2023

Verslag Prolog

ROBOT REBOOT IN SWIPL

WARD MEERSMAN

Inhoud

Inleiding

In dit project werd een command-line interface geprogrammeerd in swipl om het spel Robot Reboot te spelen. In de command-line wordt een unicode-representatie van het spel geaccepteerd via standaardinvoer om zo het spel te laden. In de interactieve modus wordt steeds de huidige staat van het spelbord weergegeven via standaard output. Daarnaast zal het programma ook een optimale oplossing kunnen berekenen en willekeurige spelborden genereren.

Ten eerste wordt beschreven hoe Robot Reboot wordt gespeeld. Daarna wordt de interne voorstelling van het bord besproken. Het derde deel behandelt hoe een optimale oplossing kan worden berekend en hoe willekeurige spelborden gegenereerd worden. De conclusie gaat uiteindelijk ook in op een mogelijke uitdaging voor een volgend project.

Besturing

Het spel Robot Reboot is een online puzzel gebaseerd op het bordspel Ricochet Robots. Het spel bestaat uit een bord omringd door muren met een aantal gekleurde robots en waarbij op willekeurige plaatsen muren werden neergezet als obstakels voor die robots. Het doel van het spel is om in zo weinig mogelijk zetten één specifieke robot naar het doel te brengen. In één zet, kan één enkele robot bewegen en die robot blijft doorgaan in dezelfde richting tot hij een obstakel tegenkomt. Een obstakel kan een muur of een andere robot zijn.

In het spel zijn de robot met index 0 (▣) en het doel (◎) allebei in het groen aangeduid om duidelijk aan te geven dat ze bij elkaar horen. De andere robots zijn in het rood aangeduid. Daarnaast herkent de speler ook duidelijk welke robot hij bedient, doordat die robot knippert. Als de speler op “tab” drukt, bestuurt de speler de volgende robot. De robots kunnen bestuurd worden met de “wasd”- of de “zqsd”-toetsen. Met ‘w’ of ‘z’ om omhoog te gaan, ‘a’ of ‘q’ om naar links te gaan, ‘s’ om naar beneden te gaan en ‘d’ om naar rechts te gaan. Als de speler de oplossing van de puzzel niet vindt, of helemaal opnieuw wil beginnen kan dat door op “spatie” te drukken. Op dat moment sluit het spel af. Als de speler de oplossing heeft gevonden, wordt hij gefeliciteerd, waarna het spel pas afsluit nadat de speler op “enter“ drukt.

Interne voorstelling van het bord

Het bord wordt voorgesteld als een lijst van alle objecten die in het bord zitten plus de hoogte en breedte van het bord. Om te vermijden dat eenzelfde muur meerdere keren in die lijst voorkomt, wordt enkel de boven – en linker zijde van het vakje met de muur bijgehouden. Zo wordt een verticale muur voorgesteld als “muur(<X>,<Y>,<X-1>,<Y>)” en een horizontale muur als “muur(<X>,<Y>,<X>,<Y-1>)”. Een robot wordt in de lijst weergegeven als “robot(<Index>,<X>,<Y>)” en het doel als “doel(<X>,<Y>)”. Als laatste worden de hoogte en breedte voorgesteld als “height(<Hoogte>)” en “width(<Breedte>)”.

Het voorbeeldbord ziet er dus als volgt uit:

[muur(0,0,0,-1),muur(1,0,1,-1),muur(2,0,2,-1),muur(3,0,3,-1),width(4),muur(0,0,-1,0),robot(1,0,0),muur(2,0,1,0),muur(4,0,3,0),muur(3,1,3,0),muur(0,1,-1,1),robot(0,2,1),muur(4,1,3,1),muur(1,2,1,1),muur(0,2,-1,2),doel(1,2),muur(2,2,1,2),muur(4,2,3,2),muur(0,3,-1,3),muur(1,3,0,3),muur(4,3,3,3),muur(0,4,0,3),muur(1,4,1,3),muur(2,4,2,3),muur(3,4,3,3),height(4)]

Algoritmes

Oplossen puzzels

Om een optimale oplossing te krijgen is het aangewezen om eerst de borden te overlopen waarop de minste zetten worden toegepast. Daarom werd gekozen een breadth first search-algoritme (BFS) toe te passen. Bij het inlezen van een bord worden eerst alle mogelijke zetten van dat bord toegepast en de nieuwe borden worden toegevoegd aan een wachtrij. Daarna wordt steeds het eerste bord uit de wachtrij gehaald, worden opnieuw alle zetten op het betreffende bord toegepast en worden die nieuwe borden toegevoegd aan de wachtrij. Dit proces herhaalt zich tot er een oplossing gevonden is.

Hieronder worden enkele optimalisaties besproken van het BFS-algoritme en hun impact op de snelheid van het algoritme. De tijdsmetingen worden telkens bepaald op de borden uit map “extra/maps_only0/”. Elk spelbord uit deze map heeft hetzelfde antwoord; namelijk (0D,0L,0U).

Het algoritme houdt ook rekening met borden in de wachtrij die reeds eerder werden bekeken. Door de interne voorstelling van een bord zijn de borden na de zetten (1L, 0D) en (0D, 1L) voor het algoritmetotaal verschillende borden, terwijl een stringvoorstelling eenzelfde bord weergeeft. Er werd daarom gekozen om de stringvoorstelling te bewaren i.p.v. de interne voorstelling van het bord.

Naam bestand	Gemiddelde (ms)
map_04x04_1.txt	64,1
map_04x04_2.txt	263,9
map_04x04_4.txt	811,7
map_04x04_8.txt	1084,3
map_08x08_1.txt	334,8
map_08x08_2.txt	1688,2
map_08x08_4.txt	5793,4
map_08x08_8.txt	42773
map_16x16_1.txt	4408,9
map_16x16_2.txt	21516,4
map_16x16_4.txt	117018,4
map_16x16_8.txt	871812

Tabel 1 Tijdsmetingen in ms voor de stringvoorstelling van een bord

De volgende optimalisatie omvat 2 versnellingen. Als eerste wordt i.p.v. de stringvoorstelling wordt een gesorteerde lijst van de robots op index met hun coördinaat opgeslagen. Omdat de robots de enige zijn die bewegen, is het niet nuttig om de muren ook te bewaren. Zoals eerder aangegeven, heeft de gesorteerde lijst als functie te vermijden dat de volgorde van de bewegingen een andere bordvoorstelling weergeeft.

De tweede optimalisatie is de volgorde van de nieuwe borden toevoegen aan de wachtrij. De zetten worden gesorteerd volgens robot index, waarbij eerst de zetten van robot 0 worden weergegeven, daarna van robot 1 en zo verder. Deze optimalisatie is vooral te zien op in de laatste laag, zo zal in elke laag eerst de zetten van robot 0 bekeken worden. Dat geeft het onderstaande tijdsmetingen:

Naam bestand	Gemiddelde (ms)
map_04x04_1.txt	1,4
map_04x04_2.txt	5,4
map_04x04_4.txt	28,5
map_04x04_8.txt	45,8
map_08x08_1.txt	2
map_08x08_2.txt	12,1
map_08x08_4.txt	66,7
map_08x08_8.txt	1331,3
map_16x16_1.txt	10,4
map_16x16_2.txt	63,6
map_16x16_4.txt	517,9
map_16x16_8.txt	18163,8

Tabel 2 Tijdsmetingen in ms waarbij enkel de robots werden bewaard en de zetten werden gesorteerd

Aangezien er geen robots kunnen verdwijnen en elke robot 4 mogelijke richtingen uit kan, worden alle zetten die op een bord kunnen worden toegepast bij deze optimalisatie al in het begin bepaald. Dat geeft onderstaande tijdsmetingen als eindresultaat:

Naam bestand	Gemiddelde (ms)
map_04x04_1.txt	1
map_04x04_2.txt	3
map_04x04_4.txt	21
map_04x04_8.txt	36,4
map_08x08_1.txt	1
map_08x08_2.txt	8
map_08x08_4.txt	55,6
map_08x08_8.txt	1227,4
map_16x16_1.txt	8
map_16x16_2.txt	53,2
map_16x16_4.txt	454,3
map_16x16_8.txt	17272,6

Tabel 3 Tijdsmetingen in ms waarbij de zetten op voorhand konden worden bepaald

Uit de tijdsmetingen is zien dat het algoritme snel werkt voor kleine borden en borden waarbij niet veel robots worden opgesteld.

Genereren puzzels

In tegenstelling tot het originele spel waar de muren een vaste plaats hebben, was het nu de bedoeling dat het bord willekeurig opgemaakt werd. Niet elk volledig willekeurig bord heeft een oplossing. Het is namelijk mogelijk dat het bord in twee verdeeld is door een lange muur met robot (▣) in sectie 1 en doel (◎) in sectie 2.

Het bord wordt gegenereerd door eerst de buiten muren te leggen. Daarna wordt er een willekeurige plaats gekozen op het bord om het doel te bepalen. Eens de plaats van het doel gekozen is, moet de startplaats van de robot met index 0 (▣) ook nog worden bepaald. Dat gebeurt aan de hand van het onderstaande algoritme.

Startend van de coördinaten van het doel wordt er willekeurig een richting gekozen en een muur op de tegenovergestelde plaats gezet. Dan wordt er een willekeurig getal gekozen tussen de coördinaat en de muur in de richting dat net gekozen is. Deze twee stappen worden dan enkele keren herhaald en op de finale plaats wordt dan robot index 0(▣) geplaatst.

Omdat het mogelijk is dat er toch een muur geplaatst wordt tussen twee andere muren en daardoor het bord nog altijd onoplosbaar is, wordt ook het genomen pad bijgehouden, en kan een coördinaat op het pad geen tussenstop meer zijn. Als er geen andere mogelijke plaatsen meer mogelijk zijn, wordt er gewoon een robot (▣) geplaatst op die coördinaat.

Als de robot (▣) eenmaal geplaatst is, worden er op alle andere coördinaten die niet in het pad zitten een muur geplaatst. Het overloopt elk vakje dat niet deelneemt aan het paden een nummer in het interval [0,10) bepaalt welke muur er geplaatst wordt. 0 is een muur links, 1 is een muur rechts, 2 is een muur boven, 3 is een muur beneden en vanaf 4 t.e.m. 9 is er geen muur. Deze muren zorgen dat het misschien makkelijker is, maar zeker niet moeilijker.

Als laatste worden de andere robots toegevoegd op een willekeurige plaats. Hierdoor bestaat de kans dat het makkelijker wordt maar ook moeilijker, als er een robot op het pad is, kan het zijn dat die eerst verplaatst zal moeten worden.

Door dit allemaal toe te passen is het mogelijk om toch een willekeurig bord te maken dat oplosbaar is.

Conclusie

Het doel van het project was om naast een command-line interface voor het spel Robot Reboot ook een optimale oplossing kan geven en willekeurige borden kan maken. In de command-line interface ziet het bord er intuïtief uit, de groene robot moet naar het groene doel. Het oplosalgoritme werk snel, maar een volgende optimalisatie zou zijn: na elke zet te kijken of het bord opgelost is en zo een laag in de zoekboom vroeger stoppen. Het maken van willekeurige oplosbare borden is niet heel simpel, maar doormiddel van het algoritme is het altijd oplosbaar.