Universiteit Leiden
The Netherlands

Opleiding Informatica

Bomb-cover

Een verzamelingenoverdekkings-probleem in de Bombermanwereld

Rintse van de Vlasakker

Begeleiders:
Rudy van Vliet & Hendrik Jan Hoogeboom

BACHELOR THESIS

Leiden Institute of Advanced Computer Science (LIACS)
www.liacs.leidenuniv.nl

23/08/2019

Abstract

In deze scriptie wordt een verzamelingenoverdekkings-probleem in de Bombermanwereld onderzocht. Het betreft een beslissingsprobleem over het legen van een Bombermanbord door middel van het afsteken van explosies met onbeperkte radius. Allereerst bewijzen we dat het probleem in $\mathcal{NP}$ zit, en dat het $\mathcal{NP}$-moeilijk is, waaruit volgt dat het $\mathcal{NP}$-volledig is. Vervolgens proberen we hetzelfde bewijzen voor een aangepaste versie van het beslissingsprobleem, waar de explosieradius beperkt is. Daarna beschrijven we een gretig algoritme voor het vinden van een oplossing voor het bijbehorende optimalisatieprobleem.

Inhoudsopgave

1 Introductie

Bomberman is een videospelletje dat werd uitgebracht in 1983 voor de MSX thuiscomputer. Er volgden talloze versies na het succes van de eerste. In de vroege jaren tweeduizend, toen Flash spellen praktisch elk huishouden met een PC binnenslopen, bleek Bomberman ook ideaal voor dat platform. Het spelletje bereikte hier zijn hoogtepunt in populariteit, en is nu niet meer weg te denken uit de videospellen-wereld.

In deze scriptie bespreken we twee beslissingsproblemen in de Bombermanwereld. Deze beide problemen gaan om het vernietigen van alle kratten op een Bombermanbord. In Hoofdstuk 2 bewijzen we dat een versie van het probleem waarbij de explosies een onbeperkte radius hebben, genaamd Bomb-cover, $\mathcal{NP}$-volledig is. We doen dit door achtereenvolgens te bewijzen dat dit probleem in $\mathcal{NP}$ zit (Lemma 1.1), en dat het $\mathcal{NP}$-moeilijk is, met behulp van een reductie vanaf 3-SAT (Lemma 1.2). Zie ook [2]. Daarna proberen we in Hoofdstuk 3 hetzelfde te doen met een versie van het probleem waarbij explosies een beperkte radius hebben (genaamd Bomb-r-cover). We bewijzen dat dit probleem in $\mathcal{NP}$ zit (Lemma 2.1). Daarna werken we een vermoeden uit dat stelt dat Bomb-r-cover $\mathcal{NP}$-moeilijk is (Vermoeden 2). In Hoofdstuk 4 doen we een stapje terug en bespreken we een reductie naar een gerelateerd beslissingsprobleem, genaamd Box-cover. Deze reductie inspireerde ons tot onze reductie in Hoofdstuk 2 en ons idee voor een reductie in Hoofdstuk 3. Als laatste analyseren we in Hoofdstuk 5 het optimalisatieprobleem behorend bij Bomb-cover en ontwikkelen we een gretige strategie voor het brute-force oplossen van dit probleem.

Het legen van Bombermanborden is al eerder onderzocht, maar vooral in de context van kunstmatige intelligentie en machine learning [4], [3]. In deze scriptie bespreken we Bomberman met de focus op computationele complexiteitstheorie.

Hier volgt een technische beschrijving van het de bombermanwereldt.

De Bombermanwereld

Bomberman is een spel dat bestaat uit een rechthoekig bord van $m * n$ vakken en een speler. Er zijn drie verschillende soorten vakken:

1. **Leeg vak:** Over dit vak kan gelopen worden. Explosies gaan tevens zonder hinder over lege vakjes heen.

2. **Muurvak:** Over dit vak kan niet gelopen worden. Explosies stoppen wanneer ze tegen een muur aankomen en laten daarbij de muur onaangetast.

3. **Kratvak:** Over dit vak kan niet gelopen worden. Explosies vernietigen kratten wanner ze geraakt worden. Het geraakte kratvak veranderd dan in een leeg vak.

We zullen gebruik maken van sprites [5] om borden die we bespreken weer te geven. De groene sprites zijn lege vakken, de grijze sprites zijn muurvakken en de bruine sprites zijn kratvakken. Zie Figuur 1. We gaan er in deze scriptie vanuit dat de randvakken van het bord muurvakken zijn. Dit is niet van belang voor de geldigheid van de resultaten, maar maakt een eenduidige beschrijving eenvoudiger.

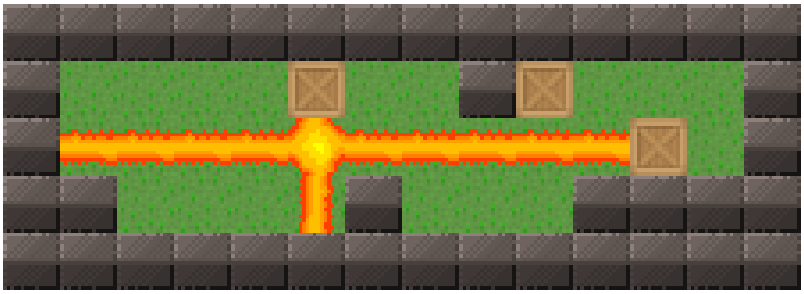
Figuur 1: Een voorbeeld van een Bombermanbord met speler.

De speler begint op een leeg vak en kan bewegen naar een aanliggend leeg vak. Diagonale beweging is niet mogelijk. De speler kan op zijn locatie dynamiet afsteken. Het dynamiet creëert een kruisvormige explosie met armen van, in principe, onbeperkte lengte. De armen van de explosie stoppen wanneer ze een krat of muur raken. De speler is onaangetast na een explosie. Zie Figuur 2.

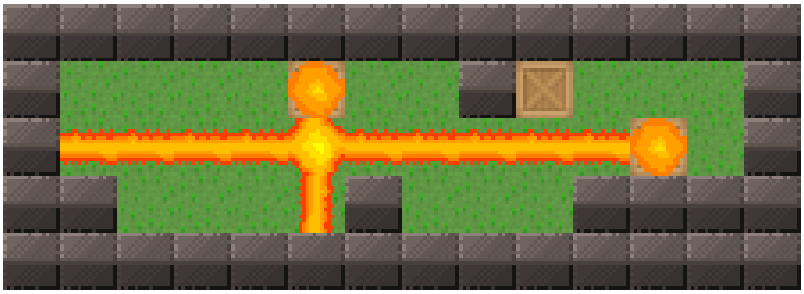
Er zijn veel videospellen die zich in deze wereld afspelen. Een populaire variant gaat als volgt: Twee spelers starten op posities op het bord zodat ze niet direct naar elkaar kunnen lopen. De spelers blazen kratten op om zo een pad naar de andere speler te openen. De kratten bevatten powerups die op de plek van de krat terecht komen nadat deze vernietigd wordt. Deze powerups kunnen vervolgens opgepakt worden en gebruikt worden om het einddoel te bereiken: de andere speler opblazen. Een aantal van deze powerups zijn beschreven in hoofdstuk 6.

Wij zullen niet een van deze videospellen bekijken, maar een losstaand probleem in de Bombermanwereld.

(a) Voor de explosie.



(b) De armen reiken tot een niet-leeg vak.



(c) De kratten aan de einden van de armen worden vernietigd.

(d) De speler staat nog op dezelfde positie.

Figuur 2: Het effect van een explosie.

2 Bomb-cover

We definiëren twee termen die ons helpen bij de definitie van het onderstaande probleem:

Definitie 1 *Een vakje is direct bereikbaar als er een pad bestaat vanaf de positie van de speler naar het vakje dat over enkel lege vakken gaat.*

Definitie 2 *Een vakje is bereikbaar als er een pad bestaat vanaf de positie van de speler naar het vakje dat over enkel kratten en lege vakken gaat.*

We definiëren in de Bombermanwereld het beslissingsprobleem **Bomb-cover**:

Probleem 1 *Bomb-cover: Gegeven een Bombermanbord, een beginpositie voor de speler, en een getal $k \in \mathbb{N}$, bestaat er een reeks van k explosies, elke explosie direct bereikbaar vanaf de vorige, die het bord leegt van bereikbare kratten?*

Aangezien de speler op de plaats moet staan van de explosie, moet elke explosie direct bereikbaar zijn vanaf de vorige. Tevens moet de eerste explosie direct bereikbaar zijn vanaf de beginpositie van de speler.

We coderen het Bombermanbord als een array van alle vakken. Het is mogelijk dit efficiënter te doen, zoals wordt besproken aan het einde van Hoofdstuk 6.

Stelling 1 *Bomb-cover is $\mathcal{NP}$-volledig.*

Bewijs. Om te bewijzen dat een beslissingsprobleem $\mathcal{NP}$-volledig is, moeten we zowel aantonen dat het beslissingsprobleem in $\mathcal{NP}$ zit, als aantonen dat het $\mathcal{NP}$-moeilijk is.

Lemma 1.1 *Bomb-cover zit in $\mathcal{NP}$*

Bewijs. We bewijzen dat Bomb-cover $\in \mathcal{NP}$. We doen dit door een polynomiaal begrensd, niet-deterministisch algoritme te beschrijven dat Bomb-cover oplost:

1. Genereer op niet-deterministische wijze een geordende rij coördinaatparen $(x_1, y_1), (x_2, y_2), \ldots, (x_k, y_k)$.
2. Voor $i = 1, 2, \ldots, k$
   - 2.1. Controleer of het vakje op coördinaat (x_i, y_i) direct bereikbaar is vanaf de huidige positie.
   - 2.2. Loop naar het vakje op coördinaat (x_i, y_i).
   - 2.3. Steek dynamiet af (en verwijder de kratten geraakt door de explosie).

3. Controleer of het bord geen bereikbare kratten meer bevat.

Als stap 2.1 of 3 onwaar oplevert, wordt de rij coördinaatparen verworpen. Als zowel stap 3 als elke uitvoering van stap 2.1 waar opleveren, is er een oplossing gevonden en stopt het algoritme.

Dit algoritme voert twee acties uit. Eerst moet het algoritme controleren of een coördinaat direct bereikbaar is. Dit kan gedaan worden met een breadth first search. Dit betekent dat deze stap polynomiaal is in de grootte van het bord, en dus in de grootte van de invoer. Daarnaast moet het effect van een explosie op het bord uitgeoefend worden: er moet bepaald worden welke kratten (indien aanwezig) vernietigd worden door een explosie op een bepaalde positie. Dit kan door de armen van de explosie af te lopen tot een krat- of muurvak gevonden wordt: $O(m+n)$. Deze stap is dus ook polynomiaal in de grootte van de invoer.
Dit algoritme genereert op niet-deterministische wijze kandidaatoplossingen van Bomb-cover en verifieert in polynomiale tijd of de kandidaatoplossing daadwerkelijk een oplossing is. Bomb-cover zit derhalve in $\mathcal{NP}$. Dit besluit het bewijs van Lemma 1.1.

Lemma 1.2 *Bomb-cover is $\mathcal{NP}$-moeilijk.*

Bewijs. We bewijzen dat Bomb-cover $\mathcal{NP}$-moeilijk is door een polynomiaal begrensde reductie van 3-SAT naar Bomb-cover te geven. Deze reductie is gebaseerd op een reductie van 3-SAT naar een geometrisch verzamelingenoverdekkings-probleem door Robert J. Fowler, Michael S. Paterson, Steven L. Tanimoto. [1]

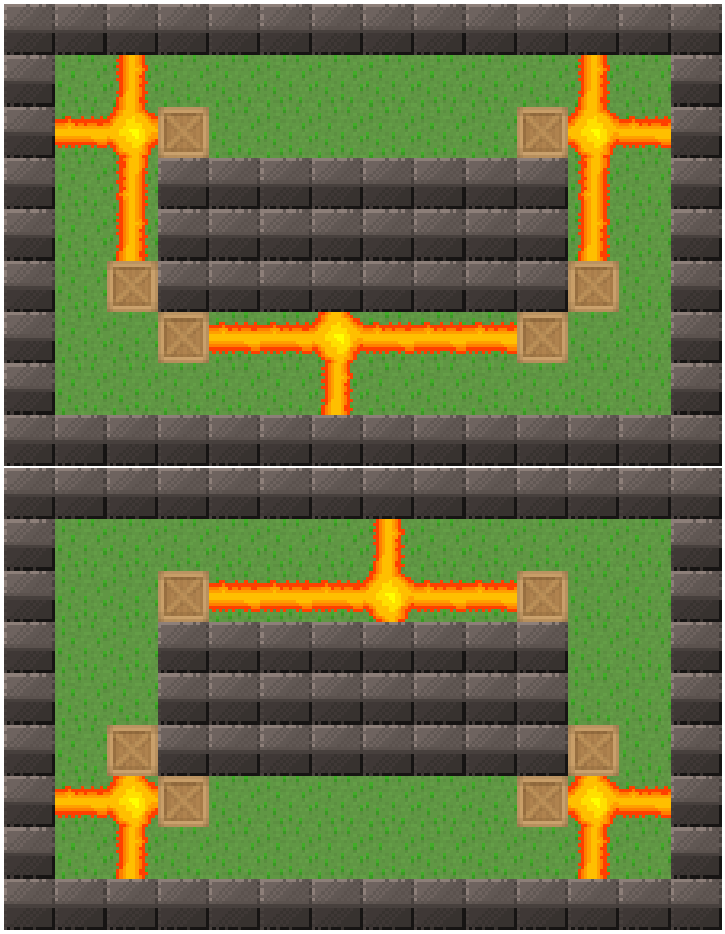
De invoer van de reductie is een 3-SAT formule met M variabelen en N clausules. De uitvoer zal een Bombermanbord zijn met kratten om te vernietigen, en een aantal explosies k, waarmee deze kratten vernietigd kunnen worden dan en slechts dan als de 3-SAT formule vervulbaar is.

We coderen elke variabele in de 3-SAT formule als een draad. Een draad is een reeks kratten die zo gepositioneerd zijn dat elk paar opeenvolgende kratten met één explosie vernietigd kan worden. Tevens kan in een draad geen enkele explosie meer dan twee kratten tegelijk vernietigen. De draden wordt binnen een "gang" van muren geplaatst zodat verschillende draden elkaar niet kunnen beïnvloeden. De draden zijn twee vakken breed zodat de speler om kratten heen kan lopen als dat nodig is om twee kratten bij elkaar te groeperen met een explosie. Zie Figuur 3.

Figuur 3: Fragment van een draad.

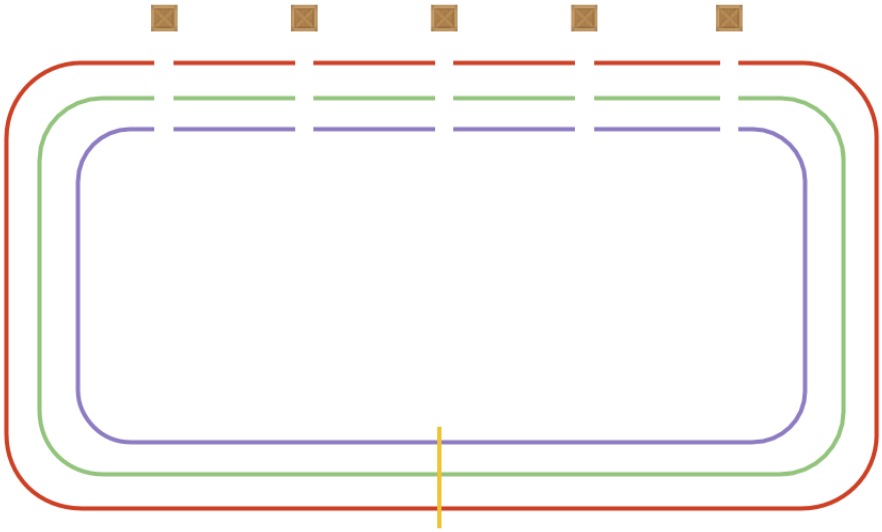
De variabelen worden gerepresenteerd door lussen van draad die een even aantal kratten bevatten. We noemen het aantal kratten in een draad corresponderend met variabele i, K_i. Er volgt dat deze

draad met minimaal $\frac{K_i}{2}$ explosies geleegd kan worden. Er zijn twee manieren om een lus te legen, zie Figuur 4. We zullen deze twee opties associëren met de mogelijke toekenningen van de variabele behorend bij de draad (waar en onwaar).



Figuur 4: De twee manieren om een lus draad te legen.

Om clausules te representeren introduceren we een tweede verzameling kratten buiten de draden, één krat voor elke clausule. We construeren een scenario waar de draad van elke betrokken variabele op een van zijn twee toekenningen het clausulekrat ook vernietigt. We doen dit door een later in te vullen opening te creëren in de draden rondom de x-coördinaat van het clausulekrat, zie Figuur 5.



Figuur 5: Schematische weergave van de hele constructie voor $N = 5$ clausules met hun clausulekratten en $M = 3$ variabelen met hun lussen van draad. De gele lijn is de ingang voor de speler

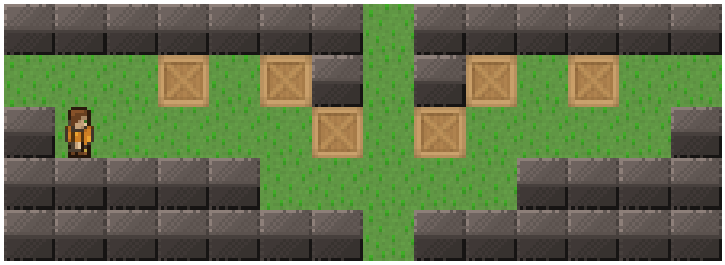
Een variabele kan op drie manieren gerelateerd zijn aan een clausule:

1. De variabele maakt de clausule waar, wanneer hij waar is.
2. De variabele maakt de clausule waar, wanneer hij onwaar is (negatie).
3. De variabele komt niet voor in de clausule.

Voor elk van deze mogelijkheden volgt nu een draad-fragment, te vinden in Figuren 6-8. Deze fragmenten zullen geplaatst worden in de eerder genoemde openingen. Tussen de openingen staat een muurvak onderin de draad. Deze muurvakken voorkomen dat de kratten uit verschillende fragmenten samen gegroepeerd kunnen worden met een explosie. De muren staan in de fragmenten reeds aan weerszijden weergegeven.

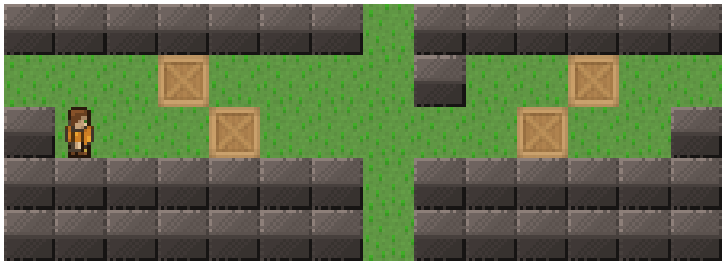
Het is niet belangrijk welke pariteit van de draad geassocieerd wordt met welke toekenning van de variabele. In dit geval gebruiken we de groepering waarbij de kratten aan de randen van de fragmenten gegroepeerd worden met een krat binnen het fragment, voor een variabele die waar is, en de andere groepering (kratten aan de rand worden gegroepeerd met een krat buiten het fragment) voor een variabele die onwaar is. We noemen deze pariteiten vanaf nu respectievelijk α en β.

Waar:
In het geval dat een variabele de clausule moet waarmaken wanneer hij waar is, maken we rondom de opening een constructie zoals in Figuur 6. Wanneer de draad pariteit α heeft, kan tussen de onderste twee kratten dynamiet afgestoken worden dat de beide onderste kratten vernietigt en nog een schokgolf door de opening stuurt, die het clausulekrat vernietigt.



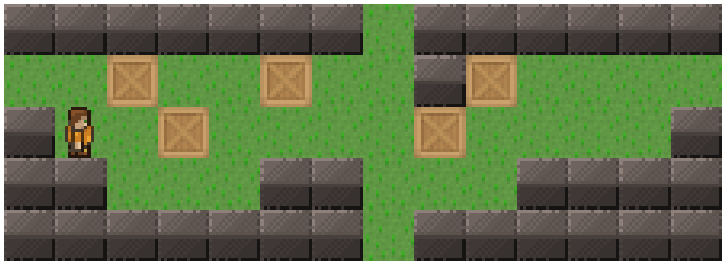
Figuur 6: Fragment dat clausulekrat gratis meepakt bij pariteit α.

Onwaar:
In het geval dat een variabele de clausule moet waarmaken wanneer hij onwaar is, maken we rondom de opening een constructie zoals in Figuur 7. Wanneer de draad pariteit β heeft, kan tussen de onderste twee kratten dynamiet afgestoken worden dat de beide onderste kratten vernietigt en nog een schokgolf door de opening stuurt, die het clausulekrat vernietigt.



Figuur 7: Fragment dat clausulekrat gratis meepakt bij pariteit β.

Niet voorkomend:
Als een variabele niet voorkomt in een clausule, mag zijn draad geen invloed hebben op het clausulekrat. Er moet echter wel een opening in de draad komen, omdat eronder gelegen draden eventueel wel het clausulekrat moeten kunnen vernietigen. We lossen dit probleem op door een extra krat in de draad te plaatsen, die net zoals het clausulekrat ook gratis meegepakt kan worden, maar dan door alle twee de pariteiten van de draad. Zie Figuur 8. De middelste krat in deze figuur is dit extra krat. Wanneer men het clausulekrat meepakt in plaats van dit extra krat, kan het extra krat niet gratis meegenomen worden en kost het legen van het bord dus een extra explosie. We noemen het aantal van dit soort extra kratten (en dus het aantal clausles waarin i niet voorkomt) in een draad behorende bij variabele i: E_i.



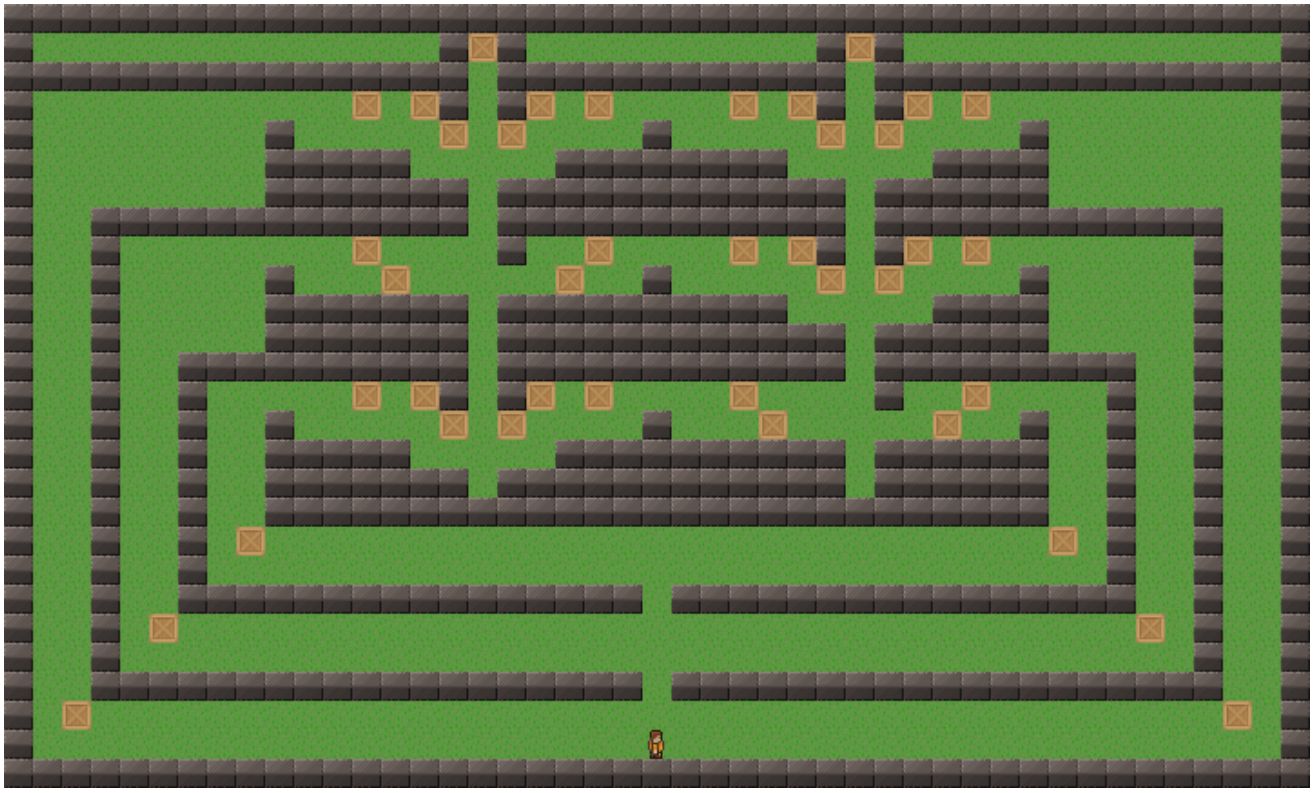
Figuur 8: Fragment dat het clausulekrat niet meepakt.

We delen het bord nu als volgt in:

- We construeren concentrische lussen van draad. Voor elke variabele een lus.
- We construeren een gang waar de speler in begint, en die leidt naar elke lus: in Figuur 5 is deze gang geel.
- We plaatsen voor elke clausule een extra clausulekrat boven de lussen, tussen de uiterste x-coördinaten van de binnenste lus.
- We maken, afhankelijk van de rol van de variabele in de clausule, een van de drie fragmenten in figuren 6-8 in elke lus op de x-coördinaten van de clausulekratten.

Als een van de draden het clausulekrat vernietigt, is de clausule waar. Als geen van de draden het clausulekrat vernietigt, is er een extra explosie nodig om dit krat weg te halen.

Hier volgt een voorbeeld van een bord gegenereerd met deze reductie en de bijbehorende k.



Figuur 9: Het bord voor $(x \vee \neg y \vee z) \wedge (\neg x \vee y \vee z)$. Voor deze formule geldt $k = 19$

De lussen representeren van binnen naar buiten de variabelen x, y, z. De clausulekratten komen van links naar rechts in dezelfde volgorde voor als de clausulen in de 3-SAT formule.

Er geldt nu dat het hele bord geleegd kan worden in $\sum_{i=1}^{M}(K_i - E_i)/2$ explosies dan en slechts dan als de 3-SAT formule vervulbaar is. Deze reductie resulteert in een bord van hoogte O(M) en breedte O(M+N). De reductie bestaat uit het construeren van M draden en N clausule scenario's. Onze reductie van 3-SAT naar Bomb-cover is in polynomiale tijd uit te voeren. Bomb-cover is derhalve NP-moeilijk. Dit besluit het bewijs van Lemma 1.2

Uit de bewijzen van Lemma 1.1 en Lemma 1.2 volgt direct dat Bomb-cover $\mathcal{NP}$-volledig is. Hiermee is Stelling 1 bewezen.

3 Bomb-r-cover

Explosies hebben tot nog toe een onbeperkte radius gehad. We definiëren een nieuwe verzameling beslissingsproblemen met een eindige radius r: enkel kratten op afstand ten hoogste r worden vernietigd door explosies. Nog steeds geldt dat muurvakken een explosie vroegtijdig kunnen stoppen. We noemen dit probleem **Bomb-r-cover**.

Probleem 2 *Bomb-r-cover: Gegeven een Bombermanbord, een beginpositie voor de speler, en een*

getal $k \in \mathbb{N}$, bestaat er een reeks van k explosies met radius r, elke explosie direct bereikbaar vanaf de vorige, die het bord leegt van bereikbare kratten?

Vermoeden 1 *Bomb-r-cover is $\mathcal{NP}$-volledig.*

Om te bewijzen dat Bomb-r-cover $\mathcal{NP}$-volledig is, zouden we moeten bewijzen dat het in $\mathcal{NP}$ zit en $\mathcal{NP}$-moeilijk is.

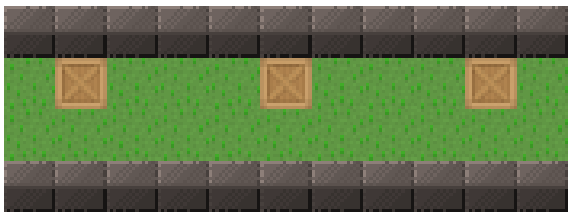
Lemma 2.1 *Bomb-r-cover zit in $\mathcal{NP}$*

Bewijs. Hetzelfde niet-deterministische, polynomiale algoritme beschreven voor Bomb-cover kan gebruikt worden om aan te tonen dat Bomb-r-cover NP is. Dit besluit het bewijs van Lemma 2.1.

Vermoeden 2 *Bomb-r-cover is $\mathcal{NP}$-moeilijk.*

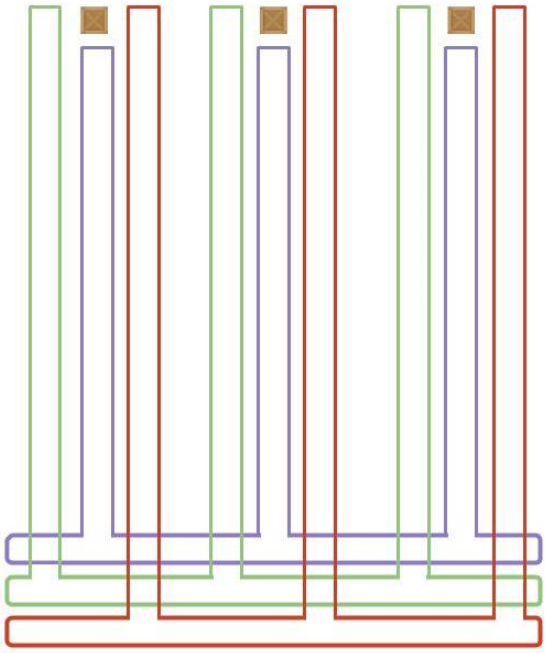
Deelbewijs. De reductie voor Bomb-cover is niet direct toe te passen op Bomb-r-cover. Er is een aantal aanpassingen benodigd.

Aangezien de radius beperkt is, moeten draden nu op vaste afstanden kratten bevatten. De kratten staan $2r$ vakjes uit elkaar. Zie Figuur 10.

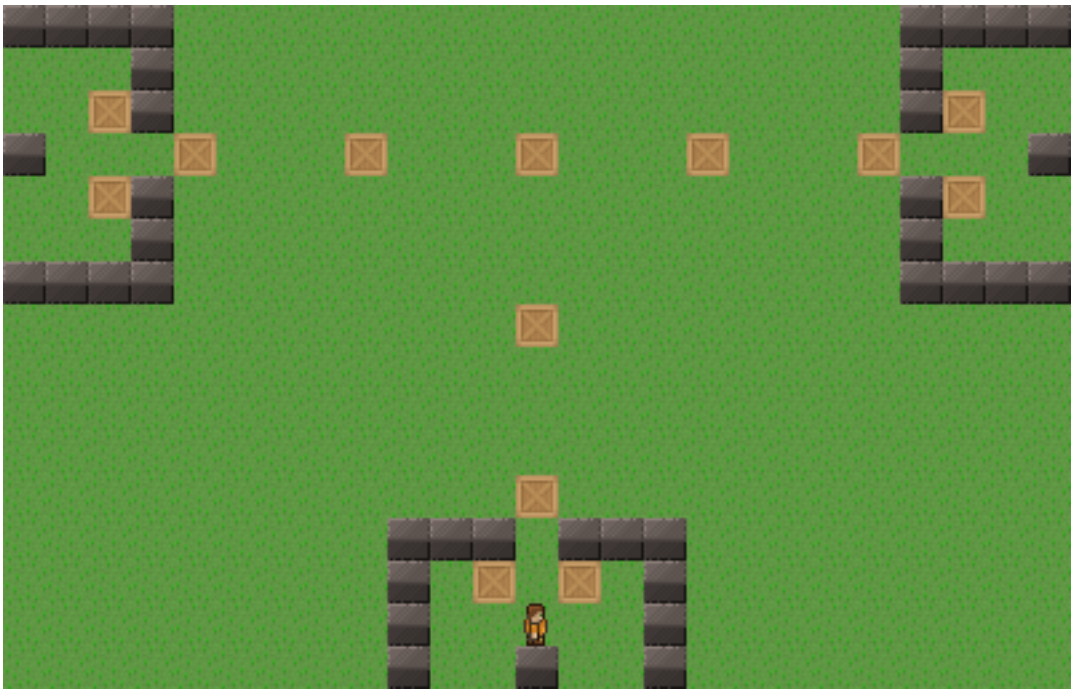


Figuur 10: Een draad voor $r = 2$.

Ook de clausulekratten moeten anders gepositioneerd worden. Omdat de radius beperkt is, kunnen draden niet meer onbeperkt onder elkaar geplaatst worden. Naarmate de radius afneemt zullen immers steeds meer van de binnenste draden dusdanig ver van de clausulekratten verwijderd zijn, dat een explosie in die draden niet meer tot het clausulekrat reikt. In plaats van de oude constructie zullen we voor deze versie van het probleem vanaf meerdere kanten het clausulekrat moeten benaderen, zie Figuur 11. We gebruiken hiervoor zeven kratten, geplaatst zoals in Figuur 12. We noemen deze zeven kratten vanaf nu een clausulepunt. Elk van de draden die participeren in de clausule kan op een van zijn pariteiten een van de zeven kratten gratis meepakken. Wanneer een of meer van de draden dit doet, kunnen de overgebleven 4, 5 of 6 kratten met drie explosies weggehaald worden. Als geen van de draden dit doet, zullen er vier explosies nodig zijn om de zeven kratten weg te halen. Door de extra kratten zijn er extra explosies nodig voor elke clausule. Hier wordt rekening mee gehouden in de berekening van k.

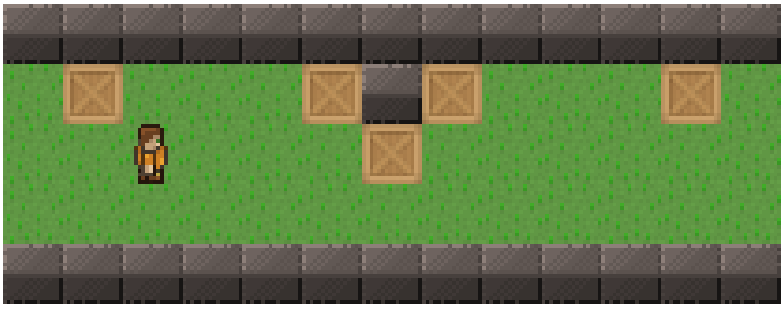


Figuur 11: Schematische weergave van de hele reductie voor $M = 3$ variabelen en $N = 3$ clausules



Figuur 12: Clausulekratten voor $r = 2$.

Bij Bomb-r-cover werken we, vanwege deze nieuwe clausulepunten, niet meer met de fragmenten voor de verschillende rollen van de variabelen. We gebruiken in plaats daarvan een constructie zoals in Figuur 13 om tijdelijk van pariteit te wisselen in het geval van een negatie. Twee van deze constructies, voor en na een clausulepunt, zorgen dat negatie correct afgehandeld wordt. Het is niet nodig nog apart rekening te houden met variabelen die niet voorkomen in een clausule. We brengen de draden van die variabelen simpelweg niet naar de clausulepunten. Als door een wissel of bocht de kratten in de draden niet goed uitkomen voor de clausulepunten, kan ook (meerdere keren achter elkaar) iets minder ruimte tussen kratten gelaten worden.

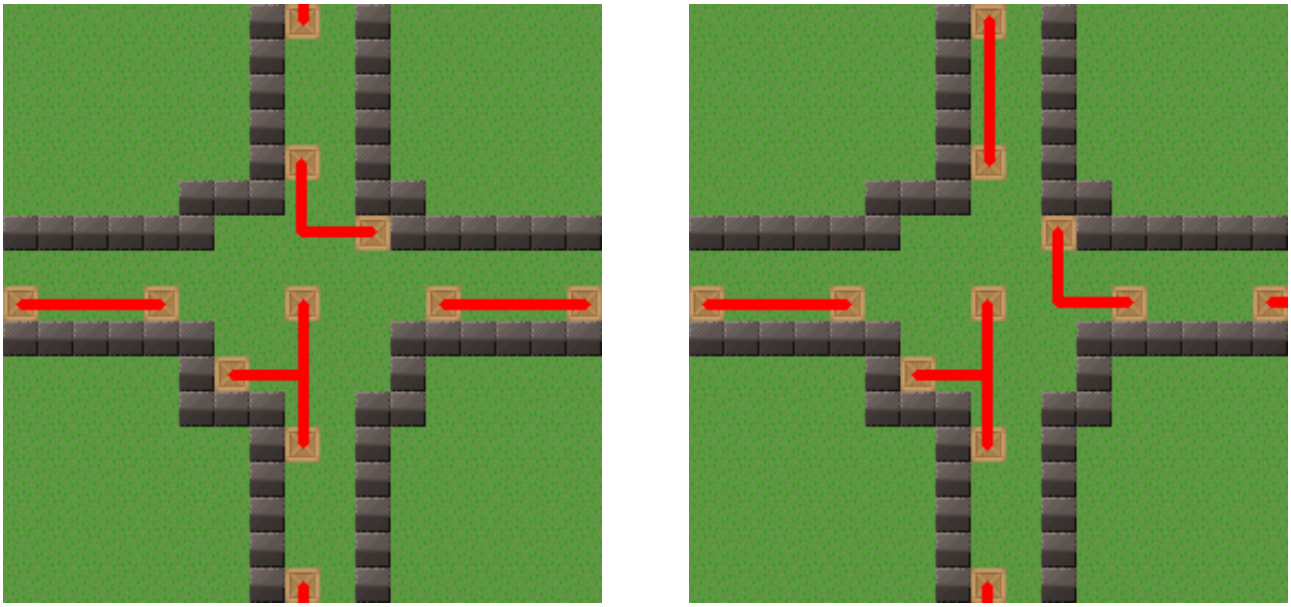


Figuur 13: Pariteit wisselen voor $r = 2$.

We delen het bord nu als volgt in (zie ook Figuur 11):

1. We construeren lussen van draad. Voor elke variabele een lus.
2. We construeren een gang waar de speler in begint, en die leidt naar elke lus.
3. We plaatsen voor elke clausule een extra clausulepunt boven de lussen.
4. We brengen de draden van de variabelen betrokken in de clausule naar dit clausulepunt toe.
5. We construeren een situatie als in Figuur 12 met deze draden en het clausulepunt.

In stap 4 hierboven lopen we tegen een probleem aan:
Omdat de draden nu van verschillende kanten bij een clausulepunt moeten komen, moet het mogelijk zijn voor de draden om elkaar te kruisen. Preciezer: Twee draden moeten elkaar kunnen kruisen, terwijl ze beide gegarandeerd hun pariteit behouden. Tot nog toe hebben we niet zo'n soort kruising kunnen vinden. Deze reductie werkt dus momenteel niet, maar zou werkend te krijgen zijn met zo'n kruising. In Figuur 14 staat een voorbeeld van een kruising die niet werkt.



(a) Voor de explosie.

(b) De armen reiken tot een niet-leeg vak.

Figuur 14: Een kruising die niet goed werkt.

In deze figuur staan twee situaties in dezelfde kruising weergegeven. Een rode lijn tussen twee kratten geeft aan dat deze kratten samen met één explosie vernietigd worden (deze kratten worden samen gegroepeerd). De linkerzijde van de horizontale draad en de onderzijde van de verticale draad zijn hetzelfde gegroepeerd in beide situaties. Er zijn echter twee manieren om verder te groeperen, weergegeven in de twee plaatjes. In Figuur 14a wordt het bovenste krat in de verticale draad gegroepeerd met een krat buiten de figuur en de meest rechter krat in de horizontale draad met een krat binnen de figuur. In Figuur 14b is dit juist omgekeerd. Beide plaatjes gebruiken evenveel explosies. Er kan dus niet gegarandeerd worden dat de pariteit van de beide draden behouden blijft.

Met een werkende kruising zou gelden dat het hele bord geleegd kan worden in $\sum_{i=1}^{M} K_i/2 + 3N$ explosies dan en slechts dan als de 3-SAT formule vervulbaar is. De term $3N$ correspondeert hierbij met de N clausulepunten. Deze reductie zou resulteren in een bord van hoogte $O(M)$ en breedte $O(N)$. Hij zou bestaan uit het construeren van M draden, N clausule scenario's en $O(M * N)$ kruisingen.

Mocht er een manier zijn om draden te kruisen zoals hierboven beschreven, dan kan met deze reductie aangetoond worden dan Bomb-r-cover NP-moeilijk is en dus, vanwege Lemma 2.1, $\mathcal{NP}$-volledig is.

4 Gerelateerd Werk: Box-cover

Bomb-cover lijkt sterk op het geometrische verzamelingenoverdekkings-probleem. De bovenstaande bewijzen voor NP-moeilijkheid zijn dan ook gebaseerd op een bewijs dat een specifieke instantie van het geometrische verzamelingenoverdekkings-probleem, Box-cover, NP-moeilijk is. Dit bewijs is gepubliceerd in 1981 door Robert J. Fowler, Michael S. Paterson, en Steven L.Tanimoto [1].

Box-cover luidt als volgt: Gegeven een verzameling punten in het (2D) vlak, en een aantal vierkanten van een gegeven formaat, is het mogelijk alle punten te bedekken met de vierkanten?

Het artikel beschrijft een reductie van 3-SAT naar Box-cover. Er worden draden van punten gebruikt om de variabelen te representeren. Alle punten in een draad staan dusdanig ver uit elkaar dat elk opeenvolgend paar punten samen overdekt kan worden door een vierkant, maar er nooit meer dan twee punten tegelijk overdekt kunnen worden. Elke draad is een lus met een even aantal punten, waardoor er twee manieren zijn (corresponderend met waar en onwaar) om alle punten in een draad te overdekken. Wij hebben een vergelijkbare constructie voor kratten (ons equivalent van de punten) en explosies (ons equivalent van vierkanten).

Om de disjuncties te representeren worden er punten geconstrueerd waar drie draden (en dus variabelen) samen komen. Tussen de drie draden staat een extra punt. Elk van de drie draden kan het extra punt "gratis" meepakken op een van zijn twee pariteiten. Als geen van de drie draden het extra punt meepakt, moet het punt met een extra vierkant overdekt worden. Om de drie draden bij elkaar te brengen moet het mogelijk zijn de draden te kunnen kruisen. Dit bleek mogelijk bij Box-cover. Het is ons niet gelukt dit voor elkaar te krijgen bij Bomb-r-cover (met beperkte explosieradius). Bij Bomb-cover (met onbeperkte explosieradius) hebben we een manier gevonden om het kruisen geheel te omzeilen: de concentrische lussen draad.

Voor meer informatie over Box-cover, zie het originele artikel [1].

5 Optimalisatieprobleem

Zoals veel beslissingsproblemen, heeft ook Bomb-cover een optimalisatie variant, die wij **Minimale Bomb-cover** noemen.

Probleem 3 *Minimale Bomb-cover: Gegeven een Bombermanbord, en een beginpositie van de speler, wat is de kleinste $k \in \mathbb{N}$ zodat er een reeks van k explosies bestaat, elke explosie direct bereikbaar vanaf de vorige, die het bord leegt van bereikbare kratten?*

We beschouwen hier weer de versie van het probleem waarbij explosies onbeperkte radius hebben. Omdat Bomb-cover NP-volledig is, is zijn optimalisatie-variant NP-moeilijk. We kunnen dus niet verwachten een polynomiaal algoritme te vinden dat het oplost. Het gretige algoritme dat volgt kan wel binnen redelijke tijd oplossingen vinden.

5.1 Gretige vernietiging

Om voor het optimalisatieprobleem het gretig legen van een Bombermanbord te analyseren, introduceren we de term vernietigingskracht.

Definitie 3 *De vernietigingskracht van een leeg vakje is gedefinieerd als het aantal kratten dat vernietigd zou worden wanneer er dynamiet op dit vakje wordt afgestoken.*

Een gretig algoritme voor het weghalen van kratten gaat als volgt: Voor elk bereikbaar vakje, bereken de vernietigingskracht. Herhaal zo lang er nog bereikbare kratten zijn:

- Loop naar het (indien meerdere, dichtstbijzijnde) vakje dat de hoogste vernietigingskracht heeft.
- Steek hier dynamiet af.
- Werk vernietigingskrachten bij.

Er zijn drie kleinere problemen die moeten worden opgelost om dit algoritme uit te voeren.

- De vernietigingskrachten voor het oorspronkelijke bord moeten worden berekend.
- Het dichtstbijzijnde, bereikbare vakje met de hoogste vernietigingskracht moet worden gevonden.
- De vernietigingskrachten moeten worden bijgewerkt wanneer er kratten vernietigd worden.

In de volgende drie paragrafen bespreken we deze deelproblemen. Voor elk probleem bespreken we de werking van een algoritme dat het oplost. We analyseren de complexiteit van elk van deze algoritmen en we geven er in twee gevallen pseudocode voor.

5.2 Het berekenen van vernietigingskrachten

Werking

Om het algoritme te beginnen moeten we de vernietigingskrachten van alle vakjes berekenen. We doorlopen hiervoor elke rij en elke kolom en houden bij of we tussen muren en/of kratten zitten. Alle vernietigingskrachten beginnen op 0. Het algoritme kijkt voor elke reeks lege vakken (horizontaal en verticaal) welk soort vakken er aan weerszijden zitten. Als de reeks tussen twee kratten zit, wordt er 2 opgeteld bij de vernietigingskrachten van alle vakken in de reeks. Als de reeks tussen een krat en een muur zit, wordt er 1 opgeteld bij de vernietigingskrachten. Als de reeks tussen twee muren zit, gebeurt er niets.

Complexiteit

Deze strategie bezoekt elk vakje van het bord maximaal vier maal en heeft dus complexiteit $O(mn)$. Dit is beter dan voor elk vakje alle vier de explosie-richtingen op lopen totdat een muur of krat gevonden is. Dat heef namelijk complexiteit $O(mn * (m + n))$.

Pseudocode

```
calculateDestructionValues():
    create map from Square to integer destructionValue
    create empty set of Squares S
    integer count = 0
    //horizontal pass
    for each row X
        for each square Y in X
            if Y is emptySquare
                    S.add(Y)
            else if Y is a crate
                if S is not empty
                    for each Square Z in S
                        destructionValue[Z] += count + 1
                    S.clear()
                count = 1
            else //Y is a wall
                if S is not empty
                    for each Square Z in S
                        destructionValue[Z] += count
                    S.clear()
                count = 0

    //vertical pass
    for each column X
        for each square Y in X
            if Y is emptySquare
                    S.add(Y)
            else if Y is a crate
                if S is not empty
                    for each Square Z in S
                        destructionValue[Z] += count + 1
                    S.clear()
                count = 1
            else //Y is a wall
                if S is not empty
                    for each Square Z in S
                        destructionValue[Z] += count
                    S.clear()
                count = 0
```

5.3 Het dichstbijzijnde vakje met de hoogste vernietigingskracht vinden

Werking

Dit kan gedaan worden met een breadth first search. Omdat een BFS eerst nabijere vakjes beschouwd, zal het gevonden vakje met de hoogste vernietigingskracht ook (een van) de dichtstbijzijnde zijn.

Complexiteit

Breadth first search in een graaf (V, E) heeft complexiteit $O(|V|+|E|)$. In ons geval geldt: $|V| = m*n$, en $|E| = n(m-1) + m(n-1)$. Dus voor een Bombermanbord is de complexiteit $O(m * n)$.

5.4 Bijwerken van vernietigingskrachten

Werking

Wanneer een krat vernietigd is, moeten de vernietigingskrachten worden bijgewerkt. We herberekenen niet voor het hele veld opnieuw de vernietigingskrachten ($O(mn)$), omdat het weghalen van een krat enkel effect heeft op de rij en kolom waar het krat zich in bevindt. We hoeven zelfs niet eens die hele rij en kolom opnieuw te berekenen.
Een betere manier is om de veranderingen van vernietigingskrachten te beschouwen vanuit het oogpunt van de lege vakjes die direct in verbinding staan met het krat dat vernietigd wordt.
Als het eerste niet lege vakje links van het vernietigde krat een krat is, hoeven de vernietigingskrachten van de lege vakjes rechts van het vernietigde krat niet aangepast te worden. In het geval dat het eerste niet lege vakje een muur is, moeten de vernietigingskrachten van de vakjes aan de rechterzijde van het vernietigde vak met 1 verlaagd worden.

Dit idee toegepast op alle vier richtingen laat enkel nog het vakje waar het nu vernietigde krat stond. De vernietigingskracht van dit vakje hebben we echter al impliciet berekend: Dit is namelijk het aantal keren dat we niet met 1 hebben hoeven verlagen.

Complexiteit

Dit algoritme heeft een slechtste geval complexiteit van $O(m + n)$, maar zal gemiddeld beter presteren. Dit omdat in plaats van de hele rij en kolom, enkel de delen die direct verbonden zijn met het vernietigde krat in beschouwing genomen worden.

Pseudocode

```
1 updateDestructionValues(integer y, integer x):
2     for each direction d
3         Square tmp = neighbour of board[y][x] in direction d
4         while(tmp is emtpySquare)
5             tmp = neighbour of tmp in direction d
6
```

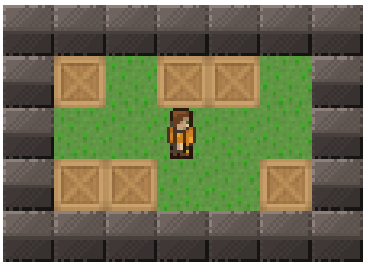
```
        if tmp is wall
            tmp2 = neighbour of board[y][x] in opposite direction of d
            while tmp2 is emptySquare
                destructionValue[tmp2] = destructionValue[tmp2] - 1;
                tmp2 = neighbour of tmp2 in opposite direction of d
        else //tmp is a crate
            destructionValue[board[y][x]] = destructionValue[board[y][x]] + 1
```

5.5 Het hele algoritme

Het hele algoritme kan nu als volgt beschreven worden.

```
create set of reachable target crates C
calculateDestructionValues()
while(C is not empty)
    Path toGoal := BFS()
    move(toGoal)
    lightDynamite()
    updateDestructionValues()
```

Zoals eerder gesteld is kunnen we niet verwachten dat dit algoritme altijd de beste oplossing geeft. Een eenvoudig tegenvoorbeeld is Figuur 15.



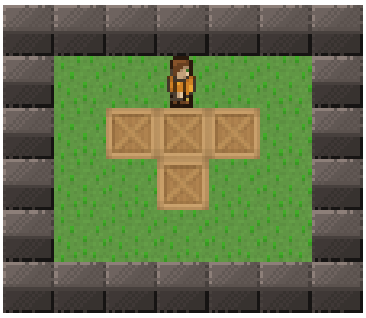
Figuur 15: Tegenvoorbeeld gretig algoritme.

Hier is het dichtstbijzijnde vakje met de grootste vernietigingskracht niet de optimale plek om dynamiet af te steken. Er zijn drie vakjes met de hoogste vernietigingskracht (3). Vanaf de speler gezien zijn dat:

1. Een vakje naar onder.
2. Een vakje naar onder en een vakje naar rechts.
3. Een vakje naar links en een vakje naar boven.

Het dichtstbijzijnste is vakje 1. Wanneer dit vakje als eerste gekozen wordt, kunnen de overgebleven kratten niet in één explosie vernietigd worden. Wanneer vakje 3 als eerste gekozen wordt, is dit wel het geval.

Het is zelfs zo dat in sommige gevallen geen enkel van de vakjes met de grootste vernietigingskracht de optimale zet is:



Figuur 16: Tegenvoorbeeld gretig algoritme.

In Figuur 16 zijn er twee vakjes met vernietigingskracht 2. Vanaf de speler gezien zijn dat:

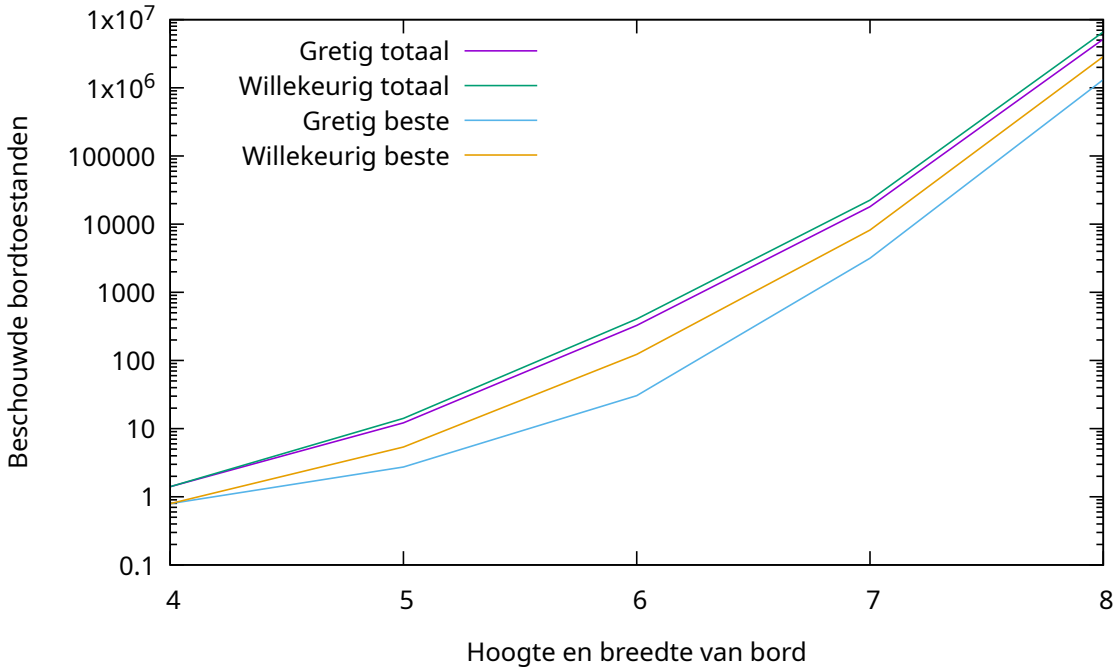
1. Twee vakjes naar onder en een naar links.
2. Twee vakjes naar onder en een naar rechts.

De rest van de vakjes heeft vernietigingskracht 1 of 0. Wanneer een van de vakjes met vernietigingskracht 2 als eerste gekozen wordt, kunnen de overgebleven vakjes niet in één explosie vernietigd worden. Wanneer op de beginpositie van de speler dynamiet afgestoken wordt, kan dit wel.

5.6 Brute kracht

De gretige strategie zoals hierboven beschreven kan dus niet gebruikt worden om Minimale Bombcover op te lossen. Zij kan echter wel toegepast worden in een brute force strategie. We kiezen steeds gretig de eerstvolgende explosie tot het bord leeg is. Dan gaan we terug naar de (vanaf achter) eerste plek in ons pad naar dit lege bord waar we een andere keuze hadden kunnen maken, en maken de op een-na gretigste keuze. Zodoende lopen we alle mogelijke opties van gretig, naar steeds minder gretig af.

Deze strategie lijkt sneller de optimale oplossing te vinden dan een willekeurige brute force aanpak. Om dit te onderzoeken vergelijken we deze twee strategien in een experiment. Beide strategieën pogen voor vijf bord-formaten op 1200 willekeurig gegenereerde borden zo snel mogelijk een oplossing te vinden. De willekeurige borden zijn omrand met een muur, en elk vakje binnen de muur heeft 50% kans om een leeg vak te zijn, 18% kans om een muurvak te zijn en 32% kans om een kratvak te zijn. Deze kansen zijn gekozen met als doel een bord te maken dat voldoende kratten bevat, en waar nog steeds veel bewandelbare vakjes zijn. De speler wordt op een willekeurig leeg vakje gezet. Beide strategieën kappen takken af in het brute-force proces die minstens zo diep gaan gaan als de diepte van de, tot nu toe, beste oplossing. We meten het totaal aantal beschouwde bordtoestanden en het aantal beschouwde bordtoestanden voordat de beste oplossing gevonden was. De gemiddeldes van de 1200 borden zijn weergegeven in Figuur 17. De bordformaten in deze figuur zijn inclusief rand. Dit betekent dat een 4x4 bord feitelijk een speelruimte heeft van 2x2.



Figuur 17: Gretig zetten kiezen in vergelijking tot willekeurig tijdens brute force.

Bordgrootte	Gretig/Willekeurig totaal	Gretig/Willekeurig beste
4x4	1.000	1.000
5x5	0.8585	0.5069
6x6	0.8049	0.2476
7x7	0.8009	0.3856
8x8	0.7830	0.4572

Tabel 1: Verhoudingen tussen gretige en willekeurige algoritmes.

De verhoudingen tussen de aantallen beschouwde bordtoestanden door het gretige en willekeurige algoritme staan in Tabel 1.

In eerste instantie valt op dat op 4x4 borden de optimale oplossing door beide algoritmes even snel gevonden is. Dit ligt aan het feit dat er vaak maar 1 zet mogelijk is op een bord van dat formaat. Daarnaast is af te lezen dat de gretige strategie niet per se sneller de optimale oplossing vindt in verhouding tot de willekeurige naarmate de bordgrootte toeneemt, maar wel steeds sneller het volledige algoritme afrondt: de verhoudingen nemen af. De oorzaak hiervan is dat het gretige algoritme meer kan snoeien, omdat het eerder een goede of de beste oplossing heeft. Dit effect is echter steeds minder sterk: de verhoudingen tussen de verhoudingen nemen ook af.

6 Conclusie en verder onderzoek

In Hoofdstuk 2 hebben we bewezen dat Bomb-cover in $\mathcal{NP}$ zit door een niet-deterministisch polynomiaal begrensd algoritme te geven dat het oplost. Daarna hebben we bewezen dat Bomb-cover $\mathcal{NP}$-moeilijk is door een reductie te geven van 3-SAT naar Bomb-cover. Deze twee bewijzen vormen samen het bewijs voor de stelling dat Bomb-cover $\mathcal{NP}$-volledig is.

In Hoofdstuk 3 beschrijven we het vermoeden dat Bomb-r-cover, hetzelfde probleem maar dan met beperkte explosieradius, $\mathcal{NP}$-volledig is. We hebben bewezen dat Bomb-r-cover in $\mathcal{NP}$ zit met hetzelfde algoritme als in Hoofdstuk 2. We hebben vervolgens gepoogd de reductie uit Hoofdstuk 2 aan te passen zodat het werkt voor een beperkte explosieradius. Veel aspecten van de reductie konden succesvol aangepast worden, maar we liepen vast op het kruisen van de draden. De $\mathcal{NP}$-volledigheid van Bomb-r-cover blijft dus een vermoeden.

In Hoofdstuk 5 beschreven we een gretig algoritme dat de optimalisatievariant van Bomb-cover oplost. Het algoritme presteerde in een experiment beter dan een willkeurige brute-force strategie.

Voor toekomstig onderzoek zouden er, naast explosieradius, talloze andere parameters gevarieerd kunnen worden. Sommige hiervan zijn power-ups in het spel:

- Explosies stoppen niet bij het raken van een krat, maar kunnen meerdere kratten doordringen. (We vermoeden dat het bewijs uit Hoofdstuk 2 met wat aanpassingen ook zal werken voor deze variant.)
- De speler kan dynamiet "over kratten heen gooien", waardoor het op het eerste lege vakje na een reeks kratten terecht komt.
- Meerdere spelers op een bord: welke kratten worden door welke speler meegepakt?

De reductie van 3-SAT naar Bomb-r-cover in Hoofdstuk 3 is niet volledig. Deze reductie afmaken door een kruising te vinden is een voor de hand liggend vervolg. Wellicht is er een reductie te vinden vanaf een ander probleem dan 3-SAT.

Daarnaast is het interessant de invoer van het algoritme in het bewijs van Lemma 1.1 anders te geven. Het algoritme is polynomiaal in de invoer als de invoer wordt gegeven als $m * n$ vakjes: alle lege vakken, muren en kratten. Een Bombermanbord kan natuurlijk ook beschreven worden met twee van de drie soorten vakken erin. De niet aangegeven vakken zijn dan van de laatste soort. Zodoende kan het bord met minder informatie beschreven worden, en is het de vraag of het NP-algoritme nog polynomiaal is in de invoer.

Referenties

[1] Robert J. Fowler, Michael S. Paterson, and Steven L. Tanimoto. Optimal packing and covering in the plane are NP-complete. *Information Processing Letters*, 12:133–137, 1981.

[2] Michael R. Garey and David S. Johnson. *Computers and Intractability*. W. H. Freeman and Company, 1979.

[3] Joseph Groot Kormelink, Madalina Drugan, and Marco Wiering. Exploration methods for connectionist q-learning in bomberman. In Ana Paula Rocha and Jaap van den Herik, editors, *Proceedings of the 10th International Conference on Agents and Artificial Intelligence*. pagina 355-362. ACM, januari 2018.

[4] Juarez Monteiro, Roger Granada, Rafael Pinto, and Rodrigo C Barros. Beating bomberman with artificial intelligence. In *Encontro Nacional de Inteligência Artificial e Computacional*, ENIAC 2018. pagina 353-364. SBC, oktober 2018.

[5] usr_share. Bomb party - the complete sprite set. https://opengameart.org/content/bomb-party-the-complete-set. Benaderd op: 2-7-2019.