Project: game engine

Verslag door Thibaud Collyn

Inleiding

In dit verslag bespreek ik kort mijn implementatie van de game engine in haskell. Ik zal kort de architectuur en functionaliteit van het project toelichten en dit ook aan de hand van een voorbeeld verduidelijken. Ook bespreek ik de testen die ik voor het project geschreven heb en het gebruik van monads.

Architectuur

Ik heb bij dit project toch redelijk veel proberen letten op het verdelen van mijn code op een duidelijke manier aan de hand van modules. Dit doe ik i.p.v. alles in de Main file te schrijven en dan met commentaar blokken alles probeer te verdelen, zoals ik bij het vorige project gedaan heb. Ik licht hier enkele hoofddelen van het project toe en verduidelijk eventueel bepaalde zaken waar nodig.

Parsen & invullen van datastructuren

Toen ik de level files die meegegeven werden aan dit project voor het eerst zag moest ik meteen denken aan JSON files en heb op basis van die vaststelling geprobeerd mijn files als een soort JSON objecten te parsen.

Het parsen gebeurt logischerwijs in de Parser.hs file en maakt gebruik van het zelfgeschreven JSON datatype dat gedefinieerd is en Datastructuren.hs(waar tevens alle eigen datatypes in staan). Een van de hoofdprincipes in het JSON datatype is dat we werken met objecten (een lijst van pairs) en pairs(een ID en een JSON datatype) zijn. Op die manier kunnen we logisch aan de hand van id-value pairs structuren zoals Player en Level opslagen.

Het omzetten naar datatypes die in de engine zelf gebruikt worden gebeurt in de LevelLoader module, en meer specifiek door de initGame functie die gestructureerd alle elementen van een gegeven JSON datatype ontleed en invult in andere datatypes zoals Player, Entity, Item,… d.m.v. specifiekere init-functies.

Onderaan de Datastructures module staan ook nog hulpfuncties gedefinieerd die gebruikt worden om de letterlijke waarde van bepaalde zelfgeschreven datatypes op te halen. Deze functies worden doorheen heel het project gebruikt om bijvoorbeeld een Argument om te zetten naar een String als men zeker weet da teen bepaald argument als waarde zeker een id bevat.

Game renderen

Het renderen van het spel gebeurt grotendeels in de GameRenderer module. Deze module is redelijk van zelfsprekend, er wordt een game datatype meegegeven aan de renderGame functie die vervolgens het huidige level van het spel zal renderen samen met de GUI van het spel.

Voor de rest wordt het renderen van het start- en eindscherm in de module StartScreen behandeld maar ook dit is redelijk triviaal.

Beide modules maken gebruik van de TextureLoader module, hierin worden alle assets ingeladen in twee datastructuren(GameTextures en GuiTextures) d.m.v. de getGame- en getGuiTextures functies.

Game logica & functies

De implementatie van de ingebouwde engine functies en condities gebeurt in de Functions module. In deze module wordt er wel gebruik gemaakt van iets complexere functies zodat er bijna altijd een Game datatype meegegeven en ook teruggegeven wordt.

De input en tevens ook de visualisatie van het startscherm, het effectieve level en het eindscherm wordt bepaald door een record state en het Game datatype dat in het Main bestand wordt opgevraagd en zo de juiste 'render' en 'handleInput' methodes opgeroepen worden.

Tot slot wordt in de GameLogica module alle input behandeld. Er wordt gedetecteerd welke acties uitgevoerd kunnen worden, of de speler naar een bepaalde locatie kan bewegen en ook 'gameState' acties.

Ik zal kort het gameState deel (ln 77 - ln 104) wat verduidelijken. De gameState functie voert constant controles uit om het spel te updaten namelijk of de speler het einde van het level bereikt heeft, of de speler nog leeft en of de speler nog in het selectie scherm zit.

De enemyActions functie zorgt er simpelweg voor dat alle entities die dood zijn, van het spel verdwijnen, en als entities schade kunnen aanrichten op de speler die dit ook om de 60 ticks doen.

Gebruik van monads en transformers

Het gebruik van monads in dit project gebeurt de Parser module waar namelijk gebruik wordt gemaakt van de parsec bibliotheek en nog specifieker de Parser monad uit Text.Parsec. Met deze monad wordt tekst uit de level bestanden omgezet naar andere datatypes.

Waar algemeen geparst kon worden is dit ook gebeurt maar sommige specifiekere delen, zoals het parsen van de level layout wordt in apparte parse methodes gesplitst(ln 59 - ln 75 in Parser.hs) en dan d.m.v. de <|> monad transformer samengevoegd. Dit is bijvoorbeeld ook het geval bij het parsen van de richting van een entity.

Voorbeeld

Hier toon ik kort een voorbeeld level waarin alle functionaliteiten van het project in te zien zijn a.d.h.v. een filmpje. Het filmpje staat in de root van het project en noemt Voorbeeld game.mp4.

Hierin zien we duidelijk dat:

- level files met meerder 'deellevels' ondersteund zijn
- een speler objecten kan opnemen
- vijanden schade kunnen aanrichten op de speler
- de speler d.m.v. een potion levenspunten bij kan krijgen
- de speler een entiteit kan verslagen
- de speler de deur kan openen

Daarna zien we ook het eindscherm dat gedenderd wordt als alle deellevels uit het huidige spel uitgespeeld zijn en tot slot als het einde van het level bereikt is er terug naar het startscherm gegaan wordt en men dan een ander level kan selecteren.

Testen

Aangezien het parsen van level bestanden makkelijker is om visueel te testen heb ik vooral game logica en functies getest. Omwille van tijdsnood heb ik echter wel minder tijd kunnen spenderen aan specifieke testen te schrijven voor het project maar de testen die er zijn tonen de correctheid van de functies wel goed.

Overzicht van de testen:

- Testen op juistheid van condities
- Testen op juistheid van enkele complexere get action functies
- Testen op correct bewegen van de speler
- Testen op item detective en interactie

Conclusie

Op vlak van opsplitsing en verdeling van code is dit project zeker beter als het vorige. Er is veel meer gebruik gemaakt van modules en deze zijn ook duidelijk opgesplitst en gedocumenteerd.

Over het algemeen zijn de functies kort en duidelijk opgesplitst maar op bepaalde plekken zoals in de GameRenderer en Functions module zitten iets complexere functies die wel wat netter geïmplementeerd konden worden maar dit zijn wel de uitzonderingen.

Op vlak van renderen is er ook redelijk veel hardgecodeert, dit was vooral om tijd te besparen en ook aangezien dit heel veel extra variabelen met zich mee zou brengen.