Programmeerproject 1:

Verslag “Space Invaders”

Maxim Brabants

0576581

Maxim.Lino.Brabants@vub.be

Academiejaar 2020-2021



Inhoudsopgave

[1 Functionaliteit 3](#_Toc64038327)

[1.1 Beweging van raket 3](#_Toc64038328)

[1.2 Matrix 4](#_Toc64038329)

[1.3 Implementatie alienvloot 4](#_Toc64038330)

[2 Beschrijving ADTs 4](#_Toc64038331)

[2.1.1 Positie 4](#_Toc64038332)

[2.1.2 Raket 6](#_Toc64038333)

[2.1.3 Kogel 6](#_Toc64038334)

[2.1.4 Alienschip 7](#_Toc64038335)

[2.1.5 Alienvloot 8](#_Toc64038336)

[2.1.6 Teken 9](#_Toc64038337)

[2.1.7 Game 9](#_Toc64038338)

[3 Afhankelijkheidsdiagram 9](#_Toc64038339)

[4 Werktijd 11](#_Toc64038340)

# Functionaliteit

We starten met het ADT te programmeren dat de posities representeert van objecten in het spel. Op die manier heeft elk spelelement een positie die bijgehouden wordt zonder dat we dat telkens apart moeten implementeren in elk ADT. De plaatsbepaling gebeurt op basis van een x- en y-coördinaat maar we moeten er later mee rekening houden dat deze moeten omgezet worden naar een pixel gebaseerde positie op het scherm.

Afgezien van de posities moeten we twee belangrijke elementen implementeren :

* We moeten er zien voor te zorgen dat er een raketje op het scherm te zien is en meer specifiek onderaan het scherm. Als de raket zichtbaar is op het scherm dan is de volgende stap om hem te laten bewegen, maar slechts van links naar rechts en omgekeerd. Ook moet deze een kogel kunnen afvuren richting de alienschepen. Om het dan volledig af te werken moeten we de beweging van de raket beperken tot de randen van het spelscherm. Onze raket mag natuurlijk niet buiten de spelzone bewegen.
* Een tweede cruciaal element is het Alienvloot. Deze moeten we aan het begin van het spel laten tekenen bovenaan het scherm, tegenover de raket en deze moet voorgesteld worden in de vorm van een rechthoek. Naast de raket moet het vloot ook kunnen bewegen van links naar rechts en omgekeerd en ook naar beneden in het geval dat deze één van de spelranden raakt.

## Beweging van raket

Ik opteer ervoor om te starten met de implementatie van de raket. Om goed te zijn moeten we onze raket aan het begin van elk spel onderaan het scherm laten tevoorschijn komen omdat deze natuurlijk naar boven, richting de alienschepen moet schieten. Vooraleer we de operaties beginnen programmeren moeten we een plek voorzien waar we deze kunnen definiëren. Hiervoor voorzien we een raket-ADT. Daarin kunnen we alle handelingen programmeren die de raket moet kunnen verrichten.

De constructor van de raket verwacht een positie zodat het spel op elk moment weet waar de raket zich bevindt. Ik voorzie een beweeg -en schietoperatie en ten slotte ook de dispatch-functie. De beweegfunctie van de raket heb ik als volgt geïmplementeerd: Herinner dat onze raket een positie heeft. Als we als argument een richting nemen en we geven deze dan vervolgens mee met de beweegoperatie van het positie-object, dan wordt de beweging in het positie-ADT afgehandeld en moeten we ons voor de rest geen zorgen maken. We moeten dan wel zien dat de implementatie van beweeg in het positie-ADT correct is.

Afbeelding met tekst

Automatisch gegenereerde beschrijving

De schietoperatie gaan we later bekijken. Eerst gaan we trachten om ook een groep van alienschepen op het scherm te krijgen in de vorm van een vloot. Ik veronderstel dat dit wat meer tijd in beslag zal nemen en dat we even over de aanpak zullen moeten nadenken aangezien we alle alienschepen zowel apart als één geheel moeten kunnen manipuleren. (alienschepen moeten individueel een kogel kunnen afvuren richting de raket maar deze moeten samen als een geheel bewegen over het scherm.)

## Matrix

Als we kijken naar hoe een alienvloot is opgebouwd, dan zien we dat deze bestaat uit een aantal rijen en kolommen met op elke positie een alienschip. We merken op dat dit te vergelijken is met een matrixvorm. Het proces om een alienvloot te implementeren zou dan ook een stuk gemakkelijker worden als we over een datastructuur zouden beschikken die de vorm aanneemt van zo’n matrix. We weten dat dit niet standaard aanwezig is in Scheme maar dat is helemaal geen probleem want we kunnen dit proberen zelf te programmeren.

Ik heb ervoor gekozen om eerst een matrix-ADT’tje / datastructuur te gaan programmeren zodat het achteraf eenvoudiger wordt om de alienschepen te gaan manipuleren. Ik doe dit op volgende wijze : Ik maak eerst een centrale vector aan waarbij elke locatie een rij-index voorstelt en daarna vul ik op zijn beurt elke locatie van de centrale vector met vectoren van dezelfde grootte. Op die manier verkrijg je een datastructuur die een matrixvorm aanneemt.

We kunnen in principe de vector vullen door zoveel keer vector-set! te doen als we kolommen nodig hebben, maar dan zou het achteraf moeilijker worden om het aantal rijen en kolommen van het alienvloot te wijzigen. Een mogelijke oplossing hiervoor zou zijn om dit in een functie onder te brengen die een index bijhoudt om zo maar één keer vector-set! te moeten neerschrijven in code. We hebben dus twee variabelen nodig die het aantal rijen en aliens per rij bijhouden zodat we door aanpassing van deze twee ons geheel vloot kunnen configureren naar gewenste grootte. Het bijhouden van deze variabelen doen we in een ‘constanten.rkt’ bestand.

Afbeelding met tekst

Automatisch gegenereerde beschrijving

## Implementatie alienvloot

Met behulp van onze matrix-datastructuur kunnen we nu eenvoudig beginnen met de implementatie van ons alienvloot. Natuurlijk hebben we ook iets nodig dat een individueel alienschip voorstelt, een object van het type Alienschip dat we dan in meerdere malen in onze matrix kunnen steken.

Als we eens nadenken over de operaties die het Alienschip-type moet bevatten dan komen deze eigenlijk zo goed als overeen met diegene die we in het raket-ADT hebben geïmplementeerd. Een alienschip moet namelijk ook tweedimensionaal kunnen bewegen (ook naar beneden maar veel verandert dat er niet aan) en kunnen schieten, wat dus ook handelingen zijn die de raket implementeert.

# Beschrijving ADTs

### Positie

Een belangrijk onderdeel waarmee rekening moet gehouden worden in bijna elk computerspel is de plaatsbepaling van bepaalde objecten in de spelwereld. Die moet namelijk bijgehouden worden doorheen het programma om te weten waar in de spelwereld deze objecten zich bevinden. (een voorbeeld vanuit het spelletje zelf : Als je bijvoorbeeld wilt weten of een afgevuurde kogel een alienschip heeft geraakt dan moet je zowel de positie van de kogel als die van het doelwit bijhouden zodat je weet wanneer het schip geraakt is). Het is daarom een geschikte methode om dit onder te brengen in een abstract datatype want als je bijvoorbeeld in je code werkt met x- en y-coördinaten en je moet dit op elke plaats in je programma telkens opnieuw gaan implementeren waar je dit nodig hebt dan kan je code erg lang en onoverzichtelijk worden. Ook zorgt dit voor codeduplicatie wat in alle gevallen moet vermeden worden. Het Positie ADT maakt een abstractie van een positie in een tweedimensionaal veld.

De volgende operaties horen bij dit type :

|  |  |
| --- | --- |
| Naam | Signatuur |
| maak-positie | (number number → Positie) |
| x | → number) |
| y | → number) |
| x! | (number → |
| y! | (number → |
| gelijk? | (Positie → boolean) |
| RAnd? | 🡪 boolean) |

* De **maak-positie** operatie zal een specifiek Positie objectje aanmaken en dit stelt dan een specifieke positie voor in de spelwereld. Deze procedure verwacht twee numbers : een x- en y-coördinaat aangezien het spel gespeeld wordt in een tweedimensionale wereld. Dit zal evalueren in een Positie object waarop je dan de rest van de operaties kan toepassen.
* Als je de waardes van de x- en y-coördinaat wilt opvragen om daarmee bijvoorbeeld berekeningen te gaan doen dan kan dit door de accessormethodes **x** en **y** op te roepen die respectievelijk de x- en y-coördinaat zullen teruggeven van een bepaald object.
* Ook is het mogelijk om deze data te wijzigen. Dit kan door de mutatormethodes **x!** en **y!** aan te roepen. Je geeft een nieuwe waarde mee in de vorm van een number aan de methode van de coördinaat die je wilt wijzigen en de desbetreffende waarde zal aangepast worden.
* De **gelijk!** operator kan gebruikt worden om te kijken of twee posities identiek zijn. Dit predicaat geeft een boolean terug indien ze gelijk zijn.

### Raket

Tijdens het spel is het natuurlijk de bedoeling dat je met de raket onderaan het scherm alle alienschepen van het vloot bovenaan probeert te vernietigen. De raket is een belangrijk element in het spel en kan dus best ook in een ADT verpakt worden met bijhorende operaties.

De volgende operaties horen bij dit type :

|  |  |
| --- | --- |
| Naam | Signatuur |
| maak-raket | (→ Raket) |
| beweeg! | (symbol → |
| Schiet! | (→ ) |

* Bij de opstart van een nieuw spel of wanneer de raket nog levens over heeft dan kan de **maak-raket** methode opgeroepen worden. Het Teken ADT zal er dan voor zorgen dat er telkens in die gevallen een nieuwe raket wordt getekend op een vaste positie in de wereld.
* Voor de **beweeg!** operatie kan er een argument worden meegegeven in de vorm van een number die dan een soort van richtingsargument voorstelt. De procedure kan dan op basis van dat argument bepalen of de raket naar links of naar rechts moet verschoven worden.
* De **schiet!** operatie laat een kogel afvuren vanaf de positie van de raket maar die is gekend dus ook hier dient niets meegegeven te worden.

### Kogel

Een kogel die door de spelwereld wordt geschoten kan gezien worden als een bewegend objectje in de ruimte. Het is een spelelement en kan dus rechtstreeks herleidt worden tot een ADT. Een kogel kan afgeschoten worden ofwel door een van de alienschepen ofwel door de raket.

De volgende operaties horen bij dit type :

|  |  |
| --- | --- |
| Naam | Signatuur |
| maak-kogel | (Positie → Kogel) |
| positie | (→ Positie) |
| positie! | (number number → |
| geraakt? | (Positie → boolean) |
| verdwijn | ( → ) |

* De **maak-kogel** operatie verwacht als argument de positie vanwaar hij wordt afgevuurd. Dit kan zoals gezegd de positie van een van de alienschepen zijn maar ook de positie van de raket.
* Je moet de **positie** van de kogel kunnen opvragen om te checken of de kogel iets heeft geraakt of niet.
* De positie moet ook kunnen aangepast worden met **positie!** want eens dat hij is afgevuurd beweegt hij zich in een rechte lijn naar boven dus moet deze positie constant worden aangepast doorheen zijn beweging.
* **geraakt?** bepaalt op basis van een positie of deze een ander object in de ruimte heeft geraakt.
* Na het botsen van een kogel tegen een ander object in de spelwereld moet de kogel natuurlijk verdwijnen zodat deze niet verder blijft bewegen.

### Alienschip

Elk alienschip apart moet de mogelijkheid kunnen hebben om zelfstandig een kogel af te vuren en uiteraard om te ontploffen wanneer deze geraakt wordt door een kogel van de raket. Een abstractie hiervan is dus zeker niet overbodig.

De volgende operaties horen bij dit type :

|  |  |
| --- | --- |
| Naam | Signatuur |
| maak-alienschip | (Positie → Alienschip) |
| vuur-kogel-af | (→ ) |
| ontplof | ( → ) |

* Elk gegenereerd alienschip heeft naast een eigen positie in de vector ook een unieke positie in de spelwereld op basis van een x- en y-coördinaat. Voor **maak-alienschip** dient er dan een positie meegegeven te worden wat dan evalueert naar een alienschip op die specifieke positie in de spelwereld.
* **vuur-kogel-af** verwacht geen argumenten aangezien het Alienschip ADT toegang heeft tot zijn positie en dus ook vanop die positie een kogel kan laten afvuren.
* Dit geldt ook voor **ontplof** omdat deze ook gewoon de positie van het alienschip neemt om te bepalen waar de ontploffing moet gebeuren.

### Alienvloot

Het geheel van alienschepen dat bovenaan het scherm beweegt abstraheren we best in een ADT zodat we het gehele vloot als één geheel kunnen manipuleren. Dit gaat het misschien makkelijker maken in het geval dat we al onze alienschepen moeten laten bewegen en dit is ook het geval in ons spel.

De volgende operaties horen bij dit type :

|  |  |
| --- | --- |
| Naam | Signatuur |
| maak-vloot | (vector <Alienschip> → Alienvloot) |
| beweeg! | (number → |
| onderkant? | (Positie → boolean) |
| Voor-alle-schepen | (<procedure> 🡪 |

* Om een Alienvloot te kunnen maken zal er een bepaalde datastructuur nodig zijn die een vector van aparte Alienschip-objecten voorstelt. Een Alienvloot kun je dan best zien als een geheel van Alienschepen die over het scherm bewegen. Dit wordt teruggegeven door **maak-vloot**. Ik opteer ervoor om het alienvloot te laten voorstellen als een matrix door een vector aan te maken van meerdere vectoren. Dat zal het vinden van een specifiek alienschip misschien makkelijker maken aangezien het zoeken in O(1) gebeurt.
* Als je een bepaald Alienschip zou willen verwijderen van het vloot (bijvoorbeeld omdat het geraakt is door de raket) dan kan je dat doen door aan de **verwijder-schip** operatie een bepaald Alienschip-object mee te geven en die wordt dan verwijderd uit de lijst van schepen.
* Een alienvloot kan afwisselend van links naar rechts of omgekeerd over het scherm bewegen tot aan de rand. Door een bepaald getal mee te geven aan **beweeg!**, kan deze procedure bepalen in welke richting het vloot moet bewegen.
* Als het vloot de rand links of rechts van het scherm raakt dan dient deze om te keren en de andere kant op te gaan. Het is dus handig als we een methode hebben die aangeeft of het vloot effectief de rand heeft bereikt. Anders zou deze gewoon van het scherm verdwijnen. Het predicaat **rand?** zal dit kunnen aangeven op basis van de positie van het vloot.
* Hetzelfde geldt voor als het vloot de onderkant van het scherm bereikt want dan is het spel namelijk gedaan en moet het stopgezet worden. Ook **onderkant?** is een handig predicaat die op basis van een positie aangeeft of het vloot de onderkant heeft bereikt.

### Teken

Dit speciale ADT zal zijn bronnen moeten halen van een voorgedefinieerde grafische bibliotheek want anders zou het voor geen enkel type mogelijk zijn om iets te kunnen laten tekenen door het Teken ADT.

De volgende operaties horen bij dit type :

|  |  |
| --- | --- |
| Naam | Signatuur |
| teken-Raket | (Positie → |
| teken-Alienschip | (Positie → |
| teken-kogel | (Positie → |

### Game

Dit stelt het centrale ADT voor dat het spel zal laten opstarten en alle benodigde ADTs zal oproepen in een logische volgorde.

|  |  |
| --- | --- |
| Naam | Signatuur |
| maak-spel | ( → Spel) |
| start-spel! | ( → |
| stop-spel! | ( → |

* **maak-spel** zal ervoor zorgen dat er zeg maar een zogezegde spelsessie wordt aangemaakt die actief wordt gehouden door een oneindige lus. Dit ADT zal grotendeels de spellogica bijhouden.
* Bij opstart moet er standaard een menu of iets dergelijks verschijnen dat de gebruiker in staat stelt om een spel te kunnen starten. Het Game ADT zal dus moeten wachten op interactie van de gebruiker. **toon-menu** moet geen argumenten krijgen want het moet onmiddellijk verschijnen bij opstart tenzij je natuurlijk vanuit het spel terug wilt navigeren.
* **start-spel!** moet alle elementen voor tijdens het spel aanroepen zodat de speler kan beginnen met spelen.
* **stop-spel!** moet het spel onderbreken en ervoor zorgen dat alle processen worden afgesloten.

# Afhankelijkheidsdiagram

Er zijn bepaalde types die niet correct gaan kunnen werken zonder het bestaan van andere. Er is dus een bepaalde afhankelijkheid die dient gerespecteerd te worden om een goede werking van het gehele programma te kunnen realiseren.

Voor mijn ontwerp ziet dat er als volgt uit :

Diagram

Description automatically generated

Op het schema is er te zien dat het Positie ADT toch wel een belangrijk en handig type is bij het ontwerp van het spel. Dit komt natuurlijk omdat al de objecten in de game (Kogel, Raket, Alienvloot, Alienschip en Rots) allemaal een voor een afhankelijk zijn van hun positie. Het Positie ADT is nodig in elk ADT van een spelelement.

Alsook staat Alienvloot in verbinding met Alienschip omdat Alienvloot een structuur is van meerdere Alienschepen. Als er geen Alienschepen zijn kan er ook geen Alienvloot bestaan. Door het Teken ADT onderscheiden we de spellogica en de tekenlogica van elkaar. Enerzijds heeft het Game ADT het Teken ADT nodig om specifieke dingen te tekenen op het scherm en anderzijds heeft het Teken ADT ook het Game ADT nodig om te weten wat er juist getekend moet worden (dubbele pijl). Daarvoor worden namelijk de specifieke teken-procedures aangeroepen.

Het vierkante vak staat niet voor een zelf gedefinieerd type maar het is een bestand dat het Teken ADT in staat stelt om grafische elementen van te halen en vervolgens te tekenen in ons spel. Het teken ADT zal weliswaar een tekenmethode bevatten voor elk spelelement.

# Werktijd

Dit zijn de reeds uitgevoerde taken per week.

|  |  |
| --- | --- |
| Week | Taak |
| Week 10 -11 | Positie ADT geïmplementeerd en begonnen met implementatie van Raket ADT. |
| Week 12 | Raket laten bewegen onderaan het scherm. |
| Week 13 | Implementatie van Matrix ADT. |
| Week 14 | Probleem met window-sizing dat veroorzaakt wordt door de grafische bibliotheek en besturingssysteem. (mail gestuurd naar Bjarno) |
| Week 15 | Alienvloot laten bewegen over het scherm |
| Week 15 - 16 | Proberen beweging van objecten te beperken tot de randen van het speelveld. |
| Week 16 - 17 | Kogel-ADT implementeren + raket kogels laten afvuren. |