Programmeerproject 1:
Verslag (Fase II) “Fire-Ant”

Kouidane Rayane

1e Bachelor Computerwetenschappen

N°0587073

Rayane.Kouidane@vub.be

Academiejaar 2021-2022

Inhoudsopgave

1. Introductie

In dit verslag zal ik mijn implementatie van het spel “Fire-Ant” bespreken. In dit spel wordt een vuurmier bestuurd door de speler. De vuurmier moet doorheen verschillende doolhoven puzzels oplossen en zich niet laten aanraken door de schorpioenen, om naar het volgend level te kunnen gaan. Doorheen de levels zijn er ook power-ups en voedsel-objecten terug te vinden. Deze zullen een effect hebben op de score of de huidige spelsituatie. Om dit te kunnen tonen op het scherm hebben we het bestand ‘Graphics.rkt’ ter beschikking gekregen. Dit bestand bevat de nodige procedures om elementen op ons scherm te kunnen tekenen, het scherm te updaten en de invoer van de speler te kunnen herkennen.

De implementatie van het project werd ingedeeld in twee fases. Dit is de implementatie van de tweede fase, hierbij moet het hele spel geïmplementeerd worden. De volgende functionaliteiten moeten aanwezig zijn:

- Meerdere schorpioenen moeten aanwezig zijn. Zowel schorpioenen die een vast pad volgen als een willekeurig pad. De vuurmier moet bij aanraking met een schorpioen een leven verliezen en het level moet herstart worden. Op willekeurige momenten zullen de schorpioenen zich sneller verplaatsen en van kleur veranderen.
- Minstens 2 soorten puzzels uit een gegeven lijst van puzzels.
- Minstens 2 soorten power-ups uit een gegeven lijst van power-ups.
- Aanwezigheid van voedsel-objecten die een effect hebben op de score.
- Het bijhouden van de huidige score, hoogste score en het aantal levens.
- Minstens 3 levels met elk hun eigen doolhof en puzzels.

In dit documenten zullen de volgende punten besproken worden: de gebruikte ADT’s met de signaturen van hun operaties, het nut van andere belangrijke bestanden, het afhankelijkheidsdiagram en de planning van de tweede fase. Uiteindelijk zal ik de verschillen tussen de voorstudie en het verslag van de tweede fase bespreken.

2. ADT’s

De informatie over de gebruikte ADT’s, de operaties en signaturen ervan zijn in deze sectie terug te vinden. In elk tabel staat de naam van de operaties en de signatuur ervan. De gebruikte operaties worden hier uitgelegd. Deze sectie bevat ook een korte beschrijving van hoe het spel moet worden opgestart en welke bestanden uitgevoerd worden.

2.1 Fire-ant-game

Om het spel te kunnen starten moet het bestand ‘fire-ant-game.rkt’ geopend worden. In dit bestand worden alle gebruikte ADT’s, hulp-procedures, constanten en andere bestanden, die bijvoorbeeld de doolhoven bevat, ingeladen. Om het spel op te starten maken we eerst een game-object aan en sturen we hiernaar het bericht ‘start’. In dit bestand heet het game-object fire-ant-game en we starten het spel dankzij de oproep van ‘`(fire-ant-game 'start)`’. Bij het opstarten van het spel worden de spel-lus en de procedure, die de invoer van de speler verwerkt, gestart.

Enkele toetsen zullen nodig zijn om het spel te kunnen herstarten of om naar een volgende level te kunnen gaan. Wanneer het spel gestart wordt kan de speler op ‘r’ duwen om een level te herstarten. Dit kan handig zijn als de speler met een puzzel vastzit en niet meer verder kan gaan. Wanneer de speler het spel heeft gewonnen of verloren dan verschijnt er een scherm en de speler kan dan op ‘f’ drukken om terug naar de eerste level te gaan. Om snel de verschillende levels te tonen heb ik de toetsen ‘1’, ‘2’ en ‘3’ ingevoerd om van het ene level naar het ander te gaan.

2.2 Game ADT

In het Game ADT zitten de procedures voor het spel- en tekenlogica. De spelomgeving wordt getekend en de invoer van de speler wordt verwerkt. In dit ADT wordt het spel gestart.

Naam	Signatuur
`make-game`	`( ∅ → Game )`
`start!`	`( ∅ → ∅ )`

Tabel 1: Operaties van het Game ADT

Het Game ADT bevat volgende operaties:
- `make-game` zal een Game-object aanmaken.
- `start!` is de procedure die het spel zal opstarten.

2.3 Draw ADT

Het Draw ADT bevat de procedures die nodig zijn voor de tekenlogica. Dit ADT zal ervoor zorgen dat het spel, op ons scherm, getekend wordt.

Naam	Signatuur
`make-draw`	`( ∅ → Draw )`
`draw-level!`	`( Level → ∅ )`
`draw-number!`	`( number → ∅ )`
`draw-maze!`	`( Level → ∅ )`
`draw-score!`	`( Score → ∅ )`
`draw-end!`	`( symbol number → ∅ )`
`clear-level!`	`( ∅ → ∅ )`
`clear-all-layers!`	`( ∅ → ∅ )`
`reset-tiles!`	`( Fire-ant pair → ∅ )`
`add-drawables-after-lost`	`( ∅ → ∅ )`
`game-loop-function!`	`( ( number → ∅ ) → ∅ )`
`game-key-function!`	`( ( symbol any → ∅ ) ∅ )`

Tabel 2: Operaties van het Draw ADT

Het Draw ADT bevat volgende operaties:
- `make-draw` maakt een Draw-object aan. Het bevat de nodige procedures om de spelsituatie te tekenen.
- `draw-level!` neemt een Level-object als argument en gaat de elementen op de juiste plaats tekenen, afhankelijk van de configuratie van het level.
- `draw-number!` neemt een getal als argument. Dit getal stelt het level-nummer voor.
- `draw-maze!` neemt een Level-object als argument en tekent het correcte doolhof.
- `draw-score!` neemt een Score-object als argument en gaat de huidige score, hoogste score, levens en extra-levens op het scherm tekenen.
- `draw-end!` zal het eindscherm tonen (afhankelijk van het symbool zal er iets anders staan).
- `clear-level!` is een procedure dat bij de level-overgang de oude elementen zal wissen.
- `clear-all-layers!` zal alle tiles van elke layer verwijderen.
- `reset-tiles!` neemt een Fire-ant-object en een lijst van Scorpion-objecten en gaat, bij level-overgangen en het herstarten van levels, de tiles in de juiste richting plaatsen.
- `add-drawables-after-lost` is een procedure dat nodig zal zijn om elementen zoals het doolhof of de score weer op het scherm te brengen na het winnen of verliezen van het spel. Omdat we de tiles van alle layers hebben verwijderd.
- `game-loop-function!` zal het venster bij elke spel-lus updaten.
- `game-key-function!` is de procedure dat de invoer van de speler zal verwerken.

2.4 Score ADT

Het Score ADT is het ADT dat de huidige score, hoogste score, aantal levens en aantal extra-levens zal opslaan en wijzigen.

Naam	Signatuur
`make-score`	`( ∅ → Score )`
`get-current-score`	`( ∅ → number )`
`get-high-score`	`( ∅ → number )`
`update-current-score!`	`( number → ∅ )`
`reset-current-score!`	`( ∅ → ∅ )`
`lives`	`( ∅ → number )`
`extra-lives`	`( ∅ → number )`
`update-lives!`	`( number → ∅ )`
`reset-lives!`	`( ∅ → ∅ )`
`update-extra-lives!`	`( number → ∅ )`
`reset-extra-lives!`	`( ∅→ ∅ )`

Tabel 3: Operaties van het Score ADT

Het Score ADT bevat volgende operaties:
- `make-score` maakt een Score-object dat de huidige score, hoogste score, levens en extra-levens zal bijhouden en kunnen veranderen.
- `get-current-score` geeft de huidige score terug.
- `get-high-score` geeft de hoogste score terug.
- `update-current-score!` wijzigt de huidige score met het gegeven getal.
- `reset-current-score!` zet de huidige score weer op 0.
- `lives` geeft het aantal levens terug.
- `extra-lives` geeft het aantal extra-levens terug.
- `update-lives!` wijzigt het aantal levens.
- `reset-lives!` zet het aantal levens op 3.
- `update-extra-lives!` wijzigt het aantal extra-levens.
- `reset-extra-lives!` zet het aantal extra-levens op 0.

2.5 Timer ADT

Het Timer ADT zal een Timer-object aanmaken dat de een stopwatch voorstelt.

Naam	Signatuur
`make-timer`	`( number → Timer)`
`update!`	`( number → ∅ )`
`reset!`	`( ∅→ ∅)`
`active?`	`( ∅ → boolean )`

Tabel 4: Operaties van het Timer ADT

Het Timer ADT bevat volgende operaties:
- `make-timer` neemt een getal (stelt het aantal seconden voor) als argument en maakt een Timer-object aan.
- `update!` zal de timer updaten dankzij het gegeven getal (delta-tijd). Dit getal wordt gegeven door de game-loop functie.
- `reset!` zet de timer op 0.
- `active?` kijkt of de timer actief is. Als de timer de gegeven tijd overschrijdt dan geeft het een #f terug, zo niet een #t.

2.6 Level ADT

Het Level ADT stelt een level voor. Hierin is de logica van het spel geïmplementeerd. De verandering van score bij het opeten van voedsel, het verliezen van een leven bij aanraking met een schorpioen… Dit is terug te vinden in dit ADT.

Naam	**Signatuur**
`make-level`	`( number → Level )`
`fire-ant-object`	`( ∅ → Fire-ant )`
`scorpion-objects`	`( ∅ → pair )`
`key-objects`	`( ∅ → pair )`
`food-objects`	`( ∅ → pair )`
`power-up-objects`	`( ∅ → pair )`
`pressure-plate-objects`	`( ∅ → pair )`
`move-fire-ant!`	`( symbol → ∅)`
`won?`	`( symbol → boolean )`
`lost?`	`( Score → boolean )`
`update!`	`(number →  ∅ )`
`update-score!`	`( Score ( symbol → ∅ ) → ∅ )`
`maze`	`( ∅ → vector )`

Tabel 5: Operaties van het Level ADT

Het Level ADT bevat volgende operaties:
In dit ADT worden de deuren geactiveerd dankzij de procedure activate-doors!. Deze zal een bericht sturen naar de Key- en Pressure-plate-objecten. Hiermee zullen de posities van de deuren van beide ADT's teruggegeven worden. Daarna wordt er een getal geplaatst, in het doolhof, op de posities van de deuren.
- `make-level` neemt een getal als invoer. Dit getal stelt een bepaalde level-configuratie voor. Deze procedure zal een Level-object aanmaken. Het Level-object zal het level kunnen updaten.
- `fire-ant-object` geeft het Fire-ant-object terug.
- `scorpion-objects` geeft een lijst met de Scorpion-objecten terug.
- `key-objects` geeft een lijst van Key-objecten terug.
- `food-objects` geeft een lijst van Food-objecten.
- `power-up-objects` geeft een lijst van Power-up-objecten terug.
- `pressure-plate-objects` geeft een lijst van Pressure-plate-objecten terug.
- `move-fire-ant`! is de procedure die ervoor zal zorgen dat mijn mier kan bewegen in functie van de gebruikersinvoer.
- `won?` geeft een boolean terug dat aangeeft of de speler klaar is met een level of het spel. Om dit weten kijkt het of de positie van de mier gelijk is aan de positie van de uitgang (als de level-configuratie gelijk is aan het laatste level dan is het spel gedaan, zo niet is het level gedaan.
- `lost?` kijkt of de speler verloren is, dus als de mier geen levens en extra-levens meer heeft.
- `update!` neemt een getal als invoer (delta-tijd) en zal de verschillende elementen zoals: puzzels, schorpioenen… updaten. De positie en het effect van elk element wordt ook geüpdatet (bijvoorbeeld score wijzigen, power-up activeren…).
- `update-score!` neemt een Score-object en een lambda, met de procedure om een het level te herstarten, als invoer. Deze operatie zal het scoreboard en het effect van de power-ups updaten.
- `maze` geeft een vector terug. Deze vector is een matrix dat het huidige doolhof voorstelt.

2.7 Position ADT

Het Position ADT maakt een Position-object en dit stelt een positie voor in een 2-dimensionaal assenstelsel.

Naam	Signatuur
`make-position`	`( number number → Position )`
`x!`	`( number → ∅ )`
`y!`	`( number → ∅ )`
`same?`	`( Position → boolean )`
`move!`	`( symbol number → ∅ )`

Tabel 6: Operaties van het Position ADT

Het Position ADT bevat volgende operaties:
- `make-position` neemt 2 getallen als invoer en maakt een Position-object aan.
- `x!` en `y!` zullen respectievelijk de x-coördinaat en y-coördinaat van een positie wijzigen met het meegegeven getal.
- `same?` geeft een boolean terug dat aangeeft of twee posities gelijk zijn.
- `move!` zal een positie wijzigen in functie van een meegegeven richting en een getal. Dit getal stelt de afstand van een beweging voor (bijvoorbeeld: één stap).

2.8 Fire-ant ADT

Het Fire-ant ADT creëert een Fire-ant-object dat een vuurmier voorstelt. Onze vuurmier zal moeten kunnen bewegen en kijken of het andere objecten raakt.

Naam	Signatuur
`make-fire-ant`	`( Position → Fire-ant )`
`position`	`( ∅ → Position )`
`direction`	`( ∅ → symbol )`
`previous-direction`	`( ∅ → symbol )`
`direction!`	`( symbol → ∅ )`
`prev-direction!`	`( symbol → ∅ )`
`move-ant!`	`( symbol vector → ∅ )`
`hit?`	`(Scorpion U Food U Power-up U Key → Boolean )`

Tabel 7: Operaties van het Fire-ant ADT

Het Fire-ant ADT bevat volgende operaties:
- `make-fire-ant` neemt een positie als argument en zal een Fire-ant-object aanmaken.
- `position` geeft de positie van de vuurmier weer.
- `direction` geeft de huidige richting van de vuurmier.
- `previous-direction` geeft de vorige richting van de vuurmier terug. Dit variabele zal nuttig zijn wanneer we de tiles van een bewegend object moeten wijzigen.
- `direction!` neemt een nieuwe richting als invoer en zal hiermee de huidige richting wijzigen.
- `prev-direction!` neemt een richting als invoer en zal de vorige richting wijzigen.
- `move-ant!` is de procedure die ervoor zorgt dat mijn vuurmier kan bewegen en niet tegen een muur botst.
- `hit?` neemt een object en 2 getallen als invoer. De 2 getallen stellen de hoogte en breedte van het object voor. Hiermee kunnen we zien of de vuurmier een ander object raakt.

2.9 Scorpion ADT

Het Scorpion ADT maakt een Scorpion-object aan. Dit object stelt een schorpioen voor. De schorpioenen kunnen een vast of een willekeurig pad volgen en kunnen op willekeurig momenten sneller gaan.

Naam	**Signatuur**
`make-scorpion`	`( Position symbol → Scorpion )`
`position`	`( ∅ → Position )`
`direction`	`( ∅ → symbol )`
`previous-direction`	`( ∅ → symbol )`
`direction!`	`( symbol → ∅ )`
`prev-direction!`	`( symbol → ∅ )`
`follow!`	`( symbol number → ∅ )`
`opposite?`	`( ∅ → boolean )`
`switch!`	`( ∅ → ∅ )`
`type`	`( ∅ → symbol )`
`switched-to-fast?`	`( ∅ → boolean )`
`speed-test`	`( ( ∅ → ∅ ) → ∅ )`

Tabel 8: Operaties van het Scorpion ADT

Het Scorpion ADT bevat volgende operaties:
- `make-scorpion` neemt een positie en een symbool als argumenten. Het symbool stelt het type schorpioen voor (normaal of willekeurig). Deze procedure maakt een Scorpion-object aan.
- `position` geeft de positie van de schorpioen terug.
- `direction` geeft de huidige richting van de schorpioen terug.
- `previous-direction` geeft de vorige richting terug.
- `direction!` is de procedure die ervoor zorgt dat we de huidige richting kunnen vervangen.
- `prev-direction!` zal de vorige richting vervangen.
- `follow!` neemt een symbool en een getal als argument. Het symbool stelt een richting voor en het getal is de delta-tijd. De schorpioen beweegt dan volgens de gegeven richting.
- `opposite?` kijkt of een schorpioen het omgekeerde pad moet volgen.
- `switch!` zal opposite? in #t veranderen als het #f is en omgekeerd.
- `type` geeft het type schorpioen terug (‘normal of ‘random).
- `switched-to-fast?` geeft een boolean terug dat aantoont of een schorpioen sneller gaat.
- `speed-test` neemt een procedure als invoer. Als de schorpioen versnellen of vertragen dan gaat het de procedure uitvoeren.

2.10 Food ADT

Het Food ADT maakt een Food-object aan. Dit object stelt voedsel voor. Ik heb gekozen om een ei als voedsel te beschouwen en is dus terug te vinden in dit ADT.

Naam	**Signatuur**
`make-food`	`( Position → Food )`
`position`	`( ∅ → Position )`
`collected?`	`( ∅ → boolean )`
`collect!`	`( ∅ → ∅ )`
`type`	`( ∅ → symbol )`
`bonus`	`( ∅ → number )`

Tabel 9: Operaties van het Food ADT

Het Food ADT bevat volgende operaties:
- `make-food` neemt een positie als argument en maakt een Food-object aan.
- `position` geeft de positie terug van het Food-object.
- `collected?` zal #t teruggeven als het voedsel opgerapen werd, zo niet een #f.
- `collect!` veranderd de collected? in een #t.
- `type` geeft een symbool terug. Dit symbool stelt het type voedsel dat het is. Er zijn in totaal 5 types: ‘egg’, ‘rare’, ‘epic’, ‘legendary’ en ‘mythical’.
- `bonus` geeft een getal weer. Dit getal stelt het bonus voor dat we zullen optellen bij de huidige-score. Elk type voedsel heeft een eigen bonus. Hoe zeldzamer het type is, hoe groter de bonus die het oplevert.

2.11 Pressure-plate ADT

Het Pressure-plate ADT maakt een Pressure-plate-object aan, dat een drukplaat voorstelt.

Naam	**Signatuur**
`make-pressure-plate`	`( Position Position Position → Pressure-plate )`
`position`	`( ∅ → Position )`
`door-position`	`( ∅ → Position )`
`block-position`	`( ∅ → Position )`
`move!`	`( Fire-ant vector → ∅ )`
`pressed?`	`( ∅ → boolean )`

Tabel 10: Operaties van het Pressure-plate ADT

Het Pressure-plate ADT bevat volgende operaties:
- `make-pressure-plate` neemt 3 posities als invoer. Deze posities zijn de posities van het drukplaat, de deur en het blok. Deze procedure maakt een Pressure-plate-object aan.
- `position` geeft de positie van het drukplaat terug.
- `door-position` geeft de positie van de deur terug.
- `block-position` geeft de positie van het blok terug.
- `move!` is de procedure die ervoor zorgt dat het blok kan bewegen, als de vuurmier ertegen duwt, en dat het niet binnen de muren kan geraken.
- `pressed?` geeft een #t als het blok op de drukplaat staat. Dus als de positie van beide elementen gelijk zijn.

2.12 Key ADT

Het Key ADT maakt een Key-object aan. Dit object stelt een sleutel voor. Bij het oprapen van een sleutel wordt een deur in het doolhof geopend.

Naam	**Signatuur**
`make-key`	`(Position Position → Key)`
`collected?`	`(∅ → boolean)`
`collect!`	`(∅ → ∅)`
`position`	`(∅ → Position)`
`door-position`	`(∅ → Position)`

Tabel 11: Operaties van het Key ADT

Het Key ADT bevat volgende operaties:
- `make-key` neemt 2 posities als argumenten. De eerste positie is deze van de sleutel en tweede is deze van de deur. Deze procedure maakt een Key-object aan.
- `collected?` geeft een boolean terug. Als de sleutel opgerapen wordt door de mier dan is deze #t zo niet blijft het #f.
- `collect!` is de procedure die de variabele collected? zal veranderen in een #t.
- `position` geeft de positie van de sleutel weer.
- `door-position` geeft de positie van de deur weer.

2.13 Power-up ADT

Het Power-up ADT maakt een Power-up-object aan. Dit object stelt een power-up voor.

Naam	**Signatuur**
`make-power-up`	`(Position symbol Timer → Power-up)`
`position`	`(∅ → Position)`
`type`	`(∅ → symbol)`
`active?`	`(∅ → boolean)`
`activate!`	`(∅ → ∅)`
`activated?`	`(∅ → boolean)`
`activate-one-time!`	`(∅ → ∅)`
`disable!`	`(∅ → ∅)`
`collected?`	`(∅ → boolean)`
`collect!`	`(∅ → ∅)`
`start-timer!`	`(number → ∅)`

Tabel 12: Operaties van het Power-up ADT

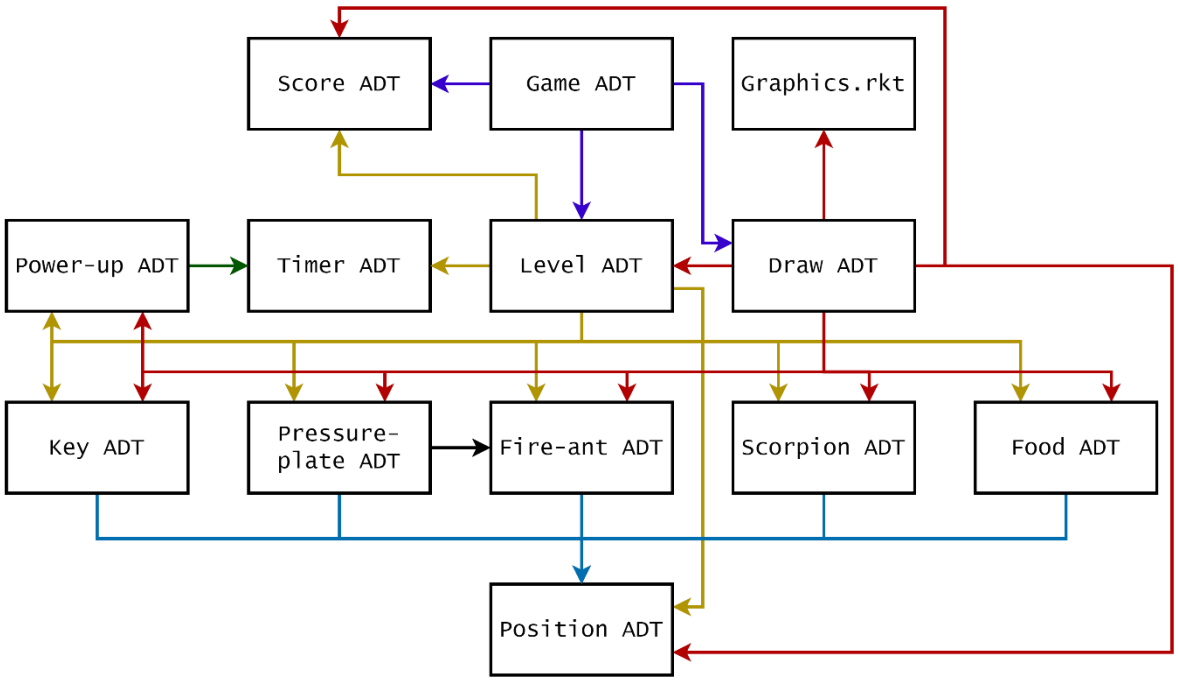
Het Power-up ADT bevat volgende operaties:
- `make-power-up` neemt een positie, een symbool en eventueel een Timer-object. Deze procedure maakt een Power-up-object aan.
- `position` geeft de positie van de power-up terug.
- `type` geeft het type power-up dat het is (voorlopig: shield of extra-life)
- `active?` zegt ons of een power-up nog actief is.
- `activate!` zal de variabele active? omzetten naar een #t.
- `activated?` kijkt of een power-up geactiveerd werd.
- `activate-one-time!` zal de variabele active? gedurende 1 spel-lus op #t zetten.
- `disable!` zal de variabele active? op #f zetten.
- `collected?` kijkt of een power-up genomen werd.
- `collect!` zal de variabele collected? op #t zetten.
- `start-timer` zal de power-up activeren gedurende de tijd dat meegegeven werd met de timer.

3. Belangrijke bestanden

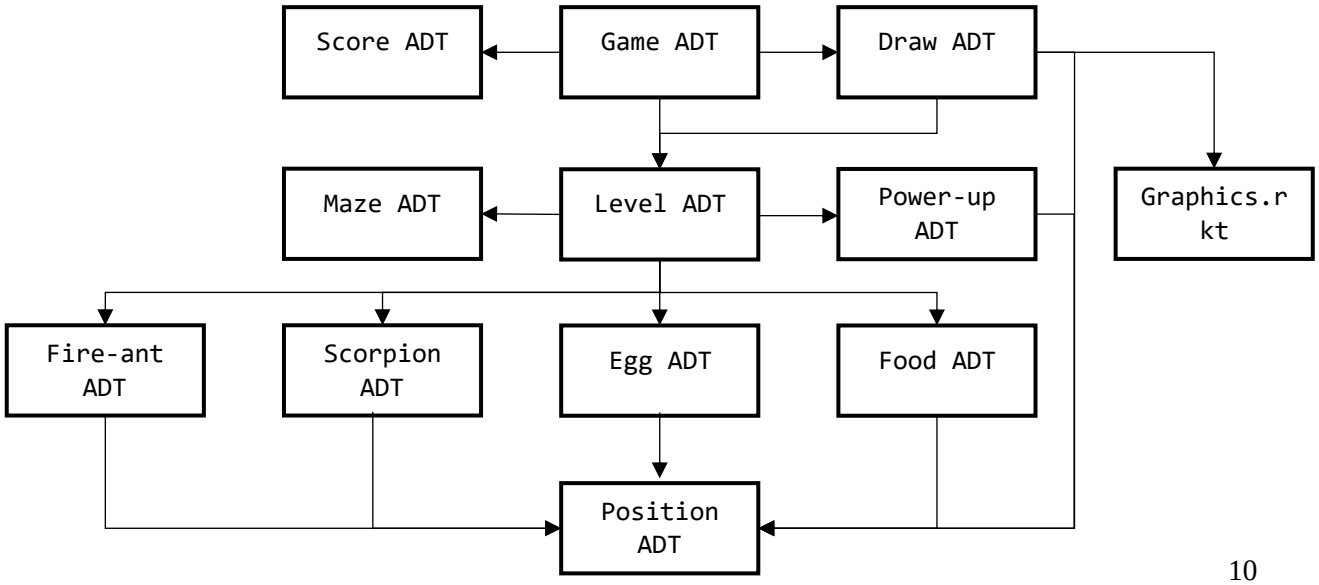
In deze sectie zullen andere belangrijke bestanden besproken worden.

- `mazes.rkt` in dit bestand zitten de matrices die de doolhoven voorstellen. Een matrix wordt hier voorgesteld door een vector van vectoren. De muren worden voorgesteld door een 1.
- `scorpion-paths.rkt` is het bestand dat de vaste paden en willekeurige paden bevat. Ik heb gekozen om 1 grote procedure te maken met daarin de verschillende paden. Hiermee moet ik niet bij elke procedure dezelfde abstracties maken om de positie en richting van een schorpioen te krijgen. Een pad van een schorpioen wordt voorgesteld als een lijst van kruispunten en richtingen. Wanneer de schorpioen een van de posities tegenkomt dan gaat het zijn richting veranderen in de meegegeven richting. Een schorpioen die een willekeurig pad volgt zal van richting veranderen wanneer het tegen een muur botst. (Het controleert of het tegen een muur botst door te kijken in de lijst van muur-posities)
- `constants.rkt` bevat alle globale constanten dat verschillende procedures nodig zullen hebben.

4. Afhankelijkheidsdiagram



Figuur 1: Afhankelijkheidsdiagram (Verslag Fase II)



Figuur 2: Afhankelijkheidsdiagram (Voorstudie Fase II)

In figuur 1 is het afhankelijkheidsdiagram van fase 2 terug te vinden en in figuur 2 deze van de voorstudie van fase 2. Een afhankelijkheidsdiagram stelt de relaties tussen de verschillende ADT's visueel voor. Eerst zal ik figuur 1 bespreken en vervolgens zal ik de verschillen tussen de twee figuren bespreken.

Het Draw ADT is het enige ADT dat afhankelijk is van het bestand 'Graphics.rkt'. Dit is omdat dit ADT ervoor zal zorgen dat alle elementen op het scherm getekend worden en om dit te doen moet het procedures kunnen gebruiken die nodig zijn om het scherm te updaten en erop te kunnen tekenen. Daarnaast is het Draw ADT afhankelijk van het Level ADT, Score ADT, Position ADT, Fire-ant ADT, Scorpion ADT, Food ADT, Power-up ADT, Pressure-plate ADT en Key ADT. Hiermee zal het de verschillende elementen (vuurmier, schorpioen, power-ups, doolhof…) in de juiste richting tekenen, de elementen verwijderen als het moet en de huidige score, hoogste score en levens tekenen.

Het Level ADT is afhankelijk van het Score ADT, Position ADT, Fire-ant ADT, Scorpion ADT, Food ADT, Power-up ADT, Pressure-plate ADT en Key ADT. In het Level ADT staan de lijsten met de begin-posities van alle elementen. De procedures, die ervoor zorgen dat de power-ups werken en dat de verschillende elementen opgerapen kunnen worden en hierbij eventueel de score veranderen, zitten ook in dit ADT. Dit kan alleen als het aan de andere ADT kan en procedures uit deze ADT's kan gebruiken. Het Level ADT is ook afhankelijk van het Timer ADT. Dit zal nodig zijn om bijvoorbeeld de schorpioenen op willekeurige momenten gedurende 10 seconden te laten versnellen.

Het Game ADT is afhankelijk van het Score ADT omdat het de score moet kunnen wijzigen wanneer we een level herstarten of wanneer we verloren zijn. Daarnaast is het ook afhankelijk van het Level ADT. Hierin staan alle procedures die nodig zijn om de spelsituatie te kunnen berekenen en updaten. Ten slotte is het nog afhankelijk van het Draw ADT hiermee kan het de huidige spelsituatie, dat berekend werd door het Level ADT, op het scherm tekenen.

Het Fire-ant ADT, Scorpion ADT, Food ADT, Power-up ADT, Pressure-plate ADT en Key ADT gebruiken allemaal procedures uit het Position ADT hiermee krijgen ze een positie in de ruimte en kunnen we ze laten bewegen door de positie te wijzigen. Het Power-up ADT is nog afhankelijk van het Timer ADT. Als we een power-up willen aanmaken dat gedurende bv.10 seconden actief is dan zullen we een timer nodig hebben. In mijn implementatie heb ik het gebruikt om de 'shield power-up' te implementeren. Omdat de bescherming 10 seconden duurt. Het Pressure-plate ADT is nog afhankelijk van het Fire-ant ADT. Wanneer de mier het blok verplaatst zal het kijken of er geen muur is, zo niet dan zal het blok in dezelfde richting als de mier bewegen.

Wanneer we naar de twee afhankelijkheidsdiagrammen kijken dan zien we een duidelijk verschil. Ten eerste zien we dat het Maze ADT verdwenen is. Ik merkte op dat ik geen ADT nodig had voor de doolhoven. Ik heb gekozen om de matrices die mijn doolhoven representeren apart te plaatsen in een bestand genaamd 'mazes.rkt'. Ten tweede kunnen we ook zien dat er geen Egg ADT meer is. Ik heb beslist om het ei te implementeren in het Food ADT omdat beide ADT's veel gelijkenissen hadden. Ten derde zien we enkele nieuwe ADT's namelijk Pressure-plate ADT, Key ADT en Timer ADT. De twee eerste ADT's stellen puzzels voor. Deze moeten opgelost worden door de speler om naar het volgende level te gaan. Het laatste ADT, het Timer ADT, zal ervoor zorgen dat we power-ups gedurende enkele seconden actief blijven en ook dat we de schorpioenen kunnen laten versnellen gedurende 10 seconden. Als laatst zien we dat het Draw ADT nu ook afhankelijk is van de andere ADT's zoals Fire-ant, Scorpion, Food, Score … dit is nodig omdat het naar bijvoorbeeld constanten gaat kijken in het ADT om te zien of een element getekend of verwijderd moet worden. Maar zodat ook de scores en levens op het scherm getoond worden.

5. Planning

Week	Actie
Week 24-25	ADT's aanpassen en aan de voorstudie werken
Week 26	Werken aan Score ADT en proberen de score op het scherm te tonen. Daarna proberen om een ei op te rapen en score te verhogen.
Week 27	Verwijderen van Egg ADT en werken op implementatie van Food ADT. Proberen om voedsel op het scherm te krijgen en score te verhogen bij het opeten van dit object
Week 28	Implementeren van de puzzles: Key ADT en Pressure-plate ADT. Proberen om de puzzels te laten werken.
Week 29-30	Implementatie van de power-ups: Power-up ADT + Timer ADT. (verwijderen van Maze ADT)
Week 31-33	Het Level ADT, Game ADT en Draw ADT wijzigen om het geheel met elkaar te werken
Week 34-35	Verder werken op alle ADT's en proberen o ze samen te linken + code proper maken + verslag schrijven
Week 36-37	Studeren voor de examens
Week 37	Indienen code en verslag

Tabel 13: Werkelijke planning Fase II

Week	Actie
Week 24-25	ADT's aanpassen en aan de voorstudie werken
Week 26	Score ADT en Food ADT implementeren en proberen om de score en het voedsel op het beeld te krijgen
Week 27	Proberen om Food ADT en Score ADT met elkaar te linken zodat de score stijgt bij het eten van voedsel
Week 28	Verder werken aan Food- en Score ADT
Week 29-30	Power-ups implementeren
Week 31-33	Power-up ADT implementeren en beginnen met de puzzels
Week 34-35	Alle ADT's met elkaar linken en het geheel testen en beginnen met het verslag
Week 36-37	Studeren voor de examens
Week 37	Indienen code en verslag

Tabel 14: Planning voorstudie fase II

In tabel 13 is mijn werkelijke planning terug te vinden en in tabel 14 deze van de voorstudie van de tweede fase. Ik heb mijn planning ongeveer kunnen volgen. Eerst moest ik alle problemen van de implementatie van de eerste fase oplossen. Dit heeft me enkele weken gekost. Daarna kon ik eindelijk aan de nieuwe ADT's beginnen. Ik probeerde om elke week meerdere ADT's aan te maken en ze met elkaar te linken. Het ADT die mij het meeste tijd heeft gekost is het Power-up ADT. Als we naar de code kijken is dit ADT niet zo groot, maar om tot dit idee te komen had ik meer tijd dan verwacht nodig. Ik heb beslist dat het Power-up ADT ervoor ging zorgen dat een power-up geactiveerd en uitgeschakeld kon worden (na een bepaalde tijd of bij het oprapen). Ik heb ook gekozen om elk van de puzzels een ADT te geven. Wanneer de ADT's klaar waren moest ik ervoor zorgen dat ze allemaal goed gelinkt waren met elkaar en dat het spel helemaal in orde was.

6. Bronnen

Bronnen	
Implementatie spel (inspiratie)	https://canvas.vub.be/courses/22822/files/folder/WPO%20Grafische%20bibliotheek/Oplossing
Bitmaps	Mier-bitmap Schorpioen-bitmap Ei-bitmap Sleutel-bitmap Achtergrond-bitmap

Tabel 15: Gebruikte bronnen