Hamster

Der Programmierwettbewerb zur Vorlesung „Algorithmen und Datenstrukturen“

Christian Borgelt

Institut für Wissens- und Sprachverarbeitung
Otto-von-Guericke-Universität Magdeburg
Universitätsplatz 2, D-39106 Magdeburg

23. April 1999

Inhaltsverzeichnis

1 Einleitung

Gut zu programmieren, das ist eine alte Weisheit (die sicher nicht nur für diese Fähigkeit gilt), lernt man nur, indem man es übt. Das geht natürlich am besten, wenn das Üben nicht einfacher Drill handwerklicher Technik ist, sondern dadurch geschieht, daß man vor zwar bewältigbare, aber auch nicht zu leichte Herausforderungen gestellt wird, bei denen man nach der Lösung selbst suchen muß. Dazu sind die einfachen Programmieraufgaben, die man im Laufe der Übungen zu einer Einführungsvorlesung der Informatik angeboten bekommt, nicht immer geeignet. Zwar mögen auch sie für diejenigen, die vorher noch keine Programmiererfahrung gesammelt haben, als Herausforderung erscheinen, da das „algorithmische Denken“ erst gelernt werden muß. Doch für diejenigen, die schon über Programmiererfahrung verfügen, später aber wohl für alle, sind sie wenig reizvoll, da der Lösungsweg meist schon in der Vorlesung behandelt wurde. Echte Herausforderungen stellen sie daher oft nicht mehr dar.

Aus dieser Überlegung heraus bieten wir diesen Programmierwettbewerb an. Das zu lösende Problem ist, so hoffen wir, leicht verständlich. Es ist so einfach, daß man auch mit geringen Kenntnissen und geringer Programmiererfahrung eine Lösung finden kann. Aber es ist auch so komplex, daß es eine immer weiter gehende Herausforderung darstellt, denn eine nachweisbar optimale Lösung ist für dieses Problem (soweit wir wissen) nicht bekannt. Man kann also seinen Ideen freien Lauf lassen. Gute Ideen sind bei diesem Problem sicher mehr wert als viel Erfahrung. Außerdem hoffen wir, daß die durch einen Wettbewerb geschaffene Konkurrenzsituation zusätzlich motivierend wirkt.

2 Das Hamsterproblem

Ein Hamster, der seinen Wintervorrat an Getreide noch nicht gesammelt hat, wird in einem Labyrinth ausgesetzt. In diesem Labyrinth ist eine bestimmte Anzahl von Maiskörnern in unterschiedlich großen Haufen verteilt Die Aufgabe des Hamsters ist es, möglichst viele Maiskörner zu sammeln und zu seinem Ausgangspunkt zu bringen. Dabei muß er die Sammelwege natürlich möglichst kurz halten, damit er bei seiner Tätigkeit nicht schon den größten Teil der gesammelten Maiskörner als Nahrung verbraucht.

In diesem Programmierwettbewerb soll ein Programm geschrieben werden, das den Hamster durch das Labyrinth steuert. Dazu steht eine überschaubare Menge von Funktionen zur Verfügung, mit denen dem Hamster bestimmte einfache Befehle gegeben werden können, z.B. „Drehe Dich um 90^o nach links!“, „Gehe ein Feld geradeaus!“, „Nimm drei Maiskörner auf!“ etc. (siehe Abschnitt 4). Die Reihenfolge und die Anzahl dieser Befehle wird durch das zu schreibende Steuerprogramm festgelegt. Hier hat der Programmierer völlig freie Hand.

Wie gut ein Programm die Aufgabe gelöst hat, wird anhand der Anzahl der gesammelten Maiskörner, der Länge des beim Sammeln zurückgelegten Weges und der Zahl der Zusammenstöße mit Wänden des Labyrinthes bestimmt (siehe Abschnitt 6.1). Die auf der Grundlage dieser Werte berechnete Punktzahl entscheidet schließlich über die Plazierung im Wettbewerb.

Eine genauere Beschreibung des Hamsters, seiner Fähigkeiten und des Labyrinthes, in dem er sich bewegt, findet sich in Abschnitt 4.

3 Das Hamsterprogramm

Das Hamsterprogramm besteht eigentlich aus zwei Programmen: Erstens dem im folgenden so genannten *Labyrinthprogramm*, das das Labyrinth verwaltet, in dem sich der Hamster befindet, und den Hamster entsprechend der gegebenen Befehle in diesem Labyrinth bewegt. Zweitens dem im folgenden so genannten *Steuerprogramm*, das dem Hamster Befehle erteilt. Nur das letztere Programm ist (teilweise) vom Wettbewerbsteilnehmer zu schreiben.

Das Labyrinthprogramm führt die vom Steuerprogramm an den Hamster gegebenen Befehle aus, stellt sicher, daß der Hamster nicht „durch Wände geht“, und berechnet die vom Hamster erreichte Punktzahl. In den Versionen dieses Programms, die eine graphische Benutzerschnittstelle besitzen, stellt dieses Programm außerdem die Bewegung des Hamsters im Labyrinth in einem Fenster auf dem Bildschirm dar.

Das Labyrinthprogramm übernimmt jedoch keinerlei Steueraufgaben. Diese liegen allein beim zu schreibenden Steuerprogramm. Es hat durch das Aufrufen von Hamsterfunktionen (d.h. das Erteilen von Befehlen an den Hamster) den Hamster zu bewegen und seine Sammeltätigkeit zu organisieren. Das Steuerprogramm hat keinen direkten Zugriff auf die Daten des Labyrinthprogramms. Es kann nur über die festgelegte Hamster-Befehlsschnittstelle mit dem Labyrinthprogramm kommunizieren.

3.1 Installation des Hamsterprogramms

Eine Installation des Hamsterprogramms ist nur notwendig, wenn nicht im HP-Pool der Universität Magdeburg gearbeitet wird. Dort ist das Programm bereits im Verzeichnis `~borgelt/src/hamster` installiert. Wird im HP-Pool gearbeitet, kann daher dieser Abschnitt übersprungen werden. Stattdessen wechselt man in das oben angegebene Verzeichnis und fährt mit dem in Abschnitt 3.2 beschriebenen Testlauf fort.

Zur Installation des Programms z.B. auf einem PC unter Linux oder Windows 95/98/NT geht man folgendermaßen vor: Das Entpacken der Dateien des Programmpaketes sollte ein Verzeichnis `hamster` erstellt haben (im folgenden Wurzelverzeichnis genannt), in dem sich die Verzeichnisse `doc`, `eiffel`, `hamster`, `mazes`, `pascal`, `unix` und `windows` befinden. Zur Installation des Programms ist in dieses Wurzelverzeichnis zu wechseln.

3.1.1 Installation unter Unix

Im Wurzelverzeichnis der Installation befindet sich die Datei `makefile`, die die Übersetzung des Hamsterprogramms unter Unix steuert. Zum Erstellen ist daher einfach `make all` einzugeben. Es wird daraufhin der Gnu-C-Compiler aufgerufen, um die Quelldateien zu übersetzen.

Gelingt die Übersetzung nicht, so liegt dies wahrscheinlich daran, daß Header-Dateien oder Bibliotheken des X-Window-Systems (X11R5/R6) nicht gefunden wurden. In der Datei `makefile` ist ein Standardpfad eingestellt, der oft (insbesondere für Linux) zutrifft: `/usr/X11R6`. Ist das X-Window-System unter einem anderen Pfad installiert (z.B. `/usr/X11R5`, `/usr/local/X11R6`, `/usr/X11`, `/usr/local/X11` o.ä.), so sind in der Datei `makefile` die Definitionen der Pfadkonstanten `X11INC` und `X11LIB` anzupassen. Ist die Programmierschnittstelle des X-Windows-Systems nicht installiert (kann z.B. bei der Linux-Standardinstallation der Fall sein: Serie `x`, Paket `xdevel`), so ist diese Schnittstelle zuerst zu installieren, da ohne sie eine Übersetzung unmöglich ist.

Steht der Gnu-C-Compiler (`gcc`) nicht zur Verfügung, kann auch mit dem Systemcompiler (`cc`) gearbeitet werden. In diesem Fall sind in der Datei `makefile` die Definitionen der Konstanten `CC` und `CFLAGS` anzupassen.

3.1.2 Installation unter Microsoft Windows

Im Wurzelverzeichnis der Installation befinden sich die Dateien `whamster.mdp` und `whamster.mak`, die die Übersetzung des Hamsterprogramms mit Microsoft C 4.0, und die Dateien `whamster.dsw`, `whamster.dsp` und `chamster.dsp`, die die Übersetzung des Hamsterprogramms mit Microsoft C 5.0 steuern. Die Datei `whamster.mdp` bzw. die Datei `whamster.dsw` ist aus der Entwicklungsumgebung des Microsoft-C-Systems über `File > Open Workspace...` zu öffnen. Beide Teilprojekte, `whamster` und `chamster`, müssen übersetzt werden.

Eine Übersetzung mit Borland C sollte ebenfalls möglich sein, doch liegt hierfür keine Projektdatei vor. Es sind zwei Programme zu erzeugen: `bin\whamster.exe` und `bin\chamster.exe`. Zum ersten gehören die Quelldateien `windows\whamster.c`, `windows\whamster.rc`, `windows\sprite.c`, `hamster\server.c` und `hamster\maze.c`, ggf. auch `windows\whamster.def`, zum zweiten die Quelldateien `hamster\control.c` und `hamster\client.c`. Als zusätzliche Include-Pfade sind im ersten Fall `hamster` und `windows`, im zweiten nur `hamster` anzugeben. Ersteres Programm ist als Standard-Windows-Anwendung, letzteres als Konsolenanwendung zu übersetzen.

3.2 Starten des Hamsterprogramms

Um das Hamsterprogramm zu starten, ruft man unter Unix das Programm `xhamster` auf, das sich nach der Übersetzung im Verzeichnis `bin` befindet. Alternativ (und einfacher) kann man das Shellscript `xh` im Wurzelverzeichnis der Hamsterinstallation starten. Unter Microsoft Windows startet man das Programm `whamster.exe`, das sich nach erfolgreicher Übersetzung ebenfalls im Verzeichnis `bin` befindet.

Es sollte ein Fenster mit dem Namen `xhamster` (Unix) bzw. `whamster` (Microsoft Windows) erscheinen, in dem ein zunächst leeres Labyrinth ohne Wände angezeigt wird (also eigentlich ein einfaches weißes Rechteck). In der unteren linken Ecke ist ein Feld grau unterlegt. Dies ist das Feld, auf dem der Hamster startet.

Da ein Testlauf mit einem völlig leeren Labyrinth nicht sehr interessant ist, sollte man das Labyrinth zunächst initialisieren. Dazu wählt man den Menüeintrag `Actions > Randomize Maze...` aus. Es erscheint eine Dialogbox, in der man die Wandwahrscheinlichkeit, die Gesamtzahl zu verteilender Maiskörner und die maximale Größe eines Maishaufens einstellen kann. Für einen ersten Test kann man die voreingestellten Werte akzeptieren und einfach `Ok` anwählen. Daraufhin wird das Labyrinth initialisiert. D.h. es werden zufällig Wände in einem quadratischen Raster gesetzt, wobei sichergestellt ist, daß jedes Feld des Labyrinthes erreichbar bleibt. Außerdem werden Kornhaufen verschiedener Größe zufällig im Labyrinth verteilt und als gelbe Kreise unterschiedlicher Größe dargestellt. Schließlich wird noch das Startfeld aus der unteren linken Ecke auf ein zufälliges Feld des Labyrinthes versetzt.

Auf dem initialisierten Labyrinth kann nun das Beispiel-Steuerprogramm gestartet werden. Dazu wählt man den Menüeintrag `Actions > Start Hamster` aus. Der Hamster erscheint als roter Kreis (in der Art eines „Pacman“) auf dem Startfeld und beginnt seine zufällige Wanderung durch das Labyrinth. Sollte statt des Hamsters eine Fehlermeldung erscheinen, daß das Programm `bin/chamster` oder `chamster` nicht gefunden/gestartet werden konnte, so muß zunächst das Hamsterprogramm ausgewählt werden. Dies geschieht über den Menüeintrag `File > Select Hamster...`. Es erscheint dann eine Dateiauswahlbox, in der das Programm `chamster` (Unix) bzw. `chamster.exe` aus dem Verzeichnis `bin` ausgewählt werden muß. Danach kann der Hamster (wie oben) über `Actions > Start Hamster` gestartet werden.

Das Beispiel-Steuerprogramm ist, wie schnell deutlich wird, nicht besonders intelligent. Der Hamster bewegt sich völlig orientierungslos durch das Labyrinth. Während dieser Wanderung wird in der Menüzeile (Unix) bzw. der Titelzeile (Microsoft Windows) des Labyrinthfensters die erreichte Punktzahl angezeigt. Nach 50 (Feld zu Feld) Schritten endet das Beispiel-Steuerprogramm und es wird eine Ergebnis-Dialogbox angezeigt, in der man ablesen kann, wie viele Maiskörner der Hamster eingesammelt, wie viele Züge er gemacht, wie oft er mit Wänden zusammengestoßen und welche Punktzahl er folglich erreicht hat.

3.3 Die Funktionen des Labyrinthprogramms

Die Funktionen des Labyrinthprogramms werden in den folgenden Abschnitten 3.3.1 bis 3.3.3 anhand der Menüstruktur erläutert. Danach folgt in Abschnitt 3.3.4 eine Beschreibung, wie man mit Hilfe der Maus ein Labyrinth manuell erzeugen oder ändern kann.

3.3.1 Das Menü „File“

Im Menü `File` finden sich Einträge zum Laden und Speichern von Labyrinthen und zum Auswählen eines Hamster-Steuerprogramms. Außerdem enthält dieses Menü, dem Standard entsprechend, einen Eintrag, mit dem das Programm verlassen werden kann.

`Reload Maze...` Das zuletzt mit `Load Maze...` geladene oder mit `Save Maze As...` gespeicherte Labyrinth wird erneut geladen. So können leicht mehrere Testläufe auf dem gleichen Labyrinth ausgeführt werden. Wurde noch kein Labyrinth geladen oder gespeichert, so wird eine Dialogbox angezeigt, in der eine Datei ausgewählt werden kann, aus der ein Labyrinth geladen werden soll.

Dieser Menüeintrag kann nicht angewählt werden, während ein Hamster-Steuerprogramm läuft.

`Load Maze...` Es wird eine Dialogbox angezeigt, in der eine Datei ausgewählt werden kann, aus der ein Labyrinth geladen werden soll. Die zu ladende Labyrinthdatei sollte über den Menüpunkt `Save Maze...` erzeugt worden sein. Das im Labyrinthfenster angezeigte Labyrinth wird nur ersetzt, wenn das Laden erfolgreich verlief. Anderenfalls wird eine Fehlermeldung ausgegeben.

Dieser Menüeintrag kann nicht angewählt werden, während ein Hamster-Steuerprogramm läuft.

`Save Maze...` Das aktuelle Labyrinth wird in der Datei gespeichert, die beim letzten Aufruf einer der Menüpunkte `Load Maze...` oder `Save Maze As...` ausgewählt wurde. Wurde noch kein Labyrinth geladen oder gespeichert, so wird eine Dialogbox angezeigt, in der eine Datei ausgewählt werden kann, in der das aktuelle Labyrinth gespeichert werden soll.

Dieser Menüeintrag kann nicht angewählt werden, während ein Hamster-Steuerprogramm läuft.

`Save Maze As...` Es wird eine Dialogbox angezeigt, in der eine Datei ausgewählt werden kann, in der das im Labyrinthfenster angezeigte Labyrinth gespeichert werden soll. Das Labyrinth wird genau in der Form abgespeichert, in der es angezeigt wird, nicht in der Form, in der es geladen oder erzeugt wurde. Daher sollte man, wenn man ein Hamster-Steuerprogramm mehr als einmal auf dem gleichen Labyrinth testen möchte, dieses Labyrinth *vor* dem ersten Testlauf abspeichern.

Dieser Menüeintrag kann nicht angewählt werden, während ein Hamster-Steuerprogramm läuft.

`Select Hamster...` Es wird eine Dialogbox angezeigt, in der ein Hamster-Steuerprogramm ausgewählt werden kann. Indem Hamster-Steuerprogramme vom Labyrinthprogramm aus ausgewählt werden können, kann man (nacheinander) mehrere Hamsterprogramme testen, ohne das Labyrinthprogramm verlassen zu müssen.

Dieser Menüpunkt kann nicht angewählt werden, während ein Hamster-Steuerprogramm läuft.

`Quit` Durch Anwahl dieses Menüpunktes wird das Labyrinthprogramm verlassen. Dieser Menüpunkt kann auch angewählt werden, während ein Hamster-Steuerprogramm läuft. In diesem Fall wird sowohl das Labyrinthprogramm als auch das Steuerprogramm beendet. Anders als beim Stoppen eines Hamsters über `Actions > Stop Hamster` wird keine Dialogbox mit dem bisher erzielten Ergebnis angezeigt.

3.3.2 Das Menü „Actions“

Im Menü `Actions` finden sich Einträge zum Initialisieren und Verändern des Labyrinthes, zum Einstellen der Bewegungsgeschwindigkeit des Hamsters, sowie zum Starten und Anhalten des Hamsters.

`Redraw` Das Labyrinthfenster wird neu gezeichnet. Dieser Menüpunkt ist wahrscheinlich nur unter Windows notwendig, da dort, wenn bei laufendem

Hamster ein Fenster verschoben wird, „Geisterbilder“ des Hamsters zurückbleiben können. Durch Auswählen dieses Menüpunktes werden diese gelöscht.

Dieser Menüeintrag kann auch angewählt werden, während ein Hamster-Steuerprogramm läuft.

`Set Maze Size...` Mit diesem Menüeintrag kann die Größe des Labyrinthes verändert werden. In der bei Anwahl dieses Menüeintrages erscheinenden Dialogbox können die x-Ausdehnung (horizontale/West-Ost-Ausdehnung) und die y-Ausdehnung (vertikale/Süd-Nord-Ausdehnung) in Feldeinheiten eingetragen werden. Das Labyrinth muß mindestens 8 Felder breit (wegen der Menüzeile des Fensters) und mindestens 1 Feld hoch sein. Die maximale Ausdehnung beträgt 64 Felder in jede Richtung.

Dieser Menüeintrag kann nicht ausgewählt werden, solange ein Hamster-Steuerprogramm läuft.

`Randomize Maze...` Das Labyrinth wird (zufällig) initialisiert. In der bei Anwahl dieses Menüeintrages erscheinenden Dialogbox können die Wandwahrscheinlichkeit, die Anzahl insgesamt zu verteilender Maiskörner und die maximale Größe eines Kornhaufens eingestellt werden. Nach Drücken des „Ok“-Knopfes wird das Labyrinth entsprechend initialisiert. Außerdem wird das Startfeld auf ein zufälliges Feld des Labyrinthes versetzt.

Dieser Menüeintrag kann nicht ausgewählt werden, solange ein Hamster-Steuerprogramm läuft.

`Clear Maze` Das Labyrinth wird gelöscht, d.h. alle Wände und alle Maiskörner werden entfernt und das Startfeld wird in die linke untere Ecke gesetzt.

Dieser Menüeintrag kann nicht ausgewählt werden, solange ein Hamster-Steuerprogramm läuft.

`Set Hamster Speed...` In der bei Anwahl dieses Menüeintrages erscheinenden Dialogbox kann die gewünschte Geschwindigkeit des Hamsters in Labyrinthfeldern pro Sekunde eingetragen werden. Der kleinste Wert dieser Geschwindigkeit beträgt 0.1, der größte 100 Felder pro Sekunde. Bei 100 Feldern pro Sekunde wird die Darstellung des Hamsters ausgeschaltet. In diesem Fall wird lediglich bei Ablauf des Hamster-Steuerprogramms die Endsituation des Labyrinthes (Maiskornverteilung) angezeigt. Wie der Hamster den Mais einsammelt, kann dagegen nicht verfolgt werden.

Dieser Menüeintrag kann auch angewählt werden, während ein Steuerprogramm läuft. Die geänderte Hamstergeschwindigkeit wird wirksam, wenn der Hamster seinen aktuellen Bewegungsschritt abgeschlossen (seine Drehung beendet oder sein Zielfeld erreicht) hat.

`Start/Stop Hamster` Mit diesem Menüeintrag kann ein Hamster-Steuerprogramm gestartet oder beendet werden (je nachdem, ob gerade ein Hamster-Steuerprogramm läuft oder nicht). Gestartet wird das Steuerprogramm, das vorher über den Menüeintrag `File > Select Hamster` ausgewählt wurde. Bei Start des Labyrinthprogramms ist das Beispielprogramm `bin/chamster` (Unix, abgestimmt auf einen Aufruf des Labyrinthprogramms über das Shellscript `xh`) bzw. `chamster.exe` (Microsoft Windows) ausgewählt.

Wird ein Hamster-Steuerprogramm über diesen Menüeintrag beendet, so wird eine Ergebnisdialogbox mit den bis zum Zeitpunkt des Abbruchs gesammelten Maiskörnern, durchgeführten Schritten, Anzahl Wandzusammenstößen und folglich erreichten Punkten angezeigt.

3.3.3 Das Menü „Help“

Unter diesem Menüeintrag findet sich leider keine echte Hilfe zur Benutzung des Hamsterprogramms, sondern nur ein Eintrag, der bei Anwahl zur Anzeige einer Dialogbox („About“-Dialogbox) führt, in der allgemeine Informationen zum Hamsterprogramm ausgegeben werden.

3.3.4 Labyrinthkonstruktion mit der Maus

Solange kein Hamster-Steuerprogramm läuft, läßt sich das im Labyrinthfenster angezeigte Labyrinth mit Hilfe der Maus verändern. Folgende Funktionen stehen zur Verfügung (die Zeichen ■□□, □■□ und □□■ bedeuten ein Drücken der linken, mittleren bzw. rechten Maustaste):

- ■□□ Setzen/Entfernen einer Wand. Es wird an dem Rand eines Labyrinthfeldes, der dem Mauszeiger am nächsten liegt, eine Wand gesetzt (falls sich dort keine Wand befindet) oder entfernt (falls sich dort eine Wand befindet).
- Ctrl ■□□ Alle Wände, die das Labyrinthfeld umgeben, auf das der Mauszeiger zeigt, werden entfernt.
- Shift ■□□ Das Startfeld, d.i. das Feld, auf dem der Hamster im Labyrinth ausgesetzt wird, wird auf das Labyrinthfeld versetzt, auf das der Mauszeiger zeigt.
- □■□ Die Größe des Maiskornhaufens auf dem Labyrinthfeld, auf das der Mauszeiger zeigt, wird um ein Maiskorn verringert.
- Shift □■□ Die Größe des Maiskornhaufens auf dem Labyrinthfeld, auf das der Mauszeiger zeigt, wird um fünf Maiskörner verringert.
- Ctrl □■□ Alle Maiskörner, die auf dem Labyrinthfeld liegen, auf das der Mauszeiger zeigt, werden entfernt.
- □□■ Die Größe des Maiskornhaufens auf dem Labyrinthfeld, auf das der Mauszeiger zeigt, wird um ein Maiskorn erhöht.
- Shift □□■ Die Größe des Maiskornhaufens auf dem Labyrinthfeld, auf das der Mauszeiger zeigt, wird um fünf Maiskörner erhöht.
- Ctrl □□■ Alle Maiskörner, die auf dem Labyrinthfeld liegen, auf das der Mauszeiger zeigt, werden entfernt.

Mit Hilfe dieser Funktionen und des Menüeintrags `Actions > Set Maze Size...` lassen sich offenbar beliebige Labyrinthe konstruieren. Es wird nicht geprüft, ob durch die mit der Maus gesetzten gesetzten Wände einige Felder das Labyrinthes vom Startfeld aus unerreichbar werden. Daher lassen sich auch Labyrinthe mit nicht rechteckigem Grundriß erzeugen, nämlich einfach dadurch, daß man durch Wände bestimmte Bereiche unzugänglich macht.

4 Die Steuerung des Hamsters

Dieser Abschnitt enthält eine genauere Beschreibung des Labyrinthes, des Hamsters, der Befehle, die ihm erteilt werden können, und wie er sich in dem Labyrinth, in dem er sich befindet, orientieren kann. Abschnitt 4.1 enthält allgemeine Erläuterungen zur Steuerung des Hamsters. Die Abschnitte 4.2 und 4.3 sind programmiersprachenabhängig. Ersterer beschreibt die Programmierschnittstelle für die Programmiersprache C, letzterer die Schnittstelle für die Programmiersprache Eiffel. Eine Programmierschnittstelle für Pascal wird zur Zeit noch entwickelt.

4.1 Allgemeine Erläuterungen

Der Hamster wird in einem Labyrinth mit rechteckigem Grundriß ausgesetzt. Dieses Labyrinth hat ein quadratisches Raster, d.h. das Rechteck, das den Grundriß bildet, ist in quadratische Felder eingeteilt. Das Labyrinth kann in x-Richtung (West-Ost-Richtung) zwischen 8 und 64 Feldern, in y-Richtung (Süd-Nord-Richtung) zwischen 1 und 64 Feldern groß sein. Auf den Rändern dieser Felder können Wände stehen und auf den Feldern können Haufen von Maiskörnern liegen, die zwischen 0 Körnern (kein Maiskornhaufen) und 255 Körnern enthalten können.

Ein solches Labyrinth kann automatisch (und dann zufällig) über den Menüeintrag `Actions > Randomize Maze...` (siehe Abschnitt 3.3.2) oder manuell mit der Maus (siehe Abschnitt 3.3.4) erzeugt werden. Erzeugt wird ein Labyrinth, indem Wände gesetzt und Maiskörner verteilt werden.

Ein Feld des Labyrinthes ist ausgezeichnet — im Labyrinthfenster an der grauen Farbe erkennbar. Auf diesem Feld wird der Hamster in das Labyrinth gesetzt. Auf dieses Feld muß der Hamster auch die von ihm gesammelten Maiskörner ablegen. Erst wenn sie auf diesem Feld abgelegt sind, gelten sie als für den Wintervorrat gesammelt und werden in der Punktzahl verbucht.

Der Hamster kann sich im Labyrinth bewegen, indem er in Blickrichtung auf das nächste Feld tritt (siehe Funktion `hms_move` bzw. Methode `move`) und indem er sich um 90^o im oder gegen den Uhrzeigersinn dreht (siehe Funktion `hms_turn` bzw. Methode `turn`). Damit man den Hamster nicht vor Labyrinthwände steuert, kann man abfragen, was der Hamster vor sich sieht (siehe Funktion `hms_look` bzw. Methode `get_look`).

Der Hamster kann sich im Labyrinth orientieren, indem er seine Positionskoordinaten abfragt (siehe Funktion `hms_pos` bzw. Methoden `get_x` und `get_y`). Diese Koordinaten werden stets relativ zum Ausgangsfeld angegeben, das die Koordinaten $(0, 0)$ hat. Die x-Koordinate ist westlich (links) vom Ausgangsfeld negativ und östlich (rechts) positiv. Die y-Koordinate ist südlich (unterhalb) des Ausgangsfeldes negativ und nördlich (oberhalb) positiv. Die Blickrichtung des Hamsters kann ebenfalls ermittelt werden (siehe Funktion `hms_dir` bzw. Methode `get_dir`).

Der Hamster kann auch die Anzahl Maiskörner zählen, die auf dem Feld liegen, auf dem er sich gerade befindet (siehe Funktion `hms_corn` bzw. Methode `get_corn`). Er kann von diesem Feld Maiskörner aufnehmen (siehe Funktion `hms_take` bzw. Methode `take`) oder auf dieses Feld Maiskörner ablegen (siehe Funktion `hms_drop` bzw. Methode `drop`).

Der Hamster kann nicht beliebig viele Maiskörner tragen. Seine Backen haben eine begrenzte Kapazität: nur zwölf Maiskörner passen hinein. Wie viele Körner der Hamster gerade trägt, kann abgefragt werden (siehe Funktion `hms_load` bzw. Methode `get_load`) Versucht der Hamster, mehr als zwölf Körner aufzunehmen, werden die überzähligen Körner liegengelassen (siehe Funktion `hms_take` bzw. Methode `take`). Gleiches gilt, wenn er versucht, auf einem Feld des Labyrinthes mehr als 255 Maiskörner anzuhäufen (siehe Funktion `hms_drop` bzw. Methode `drop`). Die nicht mehr auf das Feld passenden Körner verbleiben in seinen Backen. Diese Einschränkung gilt jedoch nicht für das Ausgangsfeld. Auf diesem Feld können beliebig viele Körner abgelegt werden. Es ist immer leer, da die hier abgelegten Körner direkt in den Wintervorrat aufgenommen werden.

Die zur Steuerung des Hamsters benötigten Funktionen sind in der Datei `hamster.h` bzw. in der Datei `hamster.e` definiert. Diese Dateien enthalten außerdem Definitionen von Konstanten, die z.B. für die Auswertung der Rückgaben einiger Funktionen bzw. Methoden nützlich sein können. Im folgenden werden diese Definitionen programmiersprachenabhängig im einzelnen besprochen.

4.2 Programmierschnittstelle für C

Die Programmierschnittstelle für die Programmiersprache C ist durch die Header-Datei `hamster.h` definiert. Diese Datei muß mit `#include "hamster.h"` in die Datei eingebunden werden, die das (zu schreibende) Hamster-Steuerprogramm enthält.

4.2.1 Typdefinitionen

In der Datei `hamster.h` ist der Typ `HAMSTER` definiert, durch den ein Hamster dargestellt wird.

```
typedef struct hamster HAMSTER; /* a hamster */
```

Die genaue Definition (welche Felder die Hamsterstruktur enthält) ist jedoch verborgen. Erstens, weil die sich die Definition der Hamsterstruktur für das Labyrinth- und für das Steuerprogramm unterscheiden, und zweitens um jeder Versuchung, auf Felder der Hamsterstruktur direkt zuzugreifen, von vornherein dadurch vorzubeugen, daß diese Felder gar nicht bekanntgegeben werden.

4.2.2 Konstanten

In der Headerdatei `hamster.h` sind folgende Konstanten definiert, die Eigenschaften des Hamsters und des Labyrinthes festlegen sowie Rückgabewerte und Parameter von Hamsterfunktionen leichter verständlich machen.

Labyrinthausdehnung

```
#define HMS_MAXXEXT   64        /* maximal x-extension of maze */
#define HMS_MAXYEXT   64        /* maximal y-extension of maze */
```

Das Labyrinth kann höchstens `HMS_MAXXEXT` × `HMS_MAXYEXT` Felder groß sein.

Maishaufengröße

```
#define HMS_MAXCORN  255        /* maximal size of corn heap */
```

Die Größe eines Maiskornhaufens auf einem Labyrinthfeld kann höchstens `HMS_MAXCORN` Maiskörner betragen. Dies gilt jedoch nur für „normale“ Labyrinthfelder. Auf dem Startfeld können beliebig viele Körner abgelegt werden.

Tragkapazität des Hamsters

```
#define HMS_MAXLOAD   12        /* maximal load of corn in cheeks */
```

Der Hamster kann höchstens `HMS_MAXLOAD` Maiskörner gleichzeitig tragen.

Ergebnisse der Funktion `hms_dir`

```
#define HMS_EAST       0        /* hamster is looking east */
#define HMS_NORTH      1        /* hamster is looking north */
#define HMS_WEST       2        /* hamster is looking west */
#define HMS_SOUTH      3        /* hamster is looking south */
```

Die Blickrichtung des Hamsters wird als ganze Zahl kodiert. Namen von Himmelsrichtungen sind jedoch sicherlich aussagekräftiger als Zahlen.

Ergebnisse der Funktion `hms_look`

```
#define HMS_EMPTY      0        /* there is an empty field ahead */
#define HMS_CORN       1        /* there is a field with corn ahead */
#define HMS_WALL       2        /* there is a wall ahead */
```

Was der Hamster sieht, wird als ganze Zahl kodiert. Namen wie `HMS_EMPTY` (das in Blickrichtung voraus liegende Feld ist leer), `HMS_CORN` (auf dem in Blickrichtung voraus liegenden Feld liegt Mais) und `HMS_WALL` (voraus liegt eine Wand) sind jedoch sicherlich aussagekräftiger als Zahlen.

Parameter der Funktion `hms_turn`

```
#define HMS_POS        1        /* positive turn (counterclockwise) */
#define HMS_NEG       -1        /* negative turn (clockwise) */
#define HMS_LEFT       1        /* left     turn (counterclockwise) */
#define HMS_RIGHT     -1        /* right    turn (clockwise) */
```

Der Funktion `hms_turn` muß als Parameter die Richtung angegeben werden, in die sich der Hamster drehen soll. Dies geschieht durch eine positive oder negative Zahl. Auch hier sind Namen wie `HMS_POS` (Drehen im mathematisch positiven Sinn), `HMS_NEG` (Drehen im mathematisch negativen Sinn), `HMS_LEFT` (Drehen von der Blickrichtung nach links) und `HMS_RIGHT` (Drehen von der Blickrichtung nach rechts) aussagekräftiger als Zahlen. Die Werte `HMS_POS` und `HMS_LEFT` entsprechen (bei Blick von oben) einem Drehen gegen den Uhrzeigersinn, die Werte `HMS_NEG` und `HMS_RIGHT` einem Drehen im Uhrzeigersinn.

4.2.3 Auskunftsfunktionen

Mit den in diesem Abschnitt beschriebenen Funktionen können Informationen über den Hamster und die Lage, in der er sich befindet, geholt werden.

`hms_pos`

Name	"get current hamster position" — Bestimmen der aktuellen Hamsterposition
Definition	`void hms_pos (HAMSTER *hms, int *x, int *y);`
Beschreibung	Es wird die Labyrinthposition des Hamsters `hms` ermittelt und die Positionskoordinaten werden in den Variablen abgelegt, auf die `x` und `y` zeigen. Die Koordinaten wachsen von West nach Ost und von Süd nach Nord. Sie beziehen sich *nicht* auf die linke untere Ecke des Labyrinthes, sondern auf das Startfeld des Hamsters. Die Koordinaten können also auch negativ werden, nämlich dann, wenn sich der Hamster westlich und/oder südlich des Startfeldes befindet. Das Startfeld selbst hat die Koordinaten $(0, 0)$.
Ergebnis	Die Funktion hat kein Ergebnis.

`hms_dir`

Name	"get current hamster direction" — Bestimmen der aktuellen Hamsterblickrichtung
Definition	`int hms_dir (HAMSTER *hms);`
Beschreibung	Es wird die Blickrichtung des Hamsters `hms` ermittelt und die Himmelsrichtung zurückgegeben, und zwar kodiert als ganze Zahl. Für diese Zahlen sind in der Headerdatei `hamster.h` symbolische Konstanten definiert, und zwar: `HMS_EAST` Der Hamster `hms` schaut nach Osten. `HMS_NORTH` Der Hamster `hms` schaut nach Norden. `HMS_WEST` Der Hamster `hms` schaut nach Westen. `HMS_SOUTH` Der Hamster `hms` schaut nach Süden.
Ergebnis	Einer der Werte `HMS_EAST`, `HMS_NORTH`, `HMS_WEST` oder `HMS_SOUTH`.

`hms_load`

Name	"get current hamster load" — Bestimmen der aktuellen Maiskornlast des Hamsters
Definition	`int hms_load (HAMSTER *hms);`
Beschreibung	Es wird die Anzahl der Maiskörner ermittelt, die der Hamster `hms` gerade trägt. Diese Zahl liegt zwischen 0 (einschließlich) und der in der Headerdatei `hamster.h` definierten Zahl `HMS_MAXLOAD` (einschließlich).
Ergebnis	Die Zahl der getragenen Maiskörner.

`hms_corn`

Name	"get amount of corn on current field" — Bestimmen der Kornmenge auf dem aktuellen Feld

Definition	`int hms_corn (HAMSTER *hms);`
Beschreibung	Es wird die Anzahl der Maiskörner bestimmt, die auf dem Feld liegen, auf dem sich der Hamster `hms` gerade befindet.
Ergebnis	Die Zahl der Maiskörner auf dem aktuellen Feld.

hms_look

Name	"get hamster's current outlook" — Bestimmen des aktuellen Ausblicks des Hamsters
Definition	`int hms_look (HAMSTER *hms);`
Beschreibung	Es wird der Ausblick des Hamsters `hms` bestimmt und kodiert als ganze Zahl zurückgegeben. Für diese Zahlen sind in der Headerdatei `hamster.h` symbolische Konstanten definiert, und zwar: `HMS_EMPTY` Vor dem Hamster liegt ein leeres Feld. `HMS_CORN` Vor dem Hamster liegt ein Feld, auf dem sich Mais befindet. `HMS_WALL` Vor dem Hamster steht eine Wand. Bei Rückgabe des Wertes `HMS_CORN` erfährt man nur, daß auf dem voraus liegenden Feld Mais liegt, nicht jedoch (da der Hamster kurzsichtig ist), wieviel dort liegt. Um die Anzahl der Maiskörner auf dem Feld zu bestimmen, muß der Hamster es betreten. Dann kann die Zahl der Maiskörner mit Hilfe der Funktion `hms_corn` ermittelt werden.
Ergebnis	Einer der Werte `HMS_EMPTY`, `HMS_CORN` oder `HMS_WALL`.

4.2.4 Handlungsfunktionen

Mit den in diesem Abschnitt beschriebenen Funktionen können Handlungen des Hamster ausgelöst werden.

hms_move

Name	"move hamster forward" — Vorwärtsbewegen des Hamsters
Definition	`int hms_move (HAMSTER *hms);`
Beschreibung	Der Hamster `hms` wird in seiner aktuellen Blickrichtung um ein Feld vorwärts bewegt. Dies gelingt natürlich nur, wenn sich vor dem Hamster keine Wand befindet.
Ergebnis	Falls sich vor dem Hamster keine Wand befindet und er folglich vorwärts bewegt werden kann, ist das Ergebnis 0. Anderenfalls wird ein Wert ungleich 0 zurückgegeben.

hms_turn

Name	"turn hamster" — Drehen des Hamsters
Definition	`void hms_turn (HAMSTER *hms, int turn);`
Beschreibung	Der Hamster `hms` wird um 90^o in die Richtung gedreht, die der Parameter `turn` angibt. Für diesen Parameter sind in der Headerdatei `hamster.h` folgende symbolische Konstanten definiert: `HMS_POS` Drehen im mathematisch positiven Sinn. `HMS_NEG` Drehen im mathematisch negativen Sinn. `HMS_LEFT` Drehen von der Blickrichtung aus nach links. `HMS_RIGHT` Drehen von der Blickrichtung aus nach rechts. Die Werte `HMS_POS` und `HMS_LEFT` entsprechen (bei Blick von oben) einem Dre-

	hen gegen den Uhrzeigersinn, die Werte `HMS_NEG` und `HMS_RIGHT` einem Drehen im Uhrzeigersinn.
Ergebnis	Die Funktion hat kein Ergebnis.

`hms_take`

Name	"take some corn" — Aufnehmen einiger Maiskörner
Definition	`int hms_take (HAMSTER *hms, int amount);`
Beschreibung	Der Hamster `hms` nimmt so viele Maiskörner auf, wie der Parameter `amount` angibt. Dazu müssen natürlich auf dem aktuellen Feld genügend viele Maiskörner liegen, sonst werden entsprechend weniger Maiskörner aufgenommen. Außerdem muß der Hamster noch mindestens `amount` Maiskörner aufnehmen können. Ist die aktuelle Maiskornlast des Hamsters vermehrt um `amount` größer als die in der Datei `hamster.h` definierte Konstante `HMS_MAXLOAD`, so werden ebenfalls entsprechend weniger Körner aufgenommen.
Ergebnis	Die Zahl der tatsächlich aufgenommenen Maiskörner.

`hms_drop`

Name	"drop some corn" — Ablegen einiger Maiskörner
Definition	`int hms_drop (HAMSTER *hms, int amount);`
Beschreibung	Der Hamster `hms` legt so viele Maiskörner ab, wie der Parameter `amount` angibt. Dazu muß der Hamster natürlich mindestens `amount` Maiskörner tragen. Trägt er weniger, werden auch entsprechend weniger Maiskörner abgelegt. Außerdem darf durch die abgelegten Körner der Maiskornhaufen auf dem aktuellen Feld nicht größer als `HMS_MAXCORN` Maiskörner werden (`HMS_MAXCORN` ist in der Headerdatei `hamster.h` definiert). Anderenfalls werden entsprechend weniger Körner abgelegt. Letztere Regelung gilt nicht für das Startfeld. Auf diesem können beliebig viele Maiskörner abgelegt werden.
Ergebnis	Die Zahl der tatsächlich abgelegten Maiskörner.

4.2.5 Die Steuerfunktion

Die in diesem Abschnitt beschriebene Funktion ist nicht vorgegeben, sondern vom Wettbewerbsteilnehmer zu schreiben. Sie soll den Hamster durch das Labyrinth steuern.

`hms_ctrl`

Name	"control hamster" — Steuern des Hamsters
Definition	`void hms_ctrl (HAMSTER *hms);`
Beschreibung	Diese Funktion soll den Hamster steuern. Sie ist vom Wettbewerbsteilnehmer zu schreiben. Sie wird aufgerufen, wenn der Hamster auf dem Startfeld im Labyrinth ausgesetzt wurde und übernimmt damit die Stelle der Funktion `main` in einem normalen C-Programm. Ein Zeiger auf den zu steuernden Hamster wird im Funktionsparameter `hms` übergeben. Er ist den in den beiden vorhergehenden Abschnitten beschriebenen Funktionen zu übergeben, wenn Informationen über den Hamster oder das Feld, auf dem er sich befindet, geholt werden sollen, oder wenn der Hamster eine Handlung ausführen soll. Die Funktion sollte erst dann beendet werden, wenn so viele Maiskörner gesammelt wurden wie möglich.
Ergebnis	Die Funktion hat kein Ergebnis.

4.3 Programmierschnittstelle für Eiffel

Die Programmierschnittstelle für die Programmiersprache Eiffel besteht aus der Datei `hamster.e`, die die Klasse `HAMSTER` definiert, und der Datei `client.e`, die eine Basis-Steuerklasse zur Verfügung stellt. Um ein eigenes Hamster-Steuerprogramm zu schreiben, muß letztere Klasse beerbt und die Methode `control` redefiniert werden. Aus dieser Methode können dann die Methoden der Hamsterklasse `HAMSTER` aufgerufen werden, um den Hamster zu steuern.

4.3.1 Konstanten

In der Datei `hamster.e` sind folgende Konstanten definiert, die Eigenschaften des Hamsters und des Labyrinthes festlegen, sowie Rückgabewerte und Parameter von Hamstermethoden leichter verständlich machen.

Labyrinthausdehnung

```
max_x:      INTEGER is  64;     -- maximal x-extension of maze
max_y:      INTEGER is  64;     -- maximal y-extension of maze
```

Das Labyrinth kann höchstens `max_x` × `max_y` Felder groß sein.

Maishaufengröße

```
max_corn:   INTEGER is 255;     -- maximal size of corn heap
```

Die Größe eines Maiskornhaufens auf einem Labyrinthfeld kann höchstens `max_corn` Maiskörner betragen. Dies gilt jedoch nur für „normale“ Labyrinthfelder. Auf dem Startfeld können beliebig viele Körner abgelegt werden.

Tragkapazität des Hamsters

```
max_load:   INTEGER is  12;     -- maximal load of corn in cheeks
```

Der Hamster kann höchstens `max_load` Maiskörner gleichzeitig tragen.

Ergebnisse der Methode `get_dir`

```
east:       INTEGER is   0;     -- hamster is looking east
north:      INTEGER is   1;     -- hamster is looking north
west:       INTEGER is   2;     -- hamster is looking west
south:      INTEGER is   3;     -- hamster is looking south
```

Die Blickrichtung des Hamsters wird als ganze Zahl kodiert. Namen von Himmelsrichtungen sind jedoch sicherlich aussagekräftiger als Zahlen.

Ergebnisse der Methode `get_look`

```
empty:      INTEGER is   0;     -- there is an empty field ahead
corn:       INTEGER is   1;     -- there is a field with corn ahead
wall:       INTEGER is   2;     -- there is a wall ahead
```

Der Ausblick des Hamsters wird als ganze Zahl kodiert. Namen wie `empty` (das in Blickrichtung voraus liegende Feld ist leer), `corn` (auf dem in Blickrichtung voraus liegenden Feld liegt Mais) und `wall` (voraus liegt eine Wand) sind jedoch sicherlich aussagekräftiger als Zahlen.

Parameter der Methode `turn`

```
pos:        INTEGER is   1;     -- positive turn (counterclockwise)
neg:        INTEGER is  -1;     -- negative turn (clockwise)
left:       INTEGER is   1;     -- left     turn (counterclockwise)
right:      INTEGER is  -1;     -- right    turn (clockwise)
```

Der Methode `turn` muß als Parameter die Richtung angegeben werden, in die sich der Hamster drehen soll. Dies geschieht durch eine positive oder negative Zahl. Auch hier sind Namen wie `pos` (Drehen im mathematisch positiven Sinn), `neg` (Drehen im mathematisch negativen Sinn), `left` (Drehen von der Blickrichtung nach links) und `right` (Drehen von der Blickrichtung nach rechts) aussagekräftiger als Zahlen. Die Werte `pos` und `left` entsprechen (bei Blick von oben) einem Drehen gegen den Uhrzeigersinn, die Werte `neg` und `right` einem Drehen im Uhrzeigersinn.

4.3.2 Auskunftsfunktionen

Mit den in diesem Abschnitt beschriebenen Methoden können Informationen über den Hamster und die Lage, in der er sich befindet, geholt werden.

get_x

Name	"get current hamster x-position" — Bestimmen der aktuellen x-Position des Hamsters
Definition	`get_x: INTEGER`
Beschreibung	Es wird die x-Position des Hamsters im Labyrinth relativ zum Startfeld bestimmt.
Ergebnis	Die x-Position des Hamsters relativ zum Startfeld.

get_y

Name	"get current hamster y-position" — Bestimmen der aktuellen y-Position des Hamsters
Definition	`get_y: INTEGER`
Beschreibung	Es wird die y-Position des Hamsters im Labyrinth relativ zum Startfeld bestimmt.
Ergebnis	Die y-Position des Hamsters relativ zum Startfeld.

`get_x` und `get_y` liefern die Labyrinthposition des Hamsters. Die Koordinaten wachsen von West nach Ost und von Süd nach Nord. Sie beziehen sich *nicht* auf die linke untere Ecke des Labyrinthes, sondern auf das Startfeld des Hamsters. Die Koordinaten können also auch negativ werden, nämlich dann, wenn sich der Hamster westlich und/oder südlich des Startfeldes befindet. Das Startfeld selbst hat die Koordinaten $(0, 0)$.

get_dir

Name	"get current hamster direction" — Bestimmen der aktuellen Hamsterblickrichtung
Definition	`get_dir: INTEGER`
Beschreibung	Es wird die Blickrichtung des Hamsters ermittelt und die Himmelsrichtung zurückgegeben, und zwar kodiert als ganze Zahl. Für diese Zahlen sind in der Datei `hamster.e` symbolische Konstanten definiert, und zwar: `east` Der Hamster schaut nach Osten. `north` Der Hamster schaut nach Norden. `west` Der Hamster schaut nach Westen. `south` Der Hamster schaut nach Süden.
Ergebnis	Einer der Werte `east`, `north`, `west` oder `south`.

`get_load`

Name	"get current hamster load" — Bestimmen der aktuellen Maiskornlast des Hamsters
Definition	`get_load: INTEGER`
Beschreibung	Es wird die Anzahl der Maiskörner ermittelt, die der Hamster gerade trägt. Diese Zahl liegt zwischen 0 (einschließlich) und der in der Datei `hamster.e` definierten Zahl `max_load` (einschließlich).
Ergebnis	Die Zahl der getragenen Maiskörner.

`get_corn`

Name	"get amount of corn on current field" — Bestimmen der Kornmenge auf dem aktuellen Feld
Definition	`get_corn: INTEGER`
Beschreibung	Es wird die Anzahl der Maiskörner bestimmt, die auf dem Feld liegen, auf dem sich der Hamster gerade befindet.
Ergebnis	Die Zahl der Maiskörner auf dem aktuellen Feld.

`get_look`

Name	"get hamster's current outlook" — Bestimmen des aktuellen Ausblicks des Hamsters
Definition	`get_look: INTEGER`
Beschreibung	Es wird der Ausblick des Hamsters bestimmt und kodiert als ganze Zahl zurückgegeben. Für diese Zahlen sind in der Datei `hamster.e` symbolische Konstanten definiert, und zwar: `empty` Vor dem Hamster liegt ein leeres Feld. `corn` Vor dem Hamster liegt ein Feld, auf dem sich Mais befindet. `wall` Vor dem Hamster steht eine Wand. Bei Rückgabe des Wertes `corn` erfährt man nur, daß auf dem voraus liegenden Feld Mais liegt, nicht jedoch (da der Hamster kurzsichtig ist), wieviel dort liegt. Um die Anzahl der Maiskörner auf dem Feld zu bestimmen, muß der Hamster es betreten. Dann kann die Zahl der Maiskörner mit Hilfe der Funktion `get_corn` ermittelt werden.
Ergebnis	Einer der Werte `empty`, `corn` oder `wall`.

4.3.3 Handlungsfunktionen

Mit den in diesem Abschnitt beschriebenen Methoden können Handlungen des Hamster ausgelöst werden.

`move`

Name	"move hamster forward" — Vorwärtsbewegen des Hamsters
Definition	`move: INTEGER`
Beschreibung	Der Hamster wird in seiner aktuellen Blickrichtung um ein Feld vorwärts bewegt. Dies gelingt natürlich nur, wenn sich vor dem Hamster keine Wand befindet.
Ergebnis	Falls sich vor dem Hamster keine Wand befindet und er folglich vorwärts bewegt werden kann, ist das Ergebnis 0. Anderenfalls wird -1 zurückgegeben.

turn

Name	"turn hamster" — Drehen des Hamsters
Definition	`turn (t: INTEGER)`
Beschreibung	Der Hamster wird um 90^o in die Richtung gedreht, die der Parameter `t` angibt. Für diesen Parameter sind in der Datei `hamster.e` folgende symbolische Konstanten definiert: `pos` Drehen im mathematisch positiven Sinn. `neg` Drehen im mathematisch negativen Sinn. `left` Drehen von der Blickrichtung aus nach links. `right` Drehen von der Blickrichtung aus nach rechts. Die Werte `pos` und `left` entsprechen (bei Blick von oben) einem Drehen gegen den Uhrzeigersinn, die Werte `neg` und `right` einem Drehen im Uhrzeigersinn.

take

Name	"take some corn" — Aufnehmen einiger Maiskörner
Definition	`take (amount: INTEGER)`
Beschreibung	Der Hamster nimmt so viele Maiskörner auf, wie der Parameter `amount` angibt. Dazu müssen natürlich auf dem aktuellen Feld genügend viele Maiskörner liegen, sonst werden entsprechend weniger Maiskörner aufgenommen. Außerdem muß der Hamster noch mindestens `amount` Maiskörner aufnehmen können. Ist die aktuelle Maiskornlast des Hamsters vermehrt um `amount` größer als die in der Datei `hamster.e` definierte Konstante `max_load`, so werden ebenfalls entsprechend weniger Körner aufgenommen.
Ergebnis	Die Zahl der tatsächlich aufgenommenen Maiskörner.

drop

Name	"drop some corn" — Ablegen einiger Maiskörner
Definition	`drop (amount: INTEGER): INTEGER`
Beschreibung	Der Hamster legt so viele Maiskörner ab, wie der Parameter `amount` angibt. Dazu muß der Hamster natürlich mindestens `amount` Maiskörner tragen. Trägt er weniger, werden auch entsprechend weniger Maiskörner abgelegt. Außerdem darf durch die abgelegten Körner der Maiskornhaufen auf dem aktuellen Feld nicht größer als `max_corn` Maiskörner werden (`max_corn` ist in der Datei `hamster.e` definiert). Anderenfalls werden entsprechend weniger Körner abgelegt. Letztere Regelung gilt nicht für das Startfeld. Auf diesem können beliebig viele Maiskörner abgelegt werden.
Ergebnis	Die Zahl der tatsächlich abgelegten Maiskörner.

4.3.4 Die Steuerfunktion

Die in diesem Abschnitt beschriebene Methode, die nicht zur Hamsterklasse sondern zur Steuerklasse gehört, ist nur in einer trivialen Form zur Funktionsprüfung (Drehen des Hamsters auf dem Ausgangsfeld) vorgegeben. Sie ist vom Wettbewerbsteilnehmer zu schreiben und soll den Hamster durch das Labyrinth steuern.

control

Name	"control hamster" — Steuern des Hamsters
Definition	`control (hamster: HAMSTER)`

Beschreibung Diese Funktion soll den Hamster steuern. Sie ist vom Wettbewerbsteilnehmer zu schreiben. Sie wird aufgerufen (von der Methode `make` der Basis-Steuerklasse `CLIENT`), wenn der Hamster auf dem Startfeld im Labyrinth ausgesetzt wurde. Der zu steuernde Hamster `hamster` ist eine Instanz der Klasse `HAMSTER` und verfügt demzufolge über alle in den beiden vorhergehenden Abschnitten beschriebenen Methoden. So bedeutet z.B. `hamster.turn(hamster.left)`, daß sich der Hamster um 90^o nach links drehen soll. Diese Methode sollte erst dann beendet werden, wenn so viele Maiskörner gesammelt wurden wie möglich.

4.4 Programmierschnittstelle für Pascal

Die Programmierschnittstelle für Pascal ist zur Zeit noch in der Entwicklung und wird daher hier noch nicht beschrieben. Sie wird der Programmierschnittstelle für C analog sein.

5 Wie man ein Hamster-Steuerprogramm schreibt

5.1 Das Beispielprogramm

In diesem Abschnitt wird beschrieben, wie man ein eigenes Hamster-Steuerprogramm schreibt, übersetzt und auf einem Labyrinth ausführen läßt.

Dem Hamster-Programmpaket ist ein Beispielprogramm beigefügt, das zeigt, wie ein Hamster-Steuerprogramm aussehen kann. Natürlich ist dieses Programm keine gute Lösung des gestellten Problems (sondern sogar eine sehr schlechte). Aber dieses Programm soll ja auch nur deutlich machen, wie man ein Hamster-Steuerprogramm schreibt, und nicht schon die Lösung mitliefern. Um ein eigenes Steuerprogramm zu schreiben, geht man am besten von diesem Beispielprogramm aus, das für die Programmiersprache C in Abschnitt 5.1.1 und für die Programmiersprache Eiffel in Abschnitt 5.1.2 beschrieben ist. Die Übersetzung eines eigenen Programms wird in Abschnitt 5.2 gezeigt. Abschnitt 5.3 schließlich erläutert, wie man das übersetzte Programm im Labyrinthprogramm startet.

5.1.1 Das Beispielprogramm in C

Das Beispielprogramm, das in Abbildung 1 gezeigt ist, steuert den Hamster auf einem mehr oder minder zufälligen Weg fünfzig Züge lang durch das Labyrinth. Während dieser Wanderschaft werden so viele Maiskörner gesammelt wie möglich. Wenn zufällig das Ausgangsfeld wieder erreicht wird, werden die gesammelten Maiskörner abgelegt.

Das Beispielprogramm besteht aus einer `while`-Schleife, in der bei jedem Durchlauf ein (Feld zu Feld) Zug des Hamsters ausgeführt wird. Der mitgeführte Zugzähler, der in jedem Schleifendurchlauf erhöht wird, stellt sicher, daß genau 50 Züge ausgeführt werden.

Am Anfang der Schleife wird die Position des Hamsters im Labyrinth durch einen Aufruf der Funktion `hms_pos` ermittelt. Die Parameter `x` und `y` müssen mit dem Adressoperator `&` versehen sein, da die Funktion `hms_pos` sich nicht für den Inhalt der Variablen `x` und `y` interessiert, sondern wissen muß, an welche Stellen sie die Positionskoordinaten zu schreiben hat (vgl. die bekannte Funktion `scanf`).

Anschließend wird geprüft, ob beide Koordinaten den Wert 0 haben. Die Position $(0,0)$ ist, siehe Abschnitt 4, das Startfeld. Auf diesem sollen alle gesammelten Maiskörner abgelegt werden. Folglich wird versucht, durch einen Aufruf der Funktion `hms_drop` so viele Maiskörner abzulegen, wie der Hamster höchstens tragen kann. Falls er weniger trägt, werden ja (siehe die Beschreibung der Funktion `hms_drop`) entsprechend weniger abgelegt. Nach diesem Aufruf sind die Backen des Hamsters auf jeden Fall leer.

Falls sich der Hamster nicht auf dem Startfeld befindet, wird durch einen Aufruf der Funktion `hms_corn` geprüft, ob sich auf dem Feld Maiskörner befinden. Hier kann auf eine genaue Abfrage der Zahl verzichtet werden, da nur interessiert, ob sich auf dem Feld Mais befindet oder nicht. Nun

```
/*----------------------------------------------------------------------
  File    : control.c
  Contents: hamster control program
  Author  : Christian Borgelt
  History : 16.10.1997 file created
----------------------------------------------------------------------*/
#include <stdio.h>
#include <stdlib.h>
#include "hamster.h"

/*----------------------------------------------------------------------
  Functions
----------------------------------------------------------------------*/

void hms_ctrl (HAMSTER *hms)
{                               /* --- hamster control function */
  int x, y;                     /* coordinates of current field */
  int movecnt = 0;              /* number of moves made */
  int turn;                     /* turn direction (HMS_POS/HMS_NEG) */

  while (movecnt < 50) {        /* make 50 (field to field) moves */
    hms_pos(hms, &x, &y);       /* get current position in maze */
    if ((x == 0) && (y == 0))   /* if we are at home (initial pos.), */
      hms_drop(hms, HMS_MAXLOAD);    /* drop all the corn we have */
    else if (hms_corn(hms))     /* if there is corn on the field, */
      hms_take(hms, HMS_MAXLOAD);    /* take as much as we can carry */
    if ((hms_look(hms) == HMS_WALL)  /* if the way is blocked */
    ||  (RAND_MAX/4 -rand() > 0)) {  /* or just because it's funny */
      turn = (RAND_MAX/2 -rand() > 0) ? HMS_POS : HMS_NEG;
      do { hms_turn(hms, turn); /* turn hamster in random direction */
      } while (hms_look(hms) == HMS_WALL);
    }                           /* until no wall blocks the way */
    hms_move(hms); movecnt++;   /* move the hamster forward */
  }                             /* and count the move made */
}  /* hms_ctrl() */
```

Abbildung 1: Ein Beispiel-Steuerprogramm in C, das dem Programmpaket beigefügt ist. Es steuert den Hamster für 50 (Feld zu Feld) Züge auf zufällige Weise durch das Labyrinth. Während seiner Wanderung nimmt der Hamster so viele Maiskörner auf wie er kann und legt sie ab, wenn er zufällig wieder das Ausgangsfeld erreicht.

wird die Zahl 0 in C als falsch, jede andere Zahl als wahr gewertet. Folglich kann das Ergebnis der Funktion `hms_corn` einfach als Wahrheitswert interpretiert werden.

Befindet sich Mais auf dem Feld, wird durch einen Aufruf der Funktion `hms_take` versucht, so viele Maiskörner aufzunehmen, wie der Hamster höchstens tragen kann. Falls seine Backen bereits (teilweise) voll sind, so werden die überzähligen Körner einfach liegengelassen. Wenn sich weniger Maiskörner auf dem Feld befinden als der Hamster noch tragen könnte, werden nur diese aufgenomen (siehe die Beschreibung der Funktion `hms_take`).

Nachdem Korn aufgenommen oder abgelegt wurde, wird die Richtung bestimmt, in die sich der Hamster bewegen soll. Dazu wird zunächst durch einen Aufruf der Funktion `hms_look` geprüft, ob der Hamster vor einer Wand steht. Ist dies der Fall, muß er auf jeden Fall seine Richtung wechseln. Aber auch wenn der Weg frei ist, wird über einen Zufallsmechanismus mit der Wahrscheinlichkeit

1/4 ein Richtungswechsel ausgelöst. Dieses zufällige Auslösen geschieht mit Hilfe der Bibliotheksfunktion `rand` (definiert in `stdlib.h`). Diese Funktion liefert bei jedem Aufruf einen Wert zwischen 0 (einschließlich) und `RAND_MAX` (einschließlich, ebenfalls in `stdlib.h` definiert). Die zurückgelieferten Werte sind über diesen Zahlenbereich statistisch gleichverteilt. Mit Wahrscheinlichkeit 1/4 sind sie daher kleiner als `RAND_MAX/4`.

Muß (oder soll) ein Richtungswechsel durchgeführt werden, so wird zunächst eine Drehrichtung festgelegt, und zwar mit gleicher Wahrscheinlichkeit eine mathematisch positive (gegen den Uhrzeigersinn) oder negative Drehung (im Uhrzeigersinn). Dann wird der Hamster in einer Schleife durch Aufrufe der Funktion `hms_turn` so lange in dieser Richtung gedreht, bis sich vor ihm keine Wand (mehr) befindet, was wieder durch Aufrufe der Funktion `hms_look` festgestellt wird.

Durch den Test in der vorhergehenden `if`-Anweisung oder durch die Drehschleife ist sichergestellt, daß sich der Hamster nicht vor einer Wand befindet. Er kann sich folglich auf das vor ihm liegende Feld bewegen, was durch einen Aufruf der Funktion `hms_move` erreicht wird. Dieser Zug wird im Zähler `movecnt` gezählt.

Ein eigenes Hamsterprogramm schreibt man vielleicht am besten, indem man das oben beschriebene Beispielprogramm kopiert und dann Änderungen an dieser Kopie vornimmt. Man könnte z.B. zunächst versuchen, eine andere Abbruchbedingung für die Sammeltätigkeit des Hamsters zu finden, so daß er auch mehr als nur 50 Züge machen kann.

5.1.2 Das Beispielprogramm in Eiffel

Das Beispielprogramm, das in Abbildung 2 gezeigt ist, steuert den Hamster auf einem mehr oder minder zufälligen Weg fünfzig Züge lang durch das Labyrinth. Während dieser Wanderschaft werden so viele Maiskörner gesammelt wie möglich. Wenn zufällig das Ausgangsfeld wieder erreicht wird, werden die gesammelten Maiskörner abgelegt. Die Steuerklasse `CONTROL`, aus der dieses Programm besteht, beerbt die Basis-Steuerklasse `CLIENT`, die bereits das Erzeugen des Hamsters (Methode `make`) enthält. Die Steuermethode `control` dieser Basis-Steuerklasse wird redefiniert.

Die Beispiel-Steuermethode besteht aus einer Schleife, in der bei jedem Durchlauf ein (Feld zu Feld) Zug des Hamsters ausgeführt wird. Der mitgeführte Zugzähler, der in jedem Schleifendurchlauf erhöht wird, stellt sicher, daß genau 50 Züge ausgeführt werden.

Am Anfang der Schleife wird die Position des Hamsters im Labyrinth durch Aufrufe der Hamstermethoden `get_x` und `get_y` ermittelt. Es wird geprüft, ob beide Koordinaten den Wert 0 haben. Die Position $(0, 0)$ ist, siehe Abschnitt 4, das Startfeld. Auf diesem sollen alle gesammelten Maiskörner abgelegt werden. Folglich wird versucht, durch einen Aufruf der Hamstermethode `drop` so viele Maiskörner abzulegen, wie der Hamster höchstens tragen kann. Falls er weniger trägt, werden ja (siehe die Beschreibung der Methode `drop`) entsprechend weniger abgelegt. Nach diesem Aufruf sind die Backen des Hamsters auf jeden Fall leer.

Falls sich der Hamster nicht auf dem Startfeld befindet, wird durch einen Aufruf der Hamstermethode `get_corn` geprüft, ob sich auf dem Feld Maiskörner befinden (genau: ob die Zahl der Maiskörner auf dem Feld größer als 0 ist). Befindet sich Mais auf dem Feld, wird durch einen Aufruf der Hamstermethode `take` versucht, so viele Maiskörner aufzunehmen, wie der Hamster höchstens tragen kann. Falls seine Backen bereits (teilweise) voll sind, so werden die überzähligen Körner einfach liegengelassen. Wenn sich weniger Maiskörner auf dem Feld befinden als der Hamster noch tragen könnte, werden nur diese aufgenomen (siehe die Beschreibung der Methode `take`).

Nachdem Korn aufgenommen oder abgelegt wurde, wird die Richtung bestimmt, in die sich der Hamster bewegen soll. Dazu wird zunächst durch einen Aufruf der Hamstermethode `get_look` geprüft, ob der Hamster vor einer Wand steht. Ist dies der Fall, muß er auf jeden Fall seine Richtung wechseln. Aber auch wenn der Weg frei ist, wird über einen Zufallsmechanismus mit der Wahrscheinlichkeit 1/4 ein Richtungswechsel ausgelöst. Dieses zufällige Auslösen geschieht mit Hilfe der Methode `random` (definiert in der Klasse `CLIENT`). Diese Funktion liefert bei jedem Aufruf einen Wert zwischen 1 (einschließlich) und dem übergebenen Parameter (einschließlich). Die zurückgelieferten Werte sind über diesen Zahlenbereich statistisch gleichverteilt. Hat der Parameter — wie im vorliegenden Fall — den Wert 4, so sind sie mit Wahrscheinlichkeit 1/4 kleiner-gleich 1.

```
----------------------------------------------------------------------
-- File    : control.e
-- Contents: customized hamster control class
-- Authors : Ilona Bl"umel and Christian Borgelt
-- History : 11.03.1999 file created
----------------------------------------------------------------------
class CONTROL
inherit CLIENT
  redefine control
end
creation
  make
feature
control (hamster: HAMSTER) is   -- hamster control function (random)
local
  movecnt: INTEGER;             -- number of moves made
  turn   : INTEGER;             -- turn direction (left/right)
  d      : INTEGER;             -- dummy variable for results
do
  from
    movecnt := 0;               -- initialize the move counter
  until
    movecnt = 50                -- make 50 field to field moves
  loop
    if     hamster.get_x = 0    -- if we are at home (initial position),
    and    hamster.get_y = 0    -- drop all the corn we have
    then   d := hamster.drop(hamster.max_load);
    elseif hamster.get_corn > 0 -- if there is corn on the field
    then   d := hamster.take(hamster.max_load);
    end                         -- take as much as he can carry
    if hamster.get_look = hamster.wall  -- if the way is blocked
    or else random(4) <= 1      -- or just because it's funny (25%)
    then if random(2) <= 1      -- randomly choose a turn direction
         then turn := hamster.pos;
         else turn := hamster.neg;
         end                    -- (50% either way)
         from
           hamster.turn(turn);  -- turn the hamster
         until                  -- in the chosen direction
           hamster.get_look /= hamster.wall
         loop                   -- repeat turning the hamster around
           hamster.turn(turn);  -- until no wall blocks the way
         end
    end
    d := hamster.move;          -- move the hamster forward
    movecnt := movecnt +1;      -- and count the move made
  end
end  -- control
end  -- CONTROL
```

Abbildung 2: Ein Beispiel-Steuerprogramm in Eiffel, das dem Programmpaket beigefügt ist. Es steuert den Hamster für 50 (Feld zu Feld) Züge auf zufällige Weise durch das Labyrinth. Während seiner Wanderung nimmt der Hamster so viele Maiskörner auf wie er kann und legt sie ab, wenn er zufällig wieder das Ausgangsfeld erreicht.

Muß (oder soll) ein Richtungswechsel durchgeführt werden, so wird zunächst eine Drehrichtung festgelegt, und zwar mit gleicher Wahrscheinlichkeit eine mathematisch positive (gegen den Uhrzeigersinn) oder negative Drehung (im Uhrzeigersinn). Dann wird der Hamster in einer Schleife durch Aufrufe der Methode `turn` so lange in dieser Richtung gedreht, bis sich vor ihm keine Wand (mehr) befindet, was wieder durch Aufrufe der Hamstermethode `get_look` festgestellt wird.

Durch den Test in der vorhergehenden `if`-Anweisung oder durch die Drehschleife ist sichergestellt, daß sich der Hamster nicht vor einer Wand befindet. Er kann sich folglich auf das vor ihm liegende Feld bewegen, was durch einen Aufruf der Hamstermethode `move` erreicht wird. Dieser Zug wird im Zähler `movecnt` gezählt.

Ein eigenes Hamsterprogramm schreibt man vielleicht am besten, indem man das oben beschriebene Beispielprogramm kopiert und dann Änderungen an dieser Kopie vornimmt. Man könnte z.B. zunächst versuchen, eine andere Abbruchbedingung für die Sammeltätigkeit des Hamsters zu finden, so daß er auch mehr als nur 50 Züge machen kann.

5.2 Übersetzung eines eigenen Steuerprogramms

Hat man ein eigenes Hamsterprogramm geschrieben, so muß es übersetzt werden. Zur Übersetzung benötigt man außer der Datei (oder den Dateien), die dieses Programm enthalten, noch weitere, in denen die Kommunikation mit dem Labyrinthprogramm implementiert ist.

5.2.1 Übersetzung eines eigenen Steuerprogramms in C

Zur Übersetzung eines Steuerprogramms in der Programmiersprache C wird noch die im Verzeichnis `hamster` enthaltene Datei `client.c` benötigt. Diese Datei enthält die Implementierung der in Abschnitt 4.2 beschriebenen Hamsterfunktionen. Sie setzen die Funktionsaufrufe in Nachrichten um, die an das Labyrinthprogramm geschickt werden.

Als Beispiel nehmen wir an, daß sich das eigene Hamster-Steuerprogramm unter dem Namen `mycontrol.c` im Verzeichnis `hamster` befindet (in dem auch die Datei `client.c` liegt). Ist dieses Verzeichnis das aktuelle Arbeitsverzeichnis, so kann der Hamster unter Unix z.B. durch einen Aufruf des Gnu-C-Compilers

```
gcc -Wall -ansi -pedantic mycontrol.c client.c -o myhamster
```

übersetzt werden. Um im HP-Pool der Universität Magdeburg das eigene Hamster-Steuerprogramm in seinem eigenen Verzeichnis halten, muß man die Dateien `hamster.h` und `client.c` aus dem Verzeichnis `~borgelt/src/hamster/` in das eigene Verzeichnis kopieren oder (besser) einen Verweis („link“) auf diese Dateien setzen (z.B. `ln -s ~borgelt/src/hamster/hamster.h hamster.h`).

Unter Microsoft C ist ein Projekt (oder genauer: „project workspace“) anzulegen, in das die Dateien `myhamster.c` und `client.c` eingefügt werden müssen. Ggf. ist das Verzeichnis `hamster` als zusätzlicher Include-Pfad anzugeben. Ähnlich geht man unter Borland C vor. Auch hier muß ein Projekt angelegt werden, in das die beiden angegebenen Dateien einzufügen sind. Dieses Projekt wird dann, sowohl unter Microsoft C als auch unter Borland C, durch Auswählen des entsprechenden Menüeintrags übersetzt. Das Programm muß als Konsolenanwendung übersetzt werden.

5.2.2 Übersetzung eines eigenen Steuerprogramms in Eiffel

Zur Übersetzung eines Steuerprogramms in der Programmiersprache Eiffel werden noch die im Verzeichnis `eiffel` enthaltenen Dateien `hamster.e` und `client.e` benötigt. Erstere Datei enthält die Implementierung der in Abschnitt 4.3 beschriebenen Hamstermethoden. Sie setzen die Funktionsaufrufe in Nachrichten um, die an das Labyrinthprogramm geschickt werden. Letztere Datei enthält eine Basisklasse für ein Hamstersteuerobjekt, die von dem Hamster-Steuerprogramm beerbt werden sollte. Sie stellt die Methode `make` zum Erzeugen eines Hamsters (diese Methode sollte `nicht` überladen werden) und die Methode `random` zum Erzeugen von (Pseudo-)Zufallszahlen zur Verfügung. Außerdem definiert diese Klasse die Methode `control`, die den Hamster steuern soll. Diese Methode ist durch die eigene Steuermethode zu überladen.

Als Beispiel nehmen wir an, daß sich das eigene Hamster-Steuerprogramm (die eigene Hamster-Steuerklasse) unter dem Namen `mycontrol.e` im Verzeichnis `eiffel` befindet (in dem auch die Dateien `hamster.e` und `client.e` liegen). Ist dieses Verzeichnis das aktuelle Arbeitsverzeichnis, so kann der Hamster unter Unix z.B. durch einen Aufruf des SmallEiffel-Compilers

```
compile mycontrol.e -o myhamster
```

übersetzt werden. (Unter S.u.S.E. Linux ist `/usr/lib/SmallEiffel/bin/compile` der SmallEiffel-Compiler. Man muß das Paket `smeiffel` aus der Serie `d` installieren.) Um im HP-Pool der Universität Magdeburg das eigene Hamster-Steuerprogramm in seinem eigenen Verzeichnis halten, muß man die Dateien `hamster.e` und `client.e` aus dem Verzeichnis `~borgelt/src/eiffel/` in das eigene Verzeichnis kopieren oder (besser) einen Verweis („link“) auf diese Dateien setzen (z.B. durch `ln -s ~borgelt/src/hamster/hamster.e hamster.e`).

5.3 Ausführen des Programms auf einem Labyrinth

Zum Ausführen des Hamster-Steuerprogramms muß zunächst das Hamster-Labyrinthprogramm gestartet werden. Dies geschieht wie in Abschnitt 3.2 beschrieben. Unter Unix ist entweder das Programm `xhamster` im Verzeichnis `bin` oder das Shellscript `xh` im Wurzelverzeichnis der Installation zu starten. Unter Microsoft Windows ist das Programm `whamster.exe` im Verzeichnis `bin` aufzurufen. Danach muß das übersetzte Hamster-Steuerprogramm (z.B. `myhamster` oder `myhamster.exe`) über die bei Anwahl des Menüeintrags `File > Select Hamster...` erscheinende Dateiauswahlbox ausgewählt werden. Anschließend kann der Hamster auf einem (ggf. vorher über den Menüpunkt `Actions > Randomize Maze...` zu initialisierenden) Labyrinth über den Menüeintrag `Actions > Start Hamster` gestartet werden.

5.4 Testausgaben aus einem Steuerprogramm

Beim Testen eines eigenen Programms, nicht nur eines Hamster-Steuerprogramms, ist es hilfreich, wenn man während des Programmlaufs Variablenwerte ausgeben kann. Dies ist bei einem Hamster-Steuerprogramm in C nicht wie sonst mit z.B. der Funktion `printf` und bei einem Hamster-Steuerprogramm in Eiffel nicht wie sonst mit den Funktionen `io.put_string`, `io.put_integer` etc. möglich, da die Tastatureingabe und Bildschirmausgabe für die Kommunikation mit dem Labyrinthprogramm verwendet werden. Dennoch kann man solche Ausgaben machen. In C muß statt `printf(s)`, wobei `s` die auszugebende Zeichenkette ist, `fprintf(stderr, s)` benutzt werden. In Eiffel ist z.B. `io.put_string(s)` durch `std_error.put_string` zu ersetzen. Diese Möglichkeit besteht sowohl unter Unix als auch unter Microsoft Windows.

6 Der Wettbewerb

In diesem Abschnitt wird zunächst erläutert, wie die Punktzahl berechnet wird, die zur Bewertung eines Programms auf einem gegebenen Labyrinth dient (Abschnitt 6.1). Diese Punktbewertung ist das einzige Kriterium für das Abschneiden eines Programmes in diesem Wettbewerb. In Abschnitt 6.2 wird der Ablauf des Wettbewerbs beschrieben.

6.1 Programmbewertung

Wie schon in Abschnitt 2 angedeutet, wird ein Hamster-Steuerprogramm anhand der Anzahl der vom Hamster gesammelten Maiskörner, der Länge des beim Sammeln zurückgelegten Weges und der Zahl der Zusammenstöße mit Wänden des Labyrinthes bewertet. Aus diesen Daten wird auf folgende Weise eine Punktzahl berechnet:

gesammeltes Maiskorn:	+20 Punkte
Schritt von Feld zu Feld:	−1 Punkt
Zusammenstoß mit Wand:	−20 Punkte
Programmabsturz:	−1000 Punkte

Ein Maiskorn gilt erst dann als gesammelt, wenn es auf dem Ausgangsfeld, d.h. dem Feld von dem der Hamster gestartet ist, abgelegt wurde. Noch vom Hamster getragene und nicht abgelegte Körner werden nicht gezählt.

Als Schritt von Feld zu Feld zählt ein erfolgreicher Aufruf der Funktion `hms_move`. Durch den Abzug von einem Punkt soll die Nahrung simuliert werden, die der Hamster für seine Fortbewegung braucht. Der Aufruf der Funktion `hms_move` ist erfolgreich, wenn sich vor dem Hamster keine Wand des Labyrinthes befindet. Zusammenstöße mit den Labyrinthwänden können daher immer vermieden werden, da sich mit der Funktion `hms_look` abfragen läßt, ob sich vor dem Hamster eine Wand befindet.

Ein erfolgloser Aufruf der Funktion `hms_move` (der Aufruf ist erfolglos, wenn sich vor dem Hamster eine Labyrinthwand befindet, so daß er sich nicht nach vorn bewegen kann), gilt als Zusammenstoß mit einer Wand.

Ein Aufruf der Funktion `hms_turn`, die den Hamster ja nur dreht, aber auf dem gleichen Feld beläßt, gilt *nicht* als Schritt. Dies soll simulieren, daß das Umherschauen, zu dem ja das Drehen oft dient, wesentlich weniger Nahrung verbraucht als das Umherwandern.

Schließlich müssen noch (hoffentlich nicht vorkommende) Abstürze oder Endlosschleifen des Steuerprogramms berücksichtigt werden. Diese werden, da es sich um schwere Fehler handelt, mit entsprechend hohem Punktabzug geahndet. Um außerdem zu vermeiden, daß Programme durch langwierige Wegsuchalgorithmen die Wettbewerbsauswertung erschweren bzw. unmöglich machen, wird eine Obergrenze für die Rechenzeit festgesetzt, die ein Hamster-Steuerprogramm je Labyrinth verbrauchen darf (Details siehe unten). Wird diese Obergrenze überschritten, wird das als Programmabsturz gewertet, das Programm abgebrochen, und die bis zum Abbruchzeitpunkt erreichte Punktzahl verwendet.

Die von einem Programm auf einem Labyrinth erzielte Punktzahl berechnet sich folglich als

$$p = k \cdot 20 - s - z \cdot 20 - a \cdot 1000,$$

wobei k die Anzahl der gesammelten und zum Ausgangsfeld gebrachten Maiskörner, s die Anzahl der vom Hamster ausgeführten Feld zu Feld Schritte, z die Anzahl der Zusammenstöße mit Labyrinthwänden ist und a den Wert 1 oder 0 hat, je nachdem, ob das Steuerprogramm abstürzte oder nicht. Die nach obiger Formel berechnete Punktzahl wird über alle Wettbewerbslabyrinthe summiert. Die so erhaltene Bewertung eines Hamsterprogramms bestimmt allein die Plazierung im Wettbewerb.

Bemerkungen

Zur Obergrenze der Rechenzeit: Als die von einem Hamster-Steuerprogramm für ein Labyrinth verbrauchte Rechenzeit wird die Zeit definiert, die die *Kommandozeilenversion* `hamster` bzw. `hamster.exe` des Labyrinthprogramms braucht, um das Hamsterprogramm auf dem Labyrinth auszuführen (unter Unix bestimmbar mit dem Befehl `time`). Die Fensterversion des Labyrinthprogramms ist gegenüber der Kommandozeilenversion *sehr* verlangsamt, damit man die Bewegungen des Hamster verfolgen kann. Eine Abschätzung der Rechenzeit kann man mit der Fensterversion erhalten, wenn man die Hamstergeschwindigkeit auf 100 Felder/Sekunde einstellt. Dadurch wird die graphische Darstellung abgeschaltet und die Ausführung ist etwa so schnell wie mit der Kommandozeilenversion.

Die Obergrenze der Rechenzeit beträgt 10 Minuten je Labyrinth. Referenzrechner ist eine Sun Sparc Ultra 2 168 MHz (vergleichbar einem Intel Pentium 200 MHz). Dieser Rechner is deutlich schneller als die HP-Workstations im HP-Pool. Wenn ein Hamsterprogramm die Obergrenze der Rechenzeit im HP-Pool einhält, ist es also auf der sicheren Seite.

Zu negativen Punktzahlen: Die Anfangspunktzahl des Hamster ist, wie ja auch aus der obigen Formel abzulesen, Null. Da er mindestens auf ein dem Ausgangsfeld benachbartes Feld gehen muß, um Maiskörner zu sammeln, jedoch Nahrung braucht, um von Feld zu Feld zu gehen, wird die Punktzahl des Programms zunächst negativ. Erst wenn der Hamster Maiskörner gesammelt und zum Ausgangsfeld gebracht hat, wird sie positiv. Es gibt keine Untergrenze für die Punktzahl. Mit anderen Worten: Der Hamster kann nicht „verhungern". Erreicht ein Hamster auf einem Labyrinth eine negative Punktzahl, so geht diese Zahl aber auch als negative Punktzahl in die Bewertung ein. Sie wird *nicht* als 0 gewertet.

6.2 Durchführung des Wettbewerbs

Am diesährigen Hamster-Programmierwettbewerb (1999) können alle Studenten teilnehmen, die im Wintersemester 1998/1999 und im Sommersemester 1999 die Vorlesung „Algorithmen und Datenstrukturen", gehalten von PD Dr. Maritta Heisel an der Universität Magdeburg, hören. Wer an dem Wettbewerb teilnehmen möchte, meldet sich bitte (am besten per E-mail) entweder bei Ilona Blümel oder bei Christian Borgelt (E-mail-Adressen: siehe unten), damit aktuelle Informationen ggf. auch per E-mail verteilt werden können. Als Programmiersprache ist, entsprechend der in der Vorlesung gelehrten Sprache, Eiffel vorgeschrieben.

Der Wettbewerb wird auf einem festgelegten, für alle Programme gleichen, aber bis zur Ausscheidung geheimen Satz von Labyrinthen durchgeführt, die zum Teil per Hand konstruiert, zum Teil zufällig erzeugt sind. Da für alle Programme die gleichen Labyrinthe verwendet werden, spielen Zufallsfaktoren keine Rolle, es sei denn, das Steuerprogramm benutzt selbst nichtdeterministische Funktionen (wie das Beispielprogramm, dies liegt dann aber in der Entscheidung des Programmierers). Die für den im vergangenen Jahr (1998) ausgetragenen Wettbewerb verwendeten Labyrinthe sind im Verzeichnis `mazes` des Hamster-Programmpakets bzw. im HP-Pool unter `~borgelt/src/hamster/mazes` verfügbar. Ein Teil dieser Labyrinthe wird wahrscheinlich wiederverwendet, mindestens die Zufallslabyrinthe werden jedoch neu erstellt.

Zur Auswertung der Programme wird die Unix-Kommandozeilenversion `hamster` des Labyrinthprogramms verwendet, um die Abarbeitung des Labyrinthsatzes durch Shellskripte automatisieren zu können. Dies sollte auf das Ergebnis der Programme keinen Einfluß haben, da sich die Kommandozeilenversion und die Versionen mit graphischer Benutzerschnittstelle in allen wesentlichen Punkten gleich verhalten.

Der Quelltext des fertigen Hamster-Steuerprogramms ist bis zum

Freitag, den 11. Juni 1999, 12:00 Uhr mittags,

entweder per E-mail an Ilona Blümel

`ilona.bluemel@cs.uni-magdeburg.de` oder
`bluemel@iws.cs.uni-magdeburg.de` bzw.

oder an Christian Borgelt

`christian.borgelt@cs.uni-magdeburg.de` oder
`borgelt@iws.cs.uni-magdeburg.de`

zu schicken oder auf Diskette in den Übungen oder in einem der Räume 03.205a (Ilona Blümel) oder 03.206 (Christian Borgelt) abzugeben.

Das Ausführen der Programme auf den Wettbewerbslabyrinthen findet direkt anschließend statt, so daß mit den Ergebnissen und natürlich der Siegerehrung noch vor Semesterende zu rechnen ist.

6.3 WWW-Seite zum Wettbewerb

Aktuelle Informationen zum Wettbewerb gibt es im WWW unter der URL

`http://fuzzy.cs.uni-magdeburg.de/~studium/hamster/hamster.html`

Auf dieser Seite werden nach Abschluß des Wettbewerbs die Ergebnisse, die Wettbewerbslabyrinthe und die Hamster-Steuerprogramme veröffentlicht.