Programmeerproject 1 (2021–2022): Fire Ant

Prof. Dr. Joeri De Koster
Bjarno Oeyen
Carlos Rojas Castillo
(programmeerproject1@soft.vub.ac.be)

1 Introductie

De bedoeling van de cursus PROGRAMMEERPROJECT 1 is om de concepten en technieken die je gezien hebt in andere opleidingsonderdelen (e.g., STRUCTUUR VAN COMPUTERPROGRAMMA'S 1 en ALGORITMEN EN DATASTRUCTUREN 1) opnieuw toe te passen. Maar in tegenstelling tot deze andere opleidingsonderdelen, zal je deze concepten en technieken moeten toepassen op een grotere code base.

Het project zal in verschillende fases moeten opgeleverd worden. Voor elke fase maak je, alvorens je begint met programmeren, eerst een **analyse** waarbij je bepaalt wat je die fase gaat doen. Je maakt een voorlopig **ontwerp** van het project door de abstracties (ADT's, ofwel Abstracte Data Types) te ontwerpen die je verwacht nodig te hebben voor je implementatie. Door op voorhand deze ADT's te bepalen, zorg je ervoor dat je tijdens het programmeren een beter idee hebt over hoe je uiteindelijke code er zal uitzien...

Vanaf zodra je een ontwerp hebt kan je beginnen met programmeren. Naast het feit dat je project beoordeeld zal worden op **functionaliteit** ("wat je project kan"), zal je project dus ook beoordeeld worden op **codekwaliteit** ("hoe je je project geïmplementeerd hebt"). Dat betekent niet alleen dat je jouw code niet onnodig complex maakt, maar ook dat je nadenkt over hoe eenvoudig het is om de code te onderhouden en in de toekomst uit te breiden met extra functionaliteit. Vermijd dus codeduplicatie, magische constanten, lange procedures met repeterende patronen en andere *code smells*.

Vanaf zodra je alle functionaliteit hebt geïmplementeerd ga je je project ook moeten **documenteren** in een technisch document. Zo'n document dient als een beschrijving voor andere programmeurs (of voor een toekomstige versie van jezelf) zodanig dat deze kunnen lezen hoe je jouw project ontworpen en geïmplementeerd hebt. Motiveer de beslissingen die je gemaakt hebt door uit te leggen hoe je verschillende functionaliteiten geïmplementeerd hebt, en kijk na in welke mate je je aan je originele ontwerp hebt gehouden. Ten laatste zal je in elke fase jouw project ook moeten **verdedigen**. Hierbij ga je niet alleen moeten aantonen dat je de gevraagde functionaliteit effectief geïmplementeerd hebt, maar ga je ook de keuzes die je in de loop van het project gemaakt hebt moeten motiveren.

2 Opdracht: Fire Ant

De opdracht voor dit academiejaar bestaat erin een eigen versie van het spelletje FIRE ANT (Vuurmier) te ontwerpen en te implementeren. Figuur 1 schetst hoe dit spel er uitziet.

Figuur 1: Het vijfde level van het FIRE ANT-spel. Een volledige walkthrough van het originele spelletje kan teruggevonden worden op YouTube: `https://www.youtube.com/watch?v=hrjjurdtoXo`.

In dit spel bestuurt de speler een *vuurmier* (rechtsbovenaan in figuur 1) dat zijn mierenkoningin tracht te redden van een aanvallend schorpioenenleger. Hiervoor moet de vuurmier levels voltooien die steeds toenemen in moeilijkheidsgraad. Elk level is een doolhof bewaakt door schorpioenen en bestaat uit verschillende soorten puzzels die op te lossen zijn door de speler. Het spel eindigt wanneer de speler al zijn levens verliest of wanneer deze alle levels succesvol vervolledigt heeft en de mierenkoningin redt.

2.1 Spelinteracties

In dit onderdeel beschrijven we kort de verschillende elementen van FIRE ANT en de manier waarop ze aan elkaar gerelateerd zijn. Deze beschrijving bevat een overzicht van de verwachte functionaliteit zoals deze in de laatste fase van het spel moet opgeleverd worden. In de eerste fase zullen slechts enkele spelelementen geïmplementeerd moeten worden (zie sectie 2.2.1).

A: De spelwereld Een spelwereld bestaat uit de verschillende spelelementen die samen zorgen voor het speelbaar spel. Doorgaans bestaat de spelwereld uit het doolhof, de vuurmier, de schorpioenen, de spelelementen horende bij de puzzels en eventuele power-ups.

B: Doolhof Een doolhof bestaat uit muren die bepalen waar al de andere spelelementen in kunnen bewegen. Het doolhof heeft een ingang (in figuur 1 bevindt deze zich bovenaan) en een uitgang (in figuur 1 bevindt deze zich onderaan) die naar het volgende level leidt. Om naar de uitgang te geraken moet de vuurmier het doolhof doorlopen en de verschillende geblokkeerde gangen en/of deuren openen door de puzzels op te lossen.

C: Vuurmier Een vuurmier wordt bestuurt door de speler en kan zowel horizontaal als verticaal bewegen. De vuurmier kan alleen wandelen in de open ruimtes (gangen) van het doolhof (e.g., waar geen muren of gesloten deuren aanwezig zijn). Tijdens het wandelen kan een mier objecten oprapen, verplaatsen, afleggen en/of gebruiken op bepaalde zones in het doolhof. De toegelaten acties zijn sterk afhankelijk van de puzzels en power-ups die aanwezig zijn in het doolhof.

D: Schorpioen De schorpioenen zijn de vijanden van het spel. Ze proberen ervoor te zorgen dat

de vuurmier niet naar het volgende level kan gaan. Wanneer de schorpioen zich op dezelfde locatie bevindt als de vuurmier, dan verliest de vuurmier een leven en wordt het level herstart. Elk schorpioen wandelt door het doolhof en varieert in *wandelsnelheid* en *wandeltraject*. Er zijn twee types van schorpioenen, elk met hun eigen gedrag en een vaste kleur:

- Schorpioenen met een *vast traject* (geel). Dit traject bestaat uit een op voorhand ingesteld start- en eindpunt met eventuele tussenpunten. Eens de schorpioen het einde van zijn traject bereikt loopt hij de punten in omgekeerde volgorde af.
- Schorpioenen met een *willekeurig traject* (groen). Deze schorpioenen hebben alleen een startpunt en een willekeurig gegenereerde startrichting. Bij elk kruispunt kiest de schorpioen een nieuwe richting voor naartoe te wandelen en keert om wanneer de gang doodloopt.

Op willekeurige momenten tijdens het spel kan een schorpioen ook tijdelijk (e.g., 10 seconden) sneller bewegen. Wanneer dit gebeurt verandert deze ook tijdelijk van kleur (e.g., blauw, ongeacht de originele kleur). Vanaf zodra een schorpioen terug aan zijn normale snelheid verder beweegt, kan die gedurende een bepaalde tijd (e.g., 20 seconden) niet opnieuw terug sneller bewegen.

E: Puzzels Een puzzel is een klein probleem dat de vuurmier moet oplossen om delen van het doolhof te openen en zo uiteindelijk de uitgang toegankelijk te maken. Bijvoorbeeld in figuur 1 moet de vuurmier 3 paarse sleutels verzamelen om de rechthoekige deur in het midden van het level te openen. De simpelste puzzel is een waar de vuurmier een sleutel moet ophalen om een deur te openen. Elk level heeft zijn eigen moeilijkheidsgraad die de complexiteit van de puzzels (e.g., het aantal interacties van de speler) in een level bepaalt. Sectie 2.2.2 bevat een uitgebreide lijst van mogelijke puzzels die je kan implementeren.

F: Eitjes en Voedsel Verspreid doorheen de spelwereld liggen er eitjes en voedsel die opgeraapt kunnen worden door de vuurmier. Wanneer de vuurmier een eitje of voedsel aanraakt, dan verdwijnt deze uit de spelwereld en krijgt de speler extra punten. Je kan er voor kiezen om meer punten te geven voor voedsel dan eitjes, of omgekeerd.

G: Scorebord en levens Onderaan het scherm kan de speler zijn of haar huidige score aflezen, alsook de hoogste behaalde score ooit. Daarnaast staat ook het aantal levens dat de speler heeft en een indicatie van het level waar de speler zich in bevindt. Het spel begint met een hoeveelheid levens dat steeds afneemt wanneer de vuurmier botst tegen een schorpioen. Wanneer de speler geen levens meer heeft, dan is het spel afgelopen en moet het spel herstart kunnen worden (zonder hiervoor het programma opnieuw uit te voeren).

H: Power-ups Een power-up is een object dat een bepaald effect heeft op het spel. Deze objecten kunnen bijvoorbeeld invloed hebben op de score, het aantal levens van de speler, vijanden... Sectie 2.2.2 bevat een uitgebreide lijst van mogelijke power-ups die je kan implementeren.

I: Levels Een level kun je beschouwen als een moeilijkheidsgraad dat heerst over het spel. Als de speler succesvol het einde van het doolhof bereikt, moet hij overgaan naar het volgende level. Elk level heeft een eigen doolhofstructuur en bepaalde posities waar de schorpioenen, eitjes, voedsel en power-ups in teruggevonden kunnen worden, samen met de spelelementen die nodig zijn voor de puzzels. Deze configuraties mag je natuurlijk zelf kiezen per level. Wanneer

het laatste level uitgespeeld is, moet de speler hiervan op de hoogte gebracht worden (bv. door een tekst op het scherm te tekenen) waarna vervolgens het spel herstart kan worden (zonder hiervoor het programma opnieuw uit te voeren).

Je zal opmerken dat sommige interacties met opzet vaag gehouden zijn. We raden je dan ook aan om je creativiteit te gebruiken om een concrete invulling te geven aan deze interacties. Uiteindelijk is het de bedoeling dat je zelf ook plezier beleeft aan het ontwikkelen van je spel!

2.2 Implementatiefases

Het programmeerproject zal in twee fases opgeleverd worden. Bekijk de planning (in tabel 1) voor een overzicht van wanneer je iets moet indienen.

Deze sectie beschrijft de benodigde vereisten van elke fase.

2.2.1 Fase 1: Elementaire componenten en handelingen

In de eerste fase leg je de basis om een werkend spel te bekomen. Op het einde van fase 1 dient je spel de volgende functionaliteit te omvatten:

- Een spelwereld (A) dat bestaat uit een doolhof (B) en een vuurmier (C) die door de speler bestuurt kan worden.
- Minstens 1 schorpioen (D) die een vast pad heeft met een begin- en eindpunt. In de eerste fase is het voldoende om alleen de vuurmier te verplaatsen naar een veilige plaats (e.g., zijn startlocatie) wanneer deze een schorpioen aanraakt: in de eerste fase zijn er nog geen levens. Deze functionaliteit zal pas in de tweede fase geïmplementeerd moeten worden.
- Verspreid doorheen het level moeten er verschillende eitjes (F) geplaatst worden. In deze fase volstaat het om eitjes te ondersteunen die verdwijnen uit de spelwereld bij aanraking: de verhoging van de score moet nog niet geïmplementeerd worden

In deze fase is het belangrijk dat je het ontwerp van je spel juist krijgt, aangezien je in de tweede fase verder zal moeten bouwen op je ontwerp (en implementatie) van deze eerste fase. Je moet dus nog geen functionaliteit voorzien die pas in de tweede fase vereist is.

Voor het tekenen van de spelsituaties, en het uitlezen van de gebruikersinvoer, stellen we een bibliotheek (zie sectie 4) ter beschikking.

2.2.2 Fase 2: Naar een volledig spel

In de tweede fase breid je het project uit met de resterende functionaliteit. Naast de functionaliteit van fase 1, moet je project dus ook de volgende functionaliteiten ondersteunen:

- Meerdere schorpioenen (D) in elk level van beide types, inclusief het verliezen van een leven wanneer de vuurmier een schorpioen aanraakt (en het herstarten van het huidige level) alsook de functionaliteit dat schorpioenen op willekeurige momenten sneller doet laten bewegen.
- Minstens 2 soorten puzzels (E) uit onderstaande lijst. Vanaf zodra je 2 puzzels uit deze lijst geïmplementeerd hebt kan je ook je eigen puzzels bedenken en toevoegen aan je spel. Wanneer een puzzel, o.w.v. een foute actie van de speler, niet meer opgelost kan worden (i.e. soft-lock)

kan het nuttig zijn om een toets (of toetscombinatie) te voorzien waar het huidige level mee kan herstart worden!

  - Een deur dat alleen maar opent na gebruik van één of meerdere sleutels.
  - Dynamiet die een zwakke muur in het doolhof beschadigd. De vuurmier kan in het level dynamiet oprapen en deze naast de zwakke muur leggen. Wanneer deze ontploft verdwijnt de muur en kan de vuurmier zich naar de andere kant van het level verplaatsen. Let op! Wanneer de mier te dicht bij de dynamiet blijft staan wanneer deze ontploft kan deze een leven verliezen.
  - Een overstroomt gebied waar de vuurmier alleen over kan met een surfplank. De vuurmier gaat in het level eerst een surfplank moeten oprapen alvorens dat deze zich eenmalig over een overstroomt gedeelte van het doolhof kan navigeren. Wanneer de vuurmier de overkant bereikt heeft dan verdwijnt de surfplank. Deze kan dus niet opnieuw gebruikt worden.
  - Een steen die op een drukplaat geduwt moet worden alvorens een speciale deur, verbonden met de drukplaat, opengaat. Verspreid in het level liggen verschillende stenen die moeten kunnen bewegen wanneer de mier tegen de steen duwt (e.g., als de mier rechts naast een steen staat, en de mier beweegt naar links dan moet de steen meebewegen, tenzij links van de steen een muur, of een ander niet-doordringbaar spelelement, staat). Wanneer de steen op dezelfde positie staat als een drukplaat, dan kan ergens anders in het doolhof een speciale deur geopend worden.

- Het ondersteunen van verschillende soorten voedsel objecten (F). Deze moeten een effect hebben op de score.
- Het bijhouden van de score, de hoogste score en het aantal levens (G). Inclusief het mechanisme waarbij het spel herstart wordt wanneer de vuurmier geen levens meer heeft.
- Minstens 2 soorten power-ups (H) uit onderstaande lijst. Vanaf zodra je 2 power-ups uit deze lijst geïmplementeerd hebt kan je ook je eigen power-ups bedenken en toevoegen aan je spel.
  - Speciale voeding met een effect op het aantal levens.
  - Een schild dat de speler tijdelijk beschermt tegen schorpioenen en/of andere soorten gevaren en dat rechtstreeks geactiveerd wordt bij botsing met deze.
  - Een wapen dat schorpioen kan vernietigen. Bijvoorbeeld een pistool dat drie kogels heeft en gebruikt kan worden door op een toets (e.g., de spatie-toets) te drukken. De kijkrichting van de vuurmier bepaalt in welke richting er een kogel wordt afgeschoten.
  - Een vriendelijk stokstaartje dat bij aanraking met de vuurmier naar de meest dichtstbijzijnde schorpioen wandelt en deze opeet.
- Minstens 3 levels (I) die elk een ander doolhof hebben met verschillende puzzels. Inclusief het mechanisme waarbij het spel herstart wordt wanneer het laatste level bereikt is.

3 Beoordeling

Voor de beoordeling van het project wordt met drie aspecten rekening gehouden: de ingeleverde code, de bijhorende verslagen en de verdediging. Elke fase wordt apart beoordeeld.

3.1 Projectcode (70%)

De projectcode zelf wordt beoordeeld op zowel functionaliteit en codekwaliteit.

Voor het onderdeel **functionaliteit** zullen we nagaan, aan de hand van de beschreven spelinteracties, in hoeverre je de opdracht correct hebt geïmplementeerd.

Niet alleen de functionaliteit van je project is belangrijk maar ook hoe je deze functionaliteit geïmplementeerd hebt. In het onderdeel **codekwaliteit** kijken we na of je code niet overmatig complex is (i.e. geen spaghetticode) alsook hoe eenvoudig het is om de code te onderhouden (e.g., eenvoudige aanpassingen aan de spelfunctionaliteit zouden niet tot een overmatige explosie van aanpassingen mogen leiden in je code). Enkele basisprincipes:

- Vervuil de globale omgeving niet! Procedures en variabelen die bij een specifiek ADT horen mogen niet globaal opgeslagen worden. Elk stukje code heeft toegang tot de globale omgeving, en door hier overmatig van gebruik te maken is het vaak onduidelijk hoe bepaalde variabelen gebruikt/geüpdate worden. Om dit te vermijden kan je de principes van *encapsulatie* toepassen om de gebruikers van een ADT alleen toegang te geven tot de interne waardes van een object die beschikbaar mogen zijn.
- Als meerdere soorten objecten dezelfde soort operaties hebben (maar met een andere implementatie) kan je deze operaties best eenzelfde naam geven (dit principe noemen we *polymorfisme*). Hierdoor vermijd je onnodige tests voor de juiste operator toe te passen/te selecteren afhankelijk van het soort (type) object.
- Vermijd *bad smells* zoals *codeduplicatie* (waarbij je procedures of delen van procedures met dezelfde, of zeer gelijkaardige, implementatie meermaals ziet terugkomen in de code) en het gebruik van *magische constanten* (waarbij er constanten zitten in het midden van een proceduredefinitie, met een onduidelijke betekenis). Deze zorgen meestal voor code die moeilijk is om te onderhouden.
- Verder kijken we ook of er een duidelijke scheiding is van spellogica (een nieuwe spelstatus *berekenen*) en tekenlogica (de nieuwe spelstatus *tekenen*). Het moet eenvoudig zijn om beiden onafhankelijk van elkaar aan te passen. Zorg er dus bijvoorbeeld voor dat je spellogica werkt onafhankelijk van de afmetingen van het speelscherm.

3.2 Verslag (15%)

Het verslag is een belangrijk onderdeel van het project en dient als leidraad voor de beoordeling ervan. Je verslag moet verzorgd zijn: zowel taalgebruik, als het lay-out. Het moet een overzicht geven van de stand van zaken van je project met een focus op het finale ontwerp en diens implementatie. Zorg ervoor dat wat je in je verslag beschrijft ook effectief kan teruggevonden worden in de ingediende code. Als er bepaalde functionaliteit ontbreekt in je spel beschrijf je dit (maak het dus duidelijk wat wel en wat niet geïmplementeerd is). Sectie 5.2 bevat een overzicht van welke informatie we verwachten dat aanwezig is in je verslag.

3.3 Verdediging (15%)

Tijdens de verdediging van je project krijg je de gelegenheid een korte demonstratie te geven waarin je duidelijk kan maken dat alle functionele vereisten volledig geïmplementeerd zijn. Zorg er dus voor dat alle gevraagde functionaliteit eenvoudig kan getest worden. Maak je puzzels dus niet te moeilijk! Ook kan het handig zijn om enkele cheat-toetsen te voorzien waarmee eenvoudig de verschillende functionaliteiten mee getest kunnen worden tijdens de verdediging of waarmee het spel eenvoudiger mee getest kan worden. Je kan bijvoorbeeld een toets voorzien waarmee je makkelijk(er) naar het volgende level kan gaan of die de vuurmier (tijdelijk) onschendbaar maakt.

Na de demonstratie krijg je vragen over je code en is het aan jou om aan te tonen dat de keuzes die je gemaakt hebt hebben geleid tot kwalitatieve code. Gebruik deze verdediging ook als een persoonlijk feedbackmoment: de informatie die we je geven bij de verdediging kan namelijk nuttig zijn voor je in de toekomst (e.g., de feedback van de eerste fase voor zal nuttig zijn voor de tweede fase, en de feedback van de tweede fase voor PROGRAMMEERPROJECT 2 in de 2^{e} bachelor).

4 Implementatie

Het project moet geïmplementeerd worden in de programmeertaal Scheme, met name de variant R^{5}RS die ook in de cursus STRUCTUUR VAN COMPUTERPROGRAMMA'S 1 gebruikt wordt[1]. Implementeer elk ADT in een apart bestand en maak gebruik van `(load "filename.rkt")` om een bestand in te laden vanuit een ander bestand.

We verwachten dat een (on)eindige lus ("de spellus") de verschillende spelelementen zal aandrijven. Deze spellus is verantwoordelijk voor...

1. Het verwerken van toetsenbordinvoer van de speler.
2. Aan de hand van de acties van de speler, en het verloop van de tijd, een nieuwe spelsituatie te berekenen.
3. Spelsituaties tonen op het scherm aan de speler.

4.1 Grafische bibliotheek

Voor het inlezen van de spelinvoer, het verloop van de tijd op te vragen, en het tekenen van de spelsituatie kan er gebruik gemaakt worden van een grafische bibliotheek die gedownload kan worden in de cursusruimte op CANVAS ("Grafische Bibliotheek/Graphics.rkt"). Deze bibliotheek is ontwikkeld met dezelfde abstractiemechanismes die gebruikt moeten worden bij de ontwikkeling van het programmeerproject en voorziet reeds een spellus. Deze spellus kan uitgebreid worden met de functionaliteit die nodig is voor jouw implementatie door de juiste operaties van de bibliotheek op te roepen. De documentatie van de grafische bibliotheek (die de abstracties van de bibliotheek beschrijft) is ook beschikbaar op CANVAS ("Grafische bibliotheek/Documentatie.pdf") samen met een testbestand ("Grafische bibliotheek/test.rkt") die het gebruik ervan illustreert.

[1] In samenspraak met de assistenten kan ook een andere variant van Scheme (keuze beperkt tot R^{6}RS, R^{7}RS, en Racket) gebruikt worden. Gelieve ons hiervoor een e-mail te sturen tijdens de loop van het academiejaar (`programmeerproject1@soft.vub.ac.be`) met motivatie. Indien je een andere programmeertaal gebruikt, zonder ons hiervan tijdig op de hoogte gebracht te hebben, kunnen er minpunten in rekening gebracht worden.

5 Documenten

Tijdens de loop van het academiejaar zal je op regelmatige tijdstippen een document moeten inleveren. Deze sectie beschrijft wat we van deze documenten verwachten.

Voorzie elk document van een voorblad en een inhoudsopgave. Vermeld op het voorblad: je naam, je studentennummer (rolnummer), je VUB e-mailadres en het academiejaar. Voorzie elk document ook van paginanummers.

5.1 Voorstudie

Een voorstudie (maximum 10 pagina's, inclusief voorblad en inhoudsopgave) is een uitgeschreven rapport waarin je beschrijft hoe je bepaalde vereisten zal implementeren. Geef aan het begin van je voorstudie een overzicht van welke functionaliteit ondersteund moet worden in de betrokken fase, en welke ADT's je verwacht dat daarvoor geïmplementeerd moeten worden.

Geef voor elk ADT een overzicht van alle operaties (inclusief de typesignaturen van deze operaties), en een beschrijving van wat elke operatie doet. Wanneer er meerdere gelijkaardige operaties zijn (e.g., zoals `rotate-clockwise` en `rotate-counterclockwise` in de documentatie van de grafische bibliotheek), mogen deze gezamenlijk uitgelegd worden. Wanneer er procedures zijn waarvan je verwacht dat je ze gaat implementeren, maar ze maken geen deel uit van het ADT zelf (met andere woorden, het kan niet rechtstreeks door een gebruiker van het ADT opgeroepen worden), dan moet het niet beschreven worden in je voorstudie! Daarnaast beschrijf je welke andere ADT's of ingebouwde Scheme-types jouw ADT encapsuleert, en de relatie tussen een ADT en de andere ADT's in je spel.

Maak gebruik van een afhankelijkheidsdiagram (zie sectie 5.3) om het algemene ontwerp van je ADT's duidelijk te maken. Om je afhankelijkheidsdiagram duidelijk te maken leg je ook uit hoe dit tot stand is gekomen door de relaties tussen de verschillende ADT's uit te leggen.

Verder moet je voorstudie ook bestaan uit een planning waarin staat wat je elke week verwacht te doen voor het project. Wees hier concreet in! Schrijf dus niet *"Week 12 – Spel implementeren"*, maar eerder *"Week 12 – Implementeren Positie ADT en Level ADT."*. Benoem bij voorkeur aan welke ADT's je gaat werken en/of welke functionaliteiten je gaat toevoegen aan de verschillende ADT's.

De voorstudie in fase 2 mag verder bouwen op het verslag van fase 1. Dat betekent dat enkel de verwachte aanpassingen voor fase 2 beschreven moeten worden. Bijvoorbeeld een ADT dat geïmplementeerd werd in fase 1, en waarvan je verwacht dat je het niet zal aanpassen in fase 2, moet niet opnieuw in detail beschreven worden in de voorstudie van fase 2.

5.2 Verslag

Een verslag (maximum 20 pagina's, inclusief voorblad en inhoudsopgave) is een document waarin je de huidige stand van zaken van je project beschrijft. Beschrijf welke functionaliteit je voorzien hebt en welke aanpassingen je verwacht te maken om de ontbrekende functionaliteit te implementeren (indien van toepassing).

Net zoals bij een voorstudie geef je een beschrijving van de ADT's: geef een korte beschrijving van elk ADT (wat doet het, wat stelt het voor) alsook een beschrijving van diens operaties (inclusief signaturen). Beschrijf echter in je verslag alleen wat ook effectief kan teruggevonden worden in je code.

Indien je ook ontbrekende functionaliteit wil bespreken, dan moet het duidelijk zijn wat aanwezig is, en wat niet.

Voeg ook aan je verslag een afhankelijkheidsdiagram (zie sectie 5.3) toe en een overzicht van wanneer je aan het project gewerkt hebt (een tijdsschema). Vergelijk zowel het afhankelijkheidsdiagram en de planning in je verslag met die van je voorstudie (bespreek dit beknopt). Identificeer welke stappen in je proces meer (of minder) tijd in beslag namen dan wat je eerst verwacht had.

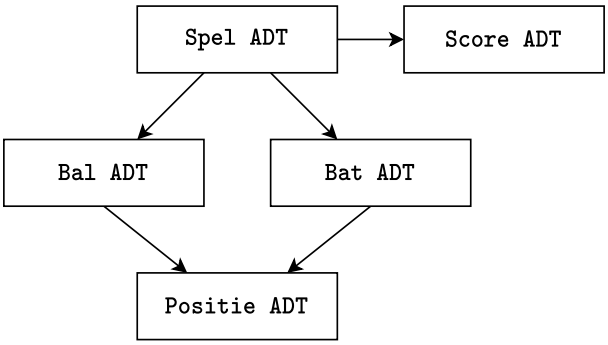
Vergeet ook niet om aan je verslag een korte beschrijving toe te voegen die uitlegt hoe je spel opgestart kan worden (bijvoorbeeld door uit te leggen welk bestand uitgevoerd moet worden met DrRacket).

Elk verslag dat op het einde van een fase ingeleverd moet worden zal als een losstaand document beschouwd worden. Dat betekent dat alle essentiële informatie moet terug te vinden zijn zonder andere documenten te moeten openen (uiteraard mag je naar andere documenten verwijzen indien je dit relevant acht). Dit betekent ook dat, in tegenstelling tot de voorstudie, het verslag van fase 2 alle ADT's moet bespreken (ook de ADT's van de eerste fase!).

Je mag aan je verslag een overzicht van je code toevoegen als bijlage/appendix (wat uiteraard niet meetelt bij de paginalimiet!): zorg er dan wel voor dat deze voorzien is van syntax highlighting (zoals in DrRacket) en van lijnnummers.

5.3 Afhankelijkheidsdiagram

Een afhankelijkheidsdiagram is een grafische manier om het ontwerp van je project mee te illustreren. In een afhankelijkheidsdiagram worden de relaties tussen de verschillende ADT's van je spel voorgesteld door middel van rechthoeken (doosjes) en pijlen. Een rechthoek stelt een ADT voor, en een pijl tussen twee rechthoeken stelt een afhankelijkheid tussen de twee ADT's voor (een pijl van A naar B betekent doorgaans dat A afhankelijk is van B). Een voorbeeld van een afhankelijkheidsdiagram (voor een fictieve implementatie van een Pong-spelletje) kan teruggevonden worden in figuur 2.



Figuur 2: Een voorbeeld van een afhankelijkheidsdiagram.

Dit afhankelijkheidsdiagram bevat vijf ADT's: een `Spel ADT`, een `Score ADT`, een `Bal ADT`, een `Bat ADT` en een `Positie ADT`.

Los gezegd is een `ADT A` afhankelijk van een `ADT B` als een object van `A` een object van `B` nodig heeft om zelf te kunnen functioneren. Concreet betekent dit dat een `ADT A` afhankelijk is van `ADT B` als

minstens één van de volgende voorwaarden voldaan is.

- In de implementatie van `ADT A` wordt er een operatie van `ADT B` gebruikt.
- In de implementatie van `ADT A` is er een variabele waarin een waarde met type `ADT B` in bijgehouden wordt.
- In één van de operaties van `ADT A` is er een operatie waarvan er verwacht wordt dat een parameter gebonden zal worden aan een waarde van type `ADT B`; of waarvan de teruggegeven waarde verwacht wordt om een waarde van type `ADT B` te zijn.

Bijvoorbeeld: De afhankelijkheid tussen `Spel ADT` en `Score ADT` (in figuur 2) zou verklaard kunnen worden omdat er in de implementatie van het Spel ADT er een variabele kan zijn die gebruikt wordt om een referentie bij te houden naar een `Score ADT`. De afhankelijkheid van zowel `Bal ADT` en `Bat ADT` op het `Positie ADT` zou verklaard kunnen worden wanneer beiden een operatie hebben die een waarde van het `Positie ADT` kan teruggeven.

Het afhankelijkheidsdiagram bevat alleen maar ADT's, inclusief de ADT's van de grafische bibliotheek (de ADT's van de grafische bibliotheek mogen, bij uitzondering, gegroepeerd worden). Indien je implementatie gebruik maakt van globale variabelen of procedures die niet in een ADT geëncapsuleerd zijn, dan moeten deze niet toegevoegd worden aan het afhankelijkheidsdiagram.

Zowel voorstudies als verslagen moeten een afhankelijkheidsdiagram bevatten. Aangezien er nog geen concrete implementatie is wanneer je aan je voorstudie werkt toont je diagram in deze documenten de afhankelijkheden waarvan je vermoedt dat ze aanwezig zullen zijn in je implementatie.

6 Praktische Afspraken

6.1 Plagiaat

Het programmeerproject is een **individuele** opdracht. Het is niet toegestaan om je project (of delen van je project) uit te besteden aan een derde, noch om projecten die je online (of elders) vindt te kopiëren en aan te passen. Ook voorstudies en verslagen moeten individueel gemaakt worden. Het materiaal dat beschikbaar is in de cursusruimte van PROGRAMMEERPROJECT 1 op CANVAS mag uiteraard gebruikt worden als basis van je ontwerp of van je code, maar het is niet de bedoeling om rechtstreeks een bestaand ontwerp of een bestaande implementatie van iemand anders over te nemen en deze (eventueel) aan te passen voor het project van dit academiejaar. Dit beschouwen we namelijk als plagiaat.

Wanneer je code wil gebruiken die je niet zelf geschreven hebt (bijvoorbeeld code die je online, in een boek, of in een cursustekst hebt gevonden) ben je verplicht om de bron te vermelden. Doe dit zowel in de code (bijvoorbeeld door er commentaar bij te plaatsen), alsook in je verslag (bijvoorbeeld in een extra sectie op het einde waarin je de bronnen die je gebruikt hebt vermeld). Wanneer je code gebruikt die je niet zelf geschreven hebt, zonder de bron te vermelden, beschouwen we dit ook als plagiaat.

Samenwerken met je medestudenten is, tot op een zeker niveau, toegestaan. Je mag ideeën uitwisselen met je mede-studenten en ook anderen om hulp vragen om een probleem met jouw code op te lossen. Wat natuurlijk niet kan is om met meerdere studenten tegelijkertijd aan hetzelfde project te werken, of om rechtstreeks code met elkaar te delen en te hergebruiken.

	Actie	Wanneer
	Projectvoorstelling	Week 8
Fase 1	**Indienen voorstudie**	**Week 10, vrijdag 26 november 2021**
	WPO "Programmeren met de Grafische Bibliotheek"	Week 11–12
	Ontvangen feedback voorstudie	Week 11–12
	WPO "Massaprogrammeren"	Week 13
	Tussentijds indienmoment	**Week 14, maandag 20 december 2021**
	Wintervakantie	Week 15–16
	Examenperiode en lesvrije week	Week 17–21
	Tussentijds indienmoment	**Week 22, maandag 14 februari 2022**
	WPO "Massaprogrammeren"	Week 22
	Indienen code en verslag	**Week 23, maandag 21 februari 2022**
	Projectverdedigingen	**Week 23**
Fase 2	**Indienen voorstudie**	**Week 26, maandag 14 maart 2022**
	Ontvangen feedback voorstudie	Week 26–27
	WPO "Massaprogrammeren"	Week 27
	Tussentijds indienmoment	**Week 28, vrijdag 1 april 2022**
	Lentevakantie	Week 29–30
	Tussentijds indienmoment	**Week 34, maandag 9 mei 2022**
	WPO "Massaprogrammeren"	Week 34
	Blokweek	Week 37
	Indienen code en verslag	**Week 37, maandag 30 mei 2022**
	Projectverdedigingen	**Zie examenrooster**

Tabel 1: Planning PROGRAMMEERPROJECT 1 in academiejaar 2021–2022. De deadline van elk indienmoment is steeds om 23:59.59 (middernacht). De weeknummers verwijzen naar de academische kalender: `https://www.vub.be/academische-kalender`.

6.2 Planning

De verschillende deadlines van het project kunnen in tabel 1 teruggevonden worden. Alle acties in het **vet** zijn deadlines van het project waarbij je ofwel iets moet inzenden op CANVAS (een voorstudie, code en/of een verslag), of waarbij je aanwezig moet zijn (een projectverdediging).

Elke deadline van het project is verplicht! Dat betekent dat je voor beide fases **A.** een voorstudie moet indienen, **B.** tweemaal je tussentijdse voortgang moet indienen, **C.** je uiteindelijke code (met bijhorend verslag) moet indienen en **D.** jouw project moet verdedigen. Iets niet indienen, of niet opdagen bij een verdediging resulteert onmiddellijk in een afwezig voor het ganse project.

Een laattijdige inzending is slechts eenmaal per fase toegestaan. Een laattijdige inzending moet binnen de 24 uur[a] na de deadline ingediend worden via het algemene e-mailadres (`programmeerproject1@soft.vub.ac.be`). Het totaal van de betrokken fase zal echter slechts 75% van de origineel behaalde punten zijn. Voorbeeld: als je normaal een 16/20 zou gehaald hebben voor een bepaalde fase, dan zal je punt van de betrokken fase gewijzigd worden in een 12/20. In uitzonderlijke situaties (e.g., overmacht) kan er van deze regel afgeweken worden.

[a] 48 uur voor de tussentijdse indienmomenten.

6.2.1 Massaprogrammeren

Naast het WPO in weken 11–12 waarbij het gebruik van de grafische bibliotheek (zie sectie 4.1) zal uitgelegd worden, zullen er ook 4 WPOs massaprogrammeren ingepland worden. Bij deze vrijblijvende lesmomenten krijg je de mogelijkheid om aan je project te werken en om vragen over het project te stellen aan je mede-studenten en de assistenten. Ook als je zelf geen vragen hebt kan het interessant zijn om hierbij aanwezig te zijn aangezien sommige vragen van je mede-studenten misschien ook interessant kunnen zijn voor jou.

6.2.2 Tussentijdse indienmomenten

De planning in tabel 1 bevat in totaal vier tussentijdse indienmomenten waarbij je jouw project, zoals je het op dat moment op je computer hebt staan, moet indienen op CANVAS. **De code die je dan indient zal niet beoordeeld worden: niet op functionaliteit en ook niet op codekwaliteit!** Na elk tussentijds indienmoment publiceren we op CANVAS een samenvatting van hoeveel jullie gemiddeld aan het project gewerkt hebben (e.g., aantal toegevoegde lijnen code). Deze berekening zal automatisch gebeuren [2], en de resultaten zullen volledig anoniem gedeeld worden op CANVAS.

Merk op! **De tussentijdse indienmomenten zijn verplicht!** Als je nog niet aan het project gewerkt hebt, upload je dus een leeg (zip-)bestand (of, als je tussen twee indienmomenten niet verder gewerkt hebt aan je project, je vorige ingediende zip-bestand).

6.2.3 Indienen

Voorstudies en verslagen moeten in PDF-formaat geüpload worden op CANVAS. **We aanvaarden geen Microsoft Word, Pages, LibreOffice of andere bestandsformaten**. Bij de tussentijdse indienmomenten en het indienmoment op het einde van elke fase verwachten we dat alle code-bestanden samengebundeld worden in één enkel ZIP-archief. **We aanvaarden geen RAR's, 7Z's, ISO's, TAR's, DMG's of andere archiefformaten.**

Voor het indienden van jullie code en verslag zullen er twee uploadmodules aangemaakt worden: één slot voor jullie code in een ZIP-archief en in het andere het verslag als PDF. Controleer dus steeds of je alle benodigde bestanden toegevoegd hebt aan je ZIP-archief en dat het verslag in PDF-formaat is opgeslagen. In CANVAS is het mogelijk om het bestand dat je geüpload hebt te vervangen met een nieuwe versie, zolang de deadline nog niet verstreken is. Maak hier gebruik van door na het uploaden zelf je bestand te downloaden, en te controleren of alle bestanden correct toegevoegd zijn. Enkel het laatste bestand geüpload op CANVAS zal bekeken worden!

6.3 Vragen

Alle vragen in verband met het project kunnen gesteld worden naar het onderstaande e-mailadres (maak gebruik van je VUB e-mail!). E-mails die gestuurd worden naar dit e-mailadres zullen in een gedeelde inbox terechtkomen die bekeken zal worden door het volledige onderwijsteam.

`programmeerproject1@soft.vub.ac.be`

Stuur dus geen berichten via CANVAS, noch e-mails naar onze persoonlijke e-mailadressen. Mogelijks ontvang je geen antwoord op deze berichten.

[2] Met behulp van een aangepaste versie van `cloc` (`https://github.com/AlDanial/cloc`).