**Exercises week 1 (9-13 January)**

**Name:** \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

The aim for this week’s exercises is to get familiar with the NetLogo IDE and practice with its basic syntax and concepts. For this, you will create a model that simulates a school of fish. Make sure to watch the videos on Canvas before you begin with the exercises.

Many species of fish, like herrings, can form very large groups in the water, called schools. These schools can take on complex patterns, for example to escape from predators or reduce energy expenditure. Fish however are most likely not aware of the school as a whole, nor do they actively try to create patterns like the one in the picture. Instead, each individual fish is thought to react only to other fish that are nearby, using a set of simple rules. The complex patterns that are formed by the group arise from the combination of the relatively simple behaviour of the individuals and their environment.

In the following exercises you will make a model that aims to reproduce some of these patterns using only a few simple rules for the agents. Then you will implement some ways to measure the grouping behaviour of agents and you will analyse the results.

Read carefully and answer the questions in a concise manner underneath the exercises in this document where asked. Add code and images where requested or instrumental to your answer. Upload this document with your answers to Canvas before **January 13th 17:00**. When submitting the exercises, also upload your final NetLogo model.

# Basic model of moving turtles

In this question we will build the basic NetLogo model of swimming fish that you will be using for the rest of the exercises.

**1.1a** Start by creating a new model in NetLogo. Set the world to wrap in both dimensions and adjust the size of the world to 50 by 50 (note that the middle coordinate is 0). Set the patch-size such that the world comfortably fits the screen of your computer.

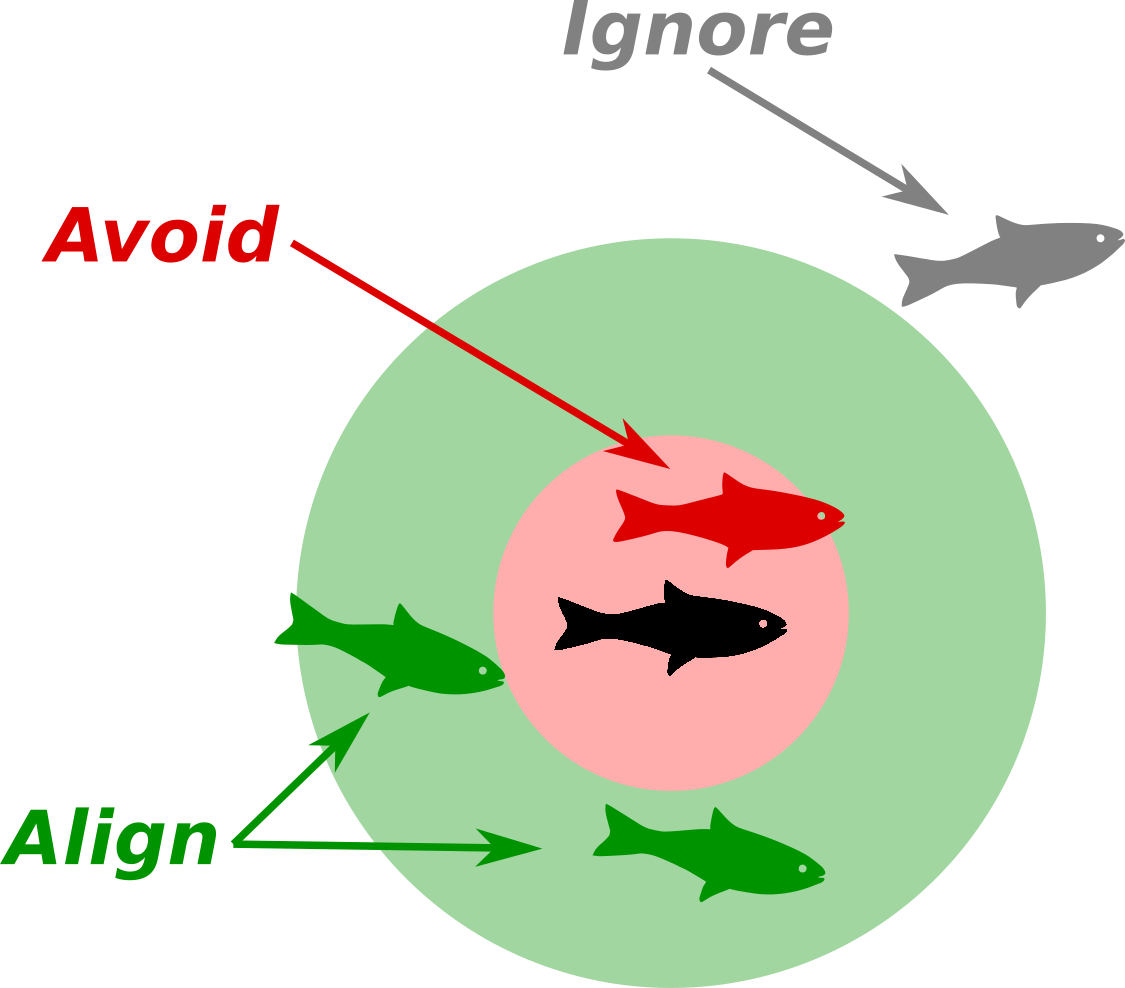
**1.1b** Now you will create a basic interface for the model. Add two buttons, one activating the function ‘setup’ and one that continuously activates the function ‘go’. Also add a slider with the global variable ‘population’ with a range between 0 and 200 and increments of 1.

**1.1c** Write the setup function in the code tab. In the setup you first reset the world and the ticks. Then you create turtles where the amount is determined by the setting of the ‘population’ slider. Also set the colour of the turtles to black, set their size to 1.5, set their shape to “fish”, and assign them a random location. As the last part of the setup function, colour all patches light-blue by setting the property pcolor to 86 (see *menu → tools → color swatches* for a list of all colours).

**1.1d** Write a simple go function that asks each fish to go forward by 0.2. Use the dictionary to lookup the function to go forward. At the end of the go function, increase the number of ticks.

Well done, you have made your first simple but functioning NetLogo model! Try it out with different amounts of fish. You may need to adjust the speed at which the model runs to see what is happening.

# Social behaviour: Align and avoid

As you saw, the turtles swim around and do not react to others. As this course is aimed at simulating social interactions, let’s add some behaviour to the turtles that takes into account their neighbours. Specifically, you will implement two rules: The first rule is that a fish avoids another fish if it gets too close. In the figure the black fish will avoid others in the red circle. If it does not need to avoid, then the second rule applies: it will align with others that are nearby. This means that the black center fish in the figure wants to swim in the same direction as others inside the green circle. If there is no-one nearby it will not change its behaviour, but continue to swim in a straight line.

**1.2a** Start by adding two new properties to the turtles called ‘schoolmates’ and ‘nearestNeighbour’. Also add a function called schooling, that does not take any parameters and does not return anything. Call this function in the go function, just before the line that makes the turtle go forward, so it can adjust its heading depending on where the other turtles are.

**1.2b** Implement the schooling function.

* First set the value of the turtle’s property ‘schoolmates’ to the neighbours of the turtle. To select the neighbours, you can use other turtles in-radius 5.
* Check if there are any schoolmates (look up any? in the dictionary). If this is the case do the following things:
  + Set the value of the nearestNeighbour property to the closest turtle in the group of turtles stored in the schoolmates property. You can use min-one-of schoolmates [distance myself] to select the neighbour with the smallest distance to the turtle.
  + Now we check which rule should be applied. If distance nearestNeighbour is smaller than 1, the turtle should avoid the nearest neighbour. You can do this by first facing the nearestNeighbour and then turning around 180 degrees (lookup face and rt in the dictionary).
  + If this is not the case, align the turtle with the averageHeading of the schoolmates. For this you can use: rt subtract-headings averageHeading schoolmates heading. subtract-headings is a standard function and heading is a property of the turtle (see dictionary). averageHeading is a function that we still need to define, which as the name suggests, determines the average heading of a group of turtles in degrees, in this case of the schoolmates.

to schooling

set schoolmates other turtles in-radius 5

if any? schoolmates

[

set nearestNeighbor min-one-of schoolmates [distance myself]

ifelse distance nearestNeighbor < 1

[

face nearestNeighbor

rt 180

]

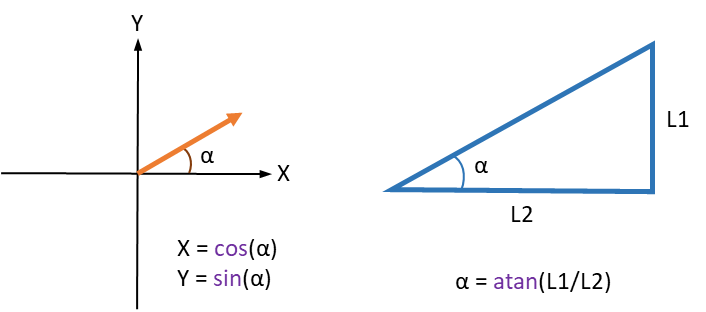
[

rt subtract-headings averageHeading schoolmates heading

]

]

end

**1.2c** Implement the averageHeading function. Normally the direction of a turtle in NetLogo is expressed in degrees. However, if you would take the average of 359 degrees and 1 degree for example this would give 180 degrees, which is not the average heading of both turtles. It should be 0. To solve this, we first need to convert the heading of a turtle to an x and y component. Then you can calculate the average x and y component of the turtles. At the end of the function this is converted back to degrees using the function atan. The code for the function is given below. Make sure that you understand what happens in this function, including the type of function, the definition of local variables and the way properties of turtles are accessed using the keyword of.

to-report averageHeading [ neighbours ]

let meanXComp mean [cos heading] of neighbours

let meanYComp mean [sin heading] of neighbours

report atan meanYComp meanXComp

end

**1.2d** You have now implemented basic schooling behaviour based on two simple rules. Run the model with a population of 30 fish and a population of 170 fish. Report what you observe in few sentences below.

Not sure if this is somewhat of a trick question, I seem to either not be skilled enough with the slider, or it is just impossible to set the fish slider to those two values.

Nevertheless, for the value of 171(to replace 170), the fish seem to move way slower, and also the “ticks” counter grows slower, so I can deduct that the population size is critical to the result of experiments. Additionally, this should also indicate that the size we choose for the world will also affect the behaviour of the agents.

# Measure alignment

In the simple schooling model you made in the previous exercises, the agents apply two rules: avoid the nearest neighbour if it is too close, else align with average direction of its neighbours. This resulted in an aligned school after some time. The number of ticks this takes differs between runs, but looking at the counter at the top it seems to be between a hundred to a couple of hundred ticks.

While such a quick estimation is great to get a feeling for the model, to really judge the effect of parameter changes or additional rules we first need to add quantitative metrics that can be compared objectively.

Let’s pick a metric for alignment: we define lonelyTurtles as the percentage of turtles who have no schoolmates (yet).

**1.3a** Add a turtle property called lonely?. Ensure that this property is set to True for each turtle upon setup. Check in the schooling function if there are any schoolmates. If so, lonely? should be False, otherwise it should remain or be set back to True. Also add a function that returns the percentage of lonely turtles.

to-report lonelyTurtles

let all count turtles

let lonely count turtles with [lonely? = true]

report (lonely / all) \* 100

end

**1.3b** Before using your lonelyTurtles function, write down what you expect to happen regarding the percentage of lonely turtles over time, and why.

Now let’s add a quick way to track the percentage of lonely turtles visually during the simulation: a monitor. Go to the *interface* tab and add a monitor. For the reporter fill in the name of the lonelyTurtles function and click OK. Then run the model a couple of times. Do you observe what you expected? If not, what is different? Your answer should include both your expectations and how your observations relate to them.

I expect the percentage to start at 100 when pressing setup, but when we press go the number should immediately change to the real value since the go function calls the schooling function which sets the true value of lonely? For the fishes. Also, the percentage of lonely fishes should decline as the schooling takes effect until it reaches 0, when all fishes have successfully joined the school.

And actually my expectations where completely correct, I was unsure of my first statement but in deed the value will be wrong until the go function is executed.

**1.3c** Another way to analyse the model in real-time is by adding a plot. Go to the interface tab and add a plot component. Give the plot a fitting name and axes labels. Then change the pen update command to plot followed by the name of your lonelyTurtles function. The result of this function will be drawn each tick as a line point in the plot. Now you have a plot, change the population size. Does this affect the percentage of lonely turtles? How? Also paste a copy of the plots to illustrate your answer (right-click on the plot -> copy image).

What I notice is that the lower the population size, the more ticks it takes for the percentage of lonely fish to decrease.

With a low population number:



With a high population number:



We can see in the last graph that it took less than 300 ticks to reach a near 0 lonely fish rate, but in the first graph at over 1.2k ticks the percentage was still around 5%

# 1.4 Add attraction behaviour

The avoid and align rules make an agent join a school when it happens to come close to it, but it does not actively pursue company. Let’s say we know the simulated fish prefers to swim in groups. Then it should go toward a group if it is alone and it sees another fish in the distance. For this we will add attraction as a behaviour (see figure on the right).

More formally: if there are no other turtles within the align distance but there are turtles in the maximum view distance, the agent should adjust its heading towards the nearest of these agents.

**1.4a** Implement the attract rule as described above. Paste the adapted schooling function below.

to schooling

set schoolmates other turtles in-radius 5

ifelse any? schoolmates

[

set lonely? false

set nearestNeighbor min-one-of schoolmates [distance myself]

ifelse distance nearestNeighbor < 1

[

face nearestNeighbor

rt 180

]

[

rt subtract-headings averageHeading schoolmates heading

]

]

;This else part creates an array of fishes that are within the view distance and sets the neighbor to be the closes of them, and then the fish faces it.

[

let closefish other turtles in-radius viewDistance

if any? closeFish

[

let nearestFish min-one-of closeFish [distance myself]

face nearestFish

]

]

end

**1.4b** Does this rule decrease the percentage of lonely fish? Explain how you got to this conclusion and include plots to support it. Base your conclusions on multiple simulations for both a population size of 30 and 170 fish and let each model run for 100 ticks. Set the view distance to 20.

*hint:* add “if ticks = 100 [stop]” to the go function to stop the simulation at exactly 100 ticks

Yes it does, The conclusion was reached after comparing the results and plots of running the experiment before applying the attraction rule (see plots in 1.3c) and after. In both cases, the ratio of lonely fish decreases dramatically faster than before:

With low population



With a high population



# 1.5 Sub-groups

So far, the fish only cared about swimming with their fellow fish without a care in the world. In this exercise, we will add a predator that the fish will try to avoid.

**1.5a** We will now create two separate breeds of fish: herring and sharks. Add the following two lines to the very top of your code (above the turtles-own):

breed [ herring one-herring ]  
breed [ sharks shark ]

Up until this point we had only the default breed, turtles. In the setup function, change *create-turtles* to *create-herring*: this will ensure that you still create the same fish, but they are now specified to be herring (‘turtles’ are all breeds combined, so also sharks).

In the setup function, add a function similar to create-herring to create a single shark. That shark should also have shape “fish”, be red and of size 7, and start at the centre coordinate (0,0).

Make sure that that only herring are asked to school (*go* function), and while schooling the herring should only look at other herring for finding schoolmates or fish within view distance (*schooling* function). Also the *lonelyTurtles* function should look at herring only. For every tick, the shark will move 0.2 forward and   
do nothing else.  **1.5b** Generally, the herring will want to avoid running into the shark. However, there are reckless herring that don’t bother changing their direction. Add a new herring property called ‘*recklessness*’. Also add a slider that determines the probability of a herring being reckless (min: 0, max: 1, increment 0.1), with the name *recklessProbability*. In the setup function, ensure that each herring has a *recklessProbability* chance of havingthe property *recklessness* set to 1 (reckless) or otherwise 0 (not reckless). Lookup random in the NetLogo dictionary to find out how to do this. Paste your setup function below.

create-herring population

[

set color black

set size 1.5

set shape "fish"

set xcor random-xcor

set ycor random-ycor

set lonely? true

set viewDistance 20

set recklessness random 2

]

**1.5c** Let’s make it a little easier to see how reckless a herring is by changing its colour. Define a local list with the colours for both levels. You can choose the colours of the levels yourself. Set herring colour using the index of a list. Paste your modified setup function below.

create-herring population

[

set size 1.5

set shape "fish"

set xcor random-xcor

set ycor random-ycor

set lonely? true

set viewDistance 20

set recklessness random 2

let colors list (67)(25)

ifelse recklessness = 0

[

set color item 0 colors

]

[

set color item 1 colors

]

]

**1.5d** Now that we have different characters, we can add the preferential behaviour. Adapt the schooling function such that each herring uses the first rule of the list below that applies to it, while ensuring that herring are lonely unless they are part of a school:

1. Avoid the nearest neighbour if it is too close  
2. Non-reckless herring: avoid the shark if it is within view distance  
3. Non-reckless herring: align with other non-reckless herring if there are any within the align distance  
4. Non-reckless herring: adjust the heading towards the nearest non-reckless herring within view distance  
5. Align with any herring within align distance  
6. Adjust heading towards the nearest herring within view distance

*Hint: you can write this out as a tree of nested IFELSE-decisions, but this may be a little difficult to puzzle out. Alternatively, you can use a series of IF statements, one for each option. Add STOP as the last command of each IF statement. This stops the schooling function when an IF statement applies and prevents following statements from being executed.*

Test with a population of 100 and set *percReckless* to 100%, now run the model a few times. What do you observe? Also paste your schooling function below.

to schooling

set schoolmates other herring in-radius 5

;neighbors check

ifelse any? schoolmates

[

set lonely? false

set nearestNeighbor min-one-of schoolmates [distance myself]

;(closeness check) if yes then avoid other herring

ifelse distance nearestNeighbor < 1

[

face nearestNeighbor

rt 180

]

;(closeness check)if not then check if reckless for further instructions

[

;(recklessness check) if yes then avoid shark, align or face.

ifelse recklessness = 0

[

;(shark check) if yes then avoid shark

ifelse any? sharks in-radius viewDistance

[

let sharksNear sharks in-radius viewDistance

let nearestShark min-one-of sharksNear [distance myself]

face nearestShark

rt 180

]

;(shark check)if not then try to align or face towards other non reckless fish

[

;(close non-reckless check)if yes then align with non reckless herring in align distance

ifelse any? schoolmates with [recklessness = 0]

[

let chillFish schoolmates with [recklessness = 0]

rt subtract-headings averageHeading chillFish heading

]

;(close non-reckless check)if not then face towards non reckless fish in view distance

[

let chillFarFish other herring with [recklessness = 0] in-radius viewDistance

]

]

]

[

;This line is for rule 5 in 1.5d, reckless fish wont avoid sharks and will try to align with close fish

rt subtract-headings averageHeading schoolmates heading

]

]

]

;(neighbors check)This else part creates an array of fishes that are within the view distance and sets the neighbor to be the closes of them, and then the fish faces it.

[

let closefish other herring in-radius viewDistance

if any? closeFish

[

let nearestFish min-one-of closeFish [distance myself]

face nearestFish

]

]

end

# 1.6 Communication

We now have fish with different characters, where some prefer to school with similar fish. Fish observe the recklessness of other fish, however, the fish are not yet communicating. In this exercise, we will create fish that make other fish more or less reckless.

**1.6a** We start with creating one fish that is very persuasive: dreaming of big adventures, he makes other fish he encounters join him in his recklessness. Create a herring-own variable called ‘*persuasive?*’. This will determine for each herring if that herring is persuasive (true) or not (false). Use the setup function to ensure that all herring but one are not persuasive. Paste your setup function below.

create-herring 1

[

set size 1.5

set shape "fish"

set xcor random-xcor

set ycor random-ycor

set lonely? true

set viewDistance 20

set persuasive? true

set recklessness 1

let colors list (67)(25)

ifelse recklessness = 0

[

set color item 0 colors

]

[

set color item 1 colors

]

]

**1.6b** Now let us build a function called ‘*communicate*’. In this function, the herring checks whether there are any herring nearby (*schoolmates*) that are persuasive. If this is the case, the herring will change its level of recklessness to that of (one of) the persuader(s). Do not forget to also update the colour of the herring. Paste your communicate function below.

to communicate

if any? schoolmates with [persuasive? = true]

[

let rizzMasters schoolmates with [persuasive? = true]

let rizzKing one-of rizzMasters

set color [color] of rizzKing

set recklessness [recklessness] of rizzKing

;set persuasive? [persuasive?] of rizzKing (this line explains why not all fishes turn reckless with the original function)

]

end

**1.6c** Now we have to call the communication function in the right place in the schooling function. Ensure that the communication takes place only for the fish that are in a school (align) and are not persuasive themselves. Observe what happens if you run the model a few times. Is the persuader able to make every herring join him in his level of recklessness? Explain why (not)? Also paste the code of your schooling function below.

to schooling

set schoolmates other herring in-radius 5

;neighbors check

ifelse any? schoolmates

[

set lonely? false

set nearestNeighbor min-one-of schoolmates [distance myself]

if not persuasive?

[

communicate

]

;(closeness check) if yes then avoid other herring

ifelse distance nearestNeighbor < 1

[

face nearestNeighbor

rt 180

]

;(closeness check)if not then check if reckless for further instructions

[

;(recklessness check) if yes then avoid shark, align or face.

ifelse recklessness = 0

[

;(shark check) if yes then avoid shark

ifelse any? sharks in-radius viewDistance

[

let sharksNear sharks in-radius viewDistance

let nearestShark min-one-of sharksNear [distance myself]

face nearestShark

rt 180

]

;(shark check)if not then try to align or face towards other non reckless fish

[

;(close non-reckless check)if yes then align with non reckless herring in align distance

ifelse any? schoolmates with [recklessness = 0]

[

let chillFish schoolmates with [recklessness = 0]

rt subtract-headings averageHeading chillFish heading

]

;(close non-reckless check)if not then face towards non reckless fish in view distance

[

let chillFarFish other herring with [recklessness = 0] in-radius viewDistance

]

]

]

While the persuasive herring is able to convince some of his peers, he is not able to convince the whole group. This is because of the nature of the schooling technique, the fishes basically surround themselves with other fishes, and since only the recklessness and the color is passed and not persuasiveness, the “recklessness infection” can only possibly be carried out to some extent, at some point, the last ring of reckless fish will not be persuasive, and the persuasive ones in the inside of that ring are not in range for the other fish to communicate.

By modifying the function to also transmit the persuasive variable, the herring in deed was able to convince everyone, which proved my hypothesis.

1.7 Communication

**1.7a** So far we had a lone persuader. But what if the persuasive fish is of a particular breed? We will now create two separate breeds of herring: persuaders and non-persuaders. Replace breed [ herring one-herring ] at the very top of your code with the following two lines:

breed [ persuaders persuader ]  
breed [ non-persuaders non-persuader ]

In the Interface tab, add a slider called ‘percOfPersuaders’, ranging from 0 to 100 with increment set to 10, representing the percentage of persuaders in the total population.

Now, in the setup function, instead of creating general herring (with create-herring), use create-persuaders and create-non-persuaders. The number of herring of each breed to be created depends on the percOfPersuaders value. They are created in the same way, the only difference is the value of persuasive?.

In the schooling function, only ask the non-persuaders to communicate.

In the communicate function, let the communicating turtles inspect if there are any persuasive fish nearby (persuaders < 5). If there are, let each non-persuader change their level of recklessness to that of the persuader that is nearest to it (and change to the respective colour).

When submitting don’t forget to hand in your final NetLogo model. We will use this to check your implementation of this exercise.

*Hint: if you want to ask something from all breeds but one, you can use ‘ask turtles with [breed != …]’. You might want to use this in some of your functions.*

**1.7b** Now it is time for some experimenting. Change the percentage of persuaders, and measure how many ticks it takes for the percentage of lonely fish to get (and stay) below 10%. Use a spreadsheet to take the average of three measurements for each percentage of persuaders, and make a graph with the number of ticks on average against the percentage of persuaders. Paste your graph below and explain your findings.

*Hint: you can make the model stop automatically by adding some simple conditional logic to the ‘go’ function. IF percentage of lonely fish is below 10%, increase a global counter variable, ELSE reset the counter. IF the percentage of lonely fish was below 10% for 10 consecutive ticks, apply the command ‘stop’.*



Chart, line chart

Description automatically generated

Explanation:

The plot was formed by taking the average of 3 runs for each possible value of “Percentage of Fish that are Persuaders” (Range 0-100, with increments of 10) where the X axis represent this range, and the Y value represents how many ticks it took for the model variable “Percentage of Lonely Fish” to stay below 10% for 10 ticks.

The latter variable represents to the ratio of fish who have not yet joined a school of fish compared to the total population. I expected the average ticks to go down as the percentage of persuaders went up, but since the persuaders “Recklessness” variable is also decided randomly, the results are directly dependant on the pseudo randomness of netlogo, and finally, 3 runs seems to be too little to draw any conclusions.

1.8 Project proposal

By making the exercises above, you have already learned a lot about NetLogo and its capabilities. This week you will also choose a type of social behaviour for your project. More information about the available options can be found on Canvas.

**1.8a** What project environment did you choose? Write 2/3 sentences on why you chose this subject.

Environment: Competition

I chose competition mainly because I consider myself a very competitive person, but also because I wanted to study the nature of humanity in a survival scenario, because I hold that most humans are good because there is no need to be bad, but if the circumstances are dire, anyone is capable of doing immoral things.

**1.8b** Formulate a research question that you will answer with your NetLogo project. Explain how you will measure this with your model, i.e. what results do you need to get out of your model in order to answer the question?

**Competition for Human Survival**: Cooperation vs Competition of Humans in a Survival Scenario

**Question to answer**: Will humans ignore their moral standings in the face of potential/imminent death in a survival situation.

**NetLogo Model Setup**:

* + **World**
    - A world with limited resources that are generated in specific locations in the world, and these locations are also limited.
    - **World Variables**:
      * Resources
        + They regenerate but are limited in the sense that they not are regenerated fast enough to keep everyone healthy or even alive.

Camps will need to decide whether to attack other camp to keep their people alive, or individual people inside a camp might choose to kill their campmates for their resources and increase their chance of survival.

* + - * Survival Camps
        + Each camp has access to specific resources, creating the possibility of cooperation between camps, or competition by violence.
      * Season
        + As the seasons change some resources are more needed for survival than others
        + Each survival camp has access to specific resources.

The idea is that this introduces the probability of survivors switching camps, and a camp even deciding to attack another camp.

* + **Agents**
    - Humans that have a distinct set of variables for survival like strength, energy, morals and more.
    - **Agent Variables**
      * Health
        + Health will be a function of several things like age, energy and hungry.
      * Age
        + Randomly Decided
      * Sex
        + Randomly Decided
      * Strength
        + Function of Age, Sex and Health, also semi random within age and sex groups.
      * Energy
        + Function of Hungriness, affected by actions humans can take.
      * Hungriness
        + Everyone starts with the same level, goes up with time and goes down when they eat.
      * Resource Needs: They indicate what does the human will need in the enar future to keep being healthy or survive.
        + Food
        + Medicinal Herbs
        + Wood
        + Water
      * Moral Level
        + The range is randomly decided, and everyone starts with the same moral level.
        + After the beginning, it is a function of health.
        + The idea is that the lower the moral of the person, the higher the chance it will do “bad” things in order to survive.
      * Group
        + The survival group they current are part off.
      * Survival Camp
        + The survival camp they are currently living in
      * Resource Kit
        + The resources they currently own.

The idea is that people can decide to kill others for their resources.

* + - * Assigned Job
        + Each survival camp will have different ways of obtaining resources, and will require humans to actively get them, spending energy

**1.8c** You have seen and used many different functions of NetLogo. Think about if and how you could use them for your own model in the next few weeks. Name three different (built-in) functions that you could possibly apply to your project and explain in what way.

1. **to-report**: This will be key to reach a conclusion since this function allows to display and analyse data in several ways, and due to my models complexity, I need to examine continuously if I’m properly designing and balancing the relationships between all the agent variables so the model is a good representation of humans.
2. **ask**: Ill need the agents to do lots of things depending of certain scenarios, so this will be key too.
3. **create**: In order to study properly my research question I will create several types of humans to study things like what if there a big percentage of immoral people at the beginning? Will they be the ones to survive? Do very good people survive less? Do older people die first?