# Programmeerproject 1:
# Verslag fase 1 "Tower Defence"

Mathis Arthur Boumaza
0555462
Mathis.Arthur.Boumaza@vub.be
Bachelor in de Computerwetenschappen

Academiejaar 2022-2023

# Inhoudsopgave

# 1 Inleiding

Een onderdeel van de Bachelor in de Computerwetenschappen, bestaat eruit om een programmeerproject te maken waarin verwacht word een spel te maken. De opdracht dit jaar is om een tower defence spel te maken. Zo'n spel bestaat uit een spelwereld met een pad erop waarop verschillende soorten monsters kunnen lopen doorheen verschillende levels en ronden die in moeilijkheidsgraad stijgen naarmate het spel vordert. Deze monsters moeten vermoord worden vooraleer ze het einde van het pad bereiken. Om ze te vermoorden word gebruik gemaakt van verschillende soorten torens die een soort projectiel schieten indien er monsters in de buurt zijn. Deze torens kan door de speler in de spelwereld geplaats zolang het niet op het pad is of op een andere toren. Indien het de speler niet lukt om te vermijden dat monsters het einde van het pad bereiken, zal hij levens verliezen en mogelijks sterven, wat het einde van het spel betekent.

Om tot een volwaardig spel te komen werd verwacht om voor fase 1 verschillende functionaliteiten te implementeren om zo tot een basis spel te komen waarop mooi verder gebouwd kan worden in de tweede fase. Om te starten, heb ik aan de hand van het Spel ADT een spelwereld voorgesteld en die onmidellijk getekend met het Teken ADT. Hierop zullen alle animaties mooi getekent worden. Daarna werd er een pad ADT aangemaakt om zo paden voor te kunnen stellen waarop de monsters kunnen wandelen. Het Teken ADT werd dan aangepast om pad objecten te kunnen tekenen op de spelwereld. Natuurlijk wil een speler zijn pad kunnen verdedigen, dit kan door torens te plaatsen in de spelwereld. Daardoor werd een toren ADT geimplementeerd om toren objecten te kunnen voorstellen en die dan te kunnen plaatsen op de spelwereld. Om deze torens te kunnen plaatsen werd het spel ADT voorzien van een functionaliteit die toelaat om op de spelwereld te klikken en zo torens te plaatsen. Evenzeer om deze torens niet overal te kunnen plaatsen, werden constraints geplaats op waar en wanneer je ze kan plaatsen in het Spel ADT. Waar heeft betrekking tot het feit dat je ze niet kan plaatsen op paden en wanneer heeft betrekking tot, enkel als je een toren gekozen hebt en geklikt hebt kan je deze soort toren beginnen plaatsen op de spelwereld. Opnieuw wil ik deze torens visualizeren dus heb ik mijn Teken ADT aangepast om dit toe te staan. Nu dat men een pad heeft en torens kan plaatsen willen we nog monsters doen lopen op een pad en deze neerschieten indien ze in de buurt zijn van een toren. Om dit te implementeren zijn verschillende zaken nodig. Meer bepaald moet men een monster kunnen voorstellen, dit doen we aan de hand van het monster ADT. Daarnaast heb ik ook een projectiel ADT om projectielen te kunnen voorstellen en zo monsters te vermoorden. Deze projectielen beginnen van de toren van oorsprong en bewegen naar een monster op het pad. Om dit nu allemaal te visualizeren werd het Teken ADT aangepast om projectielen en monsters te visualizeren. We kunnen nu alle soorten spelelementen voor stellen, maar er zit nog geen dynamica in het spel. Daarvoor heb ik een level ADT gemaakt die ons in staat stelt om een level object te maken en zo ervoor zorgt dat monsters beetje per beetje spawnen en voort bewegen van het begin van het pad tot het eind van het pad. Het zal er voor zorgen dat, indien monsters in de buurt komen van een toren, dat ze neergeschoten worden door een projectiel en dus verdwijnen, en dus ook de projectielen die aangekomen zijn verdwijnen. Het Teken ADT werd aangepast om er dan voor te zorgen dat de tiles meebewegen met de veranderde upgedate posities van de monsters en projectielen en de verdwenen monsters en projectielen effectief niet meer op het scherm gevisualiseerd zijn. Nu om de level werkelijk te starten werd nog een knop functionaliteit geimplementeerd in het Spel ADT die ons toelaat om werkelijk het spel te starten.

Het document zal meer in detail gaan over hoe ik deze verschillende functionaliteiten tot stand doen komen heb. Meer bepaald zal ik het hebben over de ADTs die ontwikkelt zijn en hun operaties en hoe ze gerelateerd zijn met elkaar door middel van een afhankelijkheidsdiagram. Evenzeer zal een overzicht gegeven worden over wanneer wat gedaan werd en ook zullen de verschillen in afhankelijkheidsdiagram en planning tussen de voorstudie en de huidige stand van zaken besproken worden. En om af te sluiten zal een beknopte uitleg volgen over hoe men het spel kan opstarten.

# 2 ADTs

Om de functionaliteiten te implementeren zijn, zoals uitgelegd, veel ADTs gebruikt geweest om tot een volwaardig spel te komen. Hier worden de ADTs en hun operaties beschreven alsook de operaties hun signaturen uitgelegd.

## 2.1 Positie ADT

Vermits verschillende functionaliteiten de notie van positie en beweging nodig hebben, bijvoorbeeld een monstertje beweegt vooruit of een toren heeft een bepaalde positie, heb ik een ADT gemaakt die het mogelijk maakt om posities op een 2 dimensional vlak voor te stellen. Dit ADT zal bij aanmaak 2 number datatypes samennemen die dan een positie in een vlak voorstelt. Verschillende operatoren werken op zo'n positie object, namelijk men kan de coordinaten ervan opvragen en veranderen, maar men kan ook vragen als een positie gelijk is aan een andere gegeven positie. Dit ADT vormt de basis voor vele andere ADTs.

| **Naam** | **Signatuur** |
|---|---|
| maak-positie-adt | (number, number → positie) |
| x | number |
| y | number |
| x! | (number → ∅) |
| y! | (number → ∅) |
| gelijk? | (positie → boolean) |
| ceil | (∅ → positie) |
| flo | (∅ → positie) |
| positie-copieer | (∅ → positie) |

Tabel 1: De operaties bevat in het Positie ADT

De operaties van het positie ADT zijn:

- *maak-positie-adt* is de aanmaakoperatie die toelaat om een positie voor te stellen in een 2 dimensionale ruimte. Het neemt 2 getallen als invoer en zal een positie ADT maken.
- $x$ en $y$ zijn de operaties om respectievelijk de x en y coordinaat van het positie object te verkrijgen
- *x!* en *y!* zijn operaties om respectievelijk de x en y coördinaat van het positie object te veranderen naar een vooropgegeven waarde.
- *gelijk?* is een operatie die nagaat als het positie object gelijk is aan een ander vooropgegeven positie object. We zeggen dat 2 positie objecten gelijk zijn indien hun respectievelijke x en y waarden gelijk zijn. Uit deze operatie komen booleans, namelijk #t als ze gelijk zijn en #f als ze niet gelijk zijn.

- *ceil* en *flo* zijn operaties die op basis van het beschouwde positie object, een nieuw positie object aanmaken maar met de coordinaten respectievelijk naar boven en naar beneden afgerond.
- *positie-copieer* is een operatie om het huidige positie object "te copieren", meer bepaald zal het een nieuw positie object aanmaken met dezelfde coordinaten als het beschouwde positie object.

## 2.2 Pad ADT

Het pad ADT zal een pad voorstellen waarop de monstertjes zullen wandelen om tot de basis te geraken. Het is dit pad object dat de speler zal moeten beschermen met behulp van torens die geplaatst kunnen worden door de speler. Dit ADT is een abstractie bovenop het positie ADT die posities zal bijhouden zodanig dat monsters over dat pad kunnen lopen en dat er geen torens op gebouwd kunnen worden.

| **Naam** | **Signatuur** |
|---|---|
| maak-pad-adt | (lijst → pad) |
| posities | vector |
| lengte | number |
| inflectie-punten | pair |
| infectie-tekens | pair |
| begin | positie |
| einde | positie |
| toren-in-pad? | (toren → boolean) |

Tabel 2: De operaties bevat in het Pad ADT

De operaties van het pad ADT zijn:

- *maak-pad-adt* is de aanmaakoperatie van het ADT, het zal het pad object aanmaken. Om dit te doen zal het een lijst van 3 elementen innemen die respectievelijk, het aantal inflectie punten (punten waarbij het verloop van het pad van richting veranderd), de inflectie tekens (tekens die zeggen in welke zin het verloop van het pad veranderd is) en de werkelijke vector van posities van het pad voorstelt.
- *posities* is een operatie die een vector bestaande uit de posities van het pad object teruggeeft.
- *lengte* is een operatie die de lengte van de vector van posities waarop het pad object gebouwd is, teruggeeft.
- *inflectie-punten* is een operatie die de lijst van inflectie punten van het pad teruggeeft.
- *inflectie-tekens* is een operatie de de lijst van inflectie tekens van het pad teruggeeft.
- *begin* en *einde* zijn operaties die respectievelijk de begin en einde positie van het pad teruggeven.
- *toren-in-pad?* is een operatie die, gegeven een toren, nagaat als die toren zich bevind op het pad. Het resultaat van deze operatie is #t als de toren zich op het pad bevind en #f als de toren zich niet op het pad bevind.

## 2.3 Monster ADT

Het monster ADT laat toe om monsters voor te stellen die op het pad zullen lopen en het einde proberen te bereiken. Dit ADT is een abstractie bovenop het positie ADT en pad ADT. Het positie ADT wordt gebruikt

om de positie van het monster object bij te houden, maar deze posities zijn dan beperkt tot posities op het pad, wat betekent dat het ook gebouwd is op het pad ADT.

| **Naam** | **Signatuur** |
|---|---|
| maak-monster-adt | (symbol, pad → monster) |
| positie | positie |
| type | symbol |
| volgende-positie! | (∅ → ∅) |
| einde? | boolean |
| gestorven? | boolean |
| verander-levens! | (∅ → ∅) |
| soort | symbol |

Tabel 3: De operaties bevat in het Monster ADT

De operaties van het monster ADT zijn:

- *maak-monster-adt* is de aanmaakoperatie die toelaat om een monster object aan te maken. Het neemt 2 dingen als invoer, het type van monster, en een pad.
- *positie* is een operatie die de huidige positie van het monster object teruggeeft.
- *type* is de operatie die het type van het monster object teruggeeft.
- *volgende-positie!* is een operatie die toelaat om het monster object naar de volgende positie te laten gaan, dit is gebaseerd op het pad ADT.
- *einde?* is de operatie die nagaat als het monster object de eind positie bereikt heeft, namelijk het einde van het pad.
- *gestorven?* is een operatie die nagaat als het monster object gestorven is.
- *verander-levens!* is een operatie die het aantal levens van het monster object zal doen zakken met 1.
- *soort* is een operatie die het soort van het object teruggeeft (in dit geval een monster symbool).

## 2.4 Toren ADT

Het toren ADT zal een toren voorstellen die projectielen zal schieten naar dichtbijzijnde monsters op het pad. Het ADT is opnieuw een abstractie die gebouwd is bovenop het positie ADT. Het positie ADT zal gebruikt worden om de toren zijn positie bij te houden.

De operaties van het toren ADT zijn:

- *maak-toren-adt* is de aanmaakoperatie die toelaat om een toren object aan te maken. Het neemt een positie als invoer, die de positie van het toren object voorstelt in de spelwereld, en een type die het type toren voorstelt.
- *positie* is een operatie die de positie van de toren in de spelwereld teruggeeft.
- *type* is een operatie die het type van de toren teruggeeft.

| Naam | Signatuur |
|---|---|
| maak-toren-adt | (positie, symbol → toren) |
| positie | positie |
| type | symbol |
| toren-posities | vector |
| in-toren? | (toren → boolean) |
| in-buurt? | (monster → boolean) |
| schiet! | (monster → ∅) |
| projectie-update! | (∅ → ∅) |
| projectielen | pair |
| soort | symbol |

Tabel 4: De operaties bevat in het Toren ADT

- *toren-posities* is een operatie die 4 posities teruggeeft, die de toren posities voorstellen. We doen dit omdat 1 positie niet voldoende is om een toren zijn positie voor te stellen, vermits een toren groter is dan 1 centrale positie.
- *in-toren?* is een operatie die nagaat als een bepaalde gegeven toren in de toren zit, meer bepaald het kijkt voor overlap van posities tussen de 2 torens. het geeft een #t terug als de torens overlap hebben, en #f als dit niet zo is.
- *in-buurt?* is een operatie die nagaat als een bepaalde monster in de buurt van de toren zit, het zal #t teruggeven als de monster zich in de buurt bevind, en #f als dit niet zo is.
- *schiet!* is de operatie die toelaat om een toren te doen schieten naar een monster op het pad indien het monster in de buurt van de toren is.
- *projectiel-update!* is een operatie die al de projectielen die de toren geschoten heeft en die nog niet aangekomen zijn aan hun bestemming, hun posities updaten om zo dichter te komen bij het monster die ze moeten raken.
- *projectielen* is de operatie die al de afgevuurde projectielen teruggeeft die nog niet toegekomen zijn bij het monster die ze moeten raken.
- *soort* is een operatie die het soort van het object teruggeeft (in dit geval een toren).

## 2.5 Projectiel ADT

Het projectiel ADT laat toe om projectielen voor te stellen in ons spel en deze door toren objecten te laten gebruiken om monster objecten te vermoorden. Dit ADT is een abstractie bovenop het positie en monster ADT. Het positie ADT zal de positie van het projectiel object bijhouden alsook de bestemming positie ervan. Daarbij zal het ook verder bouwen op het monster ADT om bij het naar afgevuurd monster zijn positie aan te komen en hem te vermoorden.

De operaties van het projectiel ADT zijn:

- *maak-projectiel-adt* is de aanmaakoperatie die ons in staat stelt om een projectiel object aan te maken. Het neemt 3 dingen als invoer, namelijk de begin positie van het projectiel object, het type van projectiel en een monster object.

| Naam | Signatuur |
|---|---|
| maak-projectiel-adt | (positie, symbol, monster → projectiel) |
| positie | positie |
| type | symbol |
| te-raken-monster | monster |
| bestemming-bereikt? | (∅ → boolean) |
| volgende-positie! | (∅ → ∅) |
| soort | symbol |

Tabel 5: De operaties bevat in het Projectiel ADT

- *positie* is de operatie die de huidige positie van het projectiel zal teruggeven.
- *type* is de operatie die het type afgevuurd projectiel teruggeeft.
- *te-raken-monster* is een operatie die het monster object teruggeeft waarnaar het projectiel afgevuurd is.
- *bestemming-bereikt?* is een operatie die nagaat als het projectiel object zijn bestemming al bereikt heeft, die bij de aanmaak bepaald wordt. Het resultaat van deze operatie is #t indien het projectiel object zijn bestemming bereikt heeft en #f indien dit niet zo is.
- *volgende-positie!* is de operatie die het projectiel object zijn positie naar zijn volgende positie zal zetten. De volgende positie wordt bepaald door bij de aanmaak van het object constanten te maken, a.d.h.v de bestemming en start positie, die de verandering van positie zullen voorstellen en deze dan iedere keer op te tellen bij de huidge positie van het projectiel object.
- *soort* is een operatie die het soort van het object teruggeeft (in dit geval een projectiel)

## 2.6 Level ADT

Het level ADT zal een level van het spel voorstellen. Het zal een heel level beheren van start tot einde en zal al de elementen die geupdate moeten worden updaten. Dit ADT is gebouwd op verschillende andere ADTs, namelijk het monster ADT om constant de monsters hun posities up te daten, het pad ADT om de monsters op het begin van het pad te zetten en ze vandaaruit laten verder wandelen tot het einde, en als laatste het toren ADT, om torens te plaatsen maar ook om naar de monsters te schieten indien ze dicht genoeg bij de gezette torens zijn.

De operaties van het level ADT zijn:

- *maak-level-adt* is de aanmaakoperatie die toelaat om een level object aan te maken. Het neemt op zijn minst 1 ding als invoer, namelijk een lijst van monster types om monsters te doen spawnen. Daarboven op is het ook mogelijk om nog meer argumenten toe te voegen, meer bepaald torens (Indien men naar het volgende level wil gaan moet het mogelijk zijn om de al gezette torens mee te nemen).
- *Pad* is de operatie die toelaat om het pad van de level terug te krijgen.
- *monsters* is de operatie die de gespawnde monsters teruggeeft in een lijst.

[1]Alles die na het puntje komt stellen optionele parameters voor, in mijn geval verwacht ik maar maximum 1 optionele parameter. Vermits ik niet wist hoe ik deze signatuur moest neerschrijven heb ik het zo gedaan.

| Naam | Signatuur |
|---|---|
| maak-level-adt | (pair . pair → level)[1] |
| pad | pad |
| monsters | pair |
| torens | pair |
| voeg-toren-toe! | (toren → ∅) |
| update-monsters! | (. symbol → ∅) |
| update-torens-projectielen-positie! | (∅ → ∅) |
| update-torens-projectielen-afschieten! | (∅ → ∅) |
| verkrijg-projectielen | (∅ → pair) |
| einde? | (∅ → boolean) |
| level-einde! | (∅ → ∅) |

Tabel 6: De operaties bevat in het Level ADT

- *torens* is de operatie die de gezette torens op de spelwereld teruggeeft in een lijst.
- *voeg-toren-toe!* is de operatie die toelaat om torens toe te voegen tot het level object.
- *update-monsters!* is de operatie die zal zorgen dat de monster zullen voortbewegen en zo naar het einde van het pad geraken.
- *update-torens-projectielen-positie!* is de operatie die de afgeschoten projectielen hun posities zal updaten, om zo voort te bewegen naar hun bestemming.
- *update-torens-projectielen-afschieten!* is de operatie die er voor zal zorgen dat torens zullen schieten naar het eerste monster dat ze tegenkomen, namelijk het eerste monster dat in hun buurt is.
- *verkrijg-projectielen* is de operatie die al de afgevuurde projectielen zal teruggeven in een lijst.
- *einde?* is de operatie die nagaat als het level object het einde bereikt heeft. Het einde van een level is namelijk de situatie waarin alle monsters van het pad verdwenen zijn, ofwel omdat ze neergeschoten zijn, of omdat ze het einde bereikt hebben.
- *level-einde!* is de operatie die het level beindigt. Dit is niet nodig voor het maken van het tower defence spel, maar is gewoon een cheat operatie.

## 2.7 Teken ADT

Het teken ADT is het ADT die het mogelijk zal maken om de voorgaande objecten op het scherm te tekenen, namelijk de spelwereld, het pad, de monsters, de toren en de projectielen. Daarbovenop is het ook het ADT die het mogelijk zal maken om bepaalde toetsen en muisklik operaties in te stellen. Dit ADT is een abstractie gebouwd op de grafische bibliotheek van Scheme. Het gebruikt deze bibliotheek om alles te tekenen op het scherm.

De operaties van het teken ADT zijn:

- *maak-teken-adt* is de aanmaakoperatie die het teken object zal aanmaken, en dus het venster waarop we tekenen aanmaakt, waarnaar we boodschappen kunnen sturen om bepaalde elementen te tekenen op het scherm en om bepaalde toetsen in te stellen. Deze operatie neemt 2 getallen binnen, namelijk het aantal horizontale en verticale pixels groot dat het venster moet zijn.

| Naam | Signatuur |
|---|---|
| maak-teken-adt | (number, number → teken) |
| teken-pad! | (pad → ∅) |
| teken-toren! | (toren → ∅) |
| teken-monsters! | (monsters → ∅) |
| teken-projectielen! | (projectielen → ∅) |
| set-muis-toets! | ((symbol ∪ char, symbol, number, number → ∅) → ∅) |
| set-spel-lus! | ((number → ∅) → ∅) |
| set-toets-procedure! | ((symbol, symbol ∪ char → ∅) → ∅) |

Tabel 7: De operaties bevat in het Teken ADT

- *teken-pad!* is de operatie die het zal toelaten een pad op het venster te tekenen.
- *teken-toren!* is de operatie die het mogelijk maakt om een toren op het venster te tekenen.
- *teken-monsters!* is een operatie die op het venster monsters tekent.
- *teken-projectielen!* is een operatie die op het venster de afgeschoten projectielen tekent.
- *set-muis-toets!* is de operatie die toelaat om bepaalde muis toetsen in te stellen.
- *set-toets-procedure!* is de operatie die toelaat om bepaalde toetsenbord toetsen in te stellen.
- *set-spel-lus!* is de operatie die toelaat om een spel lus te maken.

## 2.8 Spel ADT

Het spel ADT is de kern van dit project, dit ADT zal alles beheren om tot een volwaardig project te komen. Het ADT is een abstractie bovenop het pad, level, toren en teken ADT. Het gebruikt deze 4 ADTs om respectievelijk, een pad te maken, levels te maken en up te daten, torens op de spelwereld te plaatsen, en dit dan allemaal te tekenen en mooi te visualiseren.

| Naam | Signatuur |
|---|---|
| maak-spel-adt | (∅ → ∅) |
| start! | (∅ → ∅) |

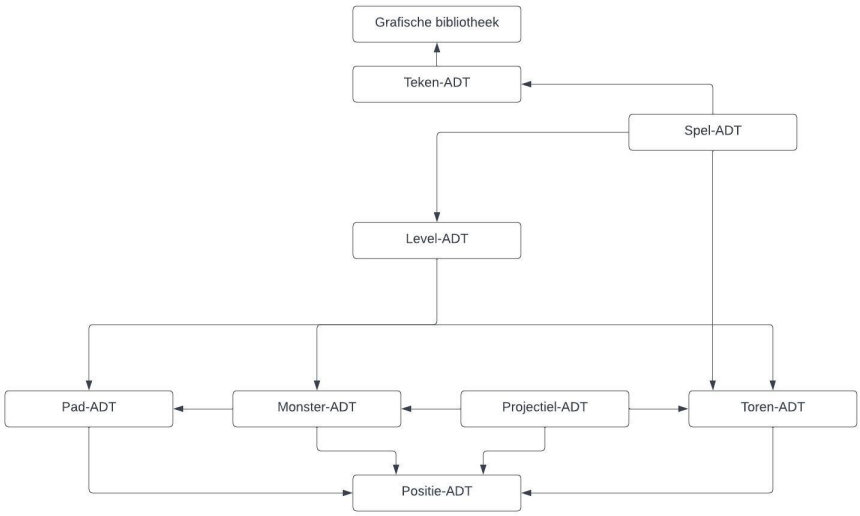
Tabel 8: De operaties bevat in het Spel ADT

De operaties van het spel ADT zijn:

- *maak-spel-adt* is de aanmaakoperatie van het spel object. Het zal de spelwereld aanmaken en de intiele zaken op het scherm tekenen (zoals het pad, achtergrond en menu).
- *start!* is de operatie die ervoor gaat zorgen dat men het spel kan starten indien men de juiste toetsen indrukt.

# 3 Afhankelijkheidsdiagram

Figuur 1 toont het afhankelijksdiagram van mijn ADTs die gebruikt geweest zijn om het project te implementeren.




Figuur 1: Afhankelijkheidsdiagram

Bij de implementatie van de functionaliteiten van mijn spel, heb ik 8 ADTs gebruikt zoals gezien op het afhankelijkheidsdiagram. Om zo'n afhankelijkheidsdiagram te lezen moet men de dozen zien als mijn ADTs en de pijlen tussen de dozen betekenen dat een ADT afhankelijk is van een ander ADT. Op de figuur kan men zien hoe mijn ADTs van elkaar afhangen:

- Het *Pad* ADT, *Monster* ADT, *Projectiel* ADT en het *Toren* ADT zijn afhankelijk van de *positie* ADT omdat ze allemaal positie objecten gebruiken om hun posities bij te houden.
- Het *Monster* ADT is ook nog afhankelijk van het *Pad* ADT omdat de monster objecten een pad nodig hebben om te weten hoe ze moeten voortbewegen, en tot waar.
- Het *Projectiel* ADT is ook nog afhankelijk van het *Toren* ADT omdat hij een toren nodig heeft vooraleer hij gemaakt en afgeschoten kan worden. En daarboven op is hij ook afhankelijk van het *Monster* ADT om te kunnen weten naar hij moet voortbewegen.
- Het *Level* ADT hangt af van verschillende ADTs. Meer bepaald het hangt af van zowel het *Pad* ADT, *Toren* ADT en *Monster* ADT. Het gebruikt het *Pad* ADT en *Monster* ADT om zo monsters te doen spawnen op het pad en dan zal het het *Toren* ADT gebruiken om projectielen te schieten naar de monsters op het pad die in de buurt zijn van de toren objecten.
- Het *Spel* ADT is afhankelijk van het *Teken* ADT maar ook van het *Level* en *Toren* ADT. Om een spel te maken moet er een pad gemaakt worden waarop monsters kunnen lopen, dit word gedaan door het level ADT. Daarbovenop heeft hij ook het *Level* ADT nodig, om het spel constant up te daten en zo dynamische elementen tot het spel te voegen. Daarnaast heeft dit ADT ook het *Toren* ADT nodig om torens te kunnen plaatsen in de spelwereld. Als laatste heeft hij ook het *Teken* ADT nodig om alles op het scherm te tekenen om ze een volwaardig spel te bekomen.

- Het *Teken* ADT is afhankelijk van de grafische bibliotheek. Die zal deze bibliotheek gebruiken om alle spelelementen op het scherm te kunnen tekenen.

# 4 Tijdsschema

In dit deel bespreken we in welke weken er aan het project gewerkt werd en wat er elke week gedaan werd om zo tot een basis tower-defense spel te geraken. Meer bepaald kan je in Tabel 9 zien hoe het project aangepakt werd.

| **Week** | **Actie** |
|---|---|
| Week 10 | Implementatie van het positie ADT. |
| Week 11 | Volledige implementatie van het Pad ADT en deels het Teken ADT en Spel ADT om paden te kunnen voorstellen en tekenen op de spelwereld. |
| Week 12 | Gedeeltelijke implementatie van het Toren ADT en het Teken ADT om torens op de spelwereld te kunnen voorstellen en tekenen. |
| Week 13 | Het Teken-ADT voorzien van boodschappen om de spellus, muisklik en toetsklik callbacks terug te krijgen van het venster. |
| Week 14 | Het Spel-ADT voorzien van procedures om de muisklik logica te implementeren en hierbij dus gebruikersinteractie toelaten en zo werkelijk torens te kunnen plaatsen (met bitmap) in de spelwereld. |
| Week 15 - 19 | Examenperiode. |
| Week 20 | Gedeeltelijke implementatie van het Monster ADT, Level ADT en Teken ADT om monsters op het pad te kunnen zetten en ze naar het einde van het pad te laten lopen en dit allemaal dan te tekenen. Daarbij ook het Spel ADT uitbreiden om een toets te introduceren om dit allemaal te starten |
| Week 21 | Volledige implementatie van het Projectiel ADT en het afwerken van het Toren, Monster, Level, Spel en Teken ADT om projectielen te kunnen afvuren en op zijn beurt, monsters geraakt door een projectiel, te laten verdwijnen van het pad en dit ook allemaal te tekenen op het scherm. |
| Week 22 | Het schrijven van het verslag en code nogmaals overlopen. |
| Week 23 | Indienen project en verslag |

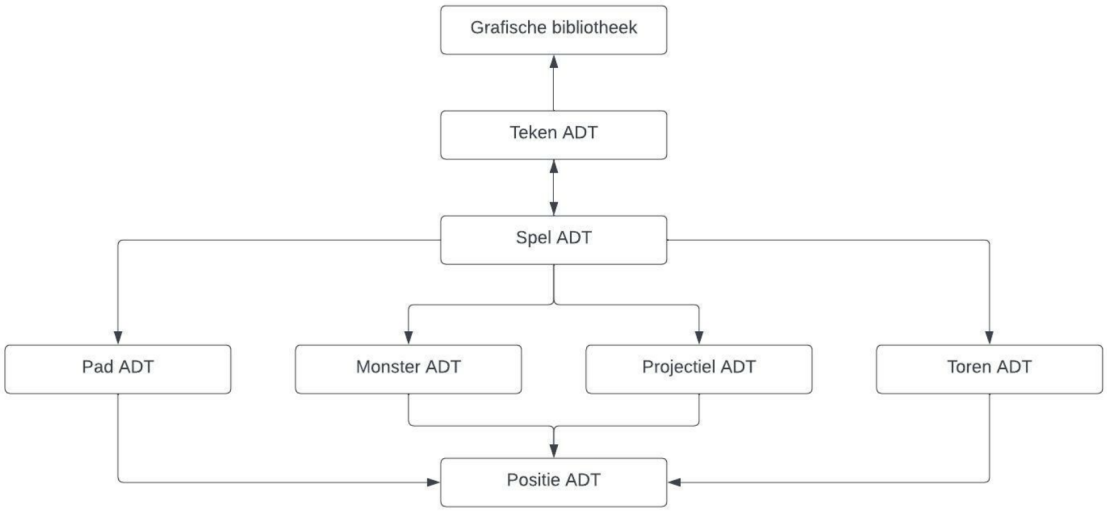
Tabel 9: Mijn tijdschema om het eerste deel van mijn tower defence game te implementeren

# 5 Voorstudie

In dit deel zal teruggekeken worden naar de voorstudie die men moest maken voor het begin van het maken van het spel. Meer bepaald zullen het afhankelijkheidsdiagram en de planning van de voorstudie vergeleken worden met de nieuwe afhankelijkheidsdiagram en het tijdsschema die werkelijk gevolgd geweest is.

## 5.1 Afhankelijkheidsdiagram

Terugkijkend naar het afhankelijksheidsdiagram van de voorstudie (Figuur 2) kan men zien dat de huidige afhankelijksheidsdiagram (AD) complexer in elkaar zit dan de vorige. Eerst en vooral, zijn er in de AD van de voorstudie minder ADTs dan nu. In ons nieuwe AD hebben we een extra ADT namelijk het Level ADT. Daarbovenop zijn er minder relaties tussen de ADTs in het vorige AD dan in de nieuwe AD. Meer specifiek, het Projectiel ADT in de voorstudie was enkel afhankelijk van het Positie ADT terwijl hij nu afhankelijk is van zowel het Monster, Toren en het Positie ADT. Daarbovenop was het monster ADT enkel afhankelijk van het Positie ADT, maar nu is hij afhankelijk van het Positie ADT maar ook het Pad ADT. En om af te sluiten was het Spel ADT in de vorige AD afhankelijk van het Pad, Monster, Projectiel, Toren ADT en Teken ADT. Maar nu is hij enkel afhankelijk van het Level ADT, Toren ADT en Teken ADT.




Figuur 2: Afhankelijkheidsdiagram opgemaakt voor voorstudie

## 5.2 Planning

Als we de planning van de voorstudie (Figuur 3) vergelijken met de echte tijdschema die gevolgd werd (Tabel 9) dan zien we dat we voor het maken van de functionaliteiten van de eerste fase, een heel andere aanpak genomen hebben. De planning van de voorstudie was heel onrealistisch en onmogelijk. Er werd ervanuit gegaan dat het mogelijk was om eerst heel het tekenlogica af te werken en dan zo verder te bouwen om de spellogica te implementeren en achteraf alles te tekenen. Terwijl wat werkelijk gedaan werd, was dat men bij de spellogica iedere keer functionaliteiten ging bijvoegen en bij ieder bijgevoegde functionaliteit het tekenlogica naargelang uitbreiden, wat een betere aanpak was.

Bij het aanmaken van de verwachte functionaliteiten heeft alles de verwachte tijd ingenomen, behalve voor het implementeren van projectielen in de spellogica. Het implementeren van de projectiel beweging naar monster functionaliteit heeft langer geduurd dan verwacht. De beweging van de projectielen naar hun bestemming was een logica die initieel zeer moeilijk mee op te komen was.

| **Week** | **Actie** |
|---|---|
| Week 10 | Implementatie van het positie ADT |
| Week 11 | Het teken ADT en spel ADT opstellen en voorzien van de basis |
| Week 12 | Volledige implementatie van het teken ADT implementeren |
| Week 13-14 | Implementatie van het pad ADT om een pad te plaatsen |
| Week 15-19 | Examenperiode |
| Week 20 | Implementatie van het monster ADT en toren ADT dat een toren en monster plaatst en het monster beweegt |
| Week 21 | Implementatie van het projectiel ADT dat een projectiel beweegt en afwerken spel ADT |
| Week 22 | Verslag schrijven |
| Week 23 | Indienen project en verslag |

Figuur 3: Planning opgemaakt voor voorstudie

# 6 Spel uitleg

Om mijn tower defence game op te starten moet men het bestand met naam *Spel.rkt* openen en deze runnen. Wanneer men dit gedaan heeft zal de spelwereld aangemaakt worden en kan men beginnen spelen. Het scherm bestaat uit 2 delen, een spelwereld deel en een menu deel. Om torens te plaatsen in de spelwereld moet men op de gewenste toren (voorlopig is er nog maar 1 soort) klikken in de menu en dan op de gewenste plaats in de spelwereld klikken om de toren te plaatsen. Als men dan de ronde wil starten moet met op de spatie toets klikken. Indien de ronde gedaan is, kan men de ronde herstarten door nogmaals op spatie te drukken. Tenslotte, indien met de ronde wil beeindigen moet geklikt worden op de escape toets (dit is een cheat toets).