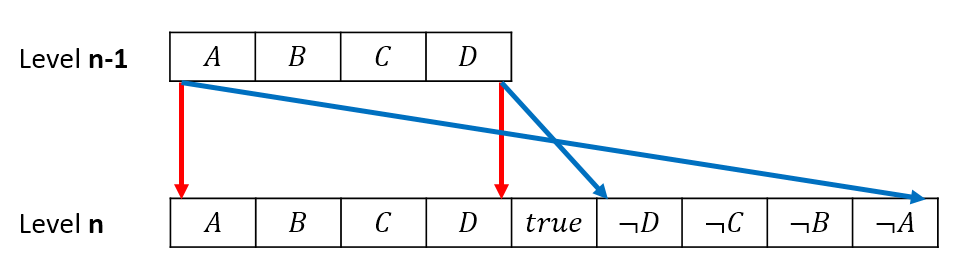
# Vraag 1

*(Aanbevolen tijd: 60 min.)*

Voor het eerste deel van deze vraag is de opdracht om methodes te maken die een lijst van booleans teruggeven volgens een bepaald patroon. Je krijgt de nodige klassenbestanden: **Drakencurve** en **Drakencurvepaneel**, die respectievlijk een **JFrame** en een **JPanel** zijn. In **Drakencurvepaneel** moet nog de code van de applicatie komen. Implementeer de volgende methodes in **Drakencurvepaneel**:

1. Schrijf een methode **List<Boolean> blockcurve(int n)** die een lijst van booleans teruggeeft van lengte n. Deze lijst bevat alternerend twee **trues** en twee **falses**:
2. Maak vervolgens methode **List<Boolean> dragoncurve(int n)**. Deze methode gaat een lijst van booleans creëren volgens een specifiek algoritme (de betekenis hiervan wordt duidelijk in deelvraag 3). Om de lijst van niveau **n** op te bouwen onderneem je de volgende stappen:
   1. Neem de lijst van niveau **n-1** (niveau 0 is een lege lijst)
   2. Voeg een **true** toe aan de lijst
   3. Voeg een gespiegelde kopie toe van de lijst van niveau **n-1** toe, waarvan je van elk element de negatie neemt (**true** wordt **false** en vice versa).



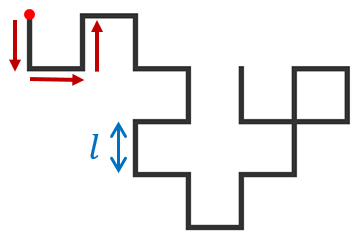
Figuur 1: Grafische voorstelling van de procedure voor het genereren van de lijst voor de methode “dragoncurve”

Hieronder een voorbeeld van de eerste 4 niveaus:

1. **true**
2. true **true** false
3. true true false **true** true false false
4. true true false true true false false **true** true true false false true false false

*Tip: maak een procedure gegeven een booleanse lijst, een nieuwe lijst maakt van een hoger niveau, zoals aangegeven in Figuur 1*.

De methodes uit deelvragen 1 en 2 gaan we gebruiken om een applicatie schrijven die “rechthoekige curves” kan tekenen. Deze curves bestaan uit een opeenvolging van lijnstukken van gelijke lengte, die allen met elkaar hoeken van 90° vormen. Een voorbeeld is hieronder getekend:



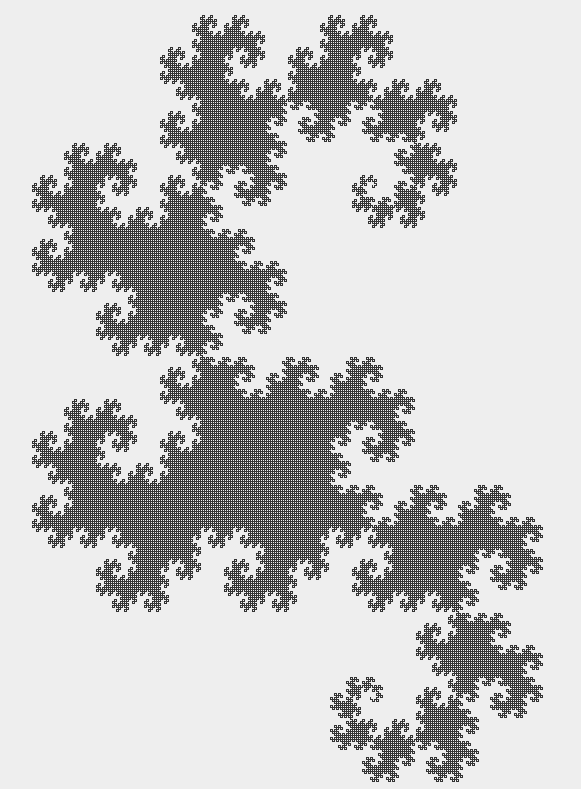
Figuur 2: voorbeeld van de rechthoekige curve

De curve begint in een gekozen punt (gericht naar beneden) met een lijnstuk van lengte . Na elk lijnstuk slaagt de curve naar links of naar rechts af; dit wordt voorgesteld door een boolean (links = **true**, rechts = **false**). Een volledige curve kan dus beschreven worden als een lijst van booleans. Voor de bovenstaande curve komt dit overeen met:

**{true, true, false, false, true, false, false, true, true, false, true, true, false, true, false, true, true, true, false, false}**

1. Maak een methode **drawcurve(Graphics g, List<Boolean> curve, int x, int y, int l)**. Deze tekent een rechthoekige curve met als startpunt de coördinaten (**x**,**y**) en lijnstuk-lengte **l**. Gegeven een lijst **curve**, wordt de rechthoekige curve getekend met de corresponderende draaiingen op het Graphics-object **g**. Test de methode (apart) uit met lijsten gegenereerd in **blockcurve** en **dragoncurve:**

* Voor **blockcurve:** gebruik , en .
* Voor **dragoncurve:** gebruik , en .

   
Figuur 3: Drakencurve van niveau 16

  
Figuur 4: Voorbeeld van een blockcurve

# Vraag 2

*(Aanbevolen tijd: 60 min.)*

Voor deze vraag is het de bedoeling dat je extra aandacht besteedt aan de structuur van je code. Denk aan een logische opdeling in klassen en probeer redundante code te vermijden.

We gaan een continue 2D-ruimte tekenen met muren en bewegende ballen. Zie ook de illustratie en de **hulpcode** op de volgende pagina.

* **Muren** zijn rechthoeken en worden voorgesteld door een positie (x,y) en een grootte (breedte, hoogte). Er worden 4 muren aan de randen van het scherm geplaatst, maar er moeten ook minstens 2 muren aan de binnenkant van de ruimte komen. Muren hebben ook een kleur.
* **Ballen** zijn **bewegende objecten** en worden voorgesteld door een positie (x,y), een snelheid (vx,vy) en een straal (r). Ballen hebben ook een kleur.
* **TekenPaneel** is een klasse die gegeven wordt. Deze klasse bevat een TimerTask gameLoop die elke 100 miliseconden herhaald wordt. Bij elke herhaling van deze gameLoop moeten de bewegende objecten verplaatst en hertekend worden. Er moet ook gecontroleerd worden of deze objecten niet botsen met de muren of met elkaar.
  + **Bal** botst met **Muur**: de bal wordt teruggezet op zijn vorige positie en de snelheidsvector van de bal wordt omgedraaid (met 180°)
  + **Bal** botst met andere **Bal**: de ballen wisselen van snelheidsvector

Er zijn twee soorten ballen:

* De eerste soort wordt geïnitialiseerd met een random snelheid en volgt een recht pad volgens die lijn.
* De tweede soort wordt ook geïnitialiseerd met een random snelheid, maar deze snelheid wordt elke 100ms opgeteld met een vector waarbij en twee verschillende random getallen zijn uit het interval . Wanneer de grootte van de snelheid groter wordt dan 5, dan wordt de snelheid genormaliseerd. De ballen van de tweede soort volgen dus een willekeurig pad.

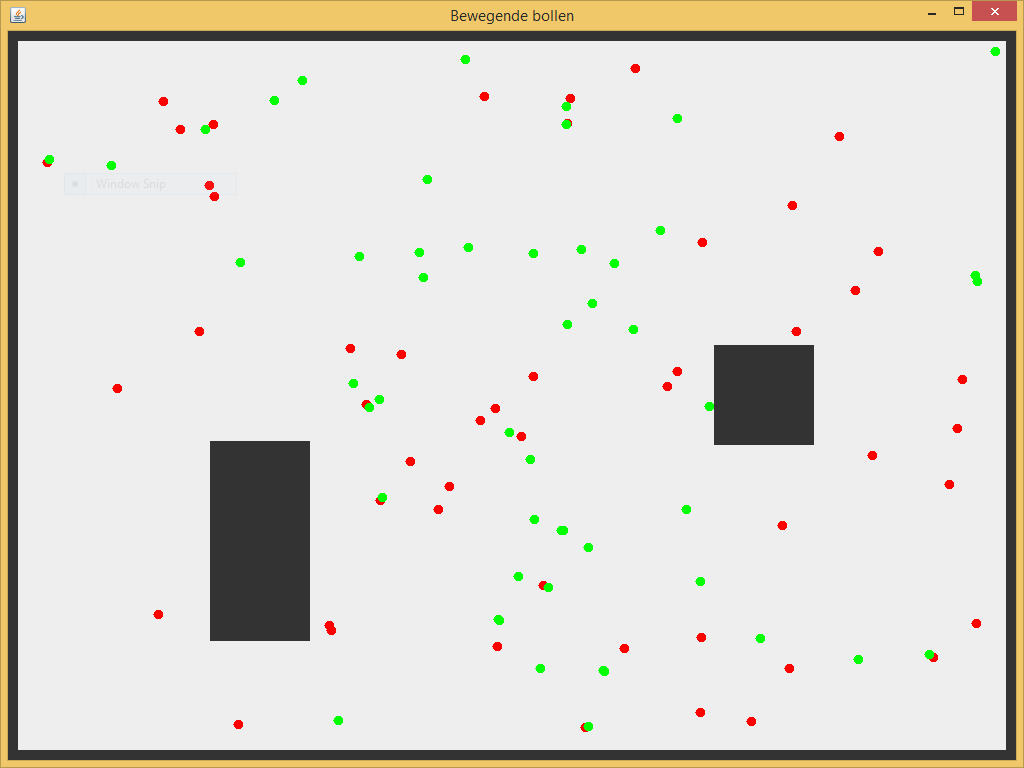
Grootheden zoals positie en snelheid stel je voor met objecten van een zelfgemaakte **Vector** klasse.  
Een **Vector** bestaat uit twee **double**s (x en y) en bevat de volgende methodes:

* **void** add(**Vector** b), de vector b wordt opgeteld
* **void** mul(**double** s), vermenigvuldigt de vector met een scalair s
* **double** dot(**Vector** b), geeft het scalair product terug met vector b
* **double** norm(), geeft de norm van de vector terug
* **void** normalize(), normalizeert de vector (zodat de norm van de vector gelijk wordt aan 1)
* **static Vector** sum(**Vector** a, **Vector** b), geeft een vector terug die gelijk is aan

Om random getallen te genereren kan je de **Random** klasse van Java gebruiken. De methode nextDouble() geeft een **double** terug die uniform verdeeld is tussen 0.0 en 1.0, nextInt(int n) geeft een willekeurige **int** terug van 0 tot en met n-1.

Om te contorleren of een Bal botst met een Muur kan je de volgende code gebruiken:  
**Let op: een Ellips wordt in Java gedefinieerd aan de hand van de omvattende rechthoek!**

import **java.awt.geom.\***;  
**Rectangle2D** rechthoek = new **Rectangle2D**.**Double**(x\_r, y\_r, breedte\_r, hoogte\_r);  
**Ellipse2D** cirkel = new **Ellipse2D**.**Double**( x\_c, y\_c, breedte\_c, hoogte\_c);  
**Area** oppervlak = new **Area**(cirkel);  
oppervlak.intersects(rechthoek); // geeft true of false



Maak het scherm dicht door vier muren aan de buitenkant van het scherm te plaatsen. Plaats ook een twee muren ergens binnenin het scherm. Maak 50 ballen die willekeurig bewegen en 50 ballen die volgens een lijn bewegen. Zorg dat elke soort een andere kleur heeft.

Laat alle ballen beginnen op een zelfde positie (niet in een muur) en geef ze een willekeurige beginsnelheid met .