CSCU9YM Practical Two
Building and exploring an ABM

This sheet contains two checkpoints.

In this practical you will develop models of the butterfly corridor system. This example was discussed in a lecture and is described in detail in Chapters 3 and 4 of Railsback and Grimm.

Download the file `ButterflyCorridors.nlogo` into your own home folder. Open your copy of this file within NetLogo. This is a starter file for a model of the butterfly corridor system. The Interface and Code tabs are empty. The Info tab contains the ODD description of this model, taken from Railsback and Grimm. Read through this carefully.

This sheet gives some hints how to complete this model but does **not** contain detailed instructions. As experienced programmers, you may be able to complete the model using the hints and referring to the **Programming Guide** and **NetLogo Dictionary** sections of the NetLogo User Manual. However, if you prefer detailed step-by-step instructions, they can be found in Chapter 4 of Railsback and Grimm.

To start with, you will implement a model containing just one butterfly in an artificial landscape containing two hills.

Interface settings

The world should be centred at the bottom left corner, and contain 150x150 patches (so the maximum x-coordinate and maximum y-coordinate of patches is 149). Reduce the patch size so the world fits on your screen. The world should not wrap.

Interface components

Initially you will need a setup button and a go button (which should be a "forever" button). Other components will be added later.

Variables

There needs to be a global variable representing the probability that a butterfly will choose to fly in a random direction rather than directly uphill. Following the textbook, we will call this variable `q` (much though this dreadful name choice offends my sensibilities as a software engineer!).

Consult the ODD description in the Info tab to see what turtle and patch variables are needed.

Setup procedure

The setup procedure needs to do the following:

- clear the model (using `clear-all` or `ca`; see the NetLogo Dictionary for details);
- create a landscape containing two hills;

- create the butterflies;
- set the global variable (use a probability of 0.4);
- call `reset-ticks` to reset the tick counter to 0.

Creating the landscape

Write a separate procedure to do this.

The code below can be used as the body of a procedure to create a landscape containing two hills. The first hill is 100 units high and is centred at (30, 30). The second hill is 50 units high and centred at (120, 100). The elevation of the land decreases linearly with distance from the centre of the hills. The code assumes patches have a user-defined variable called `elevation`.

```
ask patches
[
  let elev1 100 - distancexy 30 30   ;; elevation due to first hill
  let elev2 50 - distancexy 120 100  ;; elevation due to second hill

  ifelse elev1 > elev2  ;; set elevation to the greater of the two
    [set elevation elev1]
    [set elevation elev2]

  set pcolor scale-color green elevation 0 100
]
```

Make sure you understand how this code works. Do you understand the difference between `let` and `set`? If you are not sure, check the NetLogo Dictionary. Also, look up the meanings of the reporters `distancexy` and `scale-color`.

Creating the butterflies

Write a separate procedure to do this.

The butterfly is represented by a turtle, so you need to use `create-turtle` (or `crt`) to create it. Read about this command in the NetLogo dictionary. You can use square brackets after the command to introduce a *turtle context* containing commands to be carried out by the newly-created turtle. By using such commands, make the new turtle small enough to fit within a patch (adjust its size), position it at coordinates (85, 95) (use the `setxy` turtle command), and have it set its pen down so that it leaves a trail when it moves (use the `pen-down` turtle command).

Go procedure

The go procedure has the following behaviour:

- Ask the turtles to move
- Advance the clock by one tick
- If the number of ticks is greater than 1000, stop

As Java programmers you may wonder why there is an if statement instead of a while loop in the `go` procedure. The `go` procedure will loop because it is repeatedly invoked by the go button (because you made this a "forever" button). The if statement is needed to force the iteration to stop when the maximum desired number of ticks is reached. (Its function here is somewhat similar to a break statement in Java.)

Making the turtles move

The code below is a *turtle procedure* for moving the butterfly. The conciseness of this code illustrates the power of NetLogo for constructing complex models using very little code.

```
to move
  ifelse random-float 1 < q  ;; decide how to move by random choice
  [ uphill elevation ]    ;; move to the highest neighbouring patch
  [ move-to one-of neighbors ]   ;; or move randomly
end
```

Look up these commands / reporters in the NetLogo Dictionary: `random-float`, `uphill`, `move-to`, `one-of`, `neighbors`. Make sure you understand what they mean.

Play with your model. Does it always behave the same way every time you run it?

Now try these exercises (adapted from Railsback and Grimm):

1. Add a slider to the interface to control a variable called `num-butterflies`. Change your code to create `num-butterflies` butterflies instead of 1. Try running your model with the slider set at different positions.
2. Similarly, introduce a slider to control the probability q that a butterfly will move randomly instead of uphill. Observe what happens as you vary this probability.
3. Modify the model so that instead of all starting from the same patch, the butterflies all start from random locations within a 10x10 area of the original starting patch. Hint: add a random amount to the original starting coordinates.
4. The landscape in this model is very unrealistic: each hill is a smooth cone with a constant gradient. This gives the butterflies an unnaturally direct route to the hilltop. Make the landscape more irregular by adding a random number (from 0 to 9) to the elevation of each patch. How does this change how the butterflies move?

CHECKPOINT [BUTTERFLY 1]

Demonstrate the final model created after completing exercises 1 to 4.

For the next task, start with a copy of your completed ButterflyCorridors model. In your first model, you were able to observe the virtual butterfly corridors visually on the screen, but there was no way to measure or record the behaviour of your model so it cannot be studied

scientifically. For the next task you will introduce reporters and plots that track and record how the model behaves.

The behaviour of the model will be measured by calculating the "width" of the corridors generated by the moving butterflies. See Chapter 5 of Railsback and Grimm for a full discussion of this measure. In brief:

Corridor width = (total number of patches visited / mean distance travelled by butterflies)

How to remember which patches have been visited
Each patch will need to record whether it has been visited or not. Introduce a patch variable called `visited?` to record this. (The question mark is part of the variable name and indicates that this is a Boolean variable.) Have the setup procedure initialize this variable to `false`. When a butterfly moves onto a patch, that patches `visited?` variable should be set to true. This can be done very simply by adding the line

```
set visited? true
```

as the last line of the `move` procedure. Although `move` is a turtle procedure and `visited?` is a patch variable, the `move` procedure can update `visited?` directly because a turtle has direct access to the variables of the patch that it is on.

How to calculate the distance travelled by the butterflies
To do this, each butterfly needs to remember which patch it started from. Introduce a turtle variable called `start-patch` to record this. In the setup procedure which creates the butterflies, add the following line to the code that is executed by the butterflies as soon as they are created:

```
set start-patch patch-here
```

(Look up `patch-here` in the NetLogo Dictionary to find out more about it.)

The expression `[distance start-patch]` is a turtle reporter giving the straight line distance from a turtle's current location to its `start-patch`.

Calculating corridor width
The code below introduces an observer reporter which calculates the width of the corridors:

```
to-report corridor-width

  let num-patches-visited count patches with [visited?]
  let mean-distance mean [distance start-patch] of turtles

  report num-patches-visited / mean-distance

end
```

Observing the corridor width
To track how the corridor width changes as the butterflies move, add a plot to the interface. In the Plot dialog box, give the plot the name Corridor Width and delete any code that appears under **Pen update commands**. (Experience shows that it is safer to update the plot pen explicitly from the code in the Code tab rather than relying on the pen update commands in the Interface tab.)

In the Code tab, add the line
```
  plot corridor-width
```
to the `go` procedure immediately before the tick command.
Run the model a few times and see how the corridor width changes as the butterflies move. Try adjusting the sliders to see how this affects the plot.

Storing plot data in a .csv file
Read the NetLogo Dictionary entries for `export-plot` and `word`. Then modify the `go` procedure so that if the tick counter has reached 1000, the contents of the plot are written to a file before the model is stopped. The name of the file will include the values of the two sliders `q` and `num-butterflies`. Use the statement below. You can view the resulting file with Excel.

```
export-plot "Corridor width"
    (word "width-q-" q "-butterflies-" num-butterflies ".csv")
```

Working with a real landscape
For the final task you are going to import elevation data for a real landscape into your model. Make a copy of your model, so that you do not lose the work you have done so far.

Download the file `ElevationData.txt` and copy it into the folder containing your model. This file contains elevation data for a landscape used in a real study of butterfly movements. Each line of the file contains the x and y coordinates of a patch in the landscape, followed by its elevation.

Rewrite the procedure for making the hills in your model so that it imports data from this file and uses it to set the elevation of the patches. Your new procedure will need this code:

```
file-open "ElevationData.txt"
while [not file-at-end?]
[
  let next-X file-read
  let next-Y file-read
  let next-elevation file-read
  ask patch next-X next-Y [ set elevation next-elevation ]
]
file-close
```

You will need to add code to colour the patches to show their elevation, and initialize their `visited?` variable.
Experiment with the model using various values of `q`. What can you conclude about the relationship between `q` and the corridor width?

CHECKPOINT [BUTTERFLY 2]

Demonstrate your model with a real landscape and plots of corridor width.