Modul: Künstliche Intelligenz
Universität Rostock

Projektaufgabe: Wumpuswelt

Lösung und Dokumentation

Sven Berger, 209204572
Martin Müller, 209203328
Nils Richter, 6200839
Stephan Saß, 209204373
John Trimpop, 209206264

Inhaltsverzeichnis

1. Allgemeines

Im Rahmen des Moduls Künstliche Intelligenz der Universität Rostock für den Bachelor Studiengang Informatik ist ein Gruppenprojekt in Form von einer Programmier- und Analyseaufgabe zu erstellen.
Die Hauptaufgabe besteht darin in der sogenannten, vorgefertigten „Wumpuswelt“ einen intelligenten Agenten zu implementieren, welcher in der Lage ist, die in der Aufgabenstellung erwähnten Ziele selbständig zu lösen.
Die Wumpuswelt ist ein zweidimensionales, Schachbrett ähnliches Feld, das durch Wände begrenzt ist, Gruben- und Goldfelder enthält und in dem sich der Agent und der Wumpus, ein übel riechendes, gefährliches Monster, frei bewegen können.
Der Agent und der Wumpus verbrauchen jeweils einen Zug, um sich drehen, fortbewegen oder sonstige Aktionen durchführen zu können; Wände und Gruben sind dabei nicht passierbar, ebenfalls kann nicht diagonal gegangen werden. Der Wumpus hat um sich herum eine acht Felder große Geruchsausdehnung, welche der Agent in der Lage ist zu erkennen, sobald er sich in einem direkt daneben befindlichen Feld aufhält.

Das Projekt ist in drei verschiedene Grundaufgaben unterteilt, welche verschiedene Modifikationen der Wumpuswelt voraussetzen.
Die erste Aufgabe beschränkt sich auf Suchverfahren und ihre Effizienz. Der Wumpus befindet sich noch nicht in der Höhle. Aufgabe zwei beschäftigt sich mit genetischen Algorithmen. Der Agent „kämpft“ nun gegen den Wumpus und sollte dies möglichst effizient vollführen. Die letzte Aufgabe bezieht sich auf etwaige Verbesserungsvorschläge. Insgesamt können 100 Punkte erzielt werden.

2. Suche

2.1 Einrichten des Agenten

Sehen Sie für diese Aufgabe bitte die entsprechenden Java-Quelltexte und Java–Klassen.

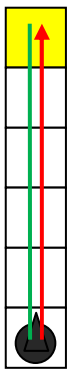
2.2 Implementieren der Suchalgorithmen

Neben den drei geforderten Algorithmen, haben wir noch eine Erweiterung der normalen A*-Suche implementiert (A*-Spezial). Sehen Sie für diese Aufgabe bitte die entsprechenden Java-Quelltexte und Java–Klassen.

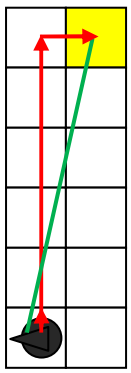
2.3 Heuristik für A*

Für A* haben wir uns folgende Heuristik ausgedacht:
Um dem Agenten einen möglichst schnellen Weg durch die Höhle zu verschaffen und alle Goldfelder einzusammeln, wird der Luftlinienabstand zwischen dem Agenten und dem am nächsten liegendem Goldfeld verwendet. Anschließend wird die gleiche Prozedur bei dem der Luftlinie am günstigsten nächsten Feld angewendet. Der kürzeste Weg zum Gold kann mit dieser Methode äußerst effizient berechnet werden.
Durch die Erweiterung A*-Spezial werden außerdem verschiedene Gesamtwege zu einem Goldfeld vorweg ermittelt und der kürzeste ausgesucht. Damit wird beispielsweise vermieden, dass sich der Agent, falls sich ein Goldfeld in diagonaler Richtung befindet, immer wieder dreht, um zu diesem Feld zu gelangen, und damit wichtige Schritte verschwendet.

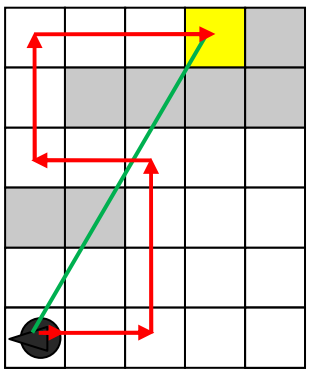
Beweis für die Gültigkeit der Heuristik:
Sei nun n die Länge eines solchen Luftlinienabstandes. Mit Länge ist hier die Anzahl der Felder gemeint, durch die die Luftlinie verläuft.
Der Agent muss nun im besten Falle, nämlich, wenn die Luftlinie genau senkrecht oder waagerecht verläuft und der Agent ihr, ohne sich herumzudrehen, folgen kann, genau n Schritte tätigen. Da er nicht diagonal gehen kann und da sich Gruben innerhalb dieser Luftlinie befinden könnten, kann der Weg um $n + k$ Schritte vergrößert werden, wobei k für die Größe der zusätzlichen Schritte steht.
Illustrationsbeispiel:



Best Case:
n Schritte



n+3 Schritte



Ein schlechter
Fall: n+10 Schritte

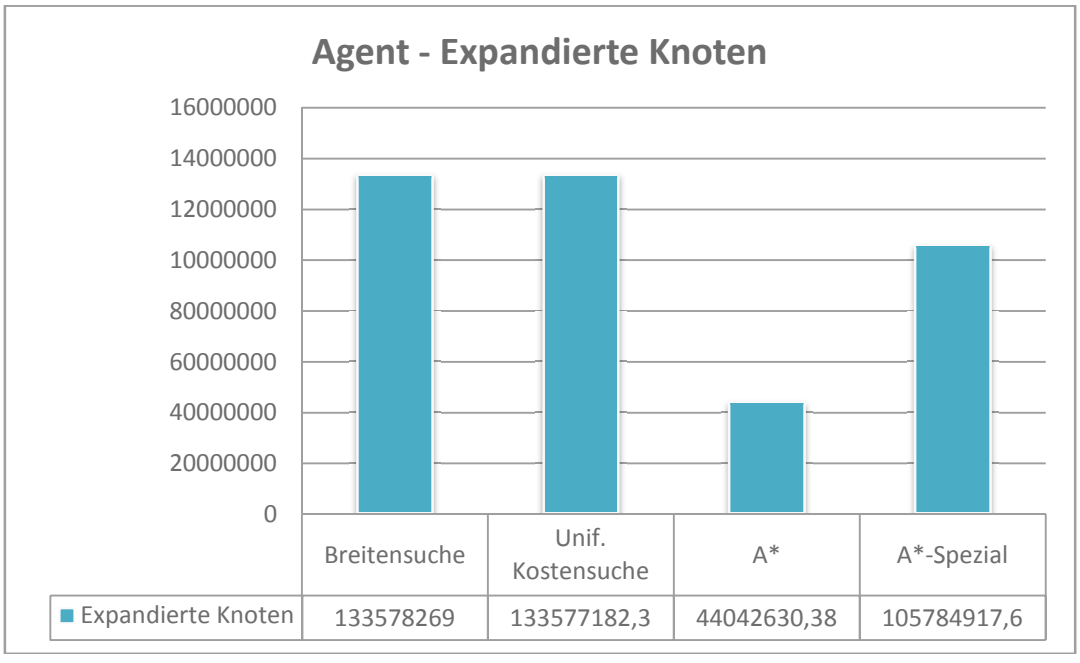
Die Heuristik ist gültig, da es sich um ein begrenztes Spielfeld handelt und die Suche nach dem kürzesten Luftlinienweg auf Grund der endlichen Menge von Goldfeldern immer terminiert. Wird auf einem Feld Gold eingesammelt, kann das Spiel, da sich kein Wumpus in der Höhle befindet, erst beendet werden, wenn jegliches Gold eingesammelt worden ist.

2.4 Testfälle der Suchalgorithmen

Für die Testfälle haben wir verschiedene Zufallseinstellungen des Wumpus-Feldes für jeden Algorithmus unabhängig häufig durchlaufen lassen. Dadurch wurden zwar nur Bruchteile aller Möglichkeiten abgedeckt, aber eine ausreichende Approximation ist dennoch gegeben:

Suchver-fahren	Anzahl der Simulationen	Durchschnitt aller besuchten Felder	Durchschnitt der Gesamtpunktzahl	Durchschnitt der expandierten Knoten
BS	7701	151260	1636500	133578269
UK	7701	151260	1636500	133577182
A*	7701	151260	1636500	44042630
A*-Spez.	7701	151260	1636500	105784918

Es ist, selbst beim Ablesen der Tabelle, schnell ersichtlich, dass A*, das effizienteste Suchverfahre ist. Die grafische Darstellung zeigt dies ebenfalls sehr deutlich:

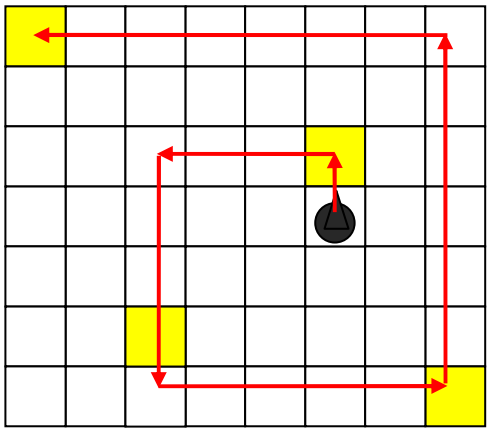


Zwar berechnet A*-Spezial einen effizienteren Weg vorweg als A*, jedoch ist, da alle Verfahren bei jedem Durchlauf immer zu dem gleichen Ergebnis kommen, A* von der in Anbetracht der Rechenleistung am besten, da es am wenigsten Knoten expandieren muss.

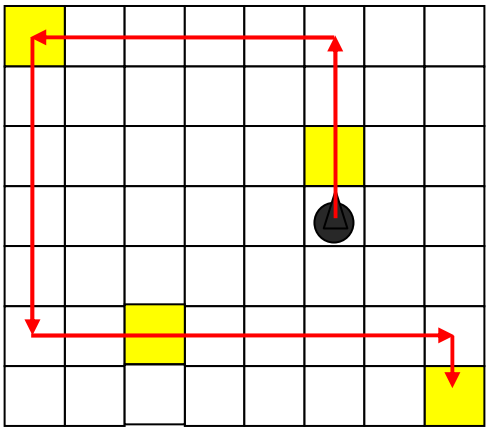
2.5 Gegenbeispiel

Nein, der Weg, den der Agent wählt, ist global gesehen *nicht* optimal, da der tatsächlich kürzeste Weg zwischen mehreren Goldfeldern geringer ausfallen kann, als ein Weg, der von unseren Suchverfahren, welche nur direkt kürzeste Wege zum Gold finden, ermittelt wird. Da die Algorithmen unterschiedlich arbeiten, kann dies natürlich vom Suchverfahren abhängen.
Wie im unten aufgeführten Beispiel leicht zu erkennen ist, wäre der linke Weg eine mögliche Beschreibung des Weges bei A*-Spezial, welches die effizienteste unser Varianten bezüglich der Wegfindung darstellt. Der beste Weg (rechts im Beispiel) ist mit sechs Schritten weniger aber noch effizienter.

Gegenbeispiel:



Weg durch A*-Spezial (31 Schritte)



Ein optimaler Weg (25 Schritte)

3. Genetische Algorithmen

3.1 Reaktiver Agent

Sehen Sie für diese Aufgabe bitte die entsprechenden Java-Quelltexte und Java–Klassen.

3.2 Regeln

Unsere Regeln lauten:

Die Implementierung dieser Regeln finden Sie in den entsprechenden Java-Quelltexten.

3.3 Genetischer Algorithmus

blabla

3.4 Einfluss von Rekombination, Mutation und Größe der Ausgangspopulation

Blabla

3.5 Einbindung stochastischer Wahrscheinlichkeiten

Blabla

4. Fragen