PATHFINDING & COLLISIEVERMIJDING TUSSEN BEWEGENDE OBJECTEN IN $\mathbb{Z}^2$

Klas ICT SE 2B
Projectgroep 15:
Rob Klein Ikink
Robin Sinnema
Finn Koorn
Joshua Terra

1e versie: 27/10/2019
Laatste revisie: 02/11/2019

INHOUD

Bericht aan de nakijker van dit document:
Op Blackboard stonden verschillende tegenstrijdige beweringen over wat dit onderzoek in diende te houden. Aangezien het voor ons onduidelijk was welke opdrachtsbeschrijving de juiste was, hebben wij de beschrijving gevolgd op de rubricspagina en hebben wij het onderzoek gemaakt over één onderdeel van het project, (pathfinding) en niet over het project in het algeheel, zoals in een van de andere beschrijvingen werd gevraagd.
Wij hopen hiermee de juiste keuze te hebben gemaakt. Zo niet, dan horen wij graag van u terug.

1. INLEIDING

Voor het graphicsproject is van ons gevraagd om een simulatie van een gerobotiseerd warenhuis te maken. In deze simulatie bewegen meerdere robots simultaan over een discreet verdeelbaar vlak, waarbij ieder punt of vrij begaanbaar of geblokkeerd is.
Doordat de robots bewegen kan een punt dat eerder vrij begaanbaar was tijdelijk geblokkeerd raken. Ons doel is om een robot van punt *A* naar punt *B* te verplaatsen, zonder dat deze robot een punt op het vlak doorkruist dat geblokkeerd is.

1.1 Hoofd- & deelvragen

Voor dit onderzoek hebben wij de volgende vragen opgesteld:

Hoofdvraag:
Hoe kan een robot op een veranderend 2-dimensionaal discreet verdeelbaar vlak van een punt *A* naar een punt *B* bewegen, zonder een punt te doorkruisen dat geblokkeerd is?

Deelvraag 1:
Hoe kan een set van objecten in $\mathbb{R}^3$ omgezet worden naar een discreet 2-dimensionaal booleanveld dat de bezetting van deze 3-dimensionale ruimte weergeeft?

Deelvraag 2:
Hoe kan in een discreet 2-dimensionaal booleanveld een pad tussen twee punten geconstrueerd worden, zodat de waarde van het veld op het pad constant is?

Deelvraag 3:
Hoe kunnen collisies worden voorspeld en vermeden wanneer het bovengenoemde veld verandert als een functie van tijd?

1.2 Methodiek

Voor dit onderzoek hebben wij verschillende onderzoeksmethoden gebruikt: initieel hebben wij vooral literatuuronderzoek gedaan over verschillende pathfindingmethoden. Hoewel dit voor het kiezen van een pathfindingalgoritme erg successvol was, is het ons hiermee niet gelukt om een collisiedetectie-algoritme te kiezen.
Via een brainstorm hebben wij twee mogelijke algoritmes bedacht die wij vervolgens via het maken van enkele prototypes getest hebben.
Een zelfde werkwijze hebben wij toegepast om de eerste deelvraag te beantwoorden.

2. RESULTATEN

2.1 Mapping van $\mathbb{R}^3$ naar $\mathbb{Z}^2$

De simulatie bestaat uit een set van objecten, die zich met discrete tijdsstappen door $\mathbb{R}^3$ bewegen. Een object heeft de vorm van een rechthoek.[1] Gegeven de positievector $\vec{v}$ en de formaat-vector $\vec{s}$ van de rechthoek, kunnen de bezette punten van het object in $\mathbb{Z}^2$ berekend worden:

$$P(\vec{v},\vec{s})=\{(x,y)|x,y\in\mathbb{Z},\vec{v}_x\leq x<(\vec{v}_x+\vec{s}_x),\vec{v}_z\leq y<(\vec{v}_z+\vec{s}_z)\}$$

2.2 Constructie van een pad in een onveranderend veld

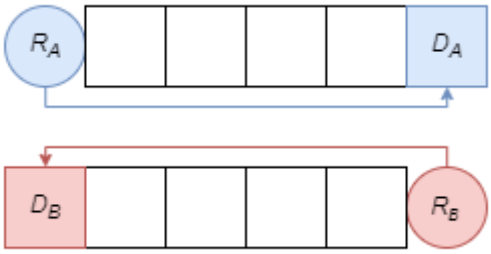
Aangezien de afstand tussen twee aangrenzende punten in $\mathbb{Z}^2$ altijd één is, kan het kortste pad tussen twee punten met het *Breadth First Search* algoritme in plaats van een algoritme zoals *Dijkstra's Shortest Path* opgelost worden: (*Kulovesi, 2010; Morin, 2014*)

```java
public List<Direction> calculatePathBFS(boolean[][] space, Vec2i src, Vec2i dest) {
    List<Vec2i> discovered = new ArrayList<>();
    Map<Vec2i, Direction> parents = new HashMap<>();
    Queue<Vec2i> queue = new LinkedList<>();

    discovered.add(src);
    queue.add(src);

    while (queue.size() > 0) {
        Vec2i pos = queue.remove();

        if (pos.equals(dest)) {
            // Retrace path
            List<Direction> path = new ArrayList<>();

            Vec2i next = pos;
            while (!next.equals(src)) {
                Direction d = parents.get(next);

                path.add(d.invert());
                next = new Vec2i(next.x + d.movement.x, next.y + d.movement.y);
            }

            Collections.reverse(path);
            return path;
        }

        for (Direction d : Direction.values()) {
            Vec2i next = new Vec2i(pos.x + d.movement.x, pos.y + d.movement.y);

            if (next.x < 0 || next.x >= space.length)           continue;       // Check X-coord inside bounds.
            if (next.y < 0 || next.y >= space[next.x].length)   continue;       // Check Y-coord inside bounds.
            if (!space[next.x][next.y] && !next.equals(dest))   continue;       // Check space not occupied.
            if (discovered.contains(next))                      continue;       // Check node already discovered.

            discovered.add(next);
            parents.put(next, d.invert());

            queue.add(next);
        }
    }

    return null;    // No path exists.
}
```

1 Om het probleem te versimplen wordt in dit onderzoek ervan uitgegaan dat de rechthoeken gealigneerd met het assenstelsel van de ruimte zijn. Als dit niet het geval is kan in plaats van de rechthoek zelf de *Axis Aligned Bounding Box* van de rechthoek gebruikt worden.

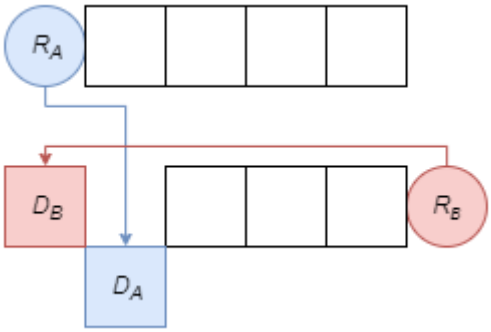
2.3 Preventie van collisies

Doordat meerdere robots zich simultaan over het vlak bewegen kunnen situaties ontstaan waarbij twee robots tegen elkaar aan zouden rijden.
Het is niet altijd mogelijk voor twee robots om om elkaar heen te bewegen, (bijvoorbeeld in de situatie in afbeelding 1) dus is het noodzakelijk om te allen tijde te voorkomen dat twee robots zich tegelijkertijd naar hetzelfde punt proberen te bewegen.



Afb. 1: Situatie waarin collisie plaatsvindt

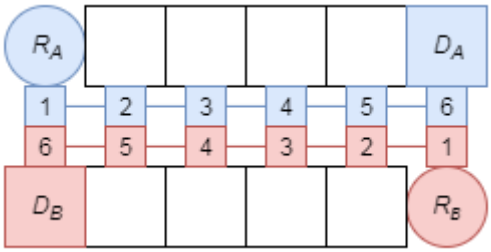
Een manier om dit te bewerkstelligen is om voor iedere robot het complete toekomstige pad bij te houden, en niet toe te staan dat een andere robot zich op het toekomstige pad van een andere robot bevindt.
Dit is echter onnodig inefficiënt: in een situatie zoals in afbeelding 2 kruist robot *A* het pad van robot *B*, maar vindt er geen collisie plaats, omdat robot *B* pas op een later tijdstip dat deel van het pad passeert.



Afb. 2: Situatie waarin geen collisie plaatsvindt

Een betere oplossing is om te kijken naar de tijdstippen waarop de paden zich voor het eerst kruisen: als er een collisie plaatsvindt in de toekomst, kan een van de robots voor dit punt wachten tot het voortzetten van zijn route geen collisie meer veroorzaakt.

Om zo'n collisie te vinden is het mogelijk om te kijken naar de tijdsstippen waarop iedere robot op welk punt is: als er een collisie plaatsvindt hebben twee robots op hetzelfde punt dezelfde tijdswaarde, of is de positie van robot *A* op tijdspunt *n* gelijk aan die van robot *B* op tijdspunt *n - 1*, en vice versa. (afbeelding 3)



Afb. 3: Locatie per tijdstip

Een andere oplossing is om iedere tijdsstap opnieuw het kortste pad te berekenen, waarbij de punten rondom alle andere robots als bezet worden gemarkeerd. Collisies worden hierbij alleen als relevant beschouwd als ze in de nabije toekomst plaatsvinden.
Een probleem met deze methode is dat het pad iedere tijdsstap opnieuw berekend moet worden, wat deze oplossing niet geschikt maakt voor grote warenhuizen met veel robots.
Een ander probleem is dat van deadlock: twee robots kunnen vast komen te zitten in een oneindig durende dans om elkaar heen, waarbij de een steeds het kortste pad van de ander blokkeert, waardoor deze zijn route moet veranderen.
Dit probleem is echter relatief makkelijk op te lossen door de gemiddelde afstand van de robot tot zijn bestemming bij te houden: als deze gedurende een langere tijd een vast minima heeft dat groter is dan nul, zit deze robot in een deadlock, en kan zijn bewegen voor korte tijd gepauzeerd worden om deze deadlock te verhelpen.

3. CONCLUSIES

Deelvraag 1: *Hoe kan een set van objecten in $\mathbb{R}^3$ omgezet worden naar een discreet 2-dimensionaal booleanveld dat de bezetting van deze 3-dimensionale ruimte weergeeft?*

Gegeven de positievector $\vec{v}$ en de formaatvector $\vec{s}$ van het object, is de set van punten die het object op het veld inneemt gelijk aan: $P(\vec{v},\vec{s})=\{(x,y)|x,y\in\mathbb{Z},\vec{v}_x\leq x<(\vec{v}_x+\vec{s}_x),\vec{v}_z\leq y<(\vec{v}_z+\vec{s}_z)\}$

Deelvraag 2: *Hoe kan in een discreet 2-dimensionaal booleanveld een pad tussen twee punten geconstrueerd worden, zodat de waarde van het veld op het pad constant is?*

Via een *Breadth First Search* kan het kortste pad tussen twee punten in het veld gevonden worden.

Deelvraag 3: *Hoe kunnen collisies worden voorspeld en vermeden wanneer het bovengenoemde veld verandert als een functie van tijd?*

Er zijn meerdere manieren om collisies te voorspellen en te vermijden: robots kunnen hun toekomstige pad bijhouden en het betreden van andermans pad uitstellen als dit tot een collisie zou leiden, of robots kunnen simpelweg hun pad steeds opnieuw berekenen, waarbij de posities van andere robots als geblokkeerd worden gemarkeerd.
Hierbij is het belangrijk om deadlock te detecteren en te verhelpen. Dit kan gedaan worden door de robot enige tijd stil te zetten als de minimale afstand tot de bestemming van de robot over langere tijd niet afneemt.
In onze simulatie hebben wij ervoor gekozen om van de tweede methode gebruik te maken.

Hoofdvraag: *Hoe kan een robot op een veranderend 2-dimensionaal discreet verdeelbaar vlak van een punt A* naar een punt *B* bewegen, zonder een punt te doorkruisen dat geblokkeerd is?

Via een *Breadth First Search* kan het kortste pad tussen twee punten in het veld gevonden worden. Door in deze search de posities van alle andere robots als bezet op te nemen, en de search iedere tijdsstap opnieuw uit te voeren, kunnen ook collisies met bewegende objecten voorkomen worden.

4. BIJLAGEN

4.1 Bronvermelding

Kulovesi, K. (2010, September 29). Why use Dijkstra’s Algorithm if Breadth First Search (BFS) can do the same thing faster? [Forum post]. Retrieved October 27, 2019, from https://stackoverflow.com/questions/3818079/why-use-dijkstras-algorithm-if-breadth-first-search-bfs-can-do-the-same-thing

Morin, P. (2014). 12.3.1 Breadth-First Search [PDF]. In P. Morin (Ed.), *Open Data Structures (in pseudocode)* (0.1Gβ, pp. 248–250). Retrieved from http://opendatastructures.org/ods-python-screen.pdf