# Vraag 1

*(Aanbevolen tijd: 20 min.)*

Morsecode is een communicatietechniek die symbolen encodeert als een opeenvolging van korte en lange signalen (bv. bieptonen, lichtsignalen, enz.). Een kort signaal wordt voorgesteld door een punt, een lang signaal als een streepje. Elk teken heeft zijn unieke voorstelling, bv: ‘A’ als **.-** , ‘X’ als **-..-**, ‘?’ als **..--..**

Het zinnetje “Hello world” wordt in Morse: **.... . .-.. .-.. --- .-- --- .-. .-.. -..**

Punten en streepjes kunnen op hun beurt ook in Morse-code worden voorgesteld:

**.** als **-. -** als **…-**

Dit betekent dat we een Morse-bericht meermaals opnieuw kunnen encoderen als een Morse-bericht:

.  -.  …--.  -.-.-….-…--.  …--….--….--.-.-.-….--.-.-….-…--.  enz.

Het vraagstuk dat je moet oplossen is het volgende: **als we een bericht in morsecode een gegeven aantal keer na elkaar coderen in morsecode, hoe lang is dan het uiteindelijke bericht in morsecode?** Je mag ervan uitgaan dat het invoerbericht zelf enkel uit streepjes en punten bestaat.

Implementeer in de package “vraag1” de methode “**repeatedmorse**”. Als invoer heb je begin-***String*** **str** en ***int*** **n** stelt het aantal keer dat het bericht herhaaldelijk na elkaar geëncodeerd moet worden. Als uitvoer geef je een **int** terug die de lengte van het finale bericht teruggeeft. De implementatie moet generiek zijn en werken voor alle mogelijke inputs (je mag veronderstellen dat de oplossing < 109)

*Alle code voor het uitvoeren en testen van de functie is al voor jou geïmplementeerd via externe libraries; je moet dus enkel de functie* ***repeatedmorse*** *implementeren en op “run” klikken.*

Tip: *de volgende methodes van* **String** *komen van pas:*

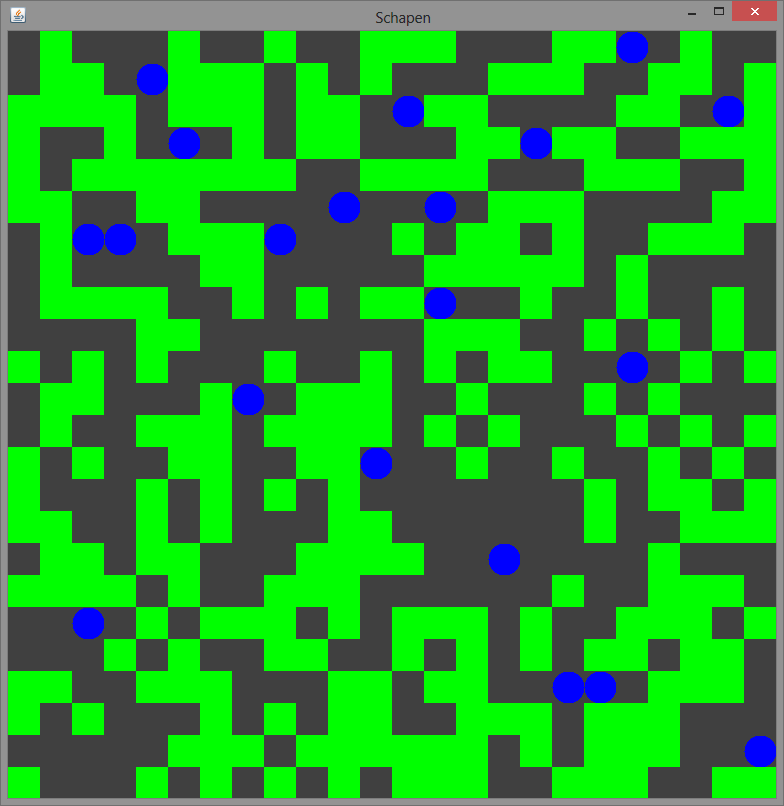
char charAt(int n): Geeft het karakter terug op de nde plaats.

int length(): Geeft de lengte terug van de string

# Vraag 2

*(Aanbevolen tijd: 60 min.)*

In deze opgave gaan we een biotoop simuleren. De wereld wordt voorgesteld door een 24x24 grid; op elk vakje (van 32x32 pixels) bevindt er zich al dan geen gras. Op dat grid lopen schapen rond. Schapen zetten elke tijdseenheid een stap in een willekeurige richting. Schapen eten gras waardoor hun energiemeter verhoogt. Wanneer schapen genoeg energie hebben, reproduceren ze; maar schapen verbruiken ook energie, en als die op is sterven ze en worden ze dus verwijderd.



Figuur 1: Voorstelling van het biotoop. Groene vakjes zijn begroeid met gras, grijze vakjes niet. Schapen worden voorgesteld als blauwe bolletjes.

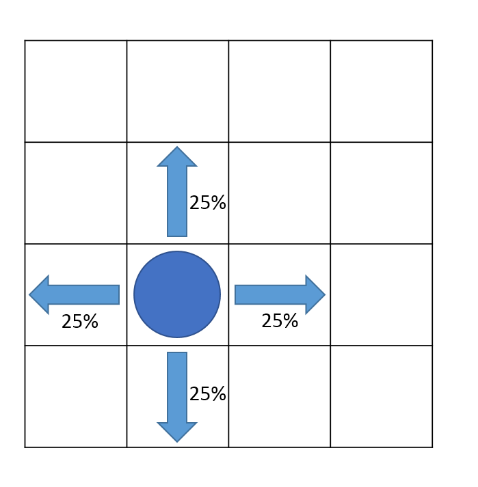
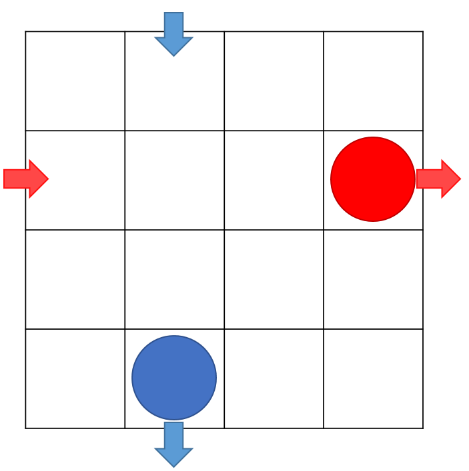
Je krijgt de twee eerste klassen: **Simulation** en **SimulationPanel**, die respectievlijk een **JFrame** en een **JPanel** zijn. In **SimulationPanel** moet nog de basislogica van de simulatie komen.

Elk grasveld heeft twee staten: gras of geen gras. Initialiseer het grasveld per vakje met 50% kans op gras. Na elke tijdseenheid is er 1% kans dat er op een leeg grasveld terug gras groeit.

15 schapen worden willekeurig op het veld geplaatst. Schapen zetten elke tijdseenheid een stap in een willekeurige richting (links, boven, rechts of beneden; zie Figuur 2). Het veld is toroidaal: als ze over de rand stappen verschijnen ze aan de tegengestelde rand van het veld, zoals in Pacman (zie Figuur 3).

Schapen hebben een energiemeter die initieel op 10 staat. Deze neemt met 1 punt af per tijdseenheid. Bereikt deze 0, dan gaat het schaap dood en wordt het verwijderd. Als een schaap op een begroeid vakje komt, wordt het gras opgegeten en verandert het in een leeg vakje; het schaap krijgt dan 10 energiepunten bij. Wanneer een schaap meer dan 30 punten heeft, baart het een nieuw schaap op dezelfde positie als het oude schaap. Dit kost het oude schaap 20 punten.

Laat de simulatie draaien met een periode van 100 ms. Teken gras als gevulde rechthoekjes (groen voor begroeid, donkergrijs voor onbegroeid) en schapen als gevulde blauwe cirkels.

Figuur 2: Bewegingsmogelijkheden van een schaap met bijgeschreven kansen.

Figuur 3: verplaatsing aan de randen van een torodoidaal veld.