

DOKUMENTATION

36 Bundeswettbewerb Informatik
2. Runde / Aufgabe 3

Quo vadis, Quax?

Teilaufgabe (a)

Bis vor kurzen hatte ich Teilaufgabe (a) noch so verstanden, dass es quasi nur ein Teil von Teilaufgabe (c) ist. Allerdings wurde vor kurzem auf www.einstieg-informatik.de ein Thread über unter anderem diese Teilaufgabe (a) erstellt, bei dem es heißt: „Insbesondere muss man dem Modul Flugergebnisse, wie sie anhand der Strategie aus (b) anfallen, übergeben können.“¹

Ich hatte die Aufgabenstellung „Schreibe ein Programmmodul, das aus den Ergebnissen durchgeführter Flüge bestimmt, [...]“ so verstanden, dass die durchgeführten Flüge die des Pathfinding-Algorithmus für eine ausgewählte Map wären und nicht welche einer eigenen Eingabe.

Ich hatte nun leider nicht mehr genug Zeit die Teilaufgabe (a) zu implementieren, so dass der Anwender selber Flüge einstellen kann.

Teilaufgabe (b)

Lösungsweg:

Um die möglichst wenige Quadrate zu scannen überlegte ich mir, dass die Drohne zuerst ein möglichst großes Gebiet scannen muss und dann dieses Gebiet, wenn es gemischt ist, in kleinere Gebiete zerlegen muss, die dann wiederum gescannt werden. Das heißt im einfachsten Fall, einer quadratischen Map ohne Wasser (siehe Abbildung 1), bräuchte es nur einen Flug, um den Weg von Quax zur Stadt zu finden.

Wenn ein gescanntes Gebiet gemischt ist, ist es am effizientesten das Gebiet in 4 gleichgroße Teilgebiete aufzuteilen (siehe Abbildung 1).

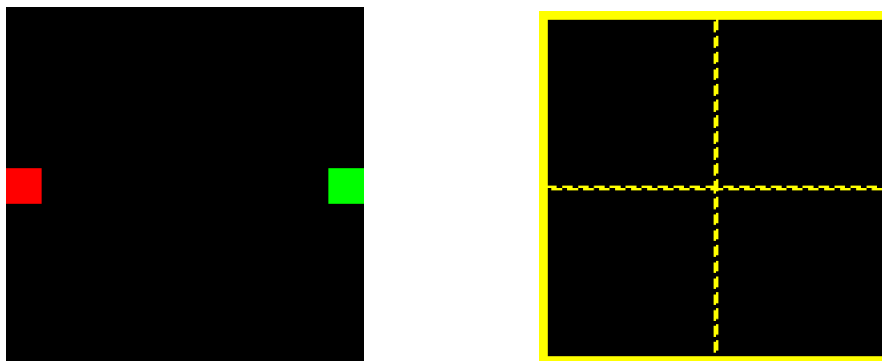


ABBILDUNG 1

¹ <https://www.einstieg-informatik.de/community/forums/topic/572/fragen-zu-aufgabe-3-quax>
(abgerufen am 08.04.2018)

Die daraus resultierende Datenstruktur (siehe Abbildung 2) heißt Quadtree oder auch Quaternärbaum.

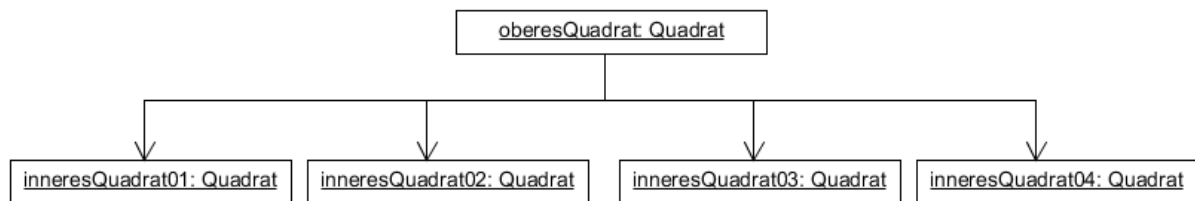


ABBILDUNG 2

Nun wollte ich diese Datenstruktur für meinen Algorithmus verwenden. Allerdings war der Start anders als bei anderen Pathfinding Algorithmen. Während andere Algorithmen schon beim Ausgangspunkt, also hier die Position von Quax, anfangen, fängt mein Algorithmus erst beim größtmöglichen Quadrat an. Das heißt also, dass mein Quadtree als erstes Quax suchen muss. Das geht noch relativ einfach, indem man einfach so lange das Quadrat in dem Quax sich befindet in innere Quadrate aufteilt, bis das Quadrat das Quax enthält Land ist, oder gemischt und eine Größe von 20x20 hat. Hierbei ist anzumerken, dass alle Pixel, die nicht die Farbe Weiß (#FFFFFF) besitzen, als Land gewertet werden.

Das Start Quadrat der Suche ist dann das ober Quadrat des gefundenen Quadrates. Nun muss der Weg vom Start Quadrat zur Stadt gefunden werden. Um nun diesen Algorithmus zu finden, hatte ich mir ein kleineres Beispiel (siehe Abbildung 3) überlegt, nach welchem ich den Algorithmus entwerfen konnte.

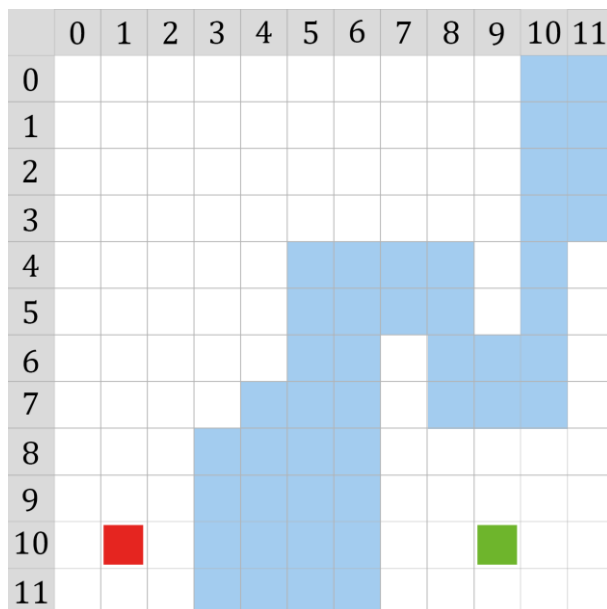


ABBILDUNG 3

Zuerst wird die Suche nach Quax wie oben geschildert durchgeführt. Das Ergebnis mit eingezeichnetem Quadtree sehen Sie in Abbildung 4.

Ein gemischtes Quadrat hat eine hellbraune Füllfarbe, ein passierbares Quadrat (Nur Land oder gemischt und größer als 20x20) eine grüne und ein nicht passierbares Quadrat ist rot.

Das Start Quadrat des Pathfindings zur Stadt ist hierbei dann das Quadrat mit den Eckpunktkoordinaten (0 | 6) links oben und (5 | 11) rechts unten. Das Quadrat (0 | 9), (5 | 11) ist nun das erste Quadrat des End Weges.

Nach meiner Strategie, wird nun das innere Quadrat überprüft, dass das letzte Quadrat des End Weges berührt und am nächsten zur Stadt ist.

Dies wäre in diesem Fall das Quadrat (3 | 9), (5 | 11), welches jedoch nicht passierbar ist. Deswegen wird das zweit nächste Quadrat (3 | 6), (5 | 8) untersucht. Dieses ist gemischt und größer als 20x20, was heißt, dass es in innere Quadrate zerlegt wird. Da es jedoch keine 1,5 Pixel große Quadrate geben kann, wird jedes Quadrat auf eine Breite von 2 aufgerundet.

Nun beginnt der Algorithmus wieder von vorne und da es nur ein Quadrat gibt, das den letzten Weg berührt, wird dieses überprüft. Das Quadrat (3 | 7), (4 | 8) ist hierbei passierbar, da Quax laut Aufgabenstellung ein gemischtes Quadrat der Größe 20x20 passieren kann.

Abbildung 5 zeigt den aktuellen Stand des Quadtree.

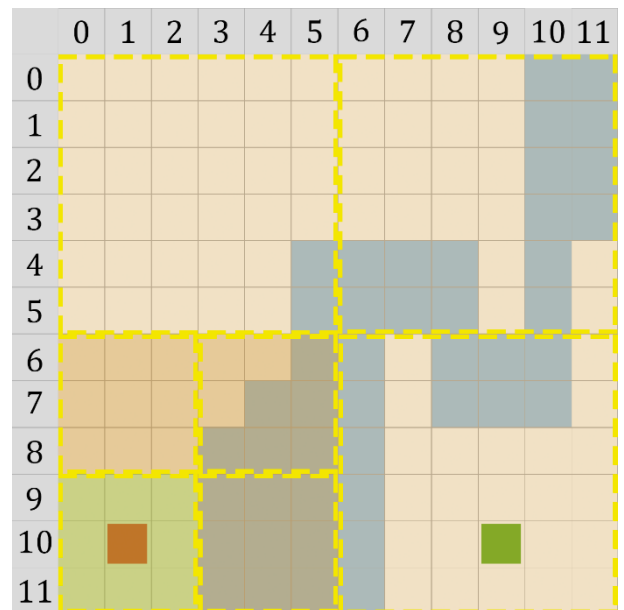


ABBILDUNG 4

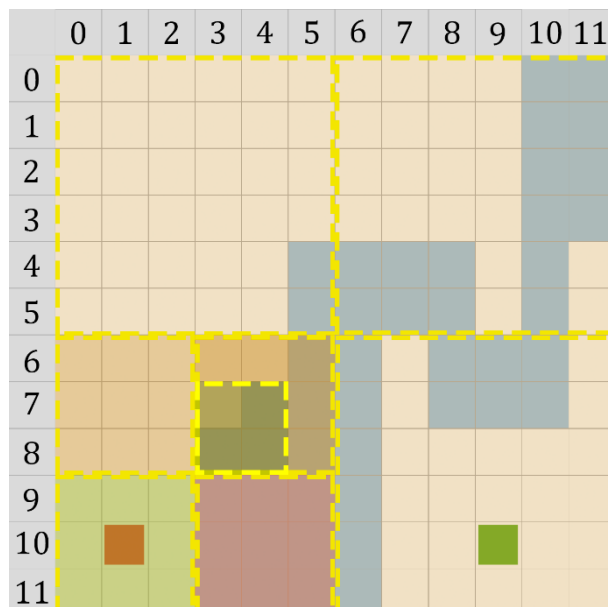


ABBILDUNG 5

Nun brauchen wir aber noch ein weiteres inneres Quadrat vom Quadrat (3 | 6), (5 | 8), das passierbar ist. Hat ein Quadrat nämlich mindestens zwei passierbare, innere Quadrate, so ist dieses Quadrat passierbar. Hier gibt es auch Sonderfälle, auf die ich in Teilaufgabe (c) zu sprechen komme.

Das Quadrat (3 | 6), (5 | 8) besitzt bisher nur ein passierbares inneres Quadrat, also muss das nächste innere Quadrat untersucht werden. Das Quadrat (4 | 7), (5 | 8) ist hierbei jedoch wieder nicht passierbar und erst das dritte innere Quadrat (4 | 6), (5 | 7) ist passierbar.

Nun hat das Quadrat (3 | 6), (5 | 8) zwei passierbare innere Quadrate. Somit ist dieses Quadrat passierbar. Außerdem ist eben das Quadrat (0 | 9), (2 | 11) passierbar, was bedeutet, dass das ober Quadrat (0 | 6), (5 | 11) passierbar ist.

Dies bedeutet, dass wir wieder ein Quadrat im Quadtree zurückgehen und beim größten Quadrat (0 | 0), (11 | 11) landen.

Dieses Quadrat hat nun ein passierbares inneres Quadrat. Folglich wird wieder das innere Quadrat überprüft, dass den letzten Weg berührt (Das Quadrat (4 | 6), (5 | 7) ist das zuletzt zum Weg hinzugefügte Quadrat) und am nächsten zur Stadt ist. Dies ist das Quadrat (6 | 6), (11, 11). Dieses ist wieder gemischt und es müssen die inneren Quadrate geprüft werden.

Abbildung 6 zeigt den aktuellen Stand des Quadtree.

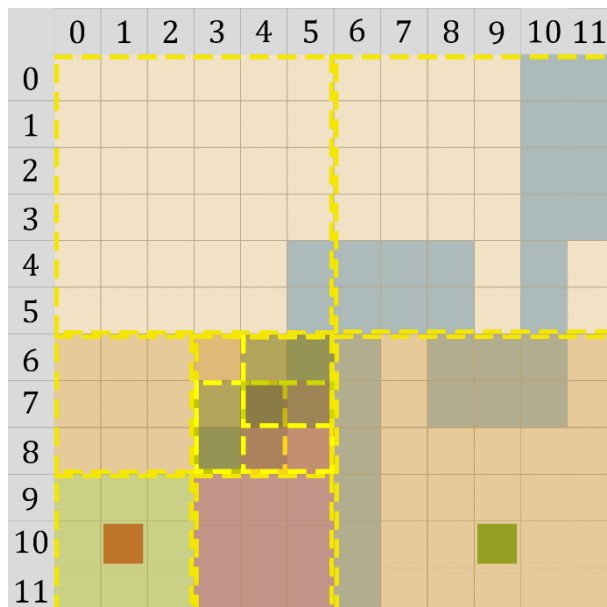


ABBILDUNG 6

Der Algorithmus geht nun immer so weiter, bis zum passierbaren Quadrat (7 | 9), (8 | 10), welches die Stadt berührt. Damit ist das Pathfinding abgeschlossen und die gesammelten Quadrate für den Weg ergeben den Pfad von Quax zur Stadt (siehe Abbildung 7).

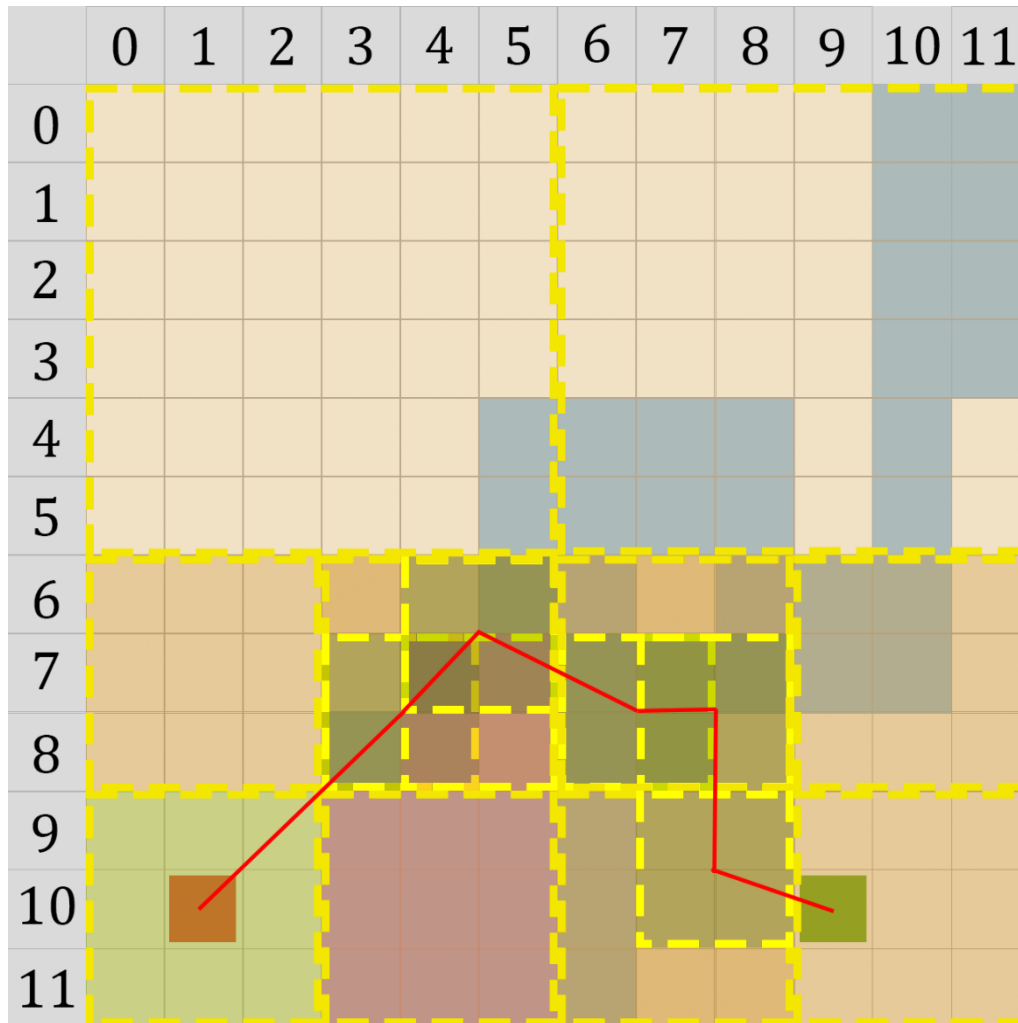


ABBILDUNG 7

Spezialfälle:

Nicht quadratische Map

Alle BwInf Beispiele sind nicht quadratische Bilder. Dies ist deshalb ein Problem, da Quax nur quadratische Bereiche scannen kann. Mir sind zwei mögliche Lösungen für dieses Problem eingefallen:

1. Ich erweitere die Map zu einem Quadrat, indem ich den hinzugefügten Bereich mit Wasser fülle, wie es in der Aufgabenstellung festgelegt ist.
2. Ich definiere mehrere Start Quadrate, die so groß wie möglich, das Rechteck füllen (siehe Abbildung 8).

Ich habe mich für die zweite Lösung entschlossen, da ich glaube, dass diese weniger zusätzliche Quax Flüge benötigt, als wenn ich die Map mit Wasser erweitere und somit ein mögliches Land Quadrat in ein Gemischt Quadrat umwandle.

Die zweite Lösung erfordert jedoch auch nochmal einen extra Algorithmus, da ich damit nichtmehr nur einen Quadtree pro Map habe, sondern N Quadrees wobei N die Anzahl der Start Quadrate ist. Diese N Quadrees, sind im Worst-case jedoch alle am Pathfinding beteiligt. Auch ist es wahrscheinlich, dass die Start Quadrate sich über mehrere Pixel überschneiden.

Der Algorithmus für ein Pathfinding über mehrere Quadrees hinweg, ist auch nicht allzu komplex. Bei der Quax Suche, wird der erste Quadtree ausgewählt, der die Quax Position enthält. Bei diesem wird dann auch das Pathfinding zur Stadt ausgeführt, bis das Start Quadrat fertig gescannt ist. Dann wird das Start Quadrat als nächstes gewählt, welches den letzten Weg berührt und am nächsten zur Stadt ist.



ABBILDUNG 8

Teilaufgabe (c)

Bedienung:

Das Programm ist eine Windows Presentation Foundation (WPF) Applikation geschrieben in C# mit Visual Studio.

Bei Programmstart wird zuerst das Hauptfenster (siehe Abbildung 9) geöffnet.

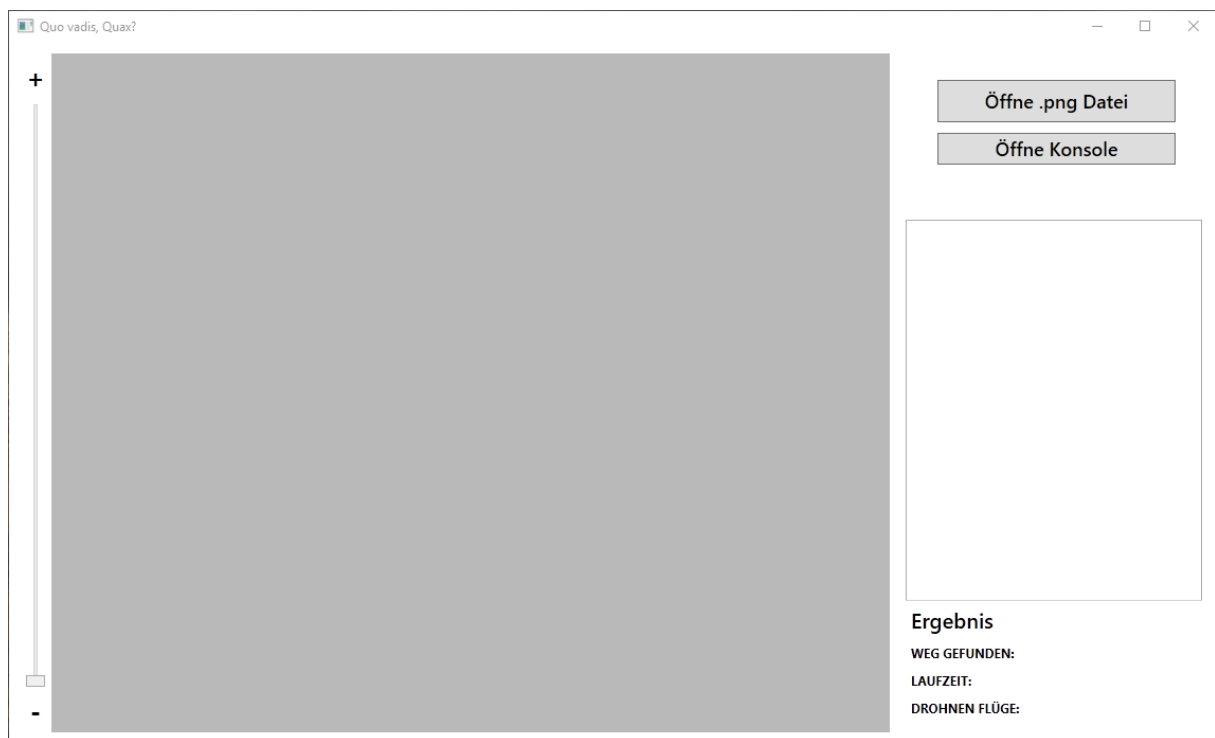


ABBILDUNG 9

Über den „Öffne .png Datei“ Button, kann man eine Map vom Typ png auswählen und über den „Öffne Konsole“ Button, wird ein neues Fenster geöffnet, das später die einzelnen Schritte des Algorithmus genau ausgibt.

Nachdem eine Map geöffnet wurde, wird diese verarbeitet und in der linken ScrollView angezeigt. Außerdem, werden die verschiedenen Quax Positionen als einzelne Tabs rechts angezeigt (siehe Abbildung 10). Jedes Quax Positions Tab besitzt ein Bild der Map, auf dem

die jeweilige Quax Position Gelb angezeigt wird. Außerdem einen „**Starte Algorithmus**“ Button, der das Pathfinding startet. Und eine Liste mit den ausgeführten Drohnen Flügen.

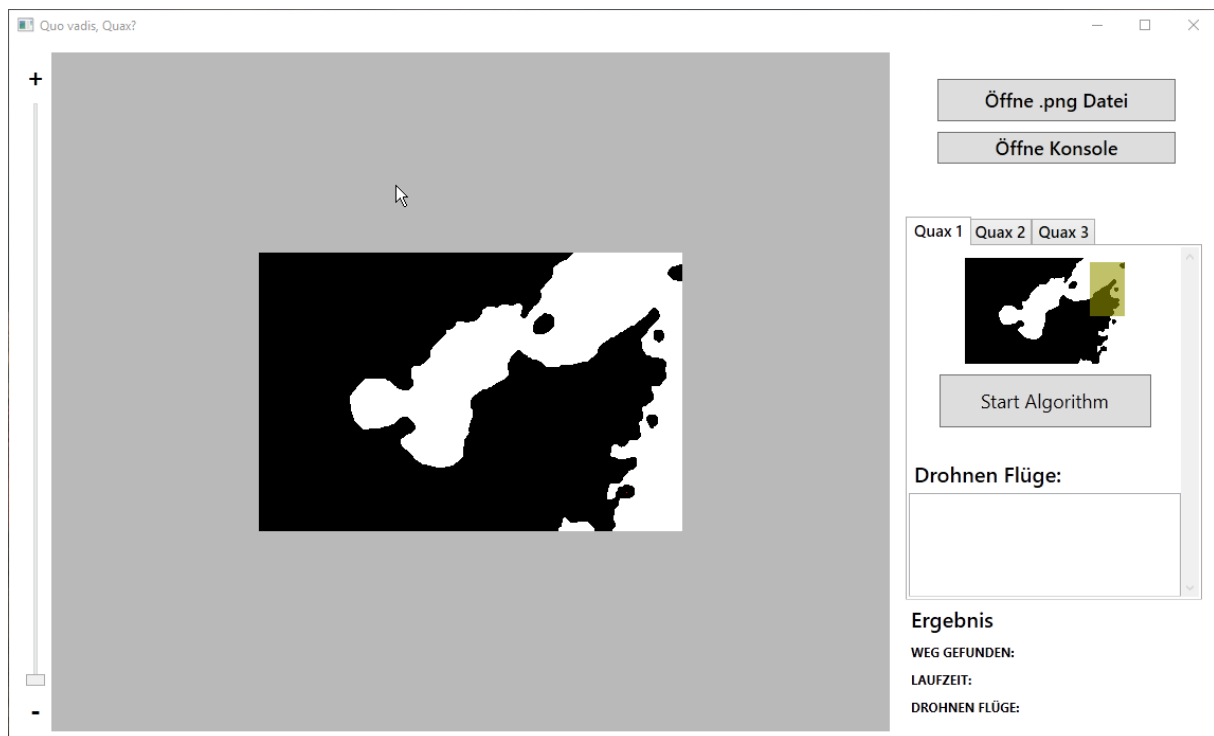


ABBILDUNG 10

Wurde der Algorithmus gestartet, berechnet das Programm kurz den Weg, falls es einen gibt, und benachrichtigt dann den Benutzer über ein PopUp-Fenster, ob ein Weg gefunden wurde (siehe Abbildung 11). Unten rechts werden die groben Daten des Algorithmus dauerhaft angezeigt. Außerdem, werden die durchgeführten Drohnen Flüge mit ihrem jeweiligen Ergebnis als Füllfarbe in die Map eingezeichnet und die Liste im Quax Positions Tab mit den Flügen in Form [FlugID Ergebnis] gefüllt.

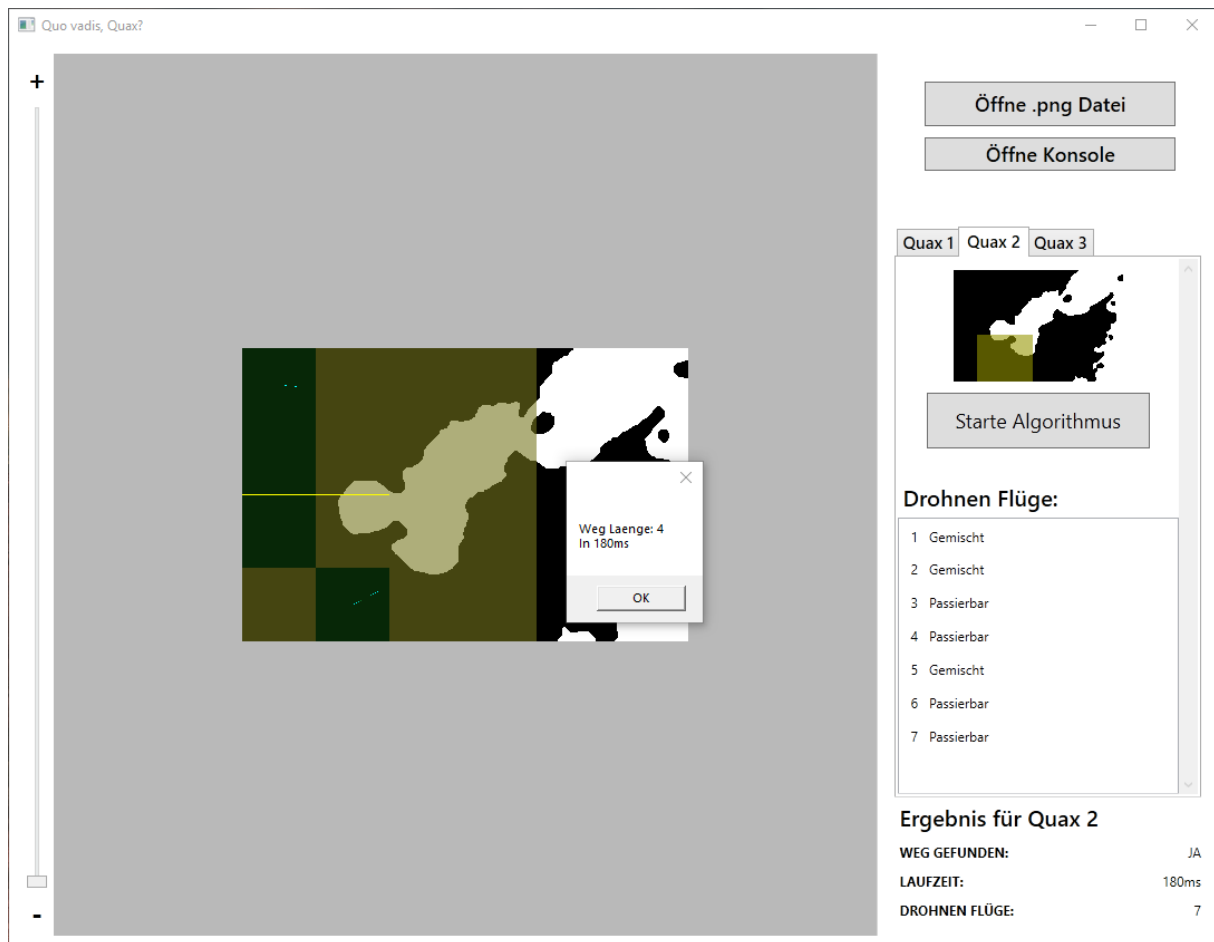


ABBILDUNG 11

Wurde außerdem vorher die Konsole geöffnet, kann man da die einzelnen Schritte des Algorithmus sehen (siehe Abbildung 13). Ein Schritt hat folgende Form:

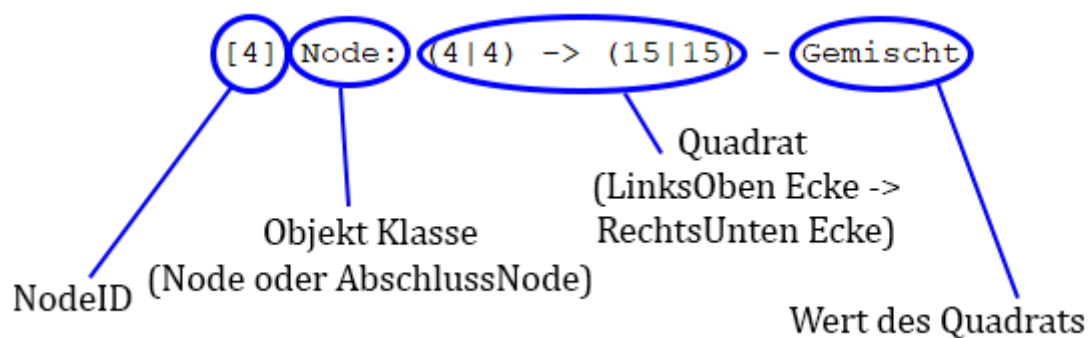
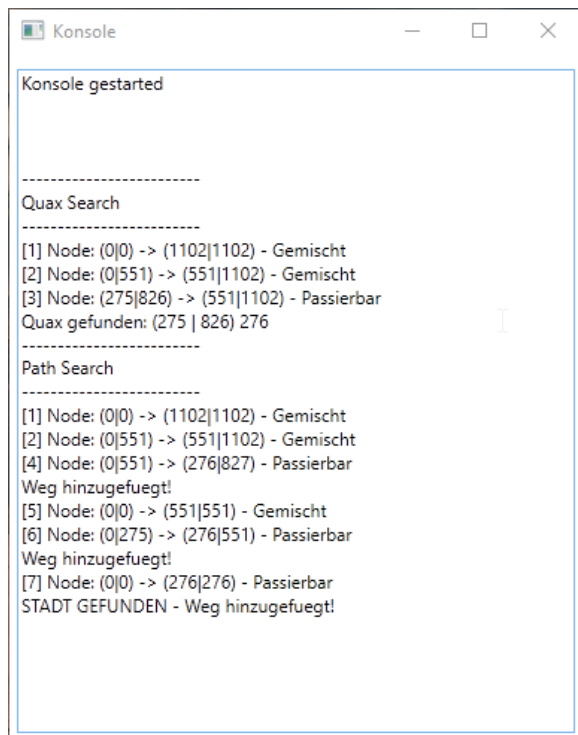


ABBILDUNG 12



```
Konsole gestartet

-----
Quax Search
-----
[1] Node: (0|0) -> (1102|1102) - Gemischt
[2] Node: (0|551) -> (551|1102) - Gemischt
[3] Node: (275|826) -> (551|1102) - Passierbar
Quax gefunden: (275 | 826) 276

-----
Path Search
-----
[1] Node: (0|0) -> (1102|1102) - Gemischt
[2] Node: (0|551) -> (551|1102) - Gemischt
[4] Node: (0|551) -> (276|827) - Passierbar
Weg hinzugefuegt!
[5] Node: (0|0) -> (551|551) - Gemischt
[6] Node: (0|275) -> (276|551) - Passierbar
Weg hinzugefuegt!
[7] Node: (0|0) -> (276|276) - Passierbar
STADT GEFUNDEN - Weg hinzugefuegt!
```

ABBILDUNG 13

Außerdem kann man, indem man einen Drohnen Flug in der rechten Liste auswählt, diesen In der Map hervorheben, damit man sieht, welcher Drohnen Flug welcher Ausschnitt in der Map ist (siehe Abbildung 14).

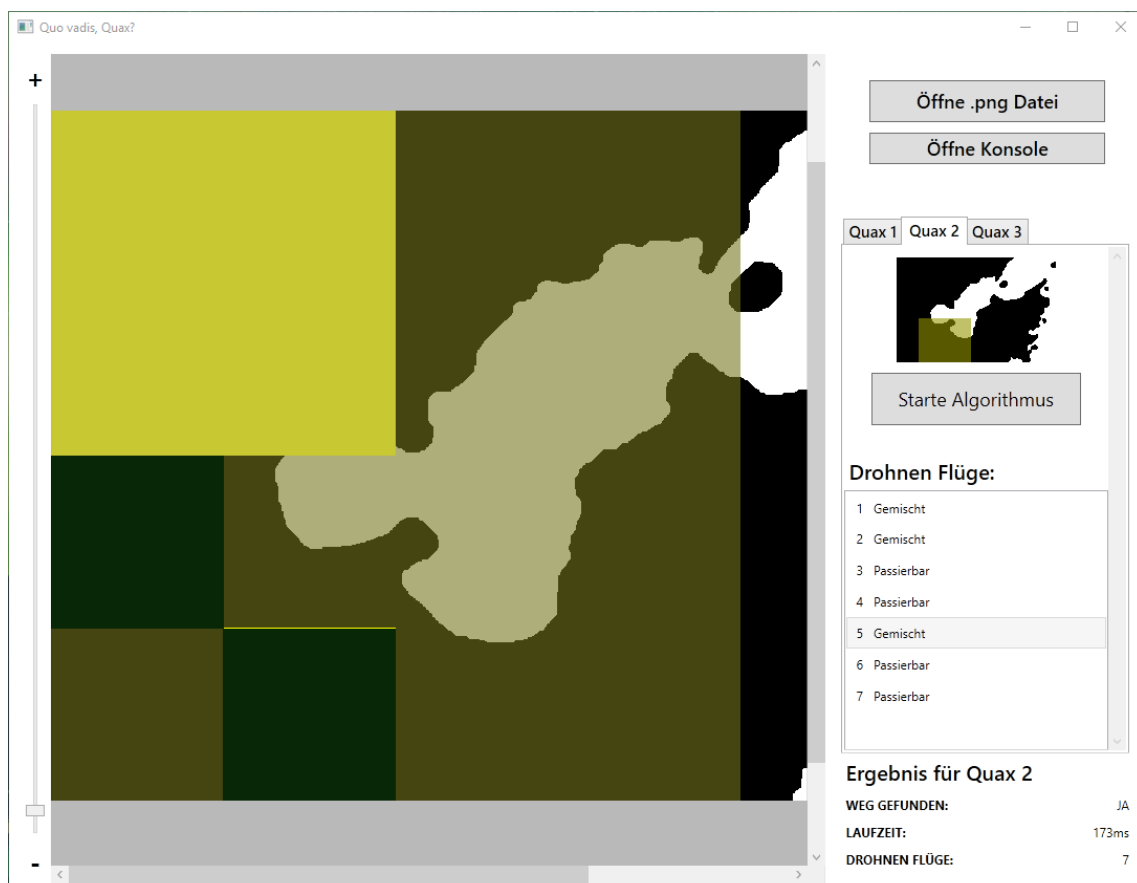


ABBILDUNG 14

Umsetzung:

Im Folgenden wird nur auf Umsetzung des Pathfindings eingegangen und zum Beispiel GUI spezifischer Code weggelassen.

Die Hauptdatenstruktur des Algorithmus ist, wie vorher schon angesprochen ein beziehungsweise mehrere Quadrees. Die Elemente des Quadrees, genannt Node, unterscheiden sich in AbschlussNode und Node, wobei AbschlussNode ein Quadrat der Breite 20x20 repräsentiert. Das besondere bei AbschlussNodes ist, dass sie eben keine inneren Quadrate mehr haben. Im Gegenteil dazu referenziert die Klasse Node ein Objekt der Klasse ChildNodes, welche wiederum vier Objekte der Klasse QuadratNode referenziert.

Die Klasse QuadratNode wiederum, erbt die abstrakte Klasse NodeElement, die die Methoden SearchQuax und SearchPath implementiert. Die Klasse Pathfinder ist quasi der Manager des Pathfindings. Über ihre Methode FindPath wird das Pathfinding gestartet. Dazu referenziert die Klasse ein Objekt der Klasse Map, die auch von NodeElement erbt und die einzelnen Wurzeln der verschiedenen Quadrees in einer Liste speichert.

Jedes Quadtree Element hat eine Referenz auf ein MapQuadrat Objekt. Die Klasse MapQuadrat erbt von der Klasse Quadrat, welche die wichtigsten Attribute eines Quadrats implementiert und dazu die Methoden BeruehrtQuadrat und BeruehrtPoint. Die Klasse MapQuadrat implementiert dazu noch einen Enum vom Typ MapTypen. Das Enum MapTypen besitzt vier Werte: Unbekannt (der Typ des Ausschnitts wurde noch nicht untersucht), Gemischt (der Typ des Ausschnitts ist gemischt und größer als 20x20), Passierbar (der Typ des Ausschnitts ist entweder Land oder Gemischt und gleich 20x20) oder Wasser (der Typ des Ausschnitts ist Wasser).

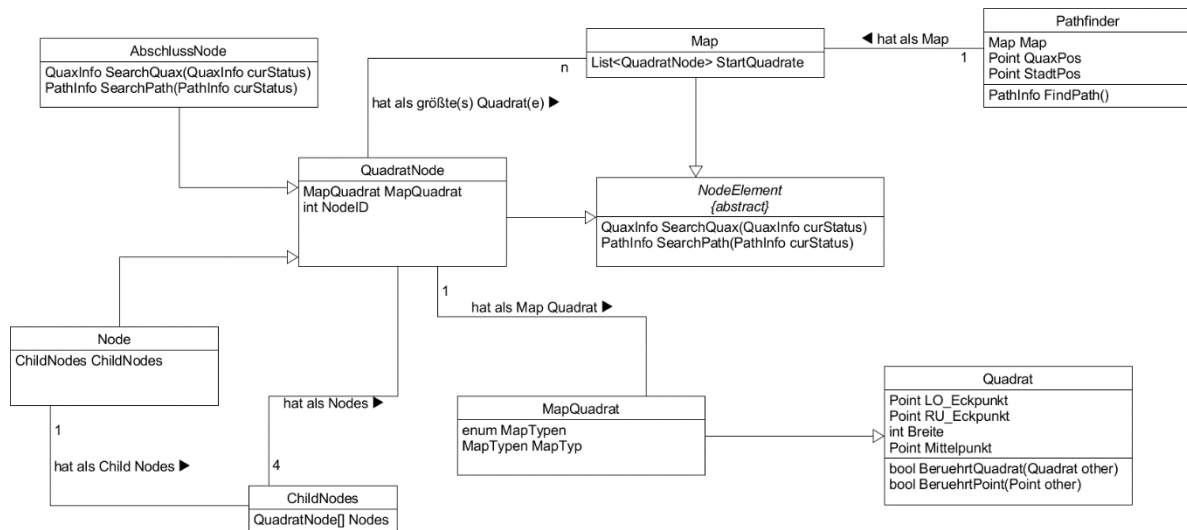


ABBILDUNG 15

Abbildung 16 zeigt ein Beispiel Objektdiagramm einer kleineren, quadratischen Map.

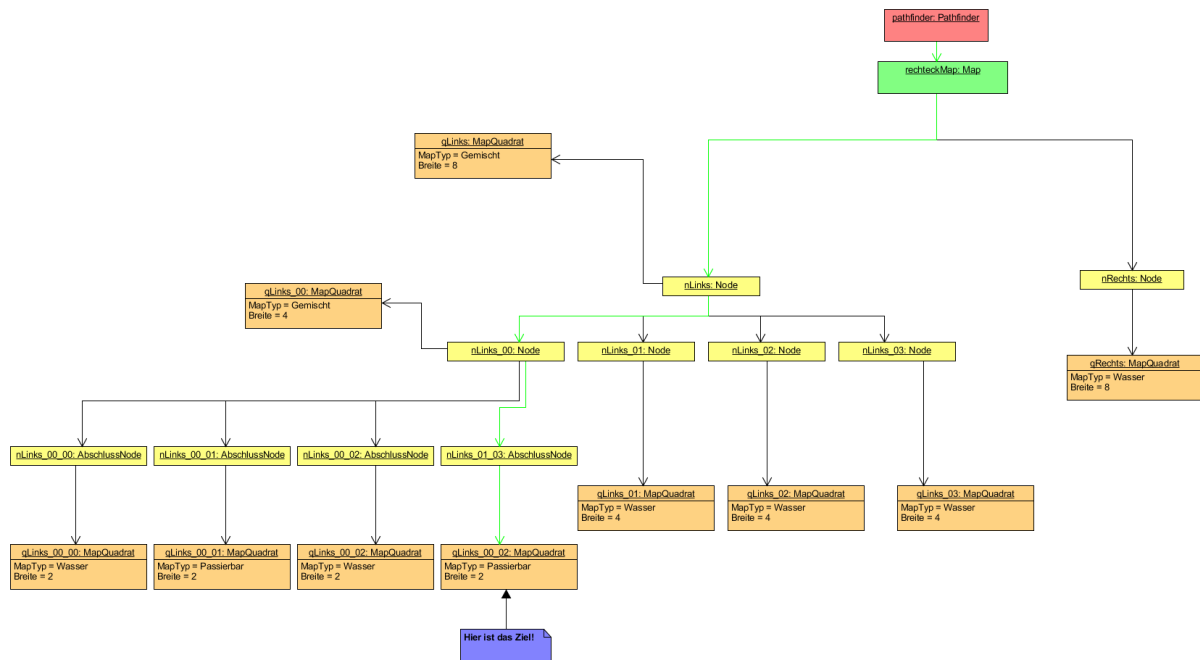


ABBILDUNG 16

Die Methoden SearchQuax und SearchPath

Kommen wir nun zu den Implementierungen der Methoden SearchQuax.

Die Methode SearchQuax hat ein Objekt der Klasse QuaxInfo als Übergabe- und Rückgabewert.

QuaxInfo
Point QuaxPos QuadratNode QuaxNode

Das Attribut QuaxPos ist hierbei die genaue Position von Quax.

Das Attribut QuaxNode speichert die Node in der Quax ist.

Klasse Map:

```
public override QuaxInfo SearchQuax(QuaxInfo curStatus)
{
    QuadratNode quaxStartNode = null;
    for (var i = 0; i < StartQuadrat.Count; i++)
        if (StartQuadrat[i].BeruehrtPoint(curStatus.QuaxPos))
            quaxStartNode = StartQuadrat[i];

    if (quaxStartNode != null)
    {
        Debug.Write("START QUADRAT:    ");
        var info = quaxStartNode.SearchQuax(curStatus);
        if (info.QuaxNode == null)
            info.QuaxNode = quaxStartNode;
        return info;
    }

    throw new Exception("Quax ist in keinem der Start Quadrate");
}
```

In der Klasse Map müssen wir nur die Start Node finden, die Quax enthält. Bei dieser können wir rekursiv die Methode SearchQuax aufrufen. Den Rückgabewert speichern wir in der

Variable info. Ist das Attribut QuaxNode von info dann bereits gesetzt können wir info einfach zurückgeben. Ansonsten ist die aktuelle Start Node die QuaxNode.

Klasse AbschlussNode:

```
public override QuaxInfo SearchQuax(QuaxInfo curStatus)
{
    if (MapQuadrat.MapTyp == MapQuadrat.MapTypen.Unbekannt)
    {
        if (NodeID == 0)
            NodeID = MapQuadrat.GetMapTyp();
        else
            MapQuadrat.GetMapTyp();
    }

    switch (MapQuadrat.MapTyp)
    {
        case MapQuadrat.MapTypen.Wasser:
            throw new Exception("Quax ist in Wasser Node");

        case MapQuadrat.MapTypen.Passierbar:
            curStatus.QuaxNode = this;
            return curStatus;

        case MapQuadrat.MapTypen.Gemischt:
            throw new Exception("Abschluss Node ist gemischt");
    }
}
```

In der Klasse AbschlussNode wird als erstes dem Quadrat, falls noch nicht geschehen, ein MapTyp zugewiesen. Je nach MapTyp wird dann weitergemacht. Bei SearchQuax sollte jedoch nur der Fall MapQuadrat.MapTypen.Passierbar auftreten, da Quax weder in Wasser stehen, noch eine AbschlussNode gemischt sein kann. Wenn die AbschlussNode dann passierbar sein sollte, was im Normalfall immer auftreten müsste, wird die QuaxNode gesetzt und das QuaxInfo Objekt zurückgegeben.

Klasse Node:

```
public override QuaxInfo SearchQuax(QuaxInfo curStatus)
{
    if (MapQuadrat.MapTyp == MapQuadrat.MapTypen.Unbekannt)
    {
        if (NodeID == 0)
            NodeID = MapQuadrat.GetMapTyp();
        else
            MapQuadrat.GetMapTyp();
    }

    switch (MapQuadrat.MapTyp)
    {
        case MapQuadrat.MapTypen.Wasser:
            Console.WriteLine("Wasser");
            throw new Exception("Quax ist in Wasser Node");

        case MapQuadrat.MapTypen.Passierbar:
            curStatus.QuaxNode = this;
            return curStatus;

        case MapQuadrat.MapTypen.Gemischt:
            Console.WriteLine("Gemischt");

            QuadratNode nodeMitQuax = null;

            for (var i = 0; i < ChildNodes.Nodes.Length; i++)
                if (ChildNodes.Nodes[i].BeruehrtPoint(curStatus.QuaxPos))
                    nodeMitQuax = ChildNodes.Nodes[i];

            if (nodeMitQuax == null)
                throw new Exception("Quax ist in keinem inneren Quadrat");

            return nodeMitQuax.SearchQuax(curStatus);
    }
}
```

Die Implementierung der Methode SearchQuax in der Klasse Node ähnelt sehr der Implementierung in der Klasse AbschlussNode. Auch hier wird erst der MapTyp überprüft und je nachdem dann gehandelt. Normal kann Quax nie in Wasser stehen. Wenn die Node passierbar ist, wurde Quax gefunden. Ist die Node jedoch gemischt, werden die ChildNodes der Node überprüft. Es wird die child node herausgesucht, die Quax enthält und dann bei der Node die Methode SearchQuax rekursiv aufgerufen.

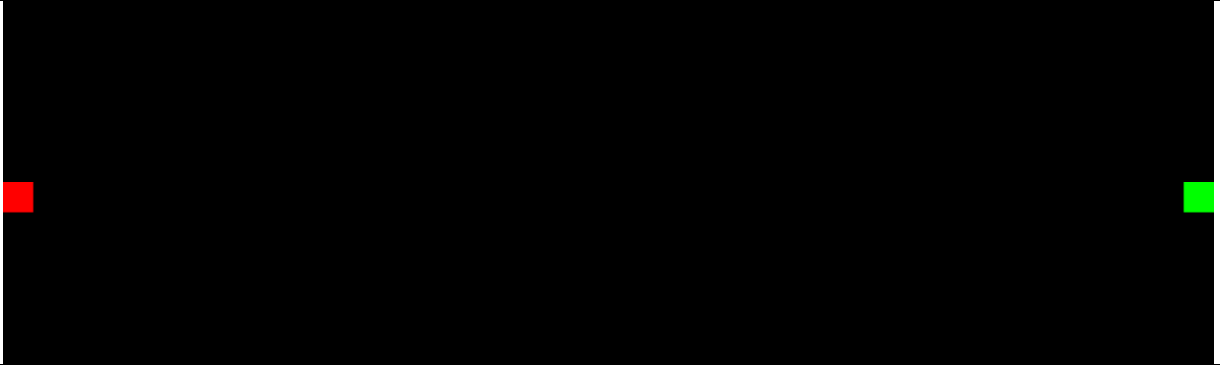
Beispiele:

Test Map 01:

Eine einfache Test Map, zum Testen des Algorithmus in der Klasse Map für mehrere Quadrees.

Die Map wird in 4 Quadrees aufgeteilt. Da die Map aber kein Wasser besitzt, bestehen alle Quadrees nur aus einer passierbaren Wurzel. Dann wird von links nach rechts durchgegangen. Immer zu dem Quadrat, welches den letzten Weg berührt und am nächsten zur Stadt ist.

Map:



Ausgabe:

Quo vadis, Quax?

+

Öffne .png Datei

Öffne Konsole

Quax 1

Starte Algorithmus

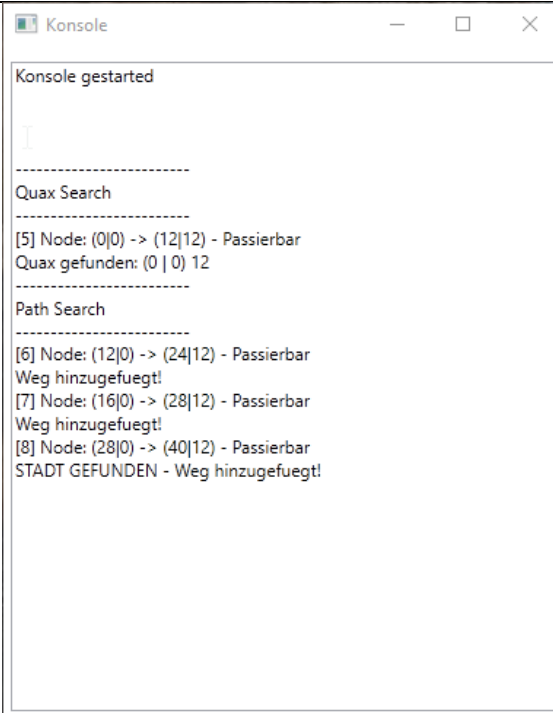
Drohnen Flüge:

- 1 Passierbar
- 2 Passierbar
- 3 Passierbar
- 4 Passierbar

Ergebnis für Quax 1

WEG GEFUNDEN:	JA
LAUFZEIT:	13ms
DROHNEN FLÜGE:	4

Konsole:

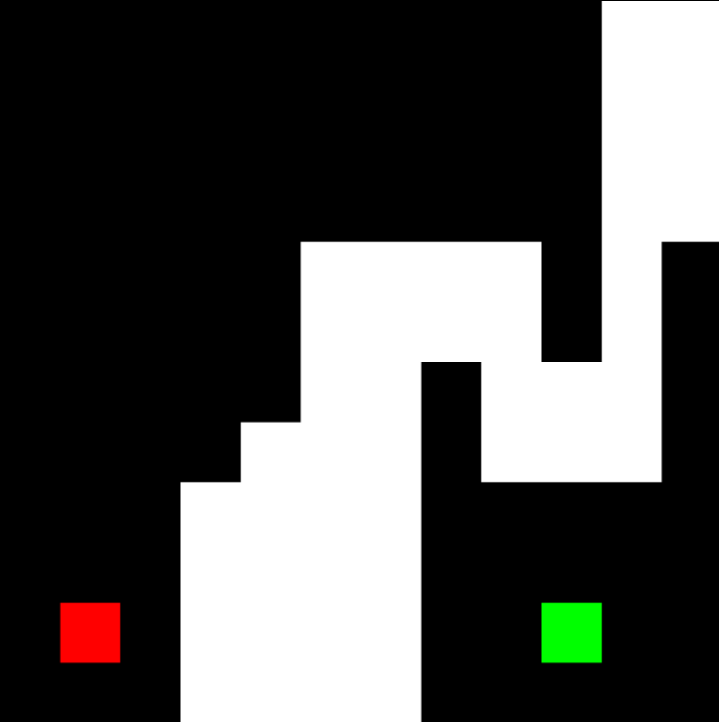
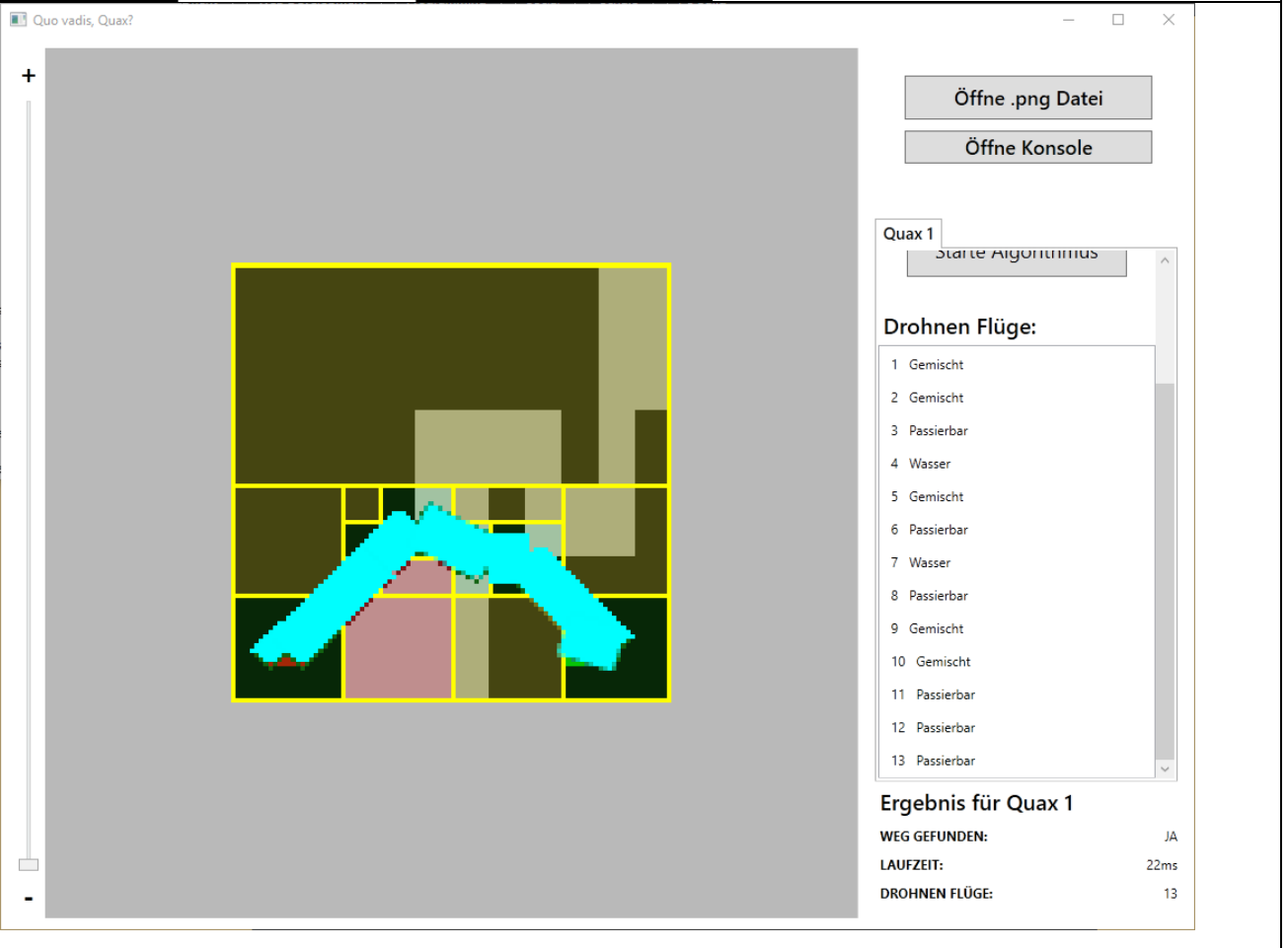



```
Konsole gestartet

|
-----
Quax Search
-----
[5] Node: (0|0) -> (12|12) - Passierbar
Quax gefunden: (0 | 0) 12
-----
Path Search
-----
[6] Node: (12|0) -> (24|12) - Passierbar
Weg hinzugefuegt!
[7] Node: (16|0) -> (28|12) - Passierbar
Weg hinzugefuegt!
[8] Node: (28|0) -> (40|12) - Passierbar
STADT GEFUNDEN - Weg hinzugefuegt!
```

Test Map 02:

Dies ist die Test Map, nach der ich meinen Algorithmus entworfen habe.

Map:	
Ausgabe:	

Konsole:	<div data-bbox="244 197 805 235"> Konsole</div> <div data-bbox="244 257 805 907"><pre>----- Quax Search ----- [1] Node: (0 0) -> (12 12) - Gemischt [2] Node: (0 6) -> (6 12) - Gemischt [3] Node: (0 9) -> (3 12) - Passierbar Quax gefunden: (0 9) 3 ----- Path Search ----- [1] Node: (0 0) -> (12 12) - Gemischt [2] Node: (0 6) -> (6 12) - Gemischt [4] Node: (3 9) -> (6 12) - Wasser [5] Node: (3 6) -> (6 9) - Gemischt [6] Abschluss Node: (3 7) -> (5 9) - Passierbar Weg hinzugefuegt! [7] Abschluss Node: (4 7) -> (6 9) - Wasser [8] Abschluss Node: (4 6) -> (6 8) - Passierbar Weg hinzugefuegt! [9] Node: (6 6) -> (12 12) - Gemischt [10] Node: (6 6) -> (9 9) - Gemischt [11] Abschluss Node: (6 7) -> (8 9) - Passierbar Weg hinzugefuegt! [12] Abschluss Node: (7 7) -> (9 9) - Passierbar Weg hinzugefuegt! [13] Node: (9 9) -> (12 12) - Passierbar STADT GEFUNDEN - Weg hinzugefuegt!</pre></div>
-----------------	--

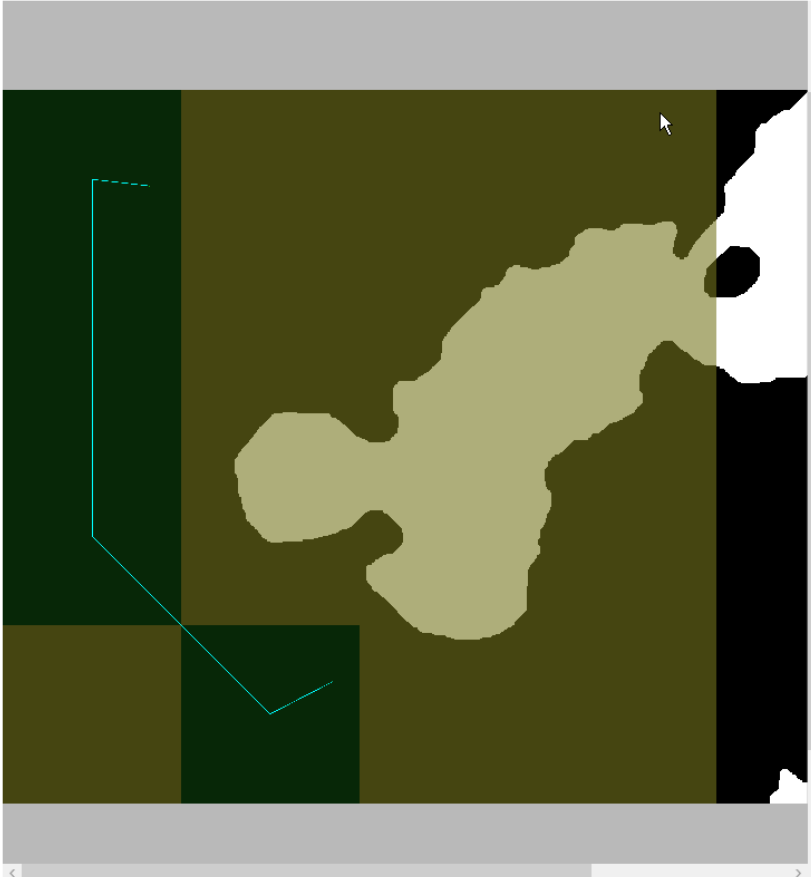
BwInf Map 01 – Quax 2:

Die erste Map von der BwInf Seite mit der zweiten Quax Position

Map:	
-------------	--

Ausgabe:


Quo vadis, Quax?



Öffne .png Datei

Öffne Konsole

Quax 1 Quax 2 Quax 3



Starte Algorithmus

Drohnen Flüge:

- 1 Gemischt
- 2 Gemischt
- 3 Passierbar
- 4 Passierbar
- 5 Gemischt
- 6 Passierbar
- 7 Passierbar

Ergebnis für Quax 2

WEG GEFUNDEN: JA

LAUFZEIT: 215ms

DROHLEN FLÜGE: 7

Konsole:

Konsole

Konsole gestartet

Quax Search

[1] Node: (0|0) -> (1102|1102) - Gemischt
[2] Node: (0|551) -> (551|1102) - Gemischt
[3] Node: (275|826) -> (551|1102) - Passierbar
Quax gefunden: (275 | 826) 276

Path Search

[1] Node: (0|0) -> (1102|1102) - Gemischt
[2] Node: (0|551) -> (551|1102) - Gemischt
[4] Node: (0|551) -> (276|827) - Passierbar
Weg hinzugefüegt!
[5] Node: (0|0) -> (551|551) - Gemischt
[6] Node: (0|275) -> (276|551) - Passierbar
Weg hinzugefüegt!
[7] Node: (0|0) -> (276|276) - Passierbar
STADT GEFUNDEN - Weg hinzugefüegt!

BwInf Map 02 – Quax 5:

Die zweite Map von der BwInf Seite mit der fünften Quax Position. (Der Algorithmus für diese Map funktioniert noch nicht richtig)

Map:

Ausgabe:

Quo vadis, Quax?

Öffne .png Datei

Öffne Konsole

Quax 1 Quax 2 Quax 3 Quax 4
Quax 5 Quax 6 Quax 7

Starte Algorithmus

Drohnen Flüge:

- 1 Gemischt
- 2 Gemischt
- 3 Gemischt
- 4 Gemischt
- 5 Gemischt
- 6 Gemischt
- 7 Gemischt

Ergebnis für Quax 5

WEG GEFUNDEN: NEIN

LAUFZEIT: 292ms

DROHNEN FLÜGE: 37

Konsole:

```
Konsole
[31] Node: (1503|592) -> (1506|595) - HILFT NICHT WEITER
[36] Node: (1503|590) -> (1506|593) - Gemischt
[37] Abschluss Node: (1503|591) -> (1505|593) - Passierbar
Weg hinzugefuegt!
[31] Node: (1503|592) -> (1506|595) - Gemischt
[34] Abschluss Node: (1503|593) -> (1505|595) - Wasser
[35] Abschluss Node: (1503|592) -> (1505|594) - Wasser
[32] Abschluss Node: (1504|593) -> (1506|595) - Wasser
[33] Abschluss Node: (1504|592) -> (1506|594) - Passierbar
Weg hinzugefuegt!
[34] Abschluss Node: (1503|593) -> (1505|595) - Wasser
[35] Abschluss Node: (1503|592) -> (1505|594) - Wasser
[32] Abschluss Node: (1504|593) -> (1506|595) - Wasser
[33] Abschluss Node: (1504|592) -> (1506|594) - Passierbar
[31] Node: (1503|592) -> (1506|595) - HILFT NICHT WEITER
[23] Node: (1498|594) -> (1508|604) - Wasser
[24] Node: (1498|585) -> (1508|595) - Gemischt
[25] Node: (1503|590) -> (1508|595) - Gemischt
[31] Node: (1503|592) -> (1506|595) - Gemischt
[34] Abschluss Node: (1503|593) -> (1505|595) - Wasser
[35] Abschluss Node: (1503|592) -> (1505|594) - Wasser
[32] Abschluss Node: (1504|593) -> (1506|595) - Wasser
[33] Abschluss Node: (1504|592) -> (1506|594) - Passierbar
[31] Node: (1503|592) -> (1506|595) - HILFT NICHT WEITER
[36] Node: (1503|590) -> (1506|593) - Gemischt
[37] Abschluss Node: (1503|591) -> (1505|593) - Passierbar
Weg hinzugefuegt!
[31] Node: (1503|592) -> (1506|595) - Gemischt
[34] Abschluss Node: (1503|593) -> (1505|595) - Wasser
[35] Abschluss Node: (1503|592) -> (1505|594) - Wasser
[32] Abschluss Node: (1504|593) -> (1506|595) - Wasser
[33] Abschluss Node: (1504|592) -> (1506|594) - Passierbar
Weg hinzugefuegt!
[34] Abschluss Node: (1503|593) -> (1505|595) - Wasser
[35] Abschluss Node: (1503|592) -> (1505|594) - Wasser
[32] Abschluss Node: (1504|593) -> (1506|595) - Wasser
[33] Abschluss Node: (1504|592) -> (1506|594) - Passierbar
[31] Node: (1503|592) -> (1506|595) - HILFT NICHT WEITER
[25] Node: (1503|590) -> (1508|595) - Gemischt
[31] Node: (1503|592) -> (1506|595) - Gemischt
[34] Abschluss Node: (1503|593) -> (1505|595) - Wasser
[35] Abschluss Node: (1503|592) -> (1505|594) - Wasser
[32] Abschluss Node: (1504|593) -> (1506|595) - Wasser
[33] Abschluss Node: (1504|592) -> (1506|594) - Passierbar
[31] Node: (1503|592) -> (1506|595) - HILFT NICHT WEITER
[36] Node: (1503|590) -> (1506|593) - Gemischt
[37] Abschluss Node: (1503|591) -> (1505|593) - Passierbar
Weg hinzugefuegt!
[31] Node: (1503|592) -> (1506|595) - Gemischt
[34] Abschluss Node: (1503|593) -> (1505|595) - Wasser
[35] Abschluss Node: (1503|592) -> (1505|594) - Wasser
[32] Abschluss Node: (1504|593) -> (1506|595) - Wasser
[33] Abschluss Node: (1504|592) -> (1506|594) - Passierbar
Weg hinzugefuegt!
[34] Abschluss Node: (1503|593) -> (1505|595) - Wasser
[35] Abschluss Node: (1503|592) -> (1505|594) - Wasser
[32] Abschluss Node: (1504|593) -> (1506|595) - Wasser
[33] Abschluss Node: (1504|592) -> (1506|594) - Passierbar
[31] Node: (1503|592) -> (1506|595) - HILFT NICHT WEITER
```

Teilaufgabe (d)

Rückblickend war es wohl ein Fehler, den Algorithmus basierend auf nur einer einzigen Map zu konzipieren, da der Algorithmus nun zwar für die gewählte Test Map funktioniert, jedoch nicht für Maps mit spezielleren Strukturen, die die Test Map nicht beinhaltet hat. Leider hatte ich keine Zeit mehr den Algorithmus an weitere Spezialfälle anzupassen und so funktioniert er für die meisten Maps noch nicht.

Bisher sind mir folgende Verbesserungsmöglichkeiten aufgefallen, die ich jedoch weder getestet noch implementiert habe. Deshalb sind die meisten nur Vermutungen:

1. Eine Node kann mehr als 2 innere, passierbare Nodes brauchen.

Es kann auch zu Fällen kommen, bei mehr als 2 Nodes überprüft werden müssen.

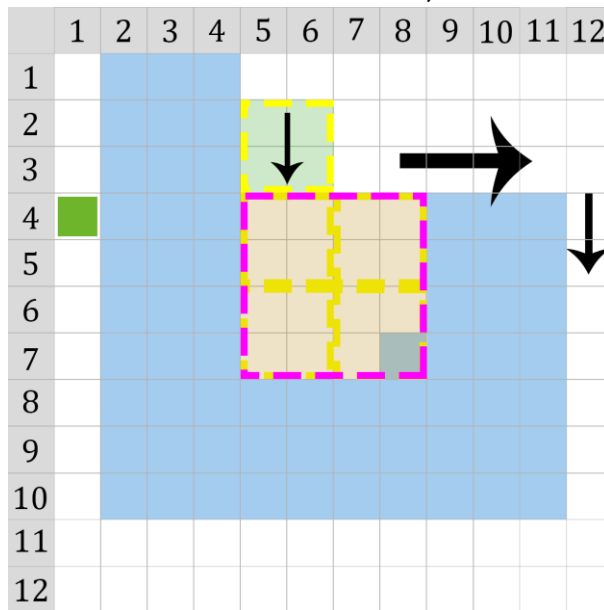


ABBILDUNG 17

Bei Abbildung 17 ist der aktuelle Weg das passierbare Quadrat (5 | 2), (6 | 3). Das aktuelle Quadrat, welches überprüft wird ist das Pinke. Die einzigst mögliche Richtung die der Pathfinding Algorithmus hier wählen sollte ist, wie die Pfeile es anzeigen, rechts um das Wasser herum. Das innere Quadrat (7 | 4), (8 | 5) wäre jedoch das letzte Quadrat, welches über den aktuellen Algorithmus überprüft werden würde. Zuvor würden sich allerdings die Quadrate (5 | 4), (6 | 5) und (5 | 6), (6 | 7) als passierbar herausstellen, da sie näher zum Ziel sind. Damit wären zwei passierbare Quadrate gefunden und die pinke Node wäre fertig. Jedoch würde der Weg in eine Sackgasse führen.

2. Definition: Quax kann durch 2 gemischte 20x20 Quadrate durch, auch wenn dazwischen ein 20x20 Quadrat Wasser liegt.

In einem Thread² auf www.einsteig-informatik.de wurde die Frage gestellt, ob Quax über ein 20x20 Wasser Quadrat drüber kann, wenn man dieses Quadrat auch in zwei 20x20 gemischten Quadraten darstellen kann.

Mein Programm macht das aktuelle Mal so mal so, abhängig davon welchen Ausschnitt überprüft wird.

² <https://www.einstieg-informatik.de/community/forums/topic/558/aufgabe-3-quax> (abgerufen 08.04.2018)

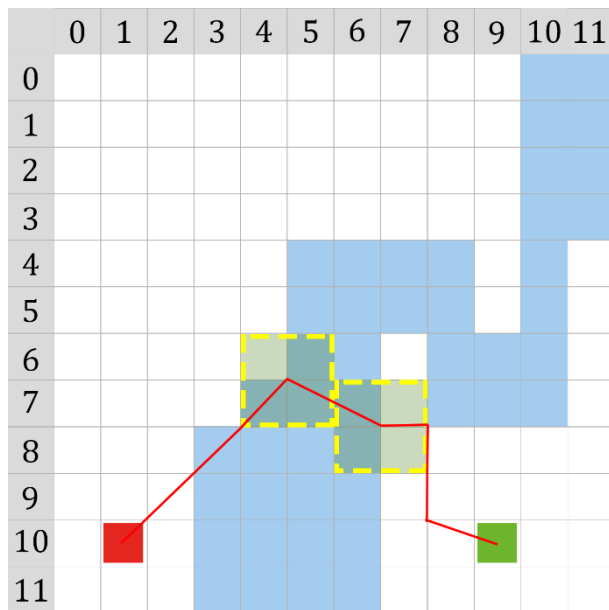


ABBILDUNG 18

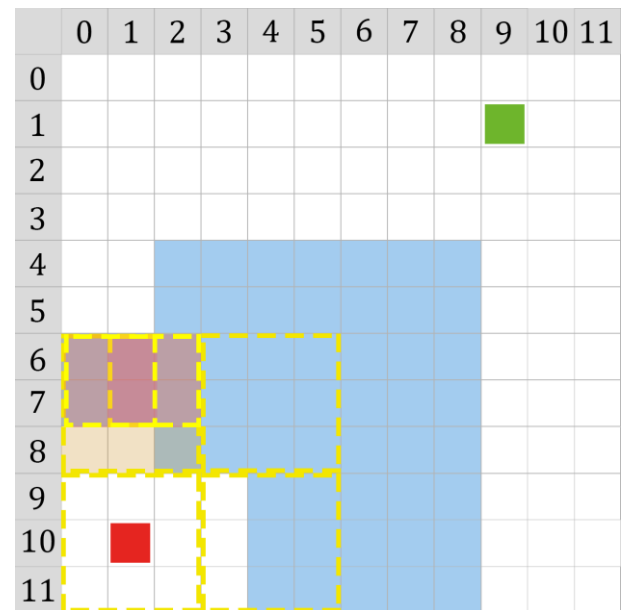


ABBILDUNG 19

In Abbildung 18 wird ein 20x20 Wasser Gebiet überquert und in Abbildung 19 wird das 20x20 Wasser gebiet als Wasser gekennzeichnet.