29. Bundeswettbewerb Informatik, 1. Runde

Lösungshinweise und Bewertungskriterien

Allgemeines

Das Wichtigste zuerst: Wir haben uns sehr darüber gefreut, dass einmal mehr so viele Leute sich die Mühe gemacht und die Zeit zur Bearbeitung der Aufgaben genommen haben.

Natürlich waren nicht alle Einsendungen perfekt, und einige eher äußerliche Anforderungen wurden häufiger missachtet. Im Einzelnen:

- Lösungsidee, Programm-Dokumentation und insbesondere auch Programm-Ablaufprotokolle und ausreichend viele Beispiele müssen ausgedruckt sein. Wir können aus Zeit- und Kostengründen keine Ausdrucke machen und auch nicht jedes eingesandte Programm ausführen.
- Noch schlechter als Einsendungen nur auf Datenträgern wären für uns übrigens Einsendungen via E-Mail oder anderen Internet-Wegen, auch wenn das für die Teilnehmer noch so praktisch wäre. Papiereinsendungen sind (zumindest zur Zeit und sicher auch noch in den nächsten Jahren) einfach unumgänglich.
- Beispiele werden als Teile des Programm-Ablaufprotokolls immer erwartet. Zu wenige Beispiele und erst recht die Nichtbearbeitung vorgegebener Beispiele führen zu Punktabzug. Es ist auch nicht ausreichend, Beispiele nur auf CD abzugeben, ins Programm einzubauen oder den Bewertern das Erfinden und Testen von Beispielen zu überlassen. Ohne abgedruckte Beispiele ist die Bewertung einer Lösung in der knappen vorhandenen Zeit nicht möglich. Leider fehlten in vielen Einsendungen die Beispiele, was oft das Erreichen der zweiten Runde verhindert hat.
- Zu einer Einsendung gehören auch lauffähige Programme. Kompilierung von Quellcode ist während der Bewertung nicht möglich. Für die gängigsten Skript-Sprachen stehen Interpreter zur Verfügung. Ohne die Abgabe eines eigenständig ausführbaren Programms

ist die Bewertung oft schwierig. Denken Sie insbesondere daran, wenn Sie eine Entwicklungsumgebung wie BlueJ benutzen, bei der die erstellten Programme ohne Weiteres nur innerhalb der Umgebung selbst laufen.

So, vielleicht denken Sie ja an diese Anmerkungen, wenn Sie (hoffentlich) im nächsten Jahr wieder mitmachen.

Auch die folgenden eher inhaltlichen Dinge sollten Sie beachten:

- Lösungsideen sollten Lösungsideen sein und keine Bedienungsanleitungen oder Wiederholungen der Aufgabenstellung. Es soll beschrieben werden, welches Problem hinter der Aufgabe steckt und wie dieses Problem grundsätzlich angegangen wird. Ein einfacher Grundsatz: Bezeichner von Programmelementen wie Variablen, Prozeduren etc. dürfen nicht verwendet werden – eine Lösungsidee ist nämlich unabhängig von solchen Realisierungsdetails.
- Auch ein Programmablauf-Protokoll soll keine Bedienungsanleitung sein. Es beschreibt nicht, wie das Programm ablaufen sollte, auch nicht die zum Ablauf nötigen Interaktionen mit dem Programm, sondern protokolliert den tatsächlichen, inneren Ablauf eines Programms. Am besten protokolliert ein Programm seinen Ablauf selbst, z. B. durch Herausschreiben von Eingaben, Zwischenschritten oder -resultaten und Ausgaben.
- Oben wurde schon gesagt, dass Beispiele immer dabei sein sollten, zumindest eines davon in einem Programm-Ablaufprotokoll. Das hat seinen Grund: An den Beispielen ist oft direkt zu sehen, ob bestimmte Punkte korrekt beachtet wurden. Viele meinen nun, wir könnten die Programme ja laufen lassen und selbst auf Beispieldaten ansetzen, und liefern keine Beispiele oder nur Beispieldaten in elektronischer Form. Das können wir aber aus Zeitmangel in der Regel nicht. Außerdem ist nicht immer sicher, dass Programme, die auf dem eigenen PC laufen, auch auf einem anderen Computer ausführbar sind. Generell muss man sich darauf einstellen, dass nur das Papiermaterial angesehen wird!
- Mit den verschiedenen Beispielen sollten Sie wichtige Varianten des Programmablaufs zeigen, also auch Sonderfälle, die Ihre Lösung behandeln kann. Die Konstruktion solcher Testfälle ist eine ganz wesentliche Tätigkeit des Programmentwurfs.

Einige Anmerkungen noch zur Bewertung:

- Pro Aufgabe werden maximal fünf Punkte vergeben, bei Mängeln gibt es entsprechend weniger Punkte. Für die Gesamtbewertung sind die drei am besten bewerteten Aufgabenlösungen maßgeblich, es lassen sich also maximal 15 Punkte erreichen. Einen 1. Preis erreichen Sie mit 14 oder 15 Punkten, einen 2. Preis mit 12 oder 13 Punkten und eine Anerkennung mit 9 bis 11 Punkten. Die Preisträger sind für die zweite Runde qualifiziert.
- Auf den Bewertungsbögen bedeutet ein Kreuz in einer Zeile, dass die (negative) Aussage in dieser Zeile auf Ihre Einsendung zutrifft. Damit verbunden ist dann in der Regel der Abzug eines oder mehrerer Punkte. Eine Wellenlinie bedeutet „na ja, hätte besser sein können“, führt aber alleine nicht zu Punktabzug. Mehrere Wellenlinien können sich aber zu einem Punktabzug addieren. Gelegentlich sind lobende Anmerkungen der Bewerter mit einem ’+’ versehen.

- Wellenlinien wurden übrigens häufig für die Dokumentation (Lösungsidee, Programm-Dokumentation, Programm-Ablaufprotokoll und kommentierter Programm-Text) verteilt, obwohl Punktabzug auch gerechtfertigt gewesen wäre.
- Aber auch so ließ sich nicht verhindern, dass etliche Teilnehmer nicht weitergekommen sind, die nur drei Aufgaben abgegeben haben in der Hoffnung, dass schon alle richtig sein würden. Das ist ziemlich riskant, da Fehler sich leicht einschleichen.

Zum Schluss:

- Sollte Ihr Name auf der Urkunde falsch geschrieben sein, können Sie gerne eine neue anfordern.
- Es ist verständlich, wenn jemand, der nicht weitergekommen ist, über eine Reklamation nachdenkt. Gehen Sie aber bitte davon aus, dass wir kritische Fälle, insbesondere die mit 11 Punkten, schon genau und mit Wohlwollen geprüft haben.

Danksagung

An der Erstellung der Lösungsideen haben mitgewirkt: Robert Czechowski und Niolai Wyderka (Junioraufgabe), Felix Frei (Aufgabe 2), Thomas Leineweber (Aufgabe 3), Felix Arends und Tobias Polley (Aufgabe 4) und Johannes Pieper (Aufgabe 5).

Die Aufgaben wurden vom Aufgabenausschuss des Bundeswettbewerbs Informatik entwickelt, und zwar aus Vorschlägen von Kirsten Riechmann (Junioraufgabe), Wolfgang Pohl (Aufgabe 1), Peter Rossmanith (Aufgabe 2), Thomas Bendig (Aufgabe 3), Ralf Punkenburg (Aufgabe 4) und Torben Hagerup (Aufgabe 5).

Junioraufgabe: Horoskope

0.1 Lösungsidee

Klären wir zuerst einige Begriffe: Das geforderte Programm soll einzelne *Horoskopsätze* bilden, drei Sätze zu verschiedenen Themen zum *Horoskop* eines Sternzeichens zusammenstellen und schließlich in der Lage sein, wiederholt für alle zwölf Sternzeichen eine gesamte *Horoskopmenge* auszugeben.

Ein Hauptproblem dieser Aufgabe besteht darin, grammatisch korrekte Sätze zu produzieren. Dabei ist vor allem auf die Deklination der Adjektive und Pronomen, abhängig vom Genus, Kasus (Fall) und Numerus (Singular oder Plural) des jeweiligen Substantivs im Satzbau zu achten. Dies kann auf verschiedene Arten gewährleistet werden:

- Eine einfache Lösung dafür ist, bereits korrekt deklinierte Wortgruppen zu speichern und dann zufällig nach dem Baukastenprinzip zusammenzusetzen.
- Eine aufwendigere Methode ist es, einzelne Wörter nach dem im Satz vorhandenen Kontext abhängig vom Fall und der Eigenschaft des Substantivs zu beugen.

In dieser Umsetzung wird für jedes Themengebiet (Schule, Gesundheit, Liebe/Freundschaft) eine Wortliste von Adjektiven und Substantiven verwendet; zu jedem Substantiv ist darin dessen Genus und Numerus enthalten. Anschließend wird zufällig ein vom Themengebiet abhängiges Satzmuster ausgewählt, für das die Fälle der einzusetzenden Wörter bekannt sind, so dass diese – ebenfalls zufällig – ausgewählt und passend dekliniert werden können. Im Themengebiet Schule kann zum Beispiel folgendes Muster betrachtet werden:

> Deine kreative Art bringt dir endlich den ersehnten Erfolg.
> Dein[e] <Adjektiv>[e|es] <Substantiv> bringt dir <Zusatz> (den|die|das) <Adjektiv>[en|e] <Substantiv>.

Dafür muss ein Substantiv für eine Eigenschaft des Lesers und ein Adjektiv, das zur Beschreibung dieser Eigenschaft dient, beispielsweise *Art* und *kreativ*, ausgewählt werden. Anhand des Genus und des Wissens, dass dieser Satzteil im Nominativ steht, kann nun das Adjektiv dekliniert werden, so dass die grammatisch korrekte Form entsteht. Im Prinzip muss dabei an den Wortstamm jeweils nur die passende Endung angehängt werden.

Um von der aktuellen Woche und dem aktuellen Jahr abhängige und reproduzierbare Ergebnisse zu erhalten, wird der Zufallsgenerator abhängig von diesen Größen initialisiert. Dadurch erhält man für jede Woche eindeutige, verschiedene Horoskopmengen.

Ein weiteres Problem ist die Gefahr, dass ein Horoskopsatz zweimal für zwei verschiedene Sternzeichen erzeugt wird. Dieses Problem wird in dieser Lösung explizit verhindert, indem eine Liste bereits generierter Horoskopsätze angelegt wird und bei jeder Erzeugung geprüft wird, ob ein Duplikat erzeugt wird.

Alternativ kann man auch das Risiko durch eine sehr große Anzahl von Möglichkeiten so gering halten, dass die Wahrscheinlichkeit, zwei identische Horoskope zu erhalten, verschwindend gering wird. Dies sollte jedoch gut begründet werden.

Satzmuster

In der Umsetzung wird beispielhaft für jeden Themenbereich ein Satzmuster verwendet, was sich jedoch beliebig ausbauen ließe. Im Folgenden bezeichnen die Ausdrücke in runden Klammern eine Auswahl aus verschiedenen Möglichkeiten, die durch „|“ getrennt werden. Die eckigen Klammern erlauben zusätzlich das Weglassen des Ausdrucks in Klammern.

Schule

Muster Dein[e] <Adjektiv>(e|es|en) <Substantiv> bringt dir <Zusatz> (den|die|das) <Ajdektiv>(e|n) <Substantiv>.

Beispiel Deine kreative Art bringt Dir endlich den ersehnten Erfolg.

Gesundheit

Muster <Zusatz> wirkt sich (der|die|das) <Adjektiv>e <Substantiv> auf dein[e|en] <Substantiv> aus.

Beispiel Langsam wirkt sich die dauernde Einsamkeit auf dein Wohlbefinden aus.

Liebe und Freundschaft

Muster Denke nicht zu viel über (den|das|die) <Substantiv> nach, sondern <Verb> (den|das|die) <Adjektiv>(e|n) <Substantiv> mit dein(em|er|en) <Substantiv>.

Beispiel Denke nicht zu viel über die Zukunft nach, sondern genieße die schöne Zeit mit deinen Freunden.

0.2 Programmdokumentation

Umsetzung

Das Programm besteht aus mehreren Klassen, die zur Verwaltung der verschiedenen Themenbereiche genutzt werden. Es gibt eine Basisklasse `Thema`, welche die Methode `generate()` bereitstellt, die einen zufälligen Horoskopsatz eines bestimmten Themenbereichs erzeugen soll. Von dieser Klasse werden drei Klassen abgeleitet, jeweils eine für die Themenbereiche Schule, Gesundheit und Liebe/Freundschaft. Dadurch ist es möglich, das Programm später einfach zu erweitern, wenn neue Themengebiete hinzukommen sollen.

Jede der Klassen lädt im Konstruktor selbstständig die verwendeten Wortlisten, da diese für jeden Themenbereich individuell sind. Bei Adjektiven und Zusatzwörtern wird dazu der Wortstamm als Zeichenkette gespeichert, bei Substantiven werden außerdem der Genus und der Numerus in einem Verbund (**`struct`**) gespeichert. Die geerbte Methode `generate()` wird dann überschrieben. Dort werden zunächst solange zufällig Wörter ausgewählt, bis eine Kombination gefunden wird, die noch nicht verwendet wurde. Dazu werden die ausgewählten Wortnummern in einem **`int`**-Array gespeichert und mit jedem Eintrag der Liste der bereits

gewählten Kombinationen verglichen. Ist eine unbenutzte Kombination erzeugt worden, wird sie verwendet und zur Liste hinzugefügt.

Anschließend werden die gewählten Wörter entsprechend der Regeln des ausgewählten Satzmusters eingesetzt.

In der Hauptmethode werden zunächst das Jahr und die Wochennummer eingelesen, für die die Horoskopmenge erstellt werden soll. Anschließend wird der Startwert (seed) für die Zufallsfunktion entsprechend der Eingaben gesetzt, so dass gleiche Eingaben zu gleichen Ergebnissen führen. Nun wird eine Instanz jeder Themenklasse erzeugt und für jedes Sternzeichen aus jedem Themengebiet ein Horoskop erzeugt und ausgegeben.

Format der Wortlisten

Für das Einlesen der Wortlisten stehen die Funktionen `readToStringVector(string filename, vector<string>* v)` für einfache Wörter und `readToSubstantivVector (string filename, vector<Substantiv>* v)` für Substantive mit Numerus und Genus zur Verfügung. Diese lesen aus den durch `filename` angegebenen Dateien die Wörter in die angegebenen Vektoren. Die Wortlisten für Adjektive und Zusatzwörtern bestehen aus jeweils einem Wortstamm pro Zeile.

Die Zeilen der Wortlisten für Substantive enthalten neben dem Substantiv selbst noch ein Zeichen für den Genus (`m` für masuklin, `f` für feminin und `n` für neutral) sowie zwei Zeichen für den Numerus (`sg` für Singular, `pl` für Plural). Diese werden durch jeweils ein Leerzeichen voneinander getrennt.

Beispiel einer Liste von Adjektiven:

```
dauernd
wiederholt
anhaltend
...
```

Beispiel einer Substantiv-Liste:

```
Zeit f sg
Momente m pl
Erlebnisse n pl
...
```

0.3 Programmablauf

Nach dem Starten werden das Jahr und die Wochennummer eingelesen. Dabei wird davon ausgegangen, dass korrekte ganze Zahlen eingegeben werden. Daraufhin wird intern der Startwert des Zufallsgenerators auf $100 \cdot \mathit{Jahr} + \mathit{Woche}$ gesetzt. Anschließend werden die Klasseninstanzen erzeugt, wobei die Wortlisten aus den Dateien gelesen werden. Nun wird nacheinander für jedes Sternzeichen zunächst der Name des Sternzeichens ausgegeben. Danach wird für jedes Thema ein Horoskopsatz generiert. Dazu werden aus den Wortlisten der Themenklasse zufällig Wörter ausgewählt. Abhängig vom Genus der gezogenen Wörter wird aus dem Satzmuster unter Zuhilfenahme einiger Reihungen, die die Endungen der Adjektive abhängig von Fall und der Bestimmtheit enthalten, das Ergebnis erzeugt und zurückgegeben, woraufhin es ausgegeben wird.

Ein vollständiger Programmablauf kann zum Beispiel so aussehen:

```
Jahr > 2010
Woche > 50

Wassermann:
Dein kreativer Wahn bringt dir endlich die erhoffte Leistung.
Bald wirkt sich die anhaltende Ertuechtigung auf deine Physiologie aus.
Denke nicht zu viel ueber die Klausuren nach, sondern geniesse die
   unerwarteten Erlebnisse mit deinen Freunden.

Fische:
Dein starkes Wesen bringt dir schliesslich die erhoffte Wirkung.
Langsam wirkt sich die wiederkehrende Anstrengung auf deine Frische aus.
Denke nicht zu viel ueber die Vergangenheit nach, sondern nutze die wenigen
    Erlebnisse mit deinen Freunden.

Widder:
Dein gewinnendes Laecheln bringt dir voruebergehend die ersehnte Wirkung.
Vielleicht wirkt sich die wiederkehrende Einsamkeit auf deine Psyche aus.
Denke nicht zu viel ueber die Verpflichtungen nach, sondern geniesse die
   naechste Zeit mit deinen Freunden.

Stier:
Deine kreative Beharrlichkeit bringt dir womoeglich den ersehnten Erfolg.
Bald wirkt sich die anhaltende Einsamkeit auf deinen Schlaf aus.
Denke nicht zu viel ueber die Verpflichtungen nach, sondern geniesse die
   wunderbare Zeit mit deiner Familie.

Zwillinge:
Deine starke Art bringt dir endlich den ersehnten Erfolg.
Irgendwann wirkt sich die anhaltende Erholung auf deine Physiologie aus.
Denke nicht zu viel ueber den Stress nach, sondern erlebe die naechsten
   Erlebnisse mit deinem Computer.

Krebs:
Dein gewinnendes Wesen bringt dir womoeglich den gewuenschten Erfolg.
Bald wirkt sich der wiederholte Stress auf deinen Schlaf aus.
Denke nicht zu viel ueber den Stress nach, sondern geniesse die guten
   Erlebnisse mit deiner Familie.

Loewe:
Dein gewinnendes Laecheln bringt dir voruebergehend die ersehnte Leistung.
Irgendwann wirkt sich die sporadische Erholung auf deinen Schlaf aus.
Denke nicht zu viel ueber die Sorgen nach, sondern erlebe die besonderen
   Momente mit deinem Computer.

Jungfrau:
Deine starke Art bringt dir voruebergehend das ersehnte Ergebnis.
Vielleicht wirkt sich die aufreibende Arbeit auf deine Psyche aus.
Denke nicht zu viel ueber die Anstrengungen nach, sondern geniesse die
   einzigartigen Momente mit deiner Familie.

Waage:
Dein starkes Wesen bringt dir schliesslich die gewuenschte Wirkung.
Bald wirkt sich die aufreibende Arbeit auf deine Frische aus.
Denke nicht zu viel ueber die Arbeit nach, sondern erlebe die naechsten
```



```
   Momente mit deinem Partner.

Skorpion:
Dein gewinnender Wahn bringt dir bald den erwarteten Erfolg.
Irgendwann wirkt sich der sporadische Stress auf deine Gesundheit aus.
Denke nicht zu viel ueber die Vergangenheit nach, sondern geniesse die
   unerwarteten Erlebnisse mit deinem Partner.

Schuetze:
Dein gewinnendes Vorgehen bringt dir schliesslich die gewuenschte Leistung.
Langsam wirkt sich der anhaltende Stress auf dein Wohlbefinden aus.
Denke nicht zu viel ueber die Anstrengungen nach, sondern nutze die
   wunderbaren Momente mit deiner Familie.

Steinbock:
Deine gewinnende Beharrlichkeit bringt dir bald den gewuenschten Erfolg.
Irgendwann wirkt sich die wiederkehrende Erholung auf deine Gesundheit aus.
Denke nicht zu viel ueber die Vergangenheit nach, sondern erlebe die
   kommenden Momente mit deiner Familie.
```

0.4 Bewertungskriterien

- Die Lösung muss in der Lage sein, eine komplette Menge an Horoskopen für alle zwölf Sternzeichen zu erzeugen.
- Jedes einzelne Horoskop soll aus drei Sätzen bestehen, denen jeweils ein eigenes Satzmuster (möglichst, aber nicht notwendigerweise mit einem spezifischen Thema) zu Grunde liegt.
- In einer kompletten Horoskopmenge soll möglichst kein Satz doppelt vorkommen. Doppelte Sätze können explizit verhindert werden oder durch das gewählte Generierungsverfahren sehr unwahrscheinlich sein. Im zweiten Fall sollte begründet sein, dass und idealerweise auch warum eine Dopplung unwahrscheinlich ist.
- Die Sätze sollen grammatisch korrekt sein. Es ist in Ordnung, diese Bedingung mit eher einfachen Mitteln zu erfüllen, etwa durch Einbindung vorgefertigter Wortgruppen in relativ grobe Satzmuster.
- Die Horoskope sollten einigermaßen abwechslungsreich sein. Der verwendbare Vorrat an Wörtern oder Wortgruppen muss also einen gewissen Umfang haben und ist idealerweise einfach erweiterbar. Außerdem sollten die Muster nicht zu einfach gestrickt sein. Es genügt z. B. nicht, die Sätze so aus zwei Satzhälften zu kombinieren, dass die Prädikat-Objekt-Phrasen immer gleich sind.
  Auch die Horoskopmengen sollten eine gewisse Abwechslung aufweisen. Die Lösung sollte in der Lage sein, unterschiedliche Kompletthoroskope zu generieren, abhängig z. B. von einem Eingabeparameter oder dem aktuellen Datum – oder vom Zufall.
- Es muss ein komplettes Horoskop dokumentiert sein. Dazu sollten zumindest einige einzelne Beispielsätze das Verfahren zur Satzbildung illustrieren.

```
startshape SEED1

rule SEED1 {
 SQUARE{}
 SEED1 {y 1.2 size 0.99 rotate 1.5 brightness 0.03}
}

rule SEED1 0.05 {SEED1 {flip 90}}

rule SEED1 0.05 {
 SQUARE{}
 SEED1 {y 1.2 s 0.99 r 1.5 b -0.5 flip 90}
 SEED1 {y 1.2 x 1.2 s 0.6 r -60 b -0.5}
 SEED1 {y 1.2 x -1.2 s 0.5 r 60 b -0.5 flip 90}
}
```

Abbildung 1: CFDG-Code des Beispiels weighting_demo

Aufgabe 1: Informartik

Im Folgenden wird davon ausgegangen, dass die Leser grundsätzlich mit dem Programm „Context Free“ und den Möglichkeiten der zugehörigen Regelsprache CFDG vertraut sind. Dennoch werden relevante Sprachelemente an einem Beispiel erläutert. Weitere Einzelheiten sind unter www.contextfreeart.org nachzulesen. Der BWINF hat für diese erste Runde eine deutsche Übersetzung der Dokumentation zu CFDG zur Verfügung gestellt.

1.1 Künstlerische Absichten

Variation eines Beispiels

Inspiriert durch das Programm-Icon von „Context Free“ sowie von einigen der mit dem Programm mitgelieferten Beispiele (etwa mtree, tangle und weighting_demo) sollte ein baumartiges Gebilde entstehen. Im Beispiel weighting_demo wird das mit einer einzigen Form (engl.: shape) realisiert (s. Abb. 1). Dieses Beispiel wird nun näher erläutert, da es die wesentlichen Aspekte der Lösung schon enthält.

Für die Form `SEED1` gibt es drei Regeln. Die erste sorgt mit dem Aufruf der Grundform `SQUARE` dafür, dass überhaupt etwas gezeichnet wird. Anschließend ruft sie die Form rekursiv auf. Die Parameter dieses Aufrufs wirken sich wie folgt aus:

y 1.2 sorgt dafür, dass der nächste Ansatz für die Form über dem gezeichneten Quadrat erfolgt – wobei „über“ sich nicht auf die absolute Vertikale, sondern auf die aktuelle Ausrichtung der Zeichnung bezieht.

size 0.99 gibt einen Skalierungsfaktor an und bewirkt, dass das nächste Objekt etwas kleiner (Wert kleiner als 1) gezeichnet wird als das letzte. ’size’ kann mit ’s’ abgekürzt werden.

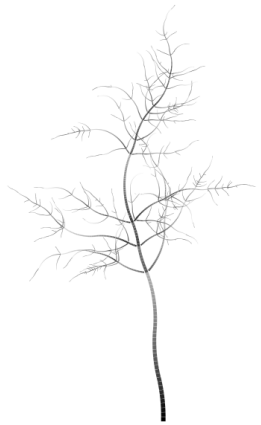

Abbildung 2: Ein durch den Code aus Abbildung 1 generiertes Bild.

rotate 1.5 gibt einen Rotationswinkel an. Der kleine Wert 1.5 bewirkt eine leichte Drehung nach links – und zwar der gesamten Zeichenebene, die damit im nächsten Schritt neu orientiert ist.

brightness 0.03 gibt praktisch eine prozentuale Änderung der Helligkeit an. Hier wird also der Helligkeitswert um 3 Prozent erhöht. Helligkeitswerte können zwischen 0 und 1 liegen, höhere Werte sind heller.

Die zweite Regel ruft die Form rekursiv auf. Dabei wird mit `flip 90` dafür gesorgt, dass die für den nächsten Aufruf gewählte Zeichenrichtung (die z.B. in der o.g. Basisregel sich durch eine leichte Drehung nach links ergibt) an der aktuellen y-Achse (das ist die aktuelle Zeichenrichtung) gespiegelt wird. Eine Drehung würde also ihre Drehrichtung wechseln.

Die dritte Regel sorgt mit drei rekursiven Aufrufen dafür, dass sich die Form gelegentlich (die Auswahlwahrscheinlichkeit dieser Regel ist mit 0.05 eher gering) dreifach verzweigt. Der erste rekursive Aufruf enstpricht praktisch der ersten Regel und sorgt für weiteres Wachstum des aktuellen Hauptzweiges. Die beiden anderen Aufrufe sorgen für Verzweigungen um 60 Grad nach rechts (`r -60`, Versatz nach rechts mit `x 1.2`) und links (`r 60`, Versatz nach links mit `x -1.2`).

Abbildung 2 zeigt ein aus dem obigen CFDG-Code generiertes Bild.

Beabsichtigte Anpassungen

In dieser Lösung soll in Abänderung des Beispiels erreicht werden:

1. eine engere und häufigere Verzweigung, um zu einer dichten Baumkrone zu kommen;

```
startshape PALMTREE

rule PALMTREE 0.56 {
        BASESHAPE {}
        PALMTREE {y 1.02 s 0.96 r 3 b 0.1 h 2}
}

rule PALMTREE 0.14 {
        PALMTREE { r 30 s 0.9}
        PALMTREE { r -30 s 0.9}
}

rule PALMTREE 0.30 {
        PALMTREE {s 0.96 r -1 b 0.1 h 2}
}

rule PALMTREE 0.005 { }

rule BASESHAPE {
        SQUARE { sat 1 }
}
```

Abbildung 3: Code des CFDG-Lösungsprogramms (5 Regeln, 16 Zeilen ohne Leerzeilen)

2. eine stärkere Verjüngung des Stamms bzw. der Äste und
3. eine farbige Darstellung, mit einer Entwicklung von Schwarz über Braun (Stamm und Äste) bis zu Grün (Blattwerk).

Dies ist mit den folgenden Mitteln zu erreichen: eine Verzweigungsregel sollte eine höhere Gewichtung haben, während der Verzweigungswinkel enger gesetzt ist (Punkt 1); die Skalierungsfaktoren für rekursive Aufrufe sollen kleiner sein (Punkt 2); und über die Regeln hinweg soll mit geeigneten Parameterwerten ein Farbverlauf realisiert werden (Punkt 3).

1.2 Programm-Dokumentation

Abbildung 3 enthält den CFDG-Code der Lösung. Die Hauptform wurde `PALMTREE` benannt, weil die entstehenden Bilder häufig an Palmen erinnern. Wie werden nun die gesetzten Ziele erreicht?

stärkere und engere Verzweigung Hierfür ist die zweite Regel verantwortlich. Sie wird mit etwa 14-prozentiger Wahrscheinlichkeit recht häufig ausgewählt. Die Verzweigung ist hier nur eine Gabelung, dafür beträgt der Winkel zwischen den Zweigen nur 60 Grad (eine Rotation um 30 Grad nach links, eine um 30 Grad nach rechts).

stärkere Verjüngung des Stamms Dies wird durch den etwas kleineren Skalierungsfaktor von 0.96 in den Regeln eins und drei erreicht. In der Verzweigungsregel wird auf den Skalierungsfaktor verzichtet, damit die Verästelung letztlich nicht zu dicht wird.

Abbildung 4: Drei „Palmen“ mit teilweise mehr als grünen Blättern.

Farbverlauf In den Regeln eins und drei sowie in der Regel zur Form `BASESHAPE` werden hierzu Parameter zur Farbsteuerung verwendet. ContextFree arbeitet mit dem HSB-Farbmodell: H = Hue steht für den Farbton, S = Saturation für die Farbsättigung und B = Brightness für den Helligkeitsgrad. Bei einem Sättigungswert von 0 gibt es keine Farben, sondern nur Grautöne. Deshalb wird in der Regel für `BASESHAPE` die Sättigung auf 1 gesetzt. Da ContextFree mit einer Helligkeit von 0 startet, ist das erste gezeichnete Objekt dennoch schwarz. Der Startwert für den Farbton ist auch 0, was der Farbe Rot entspricht. Würde man nun nur die Helligkeit ansteigen lassen (in den Regeln eins und drei mit der Steigerungsrate `b 0.1`), ergäbe sich ein Baum mit roten Ästen. Gleichzeitig muss sich also auch der Farbton ändern. In den Regeln eins und drei wird also der Parameter `h 2` angegeben, der den Farbton auf dem 360 Werte umfassenden Farbkreis mit jedem Aufruf um 2 verändert. Dabei wandert der Farbton von Rot über Grün und Blau wieder zu Rot.

Zum Farbverlauf ist noch zu ergänzen: Dass in den allermeisten Bildern, die durch den CFDG-Code der Lösung erzeugt werden, der Farbverlauf nicht oder nur leicht über Grün hinausgeht, zeigt, dass die Rekursionstiefe begrenzt ist (was u.a. durch die „Stoppregel“ 4 befördert wird). Aber es ist auch möglich, den Farbverlauf gezielt bei Grün enden zu lassen. Dafür verwendet man die Angabe `|hue` *`c`*, um einen Zielfarbton c zu setzen, und den Parameter `hue` *`rate`*`|`, um den Farbton mit einer gewissen prozentualen Rate dem Zielfarbton anzunähern. Die Lösung müsste dazu um eine Startregel ergänzt werden, die den Zielfarbton angibt (die startshape-Anweisung muss entsprechend geändert werden):

```
rule PALMTREESTART {
        PALMTREE { sat 1 |hue 130}
}
```

Außerdem ist statt der Farbänderung `h 2` die Änderungsrate `h 0.02|` zu setzen. Dann endet der Farbverlauf verlässlich bei Grün (vgl. Abbildung 5).

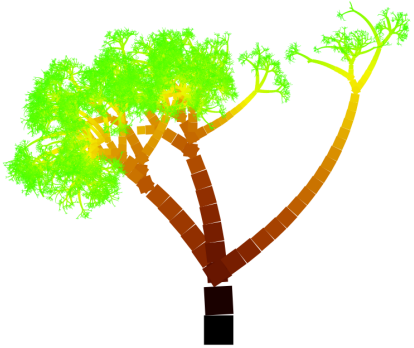

Abbildung 5: Eine „Palme“ mit garantiert nicht mehr als grünen Blättern.

1.3 Bewertungskriterien

Die Aufgabenstellung lässt für die Bearbeitung sehr viel Spielraum, die Lust am Experimentieren sollte im Vordergrund stehen. Die Herausforderung war, sich die Beschreibungssprache CFDG einschließlich ihrer Semantik anzueignen und gezielt einzusetzen. Daran hat sich die Bewertung orientiert. Konkret wurden folgende Punkte beachtet:

- Die künstlerischen Absichten müssen nachvollziehbar erläutert sein. Natürlich ist es erlaubt, sich an den Beispielen, die mit Context Free bzw. auf der zugehörigen Webseite veröffentlicht sind, zu orientieren.
- Nicht nur die Absichten müssen erläutert sein, sondern auch die Umsetzung in CFDG-Code muss erklärt werden.
- Die erfolgreiche Umsetzung der Absichten soll mit Bildvariationen gezeigt werden, die vom gleichen Programm erzeugt wurden. Dazu muss das Programm Variationen ermöglichen, was nur dann möglich ist, wenn für mindestens eine Form mehr als eine Regel definiert wurde. Die Variationen sollen also nicht durch Veränderung des Programms erzeugt werden.
- Das Spannende an Context Free ist, dass insbesondere durch Rekursion und Regelvarianten sehr kurze Programme zu komplexen Bildern führen können. Die Vorgaben der Aufgabenstellung an die Länge des Programms sollen deshalb einigermaßen eingehalten werden, wobei die Zeilenzählung im Zweifel zugunsten der Einsendung ausfällt. Wichtig ist außerdem, dass die Bilder nicht im wesentlichen doch „von Hand“ gezeichnet sind: etwa durch Programmierung komplexer Grundformen (mit der `path` -Anweisung), Regeln ohne Rekursion, sehr explizite Steuerung von Platzierung und Farbgebung der gezeichneten Formen etc. Auch hier war die Bewertung großzügig; nur bei allzu sehr „ausprogrammierten“ Bildern gab es Punktabzug.

- Lösungsidee und Umsetzung sollten deutlich machen, dass wesentliche Prinzipien von Context Free nicht völlig missverstanden wurden. Dazu gehören außer Rekursion und Regelvarianten auch, dass die meisten Steuerungsparameter relativ zum aktuellen Systemzustand wirken (relativ zur aktuellen Ausrichtung der Zeichenebene, zur aktuellen Farbwahl etc.).
- Die Aufgabenstellung fordert explizit drei Bildvariationen; diese müssen alle in der Dokumentation abgebildet sein.

Aufgabe 2: Robuttons

2.1 Lösungsidee

Diese Aufgabe lässt viele verschiedene Wege zu, die einzelnen Teilprobleme zu lösen. Im Folgenden wird zu jedem Teilproblem einer dieser Lösungswege detailliert beschrieben, und ein paar andere werden zumindest angesprochen.

Architektur

Zunächst zu der Frage, wie die (softwareseitige) Architektur eines Simulationsprogramms aussehen könnte. Aus dieser Perspektive drängen sich zwei Beobachtungen förmlich auf:

- Die Simulation besteht aus einer Handvoll Objekten, die teilweise ähnlich zueinander sind.
- Diese Objekte sollen veränderliche Eigenschaften (Größe, Anzahl, Tischform...) haben.

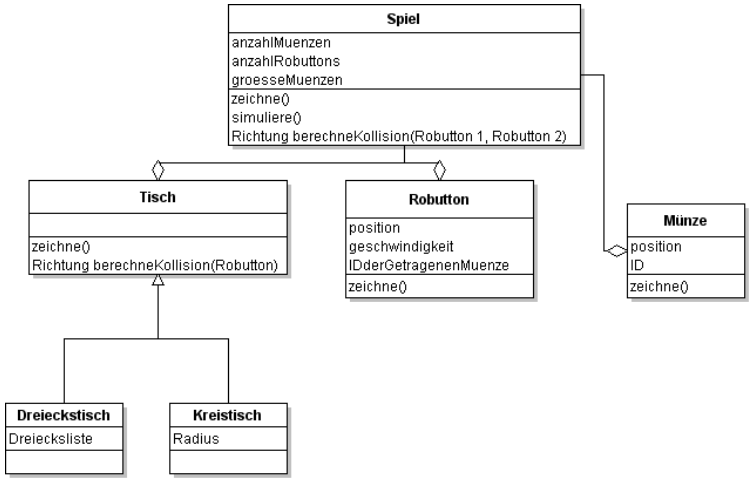

Abbildung 6: Ein UML-Diagramm für die Beispiellösung.

Daher bietet es sich an, für die Simulation ein objektorientiertes Programm zu schreiben. Abbildung 6 zeigt ein vereinfachtes, beispielhaftes UML-Diagramm für diese Lösung. Man beachte, dass der Tisch das Attribut 'Größe' trägt, da in der Aufgabenstellung gefordert ist, dass ein Tisch verschiedene Größen haben kann. Es ist z.B. möglich, ein Format zu definieren für Tische von normalisierten Größen und die eigentliche Größe des Tisches abzuspeichern.

Genauso ist es aber in Ordnung, die Größe der Münzen und Robuttons veränderlich zu halten, da dies die gleiche Bedeutung für die Simulation hat.

Der Vorteil dieser Architektur ist, dass das Programm erweiterbar gehalten wird: völlig verschiedene Tischformen können als Kindklassen von der abstrakten Basisklasse Tisch implementiert werden. Auch das Verhalten der Klassen für Münze und Robutton kann geändert werden (z.B. kann eine andere grafische Darstellung oder Kollisionsüberprüfung eingeführt werden), ohne den Rest des Programms anzupassen.

Natürlich ist eine objektorientierte Herangehensweise bei dieser Aufgabe nicht zwingend.

Anzeige

Eine „Visualisierung", also grafische Darstellung der Simulation ist gefordert. Eine Berechnung der Koordinaten allein ist nicht ausreichend. Die grafische Darstellung kann ruhig sehr simpel gehalten werden, aufwendige Effekte, Texturen, Beleuchtung oder eine aufwendige GUI wird nicht erwartet. Eine Anzeige wie in Abbildung 7 ist völlig ausreichend.

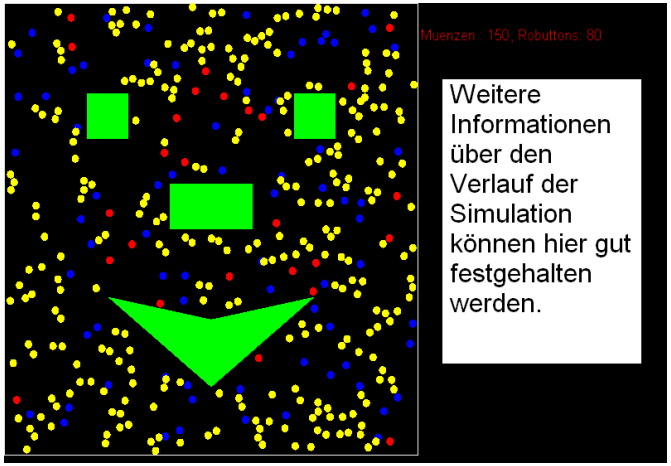

Abbildung 7: Eine mögliche Benutzungsoberfläche.

Hierbei sind die Münzen sowie die Robuttons mit bzw. ohne Münze durch die Farben gelb, rot, und blau unterscheidbar. Die Tischfläche, auf der die Münzen sich befinden dürfen, ist schwarz, die weißen Striche am Rand sowie die grünen Flächen sind Beschränkungen des Tisches. Eine Ausgabe (rechts) darüber, wie viele Münzen und Robuttons im Spiel sind, und über weitere Details der Simulation ist nützlich, aber nicht unbedingt verlangt.

Berechnungen der Anfangsposition

Zu Anfang der Simulation sollten die Positionen der Münzen und Robuttons entweder sinnvoll gewählt werden (vom Benutzer oder vom Programm), oder sie sollten zufällig positioniert werden. Die zufällige Positionierung ist nicht schwer zu realisieren: Nacheinander wird einfach für jeden Robutton und jede Münze eine neue Position ausgewürfelt, so lange bis diese Position eine gültige ist (also noch nicht besetzt ist und auf der Tischfläche liegt).

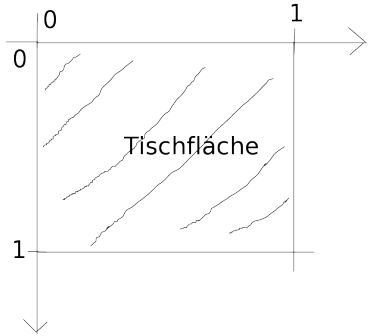

Abbildung 8: Grundfläche für Tischformen.

Modellierung des Tisches

Eine einfache Möglichkeit, möglichst viele Tischformen generieren zu können, ist die folgende Modellierung: Es wird festgelegt, dass der Tisch maximal auf einer Fläche von 1x1 definiert werden kann, die Kanten (0,0)-(0,1), (1,0)-(1,1), (0,1)-(1,1), (0,0)-(1,0) sind die Tischbegrenzung (vgl. Abbildung 8).

Weiterhin werden die Flächen, die (innerhalb der Begrenzung) *nicht* zur Tischfläche gehören, als Liste von Dreiecken implementiert. Die Dreiecksliste (0, 0), (0, 0.5), (0.5, 0.5); (0, 1), (0, 0.5), (0.5, 0.5) z. B. ergibt am Ende die in Abbildung 9 dargestellte Tischform.

Der Vorteil dieser Art der Modellierung liegt darin, alle Tischformen generieren zu können, die sich auf eckige Kanten beschränken. Auch ausgefallene Tischformen, wie die in Abbildung 7 gezeigte, können so realisiert werden. Tische mit runden Kanten können zumindest näherungsweise dargestellt werden.

Weitere Varianten: Modellierung des Tisches

Kreisförmiger Tisch Ein kreisförmiger Tisch (vgl. Abbildung 10) kann recht einfach implementiert werden, da die Berechnung der Kollision von Robuttons mit der durch den Kreis

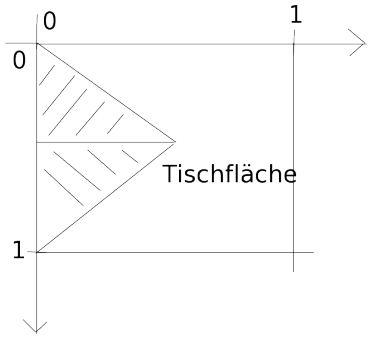

Abbildung 9: Ein Tisch aus zwei Dreiecken.

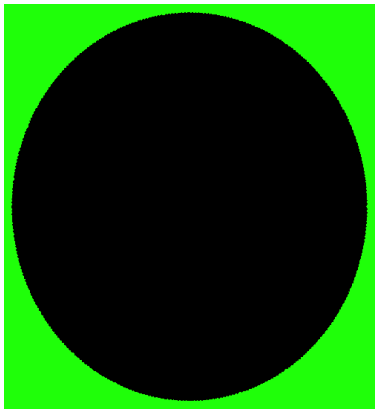

Abbildung 10: Ein kreisförmiger Tisch.

gegebenen Tischkante einfach ist.

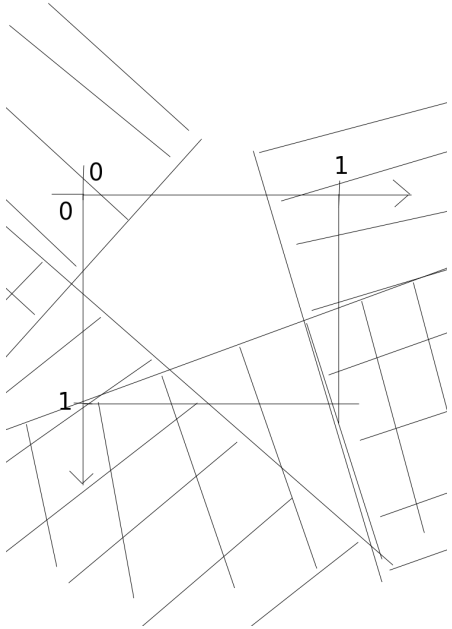

Abbildung 11: Modellierung der Tischform mit Halbebenen.

Beschränkung durch Halbebenen Analog zur Methode der Linearen Programmierung, den Suchraum einzuschränken, können hier mit einer Handvoll Halbebenen die Kanten des Tisches modelliert werden (vgl. Abbildung 11). Diese Methode erlaubt die Modellierung beliebiger konvexer Tische ohne eingebaute Hindernisse und ermöglicht eine recht leichte Kollisionserkennung, doch leider lässt sie keine konkaven Tischformen zu.

Verbundene Punkte Als Verallgemeinerung der Methode der Halbebenen kann auch einfach eine geordnete Folge von Punkten angegeben werden (vgl. Abbildung 12).

Funktionsgraphen als Begrenzung Der Tisch kann durch eine Menge von Begrenzungsgraphen und einer Seitenangabe modelliert werden. In Abbildung 13 wird der Tischbereich durch eine Gerade und eine verschobene Normalparabel definiert.

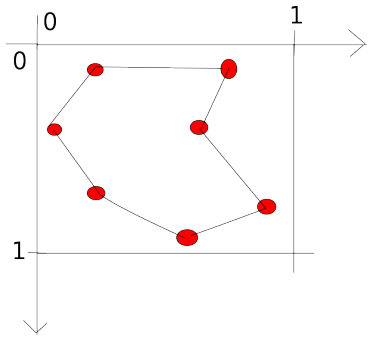

Abbildung 12: Modellierung der Tischform durch eine Punktfolge.

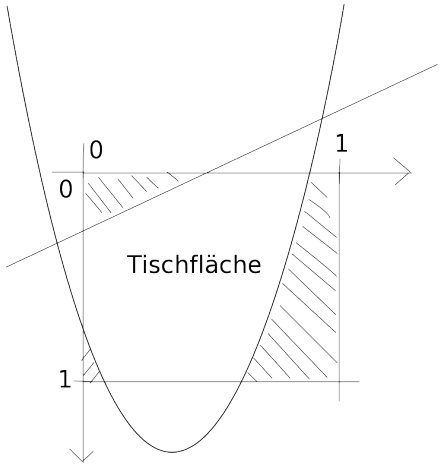

Abbildung 13: Begrenzung der Tischform durch Funktionsgraphen.

Kollision von Robutton und Robutton

Es ist einfach zu bestimmen, ob zwei Robuttons kollidieren: Zwei Robuttons R_1 und R_2 kollidieren genau dann, wenn der Abstand ihrer Mittelpunkte kleiner oder gleich dem doppelten Robutton-Radius r ist. In den folgenden Formeln sind die Robuttons im Zweifel als ihre Mittelpunkte zu lesen. Damit kann die Kollisionsbedingung so geschrieben werden ($d()$ berechnet den euklidischen Abstand zwischen zwei Punkten):

$$d(R_1, R_2) \leq 2 \cdot r$$

$$d(a,b) = \sqrt{(a.x - b.x)^2 + (a.y - b.y)^2}$$

Wenn zwei Robuttons kollidieren, sollen diese ihre Richtung zunächst um 180° ändern. Dies ist einfach zu implementieren: Ist v_i der Richtungsvektor von R_i vor der Kollision, so ist $-v_i$ der Richtungsvektor von R_i nach der Drehung um 180°. Nun sollen R_1 und R_2 noch zusätzlich um +/- 90° gedreht werden. Dies funktioniert z.B. mit Hilfe der Drehmatrix:

$$D(\alpha) = \begin{pmatrix} cos\ \alpha & -sin\ \alpha \\ sin\ \alpha & cos\ \alpha \end{pmatrix}$$

Die Vektoren v_i' nach der Kollision ergeben sich dann wie folgt (Matrix-Vektor-Multiplikation):

$$v_i' = D(\alpha_i)(-v_i),\ \alpha_i \in [-90°, +90°]$$

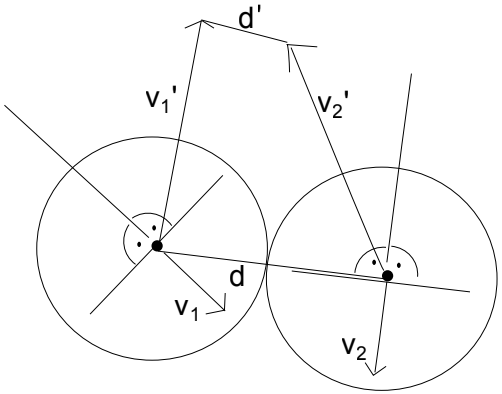

Abbildung 14: Kollisionsbestimmung.

Es stellt sich aber die Frage, ob das Intervall [−90°, +90°], aus dem α_i ausgewählt wird, je nach Situation eingeschränkt werden soll. Man betrachte Abbildung 14: v_1 und v_2 sind

die Richtungsvektoren der Robuttons vor der Kollision, v_1' und v_2' nach der Kollision. d ist der Abstand der Robuttons zum Zeitpunkt der Kollision, d' zu einem späteren Zeitpunkt, nachdem R_1 und R_2 sich in Richtung v_1' und v_2' bewegt hätten. Da $d' < d$, fliegen R_1 und R_2 nach der Kollision wieder aufeinander zu. Das bedeutet, sie kollidieren wieder und müssen direkt noch einmal ihre Richtungsvektoren ändern. Um das zu vermeiden, muss man v_1' und v_2' wiederholt zufällig erzeugen, bis $d' \leq d$ nicht gilt. Diese Methode funktioniert nur, wenn der Betrag der Richtungsvektoren kleiner als der Durchmesser der Robuttons ist.

Kollision von Robutton und Münze

Hier müssen zwei Fälle unterschieden werden:

1. Wenn ein Robutton R bereits eine Münze M trägt, so soll eine Begegnung mit einer anderen Münze als Kollision erkannt werden, sobald der Rand der Münze erreicht wird. Dies ist genau dann der Fall, wenn $d(R,M) \leq 2 \cdot r$.
2. Hat ein Robutton keine Münze, so soll er eine neue Münze erst aufheben, wenn er schon zum Großteil über ihr schwebt, da es anschaulich unplausibel ist, dass die Münze sonst überhaupt aufgehoben werden kann. Mit der Formel $d(R,M) \leq c \cdot 2r$ wird festgelegt, dass ein Robutton eine Münze erst dann aufhebt, wenn der Abstand der Mittelpunkte weniger als $c \cdot 2r$ beträgt. Mit dem Faktor c ($0 \leq c \leq 1$) kann der Abstand zwischen den Extremen „vollständige Überdeckung“ und „Randberührung“ eingestellt werden.

Kollision von Robutton und Tischkante

Auf Kollision zwischen Robuttons und Tischkante wird wie folgt geprüft: Zunächst berechnet man eine bestimmte Anzahl von Punkten auf der Kreislinie des Robuttons. Je mehr Punkte berechnet werden, desto genauer funktioniert die Kollisionserkennung, aber desto höher wird auch der Rechenaufwand. Dann prüft man für jedes Dreieck des Tisches, ob einer dieser Punkte im Dreieck liegt. Ein Punkt P liegt in einem Dreieck $D(A,B,C)$, genau dann wenn:

$$\angle(PA,PB) + \angle(PA,PC) + \angle(PB,PC) = 360°$$

Hierbei bezeichnen PA, PB und PC die Vektoren, welche P und den jeweiligen Punkt des Dreiecks verbinden.

Liegt einer der Randpunkte im Dreieck, so muss noch geprüft werden, mit welcher Seite des Dreiecks der Robutton kollidiert. Dafür wird einfach geprüft, zu welcher Seite des Dreiecks der Punkt am nächsten liegt.

Anschließend wird der Richtungsvektor nach der Reflexion berechnet (vgl. Abbildung 15: v bezeichnet den eingehenden, v' den ausgehenden Richtungsvektor des Robuttons. n ist der Normalenvektor einer Kante des Dreiecks, und es gilt $|n| = 1$. Aus den Eigenschaften des Standardskalarprodukts $*$ ergibt sich:

$$v' = v - 2(v * n)\ n$$

Durch diese Formel gilt Ausfallswinkel gleich Einfallswinkel, und der Betrag des Richtungsvektors ändert sich nicht.

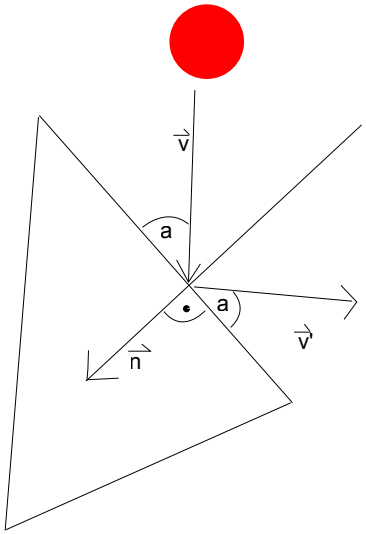

Abbildung 15: Richtung nach der Reflexion.

Beispiele, Beobachtungen

Einige Beobachtungen wurden bei allen Tischen gemacht: Die Münzen tun sich sehr schnell in Gruppen zusammen. Das ist nicht weiter verwunderlich, da eine Münze von einem Robutton nur dort abgelegt werden kann, wo bereits eine Münze ist. Je nach Anzahl der Robuttons geht dieser Prozess schneller oder langsamer. Ist die Anzahl der Robuttons ähnlich groß wie die der Münzen, so wird der Großteil der Münzen meist von den Robuttons über den Tisch spazieren geflogen, anstatt liegen zu bleiben.

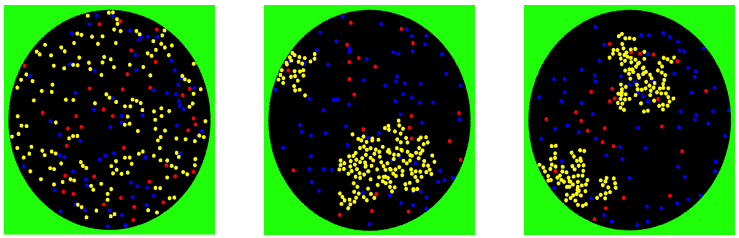

Abbildung 16: Simulationsverlauf bei einem kreisförmigen Tisch.

Kreisförmiger Tisch Die erste Simulation wird mit 100 Robuttons und 200 Münzen der Größe 0.009 gestartet (der Durchmesser des Tisches beträgt 1). Die drei Screenshots in Abbildung 16 zeigen die Entwicklung der Simulation. Beobachtung: Zunächst bilden sich viele kleine Gruppen. Nach einiger Zeit schließen sich einige dieser Gruppen zu größeren Gruppen zusammen, einige Gruppen werden komplett abgetragen. Wenn nur noch 2 oder 3 große Gruppen existieren, ändert sich die Situation kaum mehr. Die Gruppen ändern höchstens noch ein wenig ihre Position, bewegen sich also langsam über den Tisch. Auffällig ist noch, dass diese Gruppen oft am Rand des Kreises „kleben“ und sich nie direkt in der Mitte des Kreises ansiedeln. Mit größeren Münzen und Robuttons (dafür entsprechend weniger) ließ sich ähnliches Verhalten beobachten.

Konkaver Tisch Für einen Tisch mit konkaver Form zeigt Abbildung 17 eine genauere Darstellung des Verlaufs der Simulation. Man kann beobachten, wie sich langsam die Gruppen zusammentun. Wie bereits beim kreisförmigen Tisch fällt auf, dass die Gruppen sich verstärkt am Rand sammeln und dass in der Mitte der freien Fläche keine solchen Gruppen entstehen.

Konvexer Tisch Abbildung 18 zeigt einen Simulationsverlauf für einen Tisch mit konvexer Form. Im Vergleich zum konkaven Tisch fällt auf, dass die Gruppen sich weniger am Rand ansammeln. Davon abgesehen scheinen die Gruppen sich hier wahllos irgendwo zu bilden, ohne die Mitte des freien Bereichs zu meiden.

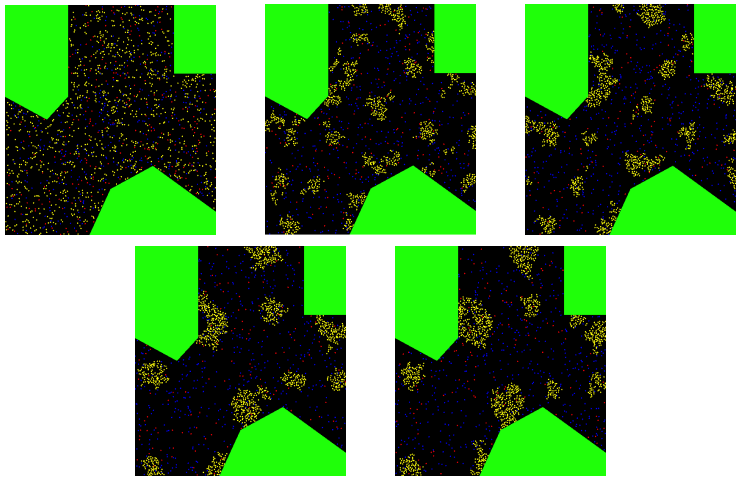

Abbildung 17: Simulationsverlauf bei einem konkaven Tisch.

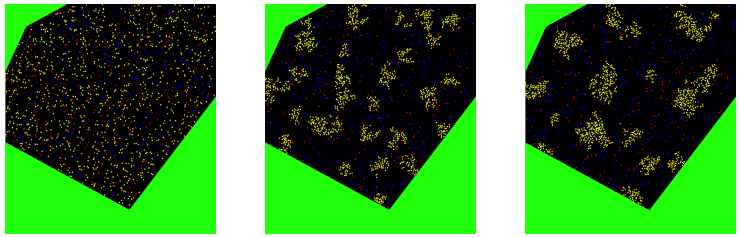

Abbildung 18: Simulationsverlauf bei einem konvexen Tisch.

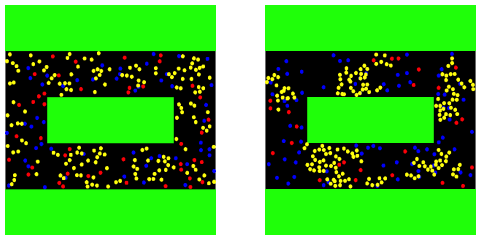

Abbildung 19: Simulationsverlauf bei einem Konferenztisch.

Konferenztisch Eine Simulation auf einem eher langweiligen Tisch in „Konferenzaufstellung“ ist in Abbildung 19 zu sehen. Es ist zu beobachten, dass die Gruppen sich recht zufällig irgendwo bilden. Außerdem ändern sich die Gruppen nach einiger Zeit kaum mehr. Versperrt eine Gruppe erstmal den Durchgang für Robuttons, so bleibt die Struktur der Gruppen größtenteils unverändert.

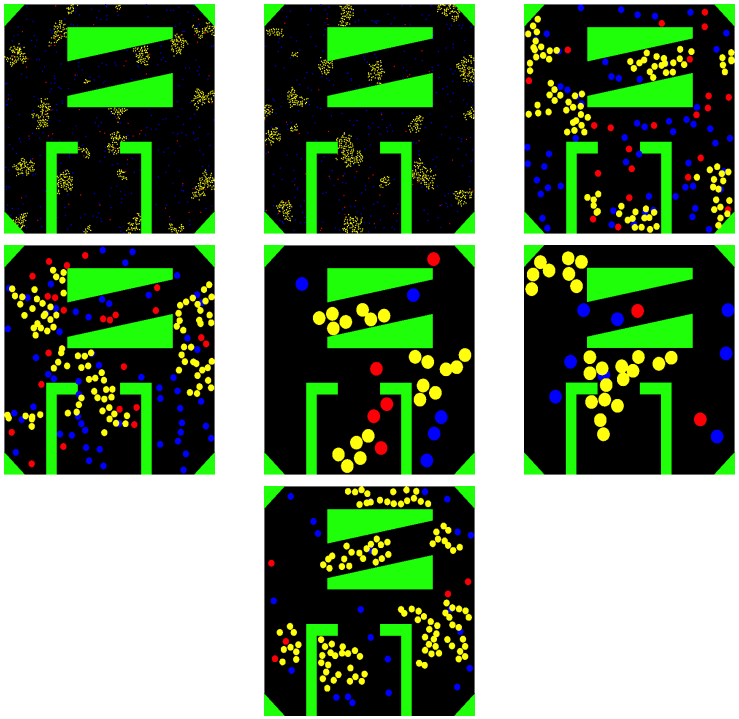

Abbildung 20: Simulationsergebnisse beim ziemlich coolen Tisch.

Ziemlich cooler Tisch Wer würde nicht gern mal an dem in Abbildung 20 gezeigten Tisch speisen? Die einzelnen Bilder zeigen hier verschiedene Simulationsergebnisse. Leider konnte ich bei diesem Tisch keine neuen Regelmäßigkeiten finden. Dafür hat es Spaß gemacht, ihn zu konstruieren.

2.2 Bewertungskriterien

- Fällt die Modellierung des Tisches komplexer aus (wie in dieser Lösung), muss sie nachvollziehbar beschrieben sein. Es ist aber in Ordnung, wenn die geforderten verschiedenen Formen jeweils spezifisch und möglicherweise sehr einfach realisiert werden. Dann sollte aber zumindest beschrieben werden, welche Tischformen möglich sind.
- Die Verfahren zur Erkennung und Behandlung der verschiedenen Kollisionsarten müssen korrekt sein und den Anforderungen der Aufgabenstellung entsprechen (etwa bzgl. der Drehungen der Robuttons). Außerdem sollten die jeweiligen Verfahren nachvollziehbar beschrieben sein, also:
  - ... wann zwei Robuttons kollidieren. Die Berechnungen, die bei einer Kollision nötig sind, sollten erklärt werden. Beim Problem des wiederholten Abprallens von zwei Robuttons wird toleriert, wenn dies unbesprochen hingenommen wird. Eine gute Einsendung müsste das Problem eigentlich erkennen und ansprechen, eine sehr gute Lösung wird das Problem behandeln.
  - ... wann ein Robutton auf eine Münze trifft und wie dies berechnet wird. Es ist nicht nötig, dass hier zwei Varianten unterschieden werden, so wie es in dieser Lösung getan wird. Allerdings muss das Ablegen einer getragenen Münze funktionieren: Münzen dürfen nicht aufeinander gelegt werden.
  - ... wann ein Robutton auf eine Tischkante trifft und wie dies berechnet wird – ggf. angepasst an die Modellierung einzelner Tischformen. Außerdem sollten das Reflexionsverhalten und die zugehörigen Berechnungen beschrieben werden.
- Eine grafische Darstellung (Visualisierung) sollte implementiert sein, die eine Beobachtung der laufenden Simulation (nicht nur zu einem anzugebenden Zeitpunkt) ermöglicht.
- Es sollten mindestens drei Simulationsläufe mit einer Abbildung dokumentiert sein. Zu den Beispielen gehören auch Antworten auf die im Aufgabentext gestellte Frage nach Beobachtungen bzgl. des Simulationsverlaufs.
- Die dokumentierten Simulationsläufe sollten einigermaßen variantenreich sein, also mindestens zwei verschiedene Formen und mindestens zwei unterschiedliche Paare von Anzahlen der Robuttons und Münzen aufweisen.

Aufgabe 3: Logistisch

3.1 Lösungsidee

Die Anzahl der Container, die von einem Standort zu einem anderen Standort an einem Tag transportiert werden müssen, ist gleich der Anzahl der Fahrzeuge, die an diesem Tag mindestens diese Strecke fahren müssen. Wenn die Fahrzeuge den Container an ihrem Ziel abgeliefert haben, können sie an dem Tag keine weitere Fahrt mehr durchführen. Für die erste Teilaufgabe gilt zusätzlich, dass kein Fahrzeug ohne Container fahren darf. Bei der zweiten Teilaufgabe dürfen Fahrzeuge, die nicht für Containerfahrten an dem Tag gebraucht werden, leer zu einem anderen Standort fahren.

Für die erste Teilaufgabe reicht es, die Wochentage der Reihe nach durchzugehen. Dabei wird geschaut, ob an allen Standorten genügend Fahrzeuge vorhanden sind, um die von diesem Standort abgehenden Fahrten durchführen zu können. Wenn nicht genügend Fahrzeuge an dem Standort vorhanden sind, waren (da keine Leerfahrten stattfinden dürfen) am Anfang der Woche noch zu wenig Fahrzeuge an diesem Standort. Dann muss die Anzahl der Fahrzeuge am Anfang der Woche für diesen Standort um die Anzahl der fehlenden Fahrzeuge erhöht werden. Wenn nun alle Tage der Woche so simuliert wurden, hat man die Anzahl der Fahrzeuge, die pro Standort am Anfang der Woche benötigt werden. Die Liste der durchzuführenden Fahrten entspricht dabei der Liste der zu transportierenden Container.

Auch bei der zweiten Teilaufgabe kann man die Wochentage der Reihe nach betrachten. Für jeden Tag betrachtet man an jedem Standort die Gesamtzahl der dort am Tag vorher ankommenden Fahrzeuge mit Containern und die Gesamtzahl der an dem Tag dort abgehenden Fahrten mit Containern. Das Maximum dieser beiden Werte gibt an, wie viele Fahrzeuge an diesem Tag an diesem Standort zur Verfügung stehen (müssen). Die Summe der drei Standortwerte gibt also die Gesamtkapazität für den Tag an. Der maximale der sechs so berechneten Tageswerte bestimmt, wie viele Fahrzeuge am Anfang der Woche für die Transporte gemietet werden müssen. Diese Fahrzeuge können nun so am Anfang der Woche auf die Standorte verteilt werden, dass die Container vom ersten Tag transportiert werden können. Die restlichen Fahrzeuge können beliebig auf die Standorte verteilt werden, da sie sonst nötigenfalls einen Tag bevor sie wirklich benötigt werden, noch eine Leerfahrt machen können, wenn sie noch nicht am benötigten Standort sind.

Mathematisch ausgedrückt wird folgende Rechnung durchgeführt: Sei $\mathrm{an}_{i,j}$ die am Tag i ($1 \leq i \leq 6$) am Standort j ($j \in \{A,B,C\}$) ankommende Zahl von Containern und $\mathrm{ab}_{i,j}$ die am Tag i am Standort j abgehende Zahl von Containern. Zusätzlich sei für einen fiktiven Tag 0 festgelegt, dass $\mathrm{an}_{0,j} = 0$ ist (es kommen keine Container am Tag vor dem ersten Tag an). Dann berechnet sich die Anzahl der benötigten Fahrzeuge für die gesamte Woche aus folgender Formel:

$$\text{Anzahl Fahrzeuge} = \max_{1 \leq i \leq 6} \left(\sum_{j \in \{A,B,C\}} \max\left(\mathrm{an}_{i-1,j}, \mathrm{ab}_{i,j}\right) \right)$$

Zusammengefasst genügt es also bei beiden Aufgabenteilen, einmal linear durch die Liste der durchzuführenden Transporte zu gehen, um die Anzahl der benötigten Fahrzeuge am Anfang der Woche zu bestimmen.

Zur Dokumentation der berechneten Ergebnisse sind folgende Ausgaben hilfreich:

- Zahl der Fahrzeuge an den Standorten am Anfang der Woche,
- durchgeführte Fahrten pro Tag (leer/nicht leer, von/nach)
- Zahl der Fahrzeuge an den Standorten jeweils am Ende des Tages

3.2 Ergebnisse

Für die BWINF-Beispieleingabe liefert unsere Lösung die unten tabellarisch dargestellten Ergebnisse. Für die Variante mit Leerfahrten ist der Wert jeweils zwar minimal, aber die am ersten Tag nicht direkt für eine Containerfahrt gebrauchten Fahrzeuge werden komplett für den Standort C gebucht. Es kann also bei der Variante mit Leerfahrten andere Ergebnisse mit gleicher Anzahl von Fahrzeugen aber anderer Verteilung der Fahrzeuge geben. Für die Standorte A und B ist aber die Mindestzahl von Fahrzeugen angegeben, die am Anfang der Woche dort vorhanden sein müssen:

Beispiel `logistisch1.txt`:

	Standort A	Standort B	Standort C	Summe
ohne Leerfahrten	3	17	25	45
mit Leerfahrten	3	7	26	36

Beispiel `logistisch2.txt`:

	Standort A	Standort B	Standort C	Summe
ohne Leerfahrten	6	14	6	26
mit Leerfahrten	5	3	16	24

Beispiel `logistisch3.txt`:

	Standort A	Standort B	Standort C	Summe
ohne Leerfahrten	6	6	6	18
mit Leerfahrten	6	6	6	18

Beispiel `logistisch4.txt`:

	Standort A	Standort B	Standort C	Summe
ohne Leerfahrten	51	4	12	67
mit Leerfahrten	3	4	47	54

3.3 Bewertungskriterien

- Es soll zumindest für den Fall ohne Leerfahrten erkannt werden, dass linear, also Tag für Tag vorgegangen werden kann. Aufwändigere Verfahren sind hier auf keinen Fall nötig. Aber auch im Fall mit Leerfahrten sollte das Verfahren nicht beliebig aufwändig sein.
- Das bzw. die gewählten Verfahren sollten für die wesentlichen Größen, nämlich die Anzahl der benötigten Fahrzeuge für die drei Standorte (mit oder ohne Leerfahrten), optimale Ergebnisse liefern – also auch nicht zu wenige Fahrzeuge mieten.
- Auch Implementierungs- oder andere Fehler sollten nicht zu falschen Ergebnissen führen.
- Die zweite Teilaufgabe, also eine Berechnung mit Leerfahrten, muss bearbeitet sein.
- Die Beispiele müssen so auf Papier gebracht werden, dass klar wird, welche Fahrten im Laufe der Woche gemacht werden müssen. Dies ist insbesondere bei der zweiten Teilaufgabe notwendig (bei der ersten entsprechen ja die Fahrten den zu transportierenden Containern).
- Zu den vorgegebenen Beispielen (mindestens drei von vier) sollen die Ergebnisse auf Papier dokumentiert sein. Für mindestens ein Beispiel sollen die Berechnungen ausführlicher dokumentiert sein (Programm-Ablauf).

Augensumme $k = x+y$	Augenkombinationen geordnete Paare (x,y)	Wahrscheinlichkeit $\mathbb{P}(W=k)$
2	(1,1)	$\frac{1}{36} = 0{,}02\overline{7}$
3	(1,2), (2,1)	$\frac{2}{36} = 0{,}05\overline{5}$
4	(1,3), (2,2), (3,1)	$\frac{3}{36} = 0{,}08\overline{3}$
5	(1,4), (2,3), (3,2), (4,1)	$\frac{4}{36} = 0{,}11\overline{1}$
6	(1,5), (2,4), (3,3), (4,2), (5,1)	$\frac{5}{36} = 0{,}13\overline{8}$
7	(1,6), (2,5), (3,4), (4,3), (5,2), (6,1)	$\frac{6}{36} = 0{,}16\overline{6}$
8	(2,6), (3,5), (4,4), (5,3), (6,2)	$\frac{5}{36} = 0{,}13\overline{8}$
9	(3,6), (4,5), (5,4), (6,3)	$\frac{4}{36} = 0{,}11\overline{1}$
10	(4,6), (5,5), (6,4)	$\frac{3}{36} = 0{,}08\overline{3}$
11	(5,6), (6,5)	$\frac{2}{36} = 0{,}05\overline{5}$
12	(6,6)	$\frac{1}{36} = 0{,}02\overline{7}$

Tabelle 1: Wahrscheinlichkeiten für die Augensumme zweier Würfel.

Aufgabe 4: Drehzahl

4.1 Lösungsidee

Die Summe zweier Würfel ist die Summe zweier unabhängiger, gleichverteilter, ganzzahliger Zufallsvariablen x und y mit Werten von 1 bis 6: Es sei $k := x+y$ in Tabelle 1.

Eine *Karte* wird durch eine natürliche Zahl von 1 bis 9 dargestellt. Eine *Kartenmenge* ist eine Menge von Karten. Ein Spielzustand wird eindeutig durch die Kartenmenge der noch nicht umgedrehten Karten beschrieben. Diese Karten bezeichnen wir als die *offenen Karten*. Wir schreiben $P(H)$ für die erwartete Punktzahl einer Kartenmenge H, wenn genau die Karten in H offen sind und der Spieler „optimal“ spielt.

Optimal zu spielen bedeutet, immer genau die Züge durchzuführen, die die danach zu erwartende Punktzahl maximieren (unter der Annahme, dass man weiterhin optimal spielt). Ein *Zug* wird mit der Menge der bei einem Zug umgedrehten Karten (eine oder zwei) identifiziert.

Bei offenen Karten H und gewürfelter Augensumme W sind nun alle Züge erlaubt, die eine oder zwei Karten umdrehen, so dass die Summe der Zahlen auf den umgedrehten Karten gleich der Würfelsumme ist. Wir bezeichnen diese Menge der *erlaubten Züge* mit $Z(H,W)$:

$$Z(H,W) := \left\{ K \subseteq H \;\middle|\; 1 \le |K| \le 2 \text{ und } \sum_{j \in K} j = W \right\} \tag{1}$$

Der optimale Zug bei offenen Karten H und Würfelsumme W ist natürlich ein erlaubter Zug K ($K \in Z(H,W)$), und zwar derjenige, der die erwartete Punktzahl in der Situation nach diesem Zug wenn noch die Kartenmenge $H \setminus K$ offen ist) maximiert. Diese maximale erwartete Punktzahl $P(H,W)$ ist (a) entweder die Summe der umgedrehten Karten (bzw. die Summe aller Karten, 45, minus der Summe der offenen Karten), falls es keinen erlaubten Zug mehr gibt,

oder (b) die über alle erlaubten Züge maximale erwartete Punktzahl $P(H \setminus K)$ für die Menge der nach dem jeweiligen Zug noch offenen Karten. Mit den eingeführten Bezeichnungen lässt sich das mathematisch so beschreiben:

$$P(H,W) = \begin{cases} 45 - \sum_{k \in H} k & \text{wenn } Z(H,W) = \{\} \\ \max_{K \in Z(H,W)} P(H \setminus K) & \text{andernfalls.} \end{cases} \tag{2}$$

Nun lässt sich auch die erwartete Punktzahl (im Folgenden gelegentlich auch „Erwartungswert“ genannt) für eine Kartenmenge $P(H)$ berechnen: Die maximalen Punktzahlen $P(H,W)$ der möglichen Würfelsummen W ($k = 1 \ldots 12$) werden mit der Wahrscheinlichkeit der jeweiligen Würfelsumme (siehe Tabelle 1) multipliziert („gewichtet“) und summiert:

$$P(H) = \sum_{k=2}^{12} \mathbb{P}(W=k) \cdot P(H,k) \tag{3}$$

Zu beobachten ist, dass $P(H)$ und $P(H,W)$ gegenseitig voneinander abhängen.

Die **Aufgabenstellung** lässt sich nun mittels dieser Definitionen wie folgt ausdrücken:

1. Gegeben H und W, für welchen Zug $K \in Z(H,W)$ ist $P(H,W) = P(H \setminus K)$ (also: welcher Zug K bringt als erwarteten Punktwert $P(H \setminus K)$ den maximalen Punktwert $P(H,W)$)?
2. Was ist der Wert von $P(\{1,\ldots,9\})$?

Hier ist der Pseudocode zur Berechnung beider Funktionen:

```
1  // Wahrscheinlichkeiten für die Summe zweier Würfel, siehe oben
2  probability[13] = { 0, 0, 1 / 36, ..., 2 / 36, 1 / 36 }
3
4  FUNCTION P(H,k):
5    result = Summe der Karten nicht in H // Game over
6
7    // eine Karte
8    IF (k ∈ H)
9      result = P(H \ {k})
10
11   // zwei Karten
12   FOR a ∈ H
13     b = k − a // zweite Karte
14     IF b ∈ H and b > a
15       IF P(H \ {a,b}) > result
16         result = P(H \ {a,b})
17
18   RETURN result
19
20
21 FUNCTION P(H):
22   result = 0
23   FOR roll = 2 .. 12
24     result = result + probability[roll] * P(H,roll)
25   RETURN result
```

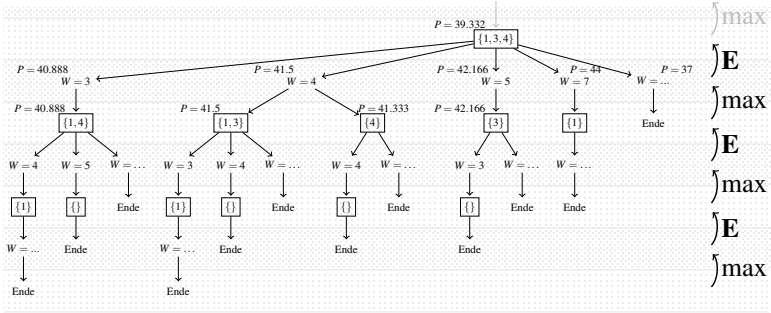

Abbildung 21: Aufrufbaum für $P(\{1,3,4\})$ mit Rückgabewerten.

Wie zu erwarten rufen sich die beiden Funktionen gegenseitig auf. Die Ausführung terminiert, weil:

- $P(H,k)$ die Funktion $P(H)$ nur mit weniger Karten in H aufruft und
- $P(H)$ die Funktion $P(H,k)$ nie mit mehr Karten in H aufruft.

Schließlich lässt sich noch beobachten, dass es für die Funktion P nur $512 = 2^9$ verschiedene Eingabewerte gibt (alle möglichen Teilmengen von $\{1,\ldots,9\}$). Der Rückgabewert kann nach der ersten Berechnung also problemlos gespeichert werden. Auf diese Art und Weise wird $P(H)$ nur höchstens 512 Mal berechnet. Dementsprechend wird $P(H,k)$ höchstens $11 \cdot 512 = 5632$ Mal berechnet.

4.2 Programmdokumentation

Eine Kartenmenge wird als Vektor von $n \leq 32$ Bits dargestellt. Die Datenstruktur ist mit der Klasse `Cards` abstrahiert.

Wenn `a` und `b` Objekte vom Typ `Cards` sind, dann ist `a.With(b)` die Vereinigungsmenge beider Kartenmengen und `a.Without(b)` ist die Kartenmenge `a` ohne die Karten in `b`. Der Ausdruck `a.ContainsCard(c)` gibt **`true`** zurück, wenn die Karte mit dem Wert `c` in `a` enthalten ist. Alle drei Operationen können mit Bitarithmetik implementiert werden.

Die Funktion $P(H)$ aus der Lösungsidee ist als `ExpectedScore` implementiert. Für das Zwischenspeichern der Rückgabewerte wird eine `std::map` verwendet.

Für die Funktion $P(H,W)$ gibt es keine direkte Entsprechung im Quelltext. Stattdessen berechnet `BestNextMove(cards, roll)` die Kartenmenge K, die umgedreht werden sollte (und darf), sodass $P(H \setminus K)$ maximal ist, wenn die Kartenmenge `cards` sichtbar auf dem Tisch

k	$\mathbb{P}(W=k)$	K	$P(\{1,3,4\} \setminus K)$
2	$\frac{1}{36}$	$\{\}$ *(keine möglichen Züge)*	37,000
3	$\frac{2}{36}$	$\{3\}$	40,888
4	$\frac{3}{36}$	$\{1,3\}$	41,500
		$\{4\}$	41,333
5	$\frac{4}{36}$	$\{1,4\}$	42,166
6	$\frac{5}{36}$	$\{\}$ *(keine möglichen Züge)*	37,000
7	$\frac{6}{36}$	$\{2,4\}$	44,000
8…12	$\frac{15}{36}$	$\{\}$ *(keine möglichen Züge)*	37,000

Tabelle 2: Wahrscheinlickeiten und erwartete Punktzahlen nach den Zügen bei $H = \{1,3,4\}$.

liegt und die Augensumme der Würfel `roll` beträgt. Ansonsten folgt die Implementation dem Pseudocode.

4.3 Programm-Ablaufprotokoll

Beispiel

$P(\{2\}) = 43 + \frac{2}{36} = 43{,}0\overline{5}$. Wir lassen unser Programm zweimal laufen, einmal für $W = 4$, dann $W = 2$. Das Ergebnis stimmt:

```
Welche Karten halten Sie in der Hand?
Bitte geben Sie eine durch Leerzeichen getrennte Liste ein:
2
Der Erwartungswert der Punktzahl lautet: 43.0556
Welche Summe wurde gewuerfelt?
4
Das Spiel ist zuende.
```

```
Welche Karten halten Sie in der Hand?
Bitte geben Sie eine durch Leerzeichen getrennte Liste ein:
2
Der Erwartungswert der Punktzahl lautet: 43.0556
Welche Summe wurde gewuerfelt?
2
Spielen Sie die Karte(n) 2.
Der Erwartungswert lautet nun 45.
```

Beispiel

Als nächstes demonstrieren wir, dass die Berechnung von $P(H)$ unserer Formel in Gleichung 3 entspricht. Dazu berechnen wir $P(\{1,3,4\})$ mit allen möglichen Verläufen. Die Ergebnisse sind in Tabelle 2 festgehalten und im Aufrufbaum aus Abbildung 21 veranschaulicht.

Augensumme	Optimaler Zug	Erwartete Punktzahl
2	2	28,8138
3	3	30,1165
4	4	30,5934
5	5	31,7285
6	6	32,5017
7	7	33,3038
8	8	34,6778
9	9	36,1272
10	1 9	33,546
11	2 9	33,2141
12	3 9	34,2641

Tabelle 3: Optimale Züge am Anfang eines Spiels

Es ergibt sich:

$$P(\{1,3,4\}) = \frac{2}{36} \cdot 40{,}8888 + \frac{3}{36} \cdot 41{,}5 + \frac{4}{36} \cdot 42{,}1667 + \frac{6}{36} \cdot 44 + \frac{21}{36} \cdot 37 = 39{,}332$$

In der Tat berechnet unser Programm:

```
Welche Karten halten Sie in der Hand?
Bitte geben Sie eine durch Leerzeichen getrennte Liste ein:
1 3 4
Der Erwartungswert der Punktzahl lautet: 39.332
```

Beispiel

Die erwartete Punktzahl $P(\{1,2,3,4,5,6,7,8,9\})$ – und damit die Antwort auf die zweite Frage in der Aufgabenstellung – wird von unserem Programm wie folgt berechnet:

```
Welche Karten halten Sie in der Hand?
Bitte geben Sie eine durch Leerzeichen getrennte Liste ein:
1 2 3 4 5 6 7 8 9
Der Erwartungswert der Punktzahl lautet: 33.0361
```

Beispiel

Zuletzt geben wir noch an, welche Karten am Anfang eines Spiels umzudrehen sind, wenn eine bestimmte Augensumme gewürfelt wurde (siehe Tabelle 3).

Die optimale Strategie zu Beginn eines Spiels ist recht einfach zu beschreiben: Beträgt die Würfelsumme 9 oder weniger, so dreht man die Karte mit dem Wert, der der Würfelsumme entspricht, herum. Andernfalls wählt man 9 und die entsprechende zweite Karte. Diese Strategie ist aber nicht auf Dauer optimal!

4.4 Alternative Lösungsansätze

Zufallssimulation

Es ist möglich, dass man durch eine Art Simulation ein relativ gutes Ergebnis erzielen kann. Dazu probiert man von der aktuellen Spielsituation eine sehr große Anzahl an Szenarien durch und wählt schließlich das Szenario, das im Schnitt die größte Punktzahl ergab. Dies ändert allerdings nichts an der rekursiven Natur des Problems (man muss ja auch in Zukunft den jeweils besten Zug auswählen). Um keine Laufzeitprobleme zu bekommen, müsste man den Spielverlauf also auf wenige Schritte in die Zukunft beschränken. Dass dies evtl. nicht korrekt ist, muss in der Dokumentation vermerkt sein.

Heuristiken

Verführerisch ist die Wahl informeller Strategien (sogenannte Heuristiken), die meist nur den nächsten Zug in den Blick nehmen. Diese Ansätze ergeben mit hoher Wahrscheinlichkeit suboptimale Ergebnisse. Ein Beispiel:

„Wenn man 2 Karten wählt, wählt man eine davon immer maximal.“ (unvollständige Strategie) versagt z.B. für $H = \{1,\ldots,7\}$ und $W = 11$: Es wählt $\{4,7\}$ als Zug (bedingter Erwartungswert 37,6995), aber $\{5,6\}$ (bedingter Erwartungswert 38,3857) wäre optimal gewesen.

Eine Reihe von Strategien versagt für $H = \{1,2,3,4\}$ und $W = 4$: Sie wählen $\{4\}$ als Zug (bedingter Erwartungswert 40,142), aber $\{1,3\}$ (bedingter Erwartungswert 40,2361) wäre optimal gewesen. Zu diesen Strategien zählen:

- „Man zieht so, daß im nächsten Zug noch möglichst viele der Augensummen mit offenen Karten darstellbar sind[1].“ hat einen Erwartungswert von 32,7925.
- „Man zieht so, daß im nächsten Zug noch möglichst wahrscheinliche Augensummen (siehe Tabelle 1) mit offenen Karten darstellbar sind[2].“ hat einen Erwartungswert von 32,8734.
- „Wenn W gewürfelt wurde und Karte W offen ist, wähle sie. Sonst wähle das Paar Karten, dessen Differenz maximal ist.“ hat einen Erwartungswert von 32,9157.

Zum Vergleich: Unser (optimales) Vorgehen hat einen Erwartungswert von 33,0361.

[1] Bei mehreren Möglichkeiten wird vorzugsweise eine einzelne Karte verwendet, oder bei Paaren das mit der größeren Differenz.
[2] Ebenso.

4.5 Bewertungskriterien

- Das gewählte Verfahren muss, einschließlich seiner Grundlagen (etwa Wahrscheinlichkeitswerte und -berechnungen), nachvollziehbar erklärt sein.
- Die vorgestellte und implementierte Strategie sollte optimal sein, also die gleichen Ergebnisse wie unsere Beispiellösung liefern. Der Einsatz von Zufallssimulationen oder Heuristiken führt hier zu einem Abzug.
- Das Verfahren sollte nicht allzu ineffizient sein. Wenn der gleiche Ansatz wie in der Beispiellösung gewählt wurde, sollte erkannt worden sein, dass sich Ergebnisse zwischenspeichern lassen. Bei besonders ineffizienten Verfahren (etwa unbeschränkt rekursiver Berechnung der durchschnittlichen Punktzahl) sollten zumindest Gedanken zur Laufzeit oder ggf. zum Speicherverbrauch vorhanden sein.
- Die erwartete durchschnittliche Punktzahl (der Erwartungswert) soll, wie in der Aufgabenstellung gefordert, angegeben werden.
- Fehlerhafte, nicht optimale Ergebnisse beruhen häufig auf der Verwendung von Zufallssimulationen oder Heuristiken. Da es hierfür anderweitig Abzüge gibt, gibt es bei fehlerhaften Ergebnissen nur dann einen vollen Abzug, wenn sie stark vom Optimum abweichen.
- Es sollen genügend Beispiele angegeben werden. Ein Spielverlauf mit unterschiedlichen Mengen offener Karten und Würfelsummen ist ausreichend.

Aufgabe 5: Pyramide

Johannes Pieper

5.1 Lösungsidee

Jede Pyramide besteht aus N Ebenen. Bis auf die oberste setzt sich jede Ebene aus drei verschiedenen Sorten von Steinen zusammen: Ecksteine, Seitensteine und volle Steine. Die oberste Ebene mit der Spitze ist die Ebene 1, darunter kommen bei einer Höhe der Pyramide von $N \geq 2$ die Ebenen 2 bis N. Ab der Ebene 2 gibt es genau 4 Ecksteine, in den weiteren Ebenen $k \in \{2,\ldots,N\}$ gibt es $4 \cdot (k-2)$ Seitensteine und $(k-2)^2$ volle Steine.

Kosten

Diese Steine gilt es nun von einem Bauplatz zum anderen zu transportieren, so dass dort wieder eine Pyramide entsteht. Wenn die abgetragenen Steine nicht direkt wiederverwendet werden können, kann ein dritter Bauplatz zur Zwischenlagerung genutzt werden. Dieser muss bei der richtigen Strategie nur für die Spitze verwendet werden. Alle anderen Steine lassen sich direkt zum neuen Bauplatz transportieren und an einer passenden Stelle platzieren.

Hat man diese Strategie nicht gefunden, ist es nötig, sich ein Kostenmaß für die verschiedenen Aktionen zu definieren, das dann zu minimieren ist. Kosten verursachen sowohl der Transport als auch Umlagerungen innerhalb eines Bauplatzes. Die Aufgabenstellung beschreibt nicht, wie diese beiden Punkte gegeneinander zu gewichten sind. Auch wird nicht deutlich, ob bei einem Transport die Umlagerung auf das Floß und zurück an die passende Stelle bereits mitgezählt werden.

Überlegungen zur Strategie

Der erste bzw. letzte Schritt beim Ab- und Aufbau der Pyramide steht immer fest. Dieses ist, aufgrund ihrer besonderen Position, der Transport der Spitze zum Lagerplatz und der spätere Transport zum Bauplatz, wo sie direkt verbaut wird. (Ausnahme ist hier die Pyramide der Größe 1, die nur aus einer einzelnen Spitze besteht. Diese kann direkt transportiert werden. Bei der Größe wäre aber selbst Ramses etwas ungehalten über seinen Architekten.)

Für die anderen Steine lohnt es sich, ihre Eigenschaften genauer zu betrachten, um die passende Strategie zu gewinnen. Volle Steine liegen (ab der zweiten Ebene von unten) immer komplett auf anderen vollen Steinen. Seitensteine liegen (auch ab der zweiten Ebene von unten) immer auf zwei vollen Steinen und auf zwei Seitensteinen. Eckstein liegen (auch ab der zweiten Ebene von unten) auf einem Eckstein, zwei Seitensteinen und einem vollen Stein. Da beim Aufbau nur mit Hilfe von vollen Steinen eine neue Ebene erreicht werden kann, wird dieser Sorte von Steinen die höchste Priorität beim Abbau eingeräumt. Immer wenn ein solcher Stein nicht mehr blockiert ist, wird er vor allen Seiten- und Eckstein zum Neubauplatz

transportiert und dort platziert. Dabei wird – wie in der Aufgabenstellung gesagt – davon ausgegangen, dass ein Stein grundsätzlich nur von oben blockiert wird. Ein Stein ist also frei, sobald keine Steine mehr auf ihm liegen, auch wenn er rundherum noch eingefasst ist.

Die Strategie beim Aufbau besteht darin, immer wieder eine Plattform, bestehend aus einem Eck-, zwei Seiten- und einem vollen Stein, zu schaffen, auf die ein weiterer Eckstein platziert werden kann. Diese Plattformen werden immer reihum um eine Lage erhöht. So entstehen auf den Ecken der Grundfläche vier etwa gleich hohe Pyramiden, die nach und nach zusammenwachsen.

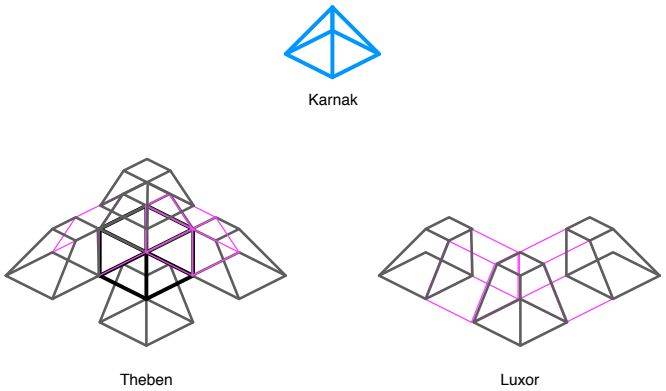

Abbildung 22: Umbau einer Pyramide der Höhe 3.

Deshalb erhalten die Seitensteine die zweite Prioritätsstufe. Nur mit ihrer Hilfe ist es möglich, eine Plattform so zu erhöhen, dass ein Eckstein gesetzt werden kann. Daher wird, wenn kein voller Stein transportiert werden kann, immer zuerst die eine Seite der aktuell zu erhöhenden Plattform mit einem Seitenstein ergänzt und anschließend die andere. Abbildung 22 zeigt einen Zwischenstand beim Umbau einer Pyramide der Höhe 3. Die Spitze liegt in Karnak; zwei Seitensteine (Ebene 3) liegen bereits in Luxor, obwohl in Ebene 2 in Theben noch Ecksteine vorhanden sind. Der einzige volle Stein ist in Theben noch von einem Eckstein blockiert und konnte deshalb noch nicht nach Luxor verbracht werden.

Erst wenn kein andersartiger Stein mehr transportiert werden kann, kommt ein Eckstein an die Reihe. Genau in der gleichen Reihenfolge, wie die Plattformen um eine Lage erhöht werden, werden anschließend auf ihnen die Ecksteine platziert. Dadurch erhält diejenige Plattform als nächstes einen Eckstein, die als erstes in ihrer aktuellen Höhe vollendet wurde und noch keinen Eckstein erhalten hat. Nach dem Abtransport werden, abgesehen von den Ecken der zweiten Ebene, immer ein voller Stein und zwei Seitensteine sowie ein weiterer Eckstein frei. Durch den Abtransport dieser Steine können dann weitere Steine wieder frei werden, die benötigt werden, um die nächste Plattform zu vollenden.

Verteilen der vollen Steine und Seitensteine Bei der Erhöhung einer Plattform bietet es sich an, zuerst eine Seite der aufzubauenden Plattform in der jeweiligen Pyramidenecke komplett um eine Reihe Seitensteine zu ergänzen. Erst danach folgt die andere Seite. Dadurch kann man die Suche nach dem nächsten freien Platz für einen Stein rekursiv machen, da die Platzierung eines Seitensteins immer vom Seitenstein quer darunter abhängig ist. (Im Handexperiment mit LEGO-Steinen hat es sich gezeigt, dass man bei der Platzierung der Seitensteine für eine Plattform auch immer erst eine Ebene voll machen kann.) Die vollen Steine müssen dabei so gesetzt werden, dass so schnell wie möglich ein Untergrund für den nächsten Seitenstein erzeugt wird. Dadurch werden an die Seite einer Plattform immer dreiecksmäßig volle Steine angelagert – je nach Strategie ein oder zwei Dreiecke gleichzeitig. Erst zum Abschluss einer Plattform werden die vollen Steine, die genau gegenüber den Ecksteinen liegen, bis zur oberen Plattformebene hochgezogen.

5.2 Umsetzung

Für die Umsetzung in einem Programm müssen Überlegungen getätigt werden, wie die beiden Pyramiden intern gespeichert werden. Außerdem fordert die Aufgabenstellung einen Transport- und Bauplan, der den Arbeitern zur Verfügung gestellt werden soll. Dazu wird ein passendes Ausgabeformat gefordert. Optional ist eine graphische Ausgabe möglich, die an dieser Stelle nicht weiter behandelt wird.

Interne Speicherung

Der jeweilige Zustand der Pyramiden lässt sich für jede Ebene in einem $k \cdot k$ Feld speichern, in dem jedes einzelne Element den Zustand eines Steins repräsentiert. Bei der abzubauenden Pyramide bietet sich an für jeden Stein zu speichern, durch wie viele Steine er blockiert wird. Erreicht er den Wert 0, so kann er in eine entsprechende Abbauqueue übernommen werden, die es jeweils für volle, Seiten- und Ecksteine gibt. Nach dem Abbau bekommt er den Wert -1 und bei den Steinen der Ebene tiefer, die er bisher mit blockiert hat, wird die Zahl entsprechend verringert.

Ähnlich wird bei der zu bauenden Pyramide gearbeitet. Doch wird hier gespeichert, wie viele Steine noch auf der darunter liegenden Ebene fehlen, bis der Stein platziert werden kann. Dadurch muss nicht für jeden einzelnen Stein überprüft werden, ob noch alle vier oberen oder unteren Steine vorhanden sind.

Bauplan

Die Ausgabe der Schritte für die Arbeiter ist ein wichtiger Bestandteil der Aufgabenstellung. In diesen Beispielen werden bei den einzelnen Schritten die Stellen der Steine durch ein dreidimensionales Koordinatensystem (k, x, y) identifiziert. Die erste Koordinate gibt jeweils die Stufe der Pyramide von oben an. Mit den anderen beiden Koordinaten wird die Lage innerhalb der Ebene spezifiziert. Dabei wird zur Vereinfachung mit $(k, 0, 0)$ immer von einer bestimmten

Ecke ausgegangen. Die Ablage auf dem Lagerplatz erfolgt ohne Koordinaten. Eine einfache Nummerierung der Steine ist nicht hilfreich, da laufende Nummern nichts über die Lage der Steine aussagen.

Der Umbau einer Pyramide mit Höhe $N = 2$ sieht dann wie folgt aus:

```
Spitze von (1, 0, 0 ) auf Lagerplatz
Eckstein von (2, 0, 0 ) nach (2, 0, 0 )
Eckstein von (2, 1, 0 ) nach (2, 1, 0 )
Eckstein von (2, 1, 1 ) nach (2, 1, 1 )
Eckstein von (2, 0, 1 ) nach (2, 0, 1 )
Spitze vom Lagerplatz auf (1, 0, 0 )
```

Setzen der Steine

Wie die Steine generell beim Aufbau gesetzt werden sollen, ist bei der Strategie beschrieben worden. Bei der Implementierung ist dabei zu beachten, dass die zu erhöhenden Plattformen bei unterschiedlichen x- und y-Koordinaten liegen. Damit unterscheiden sich auch die Richtungen, in denen nach freien Plätzen für das Setzen der Steine gesucht werden muss. Diese unterschiedlichen Koordinaten und Richtungen können im Programm durch Aufruf der Methoden mit unterschiedlichen Parametern realisiert werden.

Außerdem muss für jede Sorte von Steinen einzeln intern gespeichert werden, an welcher Plattform diese noch zur Erhöhung benötigt werden. Dieses ist der Fall, da es vorkommen kann, dass z. B. volle Steine bereits an einer anderen Plattform angelegt werden, obwohl an der vorherigen noch Seitensteine fehlen.

5.3 Ergebnisse

Die Pläne zum Umbau sind entsprechend lang, wie bereits das Beispiel für die Pyramide der Höhe $N = 4$ zeigt (31 Zeilen; der Umbauplan für Höhe 7 hat 141 Zeilen):

```
Spitze von (1, 0, 0 ) auf Lagerplatz
Eckstein von (2, 0, 0 ) nach (4, 0, 0 )
Eckstein von (2, 1, 0 ) nach (4, 3, 0 )
Seitenstein von (3, 1, 0 ) nach (4, 0, 1 )
Eckstein von (2, 1, 1 ) nach (4, 3, 3 )
Seitenstein von (3, 2, 1 ) nach (4, 1, 0 )
Eckstein von (2, 0, 1 ) nach (4, 0, 3 )
voller Stein von (3, 1, 1 ) nach (4, 1, 1 )
Seitenstein von (3, 0, 1 ) nach (4, 2, 0 )
Seitenstein von (3, 1, 2 ) nach (4, 3, 1 )
Eckstein von (3, 0, 0 ) nach (3, 0, 0 )
voller Stein von (4, 1, 1 ) nach (4, 2, 1 )
Seitenstein von (4, 1, 0 ) nach (4, 3, 2 )
Seitenstein von (4, 0, 1 ) nach (4, 2, 3 )
Eckstein von (3, 2, 0 ) nach (3, 2, 0 )
voller Stein von (4, 2, 1 ) nach (4, 2, 2 )
Seitenstein von (4, 2, 0 ) nach (4, 1, 3 )
Seitenstein von (4, 3, 1 ) nach (4, 0, 2 )
```



```
Eckstein von (3, 2, 2 ) nach (3, 2, 2 )
voller Stein von (4, 2, 2 ) nach (4, 1, 2 )
Seitenstein von (4, 3, 2 ) nach (3, 0, 1 )
Seitenstein von (4, 2, 3 ) nach (3, 1, 0 )
Eckstein von (3, 0, 2 ) nach (3, 0, 2 )
voller Stein von (4, 1, 2 ) nach (3, 1, 1 )
Seitenstein von (4, 0, 2 ) nach (3, 2, 1 )
Seitenstein von (4, 1, 3 ) nach (3, 1, 2 )
Eckstein von (4, 0, 0 ) nach (2, 0, 0 )
Eckstein von (4, 3, 0 ) nach (2, 1, 0 )
Eckstein von (4, 3, 3 ) nach (2, 1, 1 )
Eckstein von (4, 0, 3 ) nach (2, 0, 1 )
Spitze vom Lagerplatz auf (1, 0, 0 )
```

5.4 Bewertungskriterien

- Die Strategie für den Auf- und Abbau muss nachvollzierhbar erläutert werden. Dabei muss auch auf die unterschiedliche Behandlung der verschiedenen Steinsorten eingegangen werden.
- Nur die Spitze sollte zwischengelagert werden. Bei mehr zwischengelagerten Steinen muss deutlich werden, warum dieses geschieht – etwa wegen eines speziellen Kostenmodells, das Zwischenlagerungen begünstigt. Eine Strategie, die mehr als die Spitze zwischenlagert und das normale Kostenmodell verwendet (Transporte zählen gleich viel wie Umlagerungen), ist nicht optimal, weil sie unnötig viele Transporte verursacht.
- Auch unnötige Umlagerungen führen beim normalen Kostenmodell zu suboptimalen Ergebnissen.
- Das Programm sollte die gewählte Strategie korrekt implementieren. Werden die Plattformen nach und nach hochgezogen, müssen die Richtungen korrekt angegeben bzw. berechnet werden.
- Eine graphische Ausgabe ist schön, aber nicht gefordert. Auf jeden Fall muss aber ein passender und verständlicher Umbauplan für die Arbeiter erstellt werden.
- Bis zu einer Höhe von $N = 4$ lässt sich das Programm sehr einfach implementieren, daher sollten auch größere Beispiele vorliegen ($N > 4$). Für mindestens eine vernünftige Pyramidengröße ist ein Umbauplan zu dokumentieren; für weitere, größere Pyramiden genügen Auszüge bzw. Angaben zu den nötigen Transporten und Umlagerungen (im einfachsten Fall genügt die Länge des Umbauplans).

Übrigens: Eine ausführliche Mittagspause am Nilufer ist den Arbeitern nicht erlaubt, um mal eben ein Boot mit einem vollen Stein ein Boot mit einem Seitenstein überholen zu lassen ...

Aus den Einsendungen: Perlen der Informatik

Allgemeines

Einsendung zur Informatik-Olympiade 2010

Copyreight *Copyrnine, Copyrten, ...*

Die Klassifizierung von Pascal als gängige Programmiersprache ist nur mit Kirchturmdenken zu erklären, sagt Opa.

Die Methode „Schleife“ ist ein rekursiver Durchlauf.

Horoskope

Es rennt alle Sternzeichen in einer Schleife durch.

Beim „Liebessatz“ verwendet das Programm Sätze, die zu einer Mädchenzeitschrift passen.

Horoskopsätze:

Es wird in der Liebe Engpässe geben.

Denke nicht zu viel über die Sorgen nach, sondern erlebe die besonderen Momente mit deinem Computer.

Informartik

Künstlerische Absichten:

Durch den schnellen Verlauf des Lebens entsteht ein Strudel, eine Verwehung der Zeit.

Das Bild symbolisiert das Ende der schulischen Ausbildung und den Neuanfang in eine neue, interessante Welt der Arbeit. *Abwarten ...*

Ich wähle den Namen meiner Freundin (ja, auch begeisterte Informatiker sollen sowas ja gelegentlich haben).

Völlig irrational und daher unberechenbar und angsterregend sind die Knochen, die scheinbar wahllos an den Tentakeln hängen und wie Reste von Opfern, die von den Tentakeln bereits erwischt wurden, erscheinen.

Robuttons

Robuttencollision

Das erste Experiment befasst sich mit der Verteilung des Cents in einem Kreis mit Kanten.

Für die x- und y-Richtung soll jeweils eine Rechnung durchgeführt werden, um die x- und z-Richtung zu berechnen, die der Roboter versetzt werden muss, wenn er in eine bestimmte Alpha-Richtung fährt.

Logistisch

Containerrumfahrfirma

... und startet mit so wenigen Schiffen wie möglich.

... behilft sich das Programm mit der Bezeichnung „gelangweilte LKWs“.

... muss danach ermittelt werden, ob mehr Autos abgefahren sind als überhaupt vorhanden waren.

... werden so viele Container, wie gebraucht werden, zu dem minderbemittelten Ort gefahren.

Es wird deutlich, dass man mit etwas angewandter Informatik in Form eines Algorithmus geschickt planen kann, um Kosten einzusparen.

Drehzahl

Würfelsumme = $n_1 \cdot 6 + n_2$

Die innere Schleife ist für die Kontrolle der einzelnen Würde zuständig.

Pyramide

Am Ende des Umbauplans: Im Namen des Pharaos: Vorwärts, ihr Sklaven!

Namen für Karnak in ein und derselben Einsendung: Karanke, Karanka, Karannka

... da der Pharao seinerzeit ein Gesetz erließ, welches besagt, dass eine Pyramide eine Höhe von maximal 1800 Phuss (ca. 3455600 Steine) haben darf. *Gut, dass der Pharao damals schon an die Vermeidung von Integer overflows gedacht hat!*