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: 27/10/2019

INHOUD

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 kan een conflict verholpen worden tussen twee of meerdere robots, waarbij er voor een of meerdere van deze robots geen pad naar zijn bestemming bestaat, doordat een andere robot een deel van dit pad blokkeert?

2. ONDERZOEK

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

De simulatie bestaat uit een aantal 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

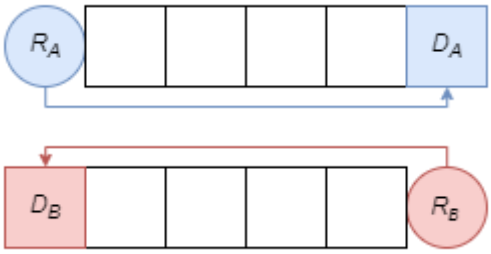
Aangezien de afstand tussen twee aangrenzende punten in $\mathbb{Z}^2$ altijd één is, kan het kortste pad tussen twee punten met het *Breadth First Search* algoritme in plaats van een algoritme zoals *Dijkstra's Shortest Path* opgelost worden: (*Kulovesi, 2010; Morin, 2014*)

```
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 versimpelen 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.

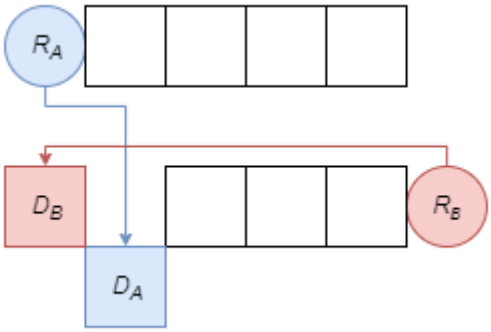
2.3 Preventie van padobstructies

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.



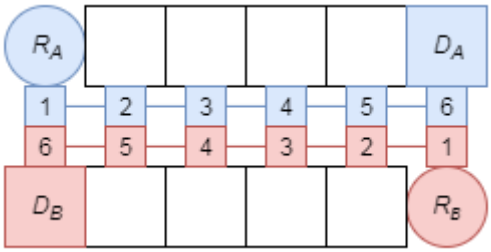
Afb. 1: Situatie waarin collisie plaatsvindt

De eenvoudigste manier om dit te bewerkstelligen is om voor iedere robot het complete toekomstige pad bij te houden, (afbeelding 3) 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 kruisen.
Om te kijken of er een collisie met een robot *A* op het pad van een andere robot *B* zal plaatsvinden zijn er twee situaties die overwogen moeten worden:[2]



Afb. 3: Locatie per tijdstip

1. A en B bewegen zich tegelijkertijd naar hetzelfde punt
Dit geval is te detecteren doordat de paden van *A* en *B* op hetzelfde punt dezelfde tijdswaarde hebben:

$$\exists X \in \mathbb{A} : T(A, X) = T(B, X)$$

2. A beweegt zich naar een punt waar B zich net uit beweegt, en/of vice versa
Als dit het geval is, is er een tijdsverschil van precies één tussen de tijdsstippen waarop de twee robots zich naar het punt verplaatsen:

$$\exists X \in \mathbb{A} : |T(A, X) - T(B, X)| = 1$$

hier is $\mathbb{A}$ de set van alle punten op het pad van *A* en *T(R, X)* het tijdstip waarop een robot *R* op een punt *X* is.

2 Hier wordt ervan uitgegaan dat voor de snelheid *s* waarmee een robot zich tussen twee punten beweegt geldt dat $1 / s \in \mathbb{Z}$.

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 kan een conflict verholpen worden tussen twee of meerdere robots, waarbij er voor een of meerdere van deze robots geen pad naar zijn bestemming bestaat, doordat een andere robot een deel van dit pad blokkeert?*

Als iedere robot bijhoudt op welke punten hij zich op welk tijdstip zal begeven, vinden collisies plaats tussen twee robots *A* en *B* wanneer $\exists X\in\mathbb{A}:T(A,X)=T(B,X)$ of $\exists X\in\mathbb{A}:|T(A,X)-T(B,X)|=1$, met $\mathbb{A}$ de set van alle punten op het pad van *A* en *T(R, X)* het tijdstip waarop een robot *R* op een punt *X* is.
Als dit het geval is, kan robot *A* wachten voordat hij een punt betreedt dat in het toekomstige pad van robot *B* ligt, totdat bovenstaande condities niet meer waar zijn.

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. Als de robot vervolgens op zijn pad het toekomstige pad van een andere robot doorkruist, kan gekeken worden of dit tot een collisie zal leiden. Wanneer dit het geval is kan de robot wachten tot dat niet langer zo is, voordat hij het pad van de andere robot doorkruist.

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