MultiKara

Ein Leitprogramm zur nebenläufigen Programmierung

Schulstufe: Gymnasium
Dauer: 8 – 12 Lektionen
Autoren: Ch. Siegel / P. Müller
Fassung: April 2012

Inhalt

1 NEBENLÄUFIGKEIT (CONCURRENCY) ... 4

Übersicht ... *4*
Lernziele ... *4*
1.1 INTERAGIERENDE ODER NEBENLÄUFIGE PROZESSE ... 5
1.2 SYNCHRONISATION ... 6
1.2.1 Synchronisation durch Zufall ... 6
1.2.2 Gleichzeitigkeit: Wenn der Zufall überfordert ist ... 7
1.2.3 Ausblick ... 7

2 DAS PHILOSOPHENPROBLEM ... 8

Übersicht ... *8*
Lernziele ... *8*
2.1 DAS PROBLEM ... 9
2.1.1 Aufbau ... 9
2.1.2 Ablauf ... 9
2.1.3 Wo liegt das Problem? ... 9
2.1.4 Das Problem ... 10
2.1.5 Lösung(en) ... 10

3 DIE MULTIKARA-UMGEBUNG ... 11

Übersicht ... *11*
Lernziele ... *11*
3.1 ALTE BEKANNTE ... 12
3.2 WAS IST NEU BEI MULTIKARA ... 14
3.2.1 Die Elemente der Welt ... 14
3.2.2 Die Elemente des Automaten ... 14
3.2.3 Der Sensor-Baukasten ... 15
3.2.4 Das Zeit-Modell und der Scheduler ... 16

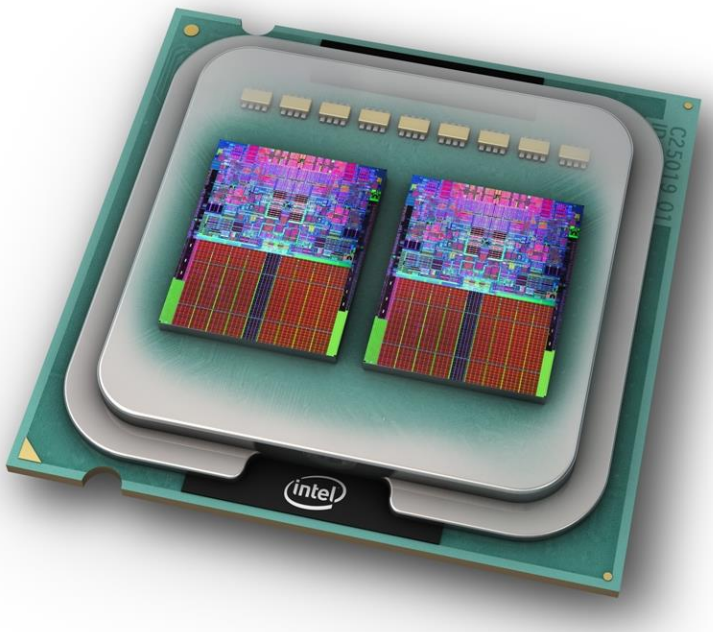
4 SYNCHRONISATIONSMECHANISMEN ... 22

Übersicht ... *22*
Lernziele ... *22*
4.1 ARBEITSTEILUNG OHNE SYNCHRONISATION ... 23
4.2 MOTIVATION VON SYNCHRONISATIONSMECHANISMEN ... 24
4.3 SYNCHRONISATIONSMECHANISMEN IM DETAIL ... 26
4.3.1 Synchronisation in der Welt: Monitor & Meeting Room ... 26
4.3.2 Synchronisation im Zustandsraum: Barrier & Critical Section ... 27

5 AUFGABEN ... 30

6 LÖSUNGEN ZU DEN AUFGABEN ... 34

1 Nebenläufigkeit (concurrency)



Übersicht

Dieses Kapitel gibt einen allgemeinen Überblick über das Konzept der Nebenläufigkeit und die Probleme, die bei nebenläufigen Prozessen auftreten können.
Vorab wird definiert, was Nebenläufigkeit ist. Als spezifische Probleme von nebenläufigen Prozessen wird das Problem von *deadlocks* erläutert.

Um diese Probleme zu beheben braucht es sogenannte Synchronisationsprozesse, welche von "aussen" oder durch geeignete Algorithmen von den Prozessen her für eine gewisse Steuerung sorgen und ein Verklemmen der Abläufe verhindern.

Lernziele

- Du verstehst was der Unterschied zwischen einem einfachen Programmablauf und nebenläufigen Prozessen ist und kannst Alltagsbeispiele nennen, die diesen Unterschied aufzeigen.
- Du kannst anhand von alltäglichen Nebenläufigkeiten erklären was ein deadlock ist.
- Du kannst zwei prinzipielle Synchronisationsmechanismen nennen und anhand von praktischen Beispielen erläutern.

1.1 Interagierende oder nebenläufige Prozesse

Im täglichen Leben sind wir uns gewohnt, dass unsere Tätigkeiten stets mit anderen zeitlichen Abläufen interagieren {aufeinanderbezogen handeln}. Wir warten darauf, dass die Verkehrsampel grün leuchtet; wir reihen uns in eine Warteschlange ein; mit einem Telefonanruf lenken wir einen anderen Menschen von seiner Arbeit ab und unterbrechen damit seinen Tagesablauf. Die Tatsache, dass wir unsere Pläne stets an unvorhergesehene, von andern verursachte Ereignisse anpassen müssen, ist so selbstverständlich, dass uns die Schwierigkeit dieser Aufgabe oft nicht bewusst wird.

Eigentlich ist der Mensch im Allgemeinen auch nicht im Stande zwei Dinge exakt gleichzeitig auszuführen, sondern er hat sich Regeln und Methoden angeeignet um scheinbar mehrere Dinge nebeneinander "am Laufen" zu haben. Tatsächlich arbeitet er aber mit Unterbrüchen mal am Einen und dann wieder am Anderen. Genauso arbeitet auch ein Prozessor eines Computers: Mal am Einen, dann wieder an einem anderen Prozess, "job", "task" oder "thread". Alle diese Prozesse auf einem Prozessor heissen in diesem Falle *nebenläufig.*

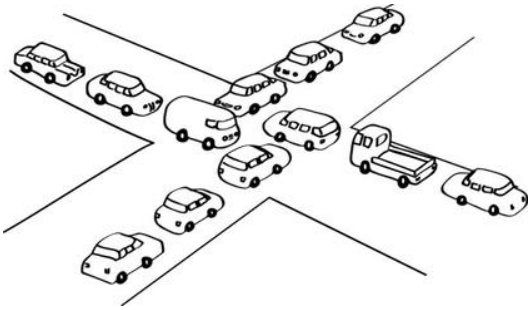
*Zwei Prozesse sind dann **nebenläufig** wenn sie unabhängig voneinander arbeiten können und es keine Rolle spielt, welcher der beiden Prozesse zuerst ausgeführt oder beendet wird.*

Derartige nebenläufige Prozesse sind jedoch ***indirekt voneinander abhängig***, da sie gemeinsame Ressourcen beanspruchen (interagieren). Dieser Sachverhalt kann zu Konflikten und Verklemmungen führen, die zu schwerwiegenden Fehlern bis hin zu Programmstillstand und dem Absturz führen können.

Bei einem System aus nebenläufigen Prozessen kann unter anderem die folgendes Problem auftreten:

Das System wird durch einen nicht mehr funktionierenden Prozess blockiert, so dass kein weiterer Prozess mehr einen Schritt tun kann; man spricht von ***deadlock***.

Beispiel: Vier Strassen kreuzen sich (Abb. rechts). Hält ein Auto auf einer Kreuzung an, kommt es zum Verkehrskollaps, da die Fahrzeuge der kreuzenden Strasse blockiert werden.



Der Vollständigkeit halber seien noch kurz weitere Probleme genannt, die bei nebenläufigen Prozessen auftreten können:

- ***Ungültige Prozessterminierung:*** Nachdem alle Prozesse (nebenläufig) in nicht vorhersehbarer Reihenfolge abgelaufen sind, kann es sein, dass einer oder mehrere Prozesse in einem ungültigen Endzustand sind.

  Beispiel: Am Ende des Startvorgangs eines PC's ist die Maus nicht aktiv. Der Maustreiber wurde zu früh geladen als der USB-Port-Treiber noch nicht fertig war. Der Maustreiber konnte der Maus keinen Port zuweisen.

- ***Verletzung von Fairness:*** Mehrere Prozesse könnten nebenläufig ausgeführt werden. Aus unerfindlichen Gründen wird ein Prozess – obwohl ausführbar – vom System (oder Prozessor) gar nie aufgerufen und macht deshalb auch keinen Fortschritt.

  Beispiel: Bei einer Strassenkreuzung hat eine Zufahrtsstrasse immer "rot", während die anderen Richtungen abwechslungsweise Rot- und Grünphasen haben.

1.2 Synchronisation

Was allen nebenläufigen Systemen (auch denjenigen aus unserem unmittelbaren Alltag) gemeinsam ist, ist das Problem, dass in einigen Fällen gewisse Dinge gleichzeitig zu erfolgen haben, in anderen Fällen muss Gleichzeitigkeit explizit ausgeschlossen werden.

Beispiel:
- Wenn vor laufender Kamera zwei Staatsoberhäupter sich die Hände schütteln (hand shake) sollen, bedingt dies, dass beide gleichzeitig anwesend sind – und anständigerweise der Eine nicht zu lange auf den Anderen warten muss.
- In der folgenden Ansprache vor eingeschalteten Mikrofonen ist es dann genau umgekehrt: Spricht die eine Person, schweigt die andere und umgekehrt.

Wie sich diese Synchronisation in nebenläufigen Systemen organisieren lässt, wird nachfolgend im Prinzip erklärt.

1.2.1 Synchronisation durch Zufall

Ein System führt nebenläufige Prozesse nicht nur abwechslungsweise aus, oft ist darüber hinaus überhaupt nicht voraussagbar, welcher Prozess zuerst startet, wann weitere Prozesse berücksichtigt werden und wann und wo die einzelnen Prozesse einen Schritt machen. Man spricht in diesem Fall von ***Nichtdeterminismus*** (die Ausführungsreihenfolge der nebenläufigen Prozesse ist zufällig). Oft ist es aber bei nebenläufigen Prozessen nötig, dass eine gewisse Ordnung im Ablauf und im Zugriff auf Ressourcen herrscht, die Prozesse also irgendwie synchronisiert werden. Wie kann man nun in einem solchen System Synchronisation erreichen?

Die ***exklusive*** Synchronisation (Verhinderung von Gleichzeitigkeit) ist insofern einfacher zu bewerkstelligen, als der Zufall dabei eine Hilfe sein kann.

Beispiel:
Sitzen zwei wortkarge Personen auf einer Bank, werden sie sich selten ins Wort fallen, da die Wahrscheinlichkeit, dass sie gleichzeitig das Wort ergreifen, relativ gering ist. So bald aber der Eine zu sprechen beginnt, muss der Andere schweigen, bis der Erstere fertig ist. Passiert es trotzdem einmal, dass beide gleichzeitig zu sprechen beginnen, schweigen sofort beide wieder und jeder beginnt nach einer zufälligen Wartezeit wieder zu sprechen.

Genau so funktioniert die Kommunikation zwischen mehreren Computern in einem Ethernet: Falls kein Computer eine Nachricht schickt, beginnt irgendein PC nach einer zufälligen Wartezeit zu senden. Beginnen zufälligerweise zwei gleichzeitig zu senden,

wird die Kollision erkannt und beide schweigen und wiederholen den Sendeversuch nach einer zufälligen Wartezeit noch einmal. Wer die kürzeste Wartezeit gewürfelt hat, beginnt dann als Erster wieder zu senden.
Zufällige Wartezeiten kombiniert mit einer Kollisionserkennung sind also eine Lösung für die Verhinderung von Gleichzeitigkeit – etwa in der Kommunikation.

1.2.2 Gleichzeitigkeit: Wenn der Zufall überfordert ist

Wenn **inklusive** Synchronisation – d.h. Gleichzeitigkeit – notwendig ist (Händeschütteln), dann ist der Zufall keine grosse Hilfe. Natürlich könnte man warten, bis zufällig zwei Prozesse am selben Ort sind. Aber genauso klein wie die Wahrscheinlichkeit, dass sich zwei Staatschefs *per Zufall* über den Weg laufen (ohne Abmachung oder äusseren Anlass) ist, ist die Wahrscheinlichkeit der zufälligen inklusiven Synchronisation.

Die Lösung von Aufgaben, die Gleichzeitigkeit verlangen, kann nicht völlig dezentral jedem einzelnen Prozess überlassen werden. Es braucht Steuerungsmechanismen, die von aussen auf die Teilnehmer einwirken. Solche Mechanismen kennen wir aus dem Alltagsleben. Damit die Bezeichnung "von aussen einwirken" richtig verstanden wird, betrachten wir eine Vereinbarung wie "Schulbeginn um 8 Uhr". Zwar richtet sich jeder täglich selbst danach ohne Interaktion mit anderen, aber die Ankündigung dieser Startzeit an alle, auch wenn nur durch einen Zettel am Anschlagbrett, ist eine Einwirkung von aussen, ein "broadcast".
Die zentrale Natur vieler Synchronisationsmechanismen erkennt man an Beispielen wie dem Dirigenten, der ein Orchester leitet oder dem Pfeifen des Schiedsrichters, der das Spiel unterbricht. Es gibt aber auch Synchronisationsmechanismen von lokaler Natur, wie die Übergabe des Stabes eines Stafettenläufers an seinen Nachfolger.

Bei der Betrachtung analoger Mechanismen, die programmgesteuerte Abläufe synchronisieren, drängt sich eine weitere Unterscheidung auf: Der Synchronisationsmechanismus kann im Programm oder in den Daten angesiedelt sein:

- Eine Programmvorschrift P1 kann lauten: "zuerst wird A ausgeführt und dann B, dann warte auf ein Signal von aussen". Hier ist der Einfluss von aussen offensichtlich.
- Ein anderes Programm P2 könnte die Anweisung enthalten: "Lösche den Eintrag X in der Datenbank D". Angenommen ein anderes Programm P3 führe genau in diesem Moment die Operation "lies X aus D heraus", so führt das zeitgleiche Löschen zu einer Inkonsistenz in der Datenbank und wird nicht toleriert werden.

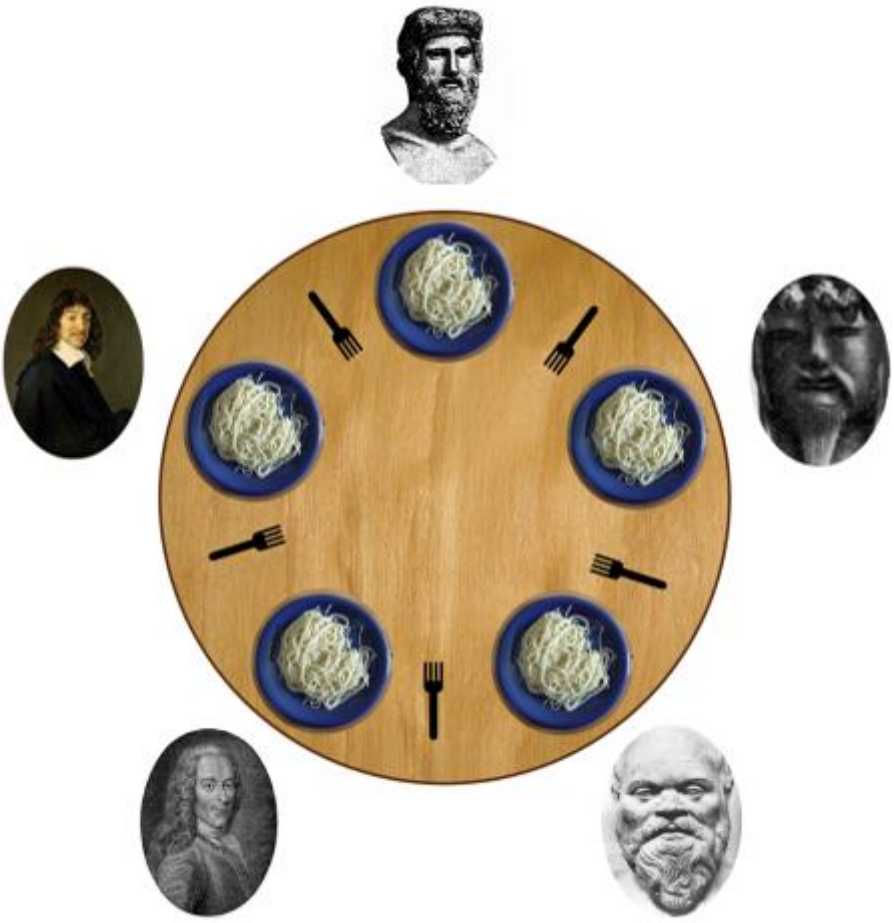
1.2.3 Ausblick

Wir werden im Rahmen dieses Leitprogramms mit MultiKara vier Synchronisationsmechanismen kennen lernen (vgl. Kap. 4). Davon sind je zwei inklusive und zwei exklusive Mechanismen und je zwei befinden sich in der Welt (Daten) und zwei im Zustandsraum, wie die folgende Übersichtstabelle zeigt:

	inklusive Synchronisation	*exklusive Synchronisation*
Welt	**Meeting Room**	**Monitor**
Zustandsraum	**Barrier**	**Critical Section**

Zuerst wenden wir uns aber einem klassischen Problem der Nebenläufigkeit zu.

2 Das Philosophenproblem



Übersicht

In diesem Kapitel wird das klassische Philosophen Problem (von Edsger Wybe Dijkstra) vorgestellt. Es veranschaulicht das Problem der Nebenläufigkeit und wie es zu Verklemmungen (deadlocks) kommen kann. Die Lösung dieses Problems ist deshalb wichtig, weil es die Konzepte liefert, die in der Kommunikation beim Zugriff auf gemeinsame Ressourcen (Datenbanken, Laufwerke, Netzwerke, verteilte Systeme,…) tatsächlich Verwendung finden.

Lernziele

- Du kannst das Philosophenproblem formulieren (Aufbau, Ablauf und Problematik).
- Du kannst mindestens eine Lösung im Ansatz skizzieren (formulieren), welche die Problematik behebt.

2.1 Das Problem

2.1.1 Aufbau

Es sitzen fünf Philosophen an einem runden Tisch, und jeder hat einen Teller mit Spaghetti vor sich. Zum Essen von Spaghetti benötigt jeder Philosoph zwei Gabeln. Allerdings waren im Haushalt nur fünf Gabeln vorhanden, die nun zwischen den Tellern liegen. Die Philosophen können also nicht alle gleichzeitig speisen.

2.1.2 Ablauf

Die Philosophen sitzen am Tisch und denken über philosophische Probleme nach. Wenn einer hungrig wird, greift er zuerst die Gabel links von seinem Teller, dann die auf der rechten Seite und beginnt zu essen. Wenn er satt ist, legt er die Gabeln wieder zurück und beginnt wieder zu denken. Sollte eine der Gabeln nicht an ihrem Platz liegen, wenn der Philosoph sie aufnehmen möchte, so wartet er, bis die Gabel verfügbar ist. Falls er eine Gabel schon in den Händen hält, legt er sie natürlich nicht ab.

2.1.3 Wo liegt das Problem?

Bevor du weiterblätterst, diskutiere mit deinem Banknachbar oder in der Gruppe, welche Probleme eintreten können. Notiere deine Ansichten nachfolgend:

……..

……..

……..

……..

……..

……..

……..

……..

……..

……..

……..

……..

2.1.4 Das Problem

Solange nur einzelne Philosophen hungrig sind, funktioniert dieses Verfahren wunderbar. Es kann aber passieren, dass sich alle fünf Philosophen gleichzeitig entschliessen zu essen. Sie ergreifen also alle gleichzeitig ihre linke Gabel und nehmen damit dem jeweils links von ihnen sitzenden Kollegen seine rechte Gabel weg. Nun warten alle fünf darauf, dass die rechte Gabel wieder auftaucht. Dies passiert aber nicht, da keiner der fünf seine linke Gabel zurücklegt. Die Philosophen verhungern.
Eine andere Variante ist die Folgende: Jeder hungrige Philosoph nimmt die zwei nächsten verfügbaren Gabeln, unabhängig davon, von welchem Philosoph sie zurückgelegt wurden. Damit wird z.B. der Fall möglich, dass je zwei Philosophen immer die Gabeln von denselben zwei Philosophen nehmen und der fünfte Philosoph verhungern müsste.

2.1.5 Lösung(en)

Wie könnte die Verklemmung der ersten beschriebenen Variante verhindert werden?
Diskutiere deine Lösungsvorschläge mit deinem Nachbar oder in der Gruppe und notiere sie nachfolgend:

……..

……..

……..

……..

……..

……..

……..

……..

……..

……..

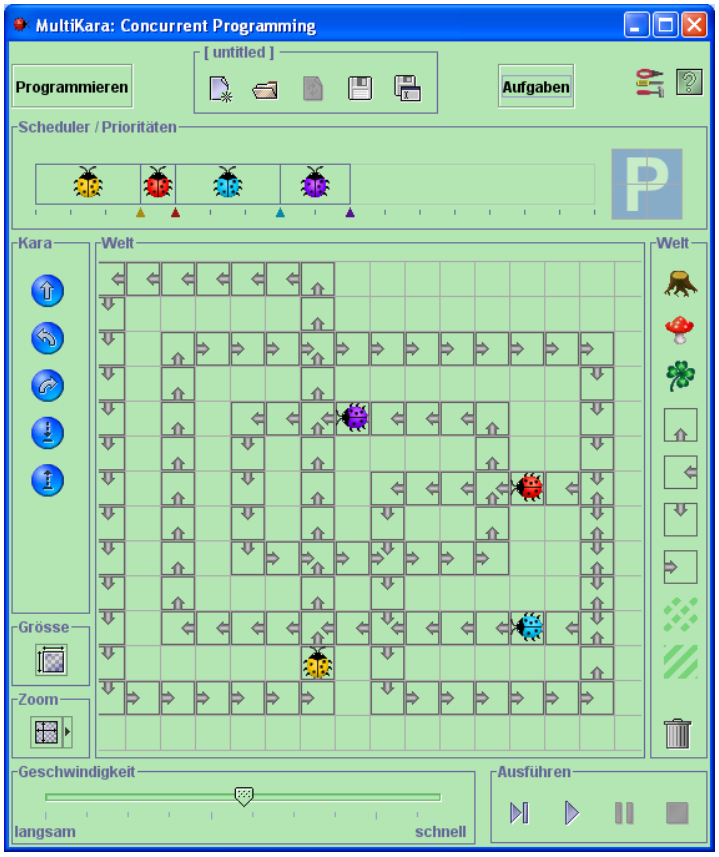
……..

……..

……..

……..

3 Die MultiKara-Umgebung



Übersicht

In diesem Kapitel wird die Lernumgebung von MultiKara vorgestellt. Ausgehend von den bekannten Elementen aus Kara werden die neuen Elemente und Mechanismen im Detail und in Verbindung mit Beispielen und Aufgaben erläutert.

Lernziele

- Du machst dich vertraut mit den neuen Elementen von MultiKara (mehrere Käfer, gerichtete Strassenelemente, Sensorbaukasten).

3.1 Alte Bekannte

Die MultiKara-Umgebung ist eine Erweiterung der Umgebung Kara. Es finden sich dieselben Elemente wie

- *Bäume,*
- *Pilze,*
- *Kleeblätter und natürlich*
- *den Käfer Kara*

Ebenso ist es mit den Aktionen: Nach wie vor vorhanden sind 5 Aktionsmöglichkeiten:

- *einen Schritt geradeaus machen,*
- *links drehen,*
- *rechts drehen,*
- *ein Kleeblatt ablegen,*
- *ein Kleeblatt aufheben.*

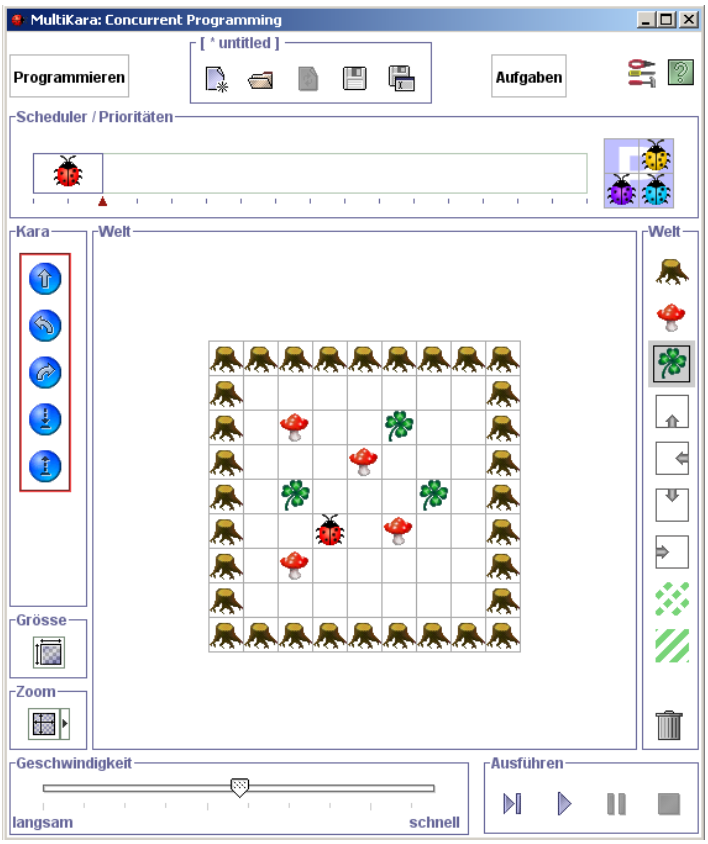


Abb. 3.1 Die MultiKara-Umgebung enthält alle altbekannten Elemente von Kara.

Bei allem, was aber über diese elementaren Bausteine hinausgeht, macht sich die Erweiterung bemerkbar und soll in den nächsten Kapiteln vorgestellt werden.

Hier aber nochmals das Weltfenster von MultiKara mit Erklärungen im Überblick:

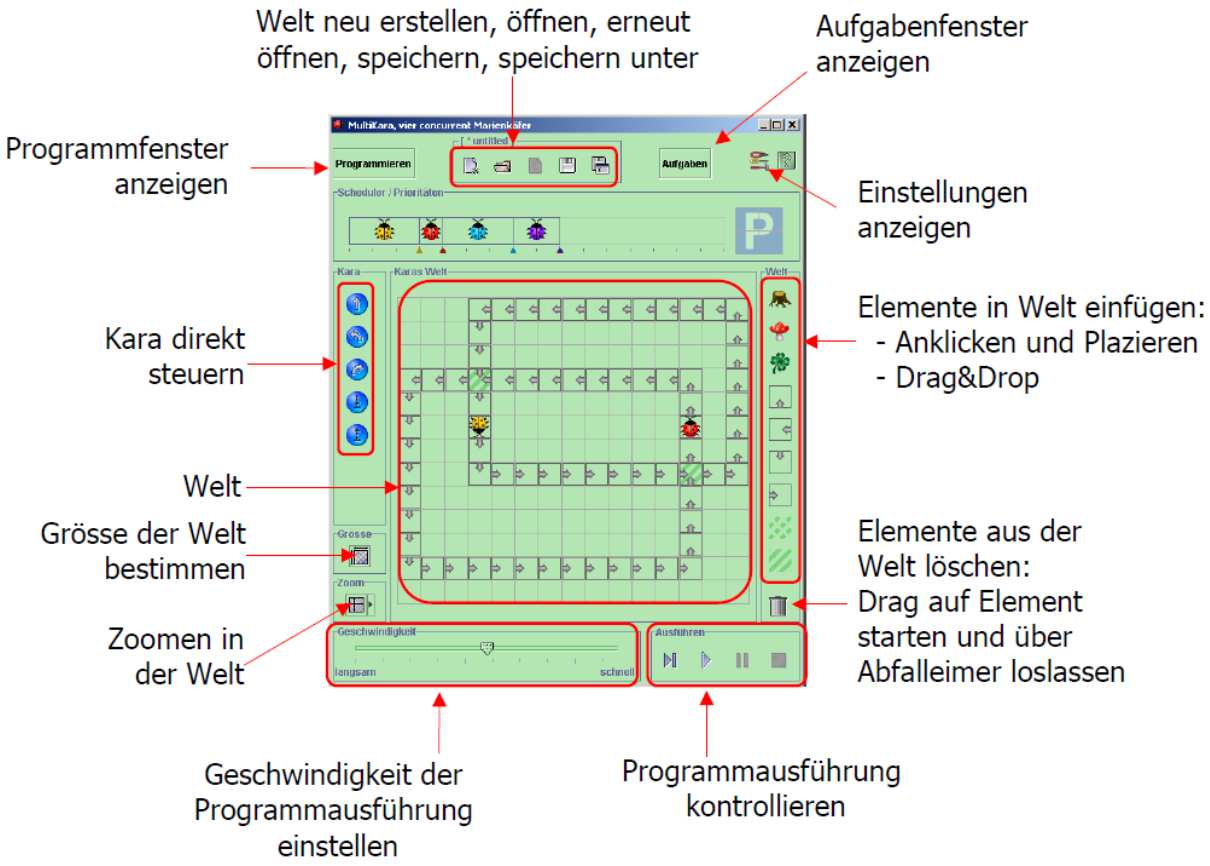


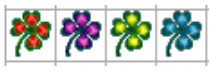
Abb. 3.2 Weltfenster von MultiKara

3.2 Was ist neu bei MultiKara

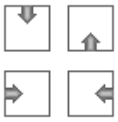
3.2.1 Die Elemente der Welt

In der MultiKara Umgebung können bis zu *vier verschiedenfarbige Marienkäfer* programmiert werden, die in derselben Welt leben.

Will man nicht mit allen Kara arbeiten, kann man die anderen auf dem Parkfeld (P) deponieren.



Neu ist, dass die Marienkäfer *Kleeblätter in ihrer Farbe* legen. Es ist also nachvollziehbar, welches Kleeblatt von welchem Käfer stammt



Neu sind auch *Strassenelemente*, die angeben, aus welcher Richtung ein Feld betreten werden darf. Falls auf einem Feld kein Strassenelement liegt, so dürfen die Käfer wie gehabt das Feld aus jeder Richtung betreten. Die Strassen erweitern das Spektrum der Aufgabenstellung um Aufgaben, bei denen der Käfer nur einen gerichteten Weg abläuft.

Zudem gibt es noch das Feld „Meeting Room“ und „Monitor“. Diese braucht man für das interagieren der verschiedenen Karas untereinander. Dazu später mehr.

Für jeden Käfer kann ein eigener endlicher Automat erstellt werden. Zur Laufzeit werden die Automaten unabhängig voneinander ausgeführt, wobei ihre Prozesse von einem „*Scheduler*“ (Planer, Disponent) ausgelöst werden. Die Automaten können nun via Welt oder mit Hilfe von Synchronisationsmechanismen miteinander interagieren.

3.2.2 Die Elemente des Automaten

In der Welt haben wir die neuen Felder „Meeting Room“ und „Monitor“. Die entsprechenden Funktionen sind bei den Automaten die Bedingungen „Barrier“ und „Critical Section“, welche den Zuständen beigefügt werden können.

Für jeden Käfer kann ein eigener endlicher Automat erstellt werden. Zur Laufzeit werden die Automaten unabhängig voneinander ausgeführt. Durch das Benutzen von „Barrier“ und „Critical Section“ wird nun aber eine Abhängigkeit von gleichen Zuständen verschiedener Karas (Instanzen von gleichen Prozessen) erzwungen. Bei der „Barrier“ werden die Karas beim bezeichneten Zustand angehalten, bis alle Karas diesen Zustand eingenommen haben, erst dann geht es weiter. Bei der „Critical Section“ darf sich jeweils nur ein Kara im bezeichneten Zustand befinden. Dieser ist ausschliesslich nur für jeweils einen Kara bestimmt. Erst wenn der entsprechende Kara diesen Zustand verlassen hat, kann der nächste (durch den Scheduler bestimmte) Kara diesen Zustand einnehmen.

3.2.3 Der Sensor-Baukasten

Eine deutliche Erweiterung erfahren haben die Sensoren. Beim Erstellen eines neuen Zustands erscheint das Sensor-Fenster, das alle bisherigen Sensoren aus Kara (*Baum vorne?*, *Baum links?* oder *Baum rechts*?, *Kleeblatt unten*? *Pilz vorne*?) enthält und zusätzlich den Standard-Sensor "*Kara vorne*?" aufweist – für den Fall, dass sich schon ein anderer Kara auf dem Feld vorne befindet.

Abb. 3.3 Die Sensor-Bibliothek von MultiKara

Interessant ist nun die Option, dass selber Sensoren erstellt werden können. Ein Klick auf "Sensoren erstellen" öffnet das entsprechende Fenster: Für jedes der vier Felder (links, rechts, vorne und das aktuelle vom Käfer aus gesehen) kann Kara nun folgende Eigenschaften prüfen (die Reihenfolge entspricht der Abbildung im nebenstehenden Fenster):

1. Befindet sich dort ein leeres Feld?
2. Befindet sich dort ein fremdes Kleeblatt?
3. Befindet sich dort ein eigenes Kleeblatt?
4. Befindet sich dort ein anderer Käfer?
5. Befindet sich dort ein Baumstamm?
6. Befindet sich dort ein Pilz?
7. Kann das Feld von unten her betreten werden?
8. Kann das Feld von links her betreten werden?
9. Kann das Feld von oben her betreten werden?
10. Kann das Feld von rechts her betreten werden?

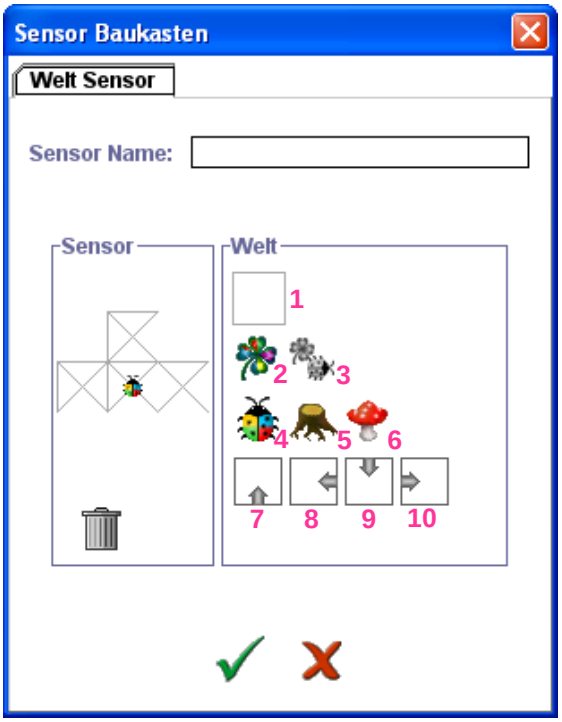
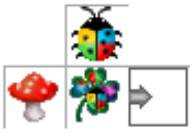


Abb. 3.4 Der Sensor-Baukasten von MultiKara erlaubt eigene Sensoren zu definieren.

Mit all den Eigenschaften sind im Prinzip 11^4 Sensoren möglich[1] (unsinnige mitgezählt), da alle 4 Positionen gleichzeitig besetzt werden können also auch ein Multisensor gebaut werden kann.

Dies kann vor Vorteil sein, muss aber nicht, denn man kann den Sensor nur als Ganzes mit „yes“ oder „no“ auswerten um damit Aktionen auszulösen:

[1] Es gibt 10 Sensoren zur Auswahl und die 11. Möglichkeit ist gar keinen Sensor für je eines der 4 Felder zu wählen.

Sensor : 

Auswertung:

Betrachten wir aber einige Eigenschaften doch näher:

a) *Kleeblätter*
Mit den Eigenschaften 2 und 3 können die Käfer *eigene Kleeblätter* von *fremden Kleeblättern* unterscheiden. Sie können aber nicht die Farbe fremder Kleeblätter auseinander halten.

b) *Strassen*
Mit den Eigenschaften 7 bis 10 ist es möglich, in der Welt eine Strasse zu definieren entlang welcher sich der Käfer bewegen kann. Entsprechende Sensoren sind aber notwendig, damit er nicht von der Strasse abkommt.

3.2.4 Das Zeit-Modell und der Scheduler

Sind mehrere Karas gleichzeitig aktiv, d.h. haben ein Programm, das nicht schon terminiert ist, dann löst der Scheduler (eine übergeordnete Instanz zur Prozesssteuerung) die einzelnen Prozessschritte von den Karas aus.
Die Zeit wird in diskrete Zeitschritte (clock ticks) unterteilt, wobei bei jedem "tick" ein Zustandsübergang irgendeines Karas ausgeführt wird.

Die Auswertung der Sensoren, die Ausführung des jeweiligen Programmschrittes und der zugehörige Zustandsübergang finden "atomar" statt, d.h. in einem Zug ohne dass parallel dazu andere Prozesse laufen oder interagieren (würde z.B. zwischen der Auswertung des Sensors "vorne frei" und dem "Schritt nach vorne" ein anderer Kara schnell auf das freie Feld sitzen, käme es zur Kollision). Dies ist speziell bei Kara. Auf normalen Prozessoren mit nebenläufigen Prozessen ist dies so nicht der Fall und es kann zu Kollisionen kommen, wenn der Programmierer dies nicht durch einen intelligenten Algorithmus (gegenseitiger Ausschluss) abfängt. Dazu später mehr.

Standardmässig wählt der Scheduler den nächsten Prozessschritt zufällig aus, wobei die Priorität für einzelne Karas über den Schieberegler (im Scheduler / Prioritäten - Feld) eingestellt werden kann. Anhand der Breite des entsprechenden Karas weiss der Scheduler, welche Karas er gegenüber anderen zeitlich bevorzugen soll. Welcher Kara als nächster einen Schritt macht, wird durch die Farbunterlegung angezeigt.

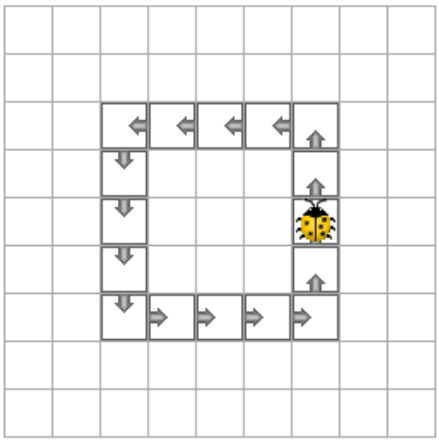
Für die Fehlersuche kann man weiterhin die Karas in Einzelschritten ihre Automaten abarbeiten lassen. Der Scheduler übernimmt aber weiterhin die Zuteilung der Prioritäten.

Will man totale Kontrolle über jeden einzelnen Kara kann auch "manuelles Scheduling" über die Einstellungen aktiviert werden. So kann der nächste Schritt (welcher Kara und wie viele Schritte) selber gewählt werden. Im Ausführen-Fenster sind die „Tasten“ für die Einzelschritte in der Farbe des entsprechenden Karas eingefärbt.

So, nun ist es aber Zeit, einige der neuen Möglichkeiten einmal auszuprobieren

Aufgabe 3.1

Ein einzelner Kara befindet sich im untenstehenden Rundkurs. Schreibe ein Programm in MultiKara, so dass der Käfer endlos Runden auf der vorgegebenen Strasse dreht.



Wenn Du die Aufgabe ohne Hilfe versuchen möchtest, lies noch nicht weiter. Vergiss aber nicht, den Automaten grafisch zu beschreiben.

Ansonsten gibt es folgend eine schrittweise Anleitung zur Lösung.

- Zeichne die Strasse auf ein 9x9 Feld und platziere den Kara irgendwo auf der Strasse mit Blickrichtung entlang der Strasse. Speichere die Welt ab.
- Bevor wir nun ins Programmieren gehen, analysieren wir die Aufgabenstellung: Versuche deshalb kurz zu beschreiben, was der Käfer machen soll.

  ……

  ……

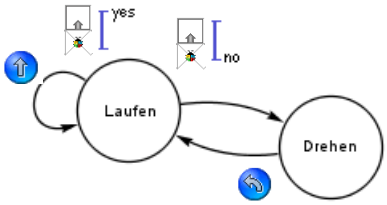
  ……..

  ……

  Lösung:
  Solange ein Strassenstück vor dem Käfer liegt, macht er einen Schritt vorwärts, ansonsten dreht er sich nach links. Dann geht es immer gleich weiter.

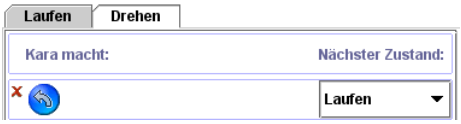
– Versuche nun den Automaten anhand der obigen Beschreibung zu zeichnen:
TIPP: Die Zustände könnten wir „Laufen“ und „Drehen“ nennen. Der Käfer muss auch in der Lage sein zu erkennen, ob noch ein Strassenstück vor ihm liegt, dazu müssen wir einen Sensor nehmen.

Lösung:



– Gehe nun in die Programmierumgebung von Kara (Klicke auf „Programmieren“).
– Erstelle einen neuen Zustand durch Klicken auf und nenne ihn z.B. „Laufen“. Warte noch mit dem ✓ klicken, wir erstellen zuerst noch den nötigen Sensor.
– Klicke auf Sensor erstellen und nenne den Sensor z.B. „Strasse vorne?“, denn es muss ja ein Strassenstück vor ihm liegen, damit er einen Schritt machen kann. Dazu ziehe die Eigenschaft auf das Feld vor dem kleinen Käfer: . Klicke dann ✓ und damit hast Du deinen ersten eigenen Sensor hergestellt.
– Ziehe nun den neu erstellten Sensor in das Feld Benutzte Sensoren des Zustandes und klicke dann abermals ✓ um nun den Zustand zu programmieren.
– Erstelle noch den Zustand „Drehen“, hier ist aber kein Sensor nötig.
– Der Zustand „Laufen“ hat nun 2 Übergänge: Entweder ist vor dem Käfer ein Strassenstück, dann kann er einen Schritt machen und er kann wieder auf den Zustand „Laufen“ verweisen, oder es ist kein Strassenstück mehr da, dann bleibt er stehen und er muss auf den Zustand „Drehen“ verweisen.

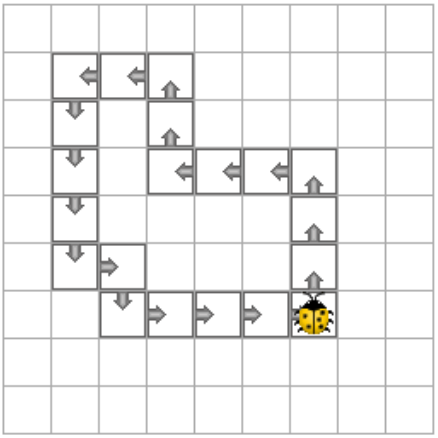
– Der Zustand „Drehen“ dreht den Käfer nur einmal nach links und verweist dann auf den Zustand „Laufen“.



– Als Letztes musst Du nur noch den start auf den Zustand „Laufen“ setzten. Bitte Abspeichern nicht vergessen. Nun kannst Du den Käfer laufen lassen.

Aufgabe 3.2

Versuche es nun selbst: Ein einzelner Kara befindet sich im untenstehenden Rundkurs. Schreibe ein Programm in MultiKara, so dass der Käfer endlos Runden auf der vorgegebenen Strasse dreht.



Beantworte vorgängig folgende Fragen:

- Welche Zustände sind nötig?

  ……

  ……

- Welche Sensoren brauchst Du?

  ……

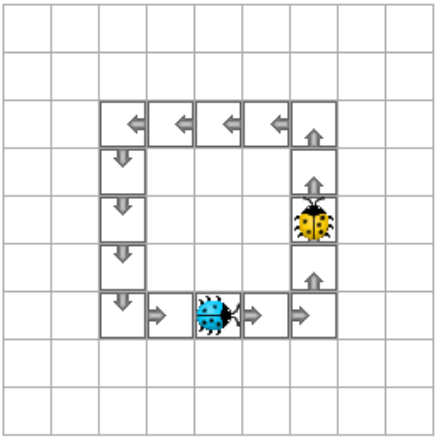
  ……

- Zeichne den Automaten mit seinen Übergängen:

- Nun kannst Du den Käfer programmieren und die nötigen Sensoren erstellen.

Aufgabe 3.3

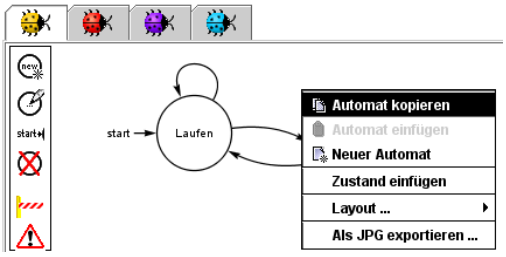
So, nun probieren wir noch aus, was der Scheduler bewirken kann. Lade dazu die Welt und dein Programm aus der Aufgabe 3.1 (Du hast sie ja abgespeichert, oder?). Setze nun noch einen zweiten Käfer wie abgebildet in den Rundkurs:



Dazu musst Du den blauen Käfer vom Parkplatz in die Welt ziehen (und noch drehen) Zudem musst Du ihn auch auf die Leiste des Schedulers ziehen und die Breite anpassen (durch Klicken und Ziehen auf das kleine blaue Dreieck am rechten unteren Rand des Rechtecks in dem sich der entsprechende Kara befindet). Es sollte dann so aussehen.

Speichere die Welt unter einem anderen Namen ab, damit Du sie jeweils genau so wieder herstellen kannst.

Nun muss der blaue Kara noch den gleichen Automaten erhalten wie der Gelbe. Gehe dazu ins „Programmieren“ und wähle den gelben Käfer durch Klicken auf die entsprechende Registerkarte aus. Im Feld, wo der Automat angezeigt wird, klicke die rechte Maustaste und wähle „Automat kopieren“.



Dann klicke auf die Registerkarte des blauen Käfers und klicke wiederum die rechte Maustaste im leeren Feld des Automaten. Wähle dann „Automat einfügen“.

Bevor Du nun die Käfer laufen lässt, beantworte die folgende Frage:

- Kann es zu einer Verklemmung kommen und wieso?

……

……

……

……

Nun lass die Käfer laufen, was stellst Du fest?

……

……

……

……

Verändere auch die Prioritäten der Käfer im Scheduler-Fenster, was stellst Du fest?

……

……

……

……

4 Synchronisationsmechanismen

Übersicht

Die neuen Elemente und Mechanismen werden nun im Detail und in Verbindung mit Beispielen und Aufgaben erläutert. Im Zentrum dieses Kapitels stehen die von MultiKara zur Verfügung gestellten Synchronisationsmechanismen, welche sowohl in der Welt als auch im Zustandsraum eine Lösung für inklusive (handshake) und exklusive Synchronisation (wechselseitiger Ausschluss) anbieten.

Lernziele

- Du machst dich vertraut mit den Synchronisationsmechanismen von MultiKara.
- Du erkennst die Notwendigkeit von Synchronisationsmechanismen und kannst diese in einer konkreten Problemstellung adäquat verwenden.

4.1 Arbeitsteilung ohne Synchronisation

Die Kapitel 1 und 2 legen nahe, dass bei nebenläufigen Prozessen – wie beim Programmieren mehrerer Karas – i.a. Synchronisationsmechanismen notwendig werden, falls es nicht zu Verklemmungen kommen soll. In Ausnahmefällen kommt man auch ohne aus, wobei die Komplexität der Problemstellung deutlich zunimmt, wie die folgenden Aufgaben zeigen werden.

Aufgabe 4.1

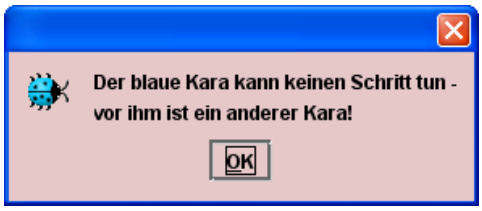
Ein einzelner Kara befindet sich in der untenstehenden eindimensionalen Welt zwischen zwei Baumstämmen. Seine Aufgabe besteht darin, alle leeren Felder mit einem Kleeblatt zu belegen und dann anzuhalten. Beschreibe den Automaten und schreibe anschliessend das zugehörige Programm in MultiKara.

Aufgabe 4.2

Die Lösung von Aufgabe 4.1 dürfte nicht allzu schwer fallen, falls man mit Kara vertraut ist. Schwieriger wird die Situation, wenn Kara noch einen Helfer bekommt (blau). Die Aufgabe ist nach wie vor, alle freien Felder mit (irgend) einem Kleeblatt zu bedecken. Beschreibe den Automaten und programmiere anschliessend.

Hinweise:

- *Das Programm sollte im nichtdeterministischen Modus bei unterschiedlichen Prioritäten immer ausführbar bleiben (keine deadlocks).*
- *Ein Kara kann kein Feld betreten, auf dem sich schon ein Kara befindet. Versucht er es trotzdem, kommt es zum Abbruch und zur folgenden Fehlermeldung:*



Aufgabe 4.3

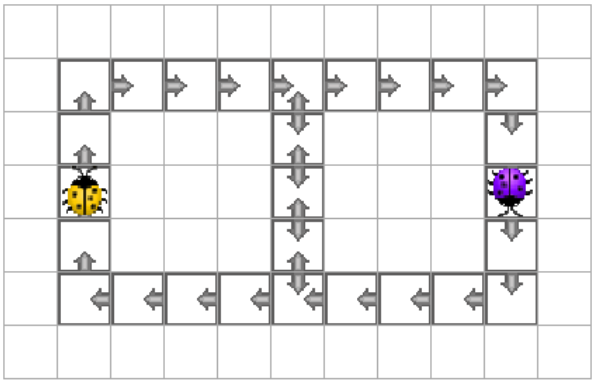
Falls du Aufgabe 3.2 erfolgreich gelöst hast, überprüfe deine Lösung mit allen vier Karas. Funktioniert dein Ansatz in der folgenden Welt immer noch? Beachte, dass du für jeden Kara dasselbe Programm (denselben Automaten) verwendest.

4.2 Motivation von Synchronisationsmechanismen

Die Sensoren und insbesondere der erweiterte Baukasten erlauben den Karas ihre unmittelbare Umgebung wahrzunehmen und in einigen Fällen so intelligent zu reagieren, dass es zu keinen Verklemmungen (deadlocks) kommt wie die Aufgaben 4.2 und 4.3 gezeigt haben. Einfach ist die Sache aber nicht gerade. Zudem hängt die prinzipielle Lösbarkeit des Problems ohne Synchronisationsmechanismen stark von der Aufgabenstellung ab. Die nachfolgende Aufgabe soll anhand eines simplen Problems die Notwendigkeit von Synchronisationsmechanismen demonstrieren.

Aufgabe 4.4: Stau im Tunnel

Die folgende Welt zeigt zwei Rundstrecken, die von zwei Karas endlos durchlaufen werden, der gelbe läuft auf der linken Strecke, der violette auf der rechten. Ein Teil der Strecke verläuft gemeinsam und wird in Gegenrichtung durchfahren.



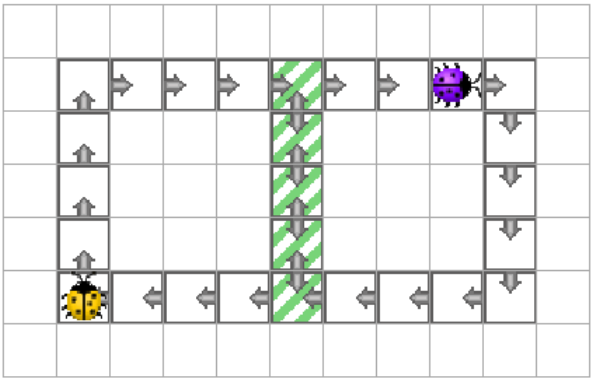
a) Baue die nebenstehend abgebildete Welt nach.

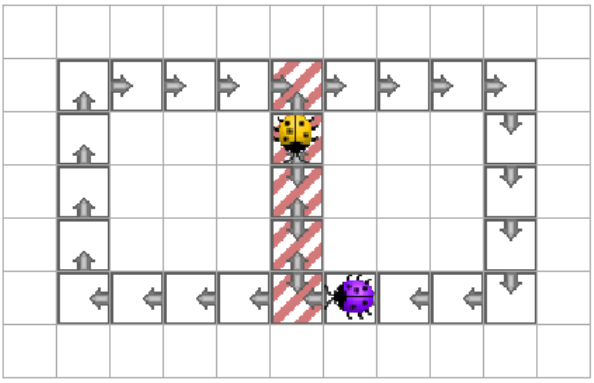
b) Lass die Karas endlos Runden drehen und überzeuge dich davon, dass es in der untenstehenden Welt zu Blockaden kommen kann.

   TIPP: Lasse zuerst nur einen Kara auf seiner Hälfte kreisen um den Automaten zu testen. Dann kopiere ihn für den anderen Käfer und lasse nun beide laufen.

c) Lassen sich die Blockaden durch Sensoren und/oder weitere Zustände verhindern?

d) Falls dir bei c) ein Lösung gelungen ist, Gratulation!! Die Umgebung MultiKara hat dafür eine sehr einfache Lösung parat. Die Lösung heisst *Monitor* und ist ein exklusiver Synchronisationsmechanismus, der nur einen Kara in einen markierten Bereich hinein lässt. Der problematische Bereich ist hier der Tunnel mit Gegenverkehr. Markiere diesen als Monitor (s. unten) und du wirst feststellen, dass sich die Probleme von selbst lösen:





Funktion des Monitors: *Links: Monitor ist* ***frei*** *(grün**)** und kann betreten werden; rechts der Monitor ist* ***belegt*** *(rot) durch einen Kara und verhindert das Eintreten des Zweiten.*

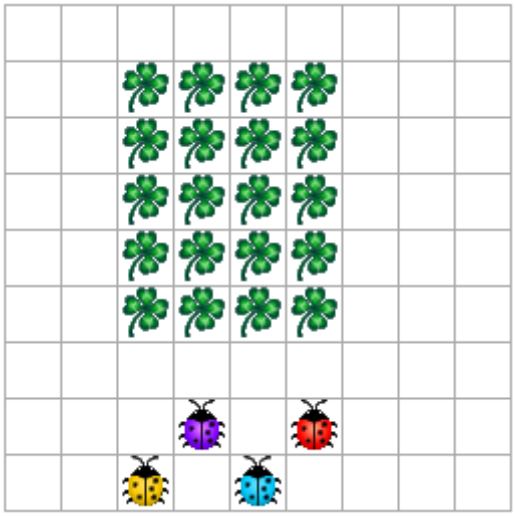
Aufgabe 4.5: Gemeinsames Kleeblattsammeln

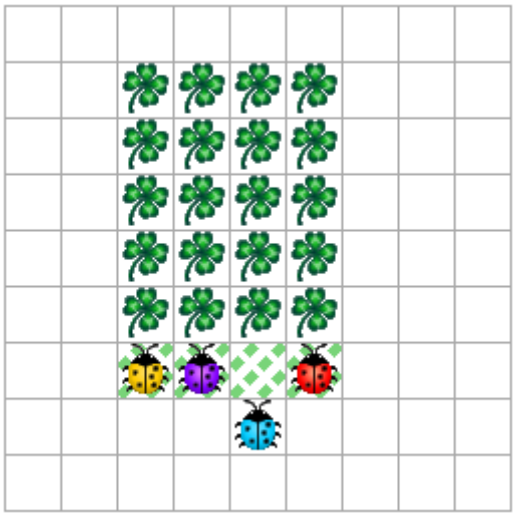
Vier Karas treffen zu unterschiedlichen Zeiten bei einem Rechteck auch Kleeblättern ein und wollen diese gemeinsam auflesen. Anstandshalber wird vereinbart, dass alle erst mit Auflesen (oder essen) beginnen, wenn sich alle unmittelbar in der Zeile unter den Kleeblättern eingefunden haben.

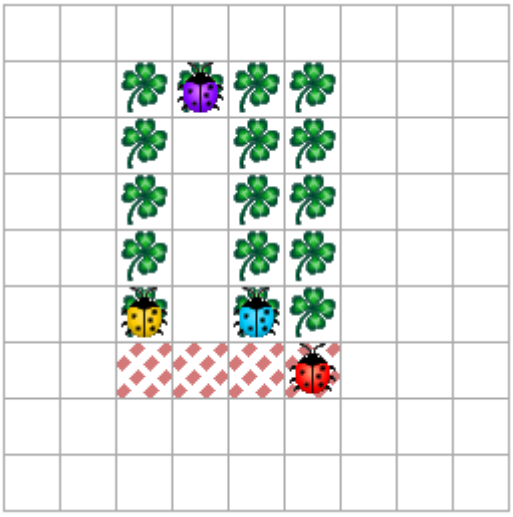
a) Baue nebenstehende Welt nach

b) Gibt es eine Möglichkeit mittels Zuständen und/oder Sensoren dafür zu sorgen, dass kein Kara vorzeitig mit "Fressen" beginn?

c) Falls es dir gelungen ist bei b) eine Synchronisation über Sensoren oder Zustände zu finden Gratulation! MultiKara bietet auch für dieses Problem eine relativ einfache Lösung an: den ***Meeting Room***. Ein Ort des Zusammentreffens, der erst wieder verlassen werden kann, wenn vorgängig alle eingetroffen sind.







Funktion des Meeting Room*: Links im Zustand "**in**" (grün): Drei Karas warten auf die Ankunft des vierten; rechts im Zustand "out"(rot): Alle vier Karas waren eingetroffen und verlassen nun mit unterschiedlicher Geschwindigkeit und Reihenfolge den Meeting Room.*

4.3 Synchronisationsmechanismen im Detail

4.3.1 Synchronisation in der Welt: Monitor & Meeting Room

In den Aufgabe 4.4 und Aufgabe 4.5 haben wir zwei Methoden kennengelernt, mit denen auf eine relativ einfache Art ein wechselseitiger Ausschluss (nur ein Kara darf in einen Bestimmten Bereich) oder die inklusive Synchronisation (alle Karas müssen zuerst an einem Ort eingetroffen sein) realisierbar ist.

Der **Monitor garantiert die exklusive Synchronisation (wechselseitiger Ausschluss)**. In MultiKara gibt es einen einzigen Monitor, der für die jeweils markierten Felder aktiv ist. Der Monitor selber kann von den Karas hingegen nicht wahrgenommen werden. Wenn ein Käfer ein Monitorfeld betritt, betritt er den globalen Monitor, in dem sich nur ein Käfer aufhalten darf. Die Koordination der Prozesse obliegt dem Scheduler. Der Monitor hat zwei Zustände: "free" und "busy"; dabei bedeutet

free: *Der Monitor ist grün gefärbt; kein Kara befindet sich im markierten Bereich und dieser kann von einem Käfer betreten werden.*

busy: *Der Monitor ist rot gefärbt; ein Kara befindet sich bereits im markierten Bereich und dieser kann von keinem weiteren Käfer betreten werden. Will trotzdem ein anderer Käfer diesen Bereich betreten, ist dieser für ihn gesperrt. Der Käfer wird schlafen gelegt, bis der Monitor wieder frei ist.*

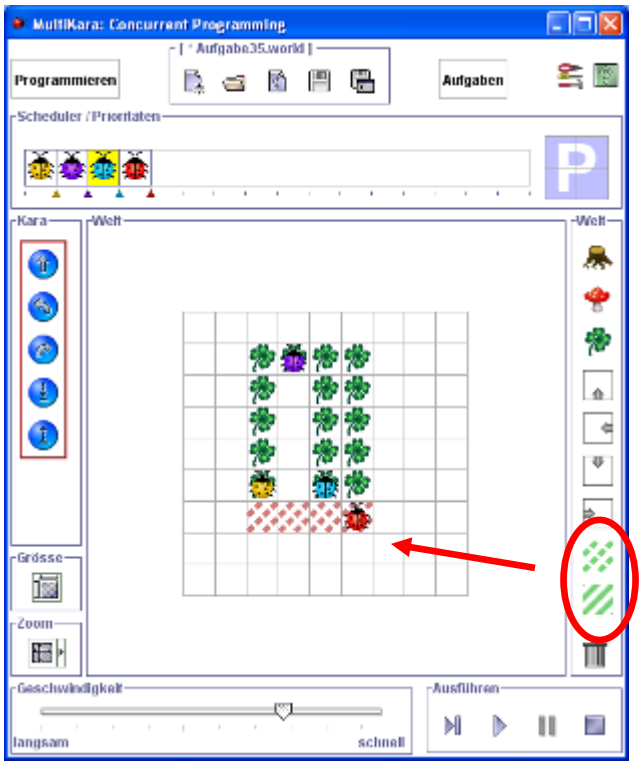


Abb. 4.1 Die beiden Synchronisationsmechanismen in der Welt von Kara: Meeting Room (punktiert) und Monitor (schraffiert).

Der **Meeting Room garantiert die inklusive Synchronisation (hand shake)** in der Welt der Karas. In MultiKara gibt es einen einzigen Meeting Room, der für die jeweils markierten Felder aktiv ist. Der Meeting Room selber kann von den Karas nicht wahrgenommen werden. Die Koordination der Prozess obliegt dem Scheduler. Der Meeting Room hat zwei Zustände: "in" und "out"; dabei bedeutet

in: *Der Meeting Room ist grün gefärbt; die Karas können diesen Bereich betreten, ihn aber nicht verlassen. Alle Käfer die diesen Bereich betreten haben, werden vom Scheduler schlafen gelegt.*

out: *Der Meeting Room ist rot gefärbt; die Karas können diesen Bereich verlassen, ihn aber nicht betreten. Versuchen sie trotzdem in den rot gefärbten Bereich einzutreten, werden sie solange schlafen gelegt, bis dieser wieder im Zustand "in" ist. Der Meeting Room wechselt in den Zustand "in", wenn der letzte Käfer den Meeting Room verlassen hat.*

4.3.2 Synchronisation im Zustandsraum: Barrier & Critical Section

Dieselben Synchronisationsmechanismen, die wir jetzt in der Welt kennengelernt haben, lassen sich auch im Zustandsraum – d.h. auf der Ebene der endlichen Automaten – realisieren. Dabei gelten folgende Entsprechungen:

	inklusive Synchronisation	*exklusive Synchronisation*
Welt	**Meeting Room**	**Monitor**
Zustandraum	**Barrier**	**Critical Section**

Barrier-Zustände:
Wenn mehrere Prozesse (mehrere Karas) gemeinsam ein Problem lösen, lässt sich der Ablauf häufig in verschiedene Phasen unterteilen. Jeder Prozess muss eine Phase beendet haben, bevor die Prozesse die nächste Phase beginnen dürfen. In MultiKara helfen die **Barrier-Zustände**, den Programmablauf in Phasen zu unterteilen. Die Barrier im Programm entspricht dem Meeting Room in der Welt. Es gibt eine einzige globale Barrier, die aktiv wird, sobald ein Prozess einen Barrier-Zustand betritt. Im Programmeditor werden Barrier-Zustände mit einer dicken, gestrichelten Umrandung gekennzeichnet (Abbildung 4.2)

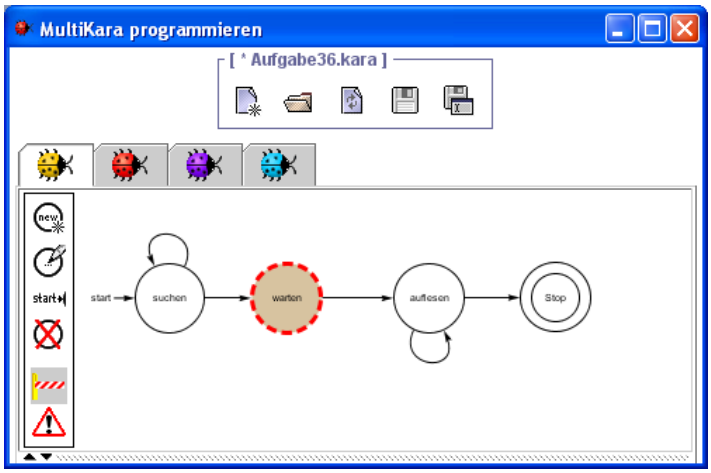


*Abb. 4.2 Der Barrier-Zustand "**warten**" garantiert, dass die Karas erst mit auflesen beginnen, wenn der letzte beim Zustand warten angelangt ist. (Das ist ein Tipp für Aufgabe 4.6)*

Aufgabe 4.6: Gemeinsames Kleeblattsammeln (Lösung mit Barrier-Zuständen)

Löse die Aufgabe 4.5, indem du anstelle des Meeting-Rooms mit Barrier-Zuständen die inklusive Synchronisation realisierst. (Nicht vergessen den Meeting Room aus der Welt von Aufgabe 4.5 entfernen).

Critical Section-Zustände:

Wenn mehrere Prozesse dieselben Ressourcen verwenden, besteht die Gefahr, dass es zu Blockaden (deadlocks, lifelocks) kommt. Mit dem Monitor-Konzept kann der gegenseitige Ausschluss in bestimmten Bereichen der Welt realisiert werden.

Mit den **Critical Section-Zuständen** kann der wechselseitige Ausschluss ortsunabhängig im Zustandsraum (der endlichen Automaten) realisiert werden. Es gibt eine **globale Critical Section**, die aus allen Critical Section-Zuständen aller Automaten besteht. Zu jedem Zeitpunkt kann höchstens ein Prozess in einem Critical Section-Zustand sein. Im Programmeditor sind die kritischen Zustände durch dicke grüne Umrandungen markiert. Ein Prozess beansprucht die Critical Section, sobald er einen kritischen Zustand betreten will. Ist die Critical Section frei, so kann der Prozess den Zustand betreten. Die kritischen Zustände aller anderen Prozesse sind dadurch gesperrt und zur Visualisierung im Programmeditor rot eingefärbt. Versucht trotzdem ein Prozess einen kritischen Zustand zu betreten, so wird er vom Scheduler schlafen gelegt und erst wieder geweckt, wenn die Critical Section wieder frei ist.[1]

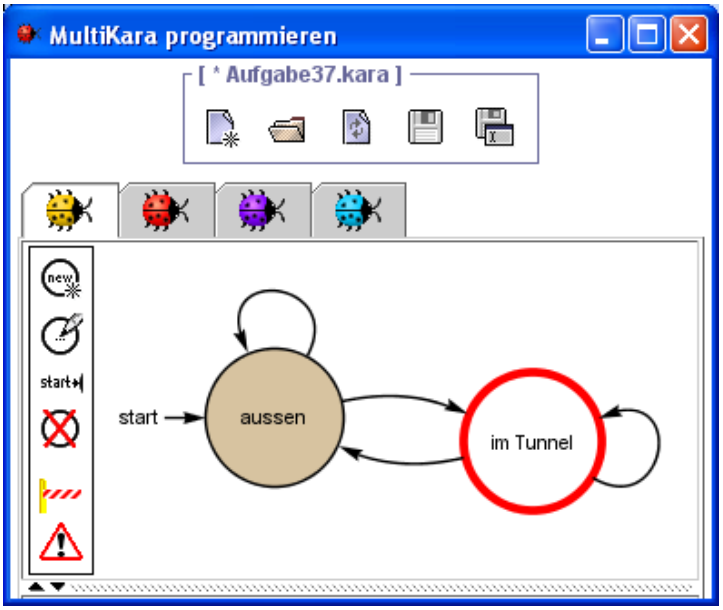
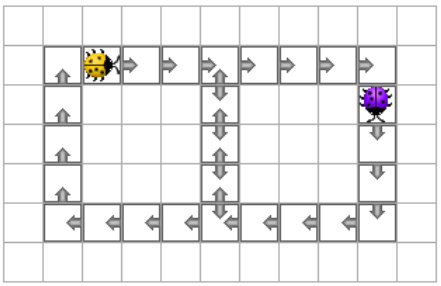


*Abb. 4.3 Der Critical Section-Zustand "**im Tunnel**" garantiert, dass maximal ein Kara sich in diesem Zustand befindet. (Das ist ein Tipp für Aufgabe 4.7)*

Aufgabe 4.7: Stau im Tunnel (Lösung mit Critical Section-Zuständen)

Löse die Aufgabe 4.4, indem du anstelle des Monitors mit Critical Section Zuständen die exklusive Synchronisation realisierst. (Nicht vergessen den Monitor aus der Welt von Aufgabe 4.4 entfernen; vgl. Abb. nebenan).



5 Aufgaben

Die nachfolgenden Aufgaben stammen mehrheitlich aus der Lernumgebung MultiKara (jar-file) und sind der besseren Übersicht und der Vollständigkeit halber nochmals wiedergegeben. Der Schwierigkeitsgrad der Aufgaben wird durch Sterne markiert von einfach (*) über mittel (**) bis schwierig (***).

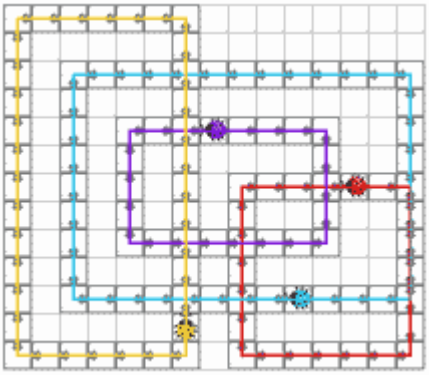
Aufgabe 5.1: Bus-Umsteige-Problem *

Zwei Karas befinden sich je auf einem Strassen-Rundkurs (3 verschiedene Welten sind gegeben), den sie endlos ablaufen sollen. Programmieren Sie die Karas so, dass jeder bei der Bushaltestelle auf den anderen wartet. Die Bushaltestelle wird repräsentiert durch das mit einem Baum angedeutete Warte-Häuschen.

Hinweis:
Überlegen Sie sich, mit welchem Mechanismus die Synchronisation der beiden Busfahrer-Käfer stattfinden soll.

Aufgabe 5.2: Verkehrssimulation *

Vier Karas möchten entlang ihrer Strasse laufen. Jeder Kara hat eine eigene Route; Gegenverkehr oder Kreuzungen sind möglich.



Programmieren Sie die Karas so, dass sie ewig im Kreis herumlaufen ohne zu kollidieren.

Aufgabe 5.3: Strecke halbieren *

Mit Hilfe von zwei Karas soll eine Strecke halbiert werden. Zu Beginn befinden sich die beiden Käfer irgendwo zwischen den Bäumen, die das Ende der Strecke markieren (oberes Bild). Die Käfer schauen zufällig entweder nach links oder nach rechts. Am Ende sollen sich die Käfer voreinander in der Mitte der Strecke befinden (unteres Bild).

Zusatzaufgabe (**)

Die Käfer sollen die Mitte der Strecke durch ein Kleeblatt (nicht zwei!) markieren. Bei einer ungeraden Anzahl Felder soll sich das Kleeblatt genau in der Mitte, bei einer geraden Anzahl auf einem der beiden mittleren Felder befinden.

Aufgabe 5.4: Tunnel *

Zwei Karas sollen endlos einem Gang entlanglaufen (in die entgegengesetzte Richtung) ohne zu kollidieren. Im Bereich des mit Bäumen begrenzten Ganges hat es genug Platz, so dass sich zwei Käfer ausweichen können. Aber in den mit Pilzen verengten Passagen ist zu eng, als dass sie aneinander vorbei gehen könnten.

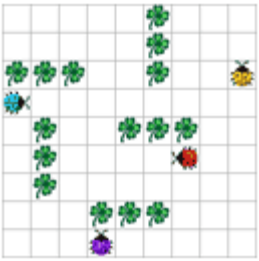
Hinweis:
Verkehrsregeln wie z.B. "Rechtsverkehr" helfen bei der Lösung der Aufgabe.

Aufgabe 5.5: Rechteck füllen **

Vier Karas möchten ein von Bäumen begrenztes Rechteck beliebiger Grösse mit Kleeblättern füllen. Jeder Kara soll das gleiche Programm verwenden und alle Karas sollen am "Füllvorgang" teilnehmen.

Aufgabe 5.6: Kleeblatt-Blöcke **

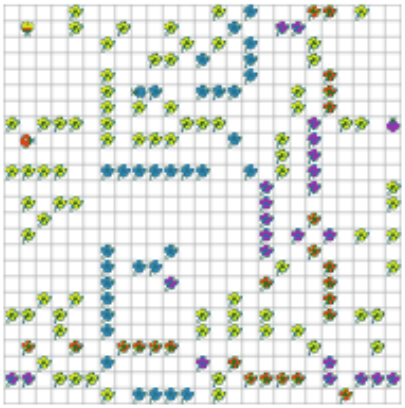
In der Welt liegen Blöcke von je drei nebeneinander liegenden Kleeblättern. Die Karas sollen einen solchen Block finden und die Kleeblätter durch eigene ersetzen.



Hinweis:
Da die Karas grüne Kleeblätter nicht von denen anderer Karas unterscheiden können, kann es vorkommen, dass ein Kara die Kleeblätter eines anderen Karas durch die eigenen ersetzt.

Aufgabe 5.7: Konsumenten (Consumer) und Produzenten (Producer) ***

In einer leeren Welt soll ein Kara zufällige Kleeblattreihen legen (Producer), die die anderen drei Karas suchen und durch eigene Kleeblätter ersetzen sollen (Consumer).



Der Producer-Kara soll Kleeblattreihen legen, deren Länge zufällig ist sind. Die Kleeblatt-Blöcke sollen einander nicht direkt berühren. Diagonal aneinander liegende Blöcke sind erlaubt.
Die Consumer-Karas sollen zufällig herum laufen und Kleeblatt-Reihen suchen. Der Einfachheit halber brauchen sie nicht zwischen Kleeblatt-Farben zu unterscheiden. Sobald ein Consumer-Kara eine Reihe von Kleeblättern gefunden hat, soll er die ganze Reihe von Kleeblättern durch eigene ersetzen.

Aufgabe 5.8: Pilze Lagern *

Zwei Karas möchten Pilze in einen Lagerraum einräumen.

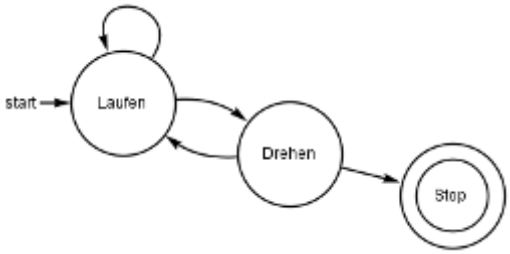
Der gelbe Marienkäfer soll die Pilze einsammeln und sie dem violetten Käfer übergeben, der dann die Pilze im Lagerraum einräumt.

Hinweis:
Es kann hilfreich sein, wenn man gewisse Stellen mit Kleeblättern markiert. Im Bild wurde z.B. der Eingang zum Lagerraum mit einem grünen Kleeblatt markiert.

6 Lösungen zu den Aufgaben

Nachfolgend finden sich Lösungsvorschläge für die Aufgaben in den Kapiteln. Wie beim Programmieren üblich, gibt es für eine Problemstellung oft mehrere (funktionierende) Lösungen. Eine vollständige Übereinstimmung des eigenen Programms mit dem Lösungsvorschlag dürfte – so man die Aufgabe ohne Hilfe gelöst hat – eher die Ausnahme sein. Trotzdem kann aus den vorliegenden Lösungen die eine oder andere Idee geholt oder abgeglichen werden.

Lösung zur Aufgabe 3.2



Lösung zur Aufgabe 3.3

Kann es zu einer Verklemmung kommen und wieso?

Wenn der Scheduler die Prioritäten gleich auf die beiden Käfer verteilt, kommt es zu keiner Verklemmung, da beide Käfer gleiche Fortschritte machen werden. Im Mittel bleibt der Abstand gleich gross.

Nun lasse die Käfer laufen, was stellst Du fest?

Die Realität ist anders, obwohl der Scheduler beiden Käfern die gleiche Priorität zukommen lässt, kann es vorkommen, dass der eine Käfer grössere Fortschritte macht und deshalb den Anderen einholt.

Bem: Damit ist die Idee, dass man selber Prioritäten im Scheduler setzen kann, nicht wirklich sinnvoll. Es ist nicht klar, ob dies ein Fehler des Karaprogramms ist.

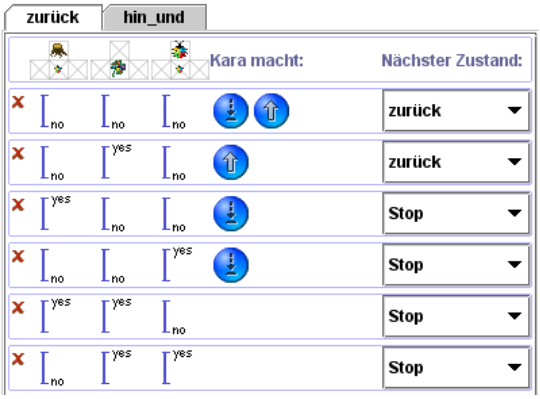
Verändere auch die Prioritäten der Käfer im Scheduler-Fenster, was stellst Du fest?

Lösung zur Aufgabe 4.1

Annahme: Kara geht zuerst nach rechts: Dann lässt sich die Aufgabe mit den folgenden zwei Zuständen lösen:

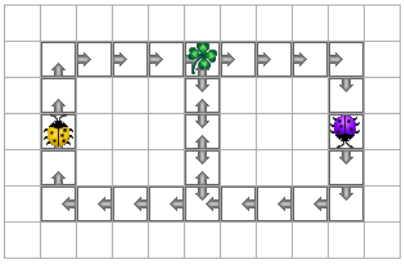
Lösungen zu Aufgabe 4.2 und Aufgabe 4.3

Die Lösungsidee:
Kara geht einmal hin und dann zurück (2 Zustände). Zum Umkehren gezwungen (hin) oder gestoppt (zurück) wird Kara durch einen anderen Kara oder einen Baumstamm. Dieser Algorithmus könnte noch zeitoptimiert werden, nur wird das Programm dann noch etwas länger (fakultative Aufgabe). Der nachfolgende Algorithmus sollte aber bei allen Karas unter allen Bedingungen (verschiedene Prioritäten, deterministischer und nicht deterministischer EA, verschiedene Anfangsausrichtungen der Karas) funktionieren, d.h. es treten unter keinen Umständen Verklemmungen auf. Der Automat ist für alle Karas identisch (einmal programmieren, dann copy / paste).

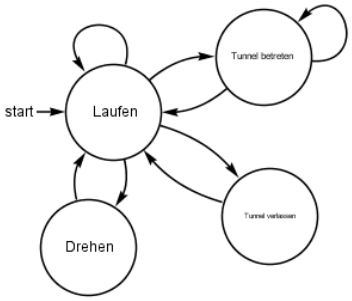


Lösung zur Aufgabe 4.4: Stau im Tunnel

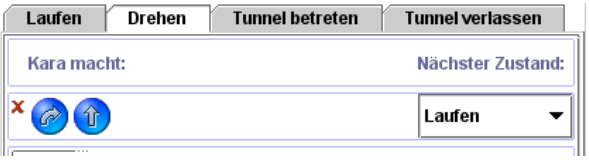
Lösung ohne Monitor:
Damit sich die Karas nicht im Tunnel im Weg stehen, müssen sie einander mitteilen können, dass der Tunnel frei ist, bzw. für den Anderen blockiert ist. Dies kann man z.B. mit einem Kleeblatt beim jeweiligen Tunneleingang machen. Liegt das Kleeblatt, darf der Käfer hinein. Wenn er durch den Tunnel ist, legt er ein Kleeblatt um dem Anderen zu signalisieren, dass der Tunnel frei ist. Diese Lösung hat zwei Unschönheiten. 1. können die Käfer nur abwechslungsweise durch den Tunnel und 2. muss zu Anfang ein Käfer durch legen eines Kleeblatts priorisiert werden:

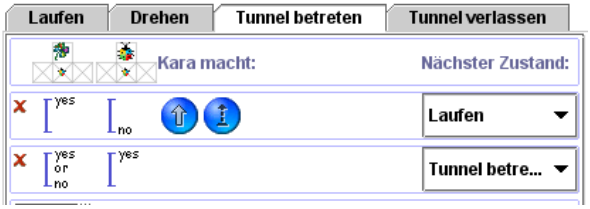


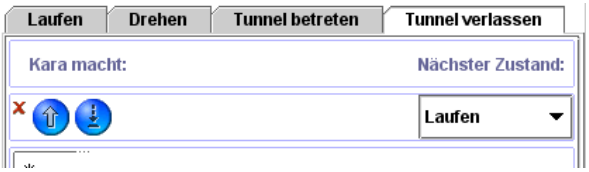
Dann kann man die Sache mit dem folgenden Automaten für beide Käfer lösen:



Die Zustände sind folgendermassen:

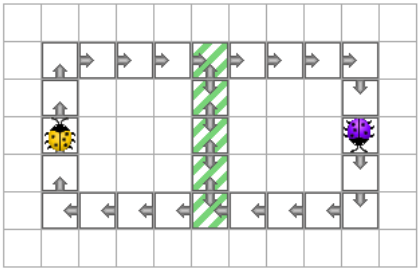




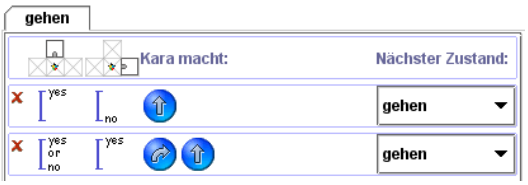


Lösung mit Monitor:

Die Welt muss im Tunnel einen Monitor haben:

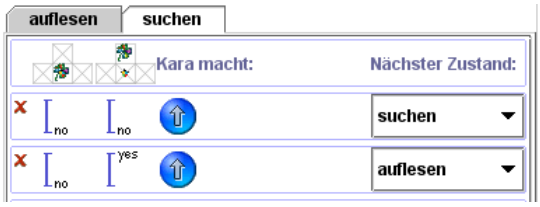


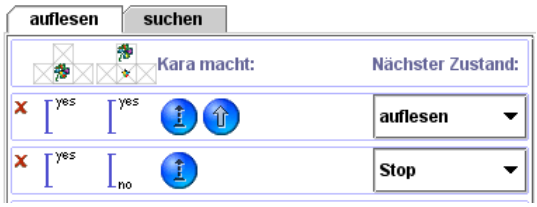
Lösung mit einem Zustand (gehen). Beide Karas haben denselben Automaten (d.h. dasselbe Programm).



Lösung zur Aufgabe 4.5

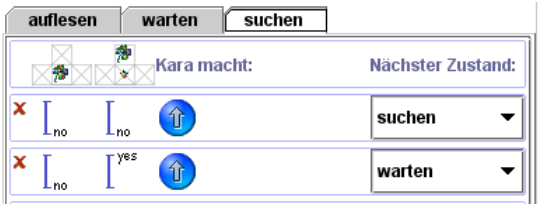
Lösung 4.5 c) mit zwei Zuständen: suchen und auflesen

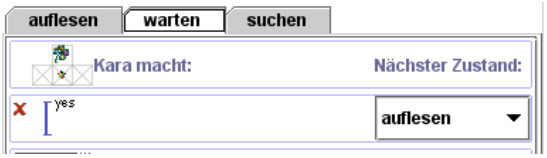


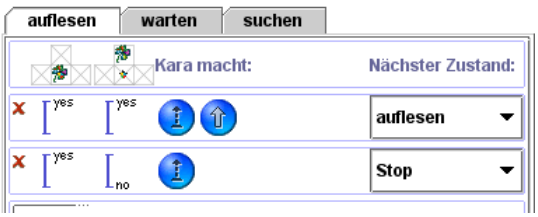


Lösung zur Aufgabe 4.6

Lösung mit drei Zuständen, wobei der mittlere Zustand "warten" nur als Barriere dient.







Lösung zur Aufgabe 4.7

Lösung mit zwei Zuständen (Stopp wird nicht benötigt), wobei der Zustand im Tunnel ein Critical Section-Zustand ist. Alle Karas haben denselben Automaten (vgl. Abb. 4.3)

Zustand: im Tunnel
(als Critical Section markiert)

Zustand: aussen
(Startzustand)