Universiteit van Amsterdam

Schrijven eerste technische rapport

Back in Bash

18 februari 2018

Tutor:
W.R.J. Vermeulen

Practicumgroep:
A1 - ALGOL

Cursus:
Programmeertalen

Student:
R.A.J. Wacanno
11741163

1 Introductie

Voor een opdracht van het vak Programmeertalen diende het spel *Back* in de programmeertaal *Bash* geïmplementeerd te worden. Zaak was echter, dat er enkel ingebouwde functie van *Bash* werden gebruikt en geen externe toepassingen. De vraag is dus: "Kun je zonder hulp van externe programma's het spel *Back* implementeren in *Bash*"

1.1 Het spel *Back*

Voor het implementeren van een spel is het uiteraard verstandig de werking en regels te kennen; hetgeen voor een simpel spel als *Back* geen grote opgave is. In deze werking zijn 3 hoofdonderdelen te ontdekken: de missie, de zet en mogelijke extra functies.

1.1.1 De missie

Het belangrijkste doel — tevens de manier waarop je het spel voltooid — is het voltooien van een missie. In de opbouw van deze missie schuilt dan ook het spelelement van *Back*. Het is als speler namelijk de bedoeling dat je een gegeven beginreeks, bestaande uit de letters `a`, `b` en `c`, omvormt tot een gegeven eindreeks. Missies bestaan vaak uit meerdere reeksen en hierbij is het dan aan de speler om in een bepaalde volgorde van reeks naar reeks te gaan om zo tot de laatste reeks van de missie te komen. Als de speler hierin slaagt heeft hij/zij het spel gewonnen.

1.1.2 De zet

Dan is de vraag: "Hoe komt een speler van de ene reeks naar de andere?" Dit gebeurd met zogenaamde zetten. Hierbij is het aan de speler een *adres* in te voeren waarmee op basis waarvan de huidige reeks vervormd wordt.

Adres
Het adres dat door de gebruiker opgegeven wordt, dient als selectie voor subreeks van dezelfde aaneenvolgende letters (bijvoorbeeld `a`, `cc` of `bbbb`). Een gegeven adres bestaat, net als de reeks, uit de letters `a`, `b` en `c`.
Hoe een adres een subreeks kan selecteren is te illustreren aan de hand van een voorbeeld. Gegeven is een reeks `abcbbca` en het volgende adres `bba` wordt ingevoerd. In dit geval zou de selectie er schematisch als volgt uitzien:

```
abcbbca
 b b  a
```

Met het opgegeven adres wordt dus de laatste `a` in de reeks geselecteerd. Op basis van dit schema kan ook de opbouw van het adres uitgelegd worden. Iedere letter in het adres dient als een ankerpunt en vanaf ieder ankerpunt wordt gezocht naar een volgende. De eerste `b` in het voorbeeld adres plaatst dus een anker op de eerstvoorkomende reeks `b`'en. Bij de eerste letter in het adres wordt altijd gezocht naar de eerstvoorkomende in de volledige reeks omdat er immers geen eerder anker is geplaatst. De daaropvolgende `b` zoekt vanaf het eerste anker naar de volgende reeks `b`'en en plaatst hier een anker. Als laatste zoekt de `a` vanaf het tweede anker naar de eerstvoorkomende reeks `a`'en (in dit geval is er maar eentje). Omdat dit echter de laatste letter in het adres is, wordt er geen anker geplaatst maar wordt deze `a` bestempeld als de uit het adres volgende selectie.

De geselecteerde subreeks wordt vervolgens uit de reeks gehaald, *geroteerd* en achteraan de reeks weer toegevoegd.

Rotatie
Rotatie klinkt misschien ietwat abstract, maar het principe is vrij eenvoudig. Bij rotatie worden de geselecteerde letters omgezet naar de volgende letters in het alfabet; *a* wordt *b* en *b* wordt *c*. In het geval dat een (reeks) `c`'en geselecteerd is zullen deze allen in een *a* worden veranderd. In het geval van de eerdergenoemde voorbeeld-zet zal dus de volgende transformatie plaatsvinden:

abcbbc**a**
⇓
abcbbc**b**

Kosten en budget
Iets waar tot op heden nog niet over gesproken is zijn de kosten en het budget. Het is namelijk niet zo dat een speler lukraak zetten kan maken zonder enige consequentie. Bij het uitvoeren van een missie begint een speler met een bepaald startbudget. Na iedere zet worden vervolgens de kosten van de adressering van het budget afgehaald. De kosten worden volgens de volgende formule berekend:

$$Kosten = 2^{\#letters\ in\ adres}$$

Bij het voorbeeldadres — bestaande uit 3 letters — zal er dus een bedrag van 8 van het budget worden afgeschreven. Indien het budget van de speler onder de 0 komt, heeft de speler verloren. Het budget kan tijdens het spelen worden opgehoogd door missie-reeksen te behalen. Door er in te slagen een doelreeks te bereiken behaalt de speler een beloning.

1.1.3 Extra functies

Onder extra functies kan men onder andere het mogelijk maken van het tussentijds opslaan van de vorderingen. Deze functies hebben echter geen invloed of spelprincipes van *Back*.

2 Methode

Bij het schrijven van het programma waren een aantal aspecten waar extra over de implementatie moest worden nagedacht. Deze aspecten dwingen teven het maken van bepaalde keuzes met betrekking tot implementatie af. Bij dergelijke keuzes is het van belang dat zij op basis van goede argumenten gekozen worden. Verkeerde keuzes kunnen in verder verloop van de implementatie immers tot problemen leiden. Hier volgen een paar van de belangrijkste aspecten die bij het implementeren van *Back* de revue hebben gepasseerd.

2.1 Wijze van selectie

Het in zekere opzichten belangrijkste, maar ook bij implementatie meest tijdrovende, onderdeel van *Back* was het selecteren van een subreeks. Na voldoende wikken en wegen is gekozen dit selecteren met behulp van reguliere expressies te doen. Dit om twee voorname redenen: het is makkelijk te implementeren en het is — naarmate reeksen en adressen langer worden — prima schaalbaar. Het gebruik van reguliere expressies is geïmplementeerd door een gegeven adres hiernaartoe om te vormen. Een adres als `bba` zal worden resulteren in de volgende reguliere expressie: `^([b]*)(b*)([b]*)(b*)([a]*))(a*)(.*)`. Er wordt dus een eerste blok voor de te selecteren subreeks gezocht. Deze bestaat uit de ankers al dan niet voorafgegaan door een aantal andere karakters. Hierna worden de daadwerkelijke subreeks en hetgeen erna komt geselecteerd. Deze wijze van selectie maakt het tevens erg gemakkelijke de reeks conform het adres te muteren.

2.2 Laden van missies

Bij het implementeren van het laden van missies kon er gekozen worden dit op twee manieren te doen. Je kan iedere ronde de benodigde eindreeks in te laden om deze vervolgens door het spel te laten gebruiken. Een andere manier is om aan het begin van het spel alle informatie uit het missie-bestand te laden en deze vervolgens in een *array* op te slaan. Voor dit laatste is uiteindelijk gekozen en hoewel dit in de eerste instantie niet veel uitmaakte, bleek dit later zeer waardevol te zijn.

2.3 Opslaan van voortgang

De keuze om de missie-gegevens allemaal tegelijk in te laden maakte het implementeren van een opslagfunctie voor het opslaan van de huidige staat van het spel erg eenvoudig. Het enige wat aan informatie opgeslagen diende te worden waren de huidige reeks, het huidige budget en de index van de huidige eindreeks in het *array* waarin deze is opgeslagen. Dit laatste was mogelijk door door de bij het laden gemaakte keuze. Het spel kon namelijk vanaf iedere eindreeks beginnen omdat deze allemaal reeds geladen waren en enkel uit het *array* gehaald dienden te worden.

3 Resultaten

Hier is een schermafbeelding van het uiteindelijke programma:

```
----------------------------------------
Target string:  abababababa
Current string: abcabcabc
Budget:         50
----------------------------------------
ccc

----------------------------------------
Target string:  abababababa
Current string: abcabcaba
Budget:         42
----------------------------------------
help
-----------------HELP--------------------
exit    - Exits game without saving
help    - Displays this info
save    - Saves current mission and quits
restart - Restarts current mission
          (from save)
----------------------------------------
save
Your progress was succesfully saved, see you later
```

4 Discussie

Al met al blijkt dat het dus mogelijk te zijn het spel *Back* in *Bash* te implementeren. Hierbij moet echter wel met bepaalde aspecten rekening worden gehouden, om zo implementatie eenvoudiger te maken.