# Opdracht 5: Dodo heeft plannen

Algoritmisch Denken en Gestructureerd Programmeren (in Greenfoot)



## Inhoudsopgave

# Inleiding

Bij de vorige opdrachten heb je jouw eigen (generieke) oplossingen bedacht en die geïmplementeerd. Je hebt ook geleerd hoe je standaard *plannen* kunt aanpassen en gebruiken.

In de volgende opdracht leer je Mimi ingewikkeldere taken uitvoeren, gebaseerd op complexere *plannen*. Je leert welke plannen informatici vaak gebruiken en gaat die plannen zelf gebruiken als onderdeel van jouw eigen oplossingen. Om dat te doen zul je opgedane kennis uit de vorige opdrachten moeten toepassen en wellicht ook combineren om te komen tot een oplossing voor de problemen in deze opdracht.

## Doelen

Het doel van deze opdracht is: **Standaard plannen toepassen**.

# Leerdoelen

Na het voltooien van deze opdracht kun je:

- standaard **plannen** herkennen en toepassen: voor een verzameling elementen het maximum/-minimum bepalen en de som en het gemiddelde berekenen;
- **type-casting** toepassen;
- een **generiek algoritme** ontwikkelen en implementeren;
- een (complex) probleem opdelen in deelproblemen en die afzonderlijk oplossen.

# Instructies

In deze opdracht ga je verder met jouw eigen code uit de vorige opdracht. Maak een kopie van jouw scenario zodat je altijd nog terug kan naar een vorige versie. Een kopie maken doe je zo:

- Open jouw scenario van de vorige opdracht.
- In de Greenfoot menu, bovenaan het scherm, selecteer je 'Scenario' en dan 'Save As ...'.
- Ga naar de folder waarin je jouw werk voor opdracht 5 wilt opslaan.
- Kies een bestandsnaam met jouw eigen naam en opdrachtnummer, bijvoorbeeld: `Opdr5_John`.

Tijdens de opdracht zul je wat vragen moeten beantwoorden. Je moet het volgende inleveren:

- Alle stroomdiagrammen: gebruik potlood en papier, of maak gebruik van de software op `https://www.draw.io/`;
- Jouw code: het bestand `MyDodo.jav` bevat al jouw code en moet ingeleverd worden;
- Het reflectieblad: vul in en lever het in.

Je moet al jouw antwoorden met je partner te bespreken. Noteer kort jullie antwoorden.

Er zijn drie soorten taken:

| | |
|---|---|
|  | **Aanbevolen**. Leerlingen met weinig programmeerervaring, of leerlingen die wat meer oefening nodig hebben, moeten deze taken allemaal afmaken. |
|  | **Verplicht**. Iedereen moet deze taken afmaken. |
|  | **Uitdagend**. Complexere taken, ontwikkeld voor leerlingen die de 2-ster opdrachten voltooid hebben en klaar zijn voor een grotere uitdaging. |

Leerlingen die 1-ster taken overslaan moeten alle 3-ster taken maken.
**Opmerking:**

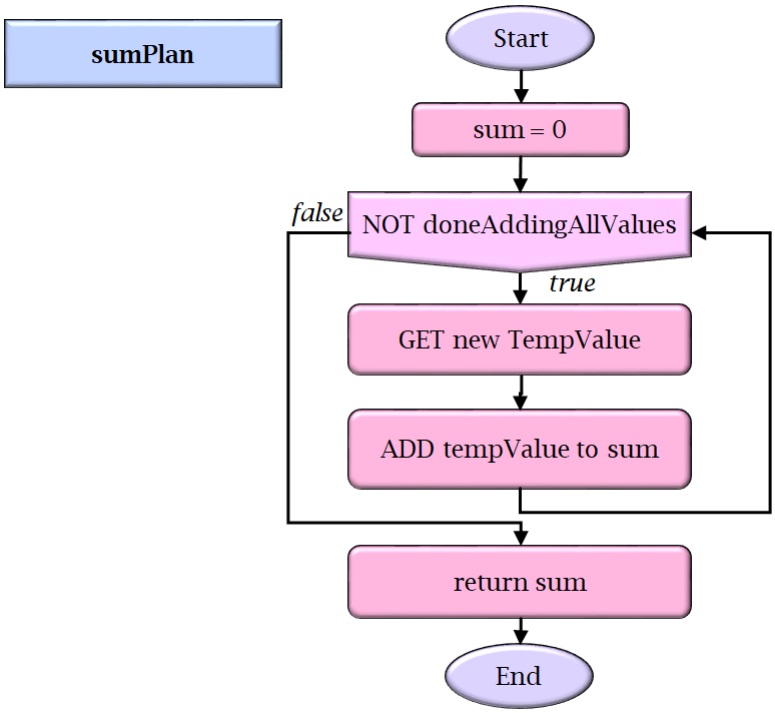
- In deze opdracht mag je alleen aanpassingen maken in `MyDodo`;
- Je mag methodes gebruiken uit de `MyDodo` of `Dodo` klasse, maar niet uit de `Actor` klasse;
- 'Teleporteren' is niet toegestaan: als Mimi ergens moet zijn, dan moet ze daar zelf stap voor stap heenlopen (ze mag er niet in één keer naar toe springen).

# Theorie

## Theorie 5.1: SommeerPlan

Het totaal berekenen (sommeren) van een aantal waarden (hieronder ook wel elementen genoemd)kun je in verschillende situaties tegenkomen. Het is op zich niet lastig om te doen, maar ook hierbij is een foutje gauw gemaakt. Het *sommeerplan* lijkt veel op het *telplan* uit Theorie 4.7. Een sommeerplan heeft de volgende structuur:

- Declareer de variabelen die je nodig hebt en initialiseer deze als volgt:
  - voor het bijhouden van het totaal: deze variabele is 0 aan het begin;
  - voor de huidige waarde: deze variabele wijst naar het eerste element.
- Controleer of je klaar bent, dus of je alle elementen doorlopen hebt.
- Binnen de loop: bepaal welke waarde opgeteld moet worden bij de som. Verhoog de som met deze waarde.
- Aan het einde van de methode: lever de som op.




Figuur 1: Stroomdiagram voor *sommeerplan*

```
/**
 * Sommeerplan: Raamwerk voor het vinden van de som van een aantal waarden
 */
public int sumPlan( ) {
   int sum = 0;                // de som begint op 0
   while ( elements left? ) { // controleer of je klaar bent
      int currentValue = ...  // verkrijg de waarde van het huidige element
      sum += currentValue;    // tel de huidige waarde op bij de som
      ...                     // ga naar het volgende element
   }
   return sum;
}
```

## Voorbeeld:

Als concreet voorbeeld bekijken we de code voor `sumEggValues( )`.

```
/**
 * Example of a sum plan: summing all the egg values in a row
 */
public int sumEggValues( ) {
   int sumOfEggValues = 0;     // set counter to 0

   while ( !borderAhead( ) ) {                // check if more steps can be taken
      Egg currentEgg  = getEgg();             // get the Egg which Dodo is standing on
      int currentEggValue = currentEgg.getValue();// get the value of the current egg
      sumOfEggValues += currentEggValue;      // add the egg value to the sum (running
          total)
```

```
        move();                                 // move to the next cell
    }

    return sumOfEggValues;
}
```

Mimi berekent het totaal van alle ei-waarden. Dat doet ze zo: Het huidige totaal wordt bijgehouden in `sumOfEggValues`. Voor elk ei dat ze vindt, bepaalt ze de waarde (opgeslagen in `currentEggValue`). Ze telt deze waarde op bij de huidige som, met behulp van `sumOfEggValues += currentEggValue;`. Dit laatste betekent hetzelfde als `sumOfEggValues = sumOfEggValues + currentEggValue;`.

## Theorie 5.2: Minimum/Maximum plan

Om de kleinste (minimum) of de grootste (maximum) van een verzameling te vinden kun je gebruik maken van het *Min/Max Plan*.

De stappen voor de *Max plan* zijn:

a) Declareer en initialiseer de variabelen die je nodig hebt:

   i) De waarde van het huidige element, bijvoorbeeld `huidigeElement`.

   ii) De grootste waarde tot nu toe gevonden, bijvoorbeeld `huidigeMax`.

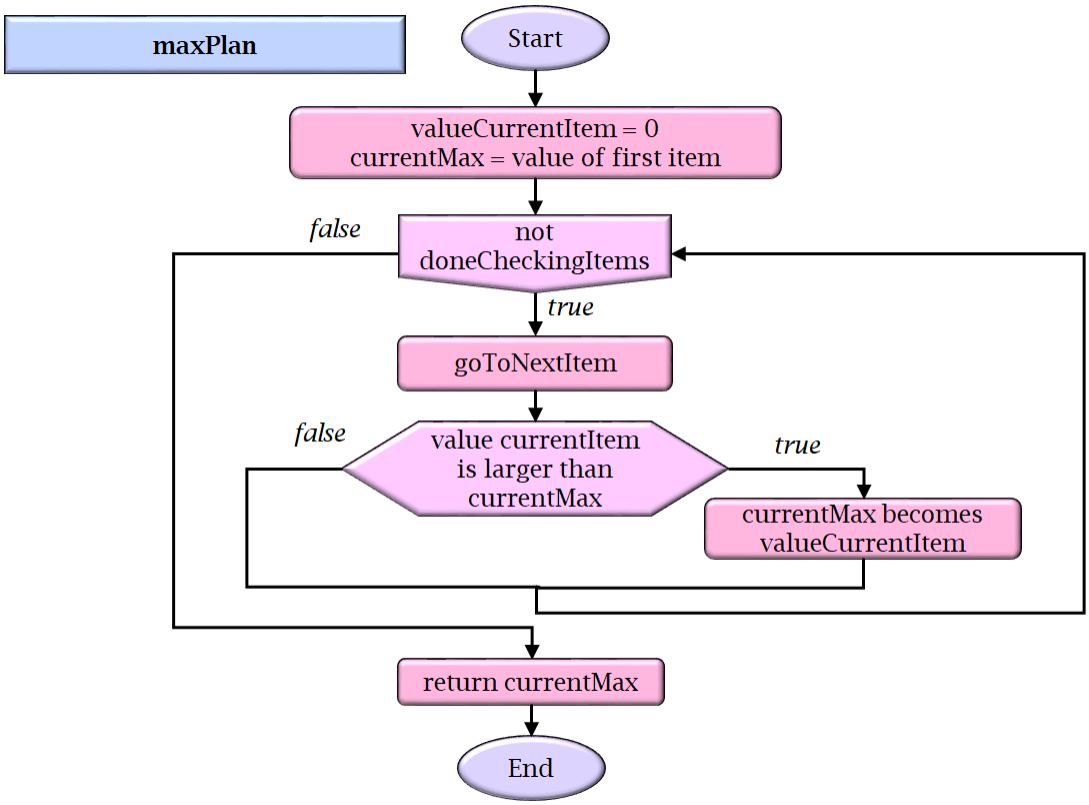
   Gebruik het eerste element uit de verzameling als beginwaarde van `huidigeMax`.

b) Doorloop alle elementen met behulp van een `while`-loop (of, zoals we in het volgende hoofdstuk zullen zien, een `for-each`-loop).

c) Vergelijk het huidige element (`huidigeElement`) met de grootste waarde die je tot nu toe gevonden hebt (`huidigeMax`).

d) Als het huidige element groter is dan de huidige grootste dan heb je nu een grotere waarde gevonden. In dat geval krijgt `huidigeMax` de waarde van `huidigeElement`.

Het vinden van het minimum gaat op ongeveer dezelfde manier.




Figuur 2: Stroomdiagram voor de Max Plan

In het volgende voorbeeld bepalen we de rij met de meeste eieren. De rijen worden doorlopen met behulp van een conditie-gestuurde herhaling: `boolean doneCheckingRows`.

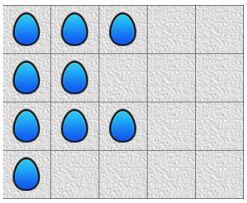
```
public int findMaxEggsInARow() {
   // declare variables needed
   int nrOfEggsInCurrentRow = 0; // value of current row
   int currentMaxEggs;           // the maximum thus far

   // use the first value encountered as the initial value for the max
   currentMaxEggs = countEggsInRow();

   while( ! doneCheckingRows() ){  // check if there are rows still to be checked
      stepDownToNextRow();         // go to next row

      nrOfEggsInCurrentRow = countEggsInRow();     // get value of current row
      if( nrOfEggsInCurrentRow > currentMaxEggs ){ //compare current to max
           // if current is the largest so far, store the new value as the max
           currentMaxEggs = nrOfEggsInCurrentRow;
      }
   }
   return currentMaxEggs;
}
```

De variabele `currentMaxEggs` wordt gebruikt om het huidige maximum bij te houden. Deze wordt geïnitialiseerd op het aantal eieren in de eerste rij. Elke volgende waarde wordt vergeleken met het huidige maximum. Als een grotere waarde gevonden wordt, dan wordt dit de nieuwe waarde van het huidige maximum en daarmee ook gebruikt voor de volgende vergelijkingen.



Figuur 3: Meer voorkomens van dezelfde maximumwaarde

Stel dat de maximumwaarde vaker voorkomt. Bijvoorbeeld, het scenario in figuur 3 toont twee rijen met hetzelfde maximum (namelijk drie eieren in één rij). Welke rij heeft nu de meeste eieren? Er zijn meerdere antwoorden mogelijk. Soms is het niet van belang welk maximum gekozen wordt. In andere gevallen zul je afhankelijk van het probleem een geschikte keuze moeten maken, bijvoorbeeld het eerste of het laatste maximum. De keuze omschrijf je in JavaDoc commentaar.

## Theorie 5.3: Type casting

In sommige gevallen kan het type van een waarde omgezet worden naar een ander type. Je kunt bijvoorbeeld een getal `int` omzetten in een kommagetal (`double` in Java). Dit heet *type casting*.

Als je een getal (`int`) deelt door een ander geheel getal (`int`), dan is het antwoord vaak een kommagetal, bijvoorbeeld $7/4 = 1,75$. Als je in Java echter geen gebruik maakt van casting dan is het resultaat van een deling van twee `int`s altijd automatisch een `int`, en geen `double`. Bijvoorbeeld:

```
int getal1 = 7;
int getal2 = 4;

System.out.println( getal1 / getal2 );
```

zal de waarde '1' afdrukken. Waarom? Omdat `getal1` en `getal2` `int`s zijn wordt de deling voor integers gebruikt die zelf weer een integer oplevert. In het voorbeeld wordt 7 door 4 gedeeld en dan naar beneden afgerond naar een geheel getal, als het ware wordt alles achter de komma er 'afgehakt'.

Maar, als je een `double` door een `int` deelt, dan is het resultaat ook een `double` omdat in zo'n geval de deling voor doubles gebruikt wordt die wel de cijfers achter de komma bewaart[2]. Als je nu twee gehele getallen wilt delen met als resultaat een kommagetal (`double`), dan moet je de teller (de noemer werkt ook) eerst casten naar een `double`. Casten doe je als volgt: zet het type tussen haakjes voor de variabele die gecast moet worden. Dus, `System.out.println( (double)getal1 / getal2 );` zal het kommagetal $1,75$ afdrukken naar de console.

```
int getal1 = 7;
int getal2 = 4;

System.out.println( (double) getal1 / getal2 );
```

## Voorbeeld

Om de waarde van `int nrEggsFound` te casten naar een `double` (een kommagetal), gebruik je: `(double )nrEggsFound`.

[2]Natuurlijk niet alle cijfers, want een `double` bestaat net als een `int` uit een maximum aantal cijfers. Dit heet de *precisie* van een type.

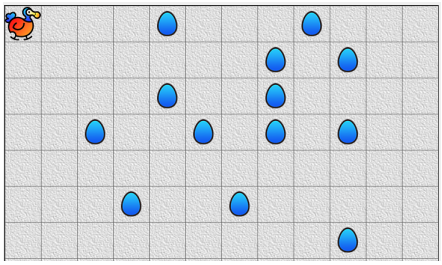
# Uitdagingen

**Lees eerst Theorie 5.1: SommeerPlan.**

## Opgave 5.1: Eieren in de wereld tellen

Nu je wat simpele, maar belangrijke methodes hebt geschreven, is het tijd voor een meer uitdagende taak. Hiervoor moet je jouw kennis uit verschillende opdrachten combineren.

Taak: *Schrijf een methode voor Mimi waarmee zij het aantal eieren in de wereld telt en dit oplevert.*



Figuur 4: Beginsituatie: Mimi begint linksbovenin

Om het programmeren en in het bijzonder het testen makkelijker te maken, verdelen we het probleem in kleinere deelproblemen die we ieder afzonderlijk kunnen aanpakken. Op dat moment houd je je alléén bezig met dat specifieke probleem. Daarna combineer je de deeloplossingen tot één geheel. Volg hierbij deze stappen:

a) Bedenk welke deelproblemen er zijn:

- Tel het aantal eieren in een rij
- Verhoog de huidige som met het aantal eieren dat in de huidige rij gevonden is
- Ga naar de volgende rij
- Loop terug naar de beginsituatie
- ...

b) Schets een hoog-niveau stroomdiagram voor de totaaloplossing. Bepaal de begin- en eindsituaties. Zijn ze gelijk aan elkaar? Dat zouden ze wel moeten zijn.

c) Schrijf en test code voor elk deelprobleem. Tips:

- **Hergebruik:** Probeer altijd methodes te gebruiken die je in een vorige opdracht al geschreven hebt. Je hoeft dan alléén de methode aan te roepen. Dat bespaart je veel tijd. Je hebt die methode namelijk al getest (en alle fouten eruit gehaald)!
- **Generiek:** Zorg dat jouw oplossing generiek is (bekijk Theorie 4.5). Zo zal jouw code ook in verschillende werelden en beginsituaties werken.
- **Print:** Voor het testen kun je op verschillende momenten waarden naar de console afdrukken, zoals het aantal eieren in een rij of het huidige totaal.

d) Combineer de submethodes tot één totaaloplossing. Test dit.

Je hebt nu een oplossing gevonden voor een groter probleem door dit eerst op te delen in kleinere, behapbare brokken (decompositie). Elk deelprobleem heb je apart ontworpen, geïmplementeerd en getest (modularisatie). De deeloplossingen heb je gecombineerd tot een totaaloplossing.

**Lees eerst Theorie 5.2: Minimum/Maximum plan.**

## Opgave 5.2: Zoek de rij met het meeste eieren

Taak: *Leer Mimi hoe ze de rij kan vinden met de meeste eieren.*
De methode levert het rijnummer op met de meeste eieren. In de console druk je het rijnummer en het aantal eieren in die rij af. Mimi moet natuurlijk ook terug naar haar beginsituatie.

a) Bedenk een geschikt algoritme. Bepaal welke variabelen je nodig hebt om alles bij te houden.

b) Teken een hoog-niveau stroomdiagram.

Tips:

- Mimi begint en eindigt in de linkerbovenhoek.
- Gebruik methodes opnieuw die je al eerder geschreven hebt.
- Maak jouw oplossing generiek (bekijk Theorie 4.5).
- Bepaal een geschikte oplossing in het geval er meer dan één rij is met hetzelfde maximum. Beschrijf je keuze in Javadoc commentaar.

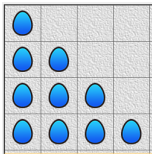
c) Test elke submethode apart. Test daarna jouw oplossing als geheel.

d) Neem een moment om even terug te kijken naar wat je gedaan hebt en hoe dat gegaan is. Heb je veel fouten gemaakt tijdens het programmeren? Wat voor fouten waren dat? Wat had je beter anders kunnen doen?

Je hebt nu geleerd het *Min/Max Plan* zelf toe te passen.

## Opgave 5.3: Monument van eieren

Taak: *Leer Mimi het volgende monument van eieren te leggen.*

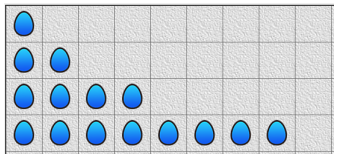


Figuur 5: Monument van eieren

- Vul de wereld zo ver mogelijk aan met dit patroon.
- Bedenk een generieke oplossing. Je mag geen 'harde' getallen gebruiken zoals '3' of '4'.

## Opgave 5.4: Stevig monument van eieren

Taak: *Leer Mimi het volgende monument van eieren te leggen.*

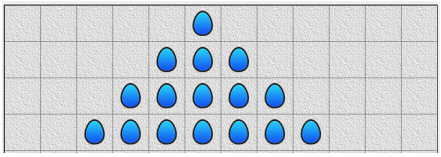


Figuur 6: Stevig monument van eieren

- Vul de wereld zo ver mogelijk aan met dit patroon.
- Bedenk een generieke oplossing. Je mag geen 'harde' getallen gebruiken zoals '3' of '4'.

## Opgave 5.5: Piramide van eieren

Taak: *Leer Mimi om een piramide van eieren te bouwen.*



Figuur 7: Piramide van eieren

- Vul de wereld zo ver mogelijk aan met dit patroon.
- Bedenk een generieke oplossing. Je mag geen 'harde' getallen gebruiken zoals '3' of '4'.

**Lees eerst Theorie 5.3: Type casting.**

## Opgave 5.6: Gemiddeld aantal eieren per rij

Mimi weet inmiddels hoe ze eieren moet tellen. Ze kan ook al bepalen hoeveel eieren er in een bepaalde rij zitten. Maar hoeveel eieren liggen er gemiddeld in elke rij? Deze taak lijkt veel op de vorige. Houd er rekening mee dat het gemiddelde waarschijnlijk een kommagetal is (en dus geen `int`). Mogelijk dat je *type-casting* moeten gebruiken.

Taak: *"Mimi berekent hoeveel eieren er gemiddeld in elke rij liggen en levert dit op."*
Gebruik de volgende stappen en het stroomdiagram hieronder om een begin te maken:

a) Ga ervan uit dat Mimi in de linkerbovenhoek begint, kijkend naar het oosten.

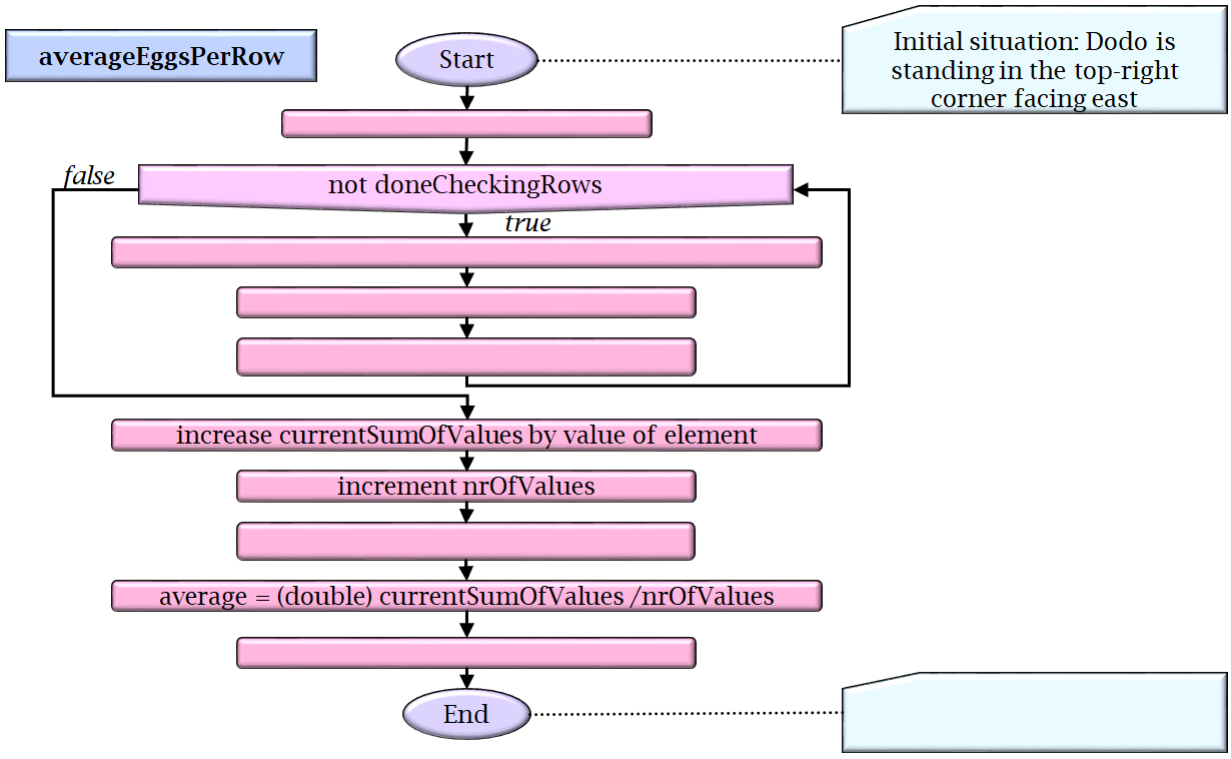
b) Schrijf op hoe je het gemiddelde uitrekent.

c) Bepaal welke variabelen je nodig hebt om het gemiddeld aantal eieren uit te kunnen rekenen.

d) Bij het berekenen van het gemiddelde gebruik je type-casting. Hiermee verander je het totaal aantal eieren (`int`) in een `double` kommagetal.

e) Test jouw methode. Test deze ook met eieren die tegen een wereldgrens aan liggen. Gaat het goed bij de eerste en laatste rij en de eerste en laatste kolom?

f) Controleer dat de begin-en eindsituaties gelijk zijn aan elkaar.




## Opgave 5.7: Pariteitsbit

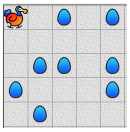
Bekijk het volgende filmpje: `https://www.youtube.com/embed/OXz64qCjZ6k?rel=0`. Dit filmpje laat het pariteitstrucje zien. Eigenlijk is het geen trucje, maar wetenschap. Met hetzelfde techniek kunnen computers fouten in data opsporen en herstellen.

Als berichten van de ene naar de andere computer worden verstuurd kunnen foutjes optreden. Soms gaat data verloren. Soms wordt data beschadigd: een 1 verandert in een 0 of andersom. Het *Pariteitsbit*-algoritme is een techniek waarmee fouten gedetecteerd kunnen worden. Door het toevoegen van extra bits kun je in sommige gevallen bepalen of er een bitje verloren is gegaan of beschadigd is, en kun je deze fout dan ook nog herstellen. Jouw taak is om Mimi te leren de Pariteitsbit-algoritme uit te voeren.

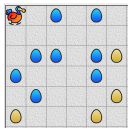
### Pariteitsbit algoritme uitgelegd

We willen er zeker van zijn dat elke kolom of rij een **even** aantal eieren (of bits) heeft. Zo kun je later bepalen of de wereld wel of niet beschadigd is. De wereld is beschadigd als er een rij of kolom is met een oneven aantal eieren (of bits). Merk op dat je in het algemeen hiermee niet kunt detecteren of er meerdere bits beschadigd zijn.

Om de wereld voor te bereiden voegen we een gouden ei toe aan iedere rij met een oneven aantal blauwe eieren. We doen hetzelfde voor de kolommen.



Figuur 8: Oorspronkelijke wereld (vóór beschadiging)



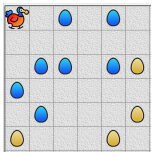
Figuur 9: Gouden pariteits-eieren toegevoegd aan oorspronkelijke wereld (vóór beschadiging)

**Fouten opsporen**

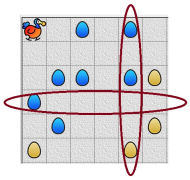
Als we controleren op fouten, bepalen we of een rij een even of oneven aantal eieren heeft. We tellen hiervoor zowel gouden eieren als blauwe eieren.

Dit zijn de mogelijkheden:

- **Ontbrekend ei**: We kunnen bepalen waar een ei verloren is gegaan door te controleren welke rij en kolom een oneven aantal eieren bevatten. In dit geval de derde rij en vierde kolom. Het ei is dus gestolen van coördinaten (4,3).
- **Extra ei**: Door een extra ei 'illegaal' aan de wereld toe te voegen raakt de wereld ook 'beschadigd'. Het opsporen gaat op hetzelfde manier als bij een ontbrekende ei.
- **Geen fout**: Als alle rijen en kolommen een even aantal eieren bevatten dan is er geen fout (of ons algoritme is niet slim genoeg om deze te ontdekken).



Figuur 10: Beschadigde wereld (ei gestolen)



Figuur 11: Beschadigde wereld: locatie van de fout is gevonden

**Fout herstellen**

Bij het ontbrekende ei kun je de fout herstellen door een nieuw ei te plaatsen op coördinaat (4,3). In andere gevallen moet je wellicht een ei weghalen.

## Fouten herstellen in Mimi's wereld

Taak: *"Leer Mimi om het Pariteitsbit algoritme uit te voeren."*
Ze moet controleren of de wereld 'beschadigd' is en deze beschadiging herstellen. De wereld kan beschadigd raken doordat een ei gestolen wordt of doordat er een ei 'illegaal' is toegevoegd.

a) Teken een hoog-niveau stroomdiagram waarmee je het algoritme beschrijft. Tip: de volgende submethodes kunnen handig zijn:

- `countEggsInRow` (deze heb je al)
- `getIncorrectRowNr` (kijkend naar het zuiden, kan deze ook gebruikt worden om een incorrecte bit in een kolom te vinden)
- `gotoIncorrectBit`
- `fixIncorrectBit`
- `goToLocation` (deze heb je al)
- ...

b) Schrijf en test elke submethode apart.

c) Test jouw methode als geheel.

d) Evalueer jouw oplossing. Werkt deze ook goed als:

- de wereld helemaal niet beschadigd is?
- er een extra ei 'illegaal' is toegevoegd aan de wereld?
- Mimi niet in de linkerbovenhoek begint?
- Mimi in de rechterbovenhoek begint, kijkend naar beneden?

Had je een slimmer algoritme kunnen gebruiken?

e) Verbeter jouw oplossing.

## Opgave 5.8: Generiek pariteitsbit-algoritme

In deze taak bekijken we generieke algoritmes. In taak 5.7 heb je het pariteitsbit-algoritme geïmplementeerd. Maar, Mimi heeft haar hoofd gestoten en ineens geen gevoel meer voor richting. Ze weet bijvoorbeeld niet waar het *noorden* is. Dat betekent dat je geen kompas-gerelateerde methodes kunt gebruiken zoals `facingEast( )` of `goToLocation`.

Taak: *Leer Mimi om het pariteitsbit-algoritme generiek uit te voeren, zonder gebruik te maken van richting.*

a) Bedenk een geschikte oplossing.

b) Pas jouw oplossing uit de vorige taak aan.

c) Leg uit wat de voordelen zijn van jouw nieuwe algoritme. Waarom is dit beter dan een algoritme waarbij Mimi een kompas nodig heeft?

d) Kun je de methode voor het tellen van eieren in een rij opnieuw gebruiken voor het tellen van eieren in een kolom? Implementeer deze verbetering.

e) Nu dat je dit verbeterde, meer generieke algoritme hebt geïmplementeerd, zie je nog andere verbeteringen die je kan doorvoeren? Implementeer deze.

f) In welke gevallen (beginsituaties) zal jouw oplossing niet werken?

# Reflectie

In deze opdracht heb je geleerd om plannen toe te passen. Ook heb je geoefend met het beschrijven van een algoritme op hoog niveau zodat duidelijk is welke deeltaken je los van elkaar kan aanpakken.

Als je ergens echt goed in wilt worden, is een van de meest belangrijke stappen om even terug te blikken op wat je deed en hoe dat ging:

## Resultaat

Ik weet dat mijn oplossing werkt omdat . . .
Ik ben trots op mijn oplossing omdat . . .
Ik kan mijn oplossing verbeteren door . . .

## Aanpak

Mijn aanpak was goed omdat . . .
Wat ik volgende keer beter anders kan doen is . . .

Geef met een smiley aan hoe het ging.

| | |
|---|---|
|  | Ik kan het |
|  | Het is me een beetje gelukt maar ik begreep het niet helemaal |
|  | Ik snapte er helemaal niks van |

| | |
|---|---|
|  | Ik kan plannen toepassen om uit een aantal waarden de kleinste/grootste getal te vinden, en de som en gemiddelde te berekenen. |
|  | Ik kan uitleggen wat type-casting is. |
|  | Ik kan een algoritme omschrijven met een hoog-niveau stroomdiagram. |
|  | Ik kan een groot probleem opdelen in deelproblemen en die afzonderlijk oplossen. |
|  | Ik kan een generiek algoritme ontwerpen en implementeren. |

# Opslaan en inleveren

Sla je werk op. Je hebt het nodig voor de volgende opdrachten.

## Opslaan

Selecteer ’Scenario’ in het Greenfoot menu, en dan ’Save’. Alle bestanden die bij het scenario horen zitten nu in één map.

## Inleveren

Lever het volgende in:

- Naam: van jou (en je partner);
- Jouw code: Het `MyDodo.jav` java bestand;
- Stroomdiagrammen:
  - Als je ze op papier hebt gemaakt: plak (foto’s van) jouw stroomdiagrammen in een (Word) bestand.
  - Als je software gebruikt hebt: sla het bestand op als .pdf of .jpg.