Aufgabenbeschreibung
Softwareentwicklungspraktikum

Wintersemester 2008/2009
Universität München
Martin Lange und Ulrich Schöpp

Teil I – Spielverlauf
Version 3

1 Einleitung

In diesem Praktikum wollen wir die Ameisenart *formica artificiosa* entwickeln.

Ameisen dieser Art leben auf einem zweidimensionalen Feld mit sechseckigen Zellen. Sie verhalten sich völlig autonom, wissen was ihre Aufgabe in der Ameisengesellschaft ist und gehen dieser fleißig nach. Wie üblich sind die Ameisen in verschiedenen Ameisenstämmen organisiert, so dass sie bei der Futtersuche nicht auf sich allein gestellt sind.

Damit sich der Zusammenschluss zu einem Ameisenstamm für die Ameisen auch lohnt, muss dieser natürlich vernünftig organisiert sein. Ein schlecht organisierter Ameisenstamm wird nicht viel Futter abbekommen, da ihm dieses von den besser organisierten Stämmen längst weggeschnappt wurde. Bei dieser Aufgabe wollen wir den Ameisen in diesem Praktikum helfen, indem wir einzelnen Ameise sagen, wie sie sich verhalten sollen, damit ihr Ameisenstamm dann erfolgreich ist.

Die erste Aufgabe im Praktikum ist dann also, die Welt der formica artificiosa zu implementieren. Mit so einer Implementierung können wir dann leicht herausfinden, wie gut sich ein Ameisenstamm gegen einen anderen behaupten kann. Wir setzen dazu einfach mehrere Ameisenstämme auf ein Feld und überlassen es dann den Ameisen, das Futter zu finden und nach Hause zu bringen. Der Ameisenstamm, der nach einer bestimmten Zeit das meiste Futter gesammelt hat, wird dann zum Sieger erklärt.

Die zweite Aufgabe besteht dann darin, sich gut organisierte Ameisenstämme auszudenken. Einen Ameisenstamm anzugeben bedeutet einfach, dass man die Ameisen auf ihren Ameisenhügel setzt und das Gehirn jeder Ameise entsprechend „programmiert“. Die Ameisen können somit verschiedene Aufgaben übernehmen, wie zum Beispiel die Futtersuche oder die Bauverteidigung. Die Ameisengehirne können sich nicht ändern und müssen daher von Anfang an so gewählt werden, dass das Verhalten der Ameise dem Ziel des Ameisenstamms zuträglich ist.

Um erfolgreiche Ameisenstämme herzustellen, wird es nötig sein, sich Werkzeuge zum Entwurf von Ameisen auszudenken. Zum Beispiel wird man die Ameisen sicher in verschiedenen Situationen ausprobieren wollen, bevor man sie dann ins Rennen schickt.

2 Aufgabe

Die Praktikumsaufgabe ist, eine Anwendung zu schreiben, mit der Ameisenstämme gegeneinander antreten können und die es erlaubt, Ameisenstämme zu entwickeln. Dieses Anwendung soll durch netzwerkfähige Client/Server-Programme realisiert werden.

Ein Server hat die Aufgabe, Ameisenstämme von mehreren Mitspielern anzunehmen und diese gegeneinander antreten zu lassen. Die Mitspieler sollen dem Server ihre Ameisenstämme übermitteln. Diese werden vom Server gespeichert. Die Mitspieler können dann vom Server erfahren, wie sich ihr Ameisenstamm gegen die anderen eingereichten Stämme schlägt. Dazu berechnet der Server, was die Ameisen machen, wenn man verschiedene Ameisenstämme auf ein Feld setzt, und welcher Ameisenstamm nach einer bestimmten Anzahl von Schritten die meisten Futterstücke zurück zu seinem Hügel gebracht hat. Die Mitspieler können sich diesen Ablauf vom Server geben lassen und so das Verhalten der Ameisen nachverfolgen.

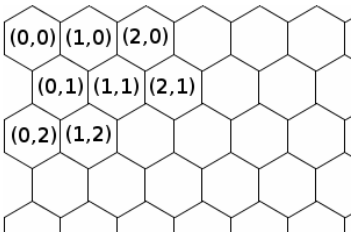
Alle Serverprogramme sollen ein gemeinsames Protokoll unterstützen (siehe Abschnitt 5), so dass alle Gruppen auf einem einzigen Server ihre Ameisenstämme gegeneinander antreten lassen können.

Neben dem Serverprogramm muss jede Gruppe natürlich auch noch eine Client-Anwendung schreiben. Mit dieser soll man Ameisenstämme entwerfen und dann beim Server einreichen können. Sie soll außerdem die Spielverläufe anzeigen können, die vom Server ausgerechnet werden und von ihm bereitgestellt werden.

Wie die Ameisenstämme entworfen werden, ist den Gruppen überlassen. Es bietet sich an, dazu ein Programm zu schreiben (evtl. als Teil des Clients), mit dem man Ameisen entwickeln und in verschiedenen Situationen ausprobieren kann.

3 Spielfeld

Das Spielfeld besteht aus einem Gitter hexagonaler Zellen in folgender Anordnung:



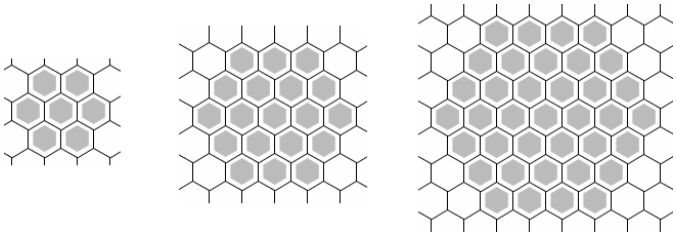
Es gibt verschiedene Spielfeldgrößen. Eine typische Größe ist 100 mal 100 Zellen. Man kann davon ausgehen, dass kein Spielfeld breiter oder höher als 1024 Zellen ist.

Jede Zelle in dem hexagonalen Gitter kann folgende Eigenschaften haben:

- Sie kann ein Hindernis enthalten (vielleicht einen Stein oder Wasser, aber das ist nicht näher spezifiziert).
- Enthält sie kein Hindernis, so können alle folgenden Fälle eintreten.
  - Sie kann *eine* Ameise enthalten.
  - Sie kann eine beliebige Anzahl von Futterstücken enthalten.
  - Sie kann ein Teil des Ameisenhügels eines bestimmten Ameisenstammes sein.
  - Sie kann Duftspuren enthalten. Jeder Ameisenstamm hat sechs verschiedene Duftmarker, von dem jeder entweder in der Zelle vorhanden ist oder nicht. Für jeden Ameisenstamm sind die sechs verschiedenen Typen von Duftmarken von 0 bis 5 durchnummeriert.

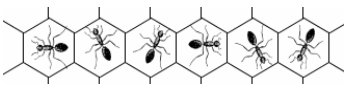
Jede Zelle am Rand des Spielfeldes enthält ein Hindernis. Eine Zelle liegt am Spielfeldrand, wenn sie weniger als sechs Nachbarn auf dem Spielfeld hat.

Man kann annehmen, dass Ameisenhügel „rund“ sind. Das heißt, dass die Ameisenhügel aus allen Zellen bestehen, die sich von der mittleren Zelle in eine bestimmten Höchstanzahl von Schritten erreichen lassen. Unten abgebildet sind Ameisenhügel vom Radius 1, 2 und 3. Ein typischer Ameisenhügel besteht aus 7 bis 50 Zellen.



Eine Ameise in einer Zelle hat folgende Eigenschaften:

- Jede Ameise gehört genau einem Ameisenstamm an.
- Jede Ameise ist in die Richtung einer der sechs Nachbarzellen ausgerichtet. Die Ausrichtung ist durch eine Zahl von 0 bis 5 gegeben. Das folgende Bild zeigt von links nach rechts die Bedeutung der Ausrichtungen 0, 1, 2, 3, 4 und 5.



- Eine Ameise kann ein Futterstück tragen.
- Jede Ameise hat ein Gehirn, das sich in einem bestimmten Zustand befindet. Wie die Gehirne genau funktionieren, wird in einem anderen Teil der Aufgabenbeschreibung spezifiziert. Dort wird dann die Programmiersprache AntBrain definiert, mit der man die Gehirne der Ameisen angeben kann.

Um das Verhalten der Ameisen zu beschreiben, ist es aber gar nicht nötig, die genaue Definition der Sprache AntBrain zu kennen. Die Ameisen verhalten sich folgendermaßen. In jedem Schritt kann die Ameise zwei Zellen sehen: die Zelle, auf der sie gerade steht, und die Nachbarzelle, in deren Richtung die Ameise ausgerichtet ist. In Abhängigkeit vom Aussehen dieser beider Zellen und ihrem Gehirnzustand macht die Ameise dann eine Aktion und nimmt einen neuen Gehirnzustand an. Dieses Verhalten werden wir dann durch eine Funktion

$$\mathit{step}\colon \mathit{Brain} \times (\mathit{Cell} \times \mathit{Cell}) \longrightarrow \mathit{Action} \times \mathit{Brain}$$

genau angeben. Dabei steht *Brain* für die Menge der möglichen Gehirnzustände der Ameise, *Cell* für die Menge aller Zellen und *Action* für die Menge aller möglichen Aktionen der Ameise.

Die Funktion *step* nimmt den momentanen Gehirnzustand und die zwei Zellen, die die Ameise sehen kann, als Eingabe und gibt eine Aktion sowie den neuen Gehirnzustand nach der Aktion aus.

Konkret ist die Menge der möglichen Aktionen jeder Ameise dabei gegeben durch:

$$\begin{aligned}\mathit{Action} = {} & \{\mathbf{skip}, \mathbf{walk}, \mathbf{pickup}, \mathbf{drop}\} \\ & \cup (\{\mathbf{turn}(i), \mathbf{leavescent}(i), \mathbf{clearscent}(i) \mid 0 \leq i \leq 5\})\end{aligned}$$

Was diese Aktionen zu bedeuten haben, wird im nächsten Abschnitt über den Spielablauf beschrieben.

4 Spielverlauf

4.1 Initialisierung

Die Teilnehmer reichen Ameisenstämme beim Server ein. So ein Ameisenstamm besteht einfach aus einer Anzahl von Ameisen, die dann später auf einem Ameisenhügel platziert werden. Für jede Ameise auf dem Ameisenhügel wird dabei ihre Ausrichtung gegeben, sowie ein Programm in der Sprache AntBrain, welches das Gehirn der Ameise definiert.

Das Spiel beginnt nun, indem die Ameisenstämme auf die Ameisenhügel des Spielfeldes gesetzt werden und ihre Gehirne mit den gegebenen AntBrain-Programmen in den Anfangszustand versetzt werden. Danach werden bis zu 1.000.000 Spielrunden ausgeführt (die genaue Zahl hängt vom Spielfeld ab) und der Ameisenstamm, auf dessen Hügel danach die meisten Futterstücke liegen, hat gewonnen.

4.2 Eine Runde

Eine Spielrunde besteht nun daraus, dass jede Ameise genau einen Schritt macht. Diese Schritte erfolgen nacheinander, d.h. zuerst macht eine Ameise einen Schritt, dann eine

andere und so weiter, bis alle genau einmal dran waren. Die Reihenfolge, in der die Ameisen in einer Spielrunde zum Zug kommen, ist dabei nicht vorgegeben. Sie darf von der Serverimplementierung abhängen.

4.3 Bewegung einer Ameise

Ein Schritt einer Ameise ist dabei folgendermaßen definiert. Sei **here** die Zelle, auf der die Ameise steht, und sei **ahead** die Nachbarzelle, in deren Richtung die Ameise schaut. Sei b der momentane Gehirnzustand der Ameise. Wir wenden dann die Funktion *step* auf die Argumente b, **here** und **ahead** an und erhalten ein Paar $(a, b') = \mathit{step}(b, \textbf{here}, \textbf{ahead})$ bestehend aus einem neuen Gehirnzustand b' und einer Aktion a. Die Ameise nimmt dann den neuen Gehirnzustand an und verhält sich folgendermaßen:

- $a = \textbf{skip}$. Die Ameise macht nichts.
- $a = \textbf{walk}$. Die Ameise geht auf die Zelle **ahead**, wenn diese nicht ein Hindernis ist oder dort bereits eine andere Ameise steht. Ansonsten bleibt sie stehen.
- $a = \textbf{pickup}$. Liegt in **here** ein Futterstück und trägt die Ameise nicht bereits eines, so hebt sie ein Futterstück auf. Ansonsten macht sie, wie bei **skip**, nichts.
- $a = \textbf{drop}$. Trägt die Ameise ein Futterstück, so lässt sie es in der Zelle **here** fallen. Ansonsten macht sie nichts.
- $a = \textbf{turn}(i)$. Die Ameise dreht sich um i Schritte nach rechts. Wenn also ihre Ausrichtung vorher d war, so ist sie nach diesem Schritt $(d + i) \bmod 6$.
- $a = \textbf{leavescent}(i)$. Die Ameise hinterlässt die Duftmarke vom Typ i und bewegt sich nicht.
- $a = \textbf{clearscent}(i)$. Die Ameise entfernt die Duftmarke vom Typ i und bewegt sich nicht. Ist die Duftmarke nicht vorhanden, so macht die Ameise nichts.

Ist nach einem solchen Schritt irgendeine Ameise von mindestens fünf fremden Ameisen umgeben, so stirbt sie und verschwindet. Stirbt eine Ameise, die Futter trägt, so verschwindet zwar die tote Ameise, das Futter bleibt jedoch liegen.

5 Interfaces

5.1 Das Spielbrett in der Übertragung vom Server zum Client

Spielbretter müssen offensichtlich zwischen einem Spielserver und einem Client übertragen werden können. Dafür geben wir Interfaces vor, die auf der Homepage des Praktikums zu finden sind. Zu Details bzgl. der einzelnen Methoden in diesen Interfaces und ihrer intendierten Bedeutung verweisen wir auf die jeweiligen JavaDoc-Dokumentationen.

Bei den Interfaces, die Übertragungen von Objekten zwischen Server und Clients standardisieren, muss eine Implementierung natürlich evtl. auch noch `Remote` oder `Serializable` implementieren. Es bleibt jeweils den Gruppen überlassen, abzuwägen, was dabei sinnvoll ist.

5.1.1 Aufstellung des Ameisenstamms

Um einen Ameisenstamm auf einem bestimmten Spielbrett aufzustellen, lässt sich der Client vom Server ein Objekt vom Typ `Arena` geben. Dieses Objekt stellt Funktionen bereit, mit denen mehrere Mitspieler ihre Ameisenstämme auf einem bestimmten Spielfeld aufstellen können und diese gegeneinander antreten lassen können.

Mit dem Interface `Arena` kann ein Mitspieler nun zunächst erfragen, wie groß die Ameisenhaufen auf dem Spielfeld sind. Mit diesem Wissen kann der dann die Ameisen präparieren, die er einreichen will.

Das Aufstellen der Ameisen erfolgt dann durch Aufruf der Methode `submitTribe` auf einem Objekt vom Typ `Arena`. Diese Methode erwartet u.a. ein Array von Objekten vom Interface-Typ `AntClientToServer`. Solche Objekte erlauben es dem Server, Informationen über eine spezifische Ameise zu erfragen, insbesondere einen String, der den Anfangszustand des Gehirns der Ameise darstellt, und ihre anfängliche Ausrichtung, aber auch einen Namen, der möglichst eindeutig sein sollte.

Die Reihenfolge beim Aufstellen ist wie folgt. Die erste Ameise (also die an Index 0 in dem vom Client übergebenen `AntClientToServer[]`-Objekt steht in der Mitte des Ameisenhügels. Danach werden die Ameisen an den Indizes 1,...,6 auf den innersten Ring um den Mittelpunkt gestellt. Ameise 1 steht dabei links neben dem Mittelpunkt, die restlichen werden in ihrer natürlichen Reihenfolge im Uhrzeigersinn aufgestellt. Danach wird mit den Ameisen 7,...,18 in derselben Weise der nächstinnere Ring gefüllt, etc.

Hinweis: Die Gehirne von Ameisen werden durch Programme der Sprache AntBrain beschrieben. Diese Sprache wird in einem anderen Teil der Aufgabenbeschreibung spezifiziert. Aus diesem Grund kann die Methode `getBrain` momentan noch so implementiert werden, dass sie eine beliebige Zeichenkette zurückliefert, welche dann vom Server, der AntBrain-Programme ausführt, ignoriert wird.

Die Rückgabe der Ergebnisse vom Server zum Client wird so gehandhabt, dass der Client beim Server einen `ResultNotifier` hinterlässt. Ein Objekt eines solchen Typs wird dann vom Server verwendet, um den Client darüber zu informieren, ob seine Anfrage – z.B. nach einer berechneten Partie – erfolgreich war oder nicht. Dieses Interface ist generisch. Dadurch kann es vom Server benutzt werden, um auf eine Anfrage nach einer Partie gleich ein Objekt vom Typ `Match` zu übergeben, bzw. den Client über den Erfolg oder Misserfolg beim Aufstellen eines Stammes zu berichtigen, wo kein Objekt vom Typ `Match` benötigt wird bzw. vorhanden ist.

5.1.2 Übertragung einer ausgerechneten Partie

Die Übertragung einer vollständig berechneten Partie vom Server zum Client erfolgt mithilfe des Interfaces `BoardServerToClient`. Dies ist mit einer Klasse zu implementieren, welche es dem Client erlaubt, Abfragen zum Spielbrett, insbesondere zur Position von Hindernissen, Ameisen, Futter, etc. zu machen, um das Spielbrett dann grafisch anzuzeigen.

Die Implementierung von `BoardServerToClient` muss es natürlich auch dem Server gestatten, die Positionen der Hindernisse, Ameisen, etc. entsprechend anzugeben. Man beachte, dass diese Implementierung nicht identisch sein muss mit der internen Darstellung eines Spielbretts im Server, welches dieser zur Berechnung der Spiele verwendet. Insbesondere kann es sinnvoll sein, bei der Berechnung Datenstrukturen zu verwenden, die einen schnellen Zugriff erlauben, während – je nach dem – es bei der Übertragung des Spielbretts an einen Client sinnvoll sein kann, möglichst wenig Speicherplatz mit solchen Objekten zu verbrauchen.

5.2 Partien

Eine Partie wird über das Interface `Match` modelliert. Ein Objekt eines solchen Typs enthält die Startaufstellung als `BoardServerToClient` sowie Informationen wie die Anzahl der Runden in dieser Partie. Es erlaubt dem Client auch, anzufragen, welche Aktionen in welchen Runden durchgeführt wurden.

Die verschiedenen Typen der am Ende von Abschnitt 4 beschriebenen Aktionen werden hier im Aufzählungstyp `ActionKind` modelliert. Die Klasse `Action` verbindet dann solch eine Aktion mit einer Position auf dem Spielbrett.