Albert-Ludwigs-Universität Freiburg
Institut für Informatik
Einführung in die Informatik
Sommersemester 2012

Prof. Dr. Wolfram Burgard
PD Dr. Cyrill Stachniss
Dipl. Ing. Jörg Röwekämper
Dipl. Inf. Markus Kuderer

PacMan

Freiwillige Zusatzaufgaben, nicht Klausurrelevant!

Aufgabe 1

Auf der Vorlesungshomepage finden Sie eine PacMan-Version, die die bisherigen Programmieraufgaben enthält. Entpacken Sie das Projekt und kompilieren Sie den Code mittels ant. Sie können das Spiel starten, indem Sie `java PacManMain` im Verzeichnis `bin` eingeben. In der Datei `src/game/control/AIControl.java` finden Sie die Steuerung der Monster. Die Methode `targetNodeReached()` wird dabei immer aufgerufen, wenn ein Monster den Mittelpunkt einer Zelle erreicht hat und eine neue Bewegungsrichtung eingestellt werden kann. Die Zellen des Spielfeldes werden hierbei als Knoten (`class MapNode`) repräsentiert, die Referenzen auf eine Menge an Nachbarknoten enthalten, die durch gültige Spielzüge erreicht werden können (ähnlich zu Listen, die in der Vorlesung vorgestellt wurden). Bei Aufruf der Methode ist der aktuelle Knoten, auf dem das Monster sitzt, durch `targetNode` referenziert. In der vorliegenden Version wird in jedem Schritt ein zufälliger Spielzug ausgewählt, den die Monster ausführen. Dazu wird mit der Methode `target_node.neighbours.size()` die Anzahl der möglichen gültigen Folgeknoten bestimmt und zufällig einer dieser Knoten ausgewählt. Die Aufgabe dieses Zusatzblattes ist es, einen verbesserten Algorithmus für die Monster zu implementieren.

1. Verändern Sie die Methode `targetNodeReached` so, dass sich die Monster immer in die aktuelle Bewegungsrichtung weiter bewegen. Ist dies aufgrund einer Wand nicht mehr möglich, soll die neue Bewegungsrichtung zufällig ausgewählt werden. Speichern Sie sich dafür die aktuelle Richtung und suchen Sie nach einem Nachbarknoten, der in der gleichen Richtung liegt. Die Position der Knoten können Sie mit der Methode `node.getPosition()` abrufen.

2. Schreiben Sie einen Algorithmus, der die Monster immer auf den Nachbarknoten bewegt, der den Euklidischen Abstand zu PacMan minimiert. Die Position des PacMan können Sie ebenfalls durch die Methode `pac.getPosition()` abrufen.

3. (Fortgeschrittene Programmieraufgabe) Schreiben Sie einen Algorithmus, der immer den kürzesten Weg durch das Labyrinth zu PacMan sucht. Implementieren Sie dafür eine Breitensuche über alle Knoten. Gehen Sie dafür wie folgt vor:

   (a) Schreiben Sie eine Klasse `SearchNode`, die als Instanzvariablen einen Referenz `node` auf einen `MapNode` sowie eine Referenz `parent` auf einen `SearchNode` enthält. Diese Klasse wird dazu verwendet, eine einfach verkettete Liste von Knoten zu speichern, die den kürzesten Weg von Pacman zu der aktuellen Monster-Position repräsentiert.

   (b) Verwenden Sie die Datenstruktur `Queue` (siehe Übungsblatt 13) für eine Breitensuche über alle Knoten. Erstellen Sie eine Queue `open_list`, die alle Knoten enthält, die noch durchsucht werden müssen, sowie eine ArrayList `closed_list`, die alle Knoten enthält, die schon durchsucht wurden.

   (c) Speichern Sie dafür zunächst den aktuellen Knoten, auf dem das Monster sitzt, in einen `SearchNode` mit parent `null`. Fügen Sie diesen Knoten in die `open_list` ein.

   (d) Führen Sie folgende Schritte so lange aus, wie sich Knoten in `open_list` befinden: Entnehmen Sie den ersten Knoten der Queue `open_list` mittels `pop()` und speichern Sie ihn als `current_node`. Fügen Sie den Knoten `current_node.node` in die ArrayList `closed_list` ein. Überprüfen Sie nun, ob PacMan auf dem aktuellen Knoten sitzt, verwenden Sie dafür die Methode `pac.isInside(current_node.node)`. Wurde PacMan gefunden, speichern Sie sich den `current_node` und beenden Sie die Suche. Ansonsten iterieren Sie über alle Nachbarn `neighbor_node` und fügen Sie das Element `SearchNode(neighbor_node, current_node)` in die Queue `open_list` ein, wenn `neighbor_node` nicht in der Queue der schon bearbeiteten Knoten `closed_list` zu finden ist.

   (e) Nach Beenden der Suche referenziert der `SearchNode current_node` den Kopf einer Liste, die den kürzesten Weg von PacMan zur Position des Monsters beschreibt. Der zweite Eintrag dieser Liste ist somit der Knoten, zu dem sich das Monster im folgenden Schritt bewegen sollte.