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 en hun proceduren beschreven worden die in het programma voorkomen. Deze ADTs zullen verdeelt worden in 3 categorieën: Controller, Modellen en Views

2.1 Controller

De controller zal centraal zijn om de interacties tussen de modellen en views te beheren.

2.1.1 Game ADT

Het `Game ADT` is het centrale ADT dat het spel opstart en beheert. Deze is verantwoordelijk voor het initialiseren van de `Player ADT`, het initialiseren en starten van de juiste `Level ADT`, het verwerken van de speler invoer en het behandelen van het spellus.

Naam	Signatuur
new-game	(list → Game)
start	(∅ → ∅)
next-level!	(∅ → ∅)
key-handler	(symbol symbol → ∅)
game-loop	(number → ∅)

Tabel 2.1: Operaties van het `Game ADT`

Het `Game ADT` bevat de volgende operaties:

- **new-game**: Deze operatie maakt een object van het `Game ADT` aan.
- **start**: Start een nieuw spel op. Eerst slaagt het alle namen van de levels (dit zijn csv bestanden) uit de "level"map op, in een lijst. Deze lijst gebruikt het om de eerste `Level ADT` object te maken. Vervolgens maakt het een nieuwe object van het `Player ADT` en het `View ADT`. Tenslotte "start"het, de spellus en gebruikers invoer met behulp van de game-loop en key-handler procedures respectievelijk.
- **next-level!**: Veranderd de huidige level (current-level) naar de volgende level in de levels lijst. De oude level wordt ook verwijderd uit de levels lijst. Het update ook de huidige level van de `View ADT`.

- **key-handler**: Neemt twee argumenten op state en key. State is de status van de toets (key) op het huidige moment. Op dit moment ondersteunt de functie enkel de pijltjes toetsen ingedrukt. Wanneer dit gebeurt, beweegt het respectievelijk de speler als deze niet tegen een muur aanloopt.
- **game-loop**: Neemt een argument op ms (milliseconde voor vorige aanroep). Als de huidige level niet uitgespeeld is dan update het de huidige level en de view object. Daarnaast als de speler sterft, herleeft de level, de speler opnieuw. In het geval dat het huidige level uitgespeeld is voer het next-level procedure op.

2.2 Modellen

De Modellen zullen verantwoordelijk zijn voor het spellogica m.a.w. alle berekening achter de schermen.

2.3 Level ADT

Het `Level ADT` is het ADT dat een level opbouwt aan de hand van csv tekst bestand. Deze is verantwoordelijk om een csv bestand te lezen en zijn elementen ervan te interpreteren, het onthouden en aanmaken van alle nodige objecten in de level (bv. schorpioenen, eiren, het doolhof, etc.), de speler te laten herstarten, het updaten van de elementen in de level, het onthouden van de startpositie (spawn) en te checken of het level is uitgespeeld.

Het `Level ADT` bevat de volgende operaties:

- **new-level**: Neemt een `Level ADT` en de naam van level csv bestand op als argument op. Deze procedure geeft een object van het type level terug. Deze level object bezit alle elementen die het level bezit.
- **init**: Deze procedure initialiseert het level door de element van het csv bestaand een na een door te geven aan het interpret! procedure met de x en y positie van de element in de csv bestand.
- **get-maze**: Geeft de `Maze ADT` object terug opgeslagen in de `Level ADT`.
- **get-eggs**: Geeft een lijst van `Egg ADT` objecten terug die opgeslagen zijn in de `Level ADT`.

Naam	Signatuur
new-level	(Player string → Level)
init	(string → ∅)
get-maze	(∅ → Maze)
get-eggs	(∅ → list)
get-scorpions	(∅ → list)
get-updates	(∅ → list)
is-finished?	(Player → boolean)
is-legal-move?	(Object[1] symbol → boolean)
update!	(∅ → ∅)
respawn	(∅ → ∅)
on-collision	((Object[1] → ∅) → ∅)
interpret!	(string number number → ∅)

Tabel 2.2: Operaties van het `Level ADT`

- **get-scorpions**: Geeft een lijst van `Scorpion ADT` objecten terug die opgeslagen zijn in de `Level ADT`.
- **get-updates**: Geeft een lijst van objecten terug die in de huidige iteratie van de spellus geüpdatet zijn.
- **is-finished?**: Geeft aan of het spel is uitgespeeld door te kijken of de `Player ADT` helemaal onderaan de doolhof is geraakt.
- **is-legal-move?**: Neemt een Object (dat een `Position ADT` bezit) en een direction ('up, 'down, 'left or 'right) als argument op. Het zoekt met met behulp van de positie van het Object naar zijn buur positie gelegen op de "direction". Tenslotte kijkt de functie of de gebuurt positie een muur is aan de hand van de `Maze ADT`.
- **update!**: De procedure update de "updates" lijst door eerst de nodige objecten te updaten (en dus ook toe te voegen aan de lijst). Vervolgens checkt het of er objecten zijn die botsen met het Player object en zo ja, voeren ze hun eigen procedure uit (bv. schorpioenen vermoorden de speler en eiren worden genomen/verdwijnen). Deze objecten worden dan ook toegevoegd aan de lijst.
- **respawn**: Deze procedure plaatst de Player terug op zijn start positie en geeft hem terug "leven".
- **on-collision**: Neemt een procedure (die wordt uitgevoerd als er een

botsing met de Player gebeurt) en een lijst van objecten die gecheckt moeten worden voor botsingen met de Player.

- **interpret!**: Deze procedure neemt een string (dat een element uit de csv bestand is) , een x en y positie (dat de positie van de string voorstelt in de csv bestand). Deze warden worden gebruikt om Objecten (zoals een Egg, Scorpion of Start Positie) maken of de `Maze ADT` aan te vullen in het `Level ADT`.

2.4 Maze ADT

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

Naam	Signatuur
new-maze	($\varnothing \rightarrow$ Maze)
get-type	($\varnothing \rightarrow$ symbol)
get-height	($\varnothing \rightarrow$ number)
get-width	($\varnothing \rightarrow$ number)
is-wall?	(number number $\rightarrow$ boolean)
set-wall!	(number number boolean $\rightarrow \varnothing$)

Tabel 2.3: Operaties van het `Maze ADT`

Het `Maze ADT` bevat de volgende operaties:

- **new-maze**: Deze operatie maakt een object van het `Maze ADT` aan.
- **get-type**: Geeft het type van de ADT terug in dit geval 'maze.
- **get-height**: Geeft het hoogte van de doolhof terug. Deze is gedefinieerd door het GRID-HEIGHT constante in het "etc/constante.rkt" bestand.
- **get-width**: Geeft het hoogte van de doolhof terug. Deze is gedefinieerd door het GRID-WIDTH constante in het "etc/constante.rkt" bestand.
- **is-wall?**: Checkt of de gegeven rij en kolom een muur is en of de positie bestaat in het `Maze ADT`.
- **set-wall!**: Zet of verwijderd een muur (op basis van de gegeven

[1] Object is een object dat een Positie ADT bezit.

boolean warden) in het `Maze ADT` op de gegeven rij en kolom.

2.5 Position ADT

De `Position ADT` is verantwoordelijk voor het bijhouden en aanpassen van de positie, oriëntatie en snelheid van een object.

Naam	Signatuur
new-position	(number number → Positie)
get-type	(∅ → symbol)
get-x	(∅ → number)
get-y	(∅ → number)
get-orientation	(∅ → symbol ∪{#f})
get-speed	(∅ → number)
set-x!	(number → ∅)
set-y!	(number → ∅)
set-orientation!	(symbol → ∅)
set-moving!	(bool → ∅)
is-moving?	(∅ → boolean)
is-colliding?	(Position → boolean)
move!	(symbol → ∅)
peek	(symbol → Position)

Tabel 2.4: Operaties van het `Position ADT`

Het `Position ADT` bevat de volgende operaties:

- **new-positie**: Deze operatie maakt een nieuw object van het `Position ADT` aan. Deze neemt 2 nummers op als argument dat respectievelijk de x- en y-coördinaat representeren in het `Maze ADT`.
- **get-type**: Geeft het type van de ADT terug in dit geval 'position.
- **get-x**: Geeft x positie terug.
- **get-y**: Geeft y positie terug.
- **get-orientation**: Geeft de oriëntatie van het object terug. Het is standaard #f.
- **get-speed**: Geeft de snelheid van het object terug. De snelheid in dit context is de snelheid waarmee het spel een tile van 1 positie naar een ander tekent. Deze is standaard 0,17.

- **set-x**: Veranderd de x positie door de gegeven nummer.
- **set-y**: Veranderd de y positie door de gegeven nummer.
- **set-orientation**: Veranderd de oriëntatie positie door de gegeven richting.
- **set-moving**: Veranderd boolean waarde van het moving variabel. De variabel geeft aan of het object nog steeds aan het bewegen is van zijn vorige positie naar zijn nieuwe positie.
- **is-moving?**: Geeft aan of het object zich nog steeds aan het verplaatsen is naar zijn nieuwe positie.
- **is-colliding?**: Checkt of het meegegeven Positie de zelfde positie heeft in de Maze.
- **move!**: Verandert het positie en oriëntatie van het `Position ADT` in de gegeven richting. Zolang deze positie nog niet aan het bewegen is naar een nieuw positie.
- **peek**: Geef de positie terug van buur positie gelegen op de meegegeven richting.

2.6 Egg ADT

Het `Egg ADT` stelt een voorwerp voor dat kan opgeraapt worden door een `Player`.

Naam	Signatuur
new-egg	(Positie → Egg)
get-type	(∅ → symbol)
get-position	(∅ → Position)
take!	(∅ → ∅)
is-taken?	(∅ → boolean)

Tabel 2.5: Operaties van het `Egg ADT`

Het `Egg ADT` bevat de volgende operaties:

- **new-egg**: Maakt een nieuw object van het `Egg ADT`.
- **get-type**: Geeft het type van de ADT terug in dit geval 'egg.

- **get-position**: Geeft het `Position ADT` object terug die het egg object bezit.
- **take!**: Veranderd taken waarden naar #t. Met andere worden het egg is opgerapen.
- **is-taken?**: Check of het `Egg ADT` is opgeraapt of niet.

2.7 Player ADT

De `Player ADT` stelt het speler voor. Deze ADT is verantwoordelijk om de conditie van de speler bij te houden. Met andere woorden zijn aantal levens, zijn positie en de huidige leven conditie van de speler.

Naam	Signatuur
new-player	$(\varnothing \rightarrow \text{Player})$
get-type	$(\varnothing \rightarrow \text{symbol})$
get-position	$(\varnothing \rightarrow \text{Position} \cup \{\#f\})$
set-position!	$(\text{Position} \rightarrow \varnothing)$
is-dead?	$(\varnothing \rightarrow \text{boolean})$
die!	$(\varnothing \rightarrow \varnothing)$
revive!	$(\varnothing \rightarrow \varnothing)$

Tabel 2.6: Operaties van het `Player ADT`

De `Player ADT` bevat de volgende operaties:

- **new-player**: Maakt een object van `Player ADT` aan.
- **get-type**: Geeft het type van de ADT terug in dit geval 'player.
- **get-position**: Geeft de `Player ADT` object terug die het player object bezit.
- **set-position!**: Geeft de `Player ADT` een nieuw `Position ADT` object.
- **is-dead?**: Checkt of de `Player ADT` dood is of niet.
- **die!**: Veranderd de alive waarden naar #f en trekt 1 leven van lives af. Met andere woorden speler is gestorven.
- **revive!**: Veranderd de alive waarden naar #t als de speler nog lives heeft. De oriëntatie wordt ook gezet naar 'down omdat dit de standaard start oriëntatie is.

2.8 Scorpion ADT

De `Scorpion ADT` is de ADT verantwoordelijk voor het spellogica van de schorpioen. Deze ADT is verantwoordelijk om het onthouden van de positie, route en volgende beweging van de schorpioen.

Naam	Signatuur
new-scorpion	(Positie list → Scorpion)
get-type	(∅ → symbol)
get-position	(∅ → Position)
¡++¿	(¡++¿ → ¡++¿)

Tabel 2.7: Operaties van het `Scorpion ADT`

Het `Scorpion ADT` bevat de volgende operaties:

- ¡++¿: ¡++¿
- ¡++¿: ¡++¿

2.9 Views

2.10 View ADT

2.11 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 2.8: Operaties van het `View_Maze ADT`

Het `View_Maze ADT` bevat de volgende operaties:

- ++: ¡++¿
- ¡++¿: ¡++¿

2.12 View_Scorpion ADT

¡++¿

Naam	Signatuur
¡++¿	(¡++¿ → ¡++¿)

Tabel 2.9: Operaties van het `View_Scorpion ADT`

Het `View_Scorpion ADT` bevat de volgende operaties:

- **¡++¿**: ¡++¿
- **¡++¿**: ¡++¿

2.13 View_Player ADT

¡++¿

Naam	Signatuur
¡++¿	(¡++¿ → ¡++¿)

Tabel 2.10: Operaties van het `View_Player ADT`

Het `View_Player ADT` bevat de volgende operaties:

- **¡++¿**: ¡++¿
- **¡++¿**: ¡++¿

2.14 Eggs ADT

¡++¿

Naam	Signatuur
¡++¿	(¡++¿ → ¡++¿)

Tabel 2.11: Operaties van het `Eggs ADT`

Het `Eggs ADT` bevat de volgende operaties:

- **¡++¿**: ¡++¿
- **¡++¿**: ¡++¿

2.15 View_Eggs ADT

¡++¿

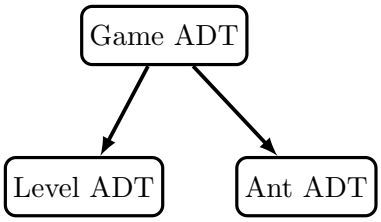
Naam	Signatuur
¡++¿	(¡++¿ → ¡++¿)

Tabel 2.12: Operaties van het `View_Eggs ADT`

Het `View_Eggs ADT` bevat de volgende operaties:

- **¡++¿**: ¡++¿
- **¡++¿**: ¡++¿

3 Afhankelijkheidsdiagram



4 Planning

...