Programmeerproject 1: FireAnt

Verslag Fase 2

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 2 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 tweede fase werd het spel uitgebreid met meerdere nieuwe functionaliteiten.

Onder andere:

- Een kan 2 soorten schorpioenen bevatten die elk op een verschillend manier bewegen. Deze schorpioenen kunnen ook tijdelijk versnellen, op een willekeurige moment.
- Het doolhof kan ook deuren bevatten die open gemaakt moeten worden door een sleutel die de speler moet nemen.
- Het doolhof bevat overstroomt gebieden die de speler enkel kan overlopen met een surfplank.
- Het doolhof bevat verschillende voedsel objecten die elk een verschillende effect heeben op de score.
- Het doolhof bevat verschillende voedsel objecten die elk een verschillende effect heeben op de score.

- Het spel bevat ook een scorebord die enkele belangrijke informaties weergeeft. Zoals hoogste score, levens, sleutels, etc.
- Het spel bevat voeding dat levens terug geeft.
- Een doolhof kan kogels bevatten, die gebruikt kunnen worden om schorpioenen dood te schieten.
- Het spel bevat 3 verschillende levels met verschillende puzzels. Het Spel herstart als het laatste level bereikt wordt

Het programma werd verdeelt 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 $\rightarrow$ Game)
start	($\varnothing \rightarrow \varnothing$)

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.

2.2 Modellen

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

2.2.1 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.
- **get-maze**: Geeft de `Maze ADT` object terug opgeslagen in de `Level ADT`.
- **get-items**: Geeft een lijst van `Items` objecten terug die opgeslagen zijn in de `Level ADT`.
- **get-items**: Geeft een lijst van `Doors` objecten terug die opgeslagen zijn in de `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.

Naam	Signatuur
new-level	(Player string → Level)
get-maze	(∅ → Maze)
get-items	(∅ → list)
get-doors	(∅ → list)
get-scorpions	(∅ → list)
get-updates	(∅ → list)
is-finished?	(Player → boolean)
is-legal-move?	(Object[1] symbol → boolean)
update!	(∅ → ∅)
kill-all!	(∅ → ∅)
clear-updates!	(∅ → ∅)
try-shooting!	(∅ → ∅)
try-opening!	(∅ → ∅)
try-surfing!	(∅ → ∅)
respawn	(∅ → ∅)

Tabel 2.2: Operaties van het `Level ADT`

- **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 kijk dan of het object die richting uit kan gaan.
- **update!**: Deze functie update alle nodige objecten die zich in het level bevinden.
- **kill-all!**: Deze functie zorgt er voor dat alle schorpioenen verwijderd worden in het volgende update iteratie.
- **clear-updates!**: Maakt de lijst leeg van de elementen die geüpdatet moeten worden
- **try-shooting!**: Kijkt of de speler een kogel kan schieten. Door een object van de type `Bullet` maken als de Speler genoeg ammo heeft.
- **try-opening!**: Kijkt of de speler tegen aan een deur loopt en of deze open gemaakt kan worden.
- **try-surfing!**: Kijkt of de speler tegen aan water loopt en of het deze kan oversteken.

- **respawn**: Deze procedure plaatst de Player terug op zijn start positie en geeft hem terug "leven".

2.2.2 Maze ADT

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

Naam	Signatuur
new-maze	(∅ → Maze)
get-type	(∅ → symbol)
get-unit	(number number → symbol)
clear-path!	(number number → ∅)
add!	(number number symbol → ∅)
is-accessible?	(number number → boolean)
get-height	(∅ → number)
get-width	(∅ → number)

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-unit**: Geeft het symbol terug dat op de meegegeven rij en kolom staat.
- **clear-path!**: Vervang de symbool, op de meegegeven positie, met 'empty.
- **add!**: Zet een symbol op de mee gegeven rij en kolom.
- **is-accessible?**: Checkt of de gegeven rij en kolom leeg ('empty) is en of de positie bestaat in het `Maze ADT`.
- **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"

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

bestand.

2.2.3 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-moving!	(bool → ∅)
set-orientation!	(symbol → ∅)
set-speed!	(number → ∅)
reset-speed!	(∅ → ∅)
is-moving?	(∅ → boolean)
is-colliding?	(Position → boolean)
move!	(symbol → ∅)
peek	(symbol → Position)
copy	(∅ → 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-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.
- **set-orientation**: Veranderd de oriëntatie positie door de gegeven richting.
- **set-speed!**: Veranderd de snelheid waarmee het object zich tussen twee posities verplaatst.
- **reset-speed!**: Veranderd de snelheid naar de standard snelheid waarden.
- **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.
- **copy**: maakt en geeft een copy van het huidige positie object.

2.2.4 Item ADT

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

Het `Item ADT` bevat de volgende operaties:

- **new-item**: Maakt een nieuw object van het `Item ADT`.
- **get-position**: Geeft het `Position ADT` object terug die het egg object bezit.

Naam	Signatuur
new-item	(Positie → Item)
get-position	(∅ → Position)
is-taken?	(∅ → boolean)
take!	(∅ → ∅)

Tabel 2.5: Operaties van het `Item ADT`

- **is-taken?**: Check of het `Item ADT` is opgeraapt of niet.
- **take!**: Veranderd taken waarden naar #t. Met andere worden het egg is opgerapen.

2.2.5 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, items, zijn positie en de huidige leven status.

Naam	Signatuur
new-player	(∅ → Player)
get-type	(∅ → symbol)
get-position	(∅ → Position ∪{#f})
get-lives	(∅ → number)
get-keys	(∅ → number)
get-boards	(∅ → number)
get-ammo	(∅ → number)
get-points	(∅ → number)
set-position!	(Position → ∅)
is-dead?	(∅ → boolean)
is-changed?	(∅ → boolean)
use!	(symbol → ∅)
collect!	(symbol number → ∅)
reset!	(∅ → ∅)
die!	(∅ → ∅)
revive!	(∅ → ∅)
update!	(∅ → ∅)

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 `Position ADT` object terug die het player object bezit.
- **get-lives**: Geeft aantal levens terug die de Player bezit.
- **get-keys**: Geeft het aantal Keys terug die de Player bezit.
- **get-boards**: Geeft het aantal Boards (surfplanken) die de speler bezit.
- **get-ammo**: Geeft het aantal Ammo terug die de Player bezit.
- **get-points**: Geeft de huidige score van de Player terug.
- **set-position!**: Geeft de `Player ADT` een nieuw `Position ADT` object.
- **is-dead?**: Checkt of de `Player ADT` dood is of niet.
- **is-changed?**: Checked of de locale variabelen van Player zijn veranderd.
- **collect!**: Verhoogd de gevraagde locale variabele van Player met de meegegeven hoeveelheid.
- **use!**: Verminderd de gevraagde locale variabele met 1.
- **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.
- **update!**: Geeft aan dat de Player niet veranderd is door de variabele "changed" naar #f te veranderen.

2.2.6 Scorpion ADT

De `Scorpion ADT` is de ADT verantwoordelijk voor het spellogica van de schorpioen. Deze ADT is verantwoordelijk om de snelheid en kleur van de schorpioen aan te passen.

Het `Scorpion ADT` bevat de volgende operaties:

Naam	Signatuur
new-scorpion	(symbol Positie list ∪ symbol → Scorpion)
get-type	(∅ → symbol)
get-position	(∅ → Position)
get-color	(∅ → symbol)
is-alive?	(∅ → bool)
try-boosting!	(number → ∅)
die!	(∅ → ∅)

Tabel 2.7: Operaties van het `Scorpion ADT`

- **new-scorpion**: Maakt een nieuwe object aan van het `Scorpion ADT`. Deze object neemt als argument symbol, dat een kleur aan geeft, een Positie (die bewaard wordt) en een lijst van argumenten of een symbol dat door de Scorpion-Green of Scorpion-Yellow wordt gebruikt.
- **get-type**: Geeft het type van de ADT terug in dit geval 'scorpion.
- **get-position**: Geeft de `Position ADT` object terug die het schorpioen object bezit.
- **get-color**: Geeft het kleur (symbol) van de Scorpion terug
- **is-alive?**: Checkt of de Scorpion levend is of niet.
- **try-boosting!**: Probeert de Schorpioen te versnellen zolang deze niet op cooldown is.
- **die!**: Veranderd de "alive" waarden naar #f

2.3 Views

De Views zijn de ADTs verantwoordelijk voor het teken van het spel en het weergeven van de modellen op een visuele manier.

2.3.1 View ADT

De `View ADT` is het centrale ADT voor het tekenlogica. Deze ADT is verantwoordelijk voor het aanmaken van de canvas, lagen en nodige Views die de Player en de andere objecten in een level voorstellen.