Korrekturrand

Model

- ~~startStrategy~~
- startArea: Area
- strategies: List<Strategy>

+ getStartArea(): Area
+ setStartArea(area: Area): void
+ addStrategy(strategy: Strategy): void
+ getStrategies(): List<Strategy>

Controller

- input: InputReader
- output: OutputWriter
- model: AbstractModel

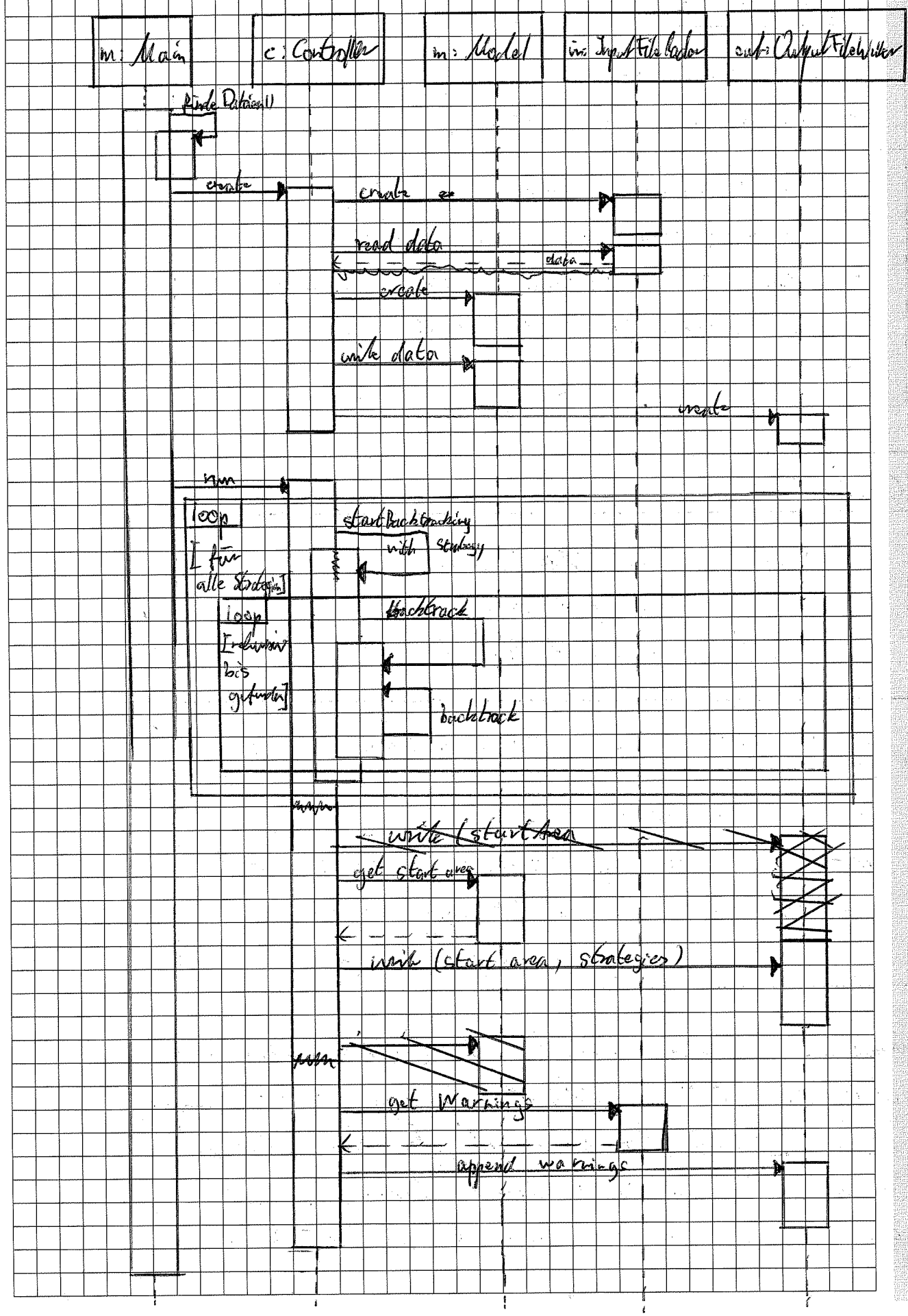
© Controller(inFile: File, outFile: File)

+ run(): void
- startBacktracking(strategy: Strategy): void
- backtrack(strategy: Strategy, point: Point, lastDecision: int): boolean

Hinweis: Ich habe Input und Output extra als abstrakte Klasse deklariert, da dies meiner Meinung nach korrekter ist als ein Interface. Falls es Probleme mit mehrfach-Vererbung geben sollte, kann man ja einfach ein Objekt anlegen :)

Korrekturrand

Sequenzdiagramm – Gesamtablauf



Korrekturrand

Nassi-Shneidermann-Diagramme

Controller (Konstruktor)
Eingabeparameter: inFile : File , outFile : File

erzeuge Input File Loader mit ~~Text~~ inFile

lese Eingabedaten

Daten valide ?

ja	nein
~~erzeuge~~ erzeuge Model schreibe Daten erzeuge Ausgabe ∅	schreibe Fehler in Ausgabe Datei wirf Exception

run

iteriere über alle Strategien des Models

startArea := Startarea des Models

startPoint := startArea.startPoint

this.startBacktracking(Strategie)

startArea := Startarea des Models

schreibe startArea und Strategien in output

warnings := hole Warnings von input

hänge Warnings an output an

Korrekturrand

startBacktracking

Eingabeparameter: strategy: Strategy

startArea := Startarea des Modells

startPoint := startArea.getStartPoint()

backtrack (strategy, startPoint, strategy.getFirstDecision())

backtracking

Eingabeparameter: strategy: Strategy, point: Point
lastDecision: int

Rückgabewert: boolean

ist Punkt innerhalb der Grenzen?
- Nein
- Ja: ist point beim Hindernis?
  - Ja
  - Nein: ist point noch frei?
    - Nein
    - Ja:

~~char value~~
decision := ~~strategy.~~getNextDecision

für step := von 0 bis 3

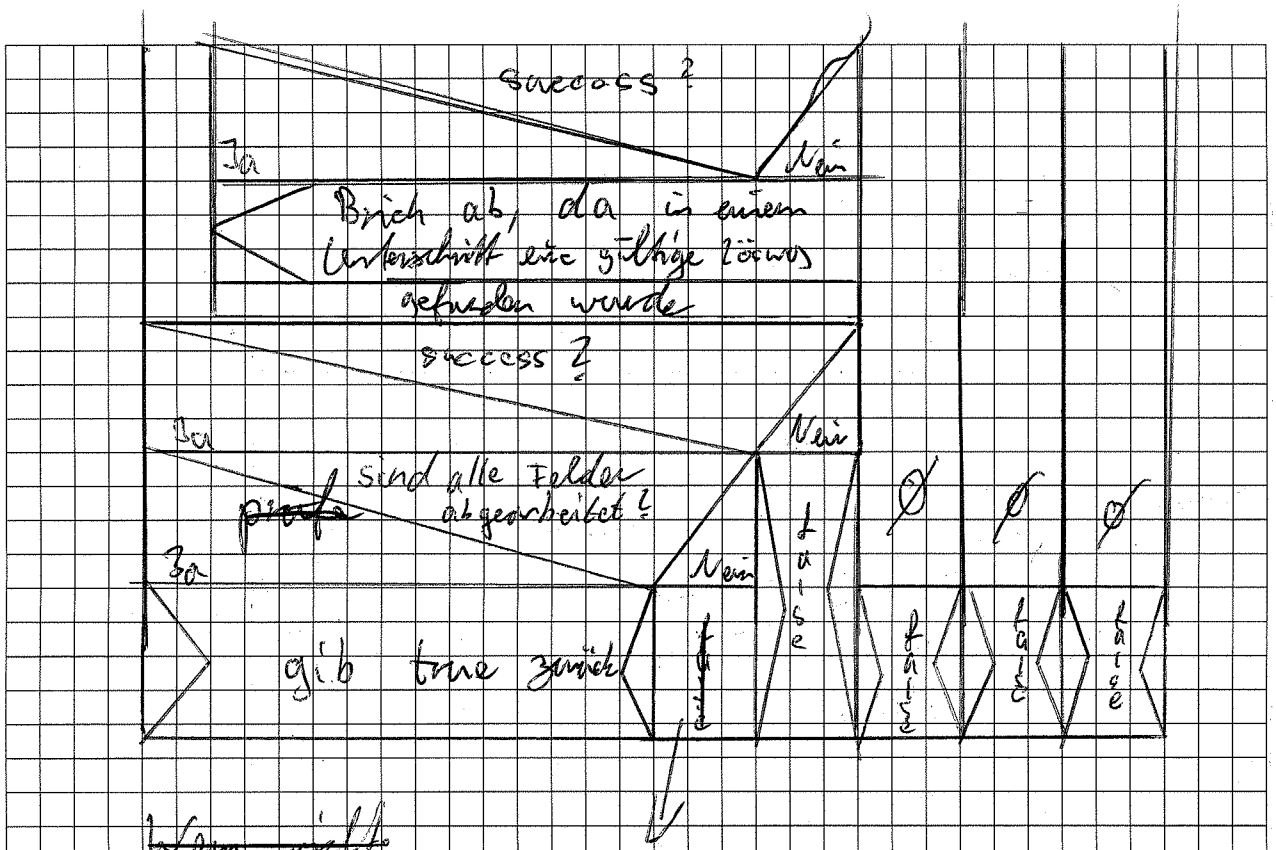
decision := strategy.getNextDecision (lastDecision, step)

value := strategy.getValueForDecision (decision)

~~strategy.area.setValue~~ Füge zur Route hinzu falls benötigt

Setze neue Value ~~ein~~ in Area ein

boolean success :=
backtrack (strategy, nächster Punkt abhängig von step, decision



Wenn nicht

* addToRoute (lastDecided (lastDecision, point))

leere die Zelle
removeFromRoute (lastDecided (point))
return false

Get Next Decision (My Strategy)

Eingabeparameter: lastDirection: int, step: int

Ausgabe: int

Ja

gib lastDecision zurück

result := leerer int

step = 0

step

0	1	2	3	sonst
result := 3	result := 0	result := 1	result := 2	nicht Exception

gib result zurück

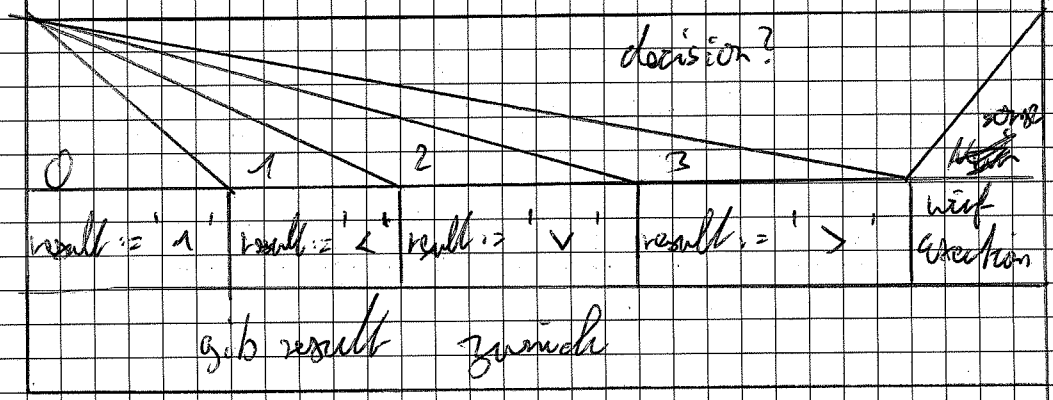
Korrekturrand

getValueForDecision (My Strategy)

Eingabe: decision b: int

Rückgabe: char

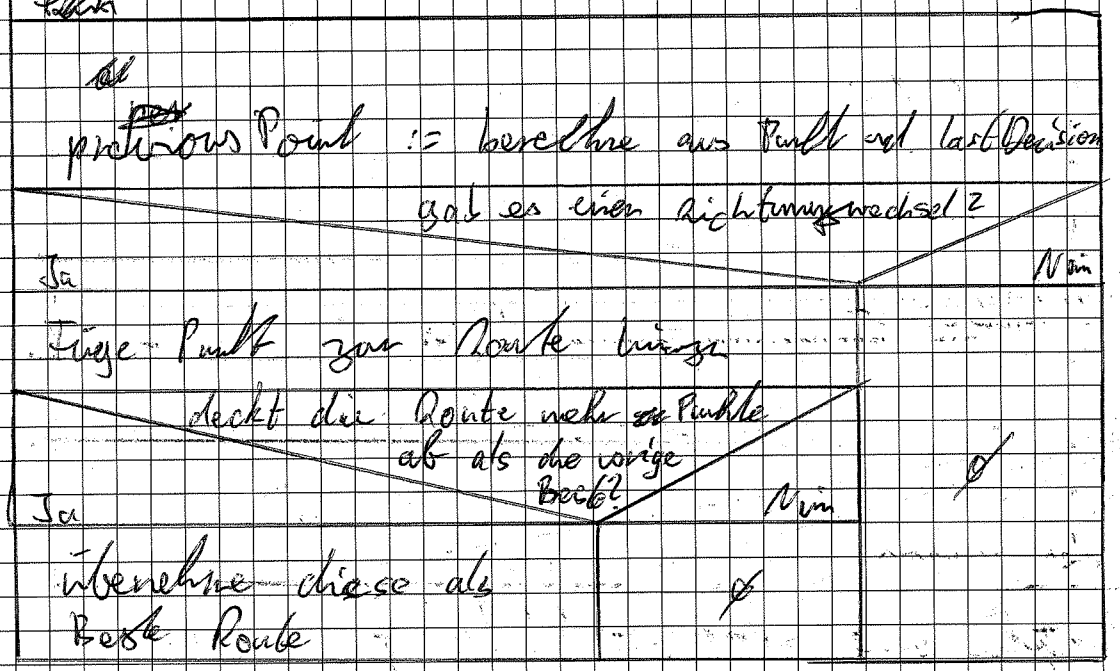
result := leerer char



gib result zurück

addToRoute (if Needed (Strategy)

Eingabe: lastDecision: int, point: Point



removeFromRouteIfNeeded
Eingabe: point: Point, decision: int

Falls der letzte Punkt in der Routenliste dieser ist

Ja | Nein

entferne einträge aus Route

gebe einen Schritt gegen die Direktion und füge den Punkt hinzu

Output

wurde

Eingabe: startArea: Area, strategies: List<Strategie>

gib startArea.getStartPoint() aus

gib startArea aus

iteriere über strategien

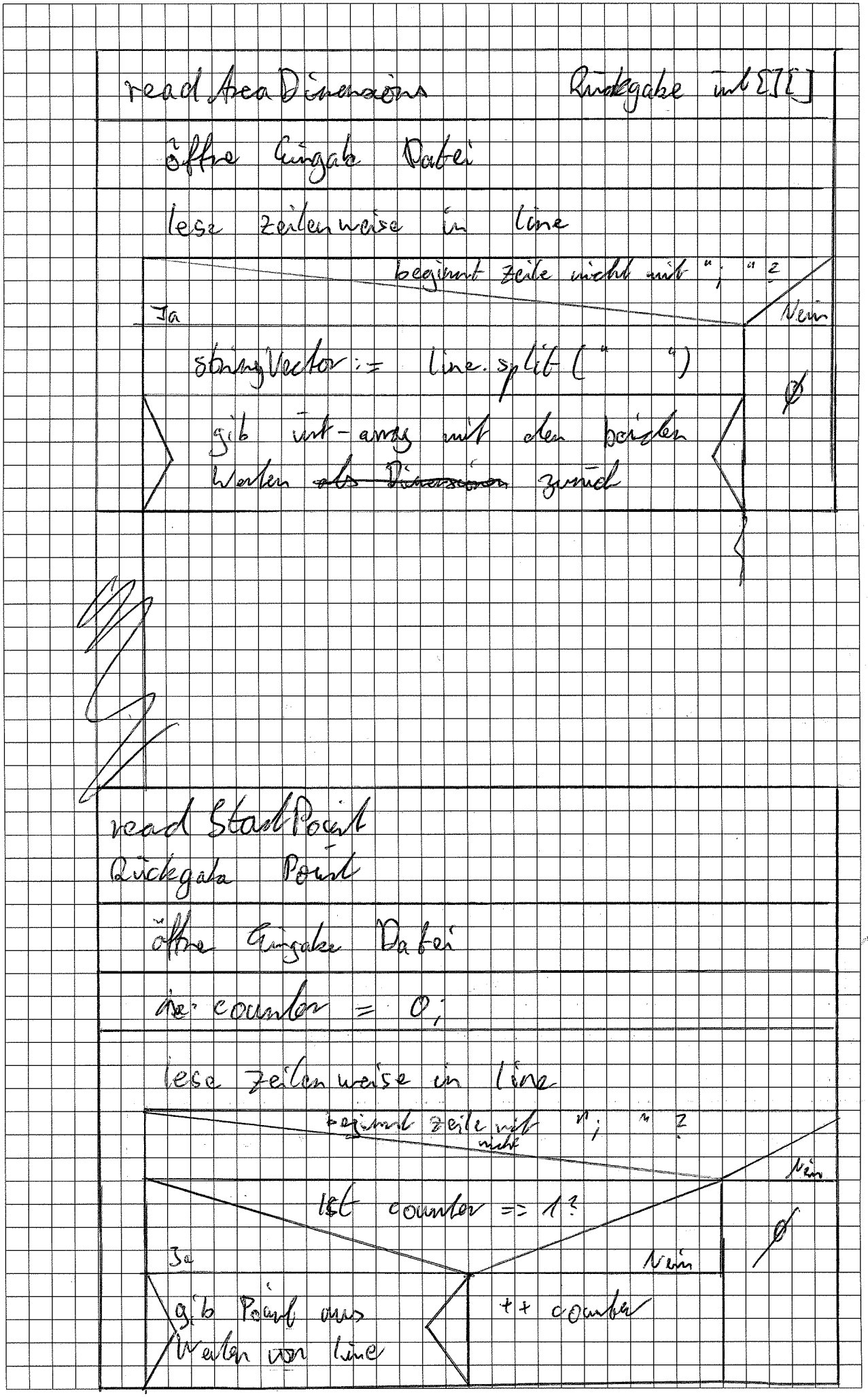
gib strategie.getName() aus:

gib strategy.getArea() aus

nimm beispielhaft die erste Strategie des Vektors

erstelle aus strategie Statistik und gib aus

öffne Datei


readAreaDimensions
Rückgabe int[2][]
öffne Eingabe Datei
lese zeilenweise in line
beginnt Zeile nicht mit "; " ?
Ja
Nein
stringVector := line.split(" ")
gib int-array mit den beiden Werten ~~als Dimensionen~~ zurück
readStartPoint
Rückgabe Point
öffne Eingabe Datei
nr. counter = 0;
lese zeilenweise in line
beginnt Zeile nicht mit "; " ?
Nein
ist counter == 1?
Ja
Nein
gib Point aus Werten von line
++ counter

Bei readBlocks ist es genauso, nur dass man noch ganz unten laufen muss und dann jede Zeile in einer Block umwandelt, der in eine Liste kommt.

(keine Zeit mehr)

Korrekturrand

Korrekturrand

Nachtrag zur Analyse

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

- Interaktion mit Dateien, um Eingabe und Ausgabe zu benutzen.
~~Dies ist durch~~
- Ein Datenmodell, dass die Daten verhält und Zugriffe regelt.
Dies sind die Strategien mit ihren jeweiligen Areas.
- Die Logik des Backtrackings. Diese wird vom Controller ausgeführt und benutzt View und Model als Kommunikations-partner.

Deshalb lässt sich mein Program grob in diese Untermodule einteilen.

Eine komplette Durchführung besteht aus:

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

Korrekturrand

Nachbarn mein Algorithmus

Über das Nachbarn-Verfahren findet man in jedem Fall ~~einen Algorithmus~~ 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.
- Überschreitet die ~~er~~ Anzahl der durch die Routenliste abgedeckten Punkte die bisher als beste Routenliste gespeicherte. So wird die neue Liste als beste Liste gespeichert.
- Falls von einem Punkt ein Rückschritt erfolgen muss, muss dieser aus der aktuellen Routenliste entfernt werden.
- Wenn der Algorithmus terminiert und keine vollständige Lösung existiert, so wird einfach mit der gespeicherten besten Routenliste die bestmögliche Abdeckung erreicht.

~~Das hatte ich beim ersten Algorithmus~~