Dokumentation der Praktischen Arbeit

Zur Prüfung zum mathematisch-technischen Softwareentwickler

Markus Faßbender
Prüflingsnummer: 142-59218
16.05.2014

Inhalt

1 Aufgabenanalyse

1.1 Analyse

Auf einem rechteckigen Feld soll ein Roboter den Boden versiegeln. Damit die frische Versiegelung nicht sofort wieder zerstört wird, muss eine Route über alle Parzellen (sofern möglich) gefunden werden, die jede Parzelle genau einmal abläuft.

Je nach Hindernissen kann ein Feld nicht vollständig abgelaufen werden. Dann soll eine Route gefunden werden, die möglichst viele Parzellen versiegelt.

Die abzufahrenden Parzellen werden als Routenplan in einer Liste gespeichert, die anschließend abgearbeitet wird. Der Roboter darf auf der letzten Parzelle stehen bleiben.

Es gibt Hindernisse, die über mehrere benachbarte Parzellen gehen können und die der Roboter nicht überwinden kann. Diese können auch so angeordnet sein, dass der Roboter umzingelt ist oder diese wie eine Wand vor sich hat.

Vorgegeben ist eine Uhrzeiger-Strategie, die eine vollständige oder bestmögliche Route findet. Dabei wird auf jeder Parzelle entschieden, in welche Richtung fortgefahren wird. Dabei sind natürlich einige Richtungen nicht sinnvoll und/oder nicht möglich.

Das Verfahren ist ein Backtracking-Algorithmus, der zur letzten Parzelle zurückspringt, wenn von einer Parzelle keine Richtung mehr möglich ist. Auf der vorherigen Parzelle wird versucht eine andere Richtung einzuschlagen usw. bis eine Lösung gefunden wurde oder keine vollständige Lösung vorhanden ist.

1.2 Datenformat und Eingabe

Die Parzellen können in einem zweidimensionalen Array abgespeichert werden. Als Typ bietet sich *char* an, weil hier einzelne ASCII-Zeichen eingetragen werden (z.B. ^, <, v, >, S, Z, H).

Alternativ könnte man auch einen eigenen Typen (eine Klasse oder ein Enum) verwenden. Das Array kann nach dem Einlesen der ersten Zeile alloziert werden, weil die Größe dann fest ist und sich nicht mehr ändert. Daher muss hier auch nicht unbedingt mit einer ArrayList o.ä. gearbeitet werden.

Die Routenliste hingegen sollte als List implementiert werden, weil anfangs nicht die Anzahl der nötigen Schritte bekannt ist.

Die beiden Hauptvariablen sollten zusammen als Klasse gekapselt werden, um einen einfachen und sauberen Zugriff zu ermöglichen.

Die Daten werden als Datei eingelesen und zeilenweise ausgewertet. Kommentarzeilen können übersprungen werden. Als erstes wird die Anzahl der Felder ausgelesen, die für das Anlegen des Arrays benötigt wird. Danach kommt die Startposition und dann kommen in mehreren Zeilen die Hindernisse. Die Startposition und Hindernisse können dann direkt in das Datenobjekt aufgenommen werden.

1.3 Ausgabeformat

Das Augabeformat besagt, dass die Lösungen nach folgendem Muster ausgegeben werden sollen:

1. Die eingelesenen Anfangsbedingungen
2. Die Lösung mit der Uhrzeiger-Strategy und ihr Routenplan
3. Die Lösung mit der eigenen Strategy und ihr Routenplan
4. Statistik über den Zustand der Parzellen (versiegelt, nicht versiegelt, durch Hindernis besetzt)

Wichtig ist dabei, dass der Routenplan nur aus Parzellen besteht, an denen ein Richtungswechsel stattfindet. So wird die Route etwas kürzer in der Ausgabe und man muss nicht bei 10x10 Parzellen einhundert Schritte ausgeben.

Als Beispiel ist in der Aufgabenstellung folgendes gegeben:
Startpunkt (1,1)
 1 2 3 4 5
1 S
2
3
4
5

Uhrzeiger-Strategy
 1 2 3 4 5
1 > > > > v
2 v < v < v
3 v ^ v ^ v
4 v ^ v ^ v
5 Z ^ < ^ <

Routenplan:
1,1 / 1,5 / 5,5 / 5,4 / 2,4 / 2,3 / 5,3 / 5,2 / 2,2 / 2,1 / 5,1

Meine Strategie
 1 2 3 4 5
1 v v < < <
2 v v v < ^
3 v v Z ^ ^
4 v > > ^ ^
5 > > > > ^

Routenplan:
1,1 / 5,1 / 5,5 / 1,5 / 1,2 / 4,2 / 4,4 / 2,4 / 2,3 / 3,3

zu versiegelnde Parzellen: 25
Hindernisparzellen: 0
versiegelte Parzellen: 25
nicht versiegelte Parzellen: 0

1.4 Anforderung an das Gesamtsystem

Das Programm kann in ein Model, eine View und einen Controller unterteilt werden. Der Controller liest über die View die Daten ein, speichert diese im Model und führt die Berechnungen durch. Anschließend werden die Ergebnisse wieder über die View ausgegeben.

Um beide Algorithmen durchführen zu können, muss ein Grundgerüst an Software bestehen:

1. Interaktion mit Dateien, um Eingabe und Ausgabe zu benutzen
2. Ein Datenmodell, das die Daten vorhält und Zugriffe regelt. Die sind die Strategien mit ihren jeweiligen Areas.
3. Die Logik des Backtrackings. Diese wird vom Controller ausgeführt und benötigt View und Model als Kommunikationspartner.

Deshalb lässt sich mein Programm grob in diese Untermodule einteilen. Die komplette Durchführung besteht aus:

- Einlesen
- Speichern der Daten
- Berechnen der Lösungen (ggf. mehrere, hier 2)
- Ausgabe der Daten

Dabei ist es wichtig mögliche Fehler zu behandeln, um Abstürze und unerwartetes Verhalten zu vermeiden. Außerdem wird eine aussagekräftige Meldung ausgegeben, welcher Fehler aufgetreten ist. Die Robustheit des Programms wird mit Testfällen überprüft.

1.5 Grenzfälle

Als Grenzfall gibt es drei Szenarien:

1. Minimal-Szenario
   Die Fläche ist von der Größe 1x1. Der Startpunkt muss also gleich dem Ziel sein.

2. Maximal-Szenario
   Die Fläche ist von der Größe 10x10. Hier ist die Laufzeit des Algorithmus maximal (falls keine Hindernisse vorhanden sind).

3. Worst-Case-Szenario
   Die Fläche ist von der Größe 10x10, aber zwei Hindernissen blockieren eine Parzelle in einer Ecke. Weil es dann keine keine vollständige Lösung gibt, müssen alle möglichen durchlaufen werden. Dies sind 97 abzuarbeitende Parzellen, weil auf 2 Hindernisse stehen und eine nie erreicht werden kann.

Es wäre theoretisch möglich, dass eine Fläche komplett voll mit Hindernissen steht, sodass nur der Startpunkt frei ist. Dies kann allerdings wieder auf den Minimalfall zurückgeführt werden.

1.6 Fehlerfälle

Die auftretenden Fehler können in grob in drei Kategorien aufgeteilt werden: technische, syntaktische und semantische Fehler. Technische Fehler liegen vor, wenn die angegebene Datei nicht vorhanden ist oder keine Zugriffsrechte vorhanden sind. Syntaktische Fehler treten auf, wenn die Eingabedatei die Formatvorgaben nicht korrekt einhält. Bei semantischen Fehlern sind fehlerhafte Werte (z.B. 0 als Reihe) enthalten.

Durch die Analyse der Aufgabenstellung und Eingabeanforderung ergeben sich folgende Fehlerfälle:

Technische Fehlerfälle

- Keine Eingabe-Datei vorhanden
- Keine Zugriffsrechte

Syntaktische Fehlerfälle

- Es fehlt die Startzeile oder die Anzahl der Felder
- Es werden keine Leerzeichen als Trennzeichen benutzt
- Es werden Fließkommazahlen eingegeben
- Ein Hindernis ist unzureichend definiert

Semantische Fehlerfälle

- Es sind mehrere Abmessungen oder Startparzellen in einer Zeile gegeben
- Es sind weniger als 1 oder mehr als 10 Parzellen in jede Richtung gegeben
- Der Startpunkt liegt außerhalb des Feldes
- Die Hindernisse liegen außerhalb des Feldes
- Der Startpunkt ist auch ein Hindernis
- Zwei oder mehr Hindernisse überlagern sich

2 Verfahrensbeschreibung

2.1 Mein Algorithmus

Meine Strategie wird ebenfalls das Backtracking-Verfahren benutzen, um eindeutig entscheiden zu können, ob es überhaupt eine gültige, vollständige Lösung gibt. Auf jeder Parzelle wird versucht zunächst die vorherige Richtung beizubehalten. Sollte dies nicht möglich sein, wird gegen den Uhrzeigersinn eine andere Richtung probiert. Falls z.B. die letzte Richtung „nach links“ war, wird danach „nach unten“, dann „nach rechts“ und dann „nach oben“ probiert. Beim Start wird mit „nach oben“ begonnen. Davon erhoffe ich, dass möglichst selten die Richtung geändert werden muss und so ein relativ kurzer Routenplan erstellt wird.

Wenn mein Algorithmus nicht weiter in die Richtung gehen kann, muss entschieden werden, ob alle Felder (außer den Hindernissen) abgearbeitet wurden. Falls ja terminiert er, sonst muss ein Schritt zurückgegangen werden. Dies wird solange wiederholt, bis eine gültige Lösung erreicht wurde oder alle Felder in alle Richtungen abgelaufen worden sind.

Über das Backtracking-Verfahren findet man in jedem Fall eine Lösung, wenn es eine gibt. Für den Fall, dass es keine komplette Lösung gibt, gehe ich so vor:

- Bei jedem Schritt wird der Punkt in die Routenliste eingetragen
- Übersteigt die Anzahl der durch die Routenliste abgedeckten Parzellen die durch die bisher beste gespeicherte Routenliste, wird die aktuelle Routenliste als neue Beste gespeichert.
- Falls von einem Punkt ein Rückschritt erfolgen muss, muss dieser aus der aktuellen Routenliste entfernt werden
- Unabhängig davon, ob der Algorithmus mit oder ohne vollständige Lösung terminiert, erreicht die gespeicherte beste Routenliste die bestmögliche Versiegelung von Parzellen.

Im Worst-Case-Szenario hat dieser einer Algorithmus daher eine Laufzeit in der Klasse $O(4\hat{}k)$, wobei k die Anzahl aller Parzellen ist. In dem unter Grenzfälle definierten Worst-Case-Szenario müssen also $4^{97} = 2{,}51 * 10^{58}$ Schritte gemacht werden.

Ein Beispiel der beiden Algorithmen:
; Abmessung der Fläche
3 3
; Startparzelle
2 2
; Eckparzelle der Hindernisse

Startparzelle

	S	

Uhrzeiger-Strategie

Z	>	v
^	^	v
^	<	<

Meine Strategie

v	<	Z
v	^	^
>	>	^

2.2 Grenzfälle

Die oben genannten Grenzfälle werden wie folgt behandelt:

Die Fläche besteht aus einer einzigen Parzelle.
Das Programm wird normal ausgeführt.

Die Fläche besteht aus 10x10 Parzellen.
Das Programm wird normal ausgeführt.

Die Fläche besteht aus 10x10 Parzellen mit einer nicht erreichbaren Ecke.
Das Programm wird normal ausgeführt, aber wahrscheinlich nicht in absehbarer Zeit terminieren.

2.3 Fehlerfälle

Die oben genannten Fehlerfälle werden wie folgt behandelt:

Es ist keine Eingabedatei vorhanden.
Das Programm gibt eine Fehlernachricht aus und beendet sich.

Es sind nicht genügend Rechte im Dateisystem vorhanden.
Das Programm gibt eine Fehlernachricht aus und beendet sich.

Es fehlt die Startzeile oder die Anzahl der Felder.
Das Programm schreibt einen Fehler in die Ausgabedatei und beendet sich.

Es sind weniger als 1 oder mehr als 10 Parzellen in jede Richtung gegeben.
Das Programm schreibt einen Fehler in die Ausgabedatei und beendet sich.

Es werden Whitespaces[1] als Trennzeichen benutzt.
Das Programm wird normal ausgeführt.

Es werden ungültige Trennzeichen benutzt.
Das Programm schreibt einen Fehler in die Ausgabedatei und beendet sich.

Es werden Fließkommazahlen eingegeben.
Das Programm schreibt einen Fehler in die Ausgabedatei und beendet sich.

Ein Hindernis wird nicht ausreichend definiert.
Das Programm wird ohne das Hindernis ausgeführt, hängt aber eine Warnung an die Ausgabe.

[1] Java erkennt die Unicode-Whitespaces: https://de.wikipedia.org/wiki/Leerraum#Unicode

Es sind mehrere Abmessungen oder Startparzellen in einer Zeile gegeben.
Das Programm schreibt einen Fehler in die Ausgabedatei und beendet sich.

Der Startpunkt liegt außerhalb des Feldes.
Das Programm schreibt einen Fehler in die Ausgabedatei und beendet sich.

Der Startpunkt ist auch ein Hindernis.
Das Programm schreibt einen Fehler in die Ausgabedatei und beendet sich.

Zwei oder mehr Hindernisse überlagern sich.
Das Programm wird normal ausgeführt.

2.4 Gesamtsystem

2.4.1 Main-Funktion

Die Mainfunktion liest die benötigten Pfade aus den Übergabeparametern aus und initialisiert damit den Controller. Anschließend startet sie den Controller. Weitere Operationen erfolgen im *Controller*.

2.4.2 Model

Die Daten werden in der Klasse *Model* verwaltet. Dabei kann nicht direkt auf die Daten, sondern nur über die Schnittstellen-Methoden auf diese Zugegriffen werden. Dies erlaubt eine leichte Handhabung für Erweiterungen und hält die Kommunikation übersichtlich.

2.4.3 View

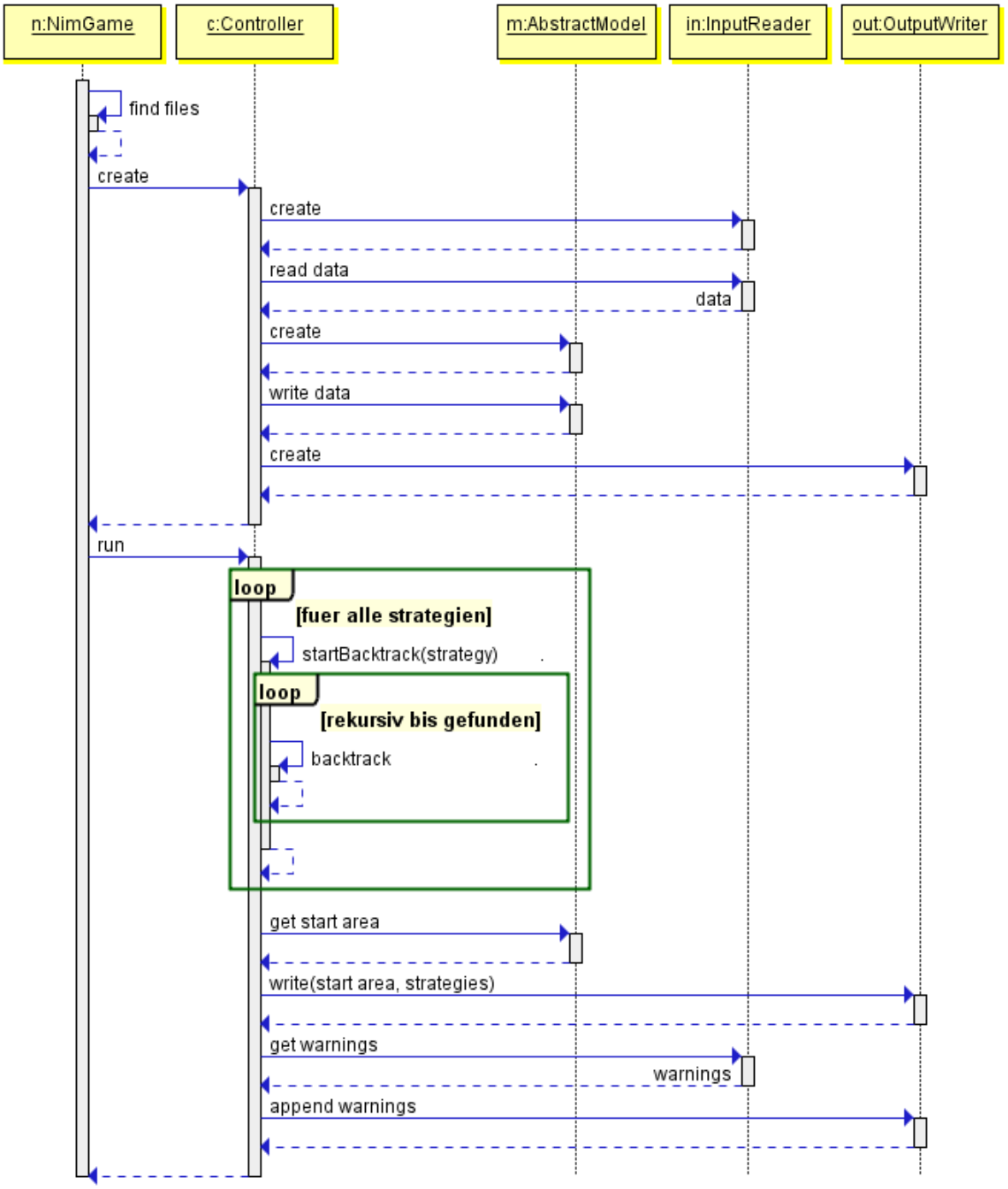
Zur View gehören sowohl Eingabe als auch Ausgabe. Hier werden also Methoden zum Einlesen von Daten (Klasse *InputFileReader*) und zur Ausgabe (Klasse *OutputFileWriter*) von der Statistik bereitgestellt. Auch Warnungen und Fehler können ausgegeben werden.

2.4.4 Controller

Der Controller beinhaltet die eigentliche Logik des Programms. Dieser startet das Einlesen aus der Datei, schreibt die Daten in das Model und führt alle Berechnungen und Durchläufe aus. Abschließend wird das Ergebnis in die Ausgabe geschrieben.

Die Strategien werden hier verwaltet und ausgeführt, sodass der gesamte Prozess abgekapselt ist.

2.5 Gesamtablauf – Sequenzdiagramm

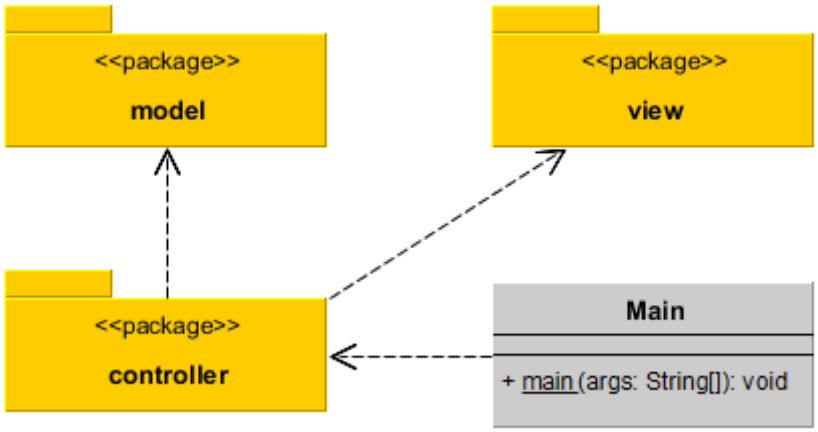


3 Programmbeschreibung

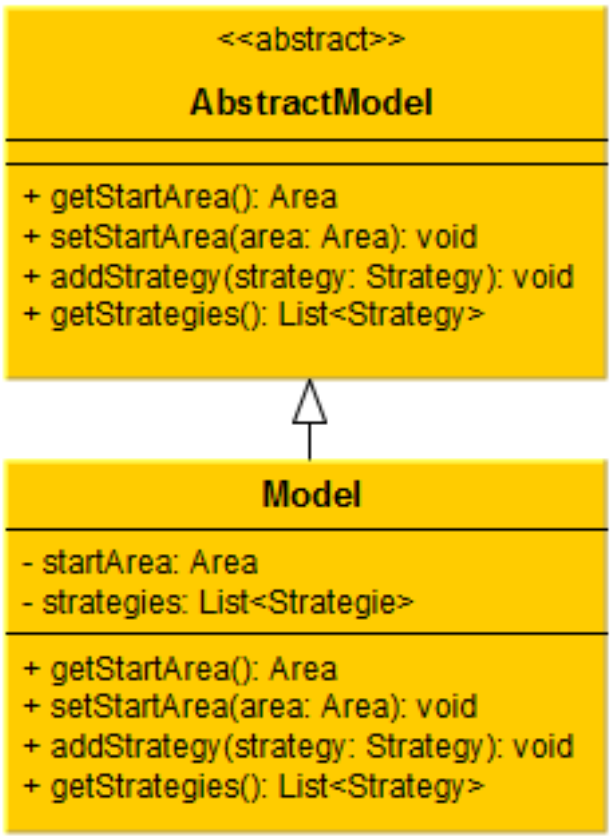
3.1 Pakete

Das Programm wird in drei Pakete unterteilt: Model, View und Controller. Dabei sind Teile des Controllers, nämlich die Strategien, als eigenes Unterpaket zusammengefasst. Für die Model und die View wurden abstrakte Klassen als Schnittstelle definiert. Diese bieten alle wichtigen Funktionen und werden jeweils in einer Unterklasse konkrete implementiert. Dadurch bleibt das Programm leicht erweiterbar.

Die UML-Gesamtübersicht aller Klassen und Methoden kann ich aufgrund der hohen Auflösung und Bildgröße leider nicht hier darstellen. Diese ist unter folgendem Pfad abgebildet: *diagrams/uml/UMLDetailed.png*.



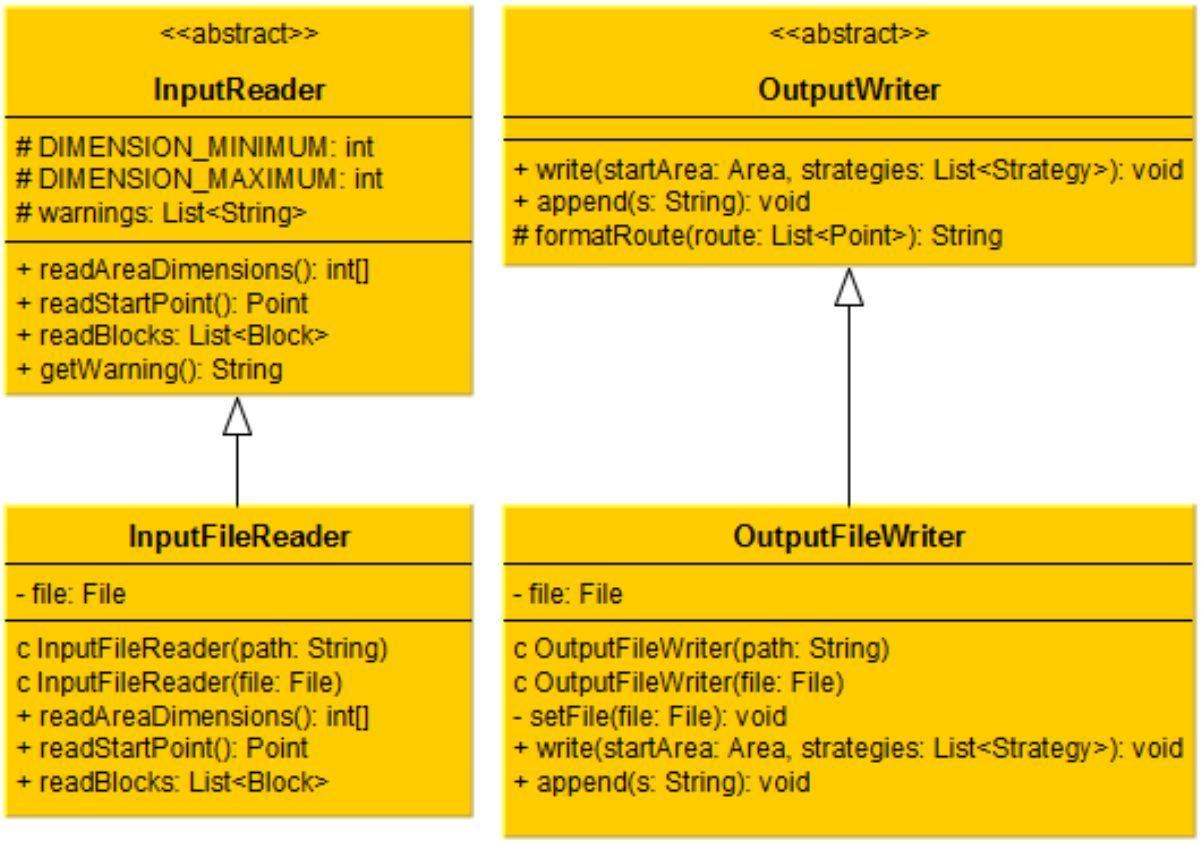
3.1.1 Model-Klassen



AbstractModel
Definiert eine Schnittstelle über die alle benötigten Daten gespeichert und geholt werden können.

Model
Konkrete Implementierung eines Models mit Variablen. Die Daten werden also im Arbeitsspeicher gehalten.

3.1.2 View-Klassen



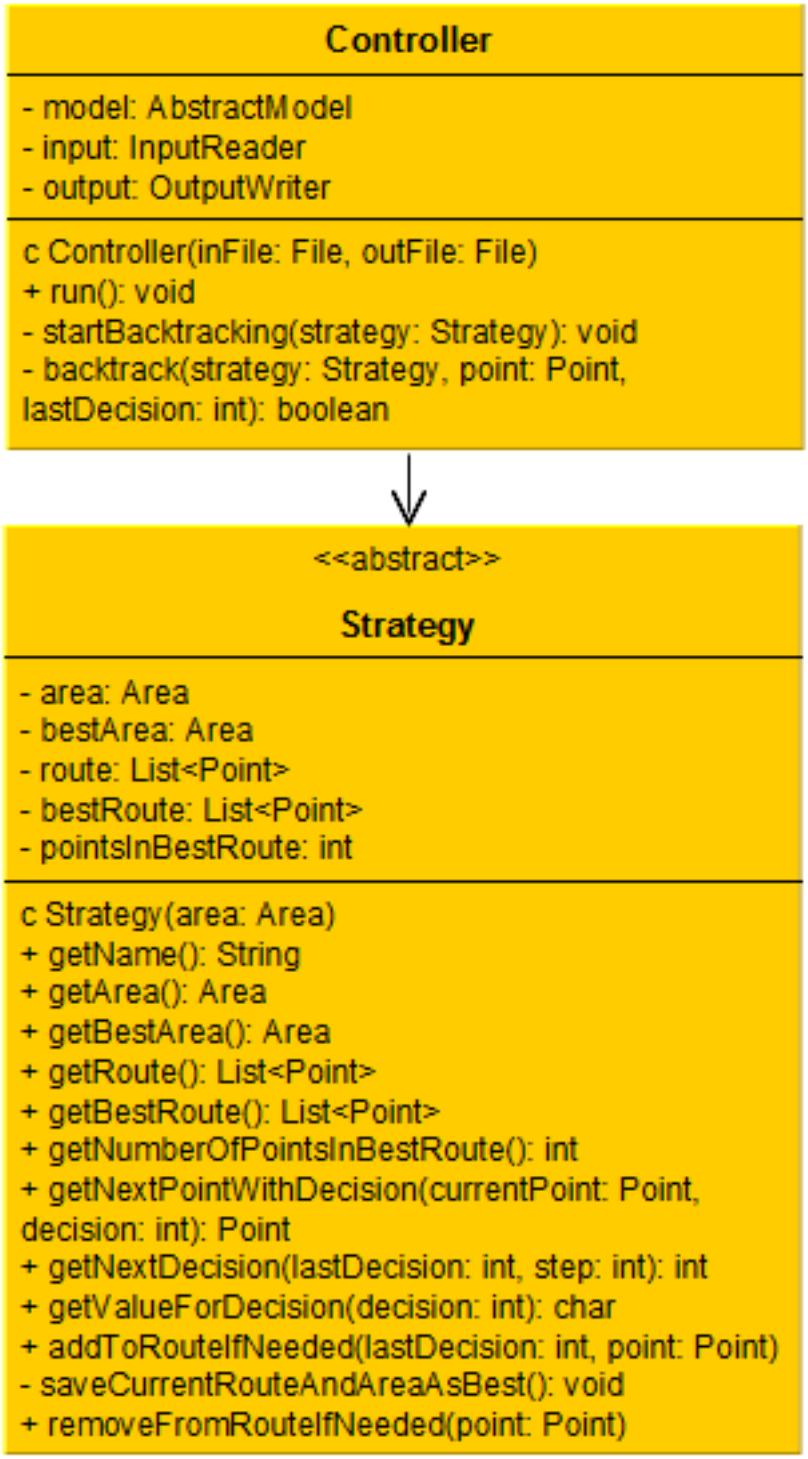
InputReader
Schnittstelle zum Einlesen von Eingabedaten.

InputFileReader
Konkrete Implementierung zum Einlesen über eine Datei.

OutputWriter
Schnittstelle zum Ausgaben von Daten.

OutputFileWriter
Konkrete Implementierung zur Ausgabe in eine Datei.

3.1.3 Controller-Klassen



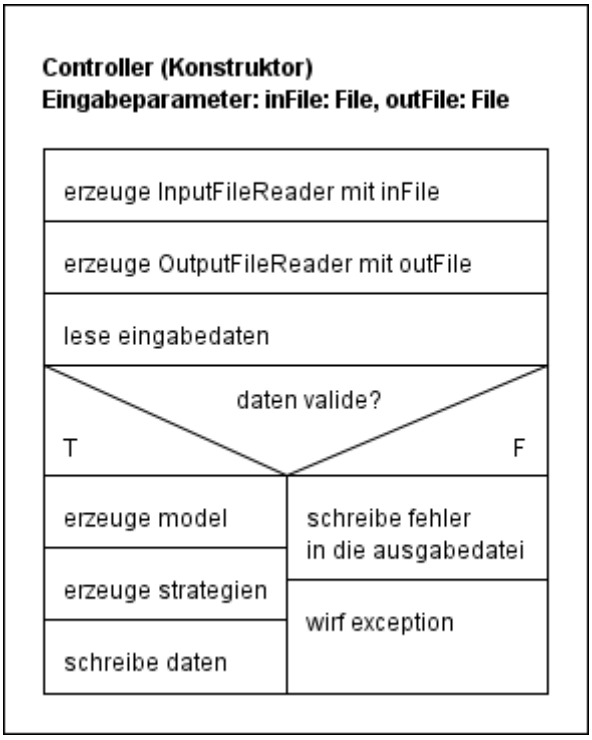
Controller
Startet das Programm und beinhaltet die Logik des Lösungsalgorithmus, in diesem Fall das Backtracking.

Strategy
Stellt eine Schnittstelle zum Ausführen einer möglichen Strategie bereit.

Präzisierung

Die wichtigsten Präzisierungen in Form von Nassi-Shneidermann-Diagrammen sind hier augelistet. Weitere Diagramme zum Einlesen von Daten, Nachhalten der Route usw. sind unter folgendem Pfad als Originaldatei und Bild abgelegt: *diagrams/structograms*.

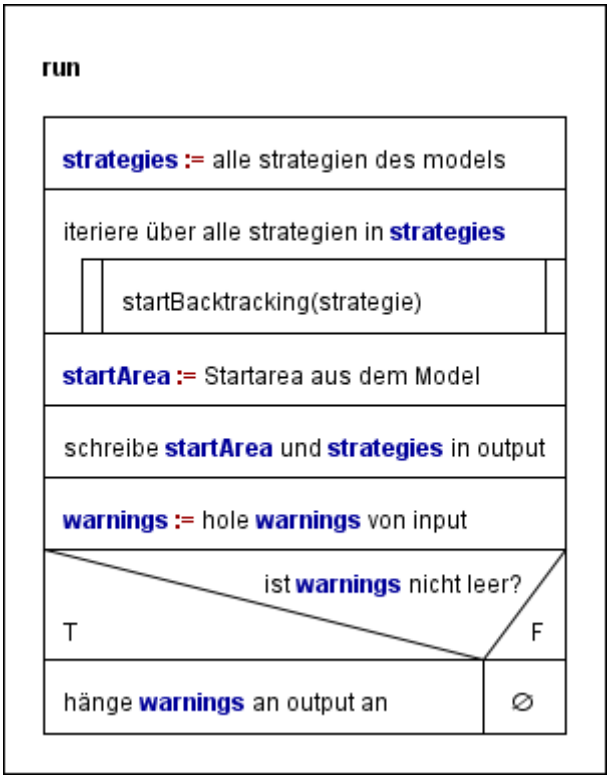
3.1.4 Präzisierung – Controller-Konstruktor



In dem Konstruktor des Controllers werden die grundlegenden Objekte erzeugt und initialisiert. Zuerst werden die Views erzeugt, also Ein- und Ausgabe. Die Ausgabe muss direkt am Anfang erzeugt werden, um im Fehlerfall diesen schreiben zu können.

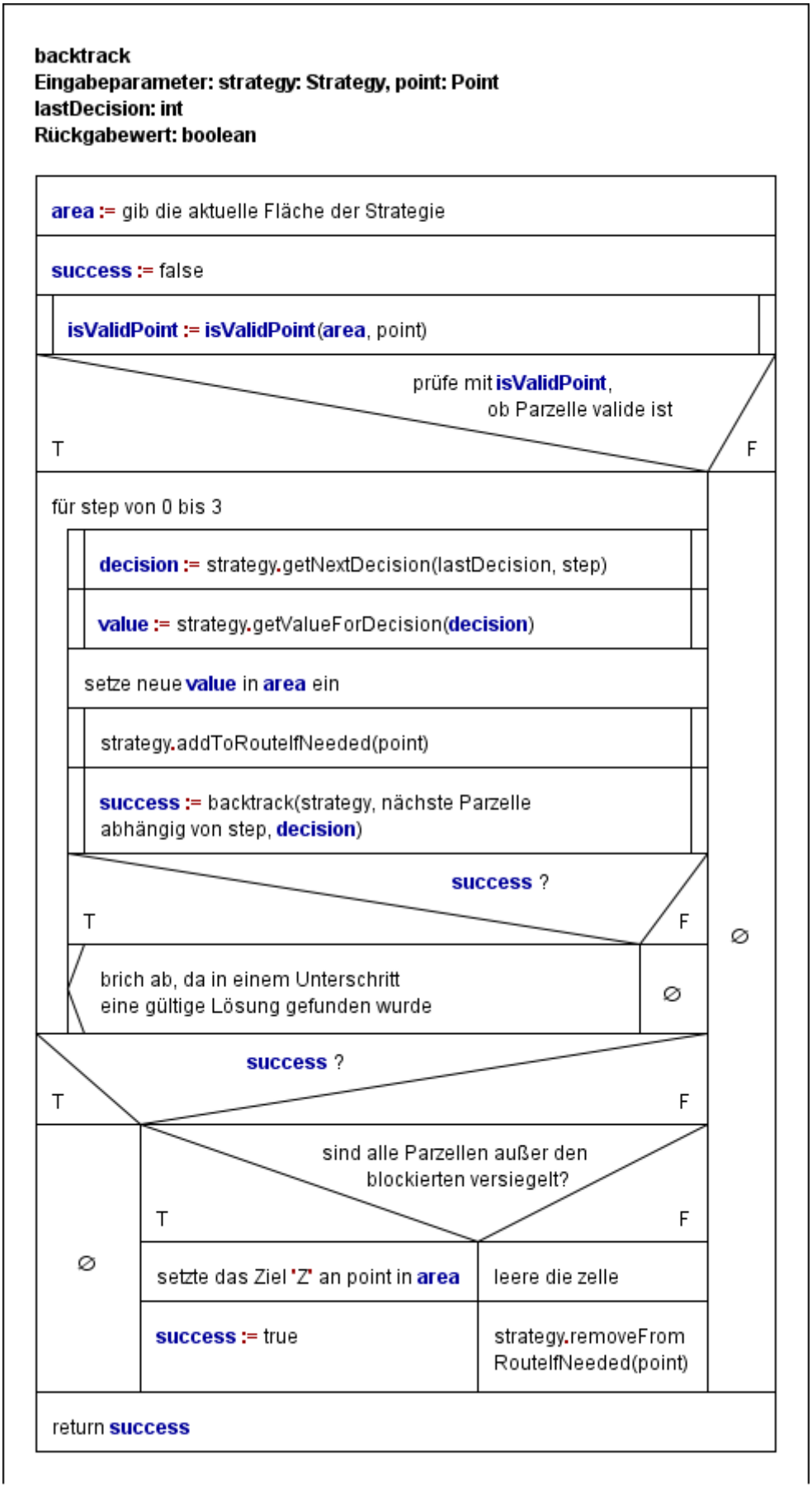
Dann werden die Daten eingelesen und mit den daraus erzeugten Strategien in das Model geschrieben. Falls Exceptions auftreten, wird die Nachricht in die Ausgabedatei geschrieben und die Exception wird weitergeworfen, um zu signalisieren, dass kein gültiges Objekt erzeugt werden kann.

3.1.5 Präzisierung – run()



Startet das eigentliche Programm. Dazu werden alle Strategien abgearbeitet, indem startBacktracking(strategie) für jede Strategie einmal ausgeführt wird. Anschließend werden die Ergebnisse in die Ausgabe geschrieben und falls Warnungen existieren, werden diese an das Ergebnis anhängt.

3.1.6 Präzisierung – backtrack()



Dies ist der eigentliche Lösungsalgorithmus: das Backtracking. Hier wird eine Strategie komplett ausgeführt, sodass ein Lösungsweg gefunden wird. Dieser ist nach vollständig, sofern dies möglich ist, oder versiegelt so viele Parzellen wie möglich.

Zuerst werden einige Prüfungen (*isValidPoint*) durchgeführt, um zu gucken ob von der Parzelle der Lösungsweg weitergesucht werden kann. Wenn nicht wird zum Ende gesprungen. Falls ja werden alle 4 weiteren Schritte gemacht. Welcher Schritt in welche Richtung geht muss die konkrete Strategie entscheiden (*getNextDecision(lastDecision, step)*), für das Backtracking an sich ist es erst mal unwichtig. Wenn in einem Unterschritt, sprich in einer gegebenen Richtung, eine vollständige Lösung gefunden wurde, wird die Schleife abgebrochen.

Wenn keine vollständige Lösung gefunden wurde, werden alle weiteren Schritte (für die weiteren Richtungen) genauso durchgeführt. Falls danach immer noch keine vollständige Lösung gefunden worden ist, muss geprüft werden ob die aktuelle Lösung bereits schon die vollständige ist. Wenn ja wird die aktuelle Position als Ziel markiert und *success* wird auf *true* gesetzt. Dadurch bricht in allen unterliegenden Rekursionsschritten auf dem Stack die Schleife ab und der Algorithmus terminiert. Wenn noch nicht alle Parzellen versiegelt sind, wird der letzte Schritt rückgängig gemacht, sodass ein anderer Weg die Parzelle versiegeln kann.

Während der Schleife werden die versiegelten Parzellen zum Routenplan hinzugefügt und wenn ein Schritt rückgängig gemacht wird auch wieder entfernt. Dies stellt sicher, dass bei einer Lösung immer ein Routenplan existiert. Ist die Lösung unvollständig werden natürlich auch alle Parzellen wieder entfernt. Da aber die Methode *addToRouteIfNeeded(point)* entscheidet, wann eine neue beste Route erzeugt wurde und diese dann Zwischenspeichert, kann später im Programmverlauf die beste Route einfach ausgelesen werden.

3.1.7 Präzisierung –getNextDecision (MyStrategy)

getNextDecision (MyStrategy)
Eingabeparameter: lastDirection: int, step: int
Rückgabewert: int

```
return (lastDecision + step) % 4
```

Diese Methode sorgt dafür, dass im ersten Schritt im die letzte Richtung beibehalten wird. In weiteren Schritten werden dann die anderen Richtungen gewählt, wobei zu berücksichtigen ist, dass nur die ganzzahligen Werte 0-3 gültig sind.

3.1.8 Präzisierung –getValueForDecision (MyStrategy)

getValueForDecision (MyStrategy)
Eingabeparameter: decision : int
Rückgabeparameter: char

result := leerer wert

decision				
0	1	2	3	default
result := '^'	**result** := '<'	**result** := 'v'	**result** := '>'	wirf Exception

return **result**

Die Richtungen werden gegen den Uhrzeigersinn ausgewertet. Wenn eine Richtung übergeben wird, die nicht zwischen 0-3 liegt, wirf eine Exception geworfen.

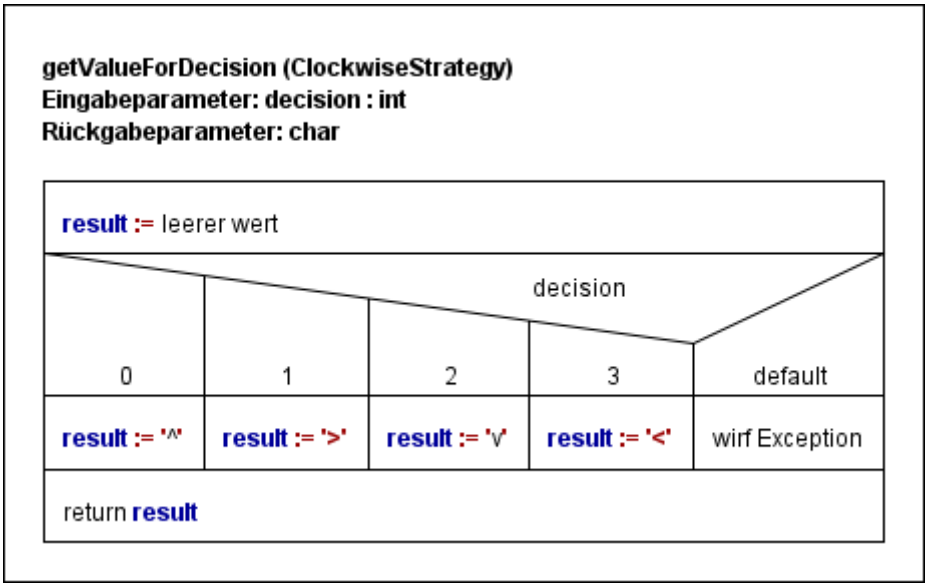
3.1.9 Präzisierung –getNextDecision (ClockwiseStrategy)

getNextDecision (ClockwiseStrategy)
Eingabeparameter: lastDirection: int, step: int
Rückgabewert: int

```
return step
```

Die Uhrzeigerstrategie probiert an jeder Parzelle immer neu die Richtungen von 0-3 aus. Daher muss nur der entsprechende Schritt als nächste Richtung zurückgegeben werden.

3.1.10 Präzisierung –getValueForDecision (ClockwiseStrategy)



Die Richtungen werden im Uhrzeigersinn ausgewertet. Wenn eine Richtung übergeben wird, die nicht zwischen 0-3 liegt, wirf eine Exception geworfen.

4 Testdokumentation

Parallel zu der Entwicklung des Softwaresystems wurden Test geschrieben, die die Funktionalität überprüfen. Bei jeder Änderung wurden die Tests durchgeführt, um mögliche Fehler frühzeitig zu aufzudecken. Die Tests sind in Form von Eingabedateien mit der Endung .in im Ordner „tests“ abgelegt.

4.1 Normalfall

Ein Normalfall liegt vor, wenn keine

Grenzfälle oder Fehlerfälle auftreten. Die in der Aufgabenstellung abgedruckten Beispiele wurden alle erfolgreich durchgeführt.

4.2 Grenzfall

Die festgelegten Grenzfälle einer minimalen und maximalen Eingabedatei sind ebenfalls in den vordefinierten Tests enthalten.

Minimalfall

Eingabe (*minimal.in)*:

```
; Abmessungen der Flaeche
1 1
; Startparzelle
1 1
; Eckepunkte
```

Ausgabe:

```
Startpunkt (1,1)
  1
1 S

Uhrzeiger-Strategie
  1
1 Z

Routenplan:
1,1

Meine Strategie
  1
1 Z

Routenplan:
1,1

zu versiegelnde Parzellen: 1
Hindernisparzellen: 0
versiegelte Parzellen: 1
nicht versiegelte Parzellen: 0
```

Maximalbeispiel

Eingabe (*maximal.in*):

```
; Abmessungen der Flaeche
10 10
; Startparzelle
1 1
; Eckepunkte
```

Ausgabe:

```
Startpunkt (1,1)
  1 2 3 4 5 6 7 8 9 10
1 S
2
3
4
5
6
7
8
9
10

Uhrzeiger-Strategie
  1 2 3 4 5 6 7 8 9 10
1 > > > > > > > > > v
2 Z v < v < v < v < v
3 ^ v ^ v ^ v ^ v ^ v
4 ^ v ^ v ^ v ^ v ^ v
5 ^ v ^ v ^ v ^ v ^ v
6 ^ v ^ v ^ v ^ v ^ v
7 ^ v ^ v ^ v ^ v ^ v
8 ^ v ^ v ^ v ^ v ^ v
9 ^ v ^ v ^ v ^ v ^ v
10 ^ < ^ < ^ < ^ < ^ <

Routenplan:
1,1 / 1,10 / 10,10 / 10,9 / 2,9 / 2,8 / 10,8 / 10,7 / 2,7 / 2,6 / 10,6 /
10,5 / 2,5 / 2,4 / 10,4 / 10,3 / 2,3 / 2,2 / 10,2 / 10,1 / 2,1

Meine Strategie
  1 2 3 4 5 6 7 8 9 10
1 v v < < < < < < < <
2 v v v < < < < < < ^
3 v v v v < < < < ^ ^
4 v v v v v < < ^ ^ ^
5 v v v v v Z ^ ^ ^ ^
6 v v v v > ^ ^ ^ ^ ^
7 v v v > > > ^ ^ ^ ^
8 v v > > > > > ^ ^ ^
9 v > > > > > > > ^ ^
10 > > > > > > > > > ^
```

```
Routenplan:
1,1 / 10,1 / 10,10 / 1,10 / 1,2 / 9,2 / 9,9 / 2,9 / 2,3 / 8,3 / 8,8 / 3,8 /
3,4 / 7,4 / 7,7 / 4,7 / 4,5 / 6,5 / 6,6 / 5,6

zu versiegelnde Parzellen: 100
Hindernisparzellen: 0
versiegelte Parzellen: 100
nicht versiegelte Parzellen: 0
```

Worst-Case-Szenario 6x6

Das Worst-Case-Szenario mit einer 10x10-Fläche würde nicht in absehbarer Zeit fertig werden, daher existiert dort ein Worst-Case-Szenario einer 6x6-Fläche. Dies ist vom Prinzip genauso aufgebaut, wird aber (auf meinem Rechner) innerhalb einiger Sekunden durchgeführt.

Eingabe (*worstcase6x6.in*):

```
; Abmessungen der Flaeche
6 6
; Startparzelle
1 1
; Eckepunkte
6 5 6 5
5 6 5 6
```

Ausgabe:

```
Startpunkt (1,1)
  1 2 3 4 5 6
1 S
2
3
4
5           H
6         H

Uhrzeiger-Strategie
  1 2 3 4 5 6
1 > > > > > v
2 v < < v < v
3 > v ^ v ^ v
4 v < ^ < ^ <
5 v > v > Z H
6 > ^ > ^ H

Routenplan:
1,1 / 1,6 / 4,6 / 4,5 / 2,5 / 2,4 / 4,4 / 4,3 / 2,3 / 2,1 / 3,1 / 3,2 / 4,2
/ 4,1 / 6,1 / 6,2 / 5,2 / 5,3 / 6,3 / 6,4 / 5,4 / 5,5

Meine Strategie
  1 2 3 4 5 6
1 v > > > > v
2 v ^ v < v <
3 v ^ v ^ > v
```

```
4 v ^ v ^ v <
5 v ^ < ^ Z H
6 > > > ^ H

Routenplan:
1,1 / 6,1 / 6,4 / 2,4 / 2,3 / 5,3 / 5,2 / 1,2 / 1,6 / 2,6 / 2,5 / 3,5 / 3,6
/ 4,6 / 4,5 / 5,5

zu versiegelnde Parzellen: 34
Hindernisparzellen: 2
versiegelte Parzellen: 33
nicht versiegelte Parzellen: 1
```

4.3 Fehlerfall

Es sind neun Tests für Fehlerfälle enthalten, weil in der Beschreibung mehrere Fälle zusammengefasst worden sind. In allen Fällen wird falls nötig eine Fehleranalyse oder eine Warnung an die Ausgabe angehängt. Danach wird das Programm korrekt beendet.

- Es fehlt die Startzeile:
  missing_start.in

- Es sind weniger als 1 oder mehr als 10 Parzellen in jede Richtung gegeben:
  zero_dimensions.in
  large_dimensions.in

- Es werden Whitespaces als Trennzeichen benutzt:
  whitespaces_divider.in

- Es werden ungültige Trennzeichen benutzt:
  invalid_divider.in

- Es werden Fließkommazahlen eingegeben:
  float_values.in

- Ein Hindernis wird nicht ausreichend definiert:
  undefined_block.in

- Es sind mehrere Abmessungen in einer Zeile gegeben:
  multiple_dimensions.in

- Der Startpunkt liegt außerhalb des Feldes:
  invalid_start.in

- Der Startpunkt ist auch ein Hindernis:
  start_is_blocked.in

- Zwei oder mehr Hindernisse überlagern sich:
  blocks_overlaying.in

4.4 Ausführliches Beispiel

4.5 Übersicht aller Tests mit erwarteter Ausgabe

5 Zusammenfassung und Ausblick

5.1 Zusammenfassung

// TODO viel länger, halbe bis eine seite

5.2 Ausblick

- Feld vorher so prüfen, dass man weiß wie viele Felder nicht belegt werden können. Ist die maximale Anzahl – nicht belegbare Felder erreicht kann der Algorithmus abbrechen, weil er ja eh nichts besseres findet:
  => vorgezogene analyse

6 Änderungen

6.1 Abweichungen

- Die Sichtbarkeit der Liste „warnings“ der Klasse InputReader von private auf protected gesetzt, damit Unterklassen die Liste bearbeiten können.

- In der Klausur am Montag habe ich für eine Richtungsentscheidung manchmal den Begriff „direction“ und manchmal „decision“ verwendet. Ich habe dies zu „decision“ vereinheitlicht.

- Dem Konstruktor der Klasse *Area* habe ich im UML-Diagramm UML den Parameter „dimensions“ der Typ int[][] übergeben. Dies war ein Schreibfehler und wurde zu int[] geändert.

- Die Methode getWarnings(): List<String> in der Klasse *InputReader* wurde in getWarning(): String geändert, die nur noch eine zusammengefasste Warnung ausgibt.

- Die Logik zur Erstellung der Routenliste wurde geändert. In der Klausur ging ich davon aus, dass ich eine Parzelle nur einfüge, wenn sich die Richtung geändert hat. Da es aber bei Rückschritten des Algorithmus schwierig ist die Liste konsistent zu halten, habe ich mich entschlossen jede abgelaufene Parzelle zu speichern bzw. immer die letzte dann zu entfernen. Für die Ausgabe werden aus der Liste nicht benötigte Parzellen entfernt.
  Daher benötigt die Methode addPointToRoute in der Klasse *Strategy* nicht mehr den Parameter lastDecision: int.

- Meine Strategie (Klasse *MyStrategy*) implementiert die Methode getNextDecision(lastDecision: int, step: int) nun so, wie ich es in der Aufgabenanalyse beschrieben habe. Die angegebene Umsetzung im Nassi-Shneidermann-Diagramm war nicht korrekt.

- In der Methode backtrack(strategy: Strategy, point: Point, lastDecision: int) muss die Boolean-Variable „success“ direkt am Anfang initialisiert werden, statt in der for-Schleife wie es im Nassi-Shneidermann-Diagramm dargestellt wurde.

- In der Methode backtrack(strategy: Strategy, point: Point, lastDecision: int) muss die Boolean-Variable „success“ auf true gesetzt werden, wenn sie es vorher nicht war und nun alle Parzellen versiegelt sind. Dies wurde im Nassi-Shneidermann-Diagramm am Ende ebenfalls falsch dargestellt.

6.2 Ergänzungen

- Ich habe einen weiteren Grenzfall als Worst-Case-Szenario definiert, der keine vollständige Lösung besitzt. Dadurch muss die maximale Anzahl an Schritten durchgeführt werden, was aber sehr lange dauert.
- Methoden zu der Klasse *Area* hinzugefügt
  - + existsCell(p: Point): boolean
  - + containsEndPoint(): boolean
  - + numberOfCells(): int

    - – numberOfCellsWithValues(possibleValues: char[]): int

- Methode zu der Klasse *Strategy* hinzugefügt
    - + getNextPointWithDecision(currentPoint: Point, decision: int): Point

- Methode zu der Klasse OutputWriter hinzugefügt
    - # formatRoute(route: List<Point>): String

- Methode zu der Klasse OutputFileWriter hinzugefügt
    - – setFile(file: File): void

- Methode zu der Klasse Controller hinzugefügt
    - – isValidPoint(point: Point, area Area): boolean

7 Benutzeranleitung

7.1 Verzeichnisstruktur der CD

Auf der CD sind folgende Verzeichnisse angelegt:

Wurzelverzeichnis

- Ausführbares Programm.
- Die Beschreibung als Master-Datei im Microsoft Word-Format und als PDF.
- „bin“ um die *.class-Dateien und das Manifest abzulegen.
- „diagrams“ enthält die verschiedenen Diagrammtypen jeweils als Orginaldateien und Bilddateien im PNG-Format.
- „documentation“ enthält die Entwicklerdokumentation.
- „scripts“ enthält Batch-Skripte für das
    - Kompilieren und erzeugen des ausführbaren Programms.
    - Automatische Durchführen aller Tests.
    - Aufräumen bzw. Löschen der erzeugten Dateien.
- „src“ enthält den Quellcode.
- „tests“ enthält die Eingabe- und Ausgabedateien der Tests.

7.2 Systemvoraussetzungen

Um das Programm auszuführen zu können, muss das Java Software-Development-Kit (SDK) in der Version 1.7 oder höher installiert sein. Die aktuelle Java-Version kann unter folgendem Link heruntergeladen werden:

http://java.com/de/download/index.jsp

Von Oracle angegebene Systemvoraussetzungen sind:[2]

- Windows 8 Desktop
- Windows 7
- Windows Vista SP2
- Windows XP SP3 (32-Bit); Windows XP SP2 (64-Bit)
- Windows Server 2008

Ein Pentium 2 266 MHz oder schnellerer Prozessor mit einem physikalischen RAM von mindestens 128 MB wird empfohlen. Außerdem benötigen Sie mindestens 124 MB freien Speicherplatz auf dem Datenträger.

7.3 Installation

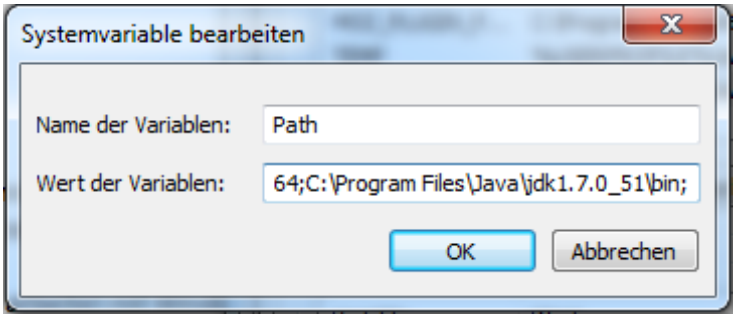
Der gesamte Inhalt der CD kann in ein beliebiges Verzeichnis kopiert werden. Wichtig ist, dass das Verzeichnis beschrieben werden darf (Schreibzugriff erlaubt).

[2] Siehe http://java.com/de/download/win_sysreq-sm.jsp (Stand 16:22 Uhr - 09.04.2014)

Um das Programm unter Windows ausführen zu können muss in der Systemvariable „Path“ der Pfad in das „bin“-Verzeichnis der Java-SDK-Installation gesetzt sein.

Die folgende Beschreibung zum Setzen der Variable arbeitet mit Windows 7 Professional:

1. Öffnen des Startmenüs
2. Rechtsklick auf „Computer“, dann Linksklick auf „Eigenschaften“
3. Über das linke Menü „Erweiterte Systemeinstellungen“ öffnen
4. In dem neuen Fenster auf „Umgebungsvariablen“ klicken
5. Bei „Systemvariablen“ die Variable „Path“ auswählen und auf „Bearbeiten…“ klicken
6. Bei „Wert der Variablen“ den Pfad zur Java-Installation mit dem Unterordner „bin“ angeben.
   In meinem Beispiel wäre das:
   C:\Program Files\Java\jdk1.7.0_51\bin;
   Dabei muss auf die Semikolon vor und nach dem Pfad geachtet werden, da diese als Trennzeichen dienen.
7. Mit Klick auf „OK“ bestätigen und ggf. die Konsole erneut öffnen.



7.4 Ausführen der Skripte

Nachdem die Systemvariable korrekt gesetzt wurde, können die Batch-Skripte aus dem Unterordner „scripts“ direkt ausführt werden. Dies kann entweder über die Konsole geschehen, indem <dateiname>.bat aufgerufen wird oder mit einem Doppelklick im Dateisystem auf die Datei. Es werden keine Übergabeparameter erwartet oder ausgewertet.

7.4.1 clear.bat

Das Script „clear.bat“ iteriert über alle erzeugten Ausgabedateien und löscht diese.

7.4.2 compile.bat

Kompiliert das Java-Programm über den Konsolenbefehl „javac“ und erzeugt daraus die ausführbare Datei NimGame.jar. Dies ist bei Änderungen am Quellcode erforderlich, um anschließend mit dem Script „run.bat“ automatisiert alle Tests auszuführen.

7.4.3 run.bat

Führt das Programm mit allen Eingabedateien aus, welches dann die entsprechenden Outputdateien erzeugt.

8 Entwicklerdokumentation

Die Entwicklerdokumentation befindet sich in Form einer vollständigen JavaDoc in dem Unterordner „documentation“. Es sind Methoden und Attribute aller Sichtbarkeitsstufen (public, protected, package, private) inkludiert. Das Programm JavaDoc erzeugt aus den entsprechenden Quellcode-Kommentaren eine vollständige Entwicklerübersicht als HTML-Dateien. Diese ist wie eine Webseite benutzbar. Durch das Öffnen der Datei „index.html“ wird die Übersicht geladen, von der man zu allen weiteren Paketen und Klassen gelangt.

JavaDoc ist ein standardisiertes Verfahren zur Dokumentation von Java-Quelltext. Vor dem zu dokumentierendem Code wird mit den Zeichen /** der Begin eines JavaDoc-Kommentars eingeleitet und mit den Zeichen */ beendet. Dies lehnt sich an mehrzeilige Kommentare an, die allerdings nicht von JavaDoc interpretiert werden.

Es wird für jede Klasse eine Beschreibung angegeben, die den (eindeutigen) Zweck der Klasse erklärt. Auch kann der Autor angegeben werden und bei Bedarf die Version des Programms.

Methoden werden ebenfalls mit einer Beschreibung des Zwecks versehen, doch gibt es noch weitere Schlüsselwörter: mit @param werden Übergabeparameter näher erklärt und mit @throws sollte der Entwickler checked und unchecked exceptions dokumentieren. Ein Beispiel dafür ist das Parsen eines Integers mit der statischen Methode *Interger.parseInt(String s)*. Falls aus dem übergebenen String kein Integer geparsed werden kann, wird in dem Fall eine FormatException geworfen, die gefangen werden kann.

Es gibt noch einige weitere Schlüsselwörter, die aber nicht so häufig verwendet und deshalb hier nicht erklärt werden. Weitere Informationen sind bei Wikipedia unter folgendem Link zu finden: https://de.wikipedia.org/wiki/Javadoc#.C3.9Cbersicht_der_Javadoc-Tags

9 Entwicklungsumgebung

Das gesamte Softwaresystem wurde unter folgendem Computer-System entwickelt und getestet:

Programmiersprache	Java 1.7 Update 51
Rechner	Intel® Pentium® CPU P6200 2.13 GHz 2.13 GHz 4 GB Arbeitsspeicher
Betriebssystem	Windows 7 Professional 64 Bit-Betriebssystem

10 Eigenhändigkeitserklärung

Ich erkläre verbindlich, dass das vorliegende Prüfprodukt von mir selbstständig erstellt wurde. Die als Arbeitshilfe genutzten Unterlagen sind in der Arbeit vollständig aufgeführt. Ich versichere, dass der vorgelegte Ausdruck mit dem Inhalt des von mir erstellten Datenträgers identisch ist. Weder ganz noch in Teilen wurde die Arbeit bereits als Prüfungsleistung vorgelegt. Mir ist bewusst, dass jedes Zuwiderhandeln als Täuschungsversuch zu gelten hat, der die Anerkennung des Prüfprodukts als Prüfungsleistung ausschließt.

Im Rahmen des auftragsbezogenen Fachgesprächs sind die Aufgabenanalyse und der Lösungsentwurf zu begründen und das Prüfungsprodukt zu erläutern.

Ort und Datum	Unterschrift des Prüfungsteilnehmers

11 Verwendete Hilfsmittel

- Eclipse Standard/SDK (32 Bit)
  Version: Kepler Service Release 1
  Entwicklungsumgebung für Java und andere Programmiersprachen.
  http://eclipse.org/
- Notepad++
  Open-Source-Texteditor für Windows
  http://notepad-plus-plus.org
- yEd
  Plattform unabhängiger Graph-Editor. Unter anderem Fähig UML-Diagramme zu erstellen.
  http://www.yworks.com/de/products_yed_about.html
- Structorizer
  Plattform unabhängiges Programm zur Erzeugung von Nassi-Shneidermann-Diagrammen.
  http://structorizer.fisch.lu/
- Quick Sequenz Diagram Editor
  Plattform unabhängiges Programm zur Erzeugung von Sequenzdiagrammen.
  http://sdedit.sourceforge.net/index.html
- GIMP
  Plattform unabhängiges Programm zur Grafikbearbeitung.
  http://www.gimp.org/