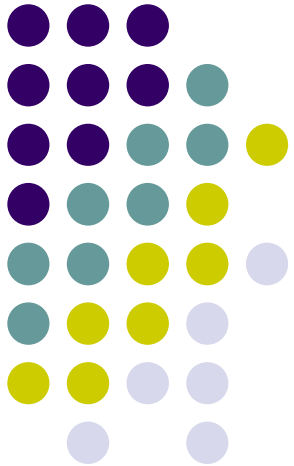
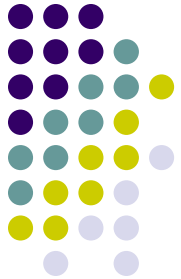
A* Engines

Spatial Planning SS07

Jan Petendi



Zu lösende Teilaufgaben



1. Sichtung und Kategorisierung der Optimierungen von A* bei Spielanwendungen
2. Entwicklung einer Architektur einer A* Engine, die parallele Anfragebearbeitung unterstützt
3. Implementierung der A* Engine für ein mobiles Gerät (PDA)

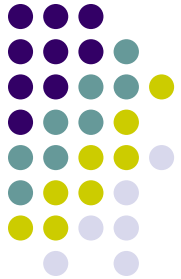
Referat:

Anforderungen, Herangehensweise, grobe technische Umsetzung

Hausarbeit mit kommentiertem Quellcode:

zusätzlich genaue technische Umsetzung

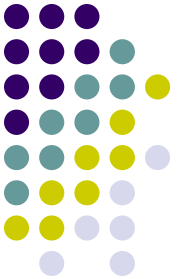
Relevanz



- A* ist generisch und findet deshalb Verwendung in vielen Bereichen
- häufig werden Lösungen entwickelt, die optimiert auf ein bestimmtes Problem sind und sich nur schlecht erweitern lassen

Notwendigkeit einer Lösung, die die generischen Eigenschaften von A* hervorhebt und somit verwendbar für eine Vielzahl von Problemen ist

Definition „A* Engine“



Eine generische Komponente zur Lösung von Problemen mit dem A Algorithmus*

Relevanz für mobile Geräte

- Stand der Technologie vergleichbar mit der von PCs vor 10 Jahren
- Markt mobiler Spiele boomt und soll in naher Zukunft PC- und Konsolenspiel überholen

  (Gartner Says Worldwide Mobile Gaming Revenue to Grow 50 Percent in 2007. Website, Juni 2007. – http://www.gartner.com/it/page.jsp?id=507467*)*

- noch keine vergleichbare Lösung entwickelt

FluPa-Guide, http://www.kinf.wiai.uni-bamberg.de/flupa/

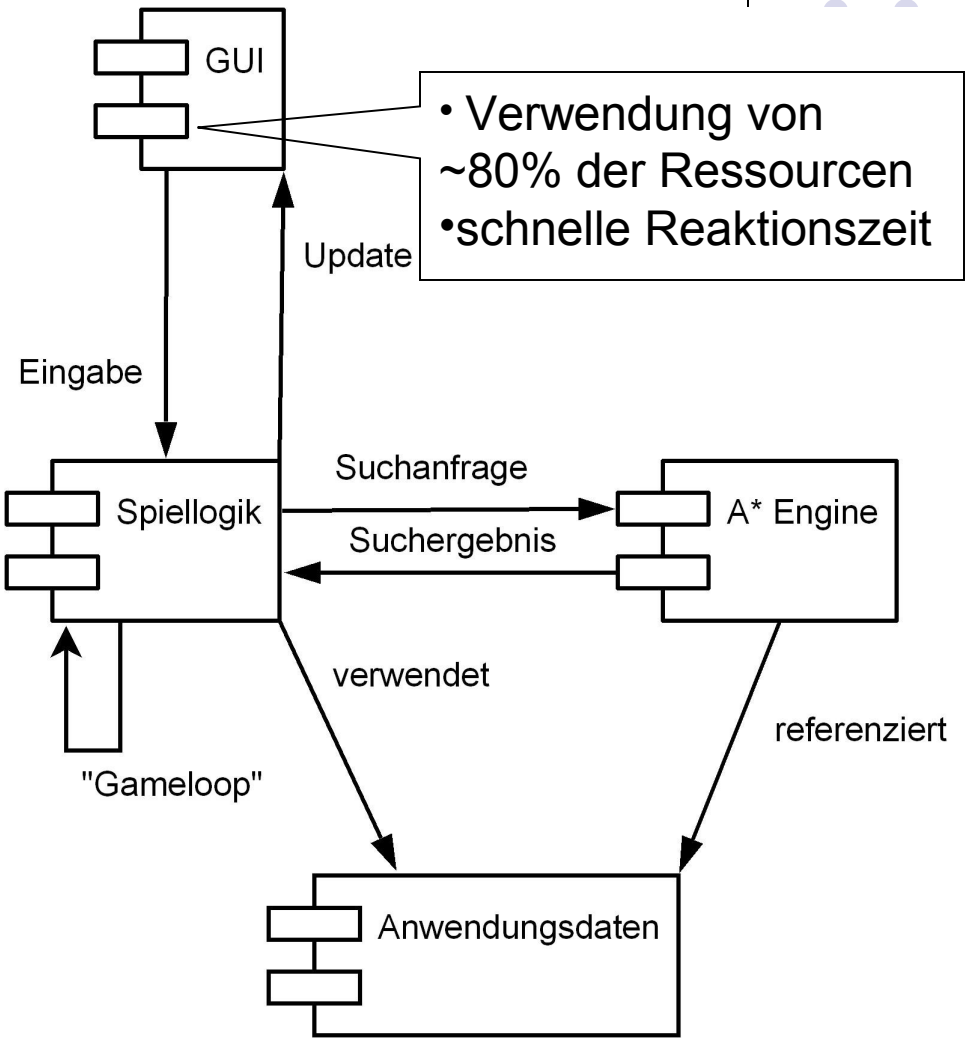
Anwendungsmöglichkeiten von A*

- allgemein: Suche in einem Graphen
  - Wegfindung
  - spezielle Spielanwendungen:
    - Terrainanalyse
    - Straßenbau
    - Städtebau
  - Planung
    - heuristic Search Planning

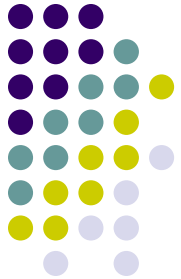
z.B. : (Bonet, Blai ; Geffner, Hector: Planning as heuristic search. In: Artificial Intelligence129 (2001), Nr. 1-2, 5-33. citeseer.ist.psu.edu/bonet01planning.html)

Anforderungen der Spielarchitektur

- saubere Integration in die bestehende Architektur
  - „angemessene“ Verwendung zugeteilter Ressourcen
  - Korrektheit und Optimierung auf Laufzeitsystem
- geringe Latenzzeit
- Belastbarkeit
  - Bearbeitung vieler (paralleler) Anfragen

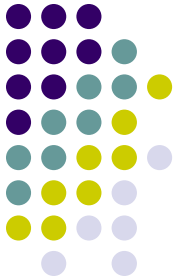


Problemstellung A*



- Speicherkomplexität
  - Breitensuche, Verwaltung von Open- und ClosedList
  - Suchraum
- Zeitkomplexität
  - Dauer für die Durchführung eines Suchschrittes
  - Anzahl der Suchschritte für das Berechnen einer Lösung

Optimierungen



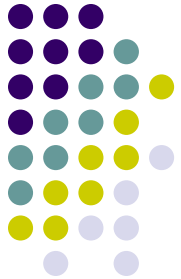
Designebene:

- Suchraumrepräsentation
- Kostenfunktion
- Heuristik

Implementierungsebene:

- Datenstrukturen
- Codeoptimierung

Suchraumrepräsentation



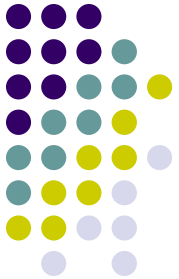
linear:

- Gitter
- Polygone
- …

hierarchisch:

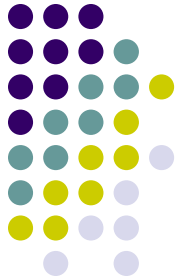
- hierarchische Clusterungen obiger Strukturen
  - ➔ Referat *Thomas B.*

Kostenfunktion



- Unterteilung des Suchraumes in „frei“ und „blockiert“ unzureichend
  - ➔ zusätzliche Informationen wie z.B. Höhe oder Terraintypen bei der Kostenfunktion berücksichtigen
  - ➔ Terraintyp „Pfad“ mit sehr geringen Kosten

Heuristik



- TradeOff: Schnelligkeit vs. Genauigkeit
  - ➔ in Spielanwendungen ist Schnelligkeit wichtiger
- Verwendung von bewährten Heuristiken und Optimierung auf die Problemstellung
  - dynamische Heuristiken
  - Visualisierung des Suchverhaltens und iteratives optimieren

➔ Stärke von A*, die auch voll ausgeschöpft werden sollte

Heuristik(2)

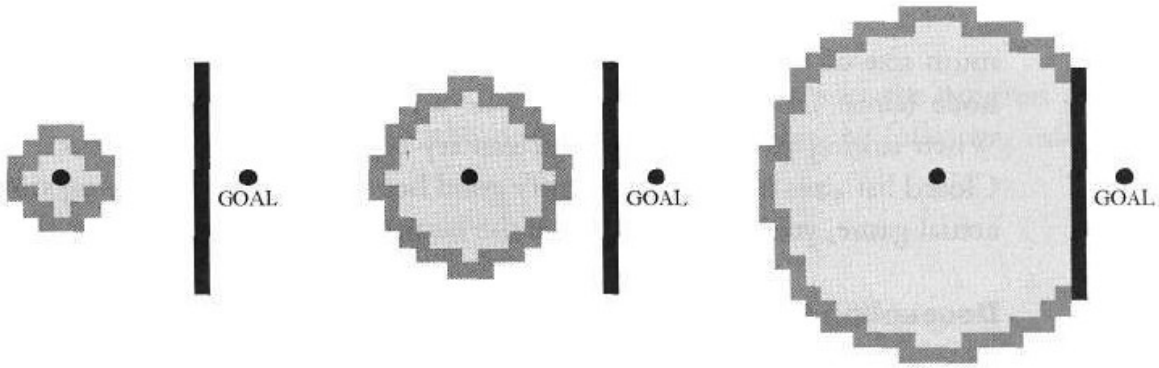


FIGURE 3.5.4. The heuristic cost of zero.

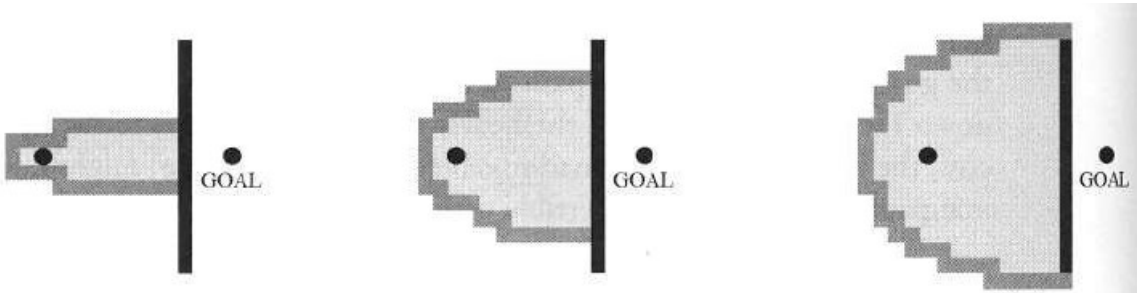
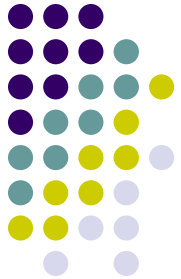


FIGURE 3.5.5. The heuristic cost using Euclidean distance to goal.

Datenstrukturen



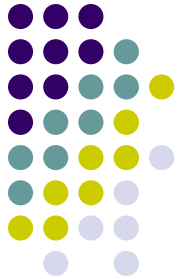
- allgemein: möglichst Datenstrukturen selbst implementieren
- OpenList: optimiert auf sortiertes Add und Remove
  - ➔ z.B. PriorityQueue implementiert durch Binary Heap Tree
- ClosedList: optimiert auf Contains
  - ➔ z.B. HashTable

➔ siehe auch: Patel, Amit,J., "Amit's Game Programming Information“ : http://theory.stanford.edu/~amitp/GameProgramming/ImplementationNotes.html

Codeoptimierungen

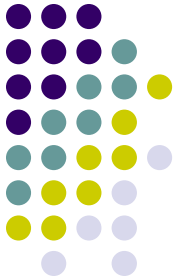
- letzter Schritt, wenn Korrektheit bewiesen wurde
- allgemein:
  - Kenntnis der Laufzeitumgebung
    - native/interpreted/virtual machine/jit compilation
  - Codeprofilation
- spezieller:
  - Objektpool besser als dynamische Speicherallozііerung (wiederholtes „new“-dispose())
  - manuelle Speicherbereinigung besser als warten auf Garbage Collector
  - …
  - siehe auch z.B.:
    Higgins, Daniel: How to Achieve Lightning-Fast A*. In: Game ProgrammingWisdom (2002), S. 133–145

Entlastung der A* Engine



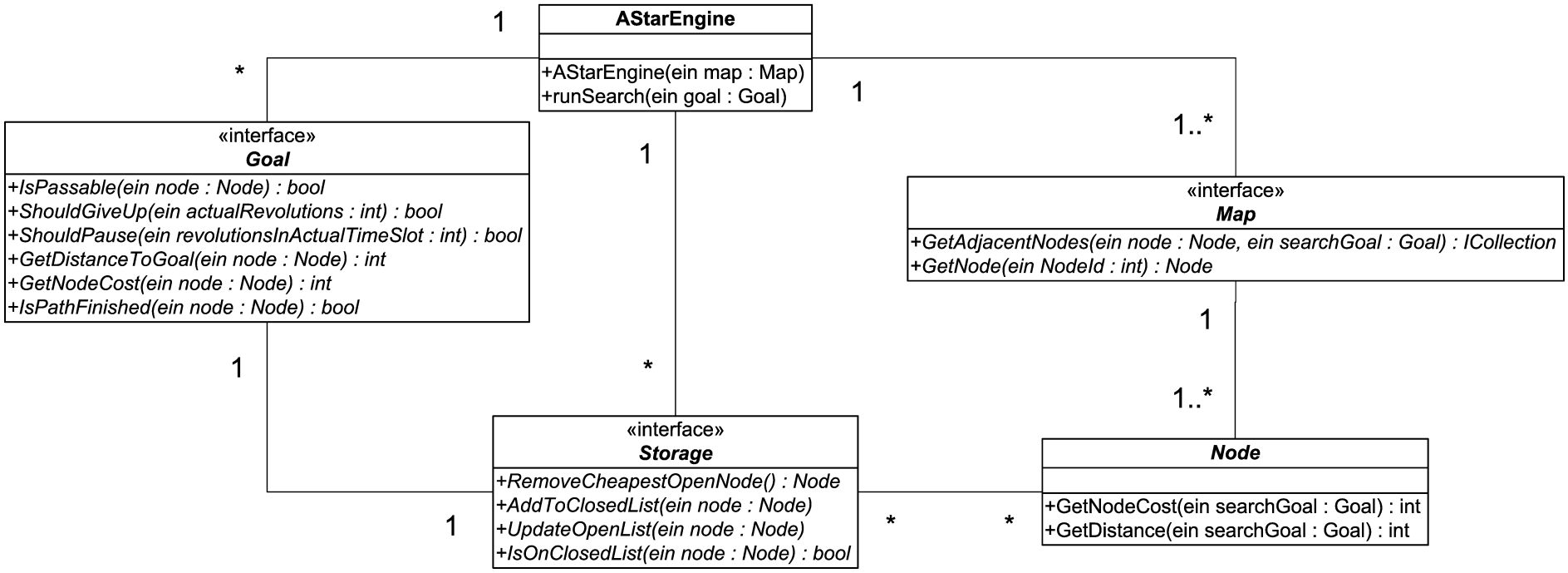
- Unterstützung durch die Spiellogik mit:
  - filtern überflüssiger Suchanfragen
  - Zusammenfassung von Gruppenbewegungen
  - „Intelligent Moving“
    - Start einer Bewegung noch vor Rückgabe des berechneten Pfades in grober Zielrichtung
    - wiederholte Berechnung kleinster Wegabschnitte

Zusammenfassung der Anforderungen

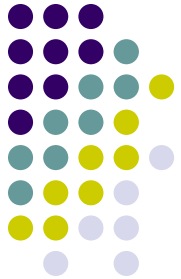


- geringe Latenzzeit
- Abstraktion von der Suchraumrepräsentation
- effiziente Verwendung von Rechen- und Speicherressourcen
- generisch - passend für eine Vielzahl von Anwendungen
- Architektur verwendbar für alle O-O-Sprachen

Struktur der A* Engine

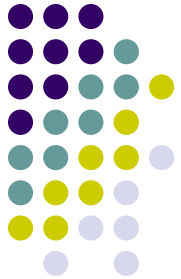


Storage



- *RemoveCheapestOpenNode()* : entfernt den für die aktuelle Suchanfrage günstigsten Knoten von der OpenList und gibt ihn zurück
- *UpdateOpenList(Node)* : fügt den Knoten zur OpenList hinzu bzw. aktualisiert diesen
- AddToClosedList(Node) : fügt den Knoten zur ClosedList hinzu
- IsOnClosedList(Node) : prüft, ob der Knoten bereits besucht wurde

Map



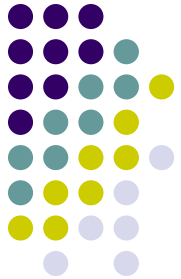
- *GetAdjacentNodes(Node):* gibt alle Knoten in direkter Nachbarschaft zurück
- *GetNode(NodeId):* gibt den Knoten mit der betreffenden Id zurück

➔ zusätzlich Methoden zur Verwaltung des Suchraums (Anzahl referenzierter Knoten, Caching von Knoten,…)

Goal

- *IsPassable(Node)* : prüft, ob der Knoten in den Pfad einbezogen werden kann
- *GetDistanceToGoal(Node)* : gibt die heuristischen Restkosten zum Ziel an
- *GetNodeCost(Node)* : gibt die Kosten dieses Knotens an
- *IsPathFinished(Node)* : prüft, ob der Zielknoten erreicht wurde
- *ShouldGiveUp(ActualRevolutions)* : prüft, ob die Höchstanzahl von Suchschritten erreicht wurde und die Suche abgebrochen werden soll
- *ShouldPause(RevolutionsInActualTimeSlot)* : prüft, ob die Höchstanzahl von Suchschritten im aktuellen Zeitfenster erreicht wurde und die Suche pausiert werden soll

Node



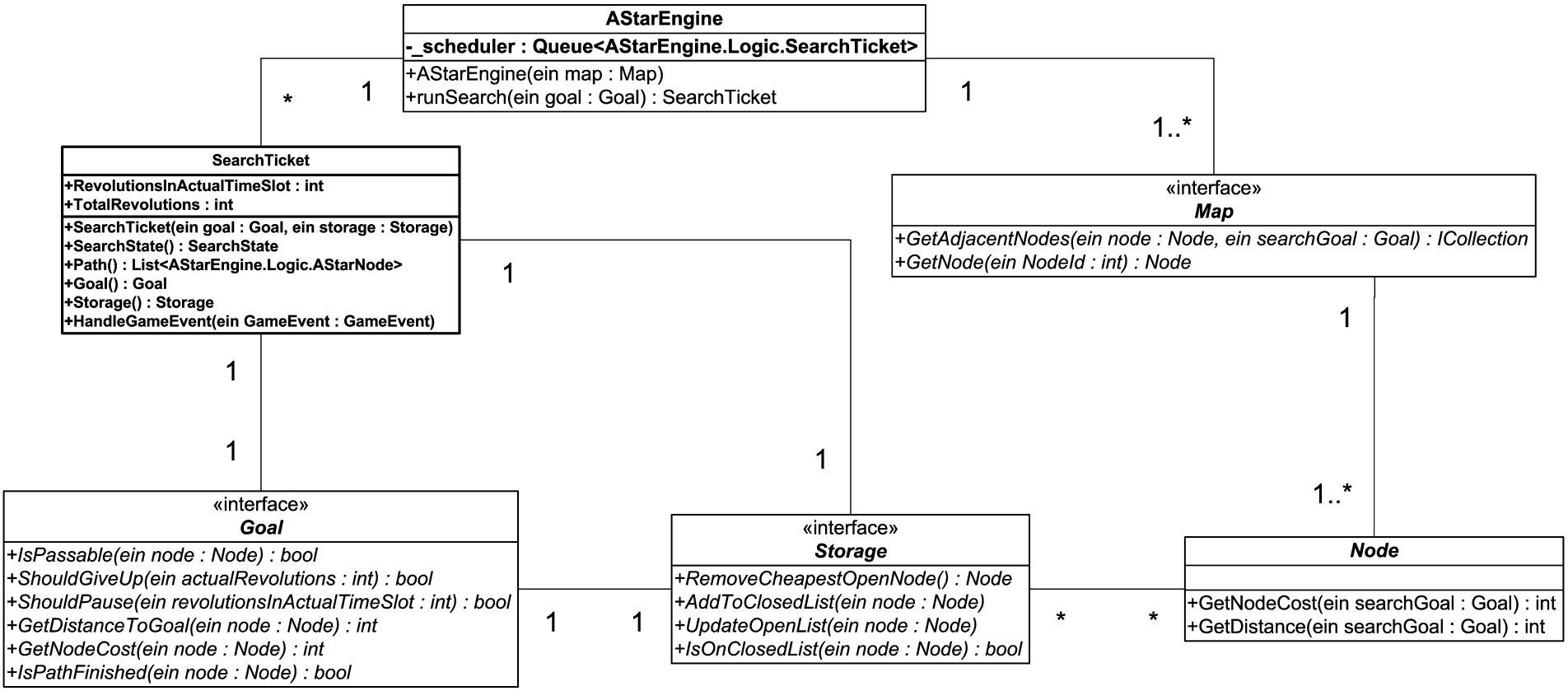
Attribute:

- tag: referenziert das Anwendungsdatenobjekt
- zusätzliche Attribute zur Unterstützung der Heuristik

Methoden:

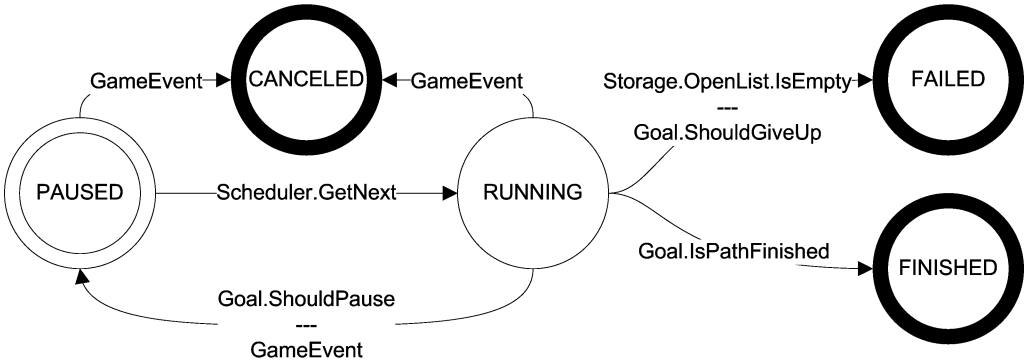
- *GetDistance(Goal)* : gibt die anfragespezifisch berechneten Restkosten an
- GetNodeCost(Goal) : gibt die anfragespezifisch berechneten Kosten an

Erweiterung für parallele Anfragebearbeitung



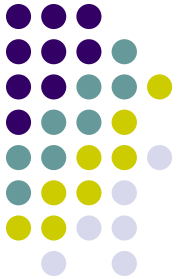
SearchTicket

allgemein: Kapselung des Zustands einer Suchanfrage und Schnittstelle zur Einflussnahme auf diesen



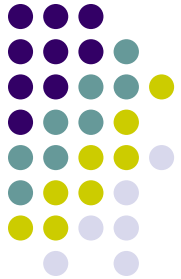
- CANCELED : die Suchanfrage wurde abgebrochen, bevor ein Pfad ermittelt werden konnte
- FAILED : es wurde kein Pfad gefunden
- FINISHED: die Suchanfrage wurde erfolgreich abgeschlossen (ein Pfad wurde gefunden)
- PAUSED: die Suchanfrage pausiert
- RUNNING : die Suchanfrage wird gerade bearbeitet

SearchTicket (2)



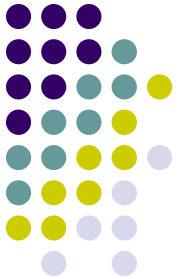
- SearchState : repräsentiert den Zustand, in welchem sich die Suchanfrage momentan befindet
- Storage : referenziert den Speicher der Suchanfrage
- Goal : referenziert die Suchanfrage selbst
- Path : referenziert den Pfad, der ermittelt wurde, hier müssen geeignete Maßnahmen getroffen werden, damit der Zugriff nur im Zustand FINISHED gewährt wird
- TotalRevolutions : enthält die Gesamtanzahl von Suchschritten, die für diese Suchanfrage aufgewendet wurden
- RevolutionsInActualTimeSlot : enthält die Anzahl von Suchschritten, die zwischen dem letzten Wechsel vom Zustand PAUSED in RUNNING aufgewendet wurden

Scheduling



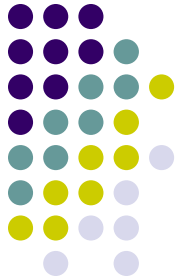
- allgemein: stark abhängig von der Problemstellung
- ➔ Priorität von Anfragen, deren Pfad im sichtbaren Bereich beginnt
- ➔ Priorität von Anfragen, die einen kurzen Pfad erzeugen (=> Heuristik)
- ➔ Priorität von Anfragen, die vom Benutzer direkt erzeugt wurden

Implementierung



- dotNet Compact Framework
  - „abgespeckte“ Version der Desktopversion
  - unterstützt von allen aktuellen Windows Betriebssystemen auf mobilen Endgeräten
  - jit-Compilation
  - Möglichkeit neben C# auch VB.net oder J# (Microsoft Java) zu verwenden [auch gemeinsam]

Implementierung (2)



- C#
  - unterstützt „event driven programming“ gut
  - Unterstützung komplexer Wertetypen - *structs*
- Visual Studio 2005 Pro
  - Emulierung mobiler Anwendungen auf dem Desktop dürftig
    - ➔ zu Demonstrationszwecken Portierung auf Desktopumgebung und Installation auf TestPDA