AI Projekt – Zenji

# Problemstellung

Mein Auftrag war es, einen Algorithmus zu schreiben, der das sogenannte Zenji Puzzle löst. Die Basisaufgabe schreibt vor, dass das «Wasser» nur gegen rechts und nach unten fliessen kann.

Das Puzzle hat eine m auf n Fläche, wobei jedes Feld rotiert werden darf. Felder dürfen jedoch nicht ausgetausch oder verschoben werden.

# Bericht

In einem ersten Schritt musste ich verstehen, was das Problem genau ist und welche Regeln darauf zutreffen. Nach einigen Suchanfragen bin ich auf [dieses Paper](https://www.jstage.jst.go.jp/article/ipsjjip/25/0/25_724/_pdf) gestossen, welches die Definition eines Pipe Link Puzzles beschreibt.

Dem Paper zufolge gibt es sieben mögliche Feldertypen.

* Vier Ecken
* Je ein horizontales und vertikales Feld
* Ein Kreuzfeld

Zudem gibt es vordefinierte Felder, die nicht mehr verändert werden dürfen. Das Ziel ist es, jedes Feld zu verbinden, so dass ein kontinuierlicher Fluss entsteht. Es ist nicht erlaubt, ein Feld leer zu lassen. Auf Basis von diesen Regeln habe ich meine ersten Überlegungen gemacht. Die ganze Vernetzung macht das Puzzle jedoch extrem komplex.

Um die korrekten Regeln des Puzzles zu erfahren bin ich an die Fragestunde gegangen. Während dieser haben Sie mir sehr weitergeholfen. Mit den korrekten Regeln und Randbedingungen (siehe Problemstellung) konnte ich mir ein besseres Bild machen. Der erste Lösungsansatz war es, das Puzzle iterativ mit IDA\* zu lösen. Nach einigen Stunden Aufwand hatte ich einen ersten Prototypen, der aber nicht vollständig korrekt funktionierte. Diesen habe ich jedoch zugunsten des A\* Algorithmus auf die Abstellgeleise gestellt.

Mit einigem Grundwissen aus dem vorherigen iterativen Ansatz, setzte ich mich an den A\* Algorithmus. Die Basisimplementation war nicht allzu schwierig. Als erstes kreierte ich ein Feld Objekt. Dieses repräsentiert ein Feld auf dem ganzen Spielbrett. Jedes Feld hat die folgenden Attribute:

* **u(p), r(ight), d(own), l(eft)**: Die Fliessfähigkeit in die jeweilige Richtung, repräsentiert durch die Zahlen 1, 2 und 3.
  + **1**: Eingehend
  + **2**: Ausgehend
  + **3**: Blockiert
* **parent**: Das vorangehende Feld
* **position:** Die Position auf dem Spielfeld, repräsentiert als Tupel
* **rotations:** Die Anzahl an rotationen gegenüber der «Urfeld»
* **g:** Kosten von Feld (0, 0) bis zum momentanen Feld
* **h**: Heuristik, die die «billigsten» Kosten vom momentanen Feld bis zum Ziel berechnet.
* **f**: Die Summe aus g und h.

Rotationen werden im Uhrzeigersinn gemacht. Das heisst die Flussrichtung von «u» wird nach einer Rotation diese von «r» sein.

«h» habe ich so gewählt, dass die Diagonale zwischen dem momentanen Feld und dem Zielfeld mit Hilfe des Satzes von Pythagoras berechnet wird. Zusätzlich potenziere ich diese Zahl mit der Anzahl an Rotationen. Auf diese Weise, werden die Felder bevorzugt, welche weniger rotiert werden müssen.

## Lösungskondition

Ein schwieriger Teil war die Festlegung der Zielkondition. Mein erster Gedanke war es, die Position des Feldes zu prüfen und falls diese die Endposition unten rechts hat, den Algorithmus als fertig einzustufen. Dieser Schritt war jedoch zu allgemein.

Nach einigen Überlegungen bin ich zur folgenden Überprüfung gekommen:

1. Wenn das momentane Feld ein Parent Feld hat, ist es sicher nicht das erste.
2. Wenn das momentane Feld rechts (unter) vom Parent Feld ist, muss das momentane Feld nach links (oben) eine 1 haben und das Parent Feld nach rechts (unten) eine 2 haben.
3. Das momentane Feld muss die Position des vorher definierten Schlussfeldes haben.

Wenn diese 3 Bedingungen zutreffen, dann habe ich ein Schlussfeld gefunden.

## Generierung der Kinderfelder

Dieser Schritt war das grösste Hinderniss bei der Komplettierung dieses Algorithmus.

Zu Beginn des Algorithmus werden die vier möglichen Rotationen des Startfeldes generiert. Falls eine der Rotationen gegen unten und rechts keinen ausgehenden Fluss hat, wird dieses Feld ausgelassen.

Diese Annahme konnte ich in die Generierung der Kinderfelder übernehmen. Für jedes Feld in der «open list» werden die nebenstehenden Felder, nach überprüfung ob die neuen Koordinaten noch im Spielfeld liegen, generiert. Also das Feld unterhalb und das Feld rechts davon.

Für jedes neue Feld werden dann direkt die möglichen Rotationen generiert und diese werden auf ihre Validität überprüft.

1. Überprüfung ob das rotierte Feld überhaupt einen eingehenden Fluss von oben oder von links hat.
2. Überprüfung ob das rotierte Feld auch einen ausgehenden Fluss nach unten von rechts hat.

Wenn 1 und 2 zutreffen, dann ist das neue, rotierte Feld brauchbar. Nun muss noch überprüft werden, ob das vorangehende Feld in die Richtung der neuen Feldes einen ausgehenden Fluss hat. Falls dies auch zutrifft, dann ist das Feld komplett in Ordnung. Nun wird dem Feld das vorangehende Feld als Parent übergeben und kann in die Kinderliste aufgenommen werden.

Jedes Kind erhält danach, falls es nicht bereits in der «closed list» ist, den «g», «h» und «f» Wert.

## Zufallsgenerierung des Spielfeldes

Nebst den von mir erstellten und validierten 2x2 und 4x4 Spielfeld habe ich noch die Möglichkeit eingebaut ein zufälliges Spielfeld generieren zu lassen. Die Pfadsuche dauert natürlich je nach Grösse länger und die Wahrscheinlichkeit einen passenden Pfad zu finden nimmt auch ab.

Die Generierung des Spielfeldes wird so gehandhabt, dass eine Matrix aus Feld Objekten erstellt wird. Diese Felder erhalten für jeden der vier Attribute **b**, **r**, **d** und **l** einen zufälligen Integer Wert zwischen 1 und 3.

Um die generierten Felder abbilden und visualisieren zu können, habe ich eine Funktion geschrieben, die das Spielfeld in die Kommandozeile druckt. Zusammen mit dem Lösungspfad kann die vollständige Lösung somit einfach nachvollzogen werden.

# Herausforderungen

Wie oben bereits erwähnt, waren die Zielprüfung und die korrekte Generierung der Kinderfelder die beiden schwierigsten Schritte.

Probleme die auftauchten waren einerseits der zu frühe Abbruch des Algorithmus weil ein falsches Zielfeld gefunden wurde oder, dass Kinderfelder als valid galten, obwohl sie nicht die korrekte Flussrichtung aufgewiesen haben. Diese Probleme habe ich mit Hilfe von Debugging und einigen hirnzerbrechenden Überlegungen lösen können.

Es gab jedoch auch lustige Probleme, die so gut versteckt waren, dass man sie auf Anhieb nicht gesehen hat. Zum Beispiel habe ich die «rotate» Funktion fälschlicherweise verkehrt implementiert und nicht ausgiebig getestet. Ich bin lange davon ausgegangen, dass diese im Uhrzeigersinn rotiert, musste dann aber peinlicherweise feststellen, dass sie die Listen in die andere Richtung dreht.