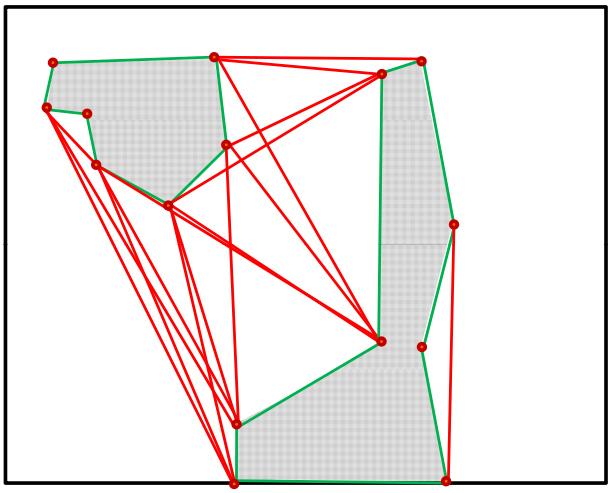
Anmerkung: Diese Definition beinhaltet einen naiven Algorithmus ($O(n^3)$) zur Konstruktion von Sichtbarkeitsgraphen.

Die Ableitung von Sichtbarkeitsgraphen, kann mehrfach optimiert werden. (O'Rourke 87: $O(n^2)$)

Beipspiel: Sichtbarkeitsgraph



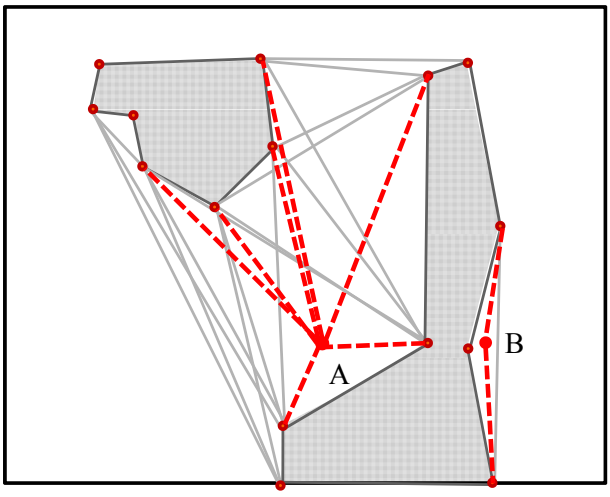
Kanten für den Knoten A die getestet und verworfen werden.



Sichtbarkeitsgraph: Rote Segmente verlaufen zwischen den Polygonen. Grüne Segmente markieren die Grenzen der Polygone.

Erweiterung um Start und Zielknoten

- Sichtbarkeitsgraph kann für statische Umgebungen vorberechnet werden.
- Bewegliche Objekte müssen vor der Berechnung in den Graphen integriert werden
- Einfügen von Start S und Ziel Z als Punkt-Polygone
- Verbinden der neuen Knoten mit allen Ecken, falls kein Polygonschnitt auftritt



Dijkstra

Verwendete Datenstrukturen:

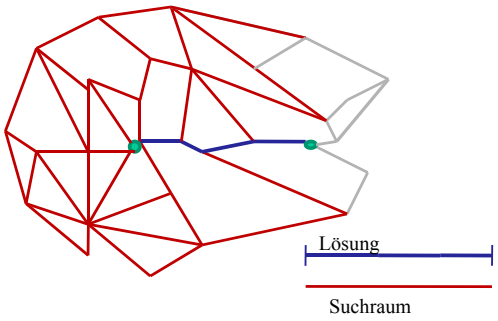
- PriorityQueue Q (enthält Pfade absteigend nach Kosten sortiert)
- Kostentabelle T (enthält Kosten für aktuell besten Pfad für alle besuchten Knoten)

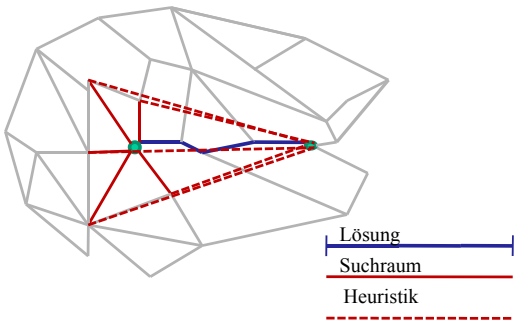
Peseudo-Code:

```
FUNCTION Path shortestPath(Node Start, Node Ziel)
    Q.insert(new Path(start,0))
     WHILE(Q.notIsEmpty())
          Path aktPath = Q.getFirst()
          IF aktPath.last()  == Ziel THEN                    //Ergebnis gefunden
             return aktPath
          ELSE
             FOR Node n  in aktPath.last().nachfolger() DO   //erweitern des aktuellen Pfades
                 Path newPath = aktPath.extend(n)
                 IF newPath.cost()<T.get(newPath.last()) THEN   //update falls bisher optimal
                      T.update(newPath.last,newPath.cost)
                      Q.insert(newPath,newPath.cost)
                 ENDIF
             ENDDO
          ENDIF
    ENDWHILE
    RETURN NULL                                              //es gibt keinen Pfad
ENDFUNCTION
```


A*-Suche

- Dijkstra‘s Algorithmus hat keine Information über Richtung in der das Ziel liegt
  => Expansion der Suche in alle Richtungen, bis das Ziel gefunden wird
- Aber es gibt Hinweise darauf in welche Richtung man suchen sollte
- A*-Suche formalisiert diese „Hinweise“ in einer optimistischen vorwärts Abschätzung: h(n,Ziel) für einen Knoten n.
- h(n,Ziel) gibt eine untere Schranke der Kosten an, die zum Erreichen des Ziels mindestens noch benötigt werden
- Verbessert Suchreihenfolge durch Sortierung nach minimalen Gesamtkosten zum Ziel
- Ermöglicht das Abschneiden von Pfad P (Pruning) sobald ihre Kosten zuzüglich der Heuristik größer sind als das beste bisherige Ergebnis:
  P.cost()+h(pfad1.last(),Ziel) > bestPath.cost()
- Standardheuristik für Abschätzung der Strecke:
  Euklidischer Abstand zwischen der jetzigen Position und des Ziels.





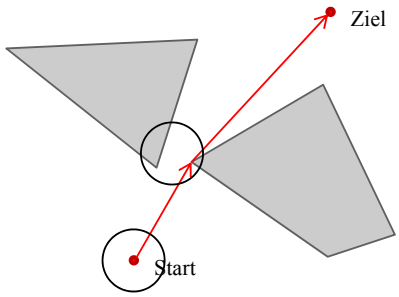
Pseudo-Code: A*-Suche

Peseudo-Code:

```
FUNCTION Path shortestPath(Node Start, Node Ziel)
    Q.insert(new Path(start,0+h(Start,Ziel)))
     WHILE(Q.notIsEmpty())
          Path aktPath = Q.getFirst()
          IF aktPath.last()  == Ziel THEN                    //Ergebnis gefunden
             return aktPath
          ELSE
             FOR Node n  in aktPath.last().nachfolger() DO    //erweitern des aktuellen Pfades
                Path newPath = aktPath.extend(n)
                IF newPath.cost() +h(newPath.getLast,Ziel)
                    <T.get(newPath.last()) THEN               //update falls bisher optimal
                    T.update(newPath.last, newPath.cost() +h(newPath.getLast,Ziel))
                    Q.insert(newPath, newPath.cost() +h(newPath.getLast,Ziel))
                ENDIF
             ENDDO
          ENDIF
    ENDWHILE
    RETURN NULL                                              //es gibt keinen Pfad
ENDFUNCTION
```

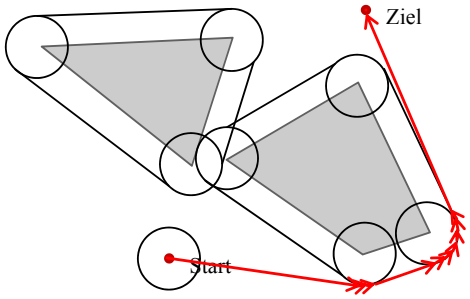

Sichtbarkeitsgraphen für ausgedehnte Objekte

- MOBs haben in der Regel eine räumliche Ausdehnung: Kreis oder Polygon
- Sichtbarkeitsgraph ist nur für Punkt-Routing geeignet
- Anpassung des Sichtbarkeitsgraphen bzgl. der Ausdehnung des Objekts: Polygone werden um die räumliche Ausdehnung erweitert (Minkowski Summe)



Probleme bei der Lösung:

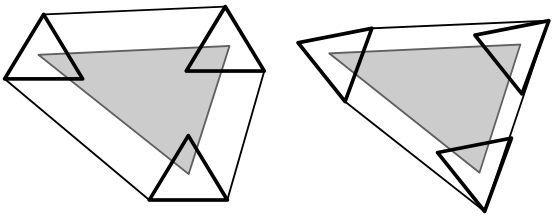
- Bei kreisförmiger Ausdehnung: Kreisförmige Umgebungen haben unendlich viele Ecken
  => Sichtbarkeitsgraph so nicht ableitbar
- Bei Polygon-Umgebung: Drehung der Objekte sollte Berückishtigt werden
  => Für jede Rotation ist eine eigene Erweiterung notwendig



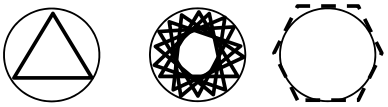
Sichtbarkeitsgraphen für ausgedehnte Objekte

Lösungsansatz:

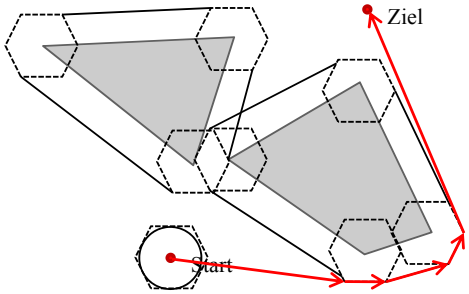
- Polygone werden durch Rotationsfläche approximiert
  => Kreis
- Kreis werden mit minimalen umgebenden Polygonen (MUP) approximiert
  => z.B. Hexagon, Octagon
- Bilde Mikowski-Summe anhand der MUPs und leite Sichtbarkeitgraphen ab.



Minkowski-Summe mit unterschiedlicher Rotation derselben Ausdehnung



Doppelte Approximation über Rotationsfläche und minimales umgebendes Hexagon



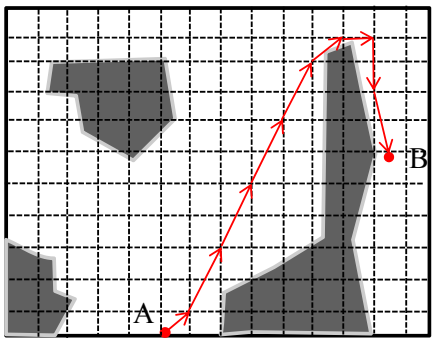
Anmerkunng:

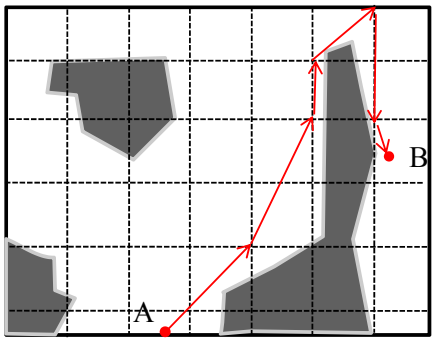
- Wege sind nicht optimal
- Durchgänge werden konservativ betrachtet
- Kurven werden eckig gelaufen
- in MMO sollte es nur eine beschränkte Auswahl an Umgebungsgrößen geben, da jede ihren eigenen Graphen benötigt

Weitere Methoden zur Wegewahl

Weitere Methoden:

- Approximation der Polygone durch Polygonen mit weniger Ecken
- hierarchisches Routing für längere Strecken
- Vorberechnung von kürzesten Wegen und Speicherung in Datenstrukturen
- Verwendung von Grid-basierten Graphen die Karte in Gridzellen einteilen und Routing über Zellenmittelpunkte laufen lassen.





Fazit:

Wegewahl ist ein altes aber immer noch aktives Forschunggebiet in sehr vielen Teilbereichen der Informatik.

Reaktion auf Spielerverhalten

Wie kann eine AI auf das Verhalten des Spielers reagieren?

- AI liest aktuellen Zustand des Spielers und leitet daraufhin eine angepasste Aktion ab.
- Auswahl der passenden Aktion über

  => Vorberechnen/Schätzen des Ergebnisses der Handlung

  => Bewerten des erwarteten Ergebnisses mit Bewertungsfunktion/Heuristik

Beispiel: Option1: Monster M schlägt Spieler S
=> Spieler S hat noch 900 HP

Option2: Monster M flieht vor S
=> Spieler S hat 1000 HP

Bewertung: Option 1 ist besser, da der Feind 100 HP verliert.

Antagonistische Suche

Problem: Einfaches Modell berücksichtigt nicht, dass der Gegner ebenfalls handelt, um selbst seinen Vorteil zu vergrößern.

Im Beispiel:

M greift S an und schlägt M für 100 HP

$\Rightarrow$ S chlägt M für 1000 HP

Ergebnis: S hat 900 HP und M ist tot

Antagonistische Suche aus der Spieltheorie gibt einen formellen Rahmen für reaktives Verhalten. Basisfall:

- Zugbasierte Spiele, Handlungen lassen sich sequentialisieren
- endliche Anzahl an Handlungsalternativen bei jedem Zug

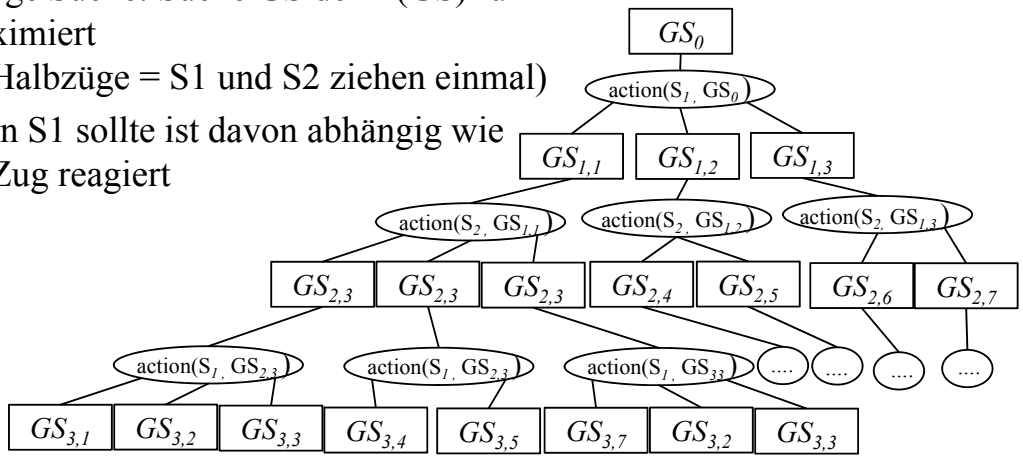
Antagonische Suche

Gegeben:

- GS_i: Spielstand vor Zug i. Spieler: S1 und S2.
- Aktionen von Spieler S_j für GS :action(S_j, GS_i) = {A_1,.., A_k)
- Bewertungsfunktion $H: GS \rightarrow IR$ (he höher desto besser für Spieler S)

Suchbaum:

- vollständiger Baum: enthält alle möglichen Spielverläufe (idR zu groß)
- unvollständige Suche: Suche GS der H(GS) für h Züge maximiert
  (1 Zug = 2 Halbzüge = S1 und S2 ziehen einmal)
- Auswahl von S1 sollte ist davon abhängig wie S2 auf den Zug reagiert

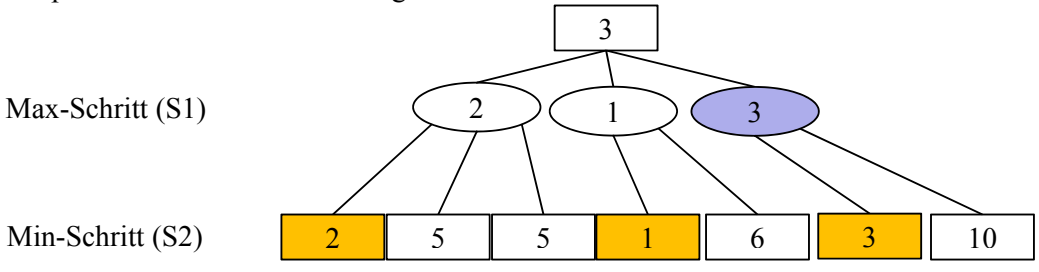


Min-Max Suche in Antagonistische Suchbäumen

- Bewerte einen Zug A so, dass H(GS) nach Reaktion von S2 (versucht H zu minimieren) maximal wird.
- Suchtiefe:
  - Vorgegebene Anzahl an Zügen

  ⇒ Zeit kann variieren und ist schwer abschätzbar

  ⇒ unruhige Stellungen machen abschneiden einiger Äste unverteilhaft
  - Iterative Deepening:
    - mehrfache Berechnung mit steigender Suchtiefe
    - bei Time-Out: Abbruch und Verwendung der letzten vollständigen Bewertung (da sich Aufwand im Schnitt verdoppelt, ist Aufwand ca. doppelt so hoch)
  - unruhige Stellungen: einzelne Äste werden weiter expandiert wenn Blätter unruhig sind.



Alpha-Beta Pruning

Idee: Wenn es schon einen Zug gibt der selbst nach Gegenreaktion noch mit α bewertet wird, können alle Äste die weniger als α Wert erzeugen gepruned werden.

- α: S1 erreicht mindestens α auf diesem Teilbaum (H(GS) > α)
- β: S2 erreicht maximal β auf diesem Teilbaum (H(GS) < β)

Algorithmus:

- Durchlaufe Suchbaum mit Tiefensuche und fülle innere Knoten bei Rückweg zur letzten Verzweigung
- beim Berechnen der inneren Knoten:
  Wenn β < α dann
  - prune restlichen Teilbaum
  - setze β-Wert für den Teilbaum falls seine Wurzel ein Min-Knoten ist
  - setze α -Wert für den Teilbaum falls seine Wurzel ein Max-Knoten ist

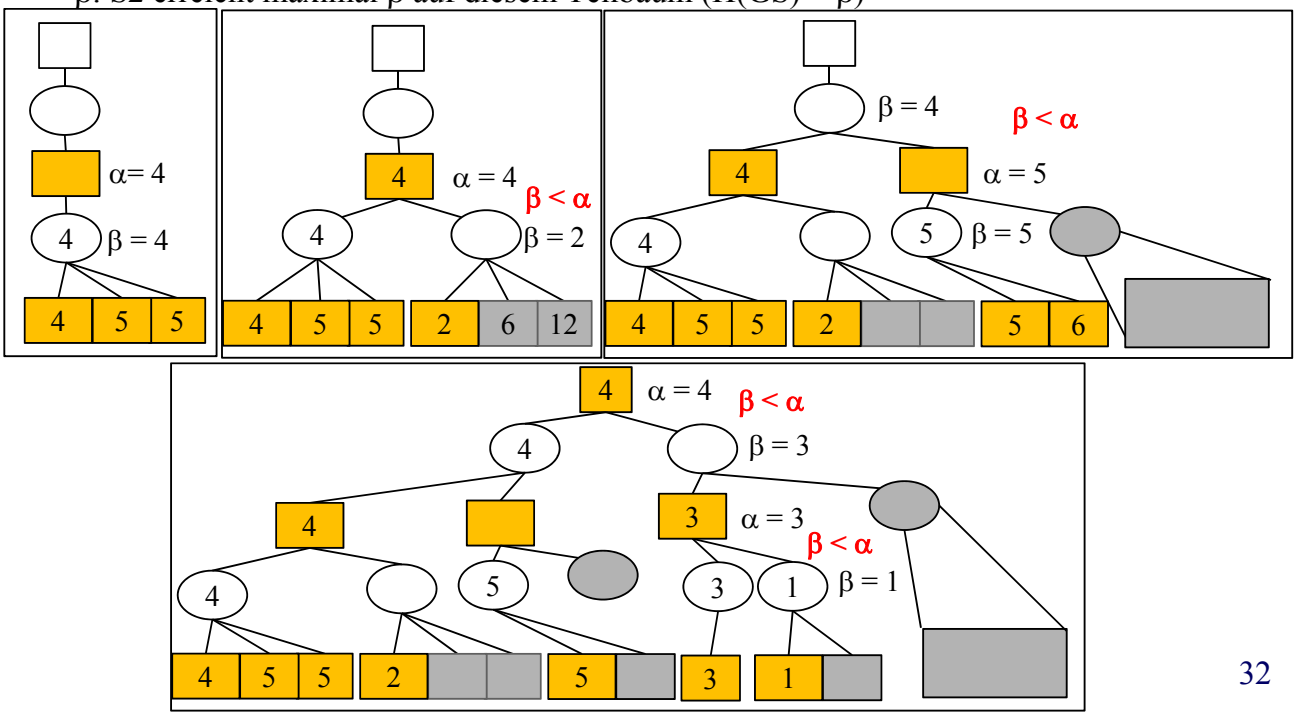
  Sonst setze β-Wert auf Minimalwert bei Min-Knoten

  setzr α-Werte auf Maximalwert bei Max-Knoten

Alpha-Beta Pruning

Idee: Wenn es schon einen Zug gibt der selbst nach Gegenreaktion noch mit α bewertet wird, können alle Äste die weniger als α Wert erzeugen gepruned werden.

- α: S1 erreicht mindestens α auf diesem Teilbaum (H(GS) > α)
- β: S2 erreicht maximal β auf diesem Teilbaum (H(GS) < β)



Anwendbarkeit für allgemeine Spiele

Erweiterung auf allgemeinere Spiele können meist durch Anpassung der Bewertungsfunktion erreicht werden:

- probabilistische Spiele: Maximieren des Erwartungswerts (z.B. Backgammon)
- unvollständige Informationen (z.B. Stratego, Skat,..)

Erweiterung auf mehrere Spieler:

- Min-Knoten berücksichtigen die Aktionen aller Spieler.
  (Hohe Zahl von Spielerreaktionen können durch Kombination mehrerer Reaktionen entstehen)

Wegfallen der Zeit-Synchronization:

- Prinzipiell möglich aber Anzahl der möglichen Spielverläufe wächst im allgemeinen Fall exponentiell
- Ohne Einschränkung des Suchraums schwierig zu berechnen

Fazit: Die Grundidee ist prinzipiell auf beliebige Spiele übertragbar, aber der Einsatz scheitert meist an der hohen Anzahl von Spielverläufen, die durch die Anzahl der Spieler, mögliche zeitliche Verläufe und allgemeinen Handlungsoptionen entstehen können.

Lernziele

- Aufbau der Steuerung von computergesteuerten Entitäten
- Regelbasierte Steuerung
- Zustandsautomaten für das Verhalten
- Typische Aufgaben von AI Engines
- Wegewahl in offenen Umgebungen
- Wegewahl mit ausgedehnten Objekten
- Antagonistischen Suchen
- Min-Max Suche
- Alpha-Beta Pruning

Literatur

- Nathan R. Sturtevant
  Memory-Efficient Abstractions for Pathfinding
  In Artificial Intelligence and Interactive Digital Entertainment Conference (AIIDE), 2007.
- Nathan R. Sturtevant, Michael Buro
  Partial pathfinding using map abstraction and refinement
  In Proceedings of the 20th national conference on Artificial intelligence, 2005.
- Joseph O'Rourke:
  Computational Geometry in C.
  2nd Edition, Cambridge University Press, 1998