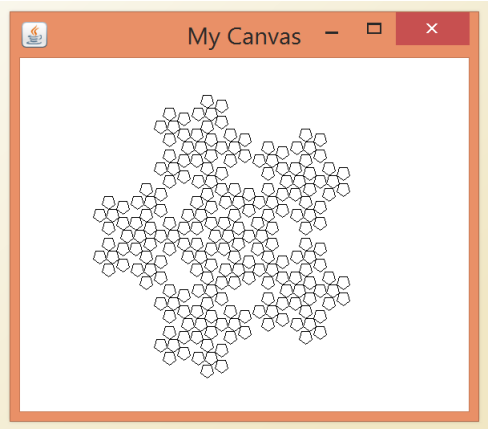
Opdracht 10: Lindenmayer systemen

– Algoritmisch Denken en Gestructureerd Programmeren in Greenfoot –

1 Inleiding

Tijdens het college zijn zogenaamde *L-systemen* behandeld. Met deze L-systemen kun je op een tamelijk simpele manier mooie, recursieve patronen maken. Hieronder staat een voorbeeld, maar er zijn heel veel patronen mogelijk.



Voordat we dieper ingaan op de werking van deze systemen, maken we kennis met nieuwe programmeeromgeving die we voor deze opdracht zullen gaan gebruiken: BlueJ. Zo leer je ook hoe het is om te werken in een ander programmeeromgeving dan Greenfoot.

2 Leerdoelen

Na het voltooien van deze opdracht kun je:

- je weg vinden in de programmeeromgeving BlueJ;
- zelf algemene klasses (die niet langer typerend zijn voor Greenfoot) implementeren;
- (eenvoudige) recursieve definities in de vorm van herschrijfregels herkennen, begrijpen en toepassen;

3 Aan de slag met BlueJ

3.1 Opstarten

Om kennis te maken met BlueJ gaan we eerst met een voorgegeven project oefenen (in BlueJ spreken we niet meer over scenario's maar over projecten).

Voorbereiding: Net als bij Greenfoot kun je het beste een vaste plek bedenken waar je de project-informatie gaat opslaan.

- Kopieer het in Blackboard voorgegeven en als *zip*-bestand ingepakte startproject naar de door jou uitgekozen opslagplek. Pak het *zip*-bestand hier uit (rechts-klikken op het bestand en *Extract All...* selecteren.

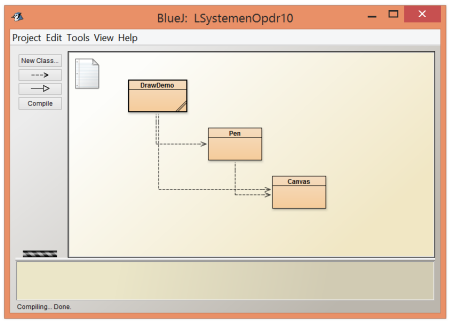
Open de BlueJ-omgeving: Deze staat geïnstalleerd op jouw PC[1].

- Open de windows verkenner en ga naar de **software (S:)** schijf. Hierop staat een map genaamd **BlueJ**. Open deze en klik vervolgens op de applicatie **BlueJ.exe**. De omgeving hoort nu op te starten.

Open het project:

- Selecteer 'Project' in het hoofdmenu en daarna 'Open Project ...'. Navigeer naar het project **'LSystemenOpdr10'** en kies 'Open'.
- Druk linksboven op de knop 'Compile'.

Als het goed is krijg je het volgende te zien:



Figuur 1: Wat je te zien krijgt na het openen van 'LSystemenOpdr10'

Het projectvenster lijkt enigszins op de manier waarop scenario's in Greenfoot werden weergegeven. De meest opvallende verschillen zijn:

- Het klassendiagram wordt in BlueJ prominenter getoond dan in Greenfoot. De reden hiervoor is dat een standaard Java programma vaak meer klassen bevat dan een gemiddelde Greenfoot applicatie, maar vooral ook omdat de klassestructuur vaak complexer is. Dit laatste komt ook omdat niet alleen de subklasse-relatie wordt weergegeven, maar (door middel van een gestippelde pijl) ook of de ene klasse de andere klasse gebruikt. Het aantal pijlen tussen klassen zal hierdoor aanzienlijk toenemen.

- De wereld waarin de instanties van klassen (de concrete objecten) worden geplaatst lijkt nu te ontbreken. Dit is deels waar. We hebben niet langer een wereld waarin zich het scenario afspeelt, maar wel nog steeds een aantal objecten waarvan we methodes kunnen aanroepen of die onderling elkaars methodes aanroepen. Je kunt dus nog steeds nieuwe instanties maken die vervolgens onderin het venster worden weergegeven en je kunt object-methodes aanroepen door ze via de rechtermuisknop te selecteren. Net als je in Greenfoot met **`new`** een nieuw object aanmaakt, roep je Klasse-methodes aan door op de klasse zelf te klikken (met je rechtermuisknop).

- De knoppen waarmee de uitvoering van een Greenfoot-scenario konden worden geregeld ontbreken geheel. Dit komt omdat objecten niet langer de klasse `Actor` uitbreiden en dus ook geen `act` methode meer hebben. In BlueJ kun je enkel nog via de rechtermuisknop methodes aanroepen.

[1]Je kunt BlueJ ook zelf op je eigen PC installeren. Ga hiervoor naar `http://www.bluej.org/` en volg de aangegeven instructies.

3.2 Objecten aanmaken en methodes uitvoeren

We gaan nu wat experimenteren met het voorgegeven project:

1. Klik met de rechtermuisknop op de `DrawDemo` klasse.
2. Kies **`new`** `DrawDemo( )` om een nieuw `DrawDemo`-object te maken.
3. Er verschijnt nu een dialog waarin je een naam kunt geven aan het zojuist gecreëerde object. Er staat al een naam ingevuld, die je gewoon kunt overnemen door meteen op 'OK' te drukken.
4. Het nieuwe-object verschijnt nu aan de onderzijde van het scherm, weergegeven als rood blokje.
5. Ook verschijnt er een nieuw venstertje met de titel 'Drawing Demo'.
6. Je kunt nu van dit nieuwe object methodes aanroepen. Klik met de rechtermuisknop op het nieuwe object (rood blokje). Probeer de 3 methodes allemaal uit.

We bespreken nu de twee andere klassen `Canvas` en `Pen`. Een object uit de `Canvas`-klasse zal een venster openen waarin getekend kan worden. Je kunt dit vergelijken met een vel papier of een doek. Er zijn verschillende methodes voor het tekenen, bijvoorbeeld van lijnen, bepaalde vormen en tekst. We gebruiken deze klasse niet direct voor ons tekenwerk, maar introduceren hiervoor een hulpklasse genaamd `Pen`. Een pen-object is zelf niet zichtbaar maar de punt hiervan bevindt zich wel steeds op een bepaalde positie op het scherm. Als we deze positie veranderen dan laat de pen een lijn achter die de nieuwe met de oude positie verbindt.

3.3 Documentatie bekijken

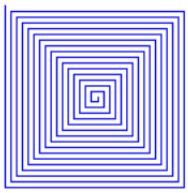
We gaan nu de `Pen`-klasse wat beter bekijken.

1. Open de `Pen`-klasse in de editor.
2. Rechtsbovenin zie je dat nu 'Source Code' geselecteerd is. Verander dit in 'Documentation'.
3. Zoek de constructor van deze klasse. Wat valt je op aan het gebruikte coördinatenstelsel?
4. Bekijk de andere methoden zodat je een idee hebt van wat je met een pen-object kun doen. Je zult dit nodig hebben in het vervolg van deze opdracht. De methodes `save` en `restore` kun je misschien niet meteen plaatsen. Deze heb je eigenlijk alleen nodig als je het optionele onderdeel aan het einde van deze opdracht wil maken.

3.4 Code uitbreiden

We gaan nu de klasse `DrawDemo` uitbreiden.

1. Open deze klasse in de editor.
2. Zoek de constructor op en verander de code zodanig het geopende tekenvenster een andere, zelfbedachte titel krijgt.
3. Bekijk verder de drie methodes die je in eerder al had aangeroepen. Er komen hier nogal wat (soms mysterieuze) onbenoemde constantes in voor, zoals 10, 4, enz. Vervang deze allemaal door klasseconstantes waarvoor je natuurlijk een zinvolle naam bedenkt.
4. Voeg zelf een methode toe voor het teken van de spiraal, vergelijkbaar met die uit onderstaand plaatje. Tip: teken van binnen naar buiten.



Figuur 2: Een spiraal

Nu we wat hebben geëxperimenteerd met de BlueJ omgeving en de voorgegeven code gaan we verder met de eigenlijke opdracht: het tekenen van L-systemen.

L-systemen

Een L-systeem bestaat uit:

- een aantal *productieregels*;
- een *startconfiguratie* (een of andere reeks van symbolen, hier van het type `String`); We zullen het voortaan hebben over de *startstring*.
- de *diepte* (een positief getal);
- een *hoek* (uitgedrukt in graden);
- een *lengte* (ook een positieve waarde waarvan de betekenis straks aan de orde komt).

Het idee is dat je de startstring gaat *herschrijven* door gebruik te maken van de productieregels. Elke productie koppelt een symbool (het *bronsymbool*) aan een tekst (de *doeltekst*).

Voor elke symbool B in de huidige string waarvoor een herschrijfregel bestaat (die dus B als bronsymbool heeft), dient dit symbool vervangen te worden door de bijbehorende doeltekst. Zodra dit voor alle symbolen in die startstring is gebeurd hebben we de volgende resultaatstring gekregen, waar we weer mee aan de slag gaan.

Dit process wordt een aantal keren herhaald, afhankelijk van de mate van detail die jouw uiteindelijke plaatje moet gaan krijgen. Het aantal opeenvolgende resultaatstrings dat nodig is voor het bereiken van de eindstring wordt ook wel de *diepte* (van het *L-systeem*) genoemd. Afhankelijk van de structuur van de regels kan dit proces enorm grote nieuwe strings produceren.

In principe heeft dit *herschrijven* van symbolen steeds een nieuwe reeks van symbolen als resultaat. Op zich is dit niet zo spannend. Het wordt pas leuk(er) als je zo'n string grafisch interpreteert, dat wil zeggen, je beschouwt de symbolen in je eindstring als *tekeninstructies*. Voor het uitvoeren van deze tekeninstructies maken we natuurlijk gebruik van de `Pen`-klasse.

Om te begrijpen hoe het herschrijven in z'n werk gaat bekijken we het volgende eenvoudige L-systeem, dat slechts één herschrijfregel heeft:

$$F \quad \rightarrow \quad F + F - F$$

Als startstring nemen we een tekst die slechts uit één teken bestaat: F. We kunnen nu de herschrijfregel toepassen en deze F vervangen door de string $F + F - F$. Hierbij zijn $+$ en $-$ geen rekenkundige plus en min, maar tekens in ons herschrijfsysteem. Vervolgens kunnen we weer de herschrijfregel toepassen bij de tweede F (die na de $+$) en deze weer vervangen. En ook de derde F (die na de $-$). Als we alle F's in de string hebben gehad zijn we klaar. We krijgen dan de resultaatstring:

$$F + F - F \quad + \quad F + F - F \quad - \quad F + F - F$$

Voor de duidelijkheid staat er extra witruimte tussen de herschreven delen van onze string. We zitten nu op diepte 2.

Ook deze string kan met de herschrijfregel weer herschreven worden, door één voor één elke F te herschrijven. Hiermee bereiken we diepte 3. En zo kunnen we doorgaan. We gaan net zo lang door met het herschrijven van de strings totdat de gewenste diepte is bereikt.

Alvorens we beschrijven hoe we de eindstring gaan omzetten in een plaatje gaan we een algoritme ontwerpen en implementeren dat in staat is om voor een gegeven L-systeem de eindstring te genereren.

We voegen hiervoor een nieuwe klasse, genaamd `LSystem` aan ons project toe. De strings representeren we natuurlijk direct als `String`. Maar wat gebruiken we voor de herschrijfregels? Hiervoor heeft Java een handig type, namelijk `Map`, waarmee je net als bij lijsten objecten kunt groeperen. Maar waar je bij lijsten de *index* gebruikte om opgeslagen elementen op te halen gebruik je in een map een *sleutel*. Voor de opslag van een element geef je niet alleen dat element aan, maar ook de sleutel waarmee het weer kan worden opgehaald. Essentieel hierbij is dat sleutels uniek zijn, dus ik kan niet twee verschillende elementen met dezelfde sleutel opslaan. De operatie om zo'n sleutel-element-paar op te slaan heeft de volgende signatuur:

```
V put(K key, V value)
```

Die ziet er een beetje vreemd uit, omdat op het eerste oog niet duidelijk is wat `V` en `K` zijn. Dit heeft te maken met het feit dat `Map`, net als `List`, een zogenaamd *generisch* type is[2]. Bij het gebruik van `Map` moet je zowel het type van de opgeslagen elementen opgeven alsook het type van de sleutels. Als we teruggaan naar ons voorbeeld en bedenken dat we herschrijfregels willen opslaan, dan ligt het voor de hand om het bronsymbool (een `Character`) als sleutel te gebruiken en de doeltekst (`String`) als element. Oftewel, onze verzameling van regels heeft type `Map<Character,String>`[3].

3.5 L-systemen implementeren

3.5.1 Nieuwe klasse `LSystem`

1. Maak een nieuwe klasse aan (door in het BlueJ linksboven op de knop 'New Class...' te drukken) en geef deze de naam `LSystem`.
2. Voeg hieraan een instantievariabele toe, bijvoorbeeld met naam `myRules` die als type `Map<Character,String>` heeft.
3. Voeg ook een import-statement **`import`** `java.util.Map;` waarmee je de `Map` klasse importeert.
4. Geef de klasse ook een constructor. Hierin wijs je aan de instantievariabele (`myRules`) een geschikt initieel object toe. Je verwacht hier misschien iets in de trant van

   ```
   myRules = new Map<Character,String>();
   ```

   maar dat is niet juist. Er zijn verschillende manieren om een nieuwe lege `map` aan te maken. Het beste kun je hiervoor **`new`** `HashMap<>` () voor gebruiken. En ook deze `HashMap` moet je importeren: **`import`** `java.util.HashMap;`.
5. Compileer je klasse en als je foutmeldingen krijgt, verbeter deze fouten voordat je verder gaat.

[2] Het verschil tussen `Map` en `List` is dat de `Map` twee generische variabelen heeft en `List`, waarbij we alleen het elementtype dienen aan te geven, slecht één.

[3] Mogelijk dat je hier **`char`** verwacht had i.p.v. `Character`. In Java is het echter niet toegestaan om primitieve waardes (zoals **`int`**, **`boolean`** en **`char`**) direct op te slaan in wat in Java *collections* genoemd worden (zoals `List` en `Map`). Hiervoor in de plaats dien je de bijbehorende objecttypes (`Integer`, `Boolean` en `Character`) te gebruiken.

3.5.2 De `main` methode

Voordat we de klasse `LSystem` nog verder gaan uitbreiden bekijken we eerst hoe deze klasse gebruikt kan worden. Dit doen we in de volgende methode genaamd `main`.

```
public static void main() {
    LSystem ls = new LSystem ();
    ls.addRule( 'F', "F-F++F+F-F-F" );
    String startString = "F-F-F-F-F";
    int diepte = 3;

    String nextString = startString;
    int i = 0;
    while ( i < diepte ) {
        nextString = ls.rewrite( nextString );
        i++;
    }
}
```

Toelichting.

- Hierin wordt eerst een instantie van de klasse `LSystem` gecreëerd. Dit `LSystem` is initieel 'leeg': het bevat nog geen regels of wat dan ook.
- Aan dit `LSystem` voegen we één regel toe (m.b.v. de methode `addRule`).
- De startstring wordt in de **`while`**-loop herschreven totdat de aangegeven diepte is bereikt. Voor het herschrijven wordt gebruik gemaakt van de methode `rewrite`.

3.5.3 Methodes voor het herschrijven

Rest ons nog de twee methodes te implementeren waarvan we in de `main` gebruik van maken: `addRule` en `rewrite`. Aan jou de opdracht om dat te doen.

1. Voeg de methode `addRule` toe aan de klasse `LSystem`. Bepaal uit bovenstaand voorbeeld wat de signatuur is en zorg ervoor dat de herschrijfregel die als twee parameters wordt meegegeven ook wordt opgeslagen in `myRules`. Dit kan met één enkel statement.
2. De methode `rewrite` is wat meer werk. Deze krijgt de huidige string mee en doorloopt deze om de symbolen waarvoor een herschrijfregel bestaat, te vervangen door de bijbehorende doeltekst (tot nog toe hadden onze voorbeeld steeds maar één regel, maar dat wordt straks anders).
   (a) Er bestaan verschillende manieren om een string te doorlopen. We beschrijven de manier die veel gelijkenis toont met het doorlopen van lijsten. Met de `String`-methode

   ```
   char charAt( int index )
   ```

   kun je een teken op positie `index` ophalen. Stel we hebben een string-variabele `str` die naar de string `"hallo"` wijst. De aanroep `str.charAt(4)` levert het teken op positie 4 op, waarbij je weer begint te tellen vanaf 0. Het resultaat zal dus het teken `'o'` zijn. Door een **`while`**-loop te gebruiken kunnen je met deze `charAt` methode alle tekens van een string bekijken. Zoek zelf op hoe je bepaalt hoe lang een gegeven string is.
   (b) Als je hier nog even mee will oefenen:
      i. Schrijf een methode

      ```
      public int countOccurrences( char c, String s)
      ```

die telt hoe vaak het opgegeven teken `c` voorkomt in `s`. Je kunt deze methode gewoon in de klasse `LSystem` zetten hoewel hij natuurlijk niets met L-systemen te maken heeft. Dit is ook maar bedoeld om even wat uit te proberen.

ii. Compileer en test je code.

StringBuilders

Voor het opbouwen van de nieuwe string heeft Java een handige klasse `StringBuilder` die eigenlijk voor dit soort taken bedoeld is. De naam zegt het eigenlijk al, met een stringbuilder kun je stapsgewijs een nieuwe string opbouwen, iets wat we tijdens het herschrijven willen doen. We laten eerst een voorbeeldje zien waarin het gebruik van `StringBuilder` wordt geïllustreerd. Ons uitgangspunt is een lijst van strings die we allemaal aan elkaar willen plakken zodat een lange string ontstaat. Neem bijvoorbeeld de volgende lijst:

```
List<String> stringList = Arrays.asList ("Eeny", "meeny", "miny", "moe");
```

De stringbuilder wordt gebruikt om het resultaat in op te bouwen. Allereerst hebben we een nieuwe lege stringbuilder nodig:

```
StringBuilder builder = new StringBuilder();
```

We gebruiken een *for-each-loop* om door de lijst heen te lopen en de `StringBuilder`-methode `append` om de huidige string toe te voegen aan wat tot nog toe is opgebouwd.

```
for ( String nextString: stringList) {
    builder.append( nextString );
}
```

Uiteindelijk willen we niet een StringBuilder maar een String als resultaat hebben. Dit kan met één instructie:

```
String result = builder.toString();
```

Na uitvoering van dit programmaatje zal `result` de waarde `"Eenymeenyminymoe"` bevatten. Als we spatie tussen de verschillende woorden kan dat eenvoudig door een extra statement aan de loop toe te voegen, direct volgend op de aanroep van `append`:

```
    builder.append( ' ' );
```

Je ziet aan dit voorbeeld dat je aan de methode `append` zowel een `String` (in dit voorbeeld `nextString`) als een **`char`** (hier, een spatie `' '`) kunt meegegeven. Kijk verder ook zeker naar de Java API van de `Stringbuilder` klasse.

(c) Declareer aan het begin van jouw herschrijfmethode een lokale variabele van het type `StringBuilder` en met een zelfgekozen, liefst zinvolle naam. Initialiseer deze variable met een nieuwe instantie van de `StringBuilder`-klasse.

(d) Doorloop nu je huidige string en bepaal teken voor teken wat er met dat teken moet gebeuren. Eigenlijk zijn er twee gevallen, afhankelijk van of er een herschrijfregel voor dat teken bestaat. Als dit laatste inderdaad het geval is dan wordt onze stringbuilder uitgebreid met de bijbehorende doelstring. Is dit niet het geval dan breiden we de stringbuilder uit met het huidige teken.

(e) Maak gebruik van de methode `get` om de regel voor een gegeven symbool in de `map` op te zoeken. Kijk weer in de API hoe deze methode werkt.

(f) Als je alle tekens gehad hebt kun je met de methode `toString` van een `StringBuilder` weer een string maken, die je uiteindelijk door `rewrite` laat opleveren.

(g) Compileer de klasse.

(h) Om je nieuwe methodes te kunnen testen kun je het beste de eerder gegeven `main` methode aan de klasse `DrawDemo` toevoegen. Zet in eerste instantie de diepte op 1 zodat je startstring maar één keer wordt herschreven. Druk (met `System.out.println`) het resultaat af en controleer of het juist is. Zo niet, verbeter je code. Omdat `main` een klasse-methode is hoef je geen `DrawDemo` instantie aan te maken, maar kun je deze direct aanroep door met de rechtermuisknop op de klasse zelf te klikken en de methode te selecteren.

3.5.4 Methode voor het tekenen

Nu we een fraaie string kunnen genereren, gaan we deze tekenen. Hiervoor is het nodig om de volgende twee waardes te weten (zoals in de introductie al werd aangegeven, worden die altijd mee opgeven bij ieder L-systeem):

1. de *hoek*: sommige symbolen uit de eindstring zorgen ervoor dat de richting waarin getekend wordt over een bepaalde hoek zal worden gedraaid. De waarde van *hoek* bepaalt deze draaiingshoek.
2. de *lengte*: andere symbolen laten de pen een lijn (inde de richting die de pen op dat moment heeft) tekenen. De waarde van *lengte* bepaald de lengte van dit lijnstuk.

Het tekenen gebeurt nu als volgt:

- Als je in je eindstring een F tegenkomt, tekent de pen een lijn van lengte *lengte*.
- Als je een $+$ tegenkomt, draai je de pen *hoek* graden naar links.
- Bij een $-$ draai je de pen *hoek* graden naar rechts.

Voor dit tekenen maken we weer een aparte klasse, ditmaal `LSDrawer` genaamd. Hoe we deze klasse gaan gebruiken zie je aan het volgende code-fragment:

```
int hoek   = 72;
int lengte = 10;
LSDrawer lsd = new LSDrawer( hoek, lengte );
lsd.drawString( nextString );
```

Dit fragment volgt direct op de **`while`**-loop uit `main`. Je kunt deze code straks zelf hieraan toevoegen zodra de klasse klaar is om getest te worden.

1. Voeg de klasse `LSDrawer` toe aan jouw project.
2. Geef deze methode een instantievariabele van het type `Pen` die je in de constructor initialiseert met een nieuw `Pen`-object.
3. Implementeer de methode `drawString` die de meegegeven tekst op het scherm tekent (zoals hierboven aangegeven). Een paar aanwijzingen:
   - De string die aan `drawString` wordt weer teken voor teken bekeken. Gebruik hiervoor weer een **`while`**-loop gecombineerd het de `charAt`-methode om het volgende teken uit de string te halen.
   - Maak een aparte hulpmethode

     ```
     public void drawChar( char c ) {
     ```

     waarin je voor het meegeven teken `c` bepaalt wat er dient te gebeuren. Roep deze methode aan vanuit `drawString`. Deze opsplitsing zorgt ervoor dat je programma leesbaar blijft. Bovendien kun je `drawChar` los uittesten.

- Het kan zij dat je even met de grootte van het scherm in de beginpositie van de pen moet experimenteren om het gehele plaatje zichtbaar te maken.

4. Compileer en test je code. Breid voor dit laatste `main` uit met de voorbeeld-code.

3.5.5 Meerdere herschrijfregels

Het bovenstaande voorbeeld heeft maar één herschrijfregel. Er zijn echter ook patronen, die meerdere regels bevatten. Elke regel herschrijft één letter naar een nieuwe string. Bij het herschrijven van een `String`, moet je dus uitzoeken welk van de regels van toepassing is. Het kan hierdoor gebeuren dat andere letters dan F, $+$ en $-$ in de eindstring terecht komen. Deze letters waren alleen bedoeld om te herschrijven: bij het tekenen hoef je met deze letters niets te doen.

5. Pas jouw programma aan zodat het volgende L-systeem getekend wordt:

   - hoek = 60
   - diepte = 3 of 4
   - lengte = 10
   - beginstring: XF
   - herschrijfregels:

$$\begin{array}{lcl} X & \rightarrow & X+YF++YF-FX--FXFX-YF+ \\ Y & \rightarrow & -FX+YFYF++YF+FX--FX-Y \end{array}$$

3.5.6 Andere L-systemen

In het project zit nog een map met de naam 'systemen' waarin je een aantal bestandjes kunt vinden met daarin L-systemen. De inhoud van deze bestanden heeft steeds de volgorde vorm:

- hoek
- lengte
- diepte
- beginstring
- herschrijfregel 1
- herschrijfregel 2
- herschrijfregel 3
- ...

Probeer ook deze systemen uit om te zien wat voor plaatjes er bij horen. Opmerking: voor het systeem **'seaweed.lst'** heb je de uitbreiding die hieronder beschreven staat nodig.

3.5.7 Nog meer L-systemen

Verder kun je natuurlijk ook het internet afspeuren om nog veel fraaiere figuren tegen te komen. Soms heb je hiervoor nog wat extra's nodig in de vorm van speciale commando's voor het besturen van de pen. Tijdens het college werd nog een drietal extra commando's getoond:

[Sla de huidige toestand (= positie en richting) van de pen voor later gebruik op. Voor dit opslaan wordt een *stapel* gebruikt: er kunnen na elkaar meerdere toestanden worden bewaard. De laatste komt bovenop de stapel te liggen en wordt er als eerste weer afgehaald.

] Haal de bovenste toestand van de stapel en gebruik deze om de pen weer in deze toestand terug te zetten.

| Teken, net als F, een lijn van lengte *lengte*. Het bijzondere aan dit symbool is dat het bij elke herschrijfstap verdubbelt. In feite hebben we voor dit | de herschrijfregel | $\rightarrow$ ||

Je kunt zelf deze commando's aan de methode `drawChar` uit de klasse `LSDrawer` toevoegen en op zoek gaan naar wat leuke plaatjes om dit te testen.

Andere uitbreidingen (plaatjes met verschillende kleuren, drie-dimensionale plaatjes) vallen buiten deze opgave.

4 Opslaan en inleveren

Opslaan

Je bent klaar met de tiende opdracht. Sla je werk op.

1. Kies het menu 'Project' bovenin het BlueJ-venster, en dan 'Save As ...'.
2. Vul de bestandsnaam aan met jouw eigen naam en het opgavenummer, bijvoorbeeld `Opdr10_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 ...'.

Inleveren

Lever de opdracht in via Blackboard.

1. Ga naar het map waar je jouw werk hebt opgeslagen.
2. Standaard wordt er bij ieder scenario een 'README.TXT' bestand gegenereerd. Open het bestand 'README.TXT' en vul hier jullie namen en studentnummers in.
3. Comprimeer de hele map tot één `.zip` bestand.
4. Lever dit ene gecomprimeerde bestand in via Blackboard. Hou je aan de deadline!