Programmeerproject 1: FireAnt

Verslag Fase 1

Erneste Richard Rwema Rugema
(erneste.richard.rwema.rugema@vub.be)
rolnummer: 0580198

Academiejaar 2021-2022

Inhoudsopgave

1 Intorductie

Dit document beschrijft het implementatie in fase 1 van het spel `Fire Ant` in scheme **r5rs**. Het maakt deel uit van het opleidingsonderdeel "Programmeerproject 1".

1.1 Spelbeschrijving

In dit spel is het de bedoeling dat een speler een *vuurmier* bestuurt, die tracht zijn mierenkoningin te redden van schorpioenenleger. Om dit te voltooien moet de speler een serie van levels, die toenemen in moeilijkheidsgraad, oplossen zodat hij tot bij de koningin geraakt. Een level bestaat uit een doolhof gevuld met puzzels, items en vijanden. Enkel door deze puzzels op te lossen en de schorpioenen te vermijden, kan de speler het level oplossen. Het spel eindigt dan tot dat alle levels opgelost zijn of tot dat de speler al zijn levens verloren heeft.

1.2 Implementatie

Voor het eerste fase zal het genoeg zijn om de elementaire elementen van het spel te implementeren.

Onder andere:

- Het spel maakt een spelwereld met een doolhof waarin een vuurmier, die door de speler kan bestuurt worden, kan rond lopen.
- Het doolhof bevat ook (een) schorpioen(en) die een vaste pad aflegt.
- De vuurmier moet bij aanraking van de schorpioen terug naar zijn start positie gaan.
- Verspreid doorheen het doolhof moeten er verschillenden eitjes geplaatst zijn, die vuurmier kan oprapen.

Het programma zal ook verdeelt worden uit een controller (zal tussen de modellen en views moeten communiceren) modellen (bezitten al de spellogica) en de views (zorgt voor het tekenlogica van het spel)

2 ADTs

In deze sectie zullen alle ADTs beschreven dat men verwacht te gebruiken, om de elementaire functionaliteiten in fase 1 van `Fire Ant` te implemente-

ren.

2.1 Game ADT

Het `Game ADT` is het centrale ADT det het spel op te start. Deze is verantwoordelijk om het initialisatie en onthouden van het juiste level en deze te updaten of te herstarten wanneer nodig. De `Game ADT` is ook verantwoordelijk om naar de invoer te luisteren en deze door te geven aan de `Level ADT`. Het luisteren zal gebeuren aan de hand van een spellus.

Naam	Signatuur
newGame	$(list \rightarrow Game)$
start!	$(\varnothing \rightarrow \varnothing)$
¡??¿	¡??¿

Tabel 2.1: Operaties van het `Game ADT`

Het `Game ADT` bevat de volgende operaties:

- **newGame**: Dit is de aanmaak operatie van het `Game ADT`. Voor objecten van `Game ADT` te maken, moet er een lijst meegegeven worden van strings. Deze strings zijn de namen van tekst bestanden die zullen gebruikt worden door het `Level ADT`.
- **start!**: Deze operatie start het spel op. Wanneer deze operatie uitgevoerd wordt zal er een venster aangemaakt worden. Ook de operatie om een nieuw ant en het corresponderende level worden aangeroepen in het spel lus van het spel.
- **¡??¿**: ¡??¿

2.2 Level ADT

Het `Level ADT` is het ADT verantwoordelijk om het level te vormen aan de hand van een tekst bestand en alles op zijn juiste plaats te zetten. Deze is verantwoordelijk om het level, characters en items te initialiseren. ¡??¿

Het `Level ADT` bevat de volgende operaties:

- **newLevel**: Dit is het aanmaak operatie van het `Level ADT`. Voor een object van het `Level ADT` aan te maken moet er een string en een object van het `Ant ADT` meegegeven worden. Het string geeft het

Naam	Signatuur
newLevel	(string Ant → Level)
update!	(Ant → ∅)
isFinished?	(∅ → boolean)
¡??¿	(¡??¿ → ¡??¿)

Tabel 2.2: Operaties van het `Level ADT`

naam van het bestand weer dat door het `Maze ADT`, die in het `Level ADT` zit, zal gebruikt worden. Het object van het `Ant ADT` is belangrijk voor het positioneren van de speler in het maze.¡??¿

- **update!**: Deze operatie is verantwoordelijk om alle objecten van het `Character ADT` en het `Item ADT`, die respectievelijk opgeslagen zijn in een lijst, te updaten. Het updaten van elk object gebeurt door hun respectievelijke ***update!*** operatie aan te roepen.
- **isFinished**: Deze operatie checkt of het level gedaan is of niet.
- **¡??¿**: ¡??¿

2.3 Position ADT

De `Position ADT` dit is een simpel ADT dat de positie in een level voorstelt. isFinished?

Naam	Signatuur
newPositie	(number number Maze → $Positie \cup \{\#f\}$)
move!	(number number → $\varnothing \cup \{\#f\}$)
eqPosistie?	(Positie → boolean)
¡??¿	(¡??¿ → ¡??¿)

Tabel 2.3: Operaties van het `Position ADT`

Het `Position ADT` bevat de volgende operaties:

- **newPositie**: Deze operatie maakt een nieuw object van het `Position ADT` aan. Die twee numbers opneemt dat respectievelijk de x- en y-coördinaat zijn in het meegegeven `Maze ADT`.
- **move!**: Verandert het positie van het `Position ADT` als dat een geldige positie is in het `Maze ADT`.

- **eqPosistie?**: Controleert dat het meegegeven Positie een gelijkwaardige positie heeft. Dat wil zeggen dat het x- en y-coördinaat een aan een gelijk zijn van elkaar.
- **¡??¿**: ¡??¿

2.4 Item ADT

Het `Item ADT` stelt elk voorwerp voor dat kan opgeraapt worden door een `Ant` (speler). ¡??¿

Naam	Signatuur
newItem	(Positie (any *rightarrow* any) → Item)
isCollected?	(∅ → boolean)
update!	(Ant → ∅)
¡??¿	(¡??¿ → ¡??¿)

Tabel 2.4: Operaties van het `Item ADT`

Het `Item ADT` bevat de volgende operaties:

- **newItem**: Maakt een nieuw object van het `Item ADT`. Het eerste argument dat de operatie neemt is een `Position ADT` dat een positie in het maze (doolhof) aangeeft. Vervolgens het tweede argument, een functie dat wordt toegepast bij het oprapen van het `Item ADT`.
- **isCollected?**: Check of het `Item ADT` is opgeraapt of niet.
- **update!**: Deze operatie checkt of het Ant object op dezelfde positie heeft als het item. Als dat het geval is wordt het te toepassen effect toegepast door het functie dat werd meegegeven aan **newItem**. Anders doet het operatie niks.

2.5 Character ADT

Het `Character ADT` moet een elke beweegbare character voorstellen in het spel. Deze is verantwoordelijk voor alle algemeen functies die een object van deze `ADT` bezit.

Het `Character ADT` bevat de volgende operaties:

- **newCharacter**: Deze operatie maakt een object van het `Character ADT` aan.

Naam	Signatuur
newCharacter	(Positie → Character)

Tabel 2.5: Operaties van het `Character ADT`

- **¡??¿**: ¡??¿

2.6 Maze ADT

Het `Maze ADT` is verantwoordelijk om het spellogica van het doolhof te construeren.

Naam	Signatuur
newMaze	(string → Maze)
isValidPosition?	(Positie → boolean)
¡??¿	(¡??¿ → ¡??¿)

Tabel 2.6: Operaties van het `Maze ADT`

Het `Maze ADT` bevat de volgende operaties:

- **newMaze**: Deze operatie maakt een object van het `Maze ADT` aan.
- **isValidPosition?**: Checkt als deze positie niet bezet is door een muur en dat het een bestaande positie is in het `Maze ADT`.
- **¡??¿**: ¡??¿

2.7 View_Maze ADT

Het `View_Maze ADT` is verantwoordelijk om het `Maze ADT` voor te stellen in teken logica gedeelte van het spel.

Naam	Signatuur
newView_Maze	(Maze → View_Maze)
¡??¿	(¡??¿ → ¡??¿)

Tabel 2.7: Operaties van het `View_Maze ADT`

Het `View_Maze ADT` bevat de volgende operaties:

- **++**: ¡++¿

- ¡++¿: ¡++¿

2.8 Scorpion ADT

¡++¿

Naam	Signatuur
¡++¿	(¡++¿ → ¡++¿)

Tabel 2.8: Operaties van het `Scorpion ADT`

Het `Scorpion ADT` bevat de volgende operaties:

- ¡++¿: ¡++¿
- ¡++¿: ¡++¿

2.9 View_Scorpion ADT

¡++¿

Naam	Signatuur
¡++¿	(¡++¿ → ¡++¿)

Tabel 2.9: Operaties van het `View_Scorpion ADT`

Het `View_Scorpion ADT` bevat de volgende operaties:

- ¡++¿: ¡++¿
- ¡++¿: ¡++¿

2.10 Ant ADT

¡++¿

Naam	Signatuur
¡++¿	(¡++¿ → ¡++¿)

Tabel 2.10: Operaties van het `Ant ADT`

Het `Ant ADT` bevat de volgende operaties:

- ¡++¿: ¡++¿
- ¡++¿: ¡++¿

2.11 View_Ant ADT

¡++¿

Naam	Signatuur
¡++¿	(¡++¿ → ¡++¿)

Tabel 2.11: Operaties van het `View_Ant ADT`

Het `View_Ant ADT` bevat de volgende operaties:

- ¡++¿: ¡++¿
- ¡++¿: ¡++¿

2.12 Eggs ADT

¡++¿

Naam	Signatuur
¡++¿	(¡++¿ → ¡++¿)

Tabel 2.12: Operaties van het `Eggs ADT`

Het `Eggs ADT` bevat de volgende operaties:

- ¡++¿: ¡++¿
- ¡++¿: ¡++¿

2.13 View_Eggs ADT

¡++¿

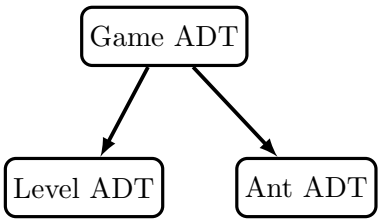
Naam	Signatuur
¡++¿	(¡++¿ → ¡++¿)

Tabel 2.13: Operaties van het `View_Eggs ADT`

Het `View_Eggs ADT` bevat de volgende operaties:

- ¡++¿: ¡++¿
- ¡++¿: ¡++¿

3 Afhankelijkheidsdiagram



4 Planning

...