2. Programmieraufgabe

Programmierparadigmen

LVA-Nr. 194.023
2023/2024 W
TU Wien

Themen:

Aufwandsabschätzung, Programmiereffizenz, Überblick über prozedurale und objektorientierte Programmierung

Ausgabe:

16. 10. 2023

Abgabe (Deadline):

30. 10. 2023, 14:00 Uhr

Abgabeverzeichnis:

`Aufgabe1-3`

Programmaufruf:

```
java Test
```

Grundlage:

Skriptum,
Abschnitt 2.1 und 2.2

das sind Vorschläge, nicht verpflichtend

Kontext

Die Simulation aus der ersten Programmieraufgabe ist viel zu oberflächlich und lässt zahlreiche der in der Realität entscheidenden Faktoren unberücksichtigt. Unser Ziel ist die Einbeziehung möglichst aller relevanten Faktoren. Damit könnten wir das Programm als vielseitiges Werkzeug einsetzen, um Ameisenbewegungen genauer zu studieren, Einflussfaktoren auf Ameisenalgorithmen zu analysieren sowie allgemein Optimierungsaufgaben lösen, für die sich Ameisenalgorithmen eignen.

Welche Aufgabe zu lösen ist

Simulation Die Lösung der ersten Programmieraufgabe ist deutlich zu verbessern. Vor allem soll die Simulationsgenauigkeit durch Berücksichtigung zahlreicher weiterer Einflussgrößen steigen. Hier sind Hinweise darauf, was berücksicht werden könnte:

- Duftspuren sind entscheidend, aber unzuverlässig. Duftspuren werden durch Regengüsse ausgedünnt oder Trägermaterialien durch Wind, Wasser und Tiere verfrachtet. Umwelteinflüsse können ständig kleinräumig auftreten, aber gelegentlich auch in großem Umfang die ganze sichtbare Umgebung betreffen. Solche Störfaktoren sollen in die Simulation einfließen. Es stellt sich die Frage, ob Störfaktoren nur hinderlich sind oder dazu beitragen, Optimierungsergebnisse langfristig zu verbessern, weil lokal aufgetretene suboptimale Wege immer wieder verworfen und neu aufgebaut werden.
- Wir sind von einem endlichen, grenzenlosen Raum ausgegangen, in dem Ameisen sich nie weit vom Bau entfernen können. Die echte Welt ist für eine Ameise viel zu groß, um sie vollständig zu erkunden. Am Abend soll jede Ameise wieder in den Bau zurückgefunden haben. Sie darf sich also nicht zu weit entfernen. Sie muss irgendwie abschätzen können, wann es Zeit ist, die Erkundung zu beenden und zurückzukehren. Um das zu simulieren, muss der Raum größer sein als die Fläche, die von den Ameisen erkundet werden kann. Das ist ein sehr großer Raum, der in außen liegenden Bereichen kaum verwendet wird. Ein Array oder eine ähnliche Datenstruktur wird aus Effizienzgründen kaum dafür geeignet sein. Besser eignen sich Datenstrukturen, die eine dünne Besiedlung effizient darstellen können. Die Bildschirmausgabe muss nicht den gesamten Raum umfassen, sondern kann sich auf einen „interessanten“ Teil beschränken.
- Der Raum, in dem eine Ameise lebt, ist nicht zweidimensional, sondern meist auf komplizierte Weise dreidimensional geformt. Ein optimaler Weg ist nicht unbedingt eine Gerade zwischen zwei Punkten. Körperliche Fähigkeiten der Ameise entscheiden, ob ein Hindernis überlaufen oder lieber umlaufen wird. In der Simulation ist es nötig, dreidimensionale Strukturen darzustellen und ein Modell dafür

zu entwickeln, welche Strukturen von den Ameisen als Teil des von ihnen besiedelten Raums überlaufen und welche umgangen werden.

- Um die Qualität von Ameisenalgorithmen überprüfen zu können, wäre es notwenig, die optimale (kürzeste) Verbindung zwischen zwei Punkten zu kennen. Nur so ist abschätzbar, wie rasch sich gefundene Wege an das Optimum annähern. In komplexen dreidimensionalen Räumen kann die Berechnung optimaler Wege aufwändig sein.

- Es hat sich gezeigt, dass Ameisen in einem bestimmten Ausmaß lernfähig sind und nicht nur einen fix vorgegebenen Algorithmus abspulen. Ältere, erfahrenere Ameisen sind bei der Futtersuche erfolgreicher als junge. Vielleicht lernen Ameisen mit der Zeit Duftspuren besser zu lesen. Vielleicht orientieren sie sich aber auch an anderen Merkmalen des Raums und finden vertraute Wege ohne Duftspuren. Jedenfalls müsste untersucht werden, wie sich der Aufbau von Ameisenstraßen ändert, wenn manche Ameisen viel begangene Wege ohne Duftspuren auf gewohnte Weise gehen.

- Eine ganz offensichtliche Abweichung der Simulation von der Natur ist die Annahme, dass Futterquellen unendliche Mengen an Futter bereithalten. Tatsächlich sind die meisten Futterquellen bald erschöpft. Eine große Stärke der Ameisen besteht darin, neue Futterquellen in ihrem Einzugsbereich sehr zuverlässig zu finden. Das Erschöpfen und Entstehen neuer Futterquellen an anderen Stellen sollte unbedingt simuliert werden.

- Eine weitere Abweichung von der Natur ist die Annahme, dass nur ein Ameisenstaat existiert. Tatsächlich existieren viele. Gelegentlich stoßen Ameisen unterschiedlicher Staaten aufeinander, jedenfalls konkurrieren sie um Futter. Wenn man vorerst davon absieht, dass dadurch Revierkämpfe ausbrechen können, stellt sich zumindest die Frage, wie sich Duftspuren von Ameisen eines Staats auf das Verhalten von Ameisen eines anderen Staats auswirken. Wenn Ameisen die Duftspuren befreundeter Ameisen von denen fremder Ameisen unterscheiden können, sollte sich ein anderes Verhalten ergeben, als wenn das nicht der Fall ist.

- Falls Ameisen Duftspuren unterschiedlicher Staaten auseinanderhalten können, dann vielleicht auch eigene Duftspuren von denen anderer Individuen. Das könnte einer Ameise ermöglichen, den eigenen Weg zurückzuverfolgen, ohne von Duftspuren anderer Ameisen abgelenkt zu werden. In den Bewegungsmustern sollte eine Unterscheidung von Duftspuren erkennbar sein.

- R. Feynman, ein wichtiger Mitbegründer moderner Quantenfeldtheorien, hat aufbauend auf der Entstehung von Ameisenstraßen eine Erklärung dafür gefunden, wieso in der Physik Bewegungen von Teilchen sicher vorhersagbar sind (Impulserhalt), obwohl einige Interpretationen der Quantenmechanik chaotisches Verhalten nahelegen: Alle den Teilchen entsprechenden Wellen überlagern sich,

wodurch sich Wellen am optimalen Pfad gegenseitig verstärken, andere, daneben liegende jedoch auslöschen. Das ähnelt der Verstärkung von Duftspuren der Ameisen am optimalen Weg, während sie sich auf anderen Wegen langsam abschwächen. Die Überlagerung von Wellen ist vermutlich effektiver als sich nur langsam abschwächende Duftspuren. Wenn es um Ameisenalgorithmen geht, wäre es vielleicht effizienter, statt Duftspuren Wellen zu verwenden, die sich bei verschobener Überlagerung gegenseitig auslöschen, damit sich optimale Wege rascher herauskristallisieren.

- Viele Ameisen bewegen sich bei der Erkundung der Umwelt auf Zickzack-Bahnen, sie „mäandern“: Statt geradeaus zu laufen, bewegen sie sich ein Stück nach links vorne, halten inne, dann nach rechts vorne, halten inne, dann wieder nach links vorne und so weiter. Dieses Verhalten führt zu Spekulationen über das Warum. Vielleicht verwenden sie das Innehalten, um Duftspuren und Gerüche aufzunehmen – wahrscheinlich nicht nur solche direkt bei ihnen, sondern auch die der Umgebung. Möglicherweise halten sie inne, um Duftspuren zu setzen, also nicht kontinuierlich beim Laufen, sondern punktweise in Abständen voneinander. Die Richtungswechsel könnten dazu dienen, Stärken von hintereinander wahrgenommenen Gerüchen zu vergleichen und damit auf eine Richtung zu schließen, aus denen sie kommen. Die im Zickzack-Muster abgelegten Duftmarken könnten Ameisen helfen, auf Linie zu bleiben, wenn sie sich zwischen den Marken hindurchbewegen. Es besteht auch die Möglichkeit, dass sich mehrere derartige Zickzack-Linien überlagern und Ameisen ihre Bewegungen danach ausrichten, so als ob sie einer Überlagerung von sich gegenseitig verstärkenden und auslöschenden Wellen folgen würden. Das sollte genauer untersucht werden.

- Das Heranschaffen von Futter ist nur eine der Aufgaben von Ameisen und ist untrennbar mit anderen Aufgaben verbunden, die ebenfalls simuliert werden sollen: Gründung eines Staats, Anlegen und Instandsetzen des Baus, Fortpflanzung, Nachwuchspflege, Verteidigung gegen Angreifer, Überwinterung und so weiter.

- Ameisenalgorithmen werden für Optimierungszwecke eingesetzt, die allgemeiner sind als konkrete Aufgaben der Ameisen. Beispielsweise sollen simulierte Ameisen optimierte Wege nicht nur zwischen Futterquellen und einem Bau finden, sondern zwischen beliebig vielen Punkten, vielleicht in bestimmter Reihenfolge. Um Simulationen zu erleichtern, wird eine Sprache benötigt, in der sich Details des Raums einfach darstellen lassen: zu besuchende Punkte, Änderungen der Positionen dieser Punkte über die Zeit, topologische Eigenheiten wie Hindernisse und deren Änderung über die Zeit sowie alle möglichen Details des zu verwendenden Algorithmuses.

- Je genauer wir unsere Simulation machen, desto länger dauern Simulationsläufe. Aus praktischer Sicht ist es unabdingbar, dass wir auf die Effizienz der Ausführung achten. Eine ganze Reihe von Optimierungsmaßnahmen ist denkbar, etwa das Wiederverwenden von Teilergebnissen anstatt häufig wiederholter gleicher Berechnungen.

- Bei vielen einstellbaren Parametern und Variationen ergeben sich sehr viele mögliche Simulationsläufe. Zu deren Analyse benötigen wir eine Datenbank mit Simulationsergebnissen. Unterschiedliche Simulationen weisen oft nur wenige Gemeinsamkeiten auf. Wir benötigen eine systematische Darstellung und grobe Kennzahlen, die Simulationsergebnisse trotzdem vergleichbar machen. Bitte binden Sie keine fertigen Bibliotheken oder Datenbanken ein, sondern entwickeln Sie eine Lösung speziell für die Aufgabe.

Bestimmen Sie selbst den Funktionsumfang Ihres Programms. Das Programm soll möglichst viel der nötigen (vorgeschlagenen oder selbst gefundenen) Funktionalität abdecken und über Testfälle überprüfen.

Funktionsumfang selbst wählen

Testen Das Programm soll wie Aufgabe 1 mittels `java Test` vom Verzeichnis `Aufgabe1-3` aus aufrufbar sein und die selbst gewählte Funktionalität überprüfen. Tests sollen ohne Benutzerinteraktion ablaufen, sodass Aufrufer keine Testfälle auswählen oder Testdaten eintippen müssen.

im richtigen Verzeichnis testen

Paradigmen und Kommentare Neben nominaler Abstraktion dürfen Sie auch Lambda-Abstraktion einsetzen. Jedoch soll jede nominale Abstraktion mit Kommentaren versehen sein, die die Abstraktion erläutern.

nominale Abstraktion nur mit Kommentaren

Setzen Sie Programmierstile des objektorientierten und des prozeduralen Paradigmas gemischt ein. Bitte streuen Sie in Ihr Programm an mindestens 5 relevanten Stellen je einen Kommentar ein, der mit „`STYLE:`“ beginnt und eine Erklärung enthält, welchem Paradigma der an dieser Stelle verwendete Programmierstil entspricht und woraus das ersichtlich ist. Abstraktionen müssen zum Paradigma passen. Dort, wo ein objektorientierter Stil verwendet wird, achten Sie bitte darauf, dass der betreffende Programmteil ausschließlich durch Abstraktionen verständlich ist sowie der Klassenzusammenhalt hoch und die Objektkopplung schwach ist. Sorgen Sie dafür, dass auch Untertypbeziehungen vorkommen. An Grenzen, an denen Stile unterschiedlicher Paradigmen aneinanderstoßen, machen Sie bitte klar, wie die Paradigmen zusammenwirken.

Paradigmen erläutern (mind. 5 Stellen)

Klassenzusammenh. hoch, Objektkopplung schwach, Untertypbeziehungen

Neben Programmtext soll die Datei `Test.java` als Kommentar die Grobstruktur und den (geplanten) Funktionsumfang der Lösung zusammenfassen sowie eine kurze, aber verständliche Beschreibung der Aufteilung der Arbeiten auf die einzelnen Gruppenmitglieder enthalten.

beschreiben: Struktur, Funktionsumfang, Aufgabenaufteilung

Wie die Aufgabe zu lösen ist

Die oben aufgezählten Vorschläge zur Verbesserung der Simulation sind als Anhaltspunkte gedacht. Sie können Punkte weglassen, abändern oder durch andere sinnvoll erscheinende Verbesserungen ersetzen.

Eine Schwierigkeit besteht in der richtigen Abschätzung des Umfangs der Arbeiten. Planen Sie nach Ihren Fähigkeiten möglichst viel ein, das Sie in der vorgesehenen Zeit zum Abschluss bringen können – das heißt, so viel Sie können, aber nicht mehr. Als groben Anhaltspunkt sollten Sie (jedes Gruppenmitglied) etwa elf mit intensiver Arbeit gefüllte Stunden investieren, keinesfalls mehr als fünfzehn. Versuchen Sie so effizient wie möglich zu arbeiten und rasch zu einer brauchbaren Lösung zu kommen. Ignorieren Sie Details, die Ihnen unwichtig erscheinen. Bedenken Sie, dass

Sie Ihre Lösung durch Testfälle überprüfen sollen und planen Sie Zeit für die Entwicklung der Testfälle und die Fehlerbeseitigung ein.

Konzept an Tutor_in

Erstellen Sie zuerst ein Konzept Ihrer geplanten Arbeiten. Schicken Sie es frühzeitig Ihrer Tutorin oder Ihrem Tutor. Sie werden so bald wie möglich erfahren, ob das Konzept ausreicht oder Änderungen nötig sind.

Paradigmen kombinieren

Eine Schwierigkeit ist der gleichzeitige Umgang mit mehreren Paradigmen. Sie müssen das prozedurale und objektorientierte Paradigma verwenden und den Übergang zwischen den Paradigmen in Kommentaren nachvollziehbar beschreiben. Lesen Sie Unterscheidungskriterien im Skriptum nach. Es wird dazu geraten, die Anzahl der Übergänge zwischen den Paradigmen klein zu halten. Im objektorientierten Teil ist auf hohen Klassenzusammenhalt, schwache Objektkopplung und die Verwendung von Untertypbeziehungen zu achten.

Für die ersten drei Aufgaben weist Ihr_e Tutor_in Sie einige Zeit nach der Abgabe bei Bedarf auf konkrete Fehler hin und bittet um Beseitigung. Dies wirkt sich nur dann auf Ihre Beurteilung aus, wenn Sie der Bitte nicht nachkommen. Sie sollten selbst (auch ohne Feedback) ein Gefühl für die Qualität Ihrer Lösungen entwickeln. Wegen der großen Zahl erwarteter Fehler können Tutor_innen nicht auf jede Kleinigkeit eingehen.

Warum die Aufgabe diese Form hat

Das Ziel dieser Aufgabe besteht *nicht* darin, wie bei anderen Aufgaben am Ende eine vollständige und perfekte Lösung abzuliefern. Unter Einhaltung der Bedingungen ist es unmöglich, eine perfekte Lösung zu produzieren. Vielmehr geht es darum, dass Sie Ihre eigenen Fähigkeiten und Grenzen beim Programmieren unter Zeitdruck als Einzelperson und Gruppe kennenlernen. Sie sollen selbst erkennen, wo Ihre Stärken liegen und zu welchen Arten von Fehlern Sie eher neigen (auf allen Ebenen von der Problemanalyse bis hin zum Testen ebenso wie hinsichtlich der Zusammenarbeit in der Gruppe). Im Hinblick darauf gibt es keine richtigen und falschen Lösungen. Es ist aber erkennbar, wie sehr Sie sich darum bemüht haben, eine unlösbare Aufgabe so gut wie möglich zu lösen.

Das Feedback durch die Tutor_innen wird genau darauf abzielen: Haben Sie sich ausreichend darum bemüht, in möglichst vielen Bereichen etwas Machbares zu machen, oder sind in zu vielen Bereichen keine Ansätze erkennbar? In letzterem Fall werden Tutor_innen Nachbesserungen verlangen, aber nicht, wenn aus Zeitdruck irgendwelche kleine Fehler passiert sind, mit denen bei einer solchen Aufgabe immer zu rechnen ist.

Sie sollen möglichst große Freiheit bei der Lösung der Aufgabe haben und selbst die Verantwortung für alles übernehmen. Niemand schreibt Ihnen vor, wie die Aufgabenstellung genau zu verstehen ist.

Diese Aufgabe stellt hohe Anforderungen an jedes Gruppenmitglied und die Zusammenarbeit in der Gruppe – eine Nagelprobe für das Funktionieren der Gruppe und zum Aufdecken möglicher Schwachstellen.

Nebenbei bietet es sich an, die Aufgabe zur Vertiefung eines überblicksartigen Verständnisses von Paradigmen einzusetzen. Der Schwerpunkt liegt auf der objektorientierten Programmierung (bei einem intuitiven Zugang) im Unterschied zur prozeduralen Programmierung.