Opdracht 2: Dobbelspel

INTRODUCTIE

Dit is de tweede practicumopdracht van de cursus Functioneel programmeren. In deze opdracht wordt er een interactief programma ontwikkeld waarmee een populair dobbelspel kan worden gespeeld tegen een aantal computertegenstanders.

LEERDOELEN

Na het maken van deze opdracht, wordt verwacht dat u

- zelfstandig een groter programma kunt schrijven en dit programma op een inzichtelijke wijze kunt structureren
- een groter probleem kunt opsplitsen in deelproblemen en voor deze deelproblemen kleine hulpfuncties kunt ontwikkelen
- kunt omgaan met het *IO* type, de standaard *IO* functies en de do-notatie
- willekeurige waarden kunt opvragen met de `System.Random` bibliotheek
- datatypen en typesynoniemen kunt gebruiken voor het representeren van informatie, en een goede afweging kunt maken over welke representatie in een bepaalde situatie het meest geschikt is
- weloverwogen gebruik maakt van lijsten en tupels om programmeerproblemen aan te pakken.

Studeeraanwijzing

Aan deze opdracht kan worden begonnen nadat de eerste elf leereenheden van de cursus zijn bestudeerd. Er is geen startversie bij deze opdracht: verwacht wordt dat u zelf een nieuwe Haskell module maakt met daarin uw uitwerking. De opdracht geeft in grote lijnen aan wat er moet gebeuren: de details mogen naar eigen inzicht worden ingevuld.

Probeer de functies in de uitwerking simpel en compact te houden. Introduceer kleine hulpfuncties die het eigenlijke werk doen. De opdracht bestaat voor een deel uit interactieve input-output. Probeer zoveel mogelijk functies puur te houden, dat wil zeggen, een definitie te geven zonder de *IO* monad te gebruiken.

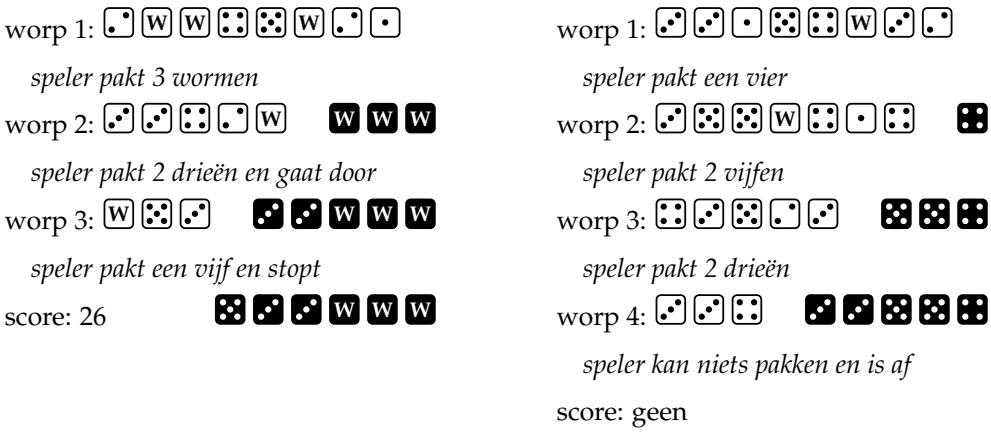
Mocht je vast komen te zitten, stel dan je vraag via yOUlearn of in een e-mail aan de begeleider. Deze opdracht heeft een studielast van ongeveer 12 uur.

Software

De opdracht is te maken met GHC. Voor de opdracht zijn geen extra bibliotheken nodig.

Inleveren

Lever uw uitwerking in via yOUlearn. Vermeld daarbij duidelijk uw naam, uw studentnummer en het versienummer van de opdracht. Uitwerkingen worden beoordeeld als voldoende of onvoldoende. Bij een onvoldoende wordt u door de examinator geïnformeerd over de aanvullende inspanning die nodig is om het resultaat te compenseren.



FIGUUR 1.1 Twee voorbeelden van een beurt: de zwarte dobbelstenen staan voor gepakte dobbelstenen

Opdracht 2: Dobbelspel

In deze practicumopdracht moet een simpel, interactief dobbelspel worden geprogrammeerd. Het spel Regenwormen van 999 Games[1] kan met meerdere spelers worden gespeeld, waarbij de spelers om de beurt met acht dobbelstenen mogen gooien. Doel van het spel is om zo veel mogelijk punten te verzamelen. Deze punten staan op stenen die in de eigen beurt veroverd kunnen worden. We gebruiken hier spelregels die eenvoudiger zijn dan in het originele Regenwormen spel.

Let op!

De opdracht bestaat uit drie onderdelen. Als eerste wordt gekeken naar een beurt van een speler. Daarna moeten tactieken worden gedefinieerd om het spel te spelen, waaronder een interactieve speler die je zelf kunt bedienen en een eenvoudige computerspeler. Als laatste wordt het spelverloop behandeld.

1 Bepalen van de score in een beurt

Een beurt wordt gespeeld met acht dobbelstenen. Iedere dobbelsteen heeft de waarden 1 tot en met 5 en een speciaal 'worm' symbool dat ook waarde 5 heeft. In een beurt is het de bedoeling om een zo goed mogelijke score te halen. De score wordt berekend door de waarden van de gepakte dobbelstenen op te tellen. Een beurt bestaat uit de volgende stappen:

a De dobbelstenen worden gegooid. In het begin van de beurt zijn dit er acht.
b De speler kiest welke dobbelstenen opzij worden gelegd. Alle dobbelstenen (tenminste één) van het gekozen symbool worden nu gepakt. Hierbij geldt dat een symbool waarvan al eerder in de beurt dobbelstenen zijn gepakt niet nogmaals gekozen mag worden. Als de speler alleen maar symbolen gooit die al gepakt zijn dan is de beurt voorbij zonder een score.
c De speler heeft nu de keus om door te gaan (naar stap a) met de resterende dobbelstenen of om te stoppen. Een speler mag alleen stoppen als hij tenminste één dobbelsteen met een worm heeft gepakt en als de score van de tot dan toe gepakte dobbelstenen geldig is. Welke scores geldig zijn verandert tijdens het spel (maar een score moet altijd hoger dan 20 zijn).

In Figuur 1.1 staan twee voorbeelden van beurten. Aan de linkerkant had de speler na worp 2 al mogen stoppen met een score van 21. Aan de rechterkant is de speler steeds verplicht om door te gaan omdat er nog geen worm is gepakt.

Opdracht

Geef declaraties voor de basisdatatypes die nodig zijn voor het spelen van een beurt. Denk hierbij in ieder geval goed na over de representatie van een dobbelsteen.

[1] http://www.999games.nl/spel/regenwormen

Opdracht

Implementeer het werpen van de dobbelstenen. Gebruik hierbij de functie *randomRIO* uit de bibliotheek `System.Random` voor het opvragen van een random waarde. Deze functie heeft als type:

$$randomRIO :: Random\ a \Rightarrow (a,a) \rightarrow IO\ a$$

Zo zal de actie $randomRIO\ (0,10 :: Int) \gg\!= print$ steeds een willekeurig getal laten zien tussen 0 en 10. Het type *Int* is dus een instantie van de typeklasse *Random*. Hierbij moet dus ook nagedacht worden over de representatie van een worp (met meerdere dobbelstenen).

Opdracht

Of een score geldig is verandert in de loop van het spel. Definieer een predicaat ($Int \rightarrow Bool$) om de geldigheid van een score te kunnen beïnvloeden. Bereken ook de score van een worp.

Opdracht

Geef een implementatie van een beurt. Vraag de gebruiker om input iedere keer als er een keuze moet worden gemaakt.

Aanwijzingen

Enkele aanwijzingen voor deze deelopdracht:

- Bij het testen kan dan het predicaat (>20) worden gebruikt omdat een score altijd hoger dan 20 moet zijn.
- Hieronder staat een mogelijke uitvoer die hoort bij de linkerkant van Figuur 1.1, waarin de invoer is onderstreept. Je bent echter vrij om hiervan af te wijken.

```
Hugs> beurt (> 20)
worp: 2,W,W,4,5,W,2,1
   pakken? W
worp: 3,3,4,2,W    (W,W,W)
   pakken? 3
   doorgaan (score is 21)? j
worp: W,5,3    (3,3,W,W,W)
   pakken? 5
   doorgaan (score is 26)? n
score: 26
```

- Denk na over de manier waarop u de uitvoer weer wilt geven op het scherm!

2 De tactiek

In een beurt heeft een speler twee dingen te kiezen: welke dobbelstenen worden er gepakt en wordt er doorgegaan of gestopt (bij een geldige score)? Een tactiek voor het uitvoeren van een beurt kan dus op de volgende manier worden gerepresenteerd:

```
type Worp = [Steen]
type GepakteStenen = [Steen]
type Tactiek = ( GepakteStenen → Worp → IO Steen   -- pakken?
               , GepakteStenen → IO Bool           -- doorgaan?
               )
```

waarbij het type *Steen* voor de symbolen van een dobbelsteen staat. Bij pakken corresponderen de twee lijsten met respectievelijk de gegooide symbolen en de eerder gepakte symbolen. Bij doorgaan correspondeert de lijst met de gepakte symbolen. Beide functies hebben een *IO* type zodat er desgewenst gebruik gemaakt kan worden van input-output en random waarden in tactieken.

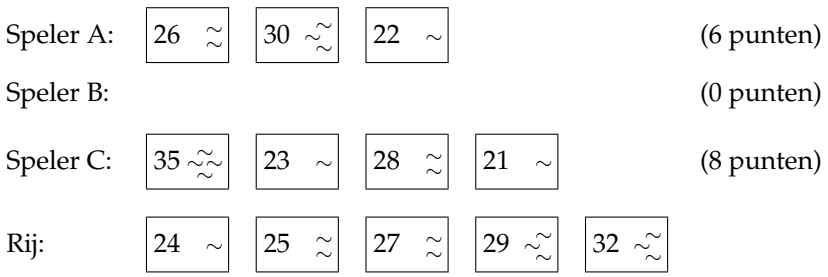
Opdracht

Implementeer tenminste twee tactieken: een *spelerTactiek* waarbij de gebruiker om invoer wordt gevraagd en een eenvoudige *computerTactiek* waarbij een keuze wordt gemaakt door de computer.

Aanwijzingen

Aanwijzingen voor deze deelopdracht:

- Ga er bij het definiëren van een tactiek vanuit dat er altijd een dobbelsteen kan worden gepakt. Zorg vervolgens dat hierop is gecontroleerd voordat de functie wordt aangeroepen.



FIGUUR 1.2 Voorbeeldsituatie tijdens het spel. De tegels 26 (bij speler A) en 35 (bij speler C) liggen open. De tegels 31, 33, 34 en 36 zijn al uit het spel genomen.

– Een simpele *computerTactiek* is om altijd het hoogste geldige symbool te kiezen en om te stoppen zodra dit kan.

Opdracht Breid de implementatie van een beurt zo uit dat deze gebruik maakt van een meegegeven tactiek.

3 Het spelverloop

We richten ons nu op het verloop van het dobbelspel en hoe het gespeeld wordt met twee of meer spelers (of computerspelers). Er zijn 16 tegels in het spel die zijn genummerd van 21 tot en met 36. Op iedere tegel is een aantal wormen afgebeeld. Iedere worm op een tegel is aan het einde van het spel een punt waard. Het aantal punten op de tegels is als volgt verdeeld:

tegels	21, 22, 23, 24	1 punt
	25, 26, 27, 28	2 punten
	29, 30, 31, 32	3 punten
	33, 34, 35, 36	4 punten

Opdracht Geef een declaratie voor het datatype *Tegel*.

In het begin van het spel worden alle 16 tegels open in een oplopende rij gelegd. Spelers proberen om de beurt deze tegels te bemachtigen en leggen deze op een stapel voor zich neer. Alleen de bovenste tegel van de stapel van een speler (de laatste die is gepakt) ligt open om afgepakt te worden door een tegenstander.

Opdracht Geef de declaraties die horen bij de oplopende rij, en voor een stapel van tegels.

Figuur 1.2 laat een voorbeeldsituatie zien. Afhankelijk van de score van een speler bij het dobbelen in zijn beurt gebeurt het volgende:
– De score komt *precies* overeen met een tegel in de rij. De speler pakt de tegel en legt deze bovenop zijn stapel.
Voorbeeld: Speler B heeft score 27 en pakt tegel 27 uit de rij.
– De score komt *precies* overeen met een open tegel van een tegenstander. De speler pakt de tegel en legt deze bovenop zijn stapel.
Voorbeeld: Speler B heeft score 26 en verovert daarmee de tegel van Speler A. Als gevolg hiervan komt bij A nu de tegel 30 open te liggen.
– In alle andere gevallen is de score ongeldig. Als de stapel van de speler leeg is verandert er niets.

Opdracht Implementeer functies die controleren op een score geldig is, en als de score geldig is zorgen dat de juiste tegel gepakt wordt. Houdt hierbij rekening met de hiervoor beschreven spelregels.

Het spel eindigt zodra de rij met tegels leeg is. De speler met de meeste punten in zijn stapel wint het spel.

Opdracht

Implementeer nu het volledige spel en zorg dat dit interactief gespeeld kan worden tegen tenminste 2 computertegenstanders. Maak hierbij gebruik van de functies die u hiervoor heeft geïmplementeerd. Het spel moet opgestart kunnen worden met de functie:

$$main :: IO\ ()$$

Leg (indien nodig) in een aanvullend document uit hoe de interactie werkt.

Aanwijzingen

Aanwijzingen voor deze deelopdracht:
- Denk goed na over een geschikte representatie voor de toestand tijdens het spel.
- Introduceer kleine hulpfuncties om het meeste werk te doen.
- Voorkom dat alle functies aan *IO* doen: de meeste hulpfuncties horen puur te zijn.
- Schrijf een functie die afhankelijk van de toestand bepaalt wat een geldige score in een beurt is.