Universiteit van Amsterdam

Technisch Rapport Bash-opdracht

Back to the Future

Bash-scriptend de toekomst in

28 februari 2018

Student:
Sam van Kampen
11874716

Tutor:
R. Klusman

Practicumgroep:
C1

Cursus:
Programmeertalen

1 Introductie

In dit technisch rapport wordt gekeken naar de structuur van het spel Back, en een implementatie van het spel in de programmeertaal Bash. Eerst zal het spel in detail besproken worden, en daarna zullen implementatiekeuzes worden verklaard, en zal gekeken worden of Bash een goede keus is als implementatietaal.

2 *Back* in detail

Kort samengevat draait *Back* om het omzetten van een start- in een eindkarakterreeks. Tussen de start- en eindreeksen kunnen tussenreeksen zitten. De startreeks moet dan via de tussenreeksen in de eindreeks getransformeerd worden. De karakterreeksen bestaan uit de letters `a`, `b` en `c`, en zijn allemaal van dezelfde lengte.

Een speler kan zetten doen, en met behulp van een of meer zetten kan de startreeks in de eindreeks worden getransformeerd. Er is een budget waarbinnen de transformatie kan gebeuren, en elke zet kost budget. Als naar een tussenreeks getransformeerd is, kan het budget verhoogd worden met een bonus.

Als de beginreeks binnen het budget omgezet wordt in de eindreeks, wint de speler.

2.1 Operaties

Om de karakterreeksen om te zetten kan de speler één soort zet doen: de **knip-draai-plakoperatie**. Deze bestaat uit drie suboperaties: de **knipoperatie**, die een opeenvolgende reeks van hetzelfde karakter uit de karakterreeks knipt; de **draaioperatie**, die elk karakter in het uitgeknipte deel verschuift in de cyclus $a \to b \to c \to a \to \ldots$ en de **plakoperatie**, die het gedraaide deel weer achteraan de karakterreeks plakt. Zie ter illustratie figuur 1.

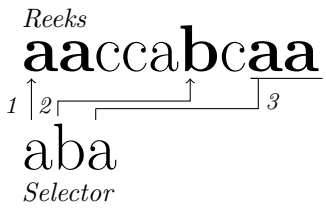
1. `aaccbb`
2. `aabb cc`
3. `aabb aa`
4. `aabbaa`

1, 2. Het deel `cc` wordt uitgeknipt.
2, 3. Het deel wordt gedraaid, waardoor de `c`'s omgezet worden in `a`'s.
4 Het deel wordt achteraan de reeks geplakt.

Figuur 1: De **knip-draai-plakoperatie** in actie.

2.2 Selectors

Om een deel uit de reeks te kiezen wordt een **selector** gebruikt. Het deel begint op de plek waar alle karakters in de selector gevonden zijn. Zie ter illustratie figuur 2. De selector bepaalt de kosten van een bepaalde zet volgens de formule $\texttt{cost}(\texttt{selector}) = 2^{|\texttt{selector}|}$.



1. Het eerste karakter van de selector is een `a`, dus als eerste wordt de `a` op positie 1 geselecteerd.
2. Het tweede karakter van de selector is een `b`, dus als tweede wordt de `b` op positie 6 geselecteerd.
3. Het derde karakter van de selector is weer een `a`. De selector selecteert dus de laatste reeks `a`'s.

Figuur 2: De **selector** in actie.

2.3 Missies, Savegames en Leaderboards

Een missie in *Back* bestaat uit een startreeks, een aantal tussenreeksen en een eindreeks, gecombineerd met het startbudget en per tussenreeks een bonus.

Omdat missies erg lang kunnen worden is het mogelijk het spel tussentijds op te slaan. Als er al een opgeslagen spel bestaat voor een bepaalde missie, wordt deze geladen.

Als een missie succesvol wordt afgerond wordt de score opgeslagen in een *leaderboard*-bestand. Deze bevat de vier hoogste scores voor de desbetreffende missie.

3 Methode

3.1 De interface

De gebruikersinterface van het programma is een *shell-esque loop* die op gebruikersinvoer wacht (hierna: de **invoerloop**). Het programma wordt aangestuurd met tekstcommando's. Met deze commando's kunnen zetten gedaan worden en kan het spel afgesloten, opgeslagen en herstart worden. De invoerloop houdt ook het huidige budget, de huidige reeks en doelreeks, en de huidige bonus bij.

Om de verschillende elementen van de interface van elkaar te scheiden is kleur gebruikt. Een neutrale kleur, blauw, is gebruikt voor de informatieve elementen van de interface. Minder belangrijke info wordt in grijs weergegeven, en gebruikersinvoer in roze. Zie hierbij ook figuur 3.

```
Back v0.1-dev
by Sam van Kampen, 2018
For help, type 'help'.

Loading mission file: missions/mission
Loaded.

Current string: abcabcabc (budget 50)
Target [1/7]: abababababa (+2)
> select c
Current string: ababcabca (budget 48)
Target [1/7]: abababababa (+2)
>
```

Figuur 3: De gebruikersinterface.

3.2 Operaties

Om een zet te maken wordt de functie `make_move` aangeroepen met de huidige reeks en een selector. Op basis van de

selector wordt bepaald wat het deel voor het knipsel, het knipsel zelf en het deel na het knipsel zijn. Deze drie delen worden uit de reeks gehaald, het knipsel wordt gedraaid en de drie delen worden in de juiste volgorde weer aan elkaar geplakt. De kosten van de selector worden ook berekend, en `make_move` geeft het resultaat en deze kosten terug. Deze kosten worden dan in de invoerloop van het budget afgehaald.

3.2.1 Knippen

Om een stuk te selecteren wordt de functie `select_part` gebruikt. `select_part` bestaat uit een *while-loop* die door de reeks heengaat en bijhoudt welke delen van de selector al gezien zijn. Als een karakter uit de selector gevonden wordt, wordt de positie en lengte van die reeks van dezelfde karakters bijgehouden. Deze worden ook teruggegeven aan het einde van de functie.

Als er aan het einde van de reeks nog ongevonden karakters in de selector zitten, selecteert de selector geen tekst. In dat geval wordt als positie en lengte `(0, 0)` teruggegeven. Ook wordt een foutmelding geprint die aan de gebruiker duidelijk maakt dat er niets geselecteerd is, en worden de kosten van de zet niet in rekening gebracht.

De knip-operatie zou ook met een *regular expression* kunnen worden geïmplementeerd, wat de code waarschijnlijk had verkort. Er is in dit geval voor een loopconstructie gekozen aangezien deze sneller is dan de equivalente regex, en er in het geval van een regeximplementatie dynamisch een regex gecreëerd moet worden, hetgeen extra complexiteit met zich meebrengt.

3.2.2 Draaien

Om een deelstring te draaien wordt de eigenschap gebruikt dat alle karakters in de deelstring hetzelfde zijn. Een van de karakters wordt dus gedraaid via de hierboven beschreven cyclus, en deze wordt dan n keer herhaald, waar n de lengte is van het uitgeknipte deel.

Een karakter wordt gedraaid door het karakter om te zetten in een decimale code, een modulo toe te passen, en de code weer om te zetten in een karakter. Zo kan het spel makkelijk aangepast worden om met een grotere cyclus, zoals het hele alfabet, te werken - in dit geval hoeft alleen de modulo aangepast te worden.

3.2.3 Plakken

De plakoperatie is simpel: in `make_move` wordt de reeks in drie delen gesplitst, en deze delen worden in de goede volgorde (het deel voor het knipsel, het deel na het knipsel en het gedraaide knipsel) weer aan elkaar geplakt met behulp van `echo`.

3.3 Missies

De eerste regel in een missiebestand bestaat uit een startreeks met bijbehorende kostenlimiet. De daaropvolgende regels bevatten mogelijke tussenreeksen en een bonus, en de laatste regel bevat de eindreeks.

Bij het inladen van een gegeven missiebestand worden de reeks-bonusparen na elkaar geladen met behulp van `read` en in een lijst opgeslagen, in het formaat `[reeks, bonus, reeks, bonus, ...]`. In de invoerloop wordt bijgehouden op welke plek in de missie-lijst het spel is. Ook wordt het huidige doel en diens bonus daar opgeslagen.

3.4 Savegames

Omdat de gehele missie-lijst beschikbaar is als het programma gedraaid wordt hoeven slechts de volgende gegevens opgeslagen te worden in een *savegame*: de huidige index in de missie-lijst, de huidige karakterreeks en het huidige budget. Met behulp van deze data en het missiebestand kan de gehele spelstaat weer gegenereerd worden. Deze gegevens worden dus ook in die volgorde, gescheiden met spaties, opgeslagen.

Een savegame kan gemaakt worden met behulp van het `save`-commando of door het programma door middel van `Ctrl-C` af te sluiten.

3.5 Leaderboards

Een leaderboard van een bepaalde missie bestaat uit de vier hoogste scores voor de missie. Deze scores worden opgeslagen als regels met eerst de naam van de persoon die een score heeft behaald, en daarna de score. Aangezien de naam spaties kan bevatten zijn deze regels niet makkelijk te ontleden met behulp van `read`, en is er gekozen om de regels te ontleden met behulp van *regular expressions*.

1. Gerard Grootman (score 12)
1. Ed Nieman (score 12)
3. Mensje van Keulen (score 10)

Figuur 4: Rangen op het leaderboard.

De leaderboards moeten gesorteerd opgeslagen worden. Omdat hiervoor het commando `sort` niet gebruikt mag worden, is gekozen om een zo simpel mogelijk sorteeralgoritme te implementeren in Bash, namelijk *Bubblesort*. Alhoewel dit niet een efficiënt sorteeralgoritme is maakt dit in de praktijk weinig uit, aangezien er maximaal vier scores in het leaderboard zijn opgeslagen.

Bij het printen van de leaderboards worden de scores een rang gegeven (zie figuur 4). Gelijke scores krijgen dezelfde rang toegewezen, en scores daarna krijgen de rang die ze zouden krijgen als scores niet gelijk waren.

4 Resultaten

Figuur 5: Enkele screenshots van Back.

5 Discussie

De opdracht was te implementeren in Bash, maar tijdens het implementeren stuitte ik wel meermaals op dingen die makkelijker te implementeren zouden zijn in andere talen. Het zou de code mijns inziens ten goede komen als het gebruik van `coreutils` (zoals `sed` en `grep`) minder restrictief zou zijn.

Mogelijke verbeteringen zijn het toevoegen van een `ncurses`-interface (alhoewel dit niet in pure Bash mogelijk is).