Algemene opzet cursus

– Algoritmisch Denken en Gestructureerd Programmeren in Greenfoot –

1 Licentie

2 Inleiding

Deze cursus richt zich primair op de kunst van het oplossen van computationele problemen. Doel van dit lesmateriaal is om leerlingen op een leuke manier kennis laten maken met algorithmisch denken en programmeren. Informatica is veel meer dan programmeren alleen! Het omvat puzzelen, creatief zijn, en het zoeken naar oplossingen.

Dit materiaal is onderscheidend omdat het zowel **leuk** als **serieus** is. Door het algoritmisch denken te koppelen aan een implementatie (het programmeren) krijgen leerlingen direct visuele feedback van het resultaat. De gebruikte Greenfoot programmeeromgeving en opdrachten zijn laagdrempelig en **speels**. De oplossing komt als het ware tot 'leven'. Dit is **leerzaam**. Doordat de oplossing **zichtbaar** is worden misconcepties snel evident. Daarnaast geeft het iets tastbaars om gezamenlijk te bespreken en te kunnen reflecteren over (kwaliteit en proces van) de gekozen oplossingsstrategie. Dit lesmateriaal is **serieus** omdat er wordt geleerd hoe op een gestructureerde manier een probleem op te lossen: van het afkaderen en ontwerpen (inclusief abstractie en decompositie) tot het implementeren, op kwaliteit beoordelen en verbeteren van een oplossing.

Dit lesmateriaal is dus primair gericht op het leren om op een gestructureerde wijze problemen te analyseren en **problemen op te lossen**. Het programmeren wordt daarbij gebruikt als middel om met de materie te experimenteren door de resultaten van een ontworpen oplossing zichtbaar te maken. Er wordt dus eerst geleerd om computationeel te denken, daarna hoe een geschikte oplossing geïmplementeerd kan worden.

De afsluitende opdracht is een onderlinge wedstrijd waarbij de leerlingen strijden om het meest optimale algoritme te ontwerpen en implementeren: wie kan de slimste Dodo maken?

We werken met enthousiasme nog met regelmaat aan ons materiaal. Tips en suggesties zijn van harte welkom! Wil je het materiaal gebruiken? Neem per email contact met ons op voor de nieuwste scenarios. Heb je nog andere vragen of opmerkingen, neem ook dan contact met ons op renske.weeda@gmail.com.

Steekwoorden: algoritmisch denken, problemen computationeel oplossen, gestructureerd werken, leren ontwerpen, objectgeoriënteerd programmeren, Java, Greenfoot

2.1 Cursusdoelen

Na het voltooien van deze cursus kan de leerling:

[1] De volledige licentie-tekst is te lezen op: https://creativecommons.org/licenses/by/4.0/.

- De fundamentele ideeën van de logica begrijpen;
- Logica toepassen om problemen snel op te lossen (zoals het sorteren van informatie);
- Algoritmes ontwikkelen, met meer diepgang dan de basishandelingen van de rekenkunde;
- Op een systematische wijze van probleemstelling tot een (generieke) oplossing komen:
  - een generieke oplossing te bedenken voor een probleem;
  - een algoritme op te stellen als een opeenvolging van stappen, keuzes of herhalingen;
  - een algoritme weer te geven in een stroomdiagram;
  - een stroomdiagram om te zetten naar programmacode;
  - gestructureerd en stapsgewijs code aanpassen, compileren, uitvoeren, debuggen en testen;
  - kwaliteit en correctheid van een ontwerp of oplossing beoordelen.
- Een programmeertaal (Java) gebruiken voor het uitdrukken en testen van een oplossing;
- Uitleggen dat de geleerde concepten worden gebruikt door wetenschappers om problemen op te lossen en te innoveren.

3 Inhoud cursus

In dit lesmateriaal nemen we een expliciete ontwerp-gebaseerde benadering aan. Dat wil zeggen dat het leren van computationele concepten en toepassingen gekoppeld is aan het implementeren van de oplossing in een concrete programmeertaal. Hierdoor ligt er zeker aan het begin ook nadruk op de programmeertaal en ontwikkelomgeving.

3.1 Probleemoplossend proces

Het proces om te komen tot een probleemoplossing omvat o.a.:

- problemen formuleren dusdanig dat een computer gebruikt kan worden om te helpen met het oplossen daarvan;
- logische ordenen en analyseren van data;
- oplossing bedenken door algoritmisch te denken (als serie van geordende stappen);
- gestructureerd werken: deelproblemen identificeren en afzonderlijk oplossingen ontwerpen, implementeren en testen;
- analyseren en redeneren over de kwaliteit van de oplossing (bv. efficiëntie en effectief gebruik van stappen en middelen);
- reflecteren over de gekozen oplossing en ontwikkelproces;
- generaliseren en het herkennen en gebruiken van oplossingen vanuit of in andere domeinen.

3.2 Algoritmische concepten:

- opeenvolging: identificeren van een reeks stappen voor een taak;
- conditionals: het nemen van beslissingen op basis van voorwaarden;
- loops (herhalingen): waarin dezelfde taak (of taken) meerdere malen plaatsvindt;
- operators: ondersteuning voor wiskundige en logische uitdrukkingen;
- variabelen: opslaan, ophalen en updaten van gegevens;
- lijsten: een verzameling van elementen.

3.3 In het praktijk:

Door de implementatie in Java (gebruikmakende van de Greenfoot ontwikkelomgeving) worden deze algoritmische concepten voor de leerling visueel. De probleemoplossende oefeningen omvatten o.a.:

- incrementeel / iteratief ontwikkelen: het ontwikkelen van een deeloplossing, deze dan uitproberen, en daarna verder gaan met het ontwikkelen van andere onderdelen;
- testen / debuggen: ervoor te zorgen dat dingen werken, het vinden en oplossen van fouten en het beoordelen van kwaliteit van een oplossing;
- hergebruik: het maken van iets door voort te bouwen op wat anderen, of jezelf, al gedaan hebben;
- abstractie / modularisatie: het opbouwen van iets groots door het samenstellen van collecties van kleinere delen.

3.4 Doelgroep

Deze cursus is opgezet voor bovenbouw leerlingen HAVO en VWO die het vak informatica volgen. Het materiaal kan ook gebruikt worden als eerste introductiecursus op HBO of Universiteit. Het materiaal is geschreven om zonder aanpassingen rechtstreeks in de klas te gebruiken. Om de docenten te begeleiden bij de implementatie wordt vanuit de RU en TU/Delft ook een bijscholingscursus aangeboden. Neem hiervoor contact op met het Pre-University College of Science (`http://www.ru.nl/pucofscience/`)

3.5 Opzet

3.5.1 Thema's van de opdrachten

1. Kennismaking met Greenfoot:
   - Greenfoot omgeving;
   - methodes, resultaten en parameters;
   - objecten en klassen;
   - de relatie tussen een algoritme, een stroomdiagram en programmacode;
   - compileren, debuggen, uitvoeren en testen.
2. Constructies:
   - generieke oplossingen bedenken voor een probleem;

- een algoritme op te stellen als een opeenvolging van stappen, keuzes (**`if`** .. `then` .. **`else`**) of herhalingen(**`while`**);
- stroomdiagrammen opstellen en omzetten naar programmacode;
- gestructureerd en stapsgewijs implementeren;
- na te denken over de kwaliteit van een oplossing.

3. Betere oplossingen:
   - combinaties van opeenvolgingen, keuzes, herhalingen (en nesting);
   - ontwerp optimalisatie;
   - abstractie, decompositie;
   - ontwerpen en implementeren van submethodes;
   - ontwerp en implementatie van een generiek algoritme.
4. Overzichtelijker en generieker:
   - logische operatoren en complexe condities;
   - return-statements;
   - complexe algoritmes ontwerpen;
   - modularisatie toepassen: submethodes afzonderlijk ontwerpen, schrijven, testen en vanuit de code aanroepen;
   - kwaliteitseisen van programma's en programmacode.
5. Dodo wordt slimmer:
   - variabelen declareren, initialiseren, waarde toekennen en gebruiken;
   - operators: waardetoekenning, vergelijkingen, rekenkundige bewerkingen, verhoging;
   - counter in een while-loop;
   - code tracing.
6. Dodo wordt blijvend slimmer:
   - instantievariabelen;
   - constructors;
   - een **`while`**-loop in Greenfoot omzetten naar een **`if`** .. `then` .. **`else`**.
7. Wedstrijd: Dodo's race
   - een complex algoritme bedenken;
   - Objecttypes en **`null`**;
   - *random* getallen;
   - klasseconstantes;
   - gebruik maken van lijsten en *for-each-loop*s;
   - Java Library Documentation gebruiken om bestaande Java methodes op te zoeken en te gebruiken;
   - reflecteren over proces en oplossing.
8. Optioneel: Sokoban
   - gebruikersinteractie in de code afhandelen;

- geneste **`if`** .. `then` .. **`else`** statements toepassen;
- de kennis uit de vorige opdrachten toepassen om zelf een spel te implementeren.

9. Extra uitdagende opdracht: Ballen en bumpers / Stuiterballen
   - een (2D) simulatie maken voor een echte (3D) situatie met een zelf geschreven programma;
   - formules uit een ander vak uitprogrammeren en toepassen.

3.5.2 Materiaal

Het materiaal van de cursus bestaat uit het volgende:

- Opdrachten: werkbladen met ieder een set afgebakende leerdoelen en daarbij behorende theorie, vragen, opgaven, (diagnostische toets) en samenvatting;
- Voorbeeld uitwerkingen van de opdrachten;
- Scenario's: voorgegeven programmacode waarmee de leerlingen aan de slag kunnen;
- Docentenhandleiding: uitleg over hoe het materiaal in te zetten in de les, inclusief lesideeën en adviezen.

4 Didactisch model

Met het didactisch model van de opdrachten kunnen leerlingen zelfverantwoordelijk leren. De uitleg en opdrachten bieden de leerlingen voldoende houvast. Daarnaast geeft het de docent de mogelijkheid in werkvormen te variëren.

4.1 Visie op het onderwijs

Leerlingen moeten de mogelijkheden hebben het volgende (actief) te ondervinden:

- ontwerpen: niet alleen bestaande dingen gebruiken en daarmee om gaan, maar ook nieuwe dingen creëren;
- belang: dingen creëren die persoonlijk zinvol en relevant zijn;
- samenwerking: werken met de producten van anderen;
- reflectie: een creatieve toepassing evalueren en heroverwegen.

4.2 Uitgangspunten materiaal

De samenstelling van de opdrachten is gebaseerd op de volgende uitgangspunten:

- Het is geschikt voor zelfstandig werken (op eigen tempo);
- De theorie wordt pas aangereikt op het moment dat de leerling met dat thema in aanraking komt of nodig heeft voor verdere verwerking;
- Theorie en opgaven wisselen elkaar af;
- Uitwerkingen van de (meeste van de) opgaven zijn beschikbaar;
- Toegevoegde lesideeën bieden aansluiting op klassikaal lesgeven waarin uitleg en verdere verdieping (o.a. reflectie) plaats kan vinden;
- Het is gratis. Er is geen boek vereist of nodig.

4.3 Rol Docent

De docent speelt een onmisbare rol in het begrijpen en de verankering van de kennis. Van de docent is de volgende ondersteuning in ieder geval gewenst:

- ondersteuning (troubleshooting) tijdens het maken van de opdrachten;
- stellen van vragen tijdens/na de opdrachten ter controle van begripsniveau;
- nabespreken van de opdrachten (in relatie tot de leerdoelen) om het verankeren van de kennis te verbeteren en laten reflecteren over de verrichte taken om tot een hoger niveau van begrijpen te komen (Bloom's taxonomie);
- voortgang controleren;
- controleren van afwisselen van rollen bij samenwerking zodat elke leerling de doelstelling haalt;
- bijsturen van (sterke) leerlingen bij de keuze voor het overslaan van bepaalde opgaven.

4.4 Voorkennis

Voor de opdrachten is geen programmeerkennis vereist. De vereiste wiskundige voorkennis komt overeen met wat in de onderbouw geleerd wordt:

- het kunnen werken met een cartesisch coördinatenstelsel;
- algebraïsche formules kunnen opstellen;
- kunnen werken met variabelen.

Bij nieuwe theorie wordt meestal een beroep gedaan op voorkennis uit de voorafgaande opdrachten. De benodigde theorie wordt vaak kort herhaald in de opdrachten, en daarna uitgebreid. Daar waar expliciet bepaalde kennis nodig is die nog weinig geoefend is, wordt er terugverwezen naar de uitleg of opgave waarin de voorkennis aan bod is gekomen.

4.5 Studielast

Totale studielast voor de 8 opdrachten bedraagt 64 klokuren inclusief eventuele instructie (lessen). Voor elke opdracht staat ongeveer 4 uur. Omdat in de eerste lessen meer tijd moet worden besteed aan het verkennen van de Greenfoot omgeving en instructie over programmeren, zijn de eerste 4 opdrachten korter. Hierdoor krijgt de leerling de optimale gelegenheid om zich de basisstof eigen te maken. Vanaf opdracht 5 worden de opdrachten duidelijk uitdagender, en daardoor ook tijdrovender. Daarnaast krijgen de opdrachten een meer open karakter waardoor de leerlingen er naar belief mogelijk meer tijd in steken om het resultaat mooier of leuker te maken.

Er wordt ervan uitgegaan dat een leerling gedurende ten minste drie-kwart van de studielasturen ondersteuning heeft en beroep kan doen op de begeleiding van een ter zake kundige informaticadocent. Op deze wijze hoeft een leerling bij knelpunten niet onnodig lang vast te zitten.