Practical One:
Introduction to Agent-Based Modelling with NetLogo

This sheet contains three checkpoints.

This practical is derived from Chapter 2, Getting Started with NetLogo, of *Agent-Based and Individual-Based Modeling* (Railsback and Grimm 2012). The chapter contains fuller explanations of the tasks in this worksheet, particularly for the third checkpoint. **You are strongly recommended to refer to this chapter while attempting the worksheet.**

The worksheet will take around 2-3 hours to complete. You should work on it in your own time and attend the scheduled lab class to get help and record your checkpoints.

The aims of the worksheet are:

- To become familiar with the NetLogo User Manual, in particular the Interface Guide, Programming Guide, and the Dictionary;
- To complete the three tutorials in the NetLogo User Manual;
- To learn the basic functions of the NetLogo user interface and how to create interfaces for your own models;
- To learn the essential concepts of NetLogo programming (agent types, code elements, and the basic organization of a program);
- To program, by following an example, a first simple model.

➢ Start up NetLogo (Start / Programs / NetLogo).

➢ From the Help menu, choose the NetLogo User Manual. This opens the NetLogo User Manual in a web browser.

➢ Work through the two Introduction sections in the User Manual: **What is NetLogo?** and **Sample Model: Party**. Have pen and paper at hand so that you can write down your answers to the questions in the Challenge and Thinking with models parts of the Sample Model: Party page.

➢ Now work through **Tutorial #1: Models**. Again, keep pen and paper handy so that you can write down your answers to the questions asked at various points in the tutorial. Make sure that you understand the various interface components found in a model: the setup and go buttons, sliders, switches, plots, monitors, how to control the View and adjust its settings, and how to open the models in the Model library.

➢ Explore the model library: open some of the models and try them out. The models in the **Sample Models** section are the most relevant ones for this module (except the **System Dynamics** section, which the module does not cover). Make sure to read the explanations

in the **Info** tab for the model, and try out some of the suggested activities. (However, it is best to postpone extending the models until you have learned a bit more about NetLogo programming.)

- Answer the self-check questions below. Write your answers in the space provided.

    - What are the four types of agents in NetLogo? Which types are mobile and which are not? For each type of agent, find a model in the library which uses agents of that type.

    - What is the shape of the "world" in the model you are currently looking at? Is it a square, a cylinder, or a torus? Where is its origin? (Hint: look at the model **settings** using the button on the Interface tab.) Make a note of which model your answer refers to.

CHECKPOINT [TUTORIAL 1]

Show the demonstrator your answers to the self-check questions above.

- Work through **Tutorial 2: Commands** in the NetLogo User Manual. This tutorial shows you how to use the **Command Center** in NetLogo's **Interface** tab to manipulate models directly. You will learn how to inspect the colours and shapes of agents and how to manipulate these.

- With the **Traffic Basic** model open, answer the self-check questions below:

    - What is the *who number* of the red car? This car is being "watched" – how can you choose a different car to watch? What is the difference between "watching" and "following" an agent? [You may need to consult the NetLogo Dictionary.]

    - The command `show shapes` can be used to see all the turtle shapes available to the model. How can you make all the turtles look like trucks? How can you make just one turtle look like a green airplane?

- Now work through **Tutorial 3: Procedures**. This teaches you the basics of using the Code tab to program your own models. Work through all the steps of the tutorial carefully. **Do not be tempted to take shortcuts by copy-and-pasting the solution into NetLogo**. When you have finished, try out the self test exercises below:

    - In the real world, animals have a limited life-span, whereas in the Tutorial 3 model it is possible for a turtle to remain alive indefinitely if it continues to consume grass. Save a copy of your model, and then make changes so that turtles die when they reach the age of 100. You will probably need a new turtle variable to keep track of each turtle's age.

    - Change your model so that each turtle has an *individual* life-span, randomly chosen to be between 70 and 100. The life-span is set when the turtle is created or hatched. As before, a turtle dies when it reaches its life span.

    - Introduce two sliders to set variables max-life-span and min-life-span (with the obvious meaning). Change your model so that an individual turtle's life span is randomly chosen to be between these two values.

CHECKPOINT [TUTORIALS 2-3]

Show the demonstrator your final model with the changes listed above.

The final exercise is based on the "mushroom hunter" example described in Chapters 1 and 2 of Railsback and Grimm. Consult Chapter 2 if you need to see detailed step-by-step instructions for this exercise. The instructions in this sheet describe what to aim for and give snippets of code for achieving various effects.

The Mushroom Hunter model is a model of foragers hunting for mushrooms in a forest. The forest is represented abstractly, as a world of black patches. Mushrooms are represented as red patches and they grow in clusters.

There are two hunters in the world, represented by turtles. At each time step each hunter moves forward one step and checks if he/she has found a mushroom (a red patch). Each hunter remembers how long it has been since he/she last found a mushroom. You will need to use a turtle variable for this. This variable should be initialized to a large value (say 999). If a hunter finds a mushroom during the current time step the variable is set to 0, otherwise it is incremented by one.

If the hunter has found a mushroom recently (say, within the last 20 time steps) he/she will continue to search in the same area, otherwise he/she will begin to search further afield. You can use the pen-down command to track the paths taken by the hunters as they search.

How can you create clusters of mushrooms? Introduce a global variable (call it num-clusters) to store the number of mushrooms. You can introduce this variable via a slider in the interface tab, or as a global variable in the code tab – the choice is yours. See the Programming Guide in the NetLogo Manual to find out how to introduce global variables. If using a global variable you should initialize it somewhere in your setup procedure.

To create a cluster of mushrooms, choose num-clusters patches to act as the centre of a cluster. Each centre patch will then ask 20 patches that are close to it to turn red. Here is a code snippet for achieving this:

```
ask n-of num-clusters patches        ;; randomly chose num-clusters centre patches
[
  ask n-of 20 patches in-radius 5    ;; each patch asks 20 nearby patches to go red
  [
    set pcolor red
  ]
]
```

How should you program the hunter's search strategy? The book suggests that to get the hunters to continue searching in the same area, they should turn by a large angle and then move forward. Otherwise they should turn by a small angle and then more forward. The code below achieves this, though you may be able to improve on this. [**Note**: this code must be executed within a **turtle context** because it contains commands to be carried out by a single turtle. This means that the code must be enclosed within some other construct such as ask turtles [...] which creates a turtle context.]

```
ifelse time-since-last-found <= 20  ;; a mushroom was found recently
  [right (random 181) - 90]         ;; make a sharp turn to stay in the same area
  [right (random 21) - 10]          ;; make a shallow turn to move out of the area
forward 1
```

CHECKPOINT [MUSHROOM HUNTERS]

Show the demonstrator your working Mushroom Hunter model.