Verslag: RPG-Engine door Lukas Barragan Torres

Een korte inleiding

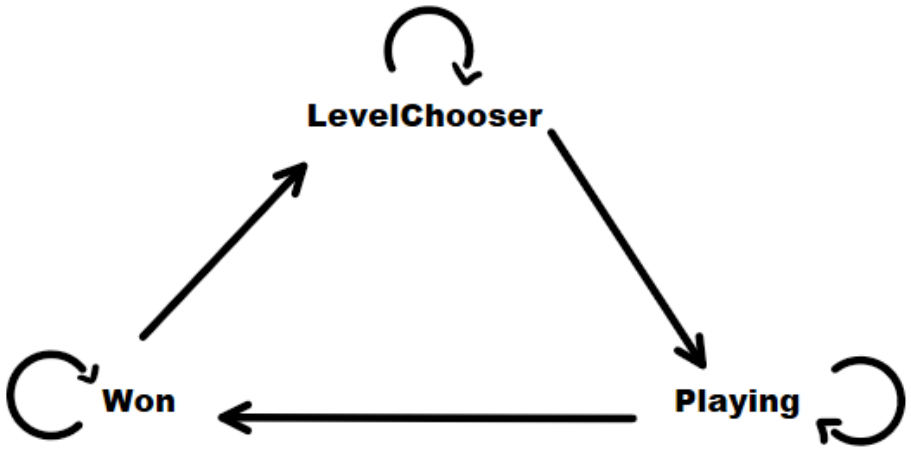
In dit verslag wordt de architectuur van de engine, de gebruikte monads en een voorbeeldlevel besproken. Ook wordt er een kort overzicht van de geschreven testen gegeven. **Ik heb de indexering gekozen waar de muren meetellen (levels/level2.txt voor de fout werd aangepast)**. De parser bibliotheek Parsec is gebruikt om de gegeven tekstbestanden te parsen, samen met `Map` uit de containers bibliotheek om een vertaling van naam naar picture bij te houden.

Architectuur van de engine

Op het hoogste niveau is het datatype `EngineState` gedefinieerd. Deze heeft drie constructoren:

1. LevelChooser
2. Playing
3. Won

De engine vloeit tussen deze drie staten op de volgende manier:



De `LevelChooser` constructor houdt een `LevelSelector` bij. Dit is een record met twee velden. Enerzijds een lijst met alle bestandsnamen die zich in de `levels` folder bevinden, anderzijds een getal die de positie van de selector voorstelt.

De `Playing` constructor neemt een `Game` als argument. Een `Game` houdt op zijn beurt een `Player` record en een lijst van `Level` records bij, samen met een backup van deze twee eerste velden die gebruikt wordt bij het herstellen van de Game wanneer de speler sterft. Het laatste veld dat de Game bijhoudt is een `PanelMode` (uitleg volgt). Verder zijn er nog `Entity` en `Item` records datatypes die op hun beurt instanties zijn van de `GameObject` klasse. Deze klasse bevat getters voor alle velden die een Entity en een Item gemeenschappelijk hebben.

De module die verantwoordelijk is voor de spelregels van het spel is de

`GameModule`. Hier zijn ook de functies geimplementeerd die door de engine ondersteunt worden. Deze module brengt de functies in de `PlayerModule` en `LevelsModule` samen tot één geheel.

Een veelvoorkomend patroon in de code zijn functies die beginnen met "on", bijvoorbeeld "onPlayer" of "onCurrectLevel". Dit zijn functies die een record uitpakken, een functie toepassen op een veld van dat record en het terug inpakken. Een gevolg van op deze manier te werken is dat code naar mijn mening leesbaarder wordt.

Het `Game` record heeft een veld `PanelMode`. Dit is op zijn beurt ook een record dat info bijhoudt over het actiepaneel dat verschijnt wanneer de speler zich naar een Item of Entity wil bewegen. De `PanelMode` heeft de volgende velden:

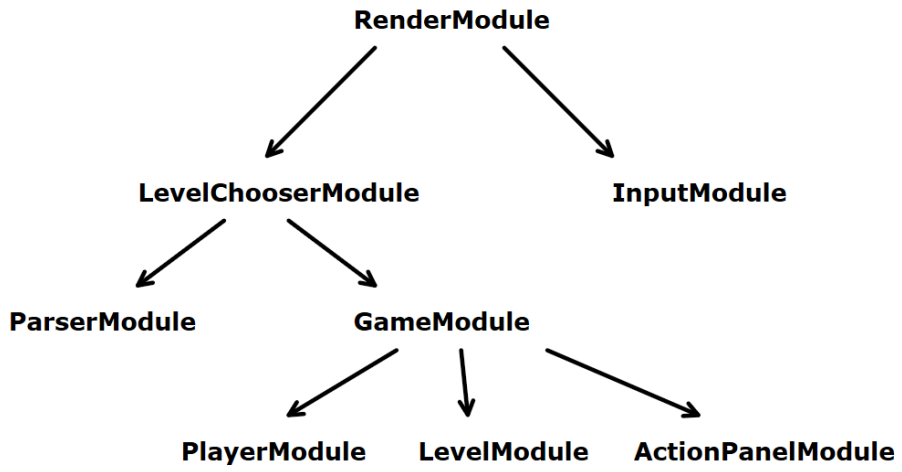
1. status (of de panel mode geactiveerd is of niet)
2. selectorPos (de index van de momenteel geselecteerde actie)
3. panelActions (alle acties in het paneel)
4. actionEntity (de entity waarop de actie uitgevoerd wordt, als die er is)

Er wordt tijdens het afhandelen van input gekeken naar de status van de panelmode. Als deze aan is, zullen de pijltjestoetsen de geselecteerde actie in het paneel kunnen veranderen. Als de modus uit is zullen de pijltjestoetsen de speler bewegen.

Acties in de game worden voorgesteld door het datatype `ConditionalAction`. Dit is een record dat een lijst van functies bevat (dit zijn de condities) en één enkele functie (dit is de actie die uitgevoerd kan worden).

Gebruikte modules

Er zijn 8 belangrijke modules gebruikt. Hun verhouding is opgesomt in volgende afbeelding:



Een pijl betekent "maakt gebruik van"

Bespreking monads en monad stacks

De gebruikte monad transformer is `ParsecT`. Deze is gebruikt in de `chooseLevelFile` functie gedefinieerd op lijn 36 in de `LevelChooserModule`. Dit zorgt ervoor dat het parsen van de geselecteerde file zonder case matching kan gebeuren binnen de IO monad. Verder is deze tranformer elke keer gebruikt wanneer twee parsers gecombineerd worden tot een nieuwe parser met een verschillend type.

Bespreking voorbeeldlevel

Ik zal `levels/level4.txt` bespreken. Dit kan u selecteren door in het laadscherm de selector naar onder te bewegen met behulp van de pijltjestoetsen.

De speler kan bewogen worden met de pijltjestoetsen. Beweeg naar de trap en ga zo naar stage 2.

In stage 2, wanneer de speler naar boven wil bewegen, verandert de status van `PanelMode` in de Game naar `On`. Nu kunnen de pijltjestoetsen gebruikt worden om een actie te selecteren. De sleutel in het spel kan gebruikt worden om de deur te openen.

In Stage drie kan op dezelfde manier het zwaard opgepakt worden. Wanneer de speler een entity wil slaan, zal de entity de speler terugslaan met een kracht gelijk aan zijn value. Nu een kleine kamikaze voor didaktische doeleinden: sla de entity met de dolk tot de speler doodgaat. Het volledige ingeladen spel wordt herstart (vanaf de backup die de `Game` bijhield). Wanneer het spel volledig wordt uitgespeeld komt de gebruiker op een winscherm terecht. Wanneer op de spatieknop gedrukt wordt, kan deze opnieuw een level file selecteren.

Overzicht testen

Testen zijn opgedeeld in twee delen. Het ene deel test de parser, het tweede deel de spelsemantiek op een test Game.

Een aantal testen voor de parser zijn bijvooreeld het parsen van:

1. Een key-value paar waarbij de value een id is
2. Een functie object met als argumenten een lijst van id's
3. Een functie object met als argument een andere functie
4. Een volledige actie
5. Een lijst

De semantische testen behandelen dan eerder dingen zoals het correct werken van:

1. `canMoveInDir`
2. `containsItem`
3. `hasNextLevel`
4. `inventoryContains`

Conclusie

Volgens mij heb ik hard ingezet op het leesbaar maken van de code. Ik denk ook dat ik mijn code duidelijk gedocumenteerd heb en dat de opsplitsing in modules logisch uit de opgave volgt. Graag had ik wat meer functionaliteit willen toevoegen, maar aankomende examens hebben ervoor gezorgd dat het bij de basis is gebleven. Naar mijn mening ziet mijn `RenderModule` er een beetje verwilderd uit met al die constanten, waardoor het daar misschien niet altijd duidelijk is wat er gebeurd. Al bij al was dit een tof project dat me veel heeft bijgeleerd over monads en monad transformers.