Algoritmisch Denken en Gestructureerd Programmeren

Martin Bruggink en Renske Smetsers-Weeda

I&I, 8 november 2018

http://course.cs.ru.nl/greenfoot/

www.informaticaunplugged.nl

Even voorstellen…

- Renske Smetsers-Weeda is docent informatica, promovendus aan de Radboud Universiteit Nijmegen en (mede)-ontwikkelaar van het lesmateriaal.
- Martin Bruggink is vakdidacticus informatica bij de TU Delft.

- Kennismaken met de kunst van het oplossen van computationele problemen
- In een reeks van opdrachten maken we Dodo steeds slimmer
- Leuk:
  - Puzzelen
  - Creatief
  - Zoeken naar (creatieve) oplossingen
- Serieus:
  - Op een gestructureerde wijze van algoritme naar een werkende programma
  - Resultaat: leesbaar en uitbreidbare code

Nieuw examenprogramma (v.a. 1 augustus 2019)

- Lesmateriaal heeft raakvlakken met domeinen:
  - B (Grondslagen)
  - D (Programmeren)
  - en daarbij ook domein A (Vaardigheden).

- In de volgende sheets:
  - Welke onderdelen van domein A, B en D worden behandeld?
- Daarna: hoe komen deze onderdelen in het lesmateriaal aan bod?

Domein B: Grondslagen en Domein D: Programmeren

Subdomein B1: Algoritmen

- De kandidaat kan een oplossingsrichting voor een probleem uitwerken tot een algoritme, daarbij standaardalgoritmen herkennen en gebruiken, en de correctheid en efficiëntie van digitale artefacten onderzoeken via de achterliggende algoritmen.

Subdomein D1: Ontwikkelen

- De kandidaat kan, voor een gegeven doelstelling, programmacomponenten ontwikkelen in een imperatieve programmeertaal, daarbij programmeertaalconstructies gebruiken die abstractie ondersteunen, en programmacomponenten zodanig structureren dat ze door anderen gemakkelijk te begrijpen en te evalueren zijn.

Subdomein D2: Inspecteren en aanpassen

- De kandidaat kan structuur en werking van gegeven programmacomponenten uitleggen, en zulke programmacomponenten aanpassen op basis van evaluatie of veranderde eisen.

B1 Algoritmen: Voorbeeldspecificaties

De kandidaat kan:

- een gegeven oplossingsrichting voor een probleem weergeven als een algoritme. Hierbij wordt verwacht dat kandidaten een algoritme op een gestructureerde wijze kunnen weergeven met bijvoorbeeld een flowchart of in pseudocode.
- Het algoritme is opgebouwd uit de basisbouwstenen opeenvolging, keuze en herhaling.
  - *vwo: Het algoritme kan recursie bevatten*
- een gegeven digitaal artefact modelleren met behulp van een algoritme.
- het gedrag van een programma onderzoeken via het onderliggende algoritme, en zo problemen (bijvoorbeeld fouten of inefficiëntie) herleiden tot aspecten van dat algoritme.
- een aantal standaardalgoritmen herkennen en gebruiken.
  - Kandidaten kennen standaardalgoritmen voor elementaire numerieke operaties zoals minimum en maximum bepalen, sommeren, en ten minste twee andere doelen zoals zoeken, sorteren, datacompressie en graafalgoritmen (bijvoorbeeld routebepaling).
  - Kandidaten kennen bij tenminste één doel ten minste twee standaardalgoritmen.
- de correctheid van een gegeven algoritme onderzoeken, en algoritmen (waaronder standaardalgoritmen), vergelijken met betrekking tot efficiëntie.

D1 Ontwikkelen: Voorbeeldspecificaties

De kandidaat kan:

- werkende programmacomponenten ontwikkelen in een imperatieve programmeertaal naar keuze en daarbij,
    - gebruik maken van aanduidingen voor dataobjecten zoals variabelen en constanten; toewijzing van waarden aan variabelen;
    - gebruik maken van de algoritmische bouwstenen opeenvolging, keuze en herhaling implementeren met behulp van controlestructuren in de gekozen programmeertaal;
    - datatypen implementeren aan de hand van elementaire datatypen en taalconstructies voor datastructuren in de gekozen programmeertaal;
    - gebruik maken van taalconstructies die abstractie ondersteunen;
    - doelgericht gebruik maken van mechanismen om de leesbaarheid van een programmacomponent te vergroten, zoals kiezen van betekenisvolle namen voor dataobjecten, gebruik van procedures en functies, commentaar, en suggestieve lay-out;
    - debuggen en testen inzetten.
- vanuit een algoritme een werkend programma ontwikkelen in de gekozen programmeertaal.

Bron: Concept Handreiking schoolexamen informatica, Januari 2018

D2 Inspecteren en aanpassen: Voorbeeldspecificaties

De kandidaat kan:

- de structuur en werking van een gegeven programmacomponent uitleggen.
- een gegeven programmacomponent evalueren aan de hand van de eigenschappen correctheid, efficiëntie, en leesbaarheid.
- een bestaande programmacomponent aanpassen
  - als gevolg van een evaluatie van bijvoorbeeld de correctheid, efficiëntie of leesbaarheid van de programmacomponent.
  - als gevolg van een veranderde of uitgebreide doelstelling.

Greenfoot

- programmeeromgeving
- maken van (eenvoudige) grafische 2D-applicaties
- programma’s worden in Java geschreven

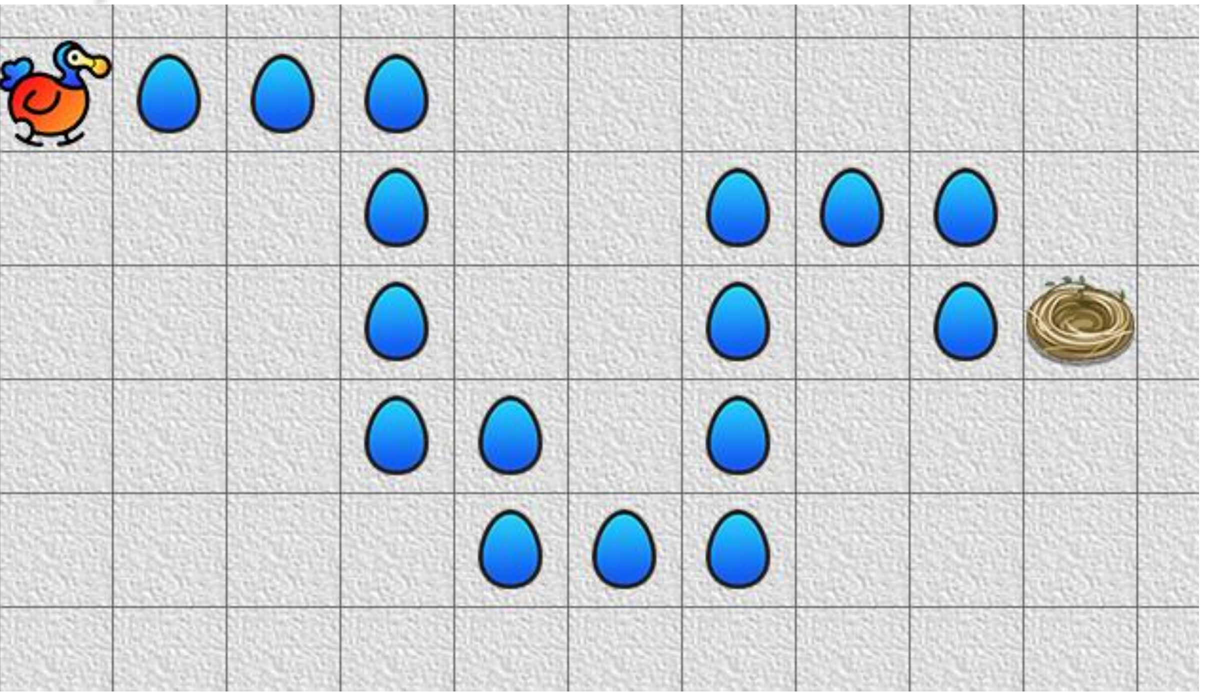
- Windows / Mac / Ubuntu / USB
- Programma kan rechtstreeks vanaf USB draaien, geen software installatie nodig op PCs!

Algoritmische concepten

- Opeenvolgingen
- Condities
- Herhalingen
- Variabelen
- Operatoren
- Methoden en parameters

Combinaties: achter/in elkaar

Bepaal of je klaar bent.
Zo niet, bepaal of ei voor je ligt..
Zo wel, neem een stap..
Zo niet, draai een kwart slag en herhaal….



Volg het pad naar het nest.

Domein A: Computational Thinking (Algoritmisch denken)

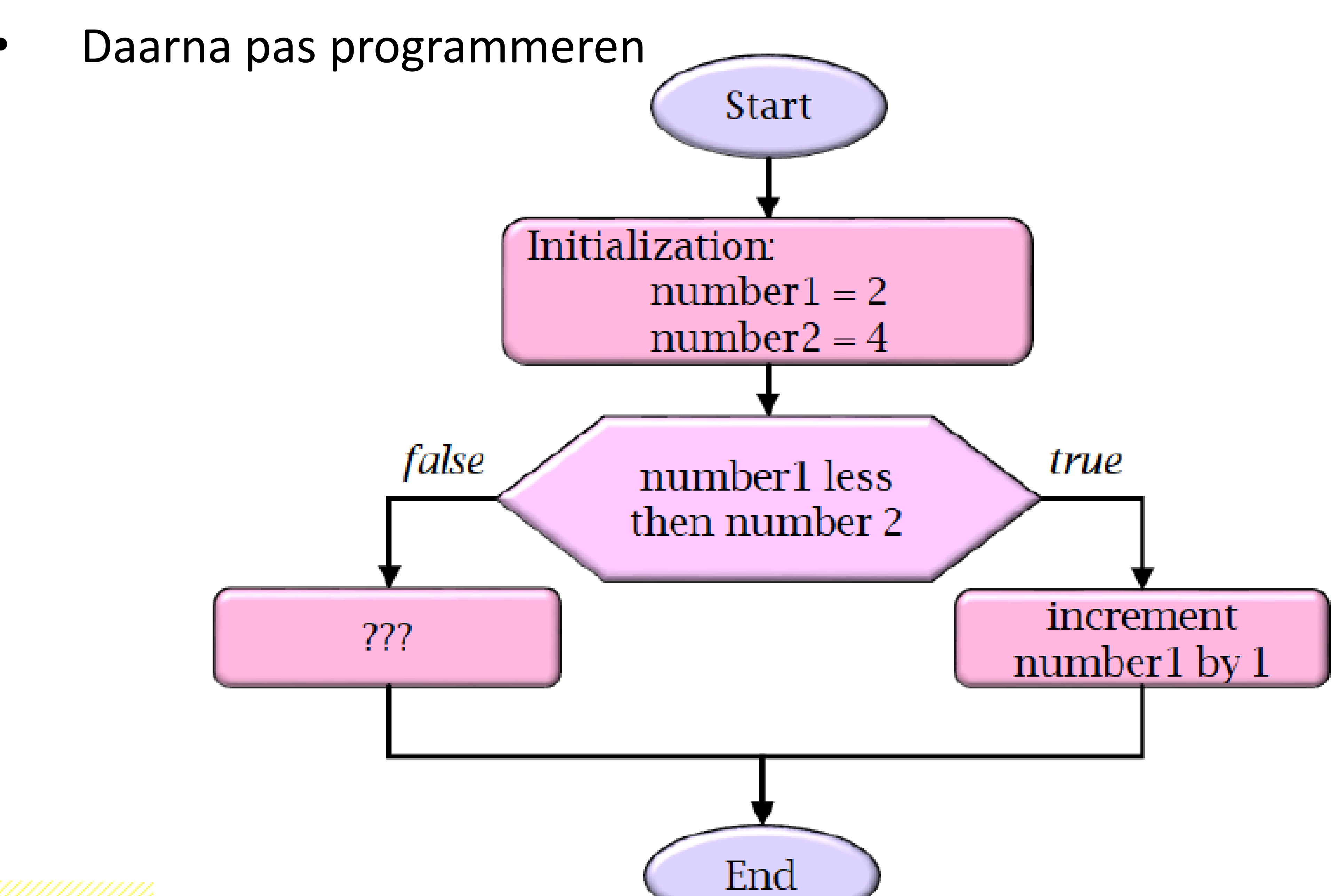
- **Gestructureerd en iteratief werken**: deelproblemen identificeren (**decompositie**) en afzonderlijke ontwerpen (flowchart), implementeren (**modularisatie**) en testen (en debuggen)
- **Abstractie toepassen** om oplossingen te kunnen herbruiken
- **Generalizatie:** algemene oplossingen maken die voor meer doeleinden inzetbaar zijn
- **Analyseren:** redeneren over kwaliteit van een oplossing (bv. leesbaarheid, correctheid en efficiëntie)
- **Testen / debuggen:** zorgen dat dingen werken, fouten opsporen
- **Reflecteren / evalueren**: kwaliteit beoordelen van gekozen oplossing en ontwikkelproces

Didactische aanpak

- Eerst denken:
  - Weg van de PC
  - Analyseren: Wat is het probleem?
  - Brainstormen: Hoe kunnen we dit oplossen?
  - Ontwerp: Algoritme als flowchart omschrijven (zonder bezig te zijn met implementatie details)
- Dan doen…
  - Bezig zijn met syntax en andere implementatiedetails
  - Debuggen en testen
- Daarna evalueren en reflecteren:
  - Hoe ging het? Kan het beter? Wat hebben we geleerd?

Expliciete koppeling algoritme (flowchart) en implementatie (code)

- Eerst denken:
  - Weg van de PC een flowchart tekenen.
- Daarna pas programmeren




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

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

Tracing van code

- Redeneren over wat code doet
- Debugging strategie

Wat doet deze code?
Geef een geschikte naam voor deze methode.

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

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

Instructie	getal1	getal2
initialisatie	6	3
getal1 = getal2;		

Tracing van code

- Redeneren over wat code doet
- Debugging strategie

verdubbelenVan2Getallen
… maar is geen 'slimme' algoritme

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

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

statement	number1	number2
initialization	6	3
number1 = number2;	3	3
number2 = number1;	3	3
if(number1 == number2)	3	3
number1 = number1 + number2;	6	3

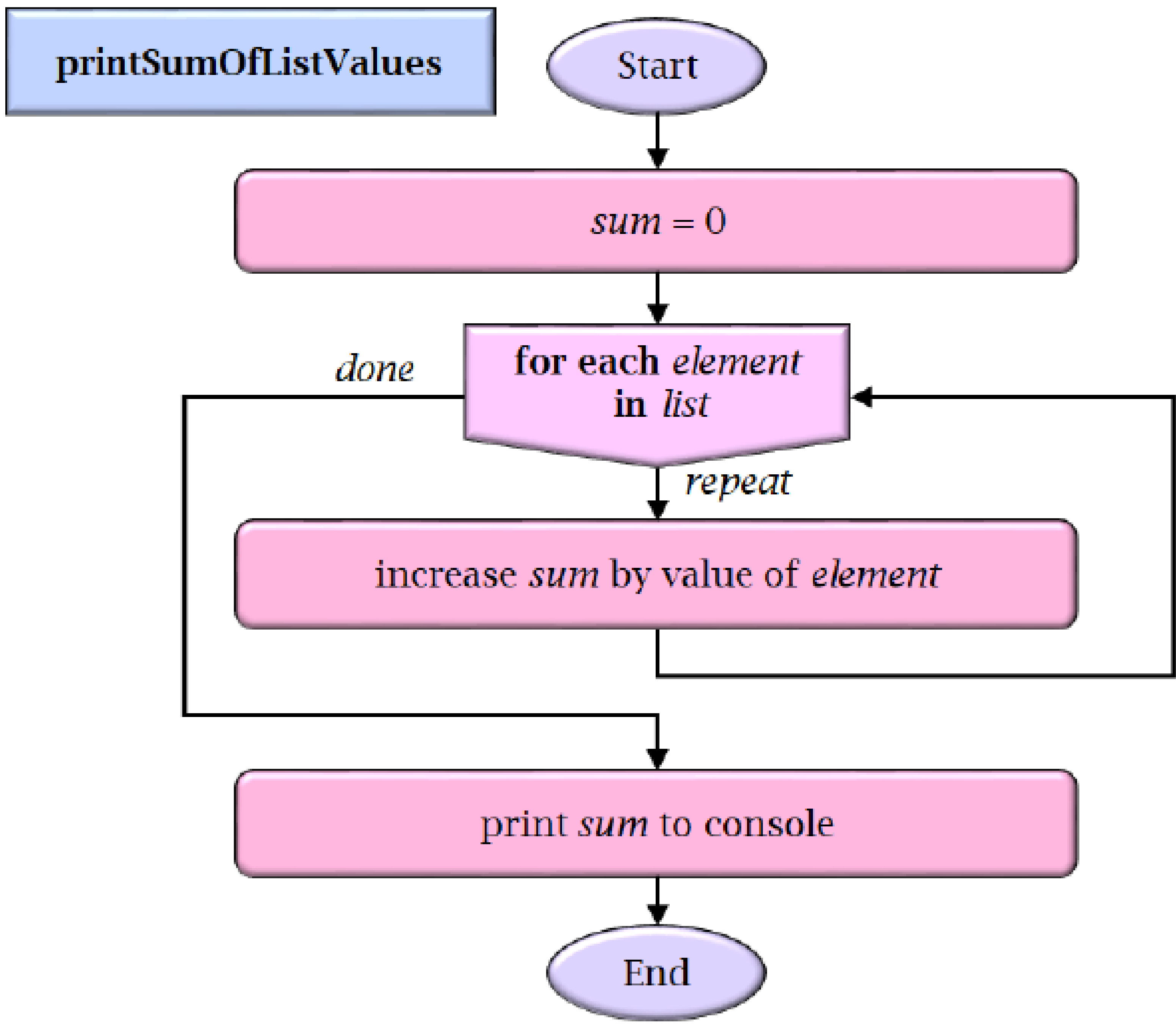
Unplugged: Tellen met kaarten

- Tellen met kaarten

Standaardalgoritmen die aan bod komen:

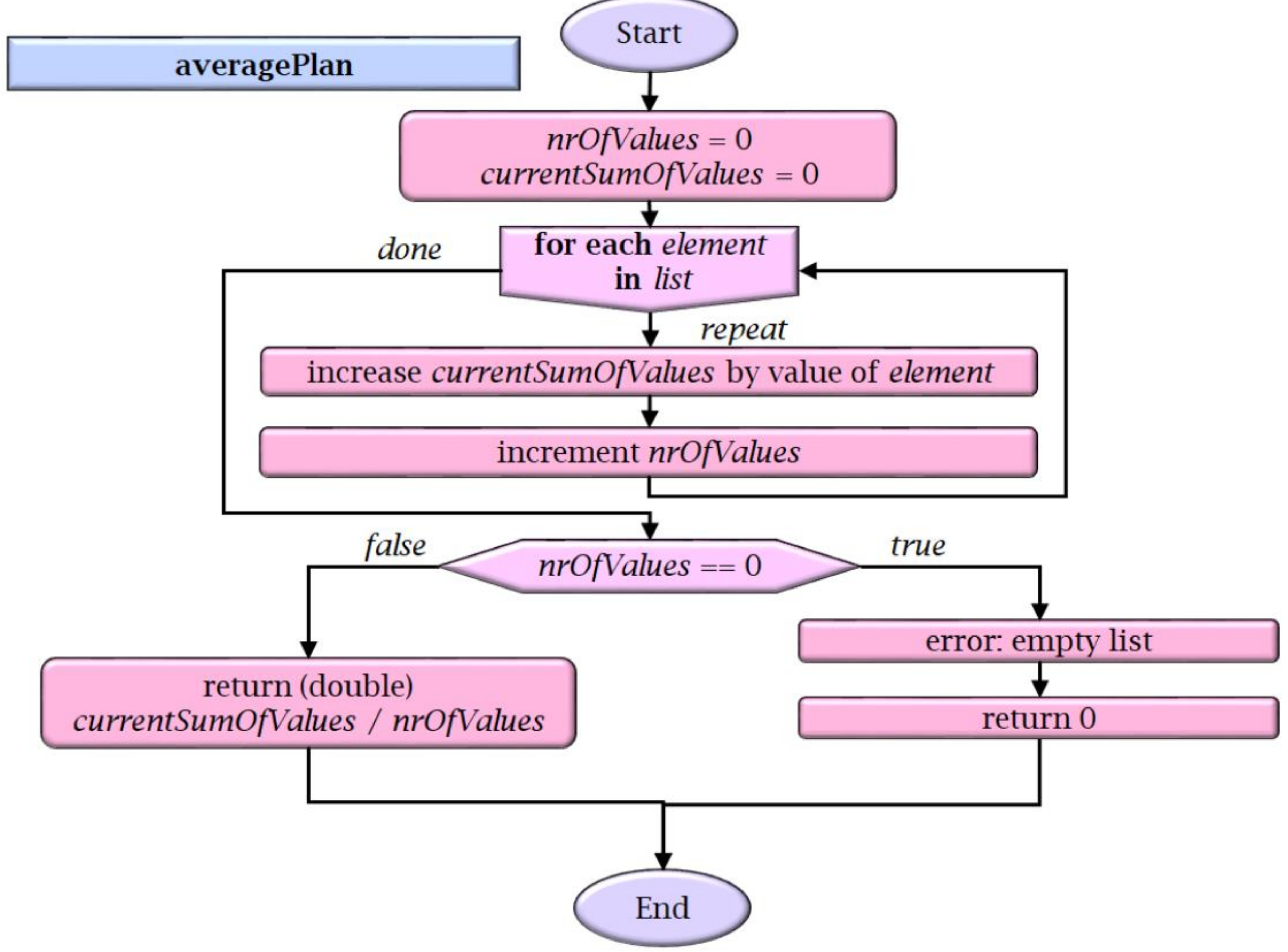
- Tellen, Sommeren, Gemiddelde bepalen
- Minimum en Maximum bepalen
- Sorteren
- Graafalgoritme
  - Dichtstbijzijnde-buur algoritme
  - Eindopdracht: Bedenk en implementeer een betere algoritme

Standaardalgoritme: sommeren van waarden in een lijst




```
/**
 *  Adds all the values in a given list of integers and prints it to the console
 */
public void printSumOfNumbersList( List<Integer> listOfNumbers ){
    int sum = 0;
    for( int numberInList:listOfNumbers ){
        sum += numberInList;
    }
    System.out.println("Sum of all values in list: " + sum);
}
```

Standaardalgoritme: gemiddelde van de waarden in een lijst berekenen



Figuur 4: Stroomdiagram: het gemiddelde berekenen van waarden in een lijst

Unplugged: Sorteren met kaarten

- Sorteren met kaarten

Unplugged: Sorteren met kaarten, retro

Hoe heb je de werkvorm ervaren?

Ga in gesprek met de leerlingen over hun resultaten:

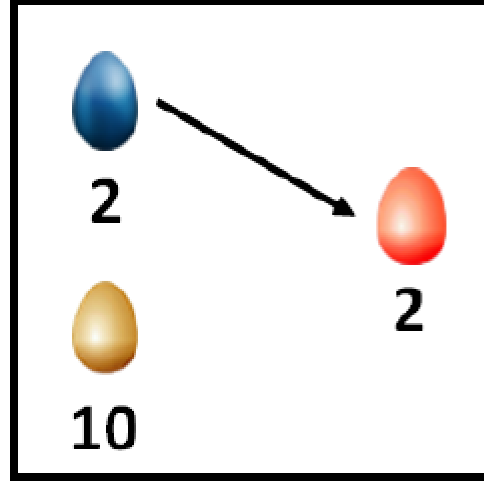
Werken de gevonden oplossingen altijd?

Hoeveel vergelijkingen heb je nodig?

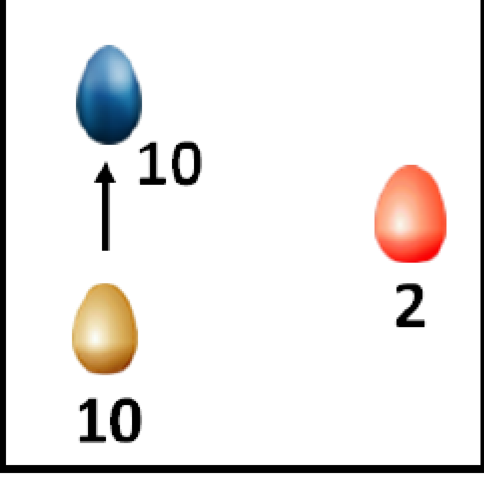
Waren de beschrijvingen van de algoritmes begrijpelijk? Waarom wel/niet?

Sorteren in het lesmateriaal

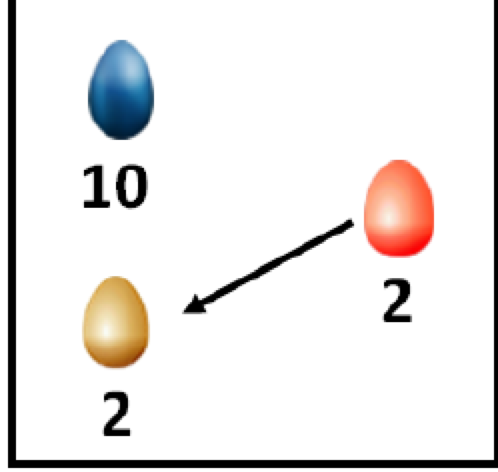
1) Verwisselplan toepassen met getallen en met strings



redEgg = blueEgg;



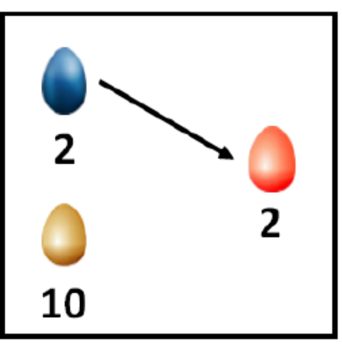
blueEgg = goldenEgg;



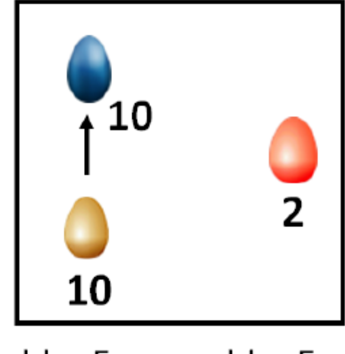
goldenEgg = redEgg;

Sorteren in het lesmateriaal

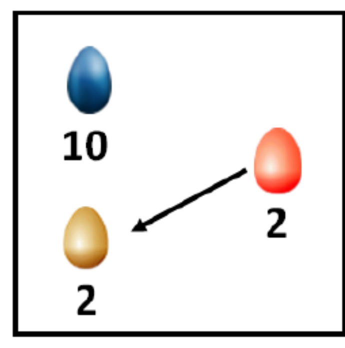
1) Verwisselplan toepassen met getallen en met strings



redEgg = blueEgg;

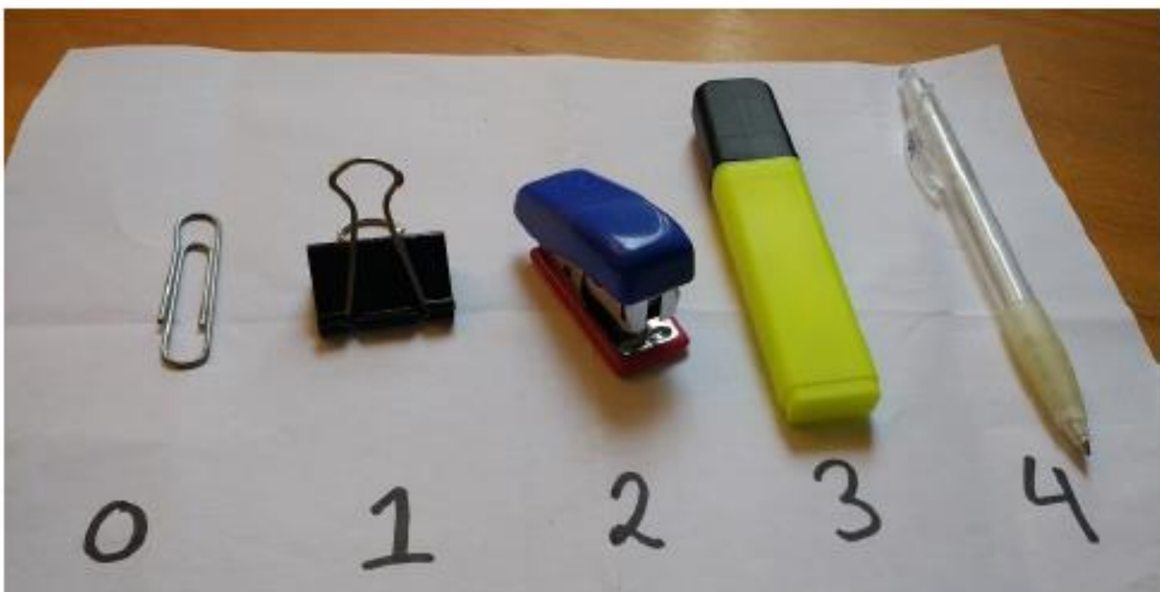


blueEgg = goldenEgg;



goldenEgg = redEgg;

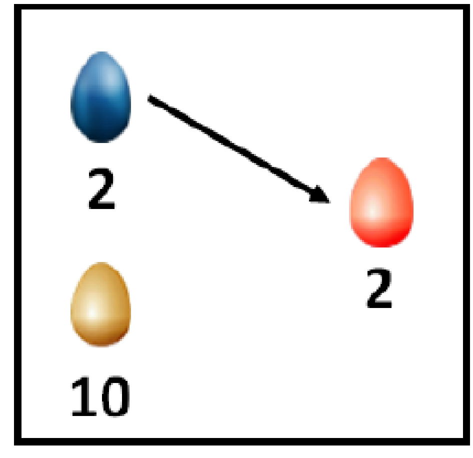
2) Lijst omkeren (hergebruik van eigen verwisselplan code)

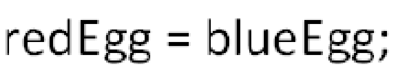


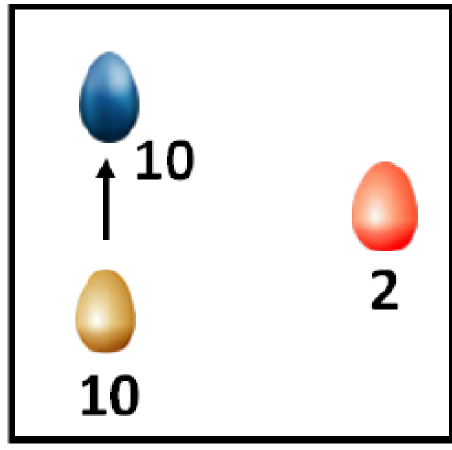
Plaats 5 dingen voor je, van kleinst naar grootst. Schrijf bij elk ding de bijbehorende index (zie het plaatje hierboven). Maak gebruik van jouw algoritme om deze te herrangschikken van grootst naar kleinst. Je mag hiervoor alleen objecten verwisselen.

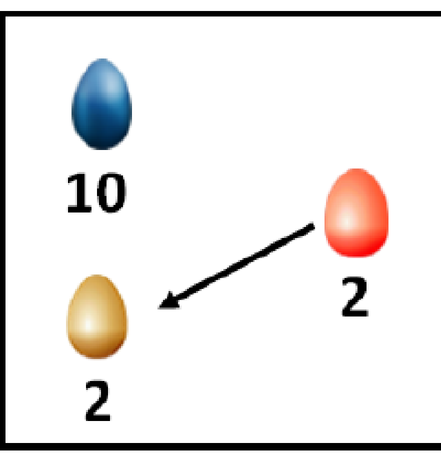
Sorteren in het lesmateriaal

1) Verwisselplan toepassen met getallen en met strings









goldenEgg = redEgg;

2) Lijst omkeren (hergebruik van eigen verwisselplan code)

3) Lijst sorteren met *Selection Sort* (hergebruik van eigen verwisselplan code)

Redeneren over efficiëntie:

a) Als je een lijst van vier getallen hebt, hoeveel vergelijkingen zijn nodig?

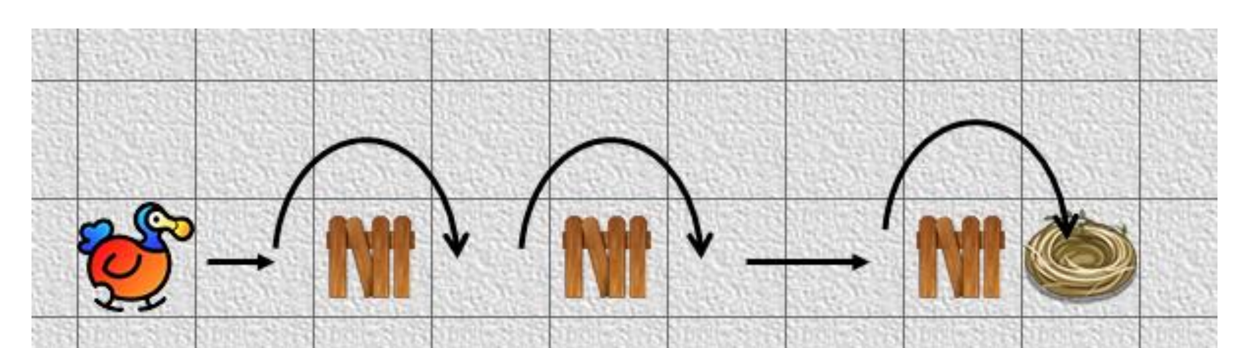
b) Als je een lijst van vijf getallen hebt, hoeveel vergelijkingen zijn nodig?

c) Hoeveel vergelijkingen zijn nodig als de lijst twee keer zo groot is, dus met tien getallen?

d) Hoeveel vergelijkingen zijn (ongeveer) nodig als de lijst tien keer zo groot is?

e) Kun je een algemene formule bedenken om het aantal vergelijkingen te bepalen voor een lijst met n elementen?

Opgave 3.6: Naar het nest lopen en hekken ontwijken

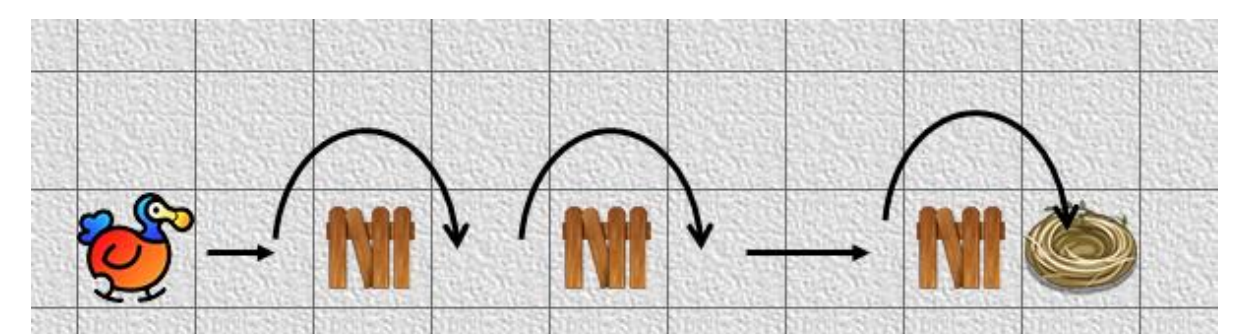


- Mimi zoekt haar nest om daar een ei in te leggen.
- Als ze onderweg een hek tegenkomt, moet ze daar overheen klimmen.
- Als ze haar nest heeft gevonden moet ze er een ei in leggen en stoppen.

- De oplossing moet *generiek* zijn, werkt dus ongeacht:
  - het aantal hekken
  - afstand tot nest.

- Je mag aannemen: Er staan **nooit** twee aaneengesloten stukken hek.

Opgave 3.6: Naar het nest lopen en hekken ontwijken



- Mimi zoekt haar nest om daar een ei in te leggen.
- Als ze onderweg een hek tegenkomt, moet ze daar overheen klimmen.
- Als ze haar nest heeft gevonden moet ze er een ei in leggen en stoppen.

Wat Mimi kan:
Vraag beantwoorden met True/False:
- **onNest**
- **fenceAhead**

Opdrachten uitvoeren:
- **move**
- **turnRight / turnLeft**
- **layEgg**

Eis:
Generieke oplossing**!**

Uitgangspunt:
Geen twee stukken hek aaneengesloten.

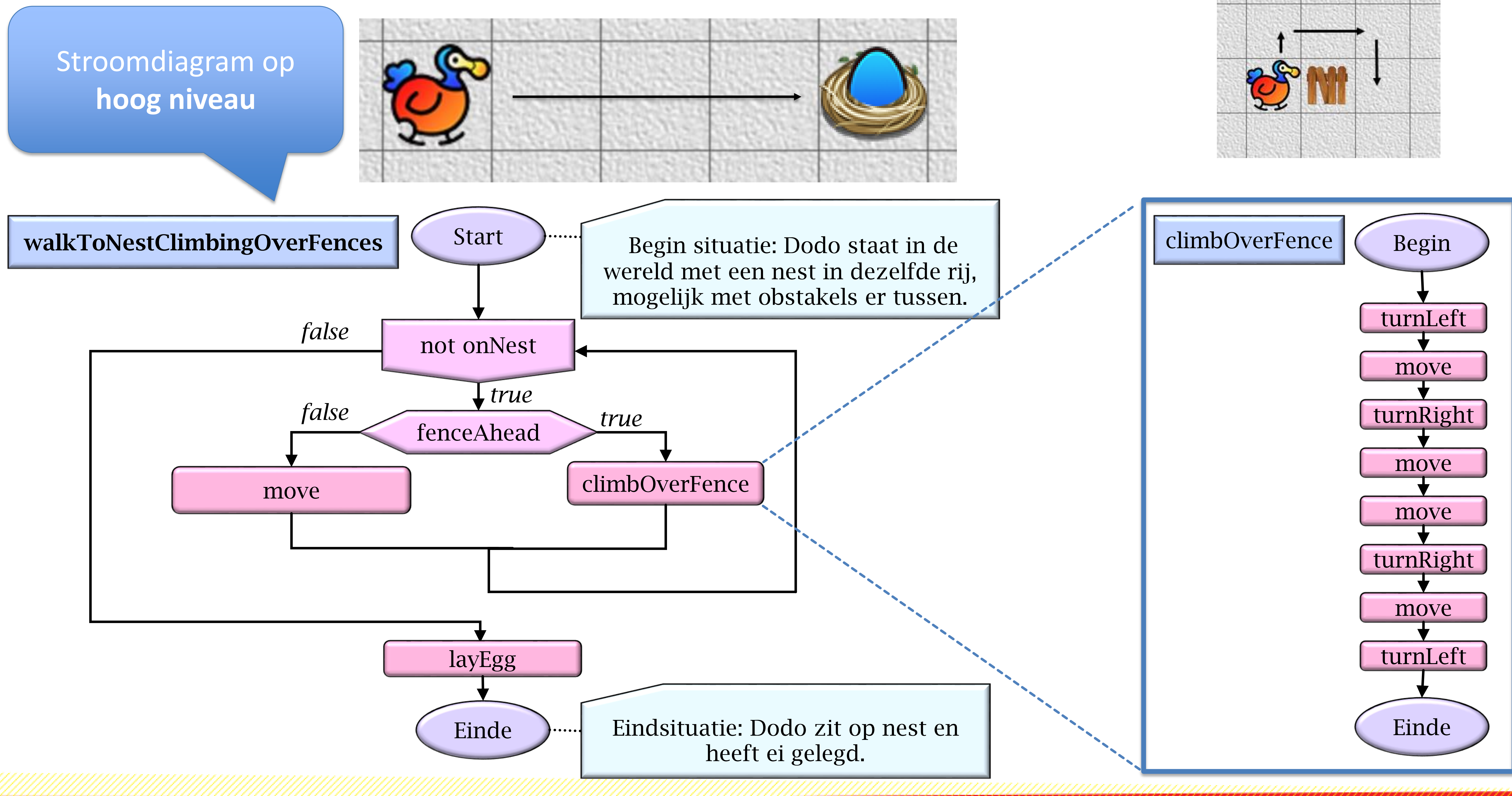
- **Opdracht**: **Schets een stroomdiagram voor de oplossing.**

Bespreking aanpak

- Hoe heb je het aangepakt?
- Is het handig om het probleem op te delen?

Tips:

- Door het stroomdiagram op hoog niveau te tekenen, wordt het probleem meteen in stukjes opgedeeld.
- Stukken zijn dan los van elkaar uit te programmeren en te testen.
- Specifieke eisen (zoals het laatste: leggen van het ei) minder snel vergeten.


Stroomdiagram op
hoog niveau
walkToNestClimbingOverFences
Start
Begin situatie: Dodo staat in de wereld met een nest in dezelfde rij, mogelijk met obstakels er tussen.
not onNest
false
true
fenceAhead
false
true
move
climbOverFence
layEgg
Einde
Eindsituatie: Dodo zit op nest en heeft ei gelegd.
climbOverFence
Begin
turnLeft
move
turnRight
move
move
turnRight
move
turnLeft
Einde

Demo

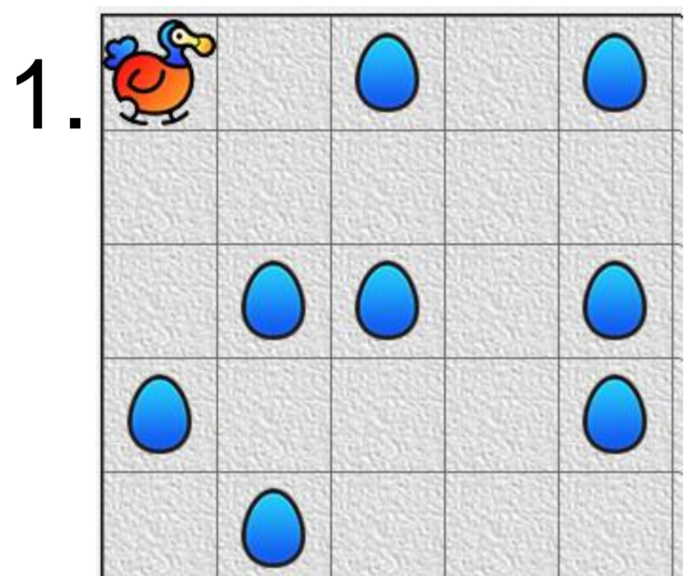
- Uitwerking van stroomdiagram naar code

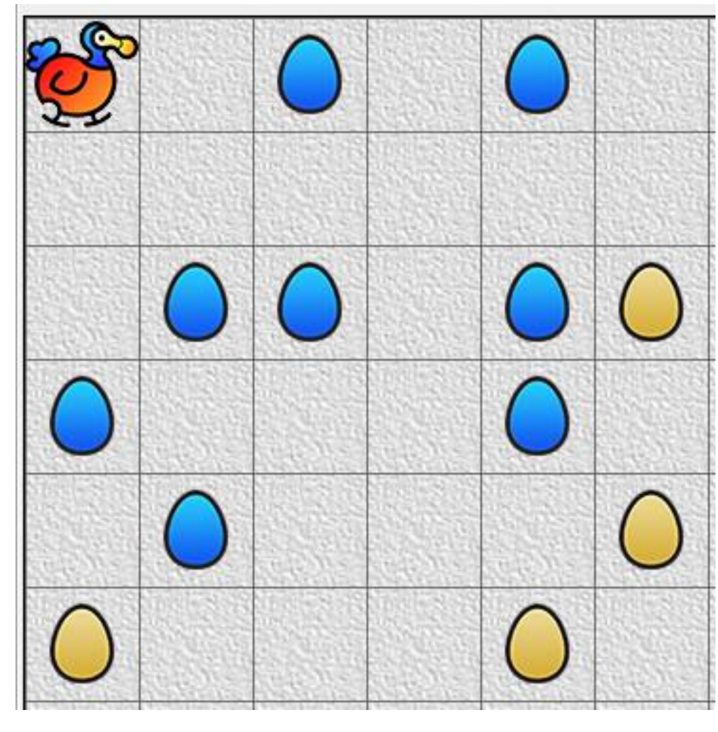
Unplugged: Magische bits

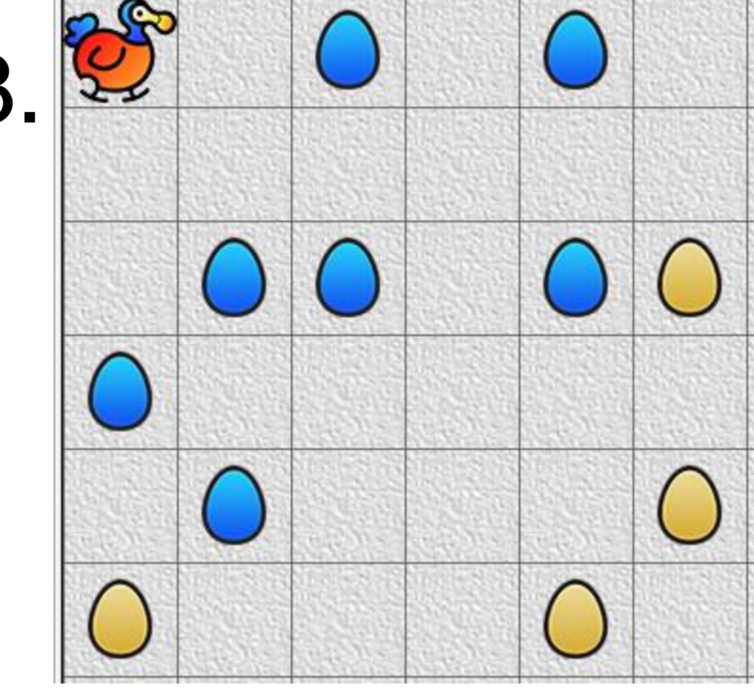
- Magische Bits

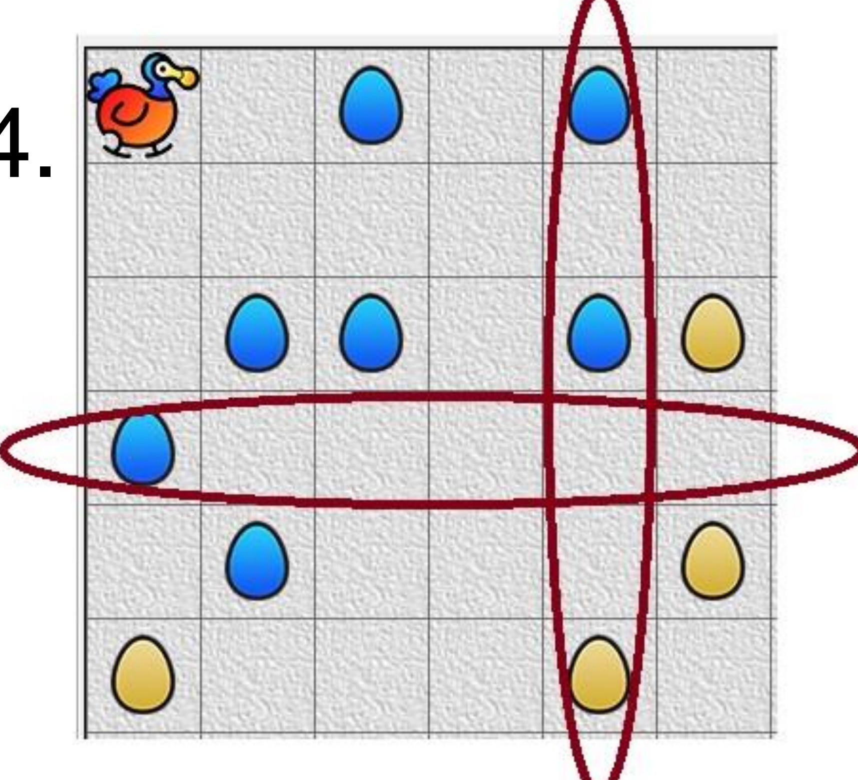
Pariteitsbit algoritme uitgelegd

1. Oorspronkelijke wereld (voor beschadiging)
2. Gouden pariteits-eieren toegevoegd aan oorspronkelijke wereld (voor beschadiging)
3. Beschadigde wereld (ei gestolen)
4. Fout opsporen (ontbrekende ei moet teruggelegd worden)









Reflectie

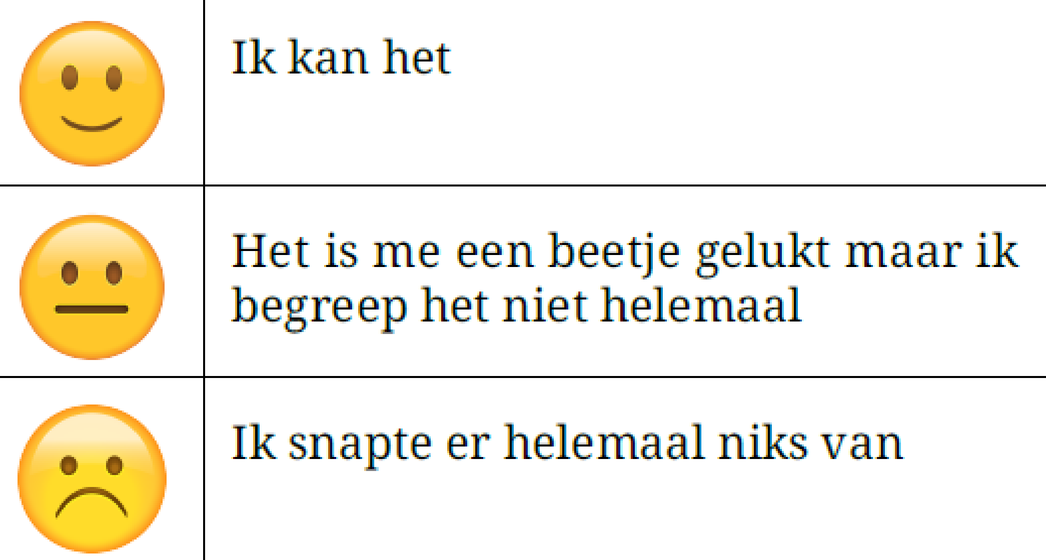
Zelfbeoordeling door leerling.

Docent krijgt snel feedback over hoe het gaat en waar meer aandacht voor nodig is.

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.

Didactische verantwoording

- Heldere **structuur** in de opbouw:
  - in het begin schools
  - steeds meer ruimte voor eigen inbreng/creativiteit
  - verschillende algoritmes leiden tot juiste resultaat
- Juiste **niveau**: stof pas aanbieden als nodig en toegepast wordt.
- **Betekenis** geven: kennis nodig om volgende complexere uitdaging te halen.
- **Zichtbaarheid**: effect van bedachte en geïmplementeerde algoritmes meteen zichtbaar.
- **Motivatie**: afsluitende opdracht is onderlinge wedstrijd tussen leerlingen voor beste (meest optimale) algoritme.

Doelgroep

- Bovenbouw HAVO en VWO
- Geen programmeer voorkennis vereist
- Zowel in het Nederlands als Engels beschikbaar

Differentiatie mogelijkheden

In de diepte:

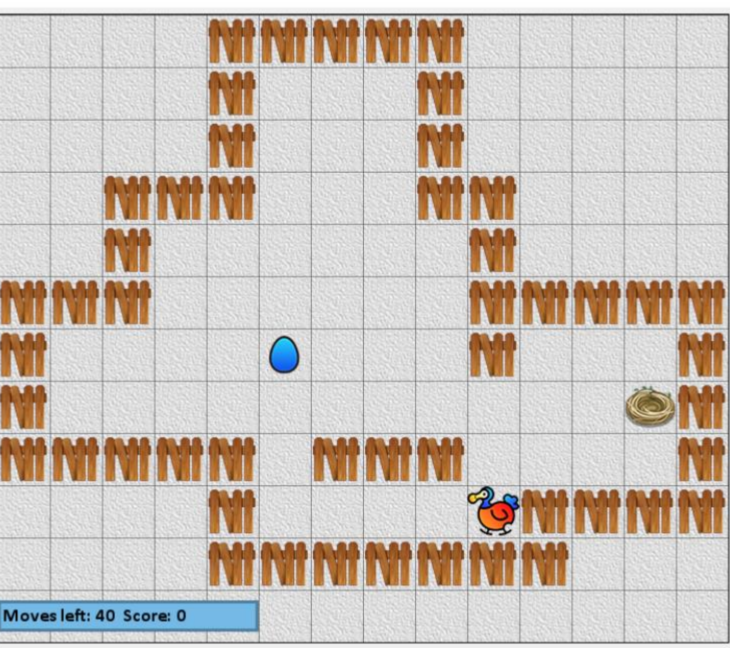
Er zijn drie soorten opgaven:

★	**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 opgaven, ontwikkeld voor leerlingen die 2-ster opdrachten voltooid hebben en klaar zijn voor een grotere en onderzoekende uitdaging.

Leerlingen die 1-ster opgaven overslaan moeten alle 3-ster opgaven maken.

In de breedte:

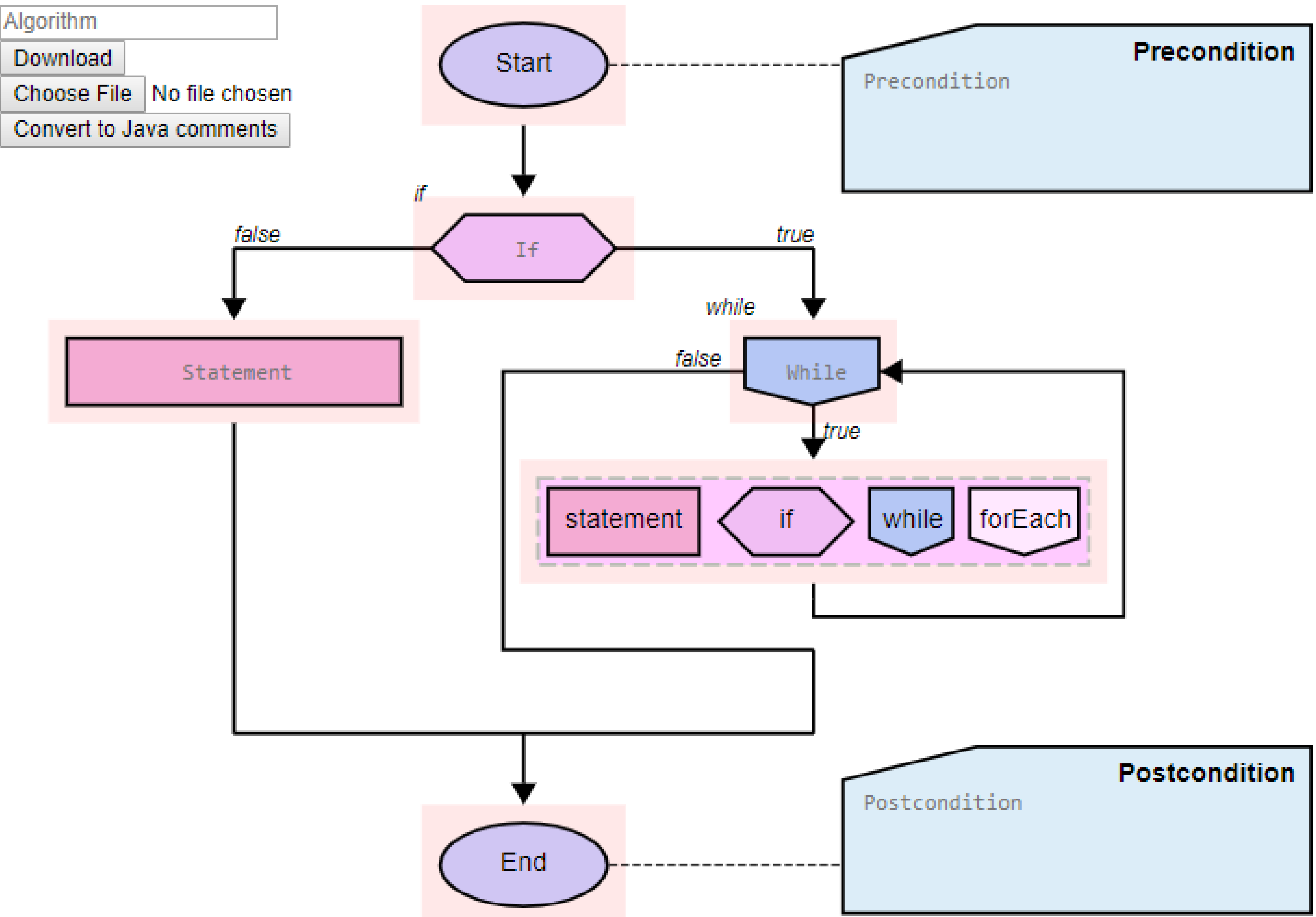
- Uitdagende vervolgopdrachten voor leerlingen die sneller/vooruit werken
- Theorie en opgaven Object georiënteerd programmeren
- Spel Sokoban & pin-ball simulatie



Materiaal

- Rechtstreeks (zonder aanpassingen) in de klas te gebruiken
- Compleet, inclusief:
  - Theorie (geen extra boeken nodig)
  - Uitwerkingen
  - Scenario's (voorgegeven code)
  - Voorbeeld toetsopgaven
  - Tool voor het maken van stroomdiagrammen
  - Afdrukbare versie als PDF
  - Keuze: Nederlands of Engels
- Beschikbaar via:
  - http://course.cs.ru.nl/greenfoot/ en binnen Informatica Actief

Nooit meer spaghetti-stroomdiagrammen met de tool:



http://course.cs.ru.nl/greenfoot/flowchart/flowcharttool.html

Ervaringen

- Ontwikkeld in samenwerking met de Radboud Universiteit
- Ingezet bij:
  - Verschillende middelbare scholen: HAVO & VWO
  - Als masterclass voor VO scholieren (onder begeleiding van studenten)
  - Cursus Object-oriëntatie (voor eerstejaars Science-studenten)

- Ervaringen:
  - Lesmateriaal zit goed in elkaar, goede uitleg + uitwerkingen + basiscode
  - Er wordt heel veel geleerd
  - Eerste 4 hoofdstukken geschikt voor HAVO en VWO, rest voor VWO
  - Totale tijdsbesteding: 30 SLU

Wat hebben we van het nieuwe examenprogramma gezien?

Subdomein B1: Algoritmen

- De kandidaat kan een oplossingsrichting voor een probleem uitwerken tot een algoritme, daarbij standaardalgoritmen herkennen en gebruiken, en de correctheid en efficiëntie van digitale artefacten onderzoeken via de achterliggende algoritmen.

Subdomein D1: Ontwikkelen

- De kandidaat kan, voor een gegeven doelstelling, programmacomponenten ontwikkelen in een imperatieve programmeertaal, daarbij programmeertaalconstructies gebruiken die abstractie ondersteunen, en programmacomponenten zodanig structureren dat ze door anderen gemakkelijk te begrijpen en te evalueren zijn.

Subdomein D2: Inspecteren en aanpassen

- De kandidaat kan structuur en werking van gegeven programmacomponenten uitleggen, en zulke programmacomponenten aanpassen op basis van evaluatie of veranderde eisen.

B1 Algoritmen: Voorbeeldspecificaties

De kandidaat kan:

- een gegeven oplossingsrichting voor een probleem weergeven als een algoritme. Hierbij wordt verwacht dat kandidaten een algoritme op een gestructureerde wijze kunnen weergeven met bijvoorbeeld een flowchart of in pseudocode.
- Het algoritme is opgebouwd uit de basisbouwstenen opeenvolging, keuze en herhaling.
    - *vwo: Het algoritme kan recursie bevatten*
- een gegeven digitaal artefact modelleren met behulp van een algoritme.
- het gedrag van een programma onderzoeken via het onderliggende algoritme, en zo problemen (bijvoorbeeld fouten of inefficiëntie) herleiden tot aspecten van dat algoritme.
- een aantal standaardalgoritmen herkennen en gebruiken.
    - Kandidaten kennen standaardalgoritmen voor elementaire numerieke operaties zoals minimum en maximum bepalen, sommeren, en ten minste twee andere doelen zoals zoeken, sorteren, datacompressie en graafalgoritmen (bijvoorbeeld routebepaling).
    - Kandidaten kennen bij tenminste één doel ten minste twee standaardalgoritmen.
- de correctheid van een gegeven algoritme onderzoeken, en algoritmen (waaronder standaardalgoritmen), vergelijken met betrekking tot efficiëntie.

D1 Ontwikkelen: Voorbeeldspecificaties

De kandidaat kan:

- werkende programmacomponenten ontwikkelen in een imperatieve programmeertaal naar keuze en daarbij,
    - gebruik maken van aanduidingen voor dataobjecten zoals variabelen en constanten; toewijzing van waarden aan variabelen;
    - gebruik maken van de algoritmische bouwstenen opeenvolging, keuze en herhaling implementeren met behulp van controlestructuren in de gekozen programmeertaal;
    - datatypen implementeren aan de hand van elementaire datatypen en taalconstructies voor datastructuren in de gekozen programmeertaal;
    - gebruik maken van taalconstructies die abstractie ondersteunen;
    - doelgericht gebruik maken van mechanismen om de leesbaarheid van een programmacomponent te vergroten, zoals kiezen van betekenisvolle namen voor dataobjecten, gebruik van procedures en functies, commentaar, en suggestieve lay-out;
    - debugging en testen inzetten.
- vanuit een algoritme een werkend programma ontwikkelen in de gekozen programmeertaal.

Bron: Concept Handreiking schoolexamen informatica, Januari 2018

D2 Inspecteren en aanpassen: Voorbeeldspecificaties

De kandidaat kan:

- de structuur en werking van een gegeven programmacomponent uitleggen.
- een gegeven programmacomponent evalueren aan de hand van de eigenschappen correctheid, efficiëntie, en leesbaarheid.
- een bestaande programmacomponent aanpassen
  - als gevolg van een evaluatie van bijvoorbeeld de correctheid, efficiëntie of leesbaarheid van de programmacomponent.
  - als gevolg van een veranderde of uitgebreide doelstelling.

Vragen? Opmerkingen? Tips? Ervaringen?

- Spreek ons aan

of

- Mail naar: renske.smetsers@science.ru.nl

Lesmateriaal:
http://course.cs.ru.nl/greenfoot/ en Informatica Actief

Unplugged activiteiten (zoals pariteitsbit ‘trucje’):
www.informaticaunplugged.nl

Stroomdiagrammen tool:
http://course.cs.ru.nl/greenfoot/flowchart/flowcharttool.html