Robot Reboot

Elias Nijs
Logisch Programeren, Universiteit Gent

8 May 2023

Robot Reboot

Elias Nijs

Logisch Programeren, Universiteit Gent

1. Introductie

In dit verslag bespreek ik mijn implementatie voor het spel robot reboot. Deze implementatie bevat drie modi. Ten eerste een interactieve modus, ten tweede een modus die de optimale oplossing zoekt voor een gegeven puzzel en tot slot een modus die een nieuwe puzzel genereert op basis van het aantal robots en dimensies van het bord.

We bespreken eerste enkele datastucturen die doorheen heel het programma voorkomen, daarna bekijken we elke van de drie modi en tot slot trekken we enkele conclusies.

2. Belangrijke Datastructuren

We bespreken eerst enkele datastructuren. De belangrijkste structuur die doorheen heel de implementatie telkens terugkomt is deze van de puzzel.

Deze ziet er als volgt uit:

```
puzzle(
     board(W, H, [wall(Dv, P)]),
     [robot(ID, Dv, P)],
     target(P)
     )
```

Er werd gekozen om bij het bord niet het hele bord bij te houden maar enkel de muren. Deze keuze werd gemaakt omdat het de logica signficant kan vereenvoudigen. Verder werd er ook gekozen om de character representaties van de muren en robots bij te houden. Ook deze keuze werd gemaakt om de logica van het programma te kunnen vereenvoudigen.

De data hierbij wordt voorgesteld door de variabelen. Hier stellen `W` en `H` respectievelijk de breedte en hoogte voor. Verder stelt Dv (Display value) de character representatie voor, en is P (Position) telkens een `vec2(X,Y)` die de positie van de muur of entiteit aanduidt. De `vec2(X,Y)` wordt doorheen het programma vaak gebruikt om posities en richtingen weer te geven.

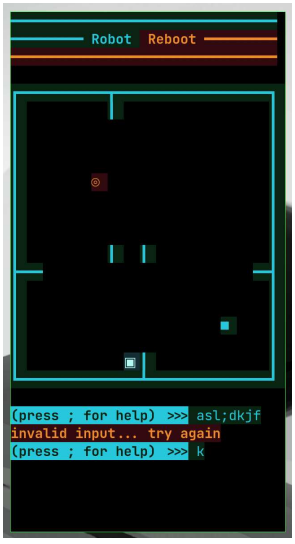
Andere predicaten zijn vanzelfsprekend of worden besproken bij hun modus indien deze enkel daar van toepassing zijn.

3. De drie modi

We bekijken nu elk van de drie modi in meer detail.

3.1. De interactive/spel modus

We beginnen bij de interactieve/spel modus. In deze modus kan de gebruiker aan de hand van de terminal het spel spelen. De code hiervoor kan men terugvinden onder `src/modes/play.pl`.



Er zijn een aantal opmerkelijke zaken aan deze modus. Deze zijn de volgende:

1. De spel loop.
2. Het session object.

3. Het gebruik van ansi escape sequences om het spel aantrekkelijker te maken.
4. Verbreeding van de weergave
5. De win/lose schermen

De spel loop is een loop die loopt tot het doel bereikt wordt of tot de speler het programma verlaat. Elke iteratie van de loop gaat als volgt. Eerst wordt de huidige puzzel afgebeeld en daarna wordt er input gevraagd aan de speler. Eens de input geleverd is wordt deze afgehandeld en belanden we zo in de volgende iteratie. De input kan een van volgende zaken zijn:

Actie	Input
Selecteer vorige robot	F
Selecteer volgende robot	D
Ga naar links	H
Ga omlaag	J
Ga omhoog	K
Ga naar rechts	L
Verlaat het spel	Q
Toon het help menu	;

Het session object bij de interactieve modus is een datastructuur die in de meeste geralteerde predicaten gebruikt wordt. Dit object ziet er als volgt uit:

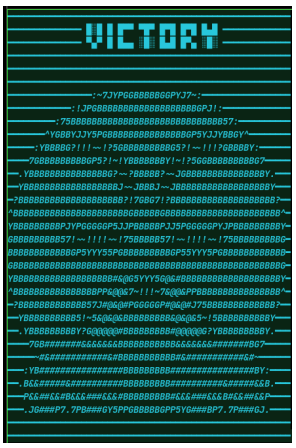
```
session( puzzle(...), ID, H )
```

Het puzzel object hierbij werd eerder besproken, `ID` is de id van de huidig geselecteerde robot en `H` is de geschiedenis van de zetten in string-vorm.

Een ander opmerkelijke punt is het gebruik van de ansi-escape characters. Deze laten ons toe om kleur en andere effecten voor onze terminal output te specifieren. Om hier goed van gebruik te kunnen maken, werden er predicaten gedefinieerd. Deze zijn terug te vinden in `src/utils/ansi.pl`. Er werd verder ook een kleuren palet gedefinieerd, dit valt terug te vinden in `src/config.pl`.

Verder werd de weergave van de puzzel ook verbreed. Dit wordt gedaan door aan de rechterkant van elke cel een spatie of extra muur toe te voegen tijdens de weergave. Op deze manier is het bord minder rechthoekig en ziet het er beter gebalanceerd uit. Intern wordt het bord wel nog in de normale dimensies opgeslaan.

Tot slot hebben we de win en verlies schermen. Deze werden toegevoed om de speler een betere ervaring te geven en zien er als volgt uit:



3.2. De oplos modus

De oplos modus zal een optimale oplossing zoeken voor een gegeven puzzel. Het belangrijkste aspect aan deze modus is het gebruikte algoritme. Het algoritme dat in deze implementatie gebruikt werd is iterative deepening. Dit werd gekozen omwille van de interne werking van prolog. Prolog gebruikt intern immers Depth-First-Search. Door een maximum diepte mee te geven en deze incrementeel te verhogen kunnen we dus heel makkelijk een iterative-deepening algoritme implementeren. De code hiervoor kan men terugvinden onder `src/modes/solve.pl`.

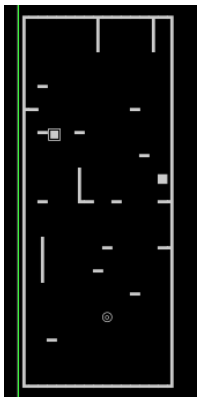
De effectieve implementatie bestaat uit drie delen: Het iterative deepening deel, een dfs deel en een logging deel. Het iteratieve deel zal steeds het dfs deel oproepen met een maximum diepte. Dit zal blijven gebeuren, met een incrementerede diepte, tot een oplossing gevonden wordt. Het dfs deel bestaat voornamlijk uit de oplossingsruimte

snoeien en de bewegingen uit voeren. De effectieve uitvoering van dfs is intern in prolog ingebouwd. Verder maken we, om dit te versnellen, gebruik van de `table` directive voor het dfs deel. Deze directive zal er voor zorgen dat prolog intern aan memorisatie doet van voorgaande oplossingen waardoor een aanzienelijk deel van de bewerkingen kan weggelaten worden. Tot slot hebben we ook nog een logging deel. Deze wordt opgeroepen bij elke nieuwe diepte om de gebruiker op de hoogte te houden van hoe ver het algoritme zit. Daarnaast wordt deze ook opgeroepen op het einde om de optimale oplossingen naar de terminal uit te schrijven. Helaas moest de logging tijdens de uitvoering weggelaten worden om aan de testen te kunnen voldoen.

We kunnen bijvoorbeeld het volgende commando uitvoeren:

```
cat 'lv3.txt' |
swipl main.pl -- --solve
```

waarbij lv3 er als volgt uitziet:



We bekomen hiervoor de volgende oplossingen:

```
1L,1U,0U,1L,1D,1R,1U,
1L,1D,0D,0R,0U,0L,0D
```

3.3. De genereer modus

De genereer modus bestaat uit een loop. Deze loop bevat 2 delen: het genereer deel en het oplos deel. Deze loop zal blijven duren tot er een puzzel gegenereerd wordt die oplosbaar is in een maximum aantal stappen. De code hiervoor kan men terugvinden onder `src/modes/generate.pl`.

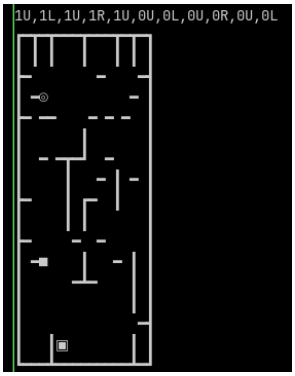
Het genereren van de puzzels wordt gedaan door, met bepaalde beperkingen in gedachte, willekeurig het aantal obstakels te bepalen, en willekeurige posities te genereren voor de obstakels en voor de robots. Eens de puzzel gegenereerd is zullen ook de display values van de muren opgelost worden door voor elke muur naar de buren te kijken en op basis hiervan de bijhorende waarde op te zoeken.

Het oplos deel bestaat erin de oplos modus, die hiervoor besproken werd, op te roepen op de nieuw gegenereerde puzzel. Indien deze een oplossing vindt binnen een bepaalde maximum diepte wordt de lus stop gezet en wordt de puzzel samen met het een optimale oplossing voor de puzzel naar de standaard uitvoer geschreven.

We kunnen bijvoorbeeld het volgende command uitvoeren:

```
swipl main.pl -- --gen=[2,8,8]
```

Dit geeft ons bijvoorbeeld het volgende resultaat:



4. Conclusie

We hebben een robot reboot implementatie succesvol gemaakt. We hebben ondervonden dat prolog een interessante taal is die ondanks zijn oude leeftijd nog hard in ontwikkeling is. In het algemeen kunnen we besluiten dat het kennen en kunnen van deze taal zeker een meerwaarde is. In het vervolg kan beter op voorhand plannen hoe bepaalde predicaten zich tegenover elkaar verhouden een meerwaarde zijn. Op deze manier kan de logica kleiner en simpeler blijven.