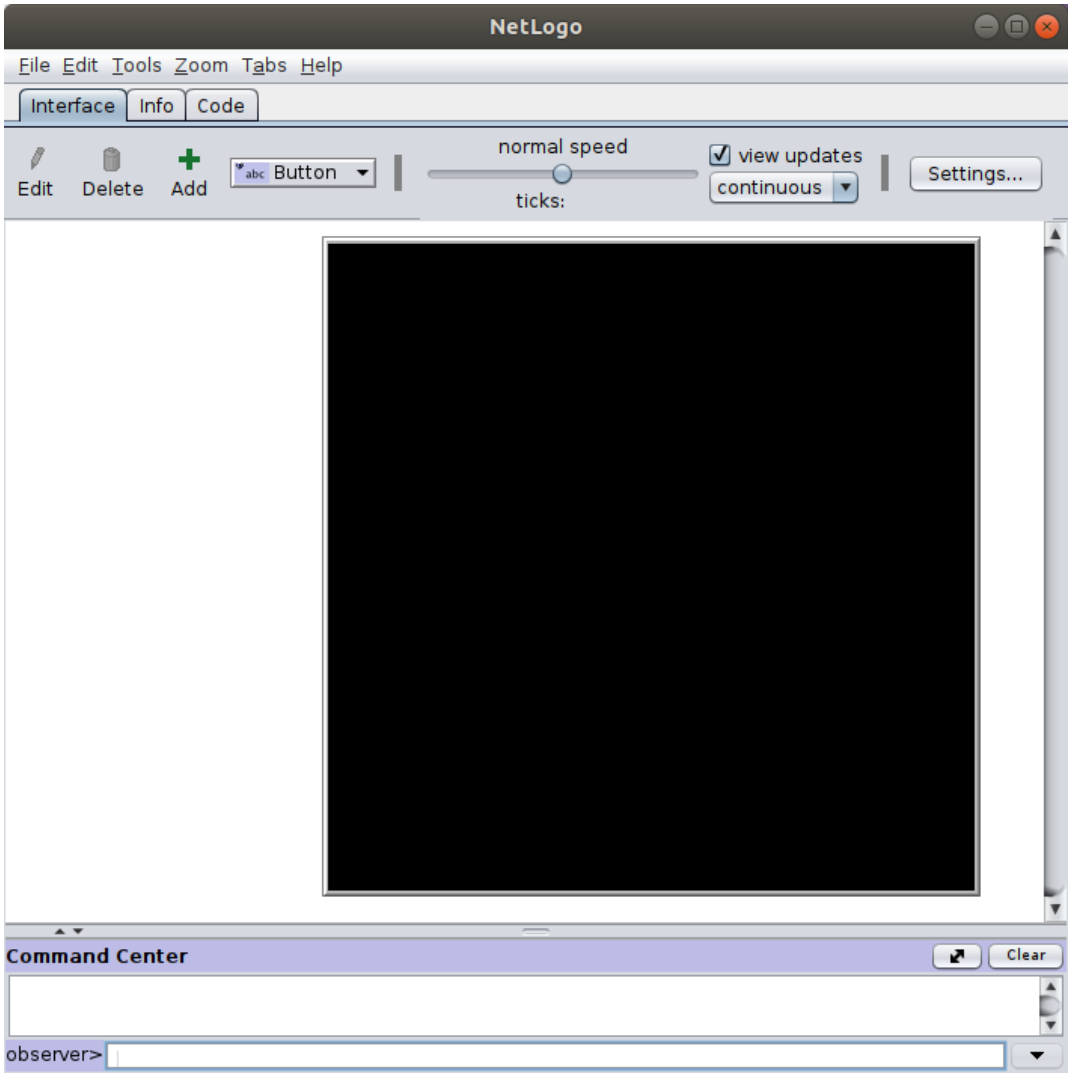
Agent Technology Practical - Lab Assignment 2
Reproduction

Report requirements

- Your report must be one .pdf file made with LaTeX. A report template can be found on Nestor.
- Since this is a coding assignment, you must hand in both a report and your NetLogo code (which should be one .nlogo file). You can upload these using the 'browse my computer' button on this assignment's upload page on Nestor.
- The names of your .pdf and .nlogo files must include the name of this course (which you may abbreviate as 'atp'), your student number(s), the name of the lab assignment, and your lab session group number. (e.g. 'atp_s1234567_s8765432_lab1_group1.pdf')
- The first page in your .pdf file must include the same information again: your student number(s), the name of this course, the name of the lab assignment, and your lab session group number.
- Reports must be in English.
- Clearly label each question you are answering.
- You may work in pairs of two students. If you do, only one of you has to hand in the lab assignment, and clearly indicate who you are working with in your report.
- **VERY IMPORTANT: For each subquestion, you must describe which model changes you made in your report (which lines of code you added, which lines you removed, which lines you modified and how, which GUI elements you added/modified, etc.) Give a short description of why your modification solves the issue. You will not receive points if something is *only* in your NetLogo code, and not in your report!**

Prerequisites

For this lab you will need NetLogo[3]. It can be downloaded at https://ccl.northwestern.edu/netlogo/download.shtml You don't have to fill in the form on the download page - clicking 'download' will work regardless. On Linux, you will have to extract the NetLogo archive and run the NetLogo executable inside. On Windows, you will have to run the installer. Once you have NetLogo launched, it should look something like this:



Before you continue, we recommend completing 'Tutorial #3: Procedures' (leftmost column, under 'Learning NetLogo'): https://ccl.northwestern.edu/netlogo/docs/. During this lab, the NetLogo Dictionary found on the same page (leftmost column, under 'Reference') may also prove useful. You can use ctrl + f to search in it.

Reproduction

In 1940, John von Neumann, a now-famous Hungarian polymath, designed what he called a 'self-reproducing automaton'. His work was published after his death in 1966 in the book '*Theory of Self-Reproducing Automata*'.[2] A self-reproducing automaton is a 'machine' that can make replicates of itself using its environment.

While we have plenty examples of biological self-reproducing automata, and some examples of digital self-reproducing automata, such as some computer viruses, and replicators in John Conway's Game of Life[1], only highly restricted real-life examples exist, which need to be fed with pre-built parts.

In 1981, Timothy J. Healy, in his '*Machine intelligence and communications in future NASA missions*' suggested that real-life self-reproducing automata could one day be used to mine the materials of asteroids, moons, or planets. However, we currently do not have the technical skills to try this out in real life, and even if we did, we may not want to, because of the costs and possible risks.

In this lab, you are going to make a model of a self-replicating space probe 'eating' the moon. You don't have to do

this from scratch! In fact, this model can be created by modifying the 'Wolf Sheep Simple 4' model found at File - Models Library - IABM Textbook - chapter 4 - Wolf Sheep Simple 4, so open that model now. Note that some of the modifications have to be made in the code, while other modifications only require you to touch the GUI. Before you continue, make sure you are using **Wolf Sheep Simple 4**, and not a different model.

Points possible: 21

Question 1: Graphical modifications

You need to make two graphical modifications. Make sure your modifications still function after setting up and running the model:
a) 1 pt. The sheep shouldn't look like sheep, but like something that could reasonably represent a space probe. You can see all available shapes at Tools - Turtle Shapes Editor. Your modifications for this subquestion should be done in the model code.

b) 1 pt. Secondly, the moon is gray, not green, so you need to change the colour of the patches that represent grass.

Question 2: Environmental modifications

a) 1 pt. Make sure the moon doesn't 'regrow'. Technically it gathers mass through e.g. meteor impacts, but for this modification we will assume that any changes in lunar mass are not significant enough to be modelled.

b) 1 pt. The moon's density isn't random. For now, let's assume the moon has a fixed density, and change the model's set-up to reflect this.

c) 2 pts. The moon is more or less round, not a square. Make sure the moon is a non-hollow gray circle with a radius in the range of 10 to 15. Recall that the formula for a circle is $x^2 + y^2 = r^2$. `ifelse` could be useful for this. If done correctly, you should see a gray moon on a black background after setup.

d) 1 pt. Change the world so it doesn't wrap around - we aren't investigating teleportation.

Question 3: GUI modifications

a) 1 pt. If we want to investigate the difference between normal mining and mining through self-reproduction we need to be able to toggle it on or off. Create a switch for toggling self-reproduction, but don't modify the code yet. **Note: when creating a switch, a new global variable will be created with the same name.**

Also set the initial number of space probes to 1, as it is easier to understand the model's behaviour if it begins with only 1 space probe. What do you see when running the model with exactly 1 space probe?

Question 4: Agent modifications

a) 1 pt. First of all, let's assume our space probes run on solar energy and can keep going forever. Change the code so they never 'die'.

b) 2 pts. A sheep won't eat grass when there is less 'energy' in the grass than the amount of energy they gain from eating. Make sure the probe always eats the moon and gains the correct amount of energy, except when the 'energy' of a piece of moon is at 0 or lower.

c) 3 pts. We probably won't want to use autonomous space probes that move around randomly. Instead, have the space probes move to *any* piece of non-eaten moon. `one-of` and `with` will be useful for this issue.

If you make space probes change their direction on each step, you may notice that only the center of the moon will get eaten, turning it into a giant donut. This can be solved by making space probes 'remember' their current target, and only change their target when it has been eaten. So, implement the following rules: if you have no target (`is-patch?` may be useful), set a random non-eaten piece of moon as target. If you have a target, check whether it has been fully eaten. If it has been fully eaten, change your target. If it has not been fully eaten, keep heading towards your target.

You may encounter an error when the moon is completely eaten, but ignore it for now.

d) 2 pts. Now that your space probes move towards pieces of moon, have them move towards the *nearest* piece of non-eaten moon at each step. One way to do this uses `min-one-of`, `distance`, and `myself`.

After running the model with this modification, you may encounter an error when no more pieces of moon exist. Make sure this error never occurs. It can be solved with tests such as `is-number?`, `any?`, and `is-patch?`.

e) 1 pt. Now, change the agent's code such that the self-reproduction toggle button actually works.
Don't worry if you only see one probe. This may be because probes can occupy the same location. Instead, look at the population graph to see how many there are.

Question 5: Model run modifications

a) 1 pt. Modify the code such that the model stops running when there is no more moon left. `stop` could be used for this.

Question 6: Additional modifications - 3 pts.

You should now have a functioning model of celestial mining using self-replicating space probes. There are still a lot of modifications possible, but you won't have to do all of them. Each of the next set of modifications has a number of 'work points' associated with them. Pick a set of modifications such that you have exactly three 'work points', implement them, and include them in your report the same way you did with the other modifications. We will not grade modifications beyond the ones needed for three work points (e.g. if you submit **c**, d, and **e**, in that order, we will ignore **e**.)

a) A slider which determines the size of the moon on set-up.
Work points: 1

b) A graph that shows the total amount of 'moon' (energy) collected as a function of time (in ticks). Make sure that space probes only lose energy when making new space probes, and not when moving!
Work points: 1

c) A slider that changes the amount of energy needed for a space probe to reproduce.
Work points: 1

d) Two sliders that change the location of the moon on set-up. Recall that the equation of a circle is $(x-a)^2+(y-b)^2=r^2$, where r is the radius and (a,b) are the circle's center coordinates. Make sure you can move the center of the moon to any location in the environment.
Work points: 2

e) A switch that allows self-replicating probes to 'eat' each other, gaining the eaten probe's energy. Make sure probes can only eat other probes, and not themselves. e.g. '`who of myself`' can be used for this. To test if it works, you can setup the model with one space probe, and use the command '`inspect a-sheep 0`' (if you haven't done any renaming).
Work points: 2

f) A switch that determines the shape of the moon on set-up (circle or ellipse). If you didn't choose **6a**, make sure the moon is twice as 'wide' as it is 'tall' when it is an ellipse. If you did choose **6a**, use the moon's variable size to determine the ellipse's width, and add a new slider to determine its height.
Work points: 2

g) Make the probes' colour a function of the energy they currently have. Make sure that their colour scales to the energy needed to reproduce.
Work points: 2

h) A switch that causes space probes not to target patches that are already being targeted by another space probe. See also question **4c**. You may need '`ask`' for this. If it works, you should see a difference in behaviour when the moon is almost completely eaten. Make sure there is no error when all of the moon has been eaten. If there is, you could use e.g. '`is-patch?`' or '`any?`' to solve it.
Work points: 2

References

[1] Martin Gardner. Mathematical games - the fantastic combinations of john conway's new solitaire game 'life'. *Scientific American*, 223:120–123, 1970.

[2] John Von Neumann and Arthur W. Burks. *Theory of Self-Reproducing Automata*. University of Illinois Press, Urbana, 1966.

[3] U. Wilensky. *NetLogo*. Northwestern University, Center for Connected Learning and Computer-Based Modeling, Evanston, IL., 1999. `http://ccl.northwestern.edu/netlogo/`.