Bericht: Teil 1, Problemlösung

Weg aus dem Irrgarten

AG_A_4
Gartlehner Lukas (0926122)
Kuroll Alexander(1029114)
Stiedl Fabian (1029117)
Sietzen Stefan(0372194)

1. Konzept

Bei der Problemstellung unserer Gruppe handelt es sich um das einlesen eines Bildes von einem Labyrinth und eine anschließende Wegfindung von einem Startpunkt zum Ausgang des Labyrinths.

Eingabe:

Ein Farbbild eines Labyrinths von beliebiger Größe.

Ausgabe:

Das Eingangsbild mit dem eingezeichneten Weg als Rote Linie durch das Labyrinth.

Voraussetzungen und Bedingungen:

- Der Kontrast zwischen Wand und Gang muss groß genug sein. Auch auf einem Graustufenbild muss es für einen menschlichen Betrachter möglich sein den Weg einzuzeichnen
- Das eingelesene Bild von einem Labyrinth muss einem Irrgarten ähneln. Es gibt also genau einen Ausgang und dieser muss von Startpunkt aus erreichbar sein
- Der Startpunkt muss ein roter Punkt im Labyrinth sein. Dieser Punkt kann jedoch in Größe variieren
- Es muss genügend Abstand zwischen Gängen und Wänden vorhanden sein. Die Gänge müssen mehrere Pixel (>=5) breit sein um eine Wegfindung zu ermöglichen
- Auch der zu erreichende Ausgang muss markiert sein mit einem grünen Balken damit der Wegfindungsalgorithmus terminiert.
- Etwaige leichte Störungen sollten das Programm nicht davon abhalten den richtigen Weg zu finden. (Salt & Pepper)

Methodik:

1. Das Bild eines Labyrinths von Festplatte einlesen
2. Entfernung etwaiger Störungen durch Median Filter
3. Erkennung von rotem und grünem Bereich

4. Findung des Start und Endpunkts
5. Farbbild wird in Graustufenbild umgewandelt
6. eventuelle Kontraststärkung und weitere Störungsbehandlung
7. konvertieren in ein Binärbild
8. Finden eines Weges aus dem Labyrinth mittels Algorithmus von Pledge

   und gleichzeitig Einzeichnung des Weges.
9. Ausgabe des Ergebnisbilds.

Evaluierung:

Für die korrekte Erkennung des Weges aus einem Labyrinth wird ein Testdatenset im Umfang von mehr als 30 Bildern erstellt. Aus diesem Datenbestand wird ein Trainingsset von 6 Bildern (<= 20%) ausgewählt. Das Trainingsset dient zur Anpassung der Parameter während der Entwicklung.

Die Entwicklung soll iterativ passieren. Bei der Evaluierung der Ergebnisse wird der Gesamte Testdatenbestand verwendet. Folgende Kriterien sind hierbei von Relevanz:

- Wird ein Weg aus dem Labyrinths gefunden?
- Ist der Weg aus dem Ausgabebild nachvollziehbar?
- Welche Eingabebilder verursachen Probleme?
- Könnte man eventuelle Vorbedingungen entschärfen?

2. Arbeitsteilung

Name	Tätigkeiten (Auflistung von Matlab-Funktionen, Kapitel im Bericht, Evaluierungen, etc.)
Lukas Gartlehner	Matlab: Pipline aufbauen, Sobel & Faltung, Binärbilderstellung, Zielkoordinaten bestimmen, Pledge Dokument: Implementierung
Stefan Sietzen	Matlab: Separierung von Rot- und Grünkanal, Autom. Finden eines guten Schwellwertes, Pledge-Verbesserungen Dokument: Methodik
Fabian Stiedl	Matlab: Kontraststärkung, Bildstörungen entfernen Dokument: Konzept
Alexander Kuroll	Matlab: Kontraststärkung, Filterverbesserung Dokument:Evaluierung

3. Methodik

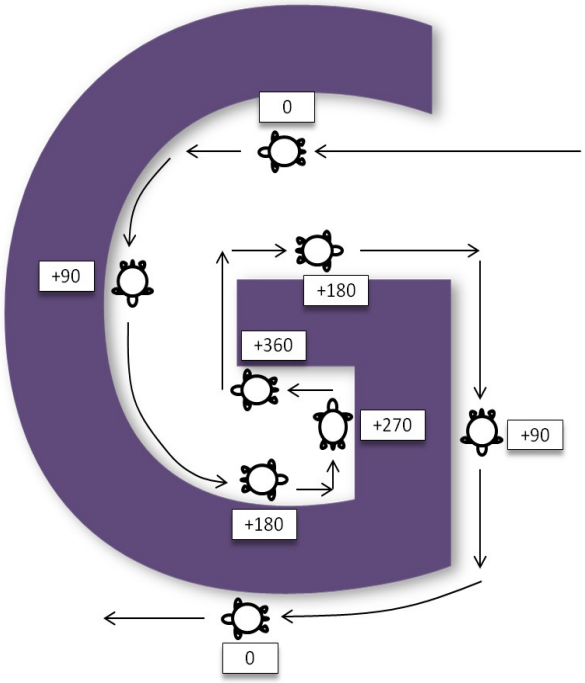
Der bei unserem Projekt umgesetzte Pledge- Algorithmus ist klar definiert, geht aber davon aus, dass Wände und Weg als vorhandene Information vorliegen. Es ging bei uns deshalb vorrangig darum, sich zu überlegen, wie man diese Information aus einem Rasterbild extrahiert. Weiters mussten wir uns überlegen, wie wir aus der reinen Pixelinformation die Startkoordinaten bekommen, und wann der Algorithmus weiß, dass er aus dem Labyrinth gefunden hat und deshalb terminieren muss. Zur

Wanderfassung haben wir verschiedenste Methoden in Erwägung gezogen, von einer Distanztransformation bis zur Richtungserkennung durch Sobel-Gradienten, sind jedoch schlussendlich zu der Erkenntnis gelangt, dass durch Umwandlung des Bildes in ein Binärbild das gesamte Labyrinth nur noch exakt Vertikale und exakt Horizontale Wände enthält (nämlich auf Pixel-Ebene) und dadurch die Wanderkennung sehr stark vereinfacht wird.
Den Startpunkt erkennen wir über eine rot gefüllte Fläche, aus der mittels Centroid-Funktion die XY-Koordinaten des Mittelpunktes gewonnen werden. Der Zielbereich, der den Ausgang sozusagen „versperrt“, damit der Algorithmus nicht daran vorbei kann, wenn er am Ausgang angelangt ist, wird sehr ähnlich erkannt: Der Mittelpunkt wieder mittels Centroid-Funktion, davon ausgehend dann wird in X- und Y-Richtung nach dem ersten nicht-grünen Pixel gesucht und damit Länge und Breite erkannt (wir gehen von einem Rechteckigen Zielbereich aus).
Nun zur Methodik des eigentlichen Pledge-Algorithmus: Grundlage für eine erfolgreiche Wand-Erkennung ist ein sauberes, störungsfreies Ausgangsbild. Deshalb wenden wir einen 3x3 Medianfilter an, der uns bis zu einem gewissen Grad sehr effizient Salt- and Pepper Störungen entfernt. Der Medianfilter wird noch vor der Start- und Endpunkterkennung, in dem RGB-Eingangsbild auf jeden der drei Kanäle angewendet. Anschließend werden je ein getrenntes Bild für den roten Startbereich, den grünen Endbereich und ein Graustufenbild für die Wegverfolgung berechnet. Die Farbkanalbilder sind Binärbilder mit dem Kriterium, dass der jeweilige Kanal um einen gewissen Schwellwert höher ist, als die anderen beiden Kanäle. Das Grauwertbild errechnet sich aus dem Maximum der drei RGB Kanäle. Dadurch wird erreicht, dass die vorher Farbigen Bereiche von Start- und Endpunkt weiß werden und nicht fälschlicherweise als Wand erkannt werden.
Aus dem Grauwertbild gilt es nun durch Anwendung eines Schwellwertfilters ein Schwarzweißbild zu erzeugen. Könnte man davon ausgehen, dass die Helligkeit des Weges immer von 128-255 ist und die der Wände immer von 0-127, würde es genügen, den Schwellwert einfach auf 127 zu fixieren. Wir wollten unser Programm jedoch etwas flexibler gestalten. Deshalb berechnen wir als geeigneten Schwellwert den Durchschnittsgrauwert des gesamten Bildes. Wenn man annimmt, dass die Wände nicht wesentlich weniger Fläche beanspruchen, als die Wege, führt dies immer zu einem Schwellwert, der zwischen dem Grauwert des Weges und der Wände liegt. Dadurch funktioniert unser Programm auch, wenn die Grauwerte z.B. für Weg und Wand 30 und 40 respektive betragen.
Sobald wir das Schwarzweißbild erzeugt haben, können wir mit dem Pledgealgorithmus beginnen. Wir beginnen am bereits erkannten Startpunkt und legen als Startrichtung „nach oben“ fest. Wir gehen nun Pixel für Pixel in diese Richtung, bis wir an eine Wand stoßen. Da Unsere Pledge-Implementierung der „Linke-Hand-Regel“ folgt (Er bewegt sich immer so an der Wand entlang, als ob er sie mit seiner linken Hand durchgängig berührt), drehen wir bei einer Kollision nach rechts:

Gleichzeitig zählen wir den „Dreh-Counter“ um 1 nach oben. Von jetzt an gibt es 3 Möglichkeiten:

- Die gerade beschriebene Kollision mit einer Wand
- Den Fall, dass die Wand zu Ende ist (wir bei Beibehaltung der Richtung „die linke Hand lösen“ müssten) und der Dreh-Counter NICHT auf 0 ist. In diesem Fall drehen wir nach links, um der Wand zu folgen.

– Und schließlich den Fall, dass die Wand zu Ende ist und der Dreh-Counter ungleich 0 ist. In diesem Fall behalten wir die Richtung bei. Diese Ausnahme verhindert, dass bei Vorkommen „Inselteil“ des Labyrinths dieses endlos umrundet wird:



Quelle: Wikipedia

Diese Methode führt bei einem fairen Labyrinth (das heißt es gibt einen Ausgang, der ohne Wände zu überspringen erreichbar ist) immer zum Ausgang. Ein spezieller Punkt kann jedoch nicht angesteuert werden.

4. Literatur

Ein Kapitel eines Lehrbuchs zur algorithmischen Geometrie, welches sich mit der Formalisierung des Problems der „Bewegungsplanung bei unvollständiger Information“ beschäftigt. Erklärt welches mathematische Modell dem Pledge Algorithmus zugrunde liegt und dass dieser vor allem bei der Wegfindung für autonome Roboter von großer Relevanz ist. (Lukas Gartlehner)

@inbook {klein2005,
author = {Rolf Klein },
title = {Algorithmische Geometrie: Grundlagen, Methoden, Anwendungen},
volume = {2},
year = {2005},
pages = {315--325},
}

In dem Kapitel des Buches von Vladimir J. Lumelsky geht es um Motion Planning von Robotern und den Einsatz des Pledge Algorithmus zur Wegfindung in einer nicht bekannten Umgebung.(Fabian Stiedl)

```
@inbook {lumelsky2005,
        author  = {Vladimir J. Lumelsky },
        title   = {Sensing, Intelligence, Motion: How Robots and Humans Move in an Unstructured
World},
        publisher = {Wiley & Sons},
        year    = {2005},
        pages   = {55--70},
}
```

5. Implementierung

Die Implementierung unserer oben beschriebenen Methodik basiert auf einer simplen Pipeline-Architektur. Das main.m Skript fungiert hierbei als Einstiegspunkt in den zentralen Algorithmus. In den ersten Zeilen der main-Funktion werden zuerst alle vorhergehenden Matlab Ausgaben gelöscht. Danach wird ein Flag für den Debug-Mode, welcher Zwischenergebnisse zu den einzelnen Schritten erlaubt, gesetzt. Standardmäßig ist der Debug-Mode auf 0 (deaktiviert) gesetzt. Anschließend wird ein User-Interface-Dialog geöffnet welcher den Benutzer auffordert eine Datei eines gültigen Typs (Bild – jpg, tif, png, gif) zu wählen.

Nach dem erfolgreichen Laden eines Bildes werden zuerst all jene Aktionen durchgeführt die der Vorverarbeitung des Bildes dienen und die darauffolgende Erkennung der Bildinhalte ermöglichen. Zuerst wird ein Medianfilter auf das Bild angewandt.

Dieses Bild wird der splitChannels-Funktion übergeben. Um den Start und Endpunkt für die Pfadsuche zu berechnen wird in dieser Funktion der Rot- sowie Grünkanal aus dem Bild extrahiert. Des weiteren wird auch ein Grauwertbild erzeugt. Durch anwenden der Centroid Funktion (centroid.m) auf den Rotkanal des Eingangsbildes wird der Startpunkt errechnet. All diese Werte werden der main.m als Rückgabeparameter übergeben. Der Grünkanal wird danach an die Hilfsfunktion findMazeExit übergeben. Hierbei wird erneut über die Centroid Funktion der Mittelpunkt des Zielbereichs gesucht. Von diesem Punkt ausgehend wird zuerst solange nach oben gegangen bis der Grünbereich endet, danach geschieht selbiges vom Mittelpunkt nach links. Somit lässt sich das (x,y)-Intervall des Zielbereichs erkennen.

Nächster Schritt in der Pipeline ist die Erzeugung eines Binärbildes. Die Hilfsfunktion goodThreshold liefert dafür einen dynamischen Threshold welcher sich aus dem Mittelwert über die gesamte Bildmatrix errechnet. Nach Erstellung des Binärbildes wird auf das Originalbild noch eine Kantendetektion mittels Sobel-Operator durchgeführt. In der ursprünglichen Konzeption wäre die Identifizierung von Kanten in X- und Y-Richtung als zusätzliche Information für die Pfadentscheidung verwendet worden. Die von uns implementierte Version des Pledge-Algorithmus benötigt diese Kantenbilder jedoch nicht mehr für eine ausreichend genaue Pfadbestimmung.

Alle bis hier beschriebenen Schritte sind als Vorverarbeitung für den eigentlichen Pfadbestimmungsalgorithmus zu betrachten. Mit der Übergabe aller Teilbilder sowie der Pfadstart- und Endpunkte an die pledgePath-Methode wird dieser Aufgerufen.

In der pledgePath.m werden zuerst eine Reihe von Flags gesetzt, welche festlegen wie sich der Cursor verhält. Der Cursor ist dabei ein Koordinatenpaar welches durch den Irrgarten verschoben wird und dabei eine rote Linie „nachzieht". Im Initialmodus bewegt sich der Cursor nach oben (direction=0) und tut dies solange, bis eine Kollision mit einer Wand erkannt wird (collisionFlag=true). In einer while-Schleife wird, wie in der Gameloop eines Computerspiels, jeweils die Position um ein Pixel in die Zielrichtung verändert (Funktion „pledgeCursorShift"). Die Hilfsfunktion „pledgeDetectCollision" prüft mittels einer 5x5 Umgebung und dem Binärbild ob in der aktuellen Richtung eine Wand das weitergehen hindert. Sollten 3 der 5 Pixel einer Richtung schwarz sein, so wird eine Wand erkannt. Für den Fall dass der Cursor sich einer Wand entlang bewegt wird weiters überprüft ob in der selben 5x5 Umgebung eine Wand „an der linken Hand" des Cursors zu finden ist. Die Kollisionsanalyse sowie die Wanderkennung basieren somit nur auf der Analyse einer 5x5 Umgebung im Binärbild.

Über das Intervall des Zielbereichs wird folglich noch geprüft ob das Ziel bereits erreicht ist. Falls dies nicht der Fall ist werden alle Flags und Stati (Richtung, Umdrehungen) an die „pledgeStrategy" Funktion übergeben. Diese setzt je nach Erfordernis die Flags und Stati für die nächste Cursorbewegung.

Um den Code auszuführen muss lediglich die main.m Datei in Matlab geladen und ausgeführt werden. Wenn die Ausgabe von Zwischeninformationen erwünscht ist, kann das Flag „debug" am Beginn der Funktion auf 1 gesetzt werden. Um die Pfadbestimmung in Animationsschritten anzuzeigen belässt man „animate" auf 1, wenn nur das Resultat ohne Animation gewünscht ist, so wird dies über „animate=0" erreicht.

6. Evaluierung

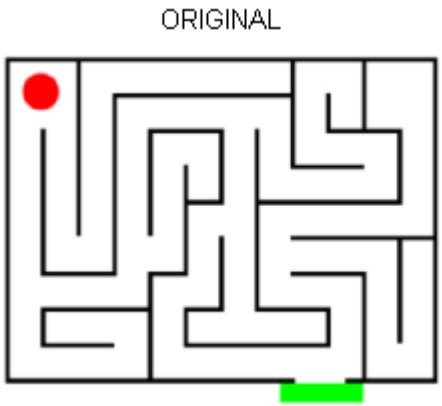
Unser Datensatz enthält 30 Testbilder, welche wir selber mit Hilfe eines Bildbearbeitungsprogramms erstellt haben, um genau unsere Bedingungen für die Findung des Weges einzuhalten. Aus diesem Datensatz haben wir 6 Bilder als „Trainingsset" verwendet, das heißt wir haben nach jedem Entwicklungsschritt immer diese Bilder als Testgegenstand herangezogen, um zu kontrollieren ob die Ergebnisse unseren Erwartungen gerecht sind, beziehungsweise ob die Implementierung eine optimale Performance bereit stellt.
Einige Labyrinthe sind komplexer als andere und brauchen beispielsweise den Umdrehungscounter des Pledge-Algorithmus um auf eine sinnvolle Lösung zu kommen. Andere Labyrinthe haben auch einen Hintergrund der mit Störpixeln versehen wurde oder mehrfarbig ist, sodass die Anwendung von Filtern und die richtige Endpunkterkennung überprüft wird.

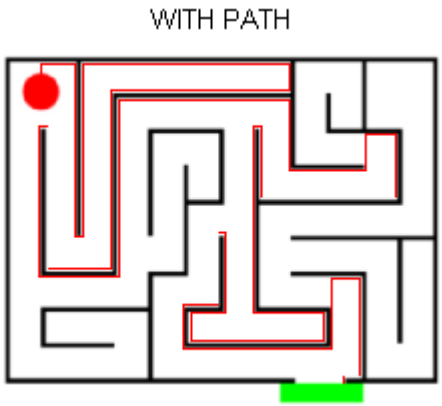
Um konkret die Performance unseres Programmes zu visualisieren haben wir bei jedem folgenden Testbeispiel eine Tabelle mit einigen interessanten Werten erstellt. „Sek" steht für die Exekutionszeit

des Programmes in Sekunden(diese variiert natürlich von Computer zu Computer), Pixel steht für die Größe des zurückgelegten Wegs in Pixeln und Lösbar zeigt ob das Labyrinth gelöst wurde.

Evaluierungskriterien:

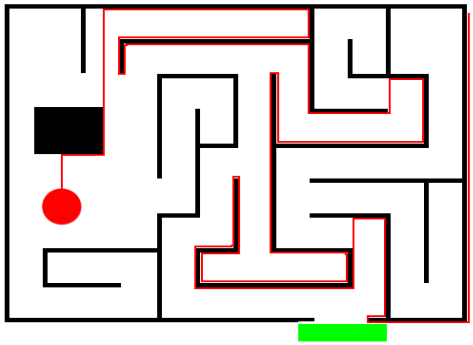
Wird ein Weg aus dem Labyrinth gefunden?
Das ist natürlich das Hauptkriterium für die korrekte Erstellung eines Pfades durch den Irrgarten und stellt gleichzeitig die Hauptaufgabe unseres Projektes dar. Bei diesem Punkt(so wie bei jedem der Kriterien) ist es wichtig, die Bedingungen zu beachten, die wir uns selber gestellt haben. Um den korrekten Weg aus dem Bild zu finden, darf das Labyrinth natürlich visuell nicht zu verzerrt sein und muss klarerweise für einen Menschen eindeutig lösbar sein. Das bedeutet, der Kontrast von Wand- und Wegpixeln darf nicht zu klein sein. Wichtig ist es auch das der grün markierte Ausgang (nahezu) direkt mit dem Spalt in der Labyrinthwand verbunden.
Bei dem Beispiel unten haben wir ein simples und störungsfreies Labyrinth aus dem erfolgreich der Weg gefunden wird.





Sek	*Pixel*	*Lösbar*
3.706820	3784	✓

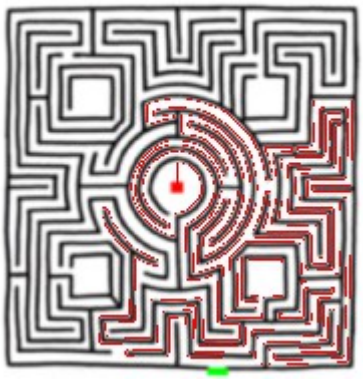
Welche Eingabebilder verursachen Probleme?
Natürlich sind wir während der Entwicklung auf einige Testbilder gestoßen welche zu Problemen bei der Findung des Wegs führen. Wie bereits oben erwähnt darf der grüne Ausgangsbereich nicht zu weit außen liegen, da es sonst zu Problemen mit der Erkennung des Endbereichs kommt und der Weg dann klarerweise außen an der Wand beim Ausgang verläuft und dann falsch terminiert. Unten sieht man ein modifiziertes Bild aus dem Datensatz bei dem dies passiert.



Sek	*Pixel*	*Lösbar*
3.706820	2760 + Pixel nach dem Ziel	x

Desweiteren darf das Bild keine Störungen beinhalten, die der Pledge-Algorithmus als Wand erkennen würde. Jedoch ist eine Bildstörung erlaubt, solange man mit den eigenen Augen erkennt wie der Weg im Labyrinth verläuft. In dem Beispiel unten wurde ein sogenannter „Spreadnoise“-Filter an einem Labyrinth angewandt. Dieser verschiebt die Pixel zwar zwischen einem bestimmten Zahlenbereich zufällig , in diesem Fall jedoch nicht so stark dass man den Weg nicht mehr erkennen kann. Wenn man die Performance-Ergebnisse unseres Programms ohne und mit dem Noise vergleicht merkt man an dass die Main-Funktion ca. 8,7 Sekunden länger braucht um den Weg zu finden. Somit vergrößern Bildstörungen möglicherweise die Exekutionszeit, der (gleiche) Weg wird aber trotzdem gefunden.

No Spread-Noise



With Spread-Noise

	Sek	*Pixel*	*Lösbar*
No Spread-Noise	60.738691	18042	✓
Spread-Noise	68.608601	19878	✓

Könnte man eventuelle Vorbedingungen weglassen?

Als wir uns das Konzept für unser Projekt überlegt haben , mussten wir einschätzen wie viele Vorbedingungen wir benötigen. Es gibt zwar noch einige, jedoch sind diese während der Entwicklung dezimiert worden. Die Bedingung dass das Labyrinth nur aus eckigen Kanten besteht haben wir zum Beispiel bei Seite gelassen, da der Pledge-Algorithmus letztenendes auch bei rund verlaufenden Wänden funktioniert hat.
Fraglich ist es schließlich auch, ob diese Vorbedingungen noch weiter dezimiert werden können. Eine Bedingung besagt das der Gang genügend breit sein muss, was natürlich wichtig ist, jedoch muss er nicht größer-gleich 5 Pixel breit sein, da der eingezeichnete Weg selbst nur 2 Pixel breit ist.
Sinnvoll wäre es eventuell auch wenn man den Ausgang nicht farbig markieren müsste, dass ist jedoch mit der von uns gewählten Implementierung nicht möglich, wäre aber durchaus realisierbar.

7. Schlusswort

Die zentrale Erkenntnis bei unserer Arbeit was dass vor allem die Einfachheit eines Algorithmus in diesem Kontext von großem Vorteil ist. Denn dadurch wird auch ein hohes Maß an Robustheit gewährleistet. Die Komplexe Problemstellung in unserem Konzept wurde von uns auf ein sehr simples Entscheidungsproblem reduziert. Auf der Grundlage eines Binärbildes entscheiden wir jeweils ob und in welche Richtung wir uns um einen Pixel bewegen. Komplexere Ansätze welche beispielsweise die Richtung von Kanten über den Sobel-Gradienten miteinbezogen hätten oder versuche die Struktur des Labyrinths zu analysieren scheitern an der Heterogenität der zu verarbeitenden Daten, denn vorrangiges Ziel bei der Entwicklung war es, die Pfadbestimmung in Bildern mit unterschiedlichster Struktur zu ermöglichen. Zu unserer eigenen Überraschung ermöglichte es uns dieser Ansatz auch mit relativ geringen Anpassungen Labyrinthe mit gekrümmt verlaufenden Wänden zu lösen.

Verbesserungen wären in diesem Problem-Setting vor allem bei der Performance des Algorithmus möglich. Denn die Robustheit des Algorithmus von Pledge geht mit dem Probieren vieler möglicher Lösungen einher. Durch die Erkennung von Grenzfällen könnte jedoch der Ablauf der Wegfindung optimiert werden. Wären beispielsweise Start- und Endpunkt nicht durch Wände getrennt, müsste der Algorithmus nicht an der Wand entlang gehen, sondern könnte einen direkten Pfad bestimmen. Auch gewisse Informationen über die Struktur des Labyrinths könnten in den Entscheidungsprozess einfließen und beispielsweise Sackgassen von vornherein als nicht betretbar kennzeichnen.

In Bezug auf die Qualität der Lösung wäre natürlich das Kriterium des kürzesten Weges eine anspruchsvolle und interessante Herausforderung an die Optimierung unserer Lösung. Doch hierbei käme die Lösung auf Ebene eines Binärbildes mittels Pledge an ihre Grenzen. Würde man eine solche Optimierung anstreben ließe sich dies wahrscheinlich am ehesten durch die Überführung der Wegdaten in einen Graphen bewerkstelligen.

Was uns aus dieser Arbeit bleibt ist jedoch, abseits der Entwicklung eines Algorithmus, wohl die Erkenntnis dass ein komplexes Problem durch die Zerlegung in viele kleine Probleme oft geradezu trivial lösbar ist.