Verslag Haskell Project 3

Inleiding

De opdracht voor dit project was om een RPG_Maker te maken en hiervoor onze eigen parser te schrijven. Ik heb dit project opgesplitst in meerdere modules, waarvan de belangrijkste: ExtraScreens, Game_Logica, GameManager, Renderer, Datastructures en de Parser zijn. Ik ga deze allemaal individueel bespreken verder in het verslag. Ik ga ook mijn verschillende testen, mijn gebruik van monads , een voorbeeld level overlopen en vervolgens ga ik over tot de conclusie, maar eerst ga ik een woordje uitleg geven over de functionalteit van mijn project en hoe mijn spelregels werken.

De functionaliteit

Als je het spel start word je gegroet door een levelkeuze scherm. Je kan deze navigeren aan de hand van de pijltjestoetsen. Eens je op het te kiezen level staat druk je op enter en wordt dat bestand opgestart. Je karakter is een ridder die zich door de levels kan verplaatsen aan de hand van de pijltjestoetsen ,je kan je enkel bewegen op de vloer. De mogelijke acties die je kan uitvoeren worden bepaald aan de hand van de tegel waar je momenteel opstaat. Deze acties worden dan links op de papieren rol opgelijst in de vorm van de naam van de actie en tussen haakjes wat ze als argument nemen, zo kan je bijvoorbeeld twee keer dezelfde actie hebben maar met verschillende argumenten, als je bijvoorbeeld wilt aanvallen met een 'dagger' inplaats van een 'zwaard'. Ik heb ervoor gekozen niet de condities mee op te lijsten maar inplaats daarvan enkel de acties te tonen die je op dat moment effectief kan uitvoeren.

Tijdens je avontuur zul je langs alle kanten aangevallen worden door monsters die je staan op te wachten in de kerkers. Deze kunnen jou raken als je op de tegel naast of op dezelfde tegel als hun staat. Je kan de monsters terugslaan door op dezelfde tegel te gaan staan en de actie 'decreaseHp' uit te voeren, hiervoor gebruik je best het sterkste wapen in je bezit om de meest mogelijke schade toe te brengen. Mocht je helaas het loodje leggen tijdens je avontuur krijg je een game-over scherm te zien met de optie om het huidige bestand opnieuw op te starten. Soms kan het dat je niet aan de trap geraakt omdat je weg geblokkeerd wordt door een deur. Hier zul je dan eerst de bijbehorende sleutel voor moeten gaan zoeken zodat je deze kan openen. Als je er echter inslaagt alle levels uit te spelen krijg je dat begeerde overwinninscherm te zien met de optie om terug naar het bestand selectie scherm te gaan.

De implementatie

De DataStructures

De datastructuren die ik heb gebruikt spreken redelijk voor zich. Ik heb ervoor gekozen in mijn Game structuur alle levels uit het ingelezen bestand bij te houden, Een speler, alle mogelijke acties op dit moment, de status waar het spel zich momenteel in bevind, de postie van mijn selector voor het bestand keuze scherm en de naam van het bestand dat momenteel geopend is. Entity's en Item's houden gewoon dezelfde velden bij als ze in het level bestand zouden hebben. In de level structuur heb ik ervoor gekozen om de layout van het level, alle entities in het level, alle items in het level, de positie van de speler in het level en de positie van de uitgang bij te houden. De layout bestaat uit rijen van Tegels die ofwel een Muur, een Vloer, de Start, het Einde of Leeg zijn. De functie structuur die uit een ID (string) bestaat en een lijst van argument, een argument is ofwel een functie zelf ofwel een string. Dan hebben we nog de actie structuur die bestaat

uit condities en een functie, de condities zijn functies die voldaan moeten zijn vooraleer we de functie kunnen uitvoeren. Er resteert enkel nog het JSON datatype dat we verder zullen bespreken in de parser.

De parser

De parser is gemaakt met hulp van 'Adrian's blog', de link naar de gebruikte hulppagina vind je ook in mijn code. Mijn parser maakt gebruik van de parsec bibliotheek. Ik heb een datastructuur JSON die alle mogelijk te parsen datatypes bevat. Voor deze datastructuren dan ook effectief te parsen maak ik gebruik van de module Text.Parsec.String die mij toelaat een 'Parser JSON' aan te maken. Deze bestaat dan uit een combinatie van meerdere parsers die aan elkaar gekleefd zijn met behulp van de monad transformer '<|>'. Hier heb ik dan een parser voor elk mogelijk te verwerken datatype. Sommige van deze subparsers zijn simpel zoals 'parseNumber' die eigenlijk gewoon alle whitespace erafknipt en dan alle getallen op een ondoorbroken lijn neemt, maar dan heb je bijvoorbeeld ook de 'parseActions' die redelijk ingewikkeld wordt. Deze gaat elke action apart verwerken in een andere Parser JSON, namelijk 'parseAction' waar dan de condties en functie gesplitst worden en in de functie 'parseFunction' verwerkt worden. Deze 'parseFunction' maakt dan een functie aan met een ID en parsed de argumenten verder in de 'parseArg' functie. De argumenten zijn ofwel een string ofwel een functie, deze worden dan respectievelijk weer verder verwerkt tot alle sub-argumenten ofwel een string zijn ofwel leeg.

De GameManager

De spel manager zorgt ervoor dat de juiste schermen vertoond worden afhankelijk van de status van het spel. We hebben vier statussen: Playing, Won, Select en Lost. De algemene 'handleInput' functie verwijst door naar de correct input handler voor onze huidge status. Mochten we momenteel in de Playing status van het spel zitten checken we hier ook elke keer of het tijd is voor onze vijanden om aan te vallen met behulp van een teller die elke stap 1 omhoog gaat, we checken hier ook als we aan het spelen zijn of onze speler dood is, zoja tonen veranderen we de spelstatus naar Lost en tonen we het game-over scherm. Na elke input roepen we ook steeds de functie 'nextGame' op die gaat checken of we ons momenteel op de uitgang bevinden en onze mogelijke acties ververst. Als we ons dan inderdaad bij de uitgang bevinden verwijderen we het huidige level uit de lijst met te spelen levels. Als dit het laatste level is veranderen we onze status naar Won en tonen we het overwinningsscherm. We hebben hier ook nog de functie 'parseFile' die een bestandsnaam als input neemt en vervolgens het spel start en configureerd op basis van dat bestand, hier verder meer over in de GameMaker.

De GameMaker

Deze module maakt ons spel aan met de functie 'startGame' die JSON als input neemt. Hier creeren we de levels en speler op basis van die JSON. Deze module spreekt eigenlijk voor zich, het zet de geparste JSON output van ons ingelezen bestand om naar de bruikbaardere datastructuren om die we eerder besproken.

De GameLogic

Deze module verwerkt alles dat onze GameManager ons opdraagt te doen. We voeren hier acties uit. De functie leave heb ik geimplementeerd door onze huidige tegel te verlaten in eender welke mogelijke richting. De functie retrieveItem pakt een item op de grond op en steekt dit in je inventory. De functie 'increasePlayerHp' vult onze levens bij met de waarde van het item dat als argument meegegeven word, dit item word dan ook gebruikt, dat wil zeggen dat het aantal te gebruiken keer van dat item met 1 verminderd word en als dit dan op 0 komt te staan word dit item uit de inventory verwijderd. De functie 'decreaseHp'

valt de entity aan die als argument meegegeven word en op dezelfde tegel als ons staat met het tweede argument dat wordt meegegeven. De schade die de entity krijgt hangt af van de waarde van het item, dit item word ook weer gebruikt.

We hebben in deze module ook de functie 'checkActions' die we eerder al vermeld hadden. Deze gaat al onze huidig mogelijke acties overlopen, dat wil zeggen alle acties die beschikbaar zijn waarvoor de condities ook voldaan zijn. We steken deze dan in het 'gameAction' veld van het meegegeven spel en geven dit terug.

De Renderer

De Renderer module zorgt ervoor dat alles wordt afgebeeld in het spelvenster. De hoofdmethode, render neemt een spel als input en geeft een afbeelding terug, gebaseerd op de huidige configuratie en status van het spel. Als de status Playing is beelden we het eerste level in onze 'levels' lijst van het spel af. Hier worden onze mogelijke acties links opgelijst op een papieren rol, onze inventory onderaan en onze levens worden afgebeeld in de vorm van een hartje rechtsbovenaan per x aantal levenspunten. Het speelveld word rechts afgebeeld aan de hand van de juiste afbeeldingen die worden opgeroepen door onze functie 'lookup' die een string als argument neemt, deze zoekt vervolgens in een map van afbeeldingen de afbeelding met dit argument als naam. Als de status Lost, Won of Selected was renderen we het bijbehorende scherm, deze bespreken we verder in het ExtraScreens deel.

ExtraScreens

De ExtraScreens module regelt alles dat te maken heeft met het Select, Overwinning- en Game-over scherm. In deze module zit de functie findLevelFiles die alle bestanden in de levelFolder die eindigen in een .txt extensie zal opzoeken. Op de resultaten van deze functie bepaal ik dus de te kiezen bestanden in het select scherm. Voor de rest spreekt deze module redelijk voor zich, ze is op dezelfde principes gebaseerd als de render module.

Gebruik van Monads en Transformers

Zoals al eerder besproken in het parser deel van dit verslag is er in mijn project gebruik gemaakt van de parsec bibliotheek die onder andere de monad Parser bevat. Parser wordt gebruikt om text naar verschillende datatypes om te zetten. Hiervoor combineer ik meerdere parsers dankzij de monad transformer '<|>' die meerdere parsers combineert.

De Testen

De testen die ik geschreven heb testen of de speler kan bewegen in de vier richtingen, soms mag hij zich hier niet bewegen want er staat een muur links en onder hem. Een tweede test test of de juiste acties verschijnen als ik op een item sta, in dit geval is het item een sleutel en zijn de acties 'retrieveItem' en 'leave'. Ik voer deze test twee keer uit, eerst met een lege inventory en dan met een volle, de tweede keer verschijnen er geen acties aangezien de condities niet voldaan zijn. Dan is er een derde test die test of mijn functie 'removeItem' correct werkt, ik geef hier een sleutel aan mee en check dan of deze uit het huidige level verwijderd wordt. De vierde test kijkt of mijn functie 'exitCheck' correct werkt, deze gaat kijken of de speler momenteel op een uitgang staat, zoja dan veranderd hij de spelstatus naar Won als dit het laatste level was, anders gaat hij door naar het volgende level. Ten slotte test ik of de entities op het spelbord mijn speler correct aanvallen als deze in hun radius komt.

Overlopen van een voorbeeld level

In de lib folder staat een video van een door mij opgesteld demo level waar de sleutel voor de deur die de uitgang blokkeerd bewaakt word door een duivel. Dit monster kan je zoals je ziet aanvallen met ofwel het zwaard in je inventory of de dagger. Pas wel op want hij kan jou al van verder aanvallen. In het midden van de strijd merk je dat je levens bijna op zijn dus je drinkt een levensbrouwsel waardoor je er terug bovenop komt. Na de heftige strijd pak je de sleutel op en als je nu naar de deur gaat krijg je de optie om deze open te doen. Je beweegt je naar de trap en krijgt het overwinningsscherm te zien.

Conclusie

Al bij al ben ik vrij tevreden met mijn project. Het enige dat ik jammer vind is dat ik een aantal keer de functie 'unsafePerformIO' heb moeten gebruiken wat niet ideaal is, helaas is het mij niet gelukt dit probleem te vermijden.