HOOFDSTUK

1 Kennismaken met Greenfoot

onderwerpen:	de interface van Greenfoot, omgaan met objecten, methodes aanroepen, een scenario uitvoeren
concepten:	object, klasse, methode-aanroep, parameter, retourwaarde

In dit boek leer je hoe je met de programmeeromgeving Greenfoot computerspellen en simulaties kunt ontwikkelen. In dit hoofdstuk zullen we naar Greenfoot zelf kijken om te zien wat je ermee kunt doen en hoe je ermee kunt werken. We doen dat door aan de slag te gaan met een aantal reeds bestaande programma's.

Zodra we een beetje vertrouwd zijn met Greenfoot, gaan we zelf een spel schrijven.

De beste manier om dit hoofdstuk (en de rest van dit boek) te lezen, is vanachter je computer, terwijl Greenfoot op het scherm geopend is en het boek op je bureau opengeslagen ligt. We vragen je tijdens het lezen regelmatig om opdrachten in Greenfoot uit te voeren. Sommige taken kun je overslaan, maar je zult het meeste leren door de tekst te lezen en alle opdrachten uit te voeren.

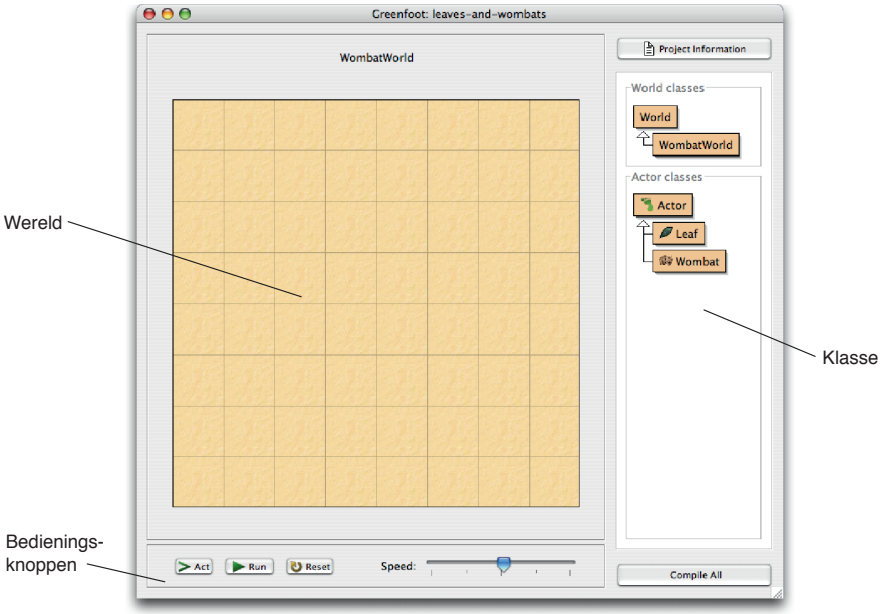
Op dit moment gaan we ervan uit dat je de Greenfoot-software en de boekscenario's geïnstalleerd hebt (op de manier zoals beschreven is in bijlage A). Heb je dit nog niet gedaan, lees dan eerst die bijlage.

1.1 Aan de slag

Start Greenfoot en open het scenario *leaves-and-wombats* dat je kunt vinden in de map *book-scenarios/chapter01*.

Opmerking
Als je Greenfoot voor de eerste keer start, zie je een dialoogvenster waarin je gevraagd wordt wat je wilt doen. Klik op *Choose a scenario*. Selecteer anders de optie **Scenario–Open** in het menu.[1]

[1] We gebruiken deze notatie als we willen dat je functies in het menu selecteert. **Scenario–Open** betekent: selecteer de optie Open in het menu Scenario.



Figuur 1.1:
Het hoofdvenster van Greenfoot

Zorg ervoor dat je het scenario *leaves-and-wombats* opent uit de map *book-scenarios/chapter01* en niet het scenario wombats dat standaard geïnstalleerd wordt bij de installatie van Greenfoot.

Als het goed is, zie je nu het hoofdvenster van Greenfoot met daarin het geopende scenario, dat er ongeveer uitziet zoals in figuur 1.1.

Het hoofdvenster is onderverdeeld in drie gebieden en bevat ook een aantal extra knoppen. Die gebieden zijn:

- De *wereld*. Het grootste gebied (in dit geval een zandkleurig lijnenrooster) wordt de wereld genoemd. In dit gebied wordt het programma uitgevoerd en zien we dingen gebeuren.
- Het *klassendiagram*. Het gebied aan de rechterkant met de beige kaders en pijlen is het klassendiagram. We komen hier zo meteen op terug.
- De *bedieningsknoppen*. De knoppen *Act, Run* en *Reset* en de instelbalk voor de snelheid onder in het scherm noemen we de bedieningsknoppen. Ook daar komen we binnenkort op terug.

1.2 Objecten en klassen

We zullen eerst stilstaan bij het klassendiagram. In het klassendiagram worden alle klassen weergegeven die in dit scenario gebruikt worden. In dit geval zijn dat `World`, `WombatWorld`, `Actor`, `Leaf` en `Wombat`.

Voor onze projecten zullen we de programmeertaal Java gebruiken. Java is een objectgeoriënteerde taal. De concepten klassen en objecten zijn elementair bij objectgeoriënteerd programmeren.

Laten we beginnen met de klasse `Wombat`. De klasse `Wombat` vertegenwoordigt het algemene concept van een wombat: de klasse beschrijft alle wombats. Zodra we in Greenfoot beschikken over een klasse, kunnen we daar objecten mee maken. Objecten worden bij het programmeren ook vaak instanties genoemd. De twee termen hebben precies dezelfde betekenis.

Figuur 1.2: Een wombat in het Narawntapu National Park in Tasmanië[2]

Een wombat (*Vombatus ursinus*) is trouwens een Australisch buideldier (figuur 1.2). Als je hier meer over wilt weten, kun je op internet heel veel informatie vinden.

Het kan zijn dat de klassen gearceerd worden weergegeven. De arcering betekent dat de klasse eerst gecompileerd moet worden. Compileren is een vorm van vertalen: de broncode van de klasse wordt vertaald naar een machinecode die jouw computer kan uitvoeren.

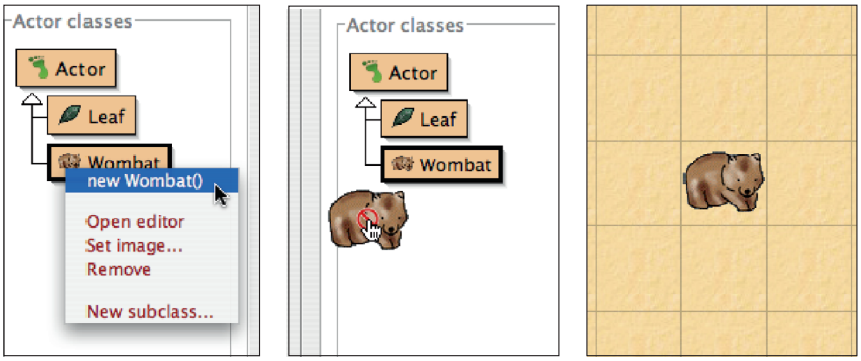
We kunnen de klassen compileren door te klikken op de knop *Compile All,* onderaan in het hoofdvenster van Greenfoot. Zodra de klassen gecompileerd zijn, verdwijnt de arcering en kun je objecten maken.

Klik eventueel op de knop *Compile All,* en rechtsklik op de klasse `Wombat`. Het contextmenu van de klasse wordt weergegeven (figuur 1.3a).[3] Met de eerste optie in dat menu, `new Wombat()` kunnen we nieuwe wombatobjecten maken. Probeer het zelf.

[2] Bron: Wikipedia, volgens de GNU Free Documentation License

[3] Onder Mac OS kun je rechtsklikken of, als je een muis hebt met maar één muisknop, Ctrl-klikken.

Je zult zien dat daardoor een kleine afbeelding van een wombatobject op het scherm verschijnt, dat je met je muis over het scherm kunt verplaatsen (figuur 1.3b). Plaats de wombat in de wereld door ergens in de wereld te klikken (figuur 1.3c).



Figuur 1.3:
a) Het contextmenu van de klasse.
b) Een nieuw object slepen.
c) Het object plaatsen.

Zodra je in Greenfoot de beschikking hebt over een klasse, kun je daar zo veel objecten van maken als je maar wilt.

Concept:
Met een klasse kun je veel objecten maken.

Oefening 1.1 Plaats nog wat meer wombats in de wereld. Maak ook wat bladeren.

Op dit moment zullen we alleen kijken naar de klassen `Wombat` en `Leaf`. We komen later op de andere klassen terug.

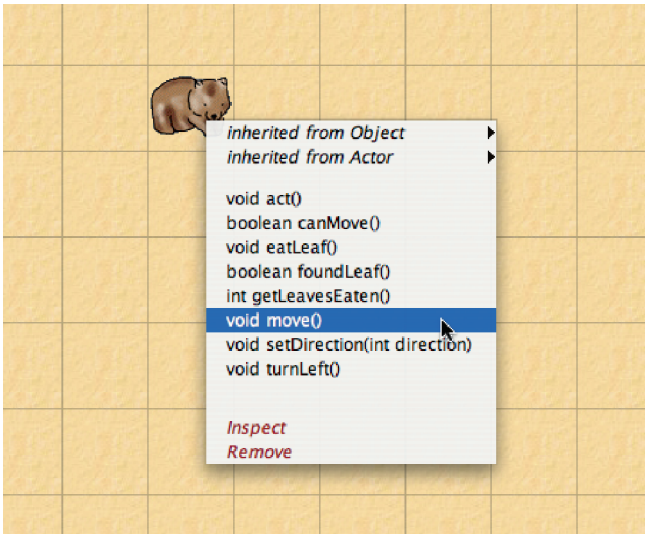
1.3 Interactie met objecten

Zodra we een aantal objecten in de wereld geplaatst hebben, kunnen we daar iets mee doen door erop te rechtsklikken. Daardoor opent het objectmenu (figuur 1.4). In het objectmenu worden alle handelingen weergegeven die het betreffende object kan uitvoeren. In het objectmenu van een wombat kunnen we zien wat een wombat kan doen, plus twee extra functies, *Inspect* en *Remove*, die we later zullen bespreken.

In Java worden deze handelingen methodes genoemd. Het kan geen kwaad om meteen te beginnen om standaardterminologie te gebruiken, dus vanaf nu zullen we ze dan ook methodes noemen. We kunnen een methode aanroepen door de betreffende methode in het menu te selecteren.

Concept:
Objecten hebben methodes. Door een methode aan te roepen, gebeurt er iets.

Oefening 1.2 Roep de methode `move()` van een van de wombats aan. Wat doet deze methode? Probeer het een aantal keer achter elkaar. Roep de methode `turnLeft()` aan. Plaats twee wombats in je wereld en laat ze naar elkaar kijken.



Figuur 1.4: Het objectmenu van een wombat

Kortom: we kunnen dingen laten gebeuren door objecten van een van de beschikbare klassen te maken en opdrachten aan de objecten te geven door hun methodes aan te roepen.

Laten we eens beter kijken naar het objectmenu. De methodes `move` en `turnLeft` worden weergegeven als:

```
void move()
void turnLeft()
```

Zoals je ziet, wordt niet alleen de naam 'methodes' weergegeven. Voor de naam staat het woord `void` en achter het woord zijn twee haakjes weergegeven. Deze twee cryptische stukjes informatie vertellen ons wat voor gegevens gebruikt moeten worden bij het aanroepen van de methode en welke gegevens de methode retourneert.

1.4 Returntypes

Het woord voor de methodenaam duidt het returntype aan. Het vertelt ons wat de methode retourneert wanneer we die aanroepen. Het woord `void` betekent in deze context 'niets': methodes met een returntype `void` retourneren geen enkele informatie. Ze voeren gewoon hun handeling uit en stoppen dan.

Concept:
Het returntype van een methode geeft aan wat een methode retourneert als je deze aanroept.

Elk ander woord dan `void` betekent dat de methode, wanneer die wordt aangeroepen, een vorm van informatie retourneert. Het woord zelf geeft aan wat voor informatie de methode retourneert. In het objectmenu van de wombat (figuur 1.4) zien we ook de woorden `int` en `boolean`. Het woord `int` is de afkorting voor 'integer' en verwijst naar gehele getallen (getallen zonder decimaalteken). Voorbeelden van integers zijn 3, 42, -3 en 12000000.

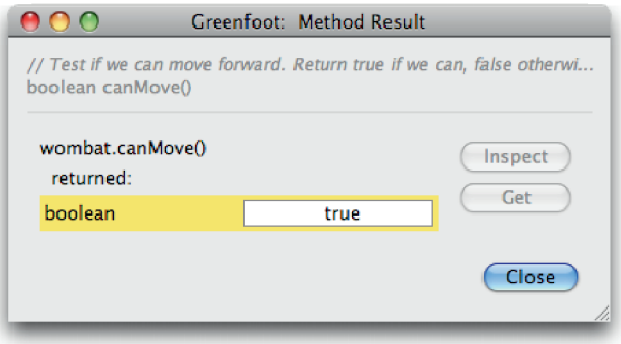
Het type `boolean` heeft slechts twee mogelijke waarden: `true` en `false`. Een methode die een boolean retourneert, retourneert ofwel de waarde `true`, ofwel de waarde `false`.

Concept:
Een methode met een returntype `void` retourneert geen waarde.

Methodes met het returntype `void` kunnen we beschouwen als opdrachten voor onze wombat. Als we de methode `turnLeft` aanroepen, gehoorzaamt de wombat en slaat linksaf. Methodes met een returntype anders dan `void` kunnen we beschouwen als vragen. Bekijk de methode `canMove`:

```
boolean canMove()
```

Wanneer we deze methode aanroepen, zien we het resultaat weergegeven in een dialoogvenster dat lijkt op figuur 1.5. Belangrijke informatie hier is het woord `true` dat deze methode retourneert. Eigenlijk hebben we de wombat zojuist gevraagd of hij kan verplaatsen en het antwoord van de wombat is 'ja' (`true`).



Figuur 1.5:
Het resultaat van een aanroep van een methode

Oefening 1.3 Roep de methode `canMove()` van je wombat aan. Is het antwoord altijd *true*? Zijn er situaties waarin de methode *false* retourneert?

Probeer ook een andere methode met een retourwaarde:

```
int getLeavesEaten()
```

Met deze methode kunnen we informatie opvragen over hoeveel bladeren de wombat heeft gegeten.

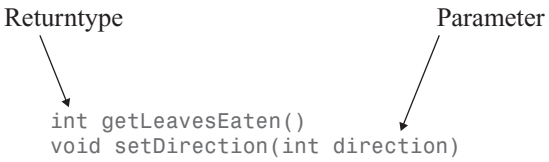
Concept:
Methodes met het returntype `void` zijn te beschouwen als **opdrachten**; methodes met een ander returntype dan `void` als **vragen**.

Oefening 1.4 Wanneer er een nieuwe wombat gemaakt is, zal de methode `getLeavesEaten()` altijd de waarde nul retourneren. Kun je een situatie creëren waarin deze methode een waarde ongelijk aan nul zal retourneren? Met andere woorden: kun je je wombat wat bladeren laten eten?

Methodes met een niet-returntype `void` vertellen meestal iets over het object (Kan het verplaatsen? Hoeveel bladeren heeft het gegeten?) maar veranderen niets aan het object. De wombat is precies hetzelfde gebleven als voordat we iets vroegen over de bladeren. Methodes met een returntype `void` zijn meestal opdrachten voor de objecten waardoor ze iets doen.

1.5 Parameters

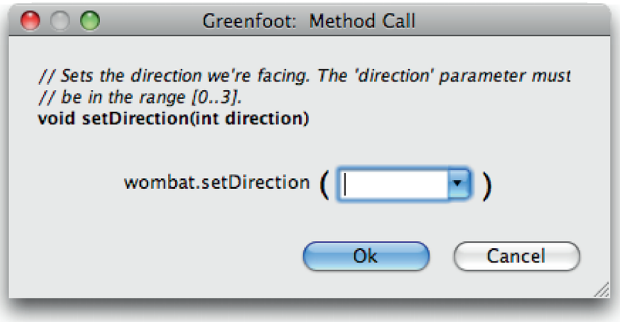
Het andere stuk in het contextmenu van de methode waar we het nog niet over gehad hebben zijn de haakjes na de naam van de methode.



Concept:
Een **parameter** is een mechanisme om een methode te voorzien van aanvullende gegevens.

De haakjes na de naam van de methode bevatten de parameterlijst. Daaraan kunnen we zien of de methode nog aanvullende informatie nodig heeft om uitgevoerd te worden en zo ja, welke informatie. Wanneer we alleen maar een aantal haakjes zonder iets daartussen zien, zoals bij alle methodes die we tot nu toe zijn tegengekomen, heeft de methode een lege parameterlijst. Met andere woorden: de methode verwacht geen parameters en zal meteen uitgevoerd worden, zodra we die aanroepen. Als er wel wat tussen de haakjes staat, verwacht de methode één of meer parameters: aanvullende informatie die nodig is om de methode uit te voeren.

Laten we de methode `setDirection` eens bekijken. We zien dat de parameterlijst de woorden `int direction` bevat. Wanneer we de methode aanroepen, opent een dialoogvenster, zoals dat in figuur 1.6.



Figuur 1.6:
Het dialoogvenster wanneer een methode aangeroepen wordt

De woorden `int direction` betekenen dat deze methode een parameter van het type `int` verwacht die een richting aangeeft. Een parameter is een extra stukje informatie dat essentieel is om deze methode uit te kunnen voeren. Elke parameter wordt gedefinieerd met twee woorden: het eerste is het type van de parameter

Concept:
Parameters en retourwaarden hebben **types**. Voorbeelden van types zijn **int** voor getallen en **boolean** voor de logische (true/false) waarden.

(hier: `int`). Deze wordt gevolgd door een naam, die ons een idee geeft over het gebruik van deze parameter. Als een methode een parameter heeft, moeten we voor die aanvullende informatie zorgen, wanneer deze methode aangeroepen wordt.

In dit geval is het type `int`, wat betekent dat de aanvullende informatie een geheel getal moet zijn. Aan de naam is te zien dat dit getal dient om op de een of andere manier een richting aan te geven.

Boven in het dialoogvenster vinden we een commentaar, waarin te lezen is dat de parameter `direction` een waarde kan hebben in het bereik van 0 tot en met 3.

Oefening 1.5 Roep de methode `setDirection(int direction)` aan. Voer een waarde voor de parameter in en kijk wat er gebeurt. Welk getal komt overeen met welke richting? Schrijf dit op. Wat gebeurt er wanneer je een getal groter dan 3 invoert? Wat gebeurt er als je een niet-geheel getal invoert, zoals een decimaal getal (2,5) of een woord (drie)?

Concept:
De specificatie van de methode met de naam en de parameters heet de **signatuur**.

De methode `setDirection` verwacht maar één parameter. Later zullen we gevallen zien waarin methodes meer dan een parameter verwachten. In dat geval bevat de parameterlijst van de methode (tussen haakjes) alle parameters die de methode verwacht.

De omschrijving van elke methode in het objectmenu, de naam van de methode en de parameterlijst, wordt de signatuur van de methode genoemd.

We zijn nu op een punt aanbeland waar je de basisbeginselen van de belangrijkste interacties met Greenfoot-objecten kent. Je kunt objecten maken van klassen, de argumenten van methodes interpreteren en methodes (met en zonder parameters) aanroepen.

1.6 Greenfoot uitvoeren

Interactie met Greenfoot-objecten is ook op een andere manier mogelijk: met de bedieningsknoppen.

Tip:
Je kunt objecten sneller in de wereld plaatsen door een klasse in het klassendiagram te selecteren en vervolgens Shift ingedrukt te houden en te klikken in de wereld.

Oefening 1.6 Plaats een wombat en een redelijk aantal bladeren in de wereld en roep dan de methode `act()` van de wombat een aantal malen aan. Wat doet deze methode? Hoe verschilt deze methode van de methode `move`? Probeer verschillende situaties uit, bijvoorbeeld als de wombat zich tegen de rand van de wereld bevindt of op een blad zit.

Oefening 1.7 Klik terwijl je een wombat geselecteerd hebt op de knop *Act* onder in het Greenfoot-venster. Wat gebeurt er?

Oefening 1.8 Wat is het verschil tussen klikken op de knop *Act* en het aanroepen van de methode `act()`? Probeer dit met verschillende wombats in de wereld.

Oefening 1.9 Klik op de knop *Run*. Wat doet deze methode?

De methode `act` is een erg elementaire methode van Greenfoot-objecten. In de volgende hoofdstukken zullen we die dan ook regelmatig tegenkomen. Alle objecten in een Greenfoot-wereld hebben deze methode `act`. Door het aanroepen van de methode `act` geef je in wezen het object de opdracht 'doe wat je wilt doen'. Als je de methode aangeroepen hebt voor jouw wombat, zul je gezien hebben dat de methode `act` er zo ongeveer voor zorgt dat de wombat de volgende dingen doet:

Concept:
Objecten die in een wereld geplaatst kunnen worden, heten acteurs.

- Als de wombat boven op een blad zit, eet deze het blad op.
- Als de wombat het blad niet opeet, zal hij vooruit lopen als dat mogelijk is.
- Als de wombat niet vooruit kan lopen, loopt hij naar links.

Bij de experimenten in de eerdere oefeningen heb je ook kunnen zien dat de knop *Act* gewoon de methode `act` van de acteurs in de wereld aanroept. Het enige verschil met het aanroepen van de methode via het objectmenu is dat de knop *Act* de methode `act` van álle objecten in de wereld aanroept. Wanneer de methode in het objectmenu aangeroepen wordt, heeft dit alleen effect op het betreffende object.

Met de knop *Run* wordt de methode `act` steeds opnieuw voor alle objecten aangeroepen, tot je klikt op de knop *Pause*.

Laten we dit allemaal nu eens bekijken in de context van een ander scenario.

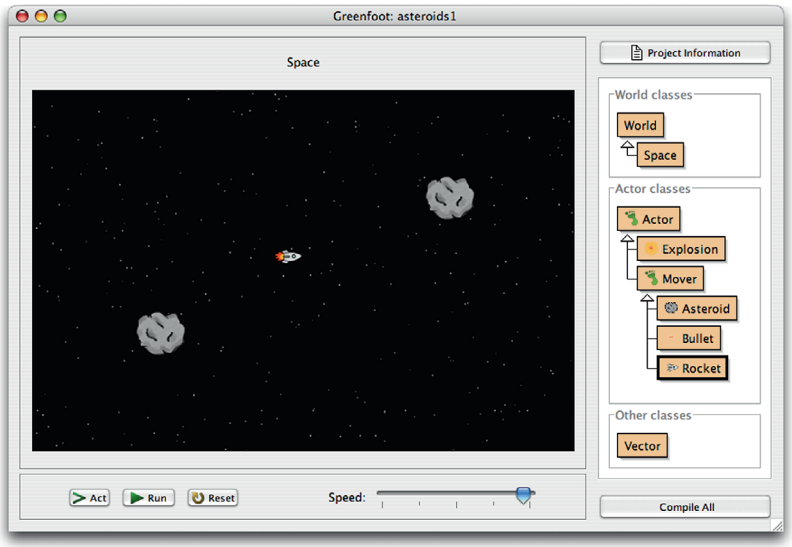
1.7 Een tweede voorbeeld

Open een ander scenario, *asteroids-1* in de map *chapter01* van de map *book-scenarios*. Dit scenario ziet er ongeveer uit als figuur 1.7, behalve dat je nog geen raket of de asteroïden op het scherm ziet.

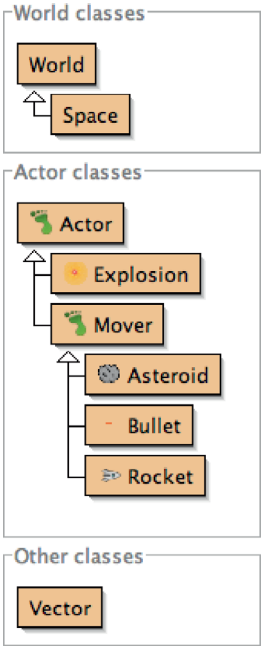
1.8 Wat zie je in het klassendiagram?

Laten we eerst eens beter kijken naar het klassendiagram (figuur 1.8). Bovenin zie je de twee klassen met de namen `World` en `Space`, die met elkaar verbonden zijn door een pijl.

De klasse `World` bestaat in alle Greenfoot-scenario's en is een in Greenfoot ingebouwde klasse. De klasse daaronder, in dit geval de klasse `Space`, beschrijft de specifieke wereld voor dit bepaalde scenario. De naam kan in elk scenario verschillen, maar bij elk scenario wordt een specifieke wereld gebruikt.



Figuur 1.7:
Het scenario asteroids1



Figuur 1.8:
Een klassendiagram

De pijl betekent 'is-een-relatie': `Space` is een `World` als het gaat om Greenfoot-werelden: `Space` is hier een specifieke Greenfoot-wereld. We kunnen ook zeggen dat `Space` een subklasse is van de klasse `World`.

Meestal zal het niet nodig zijn om objecten van wereldklassen te maken; Greenfoot doet dat zelf voor ons. Wanneer we scenario openen, creëert Greenfoot automatisch een object van de subklasse van de klasse `World`. Dat object wordt dan als wereld weergegeven in het hoofdvenster van Greenfoot. De grote zwarte afbeelding van de ruimte is een object van de klasse `Space`.