Opdracht 5: Dodo wordt slimmer

– Algoritmisch Denken en Gestructureerd Programmeren in Greenfoot –

1 Inleiding

Soms kan het handig zijn dat een object een 'geheugen' heeft. Bijvoorbeeld, dat Mimi kan bijhouden hoeveel eieren ze gelegd heeft. In deze opdracht leer je gebruik maken van variabelen om informatie te onthouden. We gaan Mimi slimmer maken!

We gaan nu complexe (samengestelde) opdrachten maken. Voor complexe opdrachten geldt dat het erg lastig is om de herhaling van de *Run* te gebruiken om deze uit te voeren. Daarom mag je in deze opdracht van jouw algoritme een eigen methode maken waarin je, als dat nodig is, gebruik kunt maken van **`while`**-loops.

2 Leerdoelen

Na het voltooien van deze opdracht kun je:

- uitleggen waarvoor **variabelen** gebruikt worden;
- naamgevingsafspraken voor variabelen benoemen;
- de stappen benoemen en uitvoeren (zoals declaratie en initialisatie) die nodig zijn voor het gebruik van een variabele;
- variabelen **initialiseren**;
- de **waardetoekenningsoperator** '=' gebruiken;
- de **vergelijkingsoperatoren** '==', '!=', '<', '<=', '>' en '>=' gebruiken;
- de **operatoren** '+, '-', '$*$', '/' en '%' gebruiken;
- de **verhogingsoperator** '++' (en verlagingsoperator '$--$') gebruiken;
- in eigen woorden uitleggen wat de rol van een **counter**-variabele is in een **`while`**-loop;
- de verschillende waarden die een variabele aanneemt kunnen bepalen gedurende de uitvoering van een (deel van een) programma (tracing).

3 Instructies

Bij deze opdracht ga je verder met jouw code uit opdracht 4. Je hebt dus het scenario nodig dat je na opdracht 4 hebt opgeslagen `Opdr4_jouwNaam`. Als eerste gaan we hiervan een kopie maken zodat je altijd terug kan naar jouw oorspronkelijke bestand:

- Open het scenario dat je na opdracht 4 hebt opgeslagen `Opdr4_jouwNaam`.
- Kies in Greenfoot 'Scenario' in het bovenste menu, en dan 'Save As ...'.
- Controleer dat je in de map staat waar je jouw werk wilt opslaan.

- Vul de bestandsnaam aan met jouw eigen naam en het opgavenummer, bijvoorbeeld: `Opdr5_Michel`.

Je zult ook een paar vragen beantwoorden. Vragen met een '(IN) ' moeten 'IN'geleverd worden. Voor die vragen heb je nodig:

- pen en papier om de stroomdiagrammen te tekenen die ingeleverd moeten worden ('(IN) '),
- een document (bijvoorbeeld, Word) open waarin je de antwoorden voor de '(IN) ' vragen kunt typen.

De overige vragen (die niet ingeleverd hoeven te worden) moet je met jouw programmeerpartner bespreken en jullie antwoord kort noteren op het opgavenblaadje.

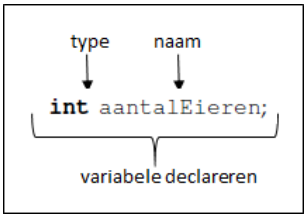
Een opmerking vooraf: in deze opdracht mag alléén de klasse `MyDodo` worden aangepast.

4 Uitleg

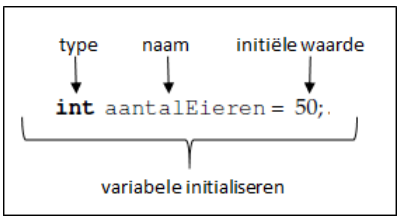
Variabelen
Variabelen worden gebruikt om gegevens in op te slaan.
Een variabele heeft een:

- Type: bijvoorbeeld **`int`**, **`boolean`**, `String`, of zoals we straks zullen zien, een klasse of een ander object. Het type ven een variabele bepaalt waar voor soort waarden deze variabele kan bevatten.
- Naam: bijvoorbeeld `Mimi` of `aantalEieren`.
- Initiële waarde: optioneel (je mag er ook later pas een waarde aan toe kennen).

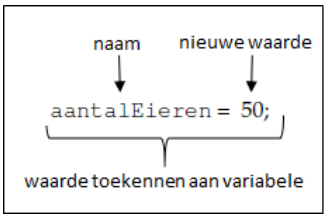
Variabele declareren:
Om een variabele te kunnen gebruiken, moet je hem eerst aanmaken, of *declareren*:



Meestal geef je deze gelijktijdig een waarde, oftewel de variabele *initialiseren*:



Waarde toekennen:
Een waarde toekennen aan een variabele doe je met '='. Het '=' teken spreek je uit als 'wordt'.



Figuur 1: De variabele `aantalEieren` wordt 50

Je kunt alleen een waarde aan een variabele toekennen als deze waarde van **hetzelfde** type is als de variabele. Dat betekent dat het type aan de linkerkant van de '=' moet gelijk zijn aan het type aan de rechterkant. In bovenstaand voorbeeld is het type van de:

- linkerkant **`int`**, want het type van `aantalEieren` is een **`int`**
- rechterkant ook **`int`**, want '50' is een **`int`**

Het gebruik van een variabele:
Bij het gebruik van (de waarde van) een variabele, hoef je het type er niet meer bij te vermelden. (Net als bij het aanroepen van een methode waar je ook niet de signatuur hoeft te herhalen bij een aanroep. Sterker nog, Java staat niet eens toe dat je bij gebruik ook het type opschrijft.) Wat je precies met een variabele kan doen (welke *operaties* je kunt toepassen) is mede afhankelijk van het type. Variabelen waarin teksten zijn opgeslagen kun je bijvoorbeeld aan elkaar plakken. Getallen kun je vergelijken, vermenigvuldigen, optellen, ophogen.
Voorbeelden zijn:

- `aantalDozijen = aantalEieren / 12;`
  de variabele `aantalDozijnen` krijgt de waarde van aantalEieren gedeeld door 12. Opmerking: de deling die hier gebruikt wordt is een zogenaamde *integerdeling*, dat wil zeggen dat de rest van de deling genegeerd (weggegegoid) wordt.
- `aantalEieren = aantalEieren + 1;`
  de variabele `aantalEieren` wordt verhoogd met 1 (bijvoorbeeld als Mimi nog een ei legt).
- `aantalEieren = aantalEieren ++;`
  de variabele `aantalEieren` wordt verhoogd met 1 ('++' is hetzelfde als '+1').

Een variabele kan je ook als parameter aan een methode meegeven. Bijvoorbeeld:
`legAantalEieren( aantalEieren );`
In deze aanroep van de methode `legAantalEieren` wordt de variabele `aantalEieren` als parameter meegegeven. Aan de hand van deze parameter weet Mimi hoeveel eieren ze moet leggen.

Levensduur:
Een variabele die in een methode is gecreëerd wordt is na de aanroep van die methode niet meer geldig en wordt vernietigd. Deze variabelen hebben dus een beperkte levensduur (soms ook wel *scope* genoemd). De levensduur begint bij de plek waar op variabele gedeclareerd is en eindigt daar waar de methode ophoudt[1]. De levensduur kun je in Greenfoot herkennen doordat in de editor gebruik wordt gemaakt van verschillende achtergrondkleuren. Bijvoorbeeld, in het volgende plaatje is de scope van de variabele **`int`** `stepsTaken` beperkt tot het witte gebied en alle gebieden die binnen het witte gebied vallen, in dit geval het roze gebied.

```
public int walkAndCountSteps(){
    int stepsTaken=0;

    while( canMove() ){
        stepsTaken++;
        move();
    }
    return stepsTaken;
}
```

Ook parameters hebben een levensduur: deze is beperkt tot body van een methode (of constructor). Bijvoorbeeld, in het volgende plaatje is te zien dat de scope van de parameter **`int`** `nrStepsToTake` beperkt is tot het witte gebied (en het roze gebied, want dat valt binnen het witte gebied) van de methode.

```
public void takeSteps (int nrStepsToTake){
    int stepsTaken = 0;

    while (stepsTaken < nrStepsToTake){
        stepsTaken++
        move();
    }

}
```

Variabelen die in een klasse gedeclareerd zijn (dus niet in een bepaalde methode maar daarbuiten) blijven gedurende de gehele levensduur van het object bestaan (hierover leer je meer in de volgende opdracht).

Naamgevingsafspraken
De naam van een variabele:

- is betekenisvol: hij komt overeen met wat de variabele betekent (uitzondering: de naam van een teller-variabele in een loop mag één letter zijn, zoals `i`);
- bestaat uit één of meer zelfstandige naamwoorden;
- bestaat uit letters en cijfers: bevat geen spaties, komma's of andere 'rare' karakters ('_' uitgezonderd);
- is geschreven in lowerCaseCamel: begint met een kleine letter, elk volgend 'woord' begint met een hoofdletter;
- bijvoorbeeld: `nrEggsFound`.

Voorbeeld:
We schrijven een methode die het kwadraat van een getal oplevert:

```
/**
* This method returns the square of a given number
*/
public int square (int number){
        // declareren en initialiseren van de uitkomst, kwadraat van number
    int result = number*number;

        // de methode levert het resultaat van het kwadrateren op
    return result;
}
```

Toevoeging:
We noemen een variabele die in een methode gedeclareerd is (en dus alleen binnen die methode gebruikt kan worden) een *lokale variabele*. In dit voorbeeld is **`int`** `result` een lokale variabele omdat deze in de methode `square` gedeclareerd is.

Vergelijkingsoperatoren:
De volgende operatoren vergelijken getallen:

Operator	Betekenis	Voorbeeld
==	is gelijk aan	`getal` == 4
!=	is NIET gelijk aan	`getal1` != `getal2`
>	is groter dan	`getal` > 3
>=	is groter of gelijk aan	`getal1` >= `getal2`
<	is kleiner dan	`getal` < 5
<=	is kleiner of gelijk aan	`getal` <= 5

Hier komt altijd een **`boolean`** als resultaat uit (dus **`true`** of **`false`**).

Rekenkundige bewerkingen:
De volgende operatoren voeren rekenkundige bewerkingen uit op getallen:

Operator	Betekenis	Voorbeeld
+	optellen	`uitkomst` = `getal` + 4
−	aftrekken	`uitkomst` = `getal` − 4
*	vermenigvuldigen	`uitkomst` = `getal` * 5
/	delen door	`uitkomst` = `getal` / 5

Verhogings- en verlagingsoperatoren:
De volgende operatoren verhogen en verlagen de waarde van een variabele met één:

Operator	Betekenis	Voorbeeld
++	met één verhogen	`uitkomst` ++
−−	met één verlagen	`uitkomst` −−

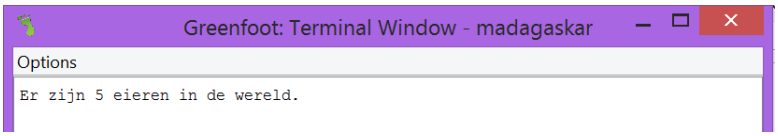
Voorbeelden:
Stel Mimi heeft 4 eieren gevonden: **`int`** `nrEggsFound` = 4;
Ze vindt er daarna nog twee: `nrEggsFound` = `nrEggsFound` + 2;
Daarna raakt ze er eentje kwijt: `nrEggsFound`−−;
Om te checken of Mimi er nu inderdaad vijf heeft, gebruiken we: `nrEggsFound` == 5. Dit is inderdaad **`true`**.

Console
Om een venster met tekst te laten verschijnen maak je gebruik een standaard Java methode: `System.out.println( String );`. De methode krijgt als parameter een `String` (tekst) mee. Het venster dat verschijnt noemen we een console. Een `String` als parameter meegeven gaat net als bij opdracht 3, opgave "Complimentje geven":

- Letterlijke tekst die moet verschijnen wordt tussen aanhalingstekens " en " gezet.
- Om de waarde van een variabele (zoals een getal) te tonen, zijn de aanhalingstekens niet nodig en wordt gewoon de variabelenaam gebruikt.
- Om teksten en variabelen te combineren wordt gebruik gemaakt van een '+'.

Voorbeeld:
Stel je wilt de tekst samen met de waarde van een variabele (bijvoorbeeld `aantalEieren`) tonen. Dit kan met de volgende aanroep:
`System.out.println("Er zijn "+ aantalEieren + " eieren in de wereld");`

Als bijvoorbeeld `aantalEieren` gelijk is aan 5, dan wordt het volgende in de console getoond:



Figuur 2: Console na aanroep van `println`

Toevoeging:
In `https://docs.oracle.com/javase/7/docs/api/java/io/PrintStream.html#println(java.lang.String)` vind je meer informatie over de Java methode `println`.

5 Opgaven

5.1 Variabelen

5.1.1 Variabelen nalopen

Tracing
Het nalopen van de variabelen en het bekijken van hun waardes op bepaalde momenten in het programma heet *tracing*. Tracing is een handige vaardigheid bij het opsporen van fouten in de code.
Voorbeeld:
Voor de volgende programmacode gaan we de variabelen `getal1`, `getal2` en `getal3` tracen:

```
int getal1 = 2;
int getal2 = 3;
int getal3 = getal1 * getal2;
getal2 = getal3 - getal1;
getal1 = getal1 + getal2 + getal3;
getal3 = getal2 * getal1;
```

We gebruiken een tabel om de waardeveranderingen van de variabelen bij te houden:

	Waarde nadat statement is uitgevoerd		
Statement	**getal1**	**getal2**	**getal3**
`int getal1 = 2;`	**2**		
`int getal2 = 3;`	2	**3**	
`int getal3 = getal1 * getal2;`	2	3	**6**
`getal2 = getal3 - getal1;`	2	**4**	6
`getal1 = getal1 + getal2 + getal3;`	**12**	4	6
`getal3 = getal2 * getal1;`	12	4	**48**

Aan het einde van het programma is de waarde van `getal1` gelijk aan 12, de waarde van `getal2` is gelijk aan 4 en de waarde van `getal3` is gelijk aan 48.

We gaan nu wat oefenen met variabelen en operaties. Bekijk de volgende stukken code. Geef telkens aan wat de waardes van de variabelen na het uitvoeren van de code zijn.

- Bedenk eerst zelf wat je denkt het antwoord is.
- Gebruik Greenfoot om jouw antwoord te controleren:
  1. Maak een **`void`** `oefenCodeDoorlopen()` methode waarin je de code zet.

2. Laat de waarden van de variabelen (op verschillende plekken in de code) afdrukken. Bijvoorbeeld, je kunt de waarde van een variabele `aantalEierenGevonden` in een apart venstertje tonen (afdrukken) met:
   `System.out.println("Waarde van aantalEierenGevonden: "+ aantalEierenGevonden);`

- Is het antwoord anders dan je verwacht had, beredeneer dan hoe de code werkt.

Onderstaande opgaven moet je zonder problemen kunnen beantwoorden. Lukt dat niet, dan heb je dit onderwerp kennelijk nog niet helemaal goed begrepen. Zorg ervoor dat je problemen met de stof hebt opgelost voordat je verder gaat met de volgende opgaven. Dat voorkomt dat je straks onnodige fouten maakt. De tijd nemen om dit nu goed te begrijpen wordt straks dubbel uitbetaald!

1. Wat wordt de waarde van **int** `aantalEierenGevonden`?

```
int aantalEierenGevonden = 3;
aantalEierenGevonden ++;
```

2. Wat wordt de waarde van **int** `aantalEierenGevonden`?

```
int aantalEierenGevonden = 2;
aantalEierenGevonden = aantalEierenGevonden + 4;
```

3. Wat wordt de waarde van **int** `aantalEierenGevonden`?

```
int aantalEierenGevonden = 1;
aantalEierenGevonden --;
```

4. Wat worden de waardes van **int** `getal1` en **int** `getal2`?

```
int getal1 = 2;
int getal2 = 4;

getal1 = getal1 + getal2;
getal2 = getal1 + getal2;
```

5. Wat worden de waardes van **int** `getal1` en **int** `getal2`?

```
int getal1 = 3;
int getal2 = 4;

getal1 ++;
if( getal1 != getal2 ){
     getal1++;
 } else {
     getal2 ++;
 }
```

6. Wat worden de waardes van **int** `getal1` en **int** `getal2`?

```
int getal1 = 2;
int getal2 = 4;

getal1 = getal2;
getal2 = getal1 * getal1;
```

7. Wat worden de waardes van **int** `getal1` en **int** `getal2`?

```
int getal1 = 2;
int getal2 = 4;

getal1 = getal2;
getal2 = getal1;
```

8. Wat worden de waardes van **int** getal1 en **int** getal2?

```
int getal1 = 6;
int getal2 = 3;

getal1 = getal2;
getal2 = getal1;
if( getal1 == getal2 ){
    getal1 = getal1 + getal2;
}
```

9. Wat worden de waardes van **int** getal1 en **int** getal2?

```
int getal1 = 6;
int getal2 = 3;

getal1 = getal2;
getal2 = getal1 * getal1;

if( getal1 < getal2 ){
    getal1 = getal1 + getal2;
} else {
    getal2 ++;
}
```

10. Wat wordt de waarde van **int** getal?

```
int getal = 3;
while( getal < 10 ) {
    getal = getal + 2;
}
```

11. Wat wordt de waarde van **int** getal3?

```
int getal1 = 10;
int getal2 = 8;
int getal3 = 4;

getal3 = doeIetsMetGetallen( getal1, getal2 );
```

Waarbij:

```
public int doeIetsMetGetallen ( int a, int b ) {
    int uitkomst = (a + b)/2;
    return uitkomst;
}
```

12. Wat worden de waardes van **int** getal1 en **int** getal2? Omschrijf in jouw eigen woorden wat deze code met getal1 en getal2 doet. Wat is daarbij de rol van **int** getalTemp?

```
int getal1 = 6;
int getal2 = 3;
int getalTemp = 0;
```

```
getalTemp = getal1;
getal1 = getal2;
getal2 = getalTemp;
```

5.1.2 Draai naar het oosten

We gaan een methode schrijven die Mimi draait zodat ze naar het oosten kijkt.

1. Roep met de rechtermuisknop de methode **`int`** `getDirection()` van `Dodo` aan. Welk getal (resultaat) hoort bij de richting waar ze opkijkt?
2. Bekijk de waarde van de variabele **`int`** `myDirection`. Doe dit door met de rechtermuisknop op Mimi te klikken en dan 'Inspect' te kiezen.

   Gebruik 'Inspect' om de tabel verder in te vullen:

Kijkrichting	Resultaat `int getDirection()`	Waarde `int myDirection` bij 'Inspect'
North		
East		
South		
West		

3. Bepaal een algoritme dat Mimi draait zodat ze naar het oosten kijkt.
4. Teken het bijbehorende stroomdiagram.
5. Schrijf de bijbehorende methode **`void`** `faceEast()`.

   Tip: Bekijk de klasse `Dodo`. Bovenaan de klasse zijn zogenaamde klassenconstanten gedefiniëerd voor de vier windrichtingen:

   ```
   public static final int NORTH   = 0;
   public static final int EAST    = 1;
   public static final int SOUTH   = 2;
   public static final int WEST    = 3;
   ```

   Je kunt nu `Dodo.NORTH` gebruiken in plaats van '0'. Bijvoorbeeld:
   `if``( getDirection() == Dodo.NORTH )` om te controleren of Mimi naar het noorden kijkt. Dit maakt de code beter leesbaar en minder foutgevoelig. Wat klassenconstanten precies zijn leer je verder in opdracht 6, maar je kunt er nu al wel gebruik van maken. Gebruik deze klassenconstanten in jouw methode.
6. Schrijf commentaar bij jouw methode.
7. Compileer en test jouw methode door deze met de rechtermuisknop aan te roepen.

We hebben nu gezien hoe je een variabele kunt vergelijken met een getal.

5.1.3 Hoeveel graden draaien?

Schrijf een methode **`int`** `degreesNeededToTurnToFaceNorth()` die aangeeft hoeveel graden Mimi (tegen de klok in) moet draaien om naar het noorden te kijken. Lukt het je om dit zonder **`if`** .. `then` .. **`else`** te schrijven?

We hebben nu gezien hoe je met een variabele kunt rekenen.

5.1.4 Draai en stap

We gaan nu een generieke methode schrijven die Mimi eerst naar een bepaalde windrichting draait en haar daarna een stap in die richting laat zetten. De windrichting wordt als parameter aan deze methode meegegeven.

1. Bepaal een algoritme dat Mimi draait zodat ze naar een gegeven richting kijkt en dan in die richting stapt. Tips:
   (a) Wat voor een `type` krijgt de parameter jouw methode?
   (b) Bedenk eerst hoe je kunt bepalen of Mimi in de juiste richting kijkt.
   (c) Probeer een algoritme te bedenken waarbij je niet voor elke richting apart beschrijft wat er moet gebeuren. Tips:
   - Dit kun je met een **`while`** doen.
   - Gebruik hierbij wat je bedacht hebt in het vorige onderdeel 1b.
2. Teken het bijbehorende stroomdiagram.
3. Zet dit stroomdiagram om in code voor de methode **`void`** `turnToDirectionAndMove(` **`int`** `direction )`.
   - Maak eerst een aparte methode **`boolean`** `facingCorrectDirection(` **`int`** `direction )` die bepaalt of Mimi al de juiste kijkrichting heeft.
4. Schrijf commentaar bij jouw methode.
5. Compileer en test jouw methode door deze met de rechtermuisknop aan te roepen.

5.1.5 Ga naar een bepaalde locatie

Schrijf een methode **`void`** `goToLocation(` **`int`** `coordX, ` **`int`** `coordY )` die Mimi naar een bepaalde locatie met coördinaten (coordX, coordY) stuurt.

1. Bedenk een algoritme waarmee Mimi naar de gegeven locatie loopt. Tip: Het kan handig zijn om verschillende gevallen te onderscheiden.
2. (IN) Teken het bijbehorende stroomschema. Tip: Gebruik een aparte submethode (en dus diagram) om te controleren of Mimi al op de gewenste locatie staat.
3. Schrijf de bijbehorende methode. Voorzie deze ook van commentaar.
4. Compileer en test jouw methode door deze met de rechtermuisknop aan te roepen en verschillende waarden in te voeren. Werkt het ook goed als je (0,0) invoert? En (2,3)? En (11,11)? En (14,14)?
5. Pas jouw programma aan zodat dit ongeldige coördinaten goed afhandelt:
   (a) Schrijf een aparte submethode **`boolean`** `validCoordinates(`**`int`** `coordX, `**`int`** `coordY)` die controleert of de coördinaten geldig zijn.
   (b) Bij ongeldige coördinaten, toon de volgende foutmelding: `showError( "Invalid coordinates");`
   (c) Bij ongeldige invoer moet Mimi blijven staan (niks doen);
   (d) Jouw programma moet voor elke wereldgrootte werken, niet alleen voor een wereld van 12 bij 12. Vraag de breedte en hoogte van de wereld op. Tip: De breedte van de wereld krijg je door eerst de wereld op te vragen: `World world = getWorld( );` en daarna de breedte op te vragen: `world.getWidth( )`.
6. Test deze aanpassing.

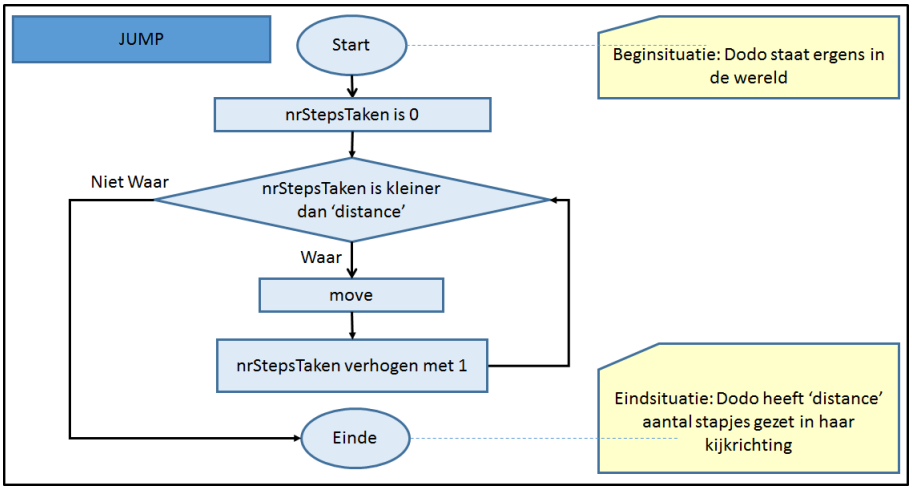
We hebben nu gezien hoe je variabelen kunt gebruiken en vergelijken. Ook hebben we gezien hoe je parameters, variabelen en getallen kunt combineren om ingewikkelde condities te maken.

5.2 Herhalingen tellen

Counter
Een variabele kan je gebruiken om te tellen hoe vaak een **`while`**-loop doorlopen is. Dan weet je namelijk als je klaar bent. Zo'n variabele noem je dan een *counter*.

Als voorbeeld kijken we naar de code van **`void`** `jump(` **`int`** `distance)` die we zijn tegengekomen in de klasse `MyDodo` van scenario 1. Deze methode laat Mimi herhaaldelijk stapjes zetten totdat ze de gevraagde afstand heeft overbrugd. De afstand is gelijk aan het aantal keren dat de **`while`**-loop doorlopen moet worden.

Stroomdiagram:



Toelichting stroomdiagram:

- Eerst wordt er een counter 'nrStepsTaken' geïnitialiseerd die bijhoudt hoeveel stappen er door Mimi zijn gezet.
- In de ruit wordt de conditie gecontroleerd.
  - Zolang de conditie 'Waar' is (dus 'nrStepsTaken' is kleiner dan de gegeven 'distance'), moeten er nog meer stappen gezet worden. Mimi zet een stap ('move') en wordt de counter 'nrStepsTaken' met ééntje verhoogd.
    Daarna wordt er teruggegaan naar de test in de ruit. Is de conditie 'nrStepsTaken is kleiner dan de gegeven distance' nog steeds 'Waar'? Dan wordt de pad van 'Waar' weer vervolgd (dit is een loop). Anders is de methode afgelopen.
  - Als de conditie 'Niet waar' is (dus 'nrStepsTaken' is groter of gelijk aan de gegeven 'distance'), dan hoeven er geen stappen meer gezet te worden. Dan is de methode afgelopen.

Code:

```
public void jump (int distance){
    int nrStepsTaken = 0;            //counter 'nrStepsTaken' initialiseren
    while( nrStepsTaken < distance){ // Onvoldoende stappen gezet? Ga dan door
        move();                      // zet een stap
        nrStepsTaken++;              // verhoog de counter met 1
    }
}
```

Toelichting code:

- Er wordt een counter gedeclareerd en geïnitialiseerd met de waarde 0: `int nrStepsTaken = 0;`.
- Eerst wordt de conditie `nrStepsTaken < distance` gecontroleerd.
  - Als de conditie **`true`** is (dus als `nrStepsTaken < distance ==` **`true`**), wordt de code tussen de accolades { en } uitgevoerd. In dit geval `move( )` gevolgd door het ophogen van de counter `nrStepsTaken++`.
    Daarna wordt er teruggegaan naar de controle van de conditie `nrStepsTaken < distance`. Als de conditie nog steeds **`true`** is, dan wordt de code tussen de accolades weer uitgevoerd (loop). Anders is de methode afgelopen.
  - Als de conditie **`false`** is (dus als `nrStepsTaken < distance ==` **`false`**), oftewel `nrStepsTaken` is groter of gelijk aan `distance`, dan is de methode afgelopen.

Toevoeging:
Het aantal keren dat de **`while`** loop nog doorlopen wordt is afhankelijk van de waarde van **`int`** `nrStepsTaken`. De variabele `nrStepsTaken` wordt per aanroep van de **`while`** telkens met één verhoogd. Op een gegeven moment zal `nrStepsTaken` dezelfde waarde hebben als `distance`. In dat geval zal de **`while`** niet meer uitgevoerd worden en is de methode afgelopen.

Een veelgemaakte fout is het vergeten om de counter in de loop te verhogen. Hierdoor komt het programma niet uit de loop en zal het eeuwig blijven herhalen.

5.2.1 (IN) While nalopen

We gaan nu wat oefenen met **`while`**. Net als bij de eerste opgave (opg. **??**) gaan we stukken code bekijken en nalopen wat er met de variabelen gebeurt. Dit geeft veel inzicht in de werking van een **`while`**. Daardoor zal je minder snel fouten maken. Bekijk de volgende stukken code en geef antwoord op de vragen. Gebruik Greenfoot om jouw antwoorden te controleren. Gebruik hiervoor de methode **`void`** `oefenCodeDoorlopen( )` die je in opg. **??** gemaakt hebt om de code in te plakken (en uit te voeren).

1. Wat zijn na afloop de waardes van `aantalStappenGezet` en `aantalStappenOmTeZetten`? Hoe vaak wordt `move();` aangeroepen?

```
int aantalStappenOmTeZetten = 8;
int aantalStappenGezet = 0;

while( aantalStappenGezet < aantalStappenOmTeZetten){
    move();
    aantalStappenGezet++;
}
```

2. Wat zijn na afloop de waardes van `counter` en `aantalStappenOmTeZetten`? Hoe vaak wordt `move();` aangeroepen? Is de code goed?

```
int aantalStappenOmTeZetten = 6;
int counter = 0;

while( counter <= aantalStappenOmTeZetten){
    move();
    counter++;
}
```

3. Wat zijn na afloop de waardes van `counter` en `aantalStappenOmTeZetten`? Waarom klopt de code nu wel?

```
int aantalStappenOmTeZetten = 4;
int counter = 0;

while( counter <= aantalStappenOmTeZetten-1){
    move();
    counter ++;
}
```

4. Vul de onderstaande tabel in voor elke keer dat de body van de while wordt uitgevoerd.

```
int getal1 = 2;
int getal2 = 5;
int counter = 0;

while( getal1 <= getal2){
     move();
     getal1 ++;
     counter ++;
 }
```

Aantal **while**-loops uitgevoerd	Waarde `getal1` na **while**	Waarde `getal2` na **while**
0		
1		
2		

5. Pas de onderstaande **conditie** van de **while** aan zodat `move( )` drie keer wordt aangeroepen:

```
int getal1 = 10;
int getal2 = 8;

while( getal1 > getal2 ){
    move();
    getal1--;
}
```

6. Bekijk de methode `jump` zoals in het bovenstaande theorieblok is aangegeven.
   (a) Voeg de methode **void** `jump(`**int** `distance)` toe aan `MyDodo`.
   (b) Hoe vaak wordt de code in de **while** van **void** `jump(`**int** `distance)` uitgevoerd als:
      i. **int** `distance` gelijk is aan 5,
      ii. **int** `distance` gelijk is aan 1?
      iii. **int** `distance` gelijk is aan 0?
      iv. `nrStepsTaken` gelijk is aan 3 en `distance` aan 5?

We hebben nu gezien hoe je een variabele kunt gebruiken om te bepalen hoe vaak een **while**-loop doorlopen moet worden.

5.2.2 Aantal stappen tot het einde van de wereld

Schrijf een methode die oplevert hoeveel stappen Mimi moet zetten vanaf haar huidige positie tot het einde van de wereld te komen. Om dit te doen laat je Mimi recht vooruit lopen (tot ze de rand van de wereld bereikt) en het aantal stappen tellen dat ze zet. Gebruik hiervoor een **`while`** en een counter. Onderaan de methode gebruik je een **`return`**-statement om het aantal stappen op te leveren. Zo kan de waarde in een andere deel van jouw programma gebruikt worden (want, als de method eindigd, dan bestaat de lokale variabele bestaat niet meer).

1. Bedenk een algoritme voor het tellen van het aantal stappen tot de rand van de wereld is bereikt. Gebruik een **`while`** en een counter-variabele. Tip: Laat je eventueel inspireren door **`void`** `jump( int distance )`.
2. Kies een geschikte naam voor de counter.
3. Teken het bijbehorende stroomdiagram.
4. Schrijf de bijbehorende methode **`int`** `walkToWorldEdgeAndCountSteps()` waarmee telt hoeveel stappen Mimi zet en dat getal oplevert.
5. Schrijf daar ook commentaar bij.
6. Compileer en test jouw methode door deze met de rechtermuisknop aan te roepen. Wat is het kleinste getal dat deze methode kan opleveren? En het grootste?

We hebben nu zelf een counter-variabele gebruikt om bij te houden hoe vaak een **`while`**-loop doorlopen wordt.

5.2.3 Aantal eieren tot de rand van de wereld tellen

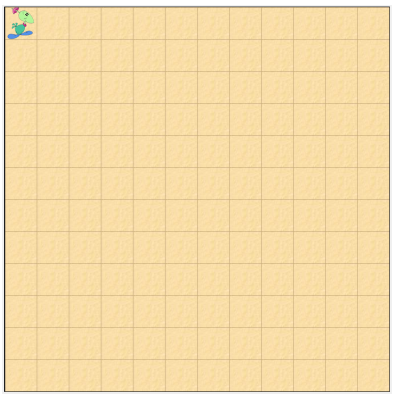
Schrijf een methode die oplevert hoeveel eieren er in een rij of kolom liggen. Om dit te doen laat je Mimi vooruit lopen (tot ze de rand van de wereld bereikt) en het aantal eieren tellen dat ze tegenkomt. Gebruik hiervoor een **`while`** en een counter. Onderaan de methode gebruik je een **`return`**-statement om het aantal eieren op te leveren.

1. Bedenk een algoritme voor het tellen van het aantal eieren die tussen Mimi en de rand van de wereld liggen. Gebruik een **`while`**-loop. Tip: Laat je eventueel inspireren door de methode **`void`** `walkToEdgeOfWorld()` die je zelf in opdracht 2 geschreven hebt.
2. Kies een geschikte variabelenaam om het aantal gevonden eieren bij te houden.
3. Teken het bijbehorende stroomdiagram. Lever het aantal gevonden eieren als resultaat op.
4. Schrijf de bijbehorende methode **`int`** `walkToWorldEdgeAndCountEggs()`.
5. Schrijf daar ook commentaar bij.
6. Compileer en test jouw methode door deze met de rechtermuisknop aan te roepen. Tip: Gebruik `println` om jouw variabele af te drukken. Zo kun je controleren of jouw methode goed werkt.

We hebben nu zelf een variabele gebruikt om bij te houden hoeveel eieren er gevonden zijn.

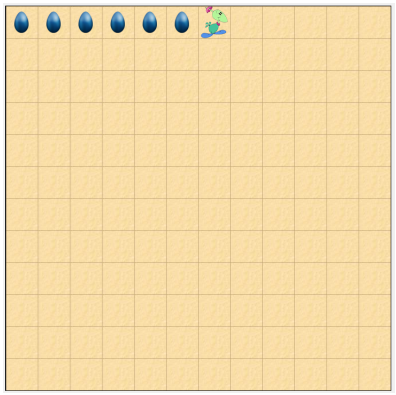
5.2.4 Spoor van eieren

We willen dat Mimi een spoor van een bepaald aantal eieren legt. De beginsituatie is als volgt:



Figuur 3: Beginsituatie

We schrijven een methode **`void`** `layTrailOfEggs(` **`int`** `nrOfEggsToLay`) waarmee Mimi een spoor van `nrOfEggsToLay` eieren achter zich laat. Dus, `layTrailOfEggs`(6) levert de volgende eindsituatie op:



Figuur 4: Eindsituatie nadat Mimi een spoor van 6 eieren legt

1. Bedenk een algoritme voor het achterlaten van een spoor van eieren. Gebruik een **`while`** met een counter-variabele.
2. Teken het bijbehorende stroomdiagram.
3. Schrijf de bijbehorende methode **`void`** `layTrailOfEggs(` **`int`** `nrOfEggsToLay`) waarmee Mimi een spoor van `nrOfEggsToLay` eieren legt.
4. Schrijf daar ook commentaar bij.
5. Compileer en test jouw methode door deze met de rechtermuisknop aan te roepen.
6. (IN) Als **`int`** `nrOfEggsToLay` gelijk is aan 6, hoe vaak wordt de **`while`** uitgevoerd?

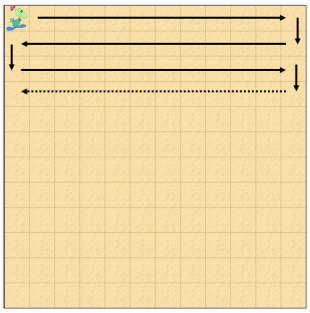
We hebben nu zelf een counter-variabele gebruikt om te bepalen hoe vaak een **`while`**-loop doorlopen moet worden.

5.3 Eieren in de wereld

We hebben nu een aantal basismethoden geschreven. Het is tijd voor een iets ingewikkeldere opdracht. Eerst leren we Mimi hoe ze kan tellen hoeveel eieren er in de hele wereld liggen. In de opgaven daarna leren we haar nog wat slimmere dingen.

5.3.1 Aantal eieren in de wereld tellen

We gaan een methode schrijven waarmee Mimi systematisch elk vak van de wereld bezoekt, zoals in het volgende plaatje aangegeven:



Figuur 5: Doorloop de hele wereld

Mimi loopt op deze wijze alle rijen af, en stopt als ze bij het laatste vakje is aangekomen. We delen het probleem op in een aantal deelproblemen:

1. Loop naar het beginpunt;
2. Loop naar het einde van de rij en tel de eieren;
3. Bepaal of het eindpunt bereikt is:
   (a) Zolang het eindpunt niet bereikt is, loop naar de volgende rij, draai en herhaal stap 2;
   (b) Is het eindpunt bereikt? Dan klaar.

Koppel de bovenstaande deelproblemen aan elkaar voor een totaaloplossing.

1. (IN) Maak eerst een schets van de stroomdiagram voor de totaaloplossing (stappen 1,2 en 3 hierboven).

   We pakken nu één voor één de deelproblemen aan:

2. Loop naar het beginpunt:
   Maak altijd zo veel mogelijk gebruik van methodes die je al eerder geschreven hebt. Je kunt die dan gewoon aanroepen. Dat is lekker makkelijk want die zijn ook al getest! Je hoeft er niet eens een stroomdiagram van te maken, want ook dat heb je al! In opgave 5.1.5 heb je de methode `goToLocation` geschreven. Beoordeel of je deze kunt hergebruiken.
3. Loop naar het einde van de rij en tel de eieren:
   In opgave 5.2.3 heb je de methode `int walkToWorldEdgeAndCountEggs()` geschreven en getest. Beoordeel of je deze kunt hergebruiken.
4. Bepaal of het eindpunt bereikt is:
   Schrijf een submethode om te bepalen of het eindpunt bereikt is. Dat kun je als volgt doen:
   (a) De `World` methodes `int getWidth()` en `int getHeight()` geven aan hoeveel cellen breed en hoog de wereld is. Roep met de rechtermuisknop beide methoden aan.

(b) Schrijf de coördinaten op van de vier hoeken van de wereld op.

(c) In welke hoek van de wereld zal Mimi staan als ze klaar is? Is dat altijd dezelfde hoek, ongeacht hoe groot de wereld is? Waar is dat afhankelijk van? Beschrijf dit met behulp van de woorden 'hoogte' en 'breedte' in plaats van echte getallen zoals '12'te gebruiken.

(d) Bedenk een algoritme dat bepaalt op welke coördinaten Mimi eindigt. Deze moet altijd werken, ongeacht hoe groot de wereld is. Tips:

- Maak gebruik van de `World` methodes **`int`** `getWidth( )` en **`int`** `getHeight( )`.
- Schrijf een methode **`boolean`** `isEven(`**`int`** `getal)` die aangeeft of een getal even of oneven is. Je kunt hiervoor modulo '%' (rest na deling) gebruiken. Een getal waarbij 'getal%2' gelijk is aan 0 is even. Is het ongelijk aan 0, dan is het getal oneven.

(e) Schrijf een methode die bepaalt of Mimi op het eindpunt staat.

5. Loop naar de volgende rij:
   Schrijf en test een submethode waarmee Mimi naar de volgende rij stapt en omdraait. Deze moet zowel aan de linker als de rechterkant van de wereld werken.

6. Koppel de bovenstaande deelproblemen aan elkaar voor een totaaloplossing:
   Schrijf een methode waarbij Mimi door de hele wereld loopt en eieren telt:

   (a) Bedenk een geschikte variabelenaam om het aantal gevonden eieren in de wereld bij te houden.

   (b) (IN) Pas jouw stroomschema (uit stap 1) aan voor de totaaloplossing, waarbij je gebruik maakt van de deeloplossingen die je zojuist gevonden hebt. Betrek ook de variabelen. Bedenk ook op welke moment(en) de variabele opgehoogd wordt.

   (c) Schrijf de bijbehorende methode **`void`** `walkThroughWorldAndCountEggs()`. Gebruik hiervoor een **`while`**. Controleer of de begin- en eindsituaties van hergebruikte methodes overeenkomen met jouw verwachting.

   (d) Toon na afloop het aantal gevonden eieren in een console met behulp van `println( )`.

   (e) Schrijf commentaar bij jouw methode.

   (f) Compileer en test jouw methode. Bepaal of het een totaaloplossing is.

Je hebt nu een totaaloplossing voor een ingewikkeld probleem gevonden. Dit heb je gedaan door deze in kleine delen op te splitsen, elk probleem afzonderlijk op te lossen, en dan bij elkaar te voegen tot een oplossing voor het probleem als geheel.

5.3.2 Bepaal de rij met de meeste eieren

Bepaal welke rij in de wereld de meeste eieren heeft. We laten Mimi hiervoor rij voor rij door de wereld heen lopen. Als ze de hele wereld bekeken heeft, toont ze een dialoog met daarin zowel welke het rijnummer van rij met de meeste eieren als de hoeveelheid eieren in die rij.

1. Bedenk een geschikt algoritme. Tips: Gebruik hiervoor een **`while`**. Laat Mimi per rij bepalen of deze meer eieren heeft dan de andere rijen die ze al gezien heeft.

2. Bedenk hoeveel variabelen je nodig zult hebben om waarden bij te houden. Tip: Je hebt waarschijnlijk 4 variabelen van type **`int`** nodig.

3. Teken in grote lijnen een stroomdiagram. Aandachtspunten:

   - Deel het probleem op in kleinere problemen.
   - Maak gebruik van methodes die je al eerder geschreven hebt.

- Teken voor die deelproblemen subdiagrammen (deze worden in de code submethodes die je los van elkaar kunt schrijven en testen).
- Maak jouw oplossing generiek door de hoogte van de wereld op te vragen (m.b.v. `getHeight( )`).
- Nadat alle rijen zijn doorlopen, toon je in een compliment-dialoog welk rij de meeste eieren heeft en hoeveel dat er zijn. Daarna mag het programma stoppen.

4. Schrijf voor ieder deelprobleem een aparte submethode.
5. Compileer en test jouw submethodes één voor één.
6. Schrijf een methode `findRowWithMostEggs( )` die overeenkomt met jouw stroomdiagram. Vanuit deze methode worden jouw submethodes aangeroepen.
7. Schrijf commentaar bij jouw methode.
8. Compileer en test jouw methode met de rechtermuisknop.
9. Roep de methode aan vanuit `act( )`.
10. Compileer, run en test jouw programma.
11. Heb je veel of weinig fouten gemaakt bij het programmeren van deze opgaven? Wat voor fouten heb je gemaakt? Wat kan je in het vervolg beter anders doen?

Typeconversie
Het type van een object of waarde kan in sommige gevallen omgezet worden in een ander type. Dit heet *typeconversie* (in het engels: type casting). Hiervoor wordt het nieuwe type tussen haakjes voor het object geschreven.

Wanneer zouden we dit willen gebruiken? Bijvoorbeeld, als je twee **`int`** getallen door elkaar deelt kan je een decimaal (**`double`**) waarde krijgen. Zonder typeconversie levert dit automatische een **`int`** op, niet een **`double`**.

Bijvoorbeeld, na het uitvoeren van de volgende code wordt '1' afgedrukt:

```
int number1 = 5;
int number2 = 3;

 System.out.println( number1 / number2 );
```

Waarom? Omdat `number1` en `number2` **`int`**s zijn, en dus ook het resultaat. 5 wordt gedeeld door 3 en wordt daarna naar beneden afgerond (eigenlijk worden de cijfers achter de komma weggegooid).

Maar, als je een **`double`** door een **`int`** deelt, dan wordt het resultaat wel een **`double`**. Dus, als je wilt delen, kun je typeconversie toepassen op de teller waarna het resultaat een **`double`** wordt. Dus, `System.out.println( (double)number1 / number2 );` drukt het volgende af: 1.6666666666666667.

Voorbeeld:
Hier volgt een voorbeeld voor MyDodo. Als we bijvoorbeeld de waarde van **`int`** `nrEggsFound` willen omzetten naar een **`double`**(decimaal getal) dan schrijven we (**`double`**) `nrEggsFound`).

5.3.3 Het gemiddelde aantal eieren per rij

Bepaal hoeveel eieren er per rij gemiddelde liggen. Deze opgave is vergelijkbaar met de vorige opgave. Omdat het gemiddelde (waarschijnlijk) een decimaal getal is, moet je gebruik maken van *casting*.

1. Introduceer een variabele met type **`double`** om het gemiddelde in op te slaan.
2. Maak met een type cast van **`int`** `nrEggsFound` een **`double`**.
3. Reken het gemiddelde uit.
4. Toon het resultaat in een console.

6 Samenvatting

Je hebt geleerd:

- variabelen te gebruiken om waardes bij te houden;
- rekenen met variabelen;
- variabelen gebruiken in condities;
- hoe je een teller in een **`while`**-loop kunt gebruiken;
- hoe je (eigen) methodes hergebruikt in andere delen van jouw code.

7 Jouw werk opslaan

Je bent klaar met de vijfde opdracht. Sla je werk op, want je hebt het nodig voor de volgende opdrachten.

1. Kies in Greenfoot 'Scenario' in het bovenste menu, en dan 'Save As ...'.
2. Vul de bestandsnaam aan met jouw eigen naam en het opgavenummer, bijvoorbeeld: `Opdr5_Michel`.

Alle onderdelen van het scenario bevinden zich nu in een map die dezelfde naam heeft als de naam die je hebt gekozen bij 'Save As ...'.