# Verslag Functioneel Programmeren

## rpg-engine

Ben De Meurichy

## Inleiding

Dit project is een combinatie van een parser voor json-achtige configuratiebestanden en een bijhorende game-engine die deze geparste bestanden omzet naar een speelbare rpg-videogame.

Er wordt gebruik gemaakt van de Parsec monadic parser bibliotheek voor haskell die toelaat om parser functies aan te maken die strings omzet in haskell functies door middel van het gebruik van Monads, deze parsers zijn combineerbaar aan de hand van Monad transformers.

In dit verslag komen de volgende zaken kort aan bod:

- de opbouw van de engine
- het gebruik van monads en monad transformers in de code
- een overzicht van alle functionaliteit met voorbeeld
- de bespreking van de testen in het project
- conclusie (+ en - punten van de geschreven code)

## Architectuur engine

Er is gekozen om één grote parser te maken voor het hele bestand dat later de geparste text omzet in de correcte datatypes met record datatypes. Dit is achteraf gezien misschien minder efficiënt maar zorgt dat de fucnties voor het inlezen wel zeer kort blijven ipv als alle velden achter elkaar geparst zouden worden.

De code is opgesplitst in verschillende modules die elk hun eigen deel afhandelen.
De visualhandler zorgt voor het renderen van de game, de actionhandler handelt de verplichte functies uit de configuratiebestanden af, etc.

`HelperFunctions.hs & VisualHandler.hs` [1]
Tijdens het spelen van de game verschijnen in de rechterkant van het scherm enkel de acties die op dat moment uitvoerbaar zijn.
Deze worden gedetecteerd door te kijken of er in een van de velden rond het player character een item of entity te vinden is.
De acties die uitvoerbaar zijn worden gemapt op de nummertoets die overeenkomt met het getal voor de actie.

`ActionHandler.hs`

Doordat er gewerkt is met een bereik van een tegel horizontaal of verticaal rond de speler / entities / items was de leave functie lastig te implementeren.
Op dit moment verplaatst de speler 1 stap zich automatisch uit het bereik van alle items en entities rond hem indien mogelijk, dit zorgt ervoor dat als de speler op een tegel staat met een item/entity in deze niet beweegt omdat er dan 2 stappen nodig zijn om uit het bereik te geraken van deze item/entity, dit is een tekortkoming van de huidige code.

`GameLogic.hs`

Voor de implementatie van de vijanden is gekozen om deze hun beurt af te handelen in in de `update` van de `play` functie van de gloss applicatie.
Hier wordt ook gekeken of de speler op een `End` tegel staat en of deze dan naar het volgende level in het bestand moet gaan of dat er naar het eindscherm mag worden gegaan.
De vijanden houden een damagecounter bij in de game die elke update 1 verhoogt wordt en als deze modulo 60 gelijk is aan 0 zullen alle vijanden een keer de player aanvallen indien deze in hun bereik is.
Er wordt ook gecheckt of de vijand verwijdert moet worden als zijn levens onder 0 zakken. Dit wordt eerder bekeken dan het aanvallen van de vijanden dus normaal zouden vijanden die dood zijn verwijderd moeten worden voor ze zouden aanvallen wat dus moet zorgen dat de speler geen schade meer krijgt van vijanden die er niet meer zijn.

`ActionHandler.hs`

Voor het vinden van items en entities in een level is er normaal gezien gezorgd dat er meerdere items/entities kunnen voorkomen met dezelfde id.
Dit wordt bereikt door eerst alle entities/items in een lijst te steken aan de hand van id en deze wordt dan gefilterd of de items/entities wel in het bereik zijn van de speler.
Uit deze lijst wordt dan het eerste element genomen.
Dit zorgt ervoor dat het vinden van een item niet heel specifiek is als er 2 items/entities met hetzelfde id in het bereik zijn van de speler maar dit leek er niet echt toe te doen omdat de acties toch op beide zou uitgevoerd kunnen worden.

`GameLogic.hs & VisualHandler.hs`

Om te beslissen welke input er gebruikt kan worden wordt er een gamestatus bijgehouden die in de huidige game. Indien dit `Levelselection` is kan enkel `w` ,`s` en `enter` gebruikt worden om een level te kiezen met de selector, hierna wordt een level ingelezen en geinitialiseerd.
Daarbij wordt de status op `Playing` gezet wat het level laadt en alle input open zet die nodig is om het spel uit te spelen.
Als het spel uiteindelijk uitgespeeld geraakt verandert de status naar `Won` wat de input beperkt tot `enter` wat de speler terug stuurt naar `Levelselection`.

Voor het renderen van het juiste scherm wordt ook gebruik gemaakt van deze gamestatus.

## Gebruik van Monads en transformers

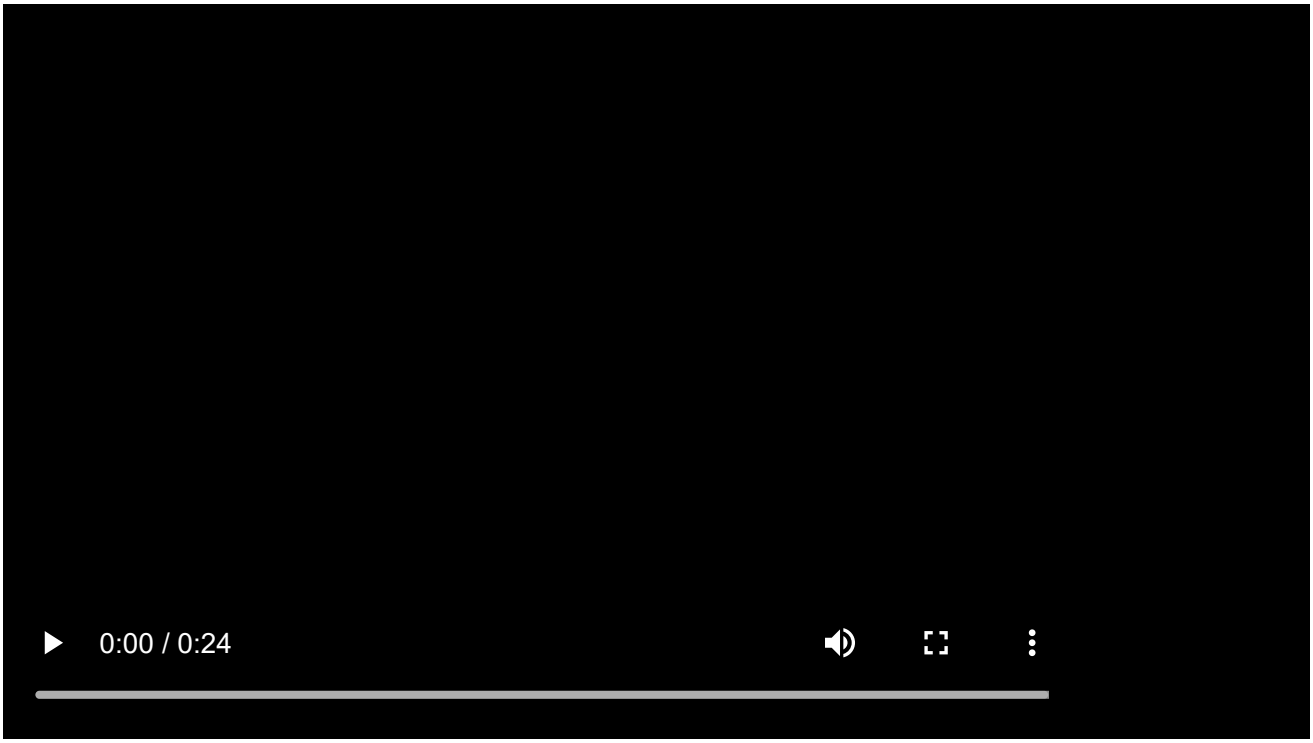
In deze engine zijn er Monads gebruikt voor het gebruik van de parsec bibliotheek.
Er is gebruik gemaakt van de `Parser` Monad uit de `Text.Parsec` module.
Deze monad dient om tekst om te zetten naar het gewenste datatype.
Dit was mogelijk door de functies uit de parsec bibliotheek zoals `sepby,char,many,...` .

Om te zorgen dat Deze Json-achtige string kan omgezet worden moeten de Parser Monads gecombineerd worden.

In het project is voor elke van de datastructucturen in `Datastructures.hs` een parser gemaakt tenzij deze enkel intern werd gebruikt.
Voor datatypes met een aantal verschillende waarden zoals richting is elke richting appart geparst.
Deze zijn dan achteraf gecombineerd via de monad transformer `<|>`. Deze is ook gebruikt om alle parsers aan elkaar te `kleven` zodat heel de json-String in 1 keer geparst kon worden.

## Overzicht functionaliteit met voorbeeld

*demo bestand is beschikbaar als demo-level.mp4 indien niet beschikbaar*




Zoals duidelijk is van de video wordt het aangepaste level ingeladen na het selecteren in het startscherm. Het speler karakter wordt op de juiste starttegel geplaatst.
De acties worden correct gedetecteerd indien de conditie waar is.
Er kunnen items opgenomen worden en deze kunnen worden gebruikt (de sleutel op de deur en zwaard op enemy,...)
De speler krijgt schade van de kat om de zoveel ticks en de kat verdwijnt als deze dood is, dit is te zien aan de hartjes beneden in het scherm.

Na de healthpotion te gebruiken komen er terug levens bij, als het drankje is opgebruikt verdwijnt deze uit de inventory.
Uiteindelijk stapt de speler op de eindtegel. Hierna is er nog een kort level om te laten zien dat er meerdere levels in 1 bestand kunnen voorkomen.
Uiteindelijk verschijnt het eindscherm nadat beide levels van het bestand zijn doorlopen.

## Bespreking testen

In de testen zijn er een aantal testen geschreven voor simpele zaken die mischien door een bepaalde fout later vreemde errors kunnen geven.

Het correct uitlezen van een bestand word nagekeken door er de waarde van de player hp uit op te vragen na het initialiseren van de game.
Er wordt getest of de startlocatie wordt gedetecteerd, hierbij moest er in de code nog -1 -1 gedaan worden op de teruggegeven coordinaten omdat de speler, items en entities op een ander coordinatensyteem zitten dan de layout (zie conclusie).

Er wordt gekeken of het player character items rond zich kan detecteren en of deze correct kan zien of hij kan rondbewegen of niet.

Als laatste wordt er ook getest of een stuk willekeurige JSON correct wordt omgezet.

## Conclusie

Dit project is zeker al een verbetering in het opsplitsen van code in modules volgens doel tegenover het patience project.
Er zijn veel meer kortere en minder complexe functies, maar dit kan hier en daar zeker nog beter.

Voor monads is enkel de parse bibliotheek gebruikt, ik wist jammer genoeg niet hoe ik `unsafePerformIO` kon vermijden dus dit heb ik wel nog een aantal keer gebruikt

Soms zijn er wel wat vreemde keuzes gemaakt die later problemen opleverden, zoals de items/ entities en speler die op een coordinatensysteem zitten dat de muren niet meeteld. De layout van een level zit echter op een coordinatensysteem waar de muren meetellen als vakje wat voor redelijk wat +1 en -1 zorgt in bepaalde functies.

Voor het renderen zijn vrij veel hardgecodeerde variabelen gebruikt omdat dit anders voor een gigantische hoeveelheid aan variabelen zou zorgen, dit zou misschien opgelost kunnen worden door mijn translaties en schalingen te berekenen aan de hand van de schermgrootte.
Dit is echter vrij ingewikkeld en niet echt doenbaar op de beperkte tijd die we nog hadden door de combinatie met andere projecten.

Gebruikte bronnen:
Voorbeeld voor JSON-parser: Adrians blog
Voorbeeld parsec bibliotheek: parsec basics
"Documentatie" parsec functies : Jake Wheat github
Bestanden inlezen in haskell : stackoverflowpagina

---

1. Bijhorende bestanden voor verduidelijking.↩