Dokumentation der Praktischen Arbeit zur Prüfung zum Mathematisch-technischen Softwareentwickler

1. April 2020

**Bjarne-Dag Herrmann**

Prüfungsnummer: 187

Programmiersprache: Java

Inhalt

[1. Aufgabenanalyse 4](#_Toc36714049)

[1.1 Analyse 4](#_Toc36714050)

[1.2 Eingabe 4](#_Toc36714051)

[1.3 Ausgabeformat 4](#_Toc36714052)

[1.4 Programmanforderung 4](#_Toc36714053)

[1.5 Sonderfälle 5](#_Toc36714054)

[1.6 Fehlerfälle 5](#_Toc36714055)

[2. Verfahrensbeschreibung 6](#_Toc36714056)

[2.1 Vorgehensweise 6](#_Toc36714057)

[2.2 Sonderfälle 6](#_Toc36714058)

[2.3 Fehlerfälle 6](#_Toc36714059)

[3. Systembeschreibung 6](#_Toc36714060)

[3.1 Programm 6](#_Toc36714061)

[3.2 Model-Classes 6](#_Toc36714062)

[3.3 View-Classes 6](#_Toc36714063)

[3.4 Controller-Classes 6](#_Toc36714064)

[4. Datenfluss 7](#_Toc36714065)

[5. Programmbeschreibung 7](#_Toc36714066)

[5.1 Pakete 7](#_Toc36714067)

[5.2 UML Diagramme 7](#_Toc36714068)

[5.3 Schnittstellen 7](#_Toc36714069)

[5.4 Präzisierung 7](#_Toc36714070)

[6. Testdokumentation 7](#_Toc36714071)

[7. Ausführliches Beispiel 8](#_Toc36714072)

[8. Zusammenfassung und Ausblick 8](#_Toc36714073)

[8.1 Zusammenfassung 8](#_Toc36714074)

[8.2 Ausblick 8](#_Toc36714075)

[9. Abweichungen und Ergänzungen zum Vorentwurf 8](#_Toc36714076)

[10. Benutzeranleitung 9](#_Toc36714077)

[10.1 Verzeichnisstruktur 9](#_Toc36714078)

[10.2 Vorbereitung des Systems 9](#_Toc36714079)

[10.3 Installation 9](#_Toc36714080)

[10.4 Kompilieren 9](#_Toc36714081)

[10.5 Programmaufruf 9](#_Toc36714082)

[11. Entwicklungsumgebung 9](#_Toc36714083)

[12. Verwendete Hilfsmittel 9](#_Toc36714084)

[13. Erklärung 9](#_Toc36714085)

[14. Anhang 9](#_Toc36714086)

[14.1 Aufgabenstellung 9](#_Toc36714087)

[14.2 In- und Output der Testdokumentation 9](#_Toc36714088)

# Aufgabenanalyse

## Analyse

Gegeben ist ein Puzzle, welches durch gegebene Felder definiert ist. Felder besitzen, da diese aneinander liegen, Nachbarn. Auf jedem Feld gehört eine dazugehörige Karte, welche dieselbe Anzahl an Kanten beziehungsweise Ecken besitzt. Zur Vereinfachung wird im Programm ein 12-teiliges Puzzle mit 5-eckige Feldern / Karten verwendet. Abbildung 1 zeigt ein mögliches Beispiel eines solchen Feldes:

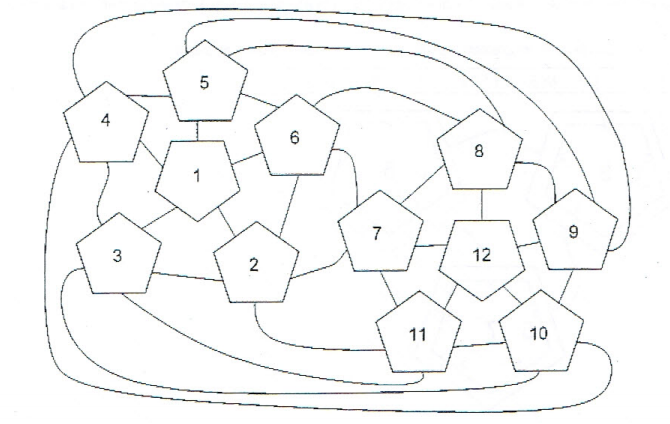


Abbildung 1: Beispiel eines 12-teiligen, 5-eckigen Puzzles

Die Karten besitzen an jeder Kante eine Ziffer – 0, 1 oder 2 – an denen jeweils eine passende Kante einer anderen Karte mit derselben Ziffer angelegt werden muss. Folglich besitzt jede Karte eine 5-stellige Ziffernfolge, die ausgehen von einer sogenannten Grundkante im Uhrzeigersinn die Kantenzugehörigkeit beschreibt. Somit gibt es – bezogen auf dieses Szenario – für eine Karte fünf Möglichkeiten, diese auf einen Platz zu legen.

Die Eingabe erfolgt über eine Datei, in der Zeilenweise die Kanteninformation pro Karte enthält. Diese wird mit Leerzeichen getrennt als Ziffer angegeben. Optional können Kommentare in der Datei enthalten sein, welche für das Programm irrelevant sind.

Ziel ist es, eine Anordnung aller gegebenen Karten zu finden, so dass alle Kanten jeder Karte an einer passenden Nachbarkante liegt (Ziffer 0 zu 0, 1 zu 1 und 2 zu 2).

Das Ergebnis wird in einer Ausgabedatei geschrieben, welche die ursprüngliche Eingabe sowie die Anordnungsinformation, also welches Feld welche Karte besitzt und an welchem Feld die Grundseite der draufliegenden Karte liegt, enthält.

## Eingabe

Die Eingabe erfolgt über eine Textdatei, in der 12 Zeilen mit den dazugehörigen Ziffernfolgen geschrieben stehen, welche die Kanten beschreiben. Kommentarzeilen werden mit einem führenden % gekennzeichnet. Abbildung 2 zeigt ein mögliches Eingabeformat:

%

% Beispiel der Aufgabenstellung IHK

%

0 1 2 1 2

2 2 1 2 0

0 0 1 2 0

1 0 1 1 2

1 2 0 0 1

0 2 2 2 1

1 2 2 1 1

2 1 0 0 2

0 2 1 0 0

2 1 1 0 1

1 1 0 1 1

2 0 1 0 1

Abbildung 2: Eingabebeispiel aus der Aufgabenstellung

## Ausgabeformat

Die Ausgabedatei beinhaltet zu Beginn den Inhalt der Eingabedatei. Im Anschluss erfolgt eine Ausgabezeile, welche die Lösung des Problems angibt (Lösung gefunden / keine Lösung gefunden). Wurde eine Anordnung der Karten gefunden, so folgt eine tabellarische Auflistung der Anordnungsinformation. Diese beinhaltet für jedes Feld die draufliegende Karte, sowie die Information, an welchem Feld die Grundseite dieser Karte liegt. Abbildung 3 zeigt ein Beispiel einer solchen Ausgabedatei:

---------------EINGABE----------------

%

% Beispiel der Aufgabenstellung IHK

%

Karte 0: [0, 1, 2, 1, 2]

Karte 1: [2, 2, 1, 2, 0]

Karte 2: [0, 0, 1, 2, 0]

Karte 3: [1, 0, 1, 1, 2]

Karte 4: [1, 2, 0, 0, 1]

Karte 5: [0, 2, 2, 2, 1]

Karte 6: [1, 2, 2, 1, 1]

Karte 7: [2, 1, 0, 0, 2]

Karte 8: [0, 2, 1, 0, 0]

Karte 9: [2, 1, 1, 0, 1]

Karte 10: [1, 1, 0, 1, 1]

Karte 11: [2, 0, 1, 0, 1]

--------------------------------------

\*\*\*\* Lösung gefunden \*\*\*\*

Eine mögliche Zuordnung der Karten auf die Puzzlefelder:

Feld | Karte | Grundkante an 5-Eck

===========================================================

0 | 1 | 2

1 | 6 | 1

2 | 2 | 2

3 | 3 | 5

4 | 11 | 4

5 | 8 | 5

6 | 5 | 8

7 | 9 | 6

8 | 10 | 5

9 | 12 | 9

10 | 4 | 3

11 | 7 | 8

Abbildung 3: Beispiel einer Ausgabedatei

* 1. Programmanforderung

Das Programm orientiert sich nach dem MVC-Entwurfsmuster. Dabei enthält der Hauptcontroller die main-Funktion. Diese dient als Schnittstelle zwischen der Eingabe, der Puzzle-Schnittstelle und Ausgabe des Programms. Abbildung 4 zeigt diesbezüglich eine Übersicht der Struktur:

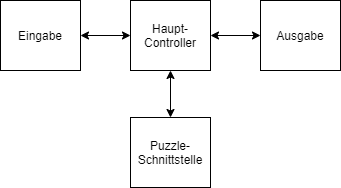


Abbildung 4: Programmanforderung

Der Haupt-Controller liest über die Eingabe-Schnittstelle die Daten ein, leitet diese verarbeitet an die Puzzle-Schnittstelle weiter, sodass das Ergebnis am Ende über die Ausgabe-Schnittstelle in passende Output-Dateien geschrieben werden kann.

Zu beachten ist, dass auf mögliche Fehler angemessen reagiert werden kann. In diesen Fällen soll bei Möglichkeit die passende Fehlermeldung ebenfalls in eine Ausgabedatei geschrieben werden oder alternativ das Programm mit einer informativen Nachricht an den Benutzer beendet werden.

Mithilfe von Testfällen – spezielle Input-Dateien – wird die Qualität und Robustheit der Software überprüft.

## Sonderfälle

Durch die Analyse der Aufgabenstellung und des Eingabeformates ergeben sich folgende Sonderfälle:

* Alle Kanten einer Karte besitzen die gleichen Ziffern.
* Alle Karten besitzen die gleichen Kantenziffern.

## Fehlerfälle

Folgend werden die Fehlerfälle aufgelistet, welche sich in technische, syntaktische und semantische Fehler kategorisieren lassen.

### Technische Fehler

* Es wird versucht, eine oder mehrere Dateien zu lesen, die nicht vorhanden sind.
* Leserechte oder Schreibrechte sind nicht freigegeben.

### Syntaktische Fehler

* Eine Zeile, welche als Kommentarzeile vorgesehen ist, ist nicht mit einem %-Zeichen zu Beginn markiert. Das heißt, diese enthält nicht erlaubte Zeichen.
* Die Kanten sind nicht mit Leerzeichen getrennt notiert.
* Es befindet sich kein Zeilenumbruch nach einer Karteneingabe.

### Semantischer Fehler

* Eine Kante wurde nicht ausschließlich mit den Gewicht 0, 1 oder 2 gewichtet.
* Eine Datei enthält nicht genau zwölf Zeilen als Input.
* Eine Kante innerhalb einer Datei besitzt nicht exakt fünf Kanten.

# Verfahrensbeschreibung

## Vorgehensweise

Zu Beginn werden die Daten mithilfe der übergebenen Argumente des Input- und Output-Pfades eingelesen. Die dadurch resultierenden Zeichenketten werden mithilfe eines Mappers in die benötigten Karten-Objekte konvertiert. Sollte beim Lesen ein Fehler entstehen, so wird die passende Fehlermeldung in die Ausgabe-Datei geschrieben.

Wurden die Karten-Objekte erstellt, wird im Anschluss der Puzzle-Controller inklusive des enthaltenen Puzzles initialisiert. Innerhalb des Puzzle-Controllers befindet sich eine Methode, die zum lösen des Problems zuständig ist. Wurde mithilfe dieser Methode ein Ergebnis berechnet, so werden die zurückgegebenen Daten erneut per Mapper zu den Ausgabe-Format konvertiert. Zuletzt wird eine Ausgabe-Datei mit dem zugehörigen Inhalt aufbereitet und innerhalb des Output-Pfades gespeichert.

Der Lösungsalgorithmus arbeitet nach dem Backtracking-Prinzip. Ausgehend von der Liste aller zu Platzierenden Karten wird versucht für die erste Karte ein passendes Feld zu finden. Dafür wird zunächst innerhalb des Puzzles ein noch nicht besetztes Feld gesucht. Ist dies gefunden, so wird versucht, die aktuelle Karte zu setzen – dabei sind Rotationen der Karte mit inbegriffen. Wurde die Karte erfolgreich gesetzt, so wird der Algorithmus rekursiv mit der nächsten Karte aufgerufen. Ab hier beginnt die gleiche logische Abfolge. Sollte eine Karte nicht platzierbar sein, so wird weiter durch die Puzzle-Felder iteriert. Wenn die Karte auf keines der Felder gesetzt werden kann, so wird die zuvor gesetzte Karte entfernt und es wird auf dieser Ebene weiter durch das Puzzle-Feld iteriert.

## Sonderfälle

Die oben aufgelisteten Sonderfälle werden wie folgt behandelt:

* Alle Kanten einer Karte besitzen die gleichen Ziffern.
  + Die Karte wird nicht rotiert, da eine Rotation das Ergebnis nicht verändert.
* Alle Karten besitzen die gleichen Kantenziffern.
  + Die Karten werden nacheinander iterativ gesetzt, da hier kein Backtracking-Algorithmus benötigt wird.

## Fehlerfälle

* Es wird versucht, eine oder mehrere Dateien zu lesen, die nicht vorhanden sind.
  + Das Programm wird mit einer Fehlermeldung in der Konsole beendet.
* Leserechte oder Schreibrechte sind nicht freigegeben.
  + Das Programm wird mit einer Fehlermeldung in der Konsole beendet.
* Eine Zeile, welche als Kommentarzeile vorgesehen ist, ist nicht mit einem %-Zeichen zu Beginn markiert. Das heißt, diese enthält nicht erlaubte Zeichen.
  + Das Programm generiert eine Output-Datei mit der Fehlermeldung und dem Inhalt der dazugehörigen Input-Datei.
* Die Kanten sind nicht mit Leerzeichen getrennt notiert.
  + Das Programm generiert eine Output-Datei mit der Fehlermeldung und dem Inhalt der dazugehörigen Input-Datei.
* Es befindet sich kein Zeilenumbruch nach einer Karteneingabe.
  + Das Programm generiert eine Output-Datei mit der Fehlermeldung und dem Inhalt der dazugehörigen Input-Datei.
* Eine Kante wurde nicht ausschließlich mit den Gewicht 0, 1 oder 2 gewichtet.
  + Das Programm generiert eine Output-Datei mit der Fehlermeldung und dem Inhalt der dazugehörigen Input-Datei.
* Eine Datei enthält nicht genau zwölf Zeilen als Input.
  + Das Programm generiert eine Output-Datei mit der Fehlermeldung und dem Inhalt der dazugehörigen Input-Datei.
* Eine Kante innerhalb einer Datei besitzt nicht exakt fünf Kanten.
  + Das Programm generiert eine Output-Datei mit der Fehlermeldung und dem Inhalt der dazugehörigen Input-Datei.

# Systembeschreibung

## Haupt-Controller (Controller)

Der Haupt-Controller (Klassenbezeichnung: Controller) besitzt die main-Methode, welche zugleich die Hauptschnittstelle zwischen der Eingabe- und Ausgabe-Komponente ist. Des weiteren ist die Puzzle-Schnittstelle angebunden, welche die Puzzle-Logik beinhaltet

## Model-Classes

Die Model-Classes beinhalten sämtliche Datenklassen, wie die Kanten und Felder.

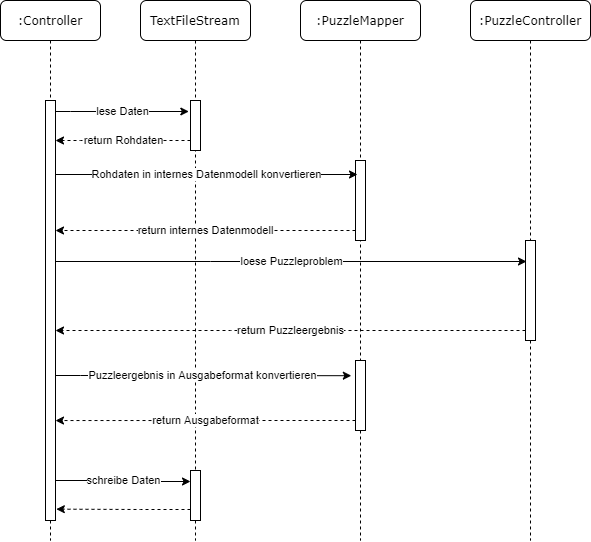
## View-Classes

Die Ausgabe wird über die View-Classes vorgenommen, welche für das Schreiben der Output-Dateien zuständig ist.

## PuzzleController-Classes

Diese beinhaltet sämtliche für das Puzzle benötigten Klassen zum Lösen des Problems.

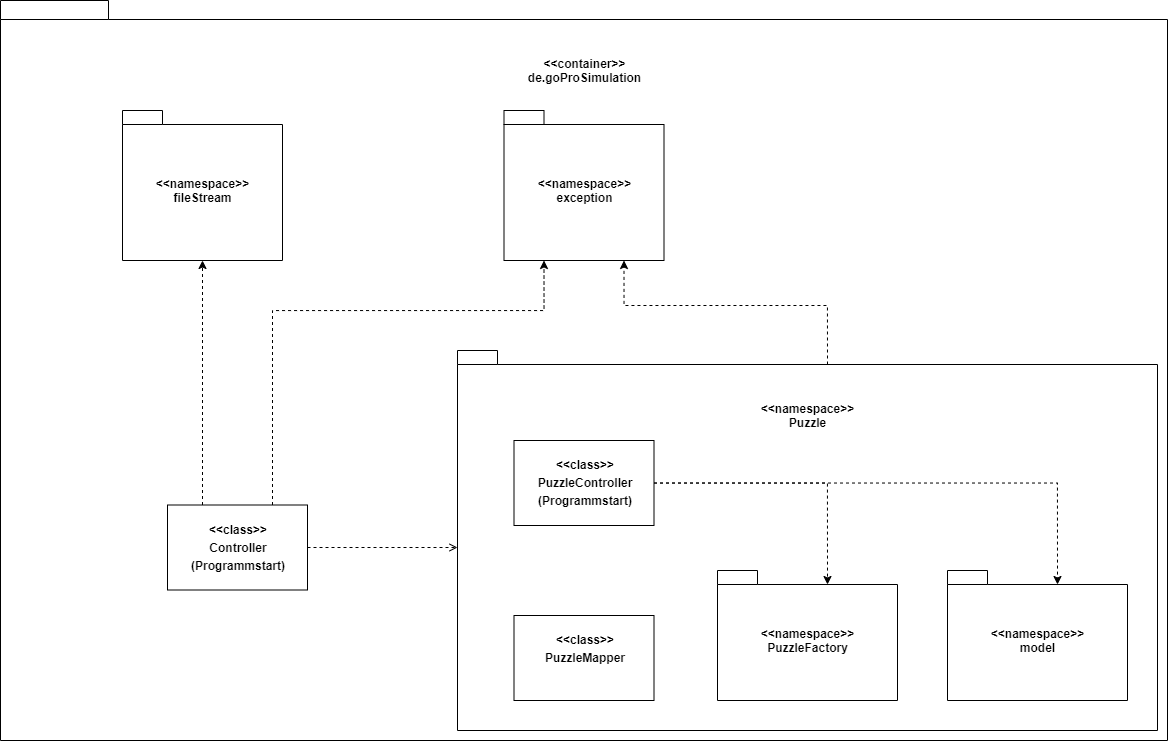
# Datenfluss



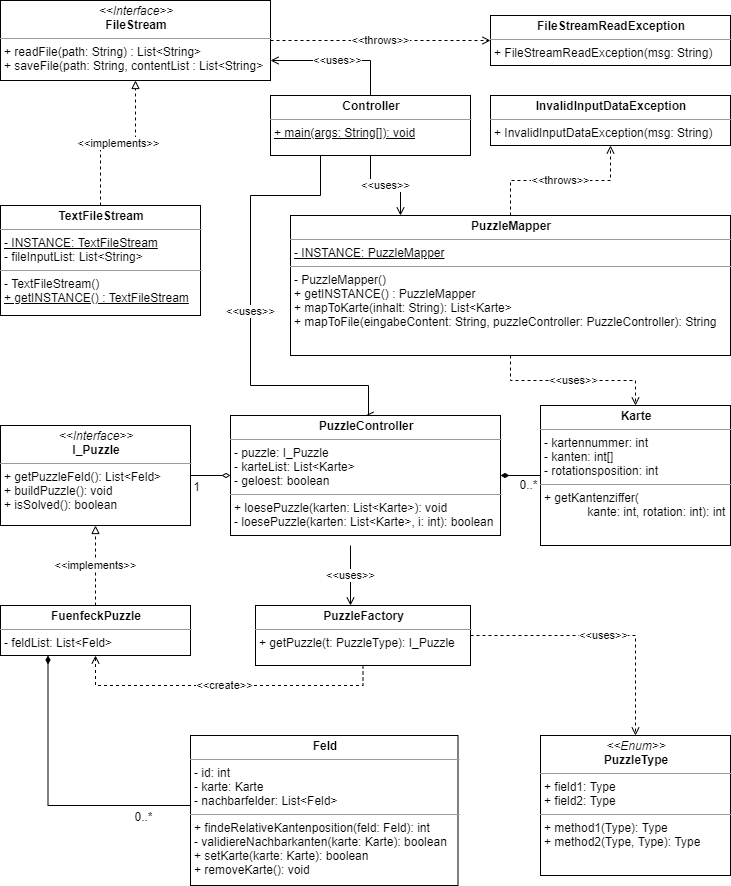
# Programmbeschreibung

## Pakete

Das Programm besitzt die klassische MVC-Model Einteilung und wurde um ein Exception-Package erweitert, welche relevante Fehlermeldungen beinhaltet. Zudem befinden sich im Controller weitere



## Klassenstruktur

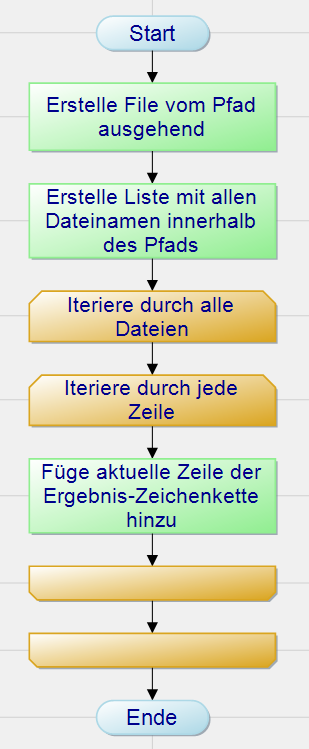


## Präzisierung

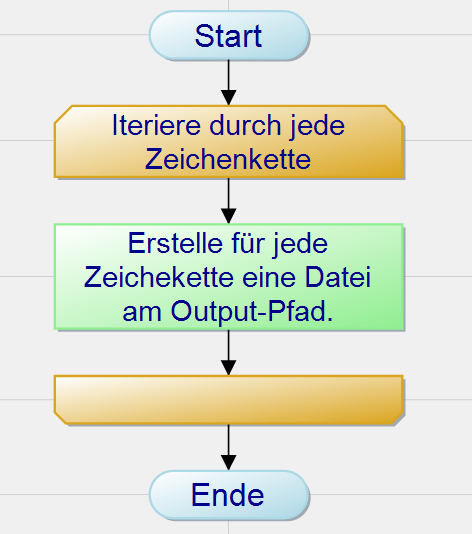
### Hauptcontroller



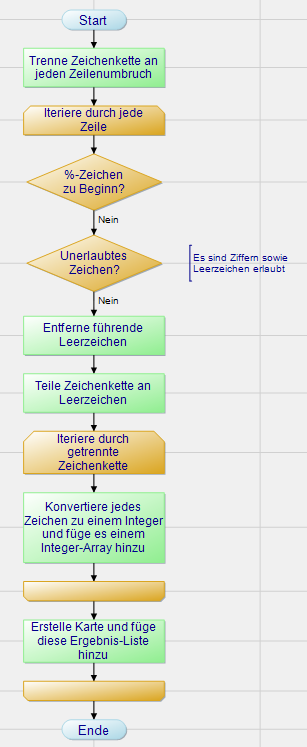
### TextFileStream – Dateien Lesen



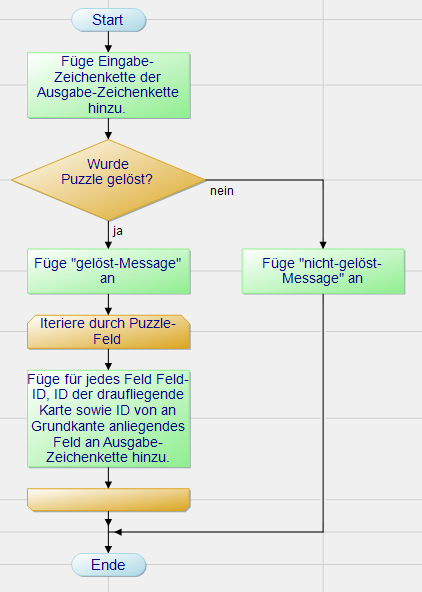
### TextFileStream – SaveFile



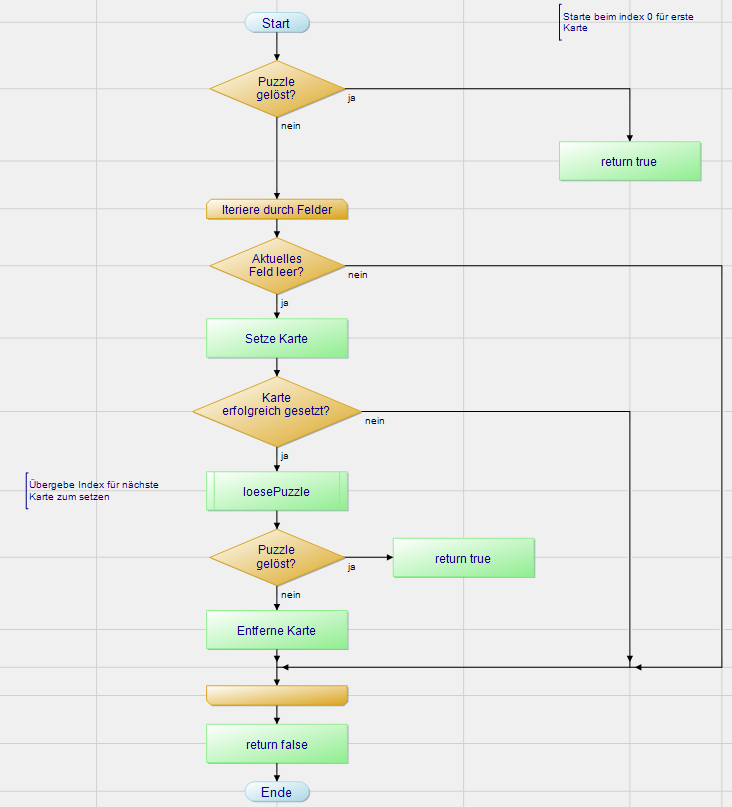
### PuzzleMapper – MapToKarte



## PuzzleMapper – MapToFile



### PuzzleController – LoesePuzzle



# Testdokumentation

Für die Qualitätssicherung wurden zusätzlich Tests erstellt, welche sich innerhalb des Input-Pfades befinden. Die gewährleisten eine funktional vollständige Anwendung, welche jederzeit gültig sein sollte. Durch Veränderungen am Programm kann folglich mit den Systemtests sichergestellt werden, dass weiterhin alle Funktionalitäten fehlerfrei sind.

## Positivfall

Der Positivfall liegt vor, wenn alle Bedingungen für eine gültige Berechnung eingehalten wurden. Dazu zählt das Einhalten der Konsistenz der weiter oben aufgezählten Fehlerfälle in Abschnitt 1.5 und 1.6. Getestet wurde der Normalfall mithilfe des Beispiels aus der IHK-Aufgabe. Zudem wurde ein weiteres Beispiel mit gültiger Eingabe hinzugefügt. Dieses enthält für alle Kanten dieselben Ziffern.

Zudem sind sechs weitere Fälle beigefügt, welche keine gültige Lösung für das gegebene Puzzle besitzen, um sicherzustellen, dass der Algorithmus auch diese richtig behandelt.

# Zusammenfassung und Ausblick

## Zusammenfassung

* Was kann erweitert werden?
* Was kann ausgetauscht werden?
* Stärken / Schwächen?

## Ausblick

* Mögliche Erweiterungen
  + Grafische Oberfläche
  + Weitere Eingabeformate
  + Anderer Algorithmus / Berechnung

# Abweichungen und Ergänzungen zum Vorentwurf

* Was wurde verändert / Ergänzt?
* Wieso wurde es verändert / Ergänzt?