Computational Geometry and Virtual Reality - RunAR

Christian Schaf, Jannis Lindenberg, Vural Yilmaz

01. Januar 2019

Inhaltsverzeichnis

Abbildungsverzeichnis

Kapitel 1

Einleitung

Im Rahmen des Moduls “Computational Geometry and Virtual Reality” wird eine Augmented Reality App entwickelt. Das Ziel dieses Projekts ist es, Objekte in einer durch AR erweiterten Realität zu erkennen und in die Spielumgebung zu integrieren. Der Algorithmus-gesteuerte Spieler soll durch den A* Algorithmus möglichst effizient den kürzesten Weg zwischen Start und Ziel nutzen, dabei variabel auf die sich ständig ändernde Spielumgebung reagieren - wird ein neues Hindernis erkannt, muss der Weg des Computergesteuerten Spielers zum Ziel neu berechnet werden. Die Motivation hinter dieser Projektidee ist unter anderem im Rahmen dieses Projekts eine kleinen Einblick in die Entwicklung von Augmented Reality zu erhalten. Zusätzlich soll in Erfahrung gebracht werden, wie weit Augmented Reality Anwendungen mit der echten Umgebung interagieren können.

Vural - AR, Spiel - Motivation - Projektidee

Kapitel 2

Grundlagen

Christian

2.1 A* Algorithmus

Der A*-Algorithmus ist ein Wegfindungs-Algorithmus, der von Peter Hart, Nils Nilsson und Bertram Raphael entwickelt wurde. Sein Ziel ist es den schnellsten Weg in einem Grafen vom Startknoten zum Zielknoten zu finden.

2.1.1 Funktionsweise

Der Ausgangspunkt bildet ein zwei-dimensionales Array in dem es Felder gibt, die entweder Pfad und Hindernis darstellen. Es wird ein Start- sowie Zielfeld festgelegt. Beide sind dem Algorithmus während der Ausführung bekannt.Sobald das Spielfeld erstellt sowie ein Start- und Zielfeld gewählt wurde, sind die Mindestvoraussetzungen erfüllt. Ein Feld wird auch als Node bezeichnet und hat drei wichtige Attribute (siehe Abbildung 2.1). Eines der Attribute sind die G-Kosten, welche die Distanz zum Start-Node darstellen. Weiterhin gibt es die *H-Kosten*, welche die Distanz zum Ziel darstellen. Das dritte Attribut sind die *F-Kosten*. Diese werden berechnet indem man die *H-Kosten* mit den *H-Kosten* addiert (siehe Formel 2.1).

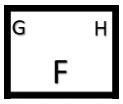


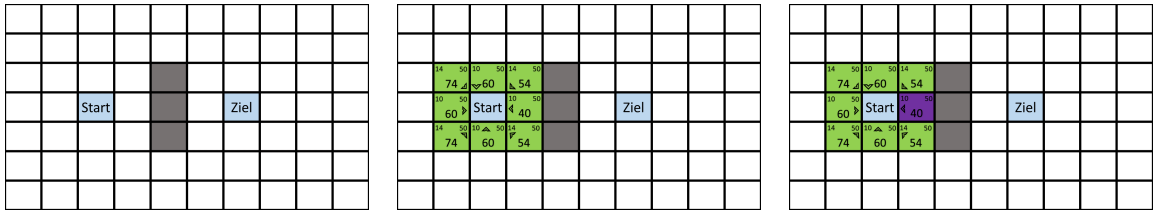
Abbildung 2.1: Beispiel Node

$$F_{cost} = G_{cost} + H_{cost} \tag{2.1}$$

Für die *H-Kosten* wird eine Heuristik verwendet. Hierbei wurde sich aus Simplizität für die *Manhatten-Methode* entschieden. Für die Kostenbrechnung allgemein musst man sich auf einen numerischen Wert für das horizontale bzw. vertikale und diagonale Bewegung zum nächsten Node. Als Wert für die horizontale bzw. vertikale Bewegung wird 10 festgelegt. Für die diagonale Bewegung wird 14 verwendet.Mit diesen Werten kann man nun die Kosten für die Nodes berechnen.

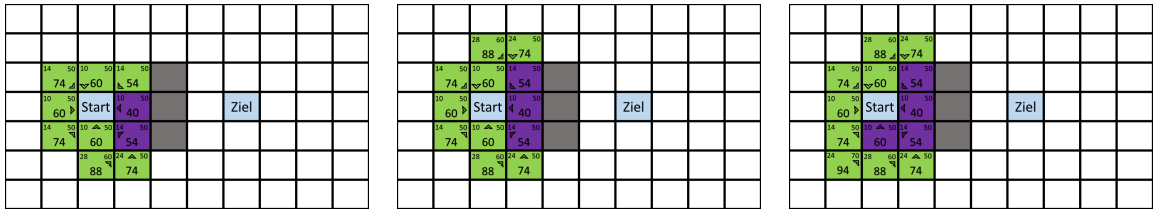
Warum?

In Schritt 1 von Abbildung 2.2 sind die hellblauen Felder als Start und Zielfeld markiert.



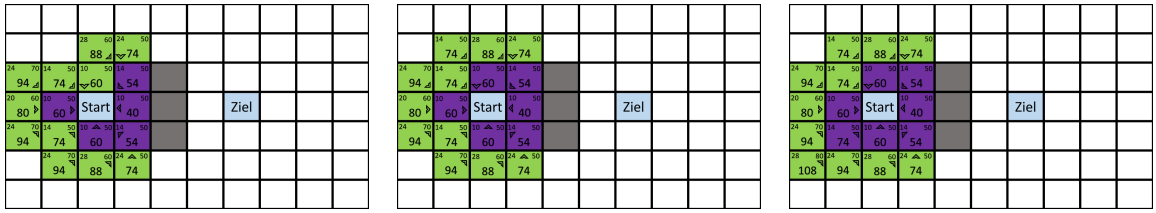
(a) Schritt 1 (b) Schritt 2 (c) Schritt 3

Abbildung 2.2: A* Ausführungsschritte 1-3



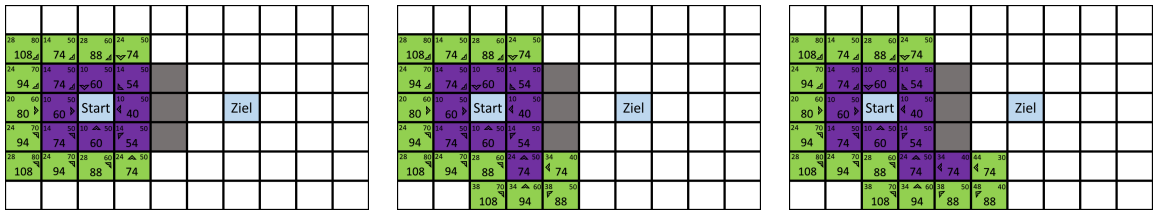
(a) Schritt 4 (b) Schritt 5 (c) Schritt 6

Abbildung 2.3: A* Ausführungsschritte 4-6



(a) Schritt 7 (b) Schritt 8 (c) Schritt 9

Abbildung 2.4: A* Ausführungsschritte 7-9



(a) Schritt 10 (b) Schritt 11 (c) Schritt 12

Abbildung 2.5: A* Ausführungsschritte 10-12

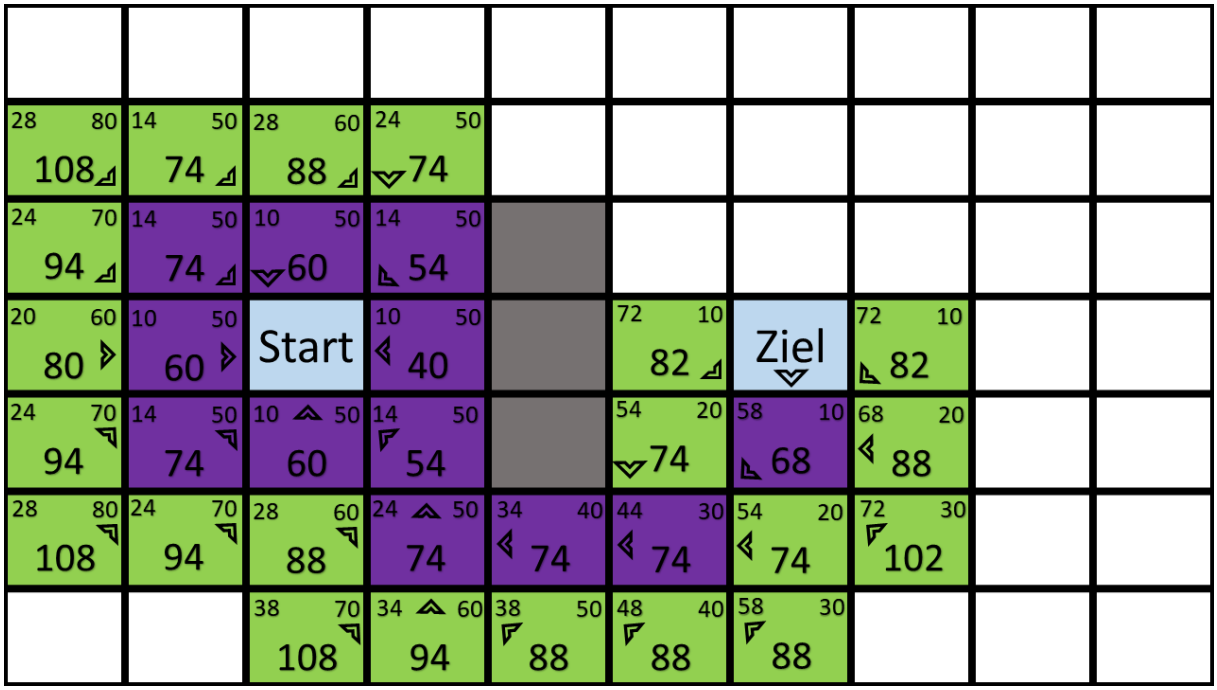


Abbildung 2.6: A* Schritt 13

2.2 Unity

Unity ist eine GameEngine die in diesem Projekt genutzt wird, um das Spiel zu entwickeln. Des Weiteren besteht Unity aus einem sehr umfangreichen Editor, mit dem ein Spieleentwickler eine große Auswahl an Tools zur Entwicklung erhält. Unity bietet sowohl die Möglichkeit in 2D, als auch in 3D zu entwickeln.

Jannis - anlehnen an Vurals aus den Grundlagen)

2.3 Vuforia

2.3.1 ImageTargets

2.3.2 Object Recognition

Scanner

Erkennung

Kapitel 3

Recherche

3.1 A* Algorithmus

Das Thema Wegfindung spielt in diesem Projekt eine wichtige Rolle. Der Computergegner soll einen Wegfindungs-Algorithmus benutzen, um den kürzesten Weg zum Ziel finden. Dabei soll die Wegfindung schnell und zuverlässig sein, auch wenn dynamisch ein oder mehrere Hindernisse auftauchen.

Zu den bekanntesten Algorithmen der Wegfindung gehören Dijkstra und A*. -hier warum A*-

3.2 Imagetargets

Jannis

3.3 Objekterkennung

Da die Objekterkennung einen großen Anteil des RunnAr Projekts ausmacht, wurde hierzu viel Recherche betrieben. Da die Objekte vom Benutzer in Echtzeit platziert und manipuliert werden können, wurde besonderes Augenmerk auf die Performanz der Implementierungsmöglichkeiten gerichtet.

3.3.1 Tensorflow

Tensorflow ist ein von Google entwickeltes Framework. Es wird oftmals in Programmen für maschinelles Lernen genutzt. Implementiert ist es in Python und C++. Da Unity von Haus aus mit C Sharp Skripten arbeitet und keine alternative Sprache genutzt werden kann, muss für Unity das TensorFlowSharp Plugin genutzt werden. Dies stellt Wrapper zur Verfügung die den nativen C++ Code in C Sharp Code umwandeln. Auf Github finden sich einige Beispielprojekte, die den Fokus auf Objekterkennung setzen. Dies ermöglicht einen schnellen Einblick auf die Funktionalitäten von Tensorflow in Kombination mit Unity. Tensorflow ermöglicht es zwar durch

trainierte Agenten eine große Variante an Objekten zu erkennen und zu klassifizieren, jedoch braucht dies seine Zeit. Die Kamera muss sich recht nah am Objekt befinden und Tensorflow verliert immer wieder den Fokus auf das Objekt. Zusätzlich ist die bounding Box, die sich um das Objekt dargestellt wird, recht grob. Bei diesem Ansatz fehlte für das Projekt also die Performanz und die Genauigkeit. Zusätzlich passte die Nähe die die Kamera zu dem Objekt haben musste nicht zu unserem Spielkonzept.

3.3.2 OpenCv

OpenCV ist eine freie Programmbibliothek zur Bildverarbeitung. Der Ansatz bei OpenCv wäre, dass Umrisse von auf dem Tisch platzierten Gegenständen erkennt werden sollten. Dies müsste jedoch auf einem hellen Untergrund geschehen, an dem sich die Objekte deutlich abheben. Der Vorteil gegenüber Tensorflow wäre hierbei, dass man keine trainierte Agenten bräuchte, die die Objekte zusätzlich klassifizieren würden. OpenCV hat jedoch den Nachteil, dass der Einsatz mit Unity kostenpflichtig ist. Die Kombination mit Unity ohne Kosten gestaltet sich als schwierig. Da Unity nur mit C Sharp Skripten arbeiten kann, müssten Wrapper erstellt werden, die die Kommunikation zu Unity ermöglichen würden. Fragwürdig wäre jedoch wie performant die Erkennung der Objekte mit OpenCv gewesen wäre.Viel Zeit und Aufwand hätten in die Entwicklung eines C++ Projekts mit OpenCv fließen müssen, bei der am Ende immer noch die Frage im Raum gewesen wäre, ob dies überhaupt flüssig in Unity laufen würde. Diesen Ansatz haben wir nach mehreren Tagen erfolgloser Recherche und Implementierungsversuchen für ineffizient eingestuft.

3.3.3 Vuforia Object Recognition

Vuforia verfügt nativ über Object Recognition. Hierbei müssen die Objekte, welche erkannt werden sollen, vorher durch eine kostenlose App für Android Geräte eingescannt werden. Hierfür kann man sich auf der Webseite von Vuforia eine Schablone ausdrucken, auf der man die zu scannenden Objekte platziert. Danach nutzt man die App und scannt die Objekte mit der Kamera auf dem mobilen Endgerät ein. Es wird virtuell ein Gitter um das platzierte Objekt dargestellt und die Bereiche die fertig gescannt wurden, werden grün markiert. Dies wiederholt man solange bis das Gitter komplett grün gefärbt ist. Beim einscannen sollte man keine Objekte nehmen die zu klein sind, da diese recht lange brauchen um eingescannt zu werden. Zusätzlich ist die Erkennung kleiner Objekte nicht sehr effektiv, da Vuforia bei kleinen Objekte nicht ausreichend Vergleichspunkte hat um die Gegenstände zu erkennen. Die Performanz der eingescannten Objekte ist sehr gut, sodass die Einschränkung, dass nur eingescannte Objekte erkennt werden können, in Kauf genommen wurde. Auch die Erkennung auf weitere Entfernung ist gegeben. Nach dem Einscannen, können diese in eine Datenbank importiert werden und problemlos in Unity eingebunden werden. Mehr hierzu im Kapitel Umsetzung.

3.4 Spielkonzept

Kapitel 4

Konzept

Nachdem sich die Idee des Projektes verfestigt hat, muss genau ausdefiniert werden, welche Funktionen die Anwendung am Tag der Abgabe erfuellen soll. Nicht alle Visionen lassen sich in dem vorgegebenen Zeitraum umsetzten, sodass eine Liste aus Funktionsanforderungen gefertigt wird. Unumgaenglich war jedoch die Wahl fuer einen Algorithmus zur Merkmalser- kennung, der das Herz der Anwendung darstellt. Somit ist bei dessen Wahl besondere Vorsicht geboten, weswegen hierfuer eine Gegenueberstellung von moeglichen Algorithmen ausgearbeitet wurde. Als Bibliothek wurde sich fuer OpenCV entschieden, welche mit der Spra- che C++ genutzt wird. Der Grund fuer diese Wahl sind die deutlich weitrei- chenderen Funde bei Recherchen, sie uebersteigen die von Java oder Python deutlich.

Jannis überarbeiten

4.1 Definition von Funktionsanforderungen

Bei der Abgabe des Projektes sollen folgende Anforderungen Teil des Funktionsumfang sein.

Donnerstag zusammenmachen

Christian - evtl als Tabelle

4.2 Spielkonzept

Auf der Skizze aus Abbildung 1 ist ein Musterspielfeld abgebildet. Die Spielfigur (gruener Pull- over), welche sich zu Beginn des Spiels am unteren Rand des Spielfelds befindet, muss den Hindernissen ausweichen. Hindernisse koennen zum Einen aus Image Targets bestehen, auf die in der Spielumgebung 3D Objekte gemappt werden, oder zum Anderen aus realen Objekten, die Im Spielfluss erkannt werden und so ebenfalls ein 3-dimensionales Hindernis fuer den Spieler darstellen. Der Start und das Ziel sind ebenfalls durch Muster gekennzeichnet.

Jannis

4.2.1 Entwicklung des Spielkonzepts

4.2.2 Finales Spielkonzept

Kapitel 5

Umsetzung

Jannis

5.1 Implementierung des A* Algorithmus

Christian

5.2 ImageTargets

Design der Images und Erkennung der Images — Jannis

5.3 Erkennung von Objekten

Jannis + Vural

5.4 UI

Jannis

Kapitel 6

Zusammenfassung und Ausblick

6.1 Zusammenfassung

Christian

6.2 Ausblick

Christian

6.3 Fazit

Christian

Kapitel 7

Anhang

7.1 A* Beispiel