Summary

This walkthrough is meant to be used as a loose script and general resource to be shared as part of a standalone, 3-hour workshop at DHSI2018 on agent-based modelling. We use NetLogo as the platform for the illustration of agent-based modelling principles because it runs on Windows, Mac, and Linux, and it is both powerful while being simultaneously the easiest to get started with of the available platforms. If anyone in the workshop goes forward with agent-based modelling it may be necessary to swap to another platform for a feature that NetLogo doesn't have or even to write your own code if you need to maximize performance. However, even in such cases NetLogo will likely remain in your toolkit as a fast way to do some initial prototyping and/or to help bring other people in the agent-based modelling community.

From the NetLogo Home Page and the NetLogo FAQ the following should be noted:

1. NetLogo is a multi-agent programmable modelling environment. It is used by many tens of thousands of students, teachers and researchers worldwide. It also powers HubNet participatory simulations. It is authored by Uri Wilensky and developed at the CCL.
2. The "Logo" part is because NetLogo is a dialect of the Logo language. "Net" is meant to evoke the decentralized, interconnected nature of the phenomena you can model with NetLogo, including network phenomena.

The workshop is organized roughly as follows:

1. **Setup.** Ideally this is looked after by everyone in advance because it was requested in the welcome email but past history suggests that this will not be the case. People need to download NetLogo as soon as possible because the WiFi connection can be slow and the tool really is needed to enjoy the workshop. The install can be handled in parallel with the rest of the workshop. ~5 min
2. **Overview of Real-World Examples.** We'll ask everyone to introduce themselves and provide an example of an agent-based model that they have either heard about or used that they believe has some value. ~10 min
3. **Exploring Existing Models.** NetLogo includes a substantial library of pre-built models from a range of disciplines and we'll review about of 6 of these to get a handle on what agent-based models can do and how Netlogo works. Small challenges will be given throughout to generate familiarity with both agent-based modelling principles and the NetLogo interface and code. ~75 min
4. **Coding in NetLogo Primer.**
5. **Independent Work.** Workshop participants will self-divide into teams in order to either solve a set of model modification challenges drawn from the existing models already seen *or* extend the model introduced in the NetLogo primer. ~75 min

Setup

We'll use NetLogo 6.0.3 for the entire workshop. Recent, earlier versions should work for the workshop as well, but no promises.

> NetLogo Web is also available and is a version that runs entirely in a browser. It has a reduced feature set and is recommended only for showing models in cases where the desktop version may be unavailable. We won't be using it today but it should be remembered for the future. There is also a 3D version of NetLogo that is included in the base install. We'll just be looking at the 2D, top-down/map-view version today but the principles and practices are transferrable.

1. Download Netlogo 6.0.3 from https://ccl.northwestern.edu/netlogo/download.shtml. Note that filling in your information to get to the actual download page is optional.
2. Note the caveats for your operating system at the bottom of the page with the actual link to the download for your operating system. In particular, note what it says about Java and any potential install hiccups.
3. Begin installing NetLogo as you would any other program.
4. Out of the install process you'll typically end up with a folder called something like "NetLogo 6.0.3". Put this wherever you would like the NetLogo program to be kept. If you're not sure you want to keep using it *or* you want to be able to find it easily to play with then the Desktop is fine (easy to find to delete in the first case and easy to find and use in the second). *Do not separate out the contents of the folder.*
5. Open the folder and inside should be an application called "NetLogo 6.0.3". Run this and as long as it starts without complaint you should be good to go.

Overview of Real-World Examples

Start with asking participants why they are here, what uses of agent-based models have they heard about or seen? What led them to this workshop?

If an example is needed I can talk generally about attempting to develop better ambient NPCs for video games, modelling military conflicts, political models, etc.

Principles to illustrate

Declare these up front and then return to them over the course of the workshop to illustrate them

1. **Keep things simple.** We model to understand and the simpler the model the easier it is to understand. Yes, in the pursuit of explanatory power there is a balance to be struck between simplicity and reality. Simplicity is always to be preferred as the initial side to err on. Avoid the temptation to model reality: if we

could understand it we wouldn't need to model it! (this is only partially true, there might be value in running peturbed versions of reality.) Another way to put this, don't let your map even approach becoming the territory.
2. **Look, look, and look again.** Even simple models can easily have subtle yet interactions. It is really important to know your model from the code to the output and from the output to the code. To do this you'll need to look at your code and your output *a lot*, especially if you think you have something that is non-trivial.
3. **Models explain in ways that formulas do not.** There is very likely a mathematical formula or notation that can succinctly capture your model. While there is likely pressure to express your model (or a portion thereof) in pure notation this is something to be resisted. Converting to such notation often further simplifies what is already a simplification, resulting in a loss of information. Additionally, such notation can be pretty esoteric, making it inaccessible to anyone who isn't an expert. Let your code stand on its own whenever you can.

Rabbits Grass Weeds (Using the Interface & Info Panes)

Summary: *Rabbits Grass Weeds* is a sample model from biology that illustrates population cycles. It is also paradigmatic for what an agent-based model actually is and it is for this reason that we are looking at it first. We'll be able to quickly get a good handle on how the interface pane works and what sorts of information to expect in the info pane. We'll return to a model like this at the end of the workshop.

Loading: To load the model, from menu choose `File` > `Models Library`. A window should pop up with a list of folders and potential models to be chosen down the left side and a space for summary information on the right side. Navigate the folders on the left with `Sample Models` > `Biology` > `Rabbits Grass Weeds`.

While the Info pane has a description about what the model is meant to illustrate and how one of the nice things about agent-based models is that much can be figured out just be running the model. *Most* models in NetLogo will have a two step process for running the model and we'll follow this right now:

1. **setup**. There will likely be a button called "setup" or similar. Clicking this button will load whatever values the model is intended to run under as a default and usual change the central visualization pane from being empty to having initial content. Click "setup" now.
2. **go**. Models will typically also have a "go" button that sets the model in motion when clicked. There are two types of go-buttons. The most common is a "run forever" button that when clicked will run the model until a stop condition is met (which a model may not have). "Run forever" buttons are identified by the pair of arrows that each point to the tail of the other. Some models will also have "step once" buttons that perform

a single step of the simulation when clicked. Click “go” now to start the model running. “go” can be clicked again to stop the simulation.

In addition to the buttons there are sliders that can be used to control the simulation. Notice that these are stacked. The intent here is that the order of things to be done is top to bottom. So:

1. Stop the simulation by pressing “go” a second time.
2. Use the top slider to set the initial population size.
3. Setup and then start the simulation again.
4. Play with the bottom silders while the simulation is running to see the immediate effects *or* stop the simulation, set the sliders, setup the simulation, and then run it again.

What we are seeing here are really two types of agents interacting together to produce the simulation. The first type of agent are the rabbits and they are representative of what most people think of when they think about agent-based models. Each rabbit has its own set of properties, such as an energy level and location on the grid, and moves around on the basis of these properties, interacting with other agents as scripted to do so. Within this model these agents are represented as small rabbit icons but in the lingo of NetLogo these rabbits are really a type of turtle that has been assigned a specific icon. By default agents are triangles and called turtles because this is how the logo programming langugage that NetLogo shares a history with represented and referred to agents.

The second type of agent is really a NetLogo-specific agent type that is prebuilt with a core set of properties to ease the coding required in many models. This type of agent is the patch and they are best thought of as the cells within the grid. They are different from agents proper because they cannot move around but they can and often do have their own individual properties that drive interactions with other agents and surrounding patches. While patches *could* be created with turtle-type agents (or vice versa and we’ll see this shortly) having them predefined in such a way that locks then in place in relation to all other patches allows NetLogo to have built in commands for one patche to interact with other patches. This is a real boon.

> Spend a moment looking at the Info panel. In particular, give the suggestions in THINGS TO TRY a try and discuss what you find interesting with your group.

Lastly, open the Code panel. Just leave everything how it is (we’ll get to playing with code shortly) and simply note that there really isn’t a lot of code driving this simulation despite its ability to power to simulate hundreds of rabbits and 41 x 41 = 1,681 patches

Take Aways:

- Eventually you’ll be naming buttons and other components of the interface. Make sure that the names are descriptive and fit the intuitions of users.
- There are two broad classes of agent to consider when building a model in NetLogo: turtles and patches.

Turtles move around on a grid of patches and the patches remain locked in place while any turtles move across them.

Wealth Distribution (More Interface Control)

Summary: *Wealth Distribution* is a sample model from economics / sociology that illustrates how even conditions that seem like they would promote equality of wealth to a large degree can actually produce significant disparity.

Loading: To load the model, from menu choose `File` > `Models Library`. A window should pop up with a list of folders and potential models to be chosen down the left side and a space for summary information on the right side. Navigate the folders on the left with `Sample Models` > `Sociology` > `Wealth Distribution`.

Get a feel for this simulation by running it a few times and looking at the Info pane. Change the slider values both before the simulation starts and after.

> It is possible to crash the simulation by increasing the value of num-people while the simulation is running. Why might this happen?

Questions like “How is it that rich agents get rich in the first place?” and/or “How do they stay rich?” often come up with this model. Currently there isn’t much that we can do to figure this out because the information provided by the simulation is provided at the level of the *population* and we need to look at individuals. NetLogo has a helpful point-and-click interface for inspecting and watching agents within simulations.

To take advantage of NetLogo’s ability to inspect individual agents:

1. Pause the simulation.
2. Right-click on an agent that you would like to inspect. From the menu that pops up highlight the name of the turtle you would like to inspect and from the submenu choose to inspect that turtle.
3. A pop-up window will give you a close zoom of the area of the map that the turtle occupies at the top and a summary of that turtle’s properties at the bottom.
4. Note that the slider below the map-view can be used to zoom in or out. Also note that there is a “watch” button that will highlight the agent on the main map.
5. Turn on “watch” and restart the simulation.

Now we can watch the rich agent but things are moving pretty fast, too fast for us to figure out what is going on. We have two ways to address this:

1. Adjust the “speed slider” at the top of the interface pane.
2. Add a button that will only allow the simulation to move forward one step (or “tick”) each time it is pressed.

Try the first.

To add a step button let's begin by looking at the current "go" button by right-clicking on it and choosing "Edit…" The window that comes up shows all the settings that power this button. It tells us that the button operates from the perspective of the observer (that's us!) and that when clicked it will run forever. Further, it calls a function called "go" (from the code pane) and that the name displayed is also "go". With this information in hand we can create our own step button, as follows:

1. At the top of the interface pane there is a button called "Add" and immediately to the right of this button is a drop-down menu. Click this drop-down menu and choose "Button".
2. Somewhere in the interface pane (directly under "num-grain-grown"?) click with the mouse crosshairs to add a button of the default size.
3. From the window that pops up put "go" in the Commands box and "step" in the Display Name box.
4. Choose "OK".
5. Click the new step button to advance the simulation one step.

> As we watch a rich agent what is happening to their wealth and why? Note that at least some of this is a modelling choice. How might you model things differently to better align with your understanding of the world?
>
> Additionally, note that you can actually *change* the properties of any agent using the inspector. What might this be useful for?
>
> Share your ideas within your group.

Just because a model was built for one purpose doesn't mean that it can't be useful for another, either directly or with some adaptation. As an example, consider that someone might notice that the settings of the model seem to change the intensity of land use and it would be reasonable for them to want to see this intensity plotted. We can create a plot to do this by:

1. Returning to the drop-down menu that allows interface items to be added and choosing "plot".
2. Place the crosshairs to the right of the main visualization area and click. A plot of the default size will be placed (it can be moved later) and an input window will be automatically opened with some default values.
3. Change the name of the plot to "Occupied Land". Similarly change the axis labels.
4. What we want to plot is the number of patches with turtles divided by the total number of patches. The command to do this is:

   ```
   plot (count patches with [any? turtles-here] / count patches)
   ```

5. Click "ok" to exit and restart the simulation.

Participants should notice that the plot is basically a flat line at the bottom of the chart. This is because the scale of the Y axis is too large. The chart can be right-clicked on and the default, maximum Y axis value reset. 0.25 is likely a good value.

Take Aways:

- Individual turtles can be watched, inspected, and modified.
- New plots can be added to track information.

Termites (Reading Code and Adding a Seed)

Summary: *Termites* is a sample model from biology that illustrates how very simple instructions can lead to seemingly complicated—and even conscious—behaviour. The turtles in this models are “termites” and (roughly) follow two basic rules:

1. Walk around randomly until you bump into an object.
2. If you bump into an object and you are carrying an object then put that object down, otherwise pick up the object bumped into.

Loading: To load the model, from menu choose `File` > `Models Library`. A window should pop up with a list of folders and potential models to be chosen down the left side and a space for summary information on the right side. Navigate the folders on the left with `Sample Models` > `Biology` > `Termites`.

Get a feel for this simulation by running it a few times and looking at the Info pane. Change the slider values both before the simulation starts and after.

> Come to an agreement with your group what the rules of behaviour are for the termites simply by watching the simulation. Write these rules down.
>
> Next, open the code panel and read the code that is powering this simulation. Once this has been done work come to an agreement with your group about *how* the simulation actually works and write down the process in plain language.
>
> Once the agreement is written down transfer this description to the code panel using `;;` to insert comments.

At this point participants have likely noticed that their simulations are not producing the same output. Someone has likely said something like “Look at this!” and someone else has responded with something like “Mine isn’t doing that.” This is happening because the pseudo-random number generator within each implementation of NetLogo in the class is taking a different seed value. We can align all the simulations by passing them all the same seed value.

To do this we:

1. Create a slider on the interface attached to a variable called “seed” and setting the default range to be 0 to 100.
2. Modify the code from:

    ```
    to setup
        clear-all
        set-default-shape turtles "bug"
        ;; randomly distribute wood chips
        ask patches
        ...
    ```

    To:

    ```
    to setup
        random-seed seed
        clear-all
        set-default-shape turtles "bug"
        ;; randomly distribute wood chips
        ask patches
        ...
    ```

Now, assuming that the same version of NetLogo is used and the underlying architectures are similar enough (e.g. a 32bit processor may not produce the same results on the same seed as a 64bit processor) the results will be the same if the same seed is used.

> Under what situations would having the ability seed the pseudorandom number generator be useful? When would it not be useful.

Fortunately we can easily have the best of both worlds by adding a switch to the interface that will control whether or not the pseudorandom number generator will be seeded by the slider. We do this as follows:

1. Create a switch on the interface attached to a global variable called “use_seed”.
2. Modify the code from:

    ```
    to setup
        random-seed seed
        clear-all
        set-default-shape turtles "bug"
        ;; randomly distribute wood chips
        ask patches
        ...
    ```

To:

```
to setup
    if use_seed = TRUE [random-seed seed]
    clear-all
    set-default-shape turtles "bug"
    ;; randomly distribute wood chips
    ask patches
    ...
```

Note that it is possible to write “true” instead of “TRUE” and that both turn red to indicate that they are keywords in the NetLogo language. It is also possible to split the conditional statement over two lines, such as:

```
if use_seed = TRUE
[random-seed seed]
```

or something like:

```
if use_seed = TRUE [
    random-seed seed
]
```

NetLogo is (mostly?) whitespace insensitive so the code can be formatted with spaces and line breaks to make it easier to read.

Take Aways:

1. There is no substitute for reading the underlying code in terms of understanding how a simulation actually works. Fortunately, NetLogo code is pretty easy to read and understand. =)
2. We can replicate simulation runs by passing the pseudorandom number generator a seed value.

Life (Code editing)

Summary: *Life* (aka *The Game of Life*) is a simple cellular automata that also illustrates how very simple instructions can lead to a rich set of behaviours. Created by John Horton Conway this is a very famous model and it is likely that at least some participants have at least heard of it and/or played it before. There are no turtles here, only patches with each patch having its state set by some mechanism from outside the model initially and then on the basis of the states of other patches once the simulation begins.

Loading: To load the model, from menu choose `File` > `Models Library`. A window should pop up with a list of folders and potential models to be chosen down the left side and a space for summary information on the right side. Navigate the folders on the left with `Sample Models` > `Computer Science` > `Cellular Automata` > `Life`.

Get a feel for this simulation by running it a few times and looking at the Info pane. Change the slider values both before the simulation starts and after.

> Determine the conditions for life and death of a cell by observing the simulation in action *and* looking at the underlying code. Use each to confirm the other and confirm with your group.
>
> Add a set of tools to the interface that will let you set the life and death conditions from the interface. Decide how to do this with your group and then do this individually.

There are a number of ways to do this. One is to add two selection tools (drop-downs?) to the interface attached to the global variables "live" and "die" and then to change the code panel from:

```
ask patches
    [ ifelse live-neighbors = 3
        [ cell-birth ]
    [ if live-neighbors != 2
        [ cell-death ] ] ]
```

to:

```
ask patches
    [ ifelse live-neighbors = live
        [ cell-birth ]
    [ if live-neighbors != die
        [ cell-death ] ] ]
```

But "die" is the name of a built-in function in Netlogo (notice that it is blue) so we get an error. We'll change our names to accommodate this:

```
ask patches
    [ ifelse live-neighbors = life
        [ cell-birth ]
    [ if live-neighbors != death
        [ cell-death ] ] ]
```

Note that we could also have just used one variable to act as a threshold for being alive and calculated when to die from this:

```
ask patches
    [ ifelse live-neighbors = life
        [ cell-birth ]
    [ if live-neighbors != life - 1
        [ cell-death ] ] ]
```

> What are the benefits and detriments of each of the above approaches (two variables vs one variable)?

As participants are playing with the settings it may seem that anything that walks away from 3/2 (live/die) isn't very exciting. For example, 4/1 doesn't have a lot of dynamics but here is one that works:

```
XXX
000
XXX
```

> How could we be certain that there isn't anything "interesting" in one of these other simulations? What would "interesting" even mean in this case?
>
> Right click on the visualization space and choose "Edit". Change the size of the space and run *Life* again. Reset to the original size of 50x50 and then turn off "World wraps horizontally" and "World wraps vertically". Run *Life* again. Confer with your group and list the conse consequences of these changes.
>
> Discuss in your group how to describe the shape of an unwrapped world and a world with full wrapping on. Once you can do this imagine that instead of a 2D space we were working with a 3D space and then decide on how to describe what that space would look like with full wrapping.

Take Aways:

1. Simulations can be made purely with patches in NetLogo. Such simulations are called "cellular automata".
2. Being able to be clear about what you are looking for, for what is "interesting", is at least as important as writing the code and knowing how to use the tool.
3. Simple and seemingly innocuous choices such as the shape and size of the world have significant effects and should not be made lightly.

Life Turtle-Based (More than one way to…)

Summary: This is a rebuild of *Life* that uses turtles and some other fanciness rather than patches.

Loading: To load the model, from menu choose `File` > `Models Library`. A window should pop up with a list of folders and potential models to be chosen down the left side and a space for summary information on the right side. Navigate the folders on the left with `Sample Models` > `Computer Science` >

`Cellular Automata` > `Life Turtle-Based`.

> Have a look at the code of this simulation. How is it different from the *Life*? What seem to be the benefits and detriments of pursuing each approach?

A choice to be made based on time and/or the current level of understanding being shown…

If it seems necessary to lock in an understanding of how to interact with the interface and make some light changes to the code then have the participants modify this version to offer *both* a random seed control *and* control the life and death criteria in the interface.

Take Aways:

1. There is more than one way to produce a model.

Segregation (BehaviorSpace & BehaviorSearch)

Summary: This model illuminates how even a low preference to be surrounded by people who "resemble" oneself can lead to high levels of division between agents of different types. This is another simulation that could be done either with patches or turtles. In this case it is done with turtles.

Loading: To load the model, from menu choose `File` > `Models Library`. A window should pop up with a list of folders and potential models to be chosen down the left side and a space for summary information on the right side. Navigate the folders on the left with `Sample Models` > `Social Science` > `Segregation`.

Get a feel for this simulation by running it a few times and looking at the Info pane. Change the slider values both before the simulation starts and after.

> What does your group agree is strange/surprising/interesting about what is happening here?
>
> One thing that might be noticed is something akin to the claim on the info pane that a preference for 30% similarity leads to a rough average of 70% similarity in practice. How can this be confirmed?

To get these kind of confirmation we typically need a lot of single runs and that will take a long time and a lot of clicking. NetLogo has an answer for this: BehaviorSpace.

BehaviorSpace is a tool that allows NetLogo to run a simulation many times. It captures information about each run and drops the output into a file for further analysis. If you want to do more than some simple intuition testing and idea exploring then BehaviorSpace is going to be an important tool in your kit.

We'll use BehaviorSpace to get find out what a 30% preference for similarity amongst one's neighbours averages out to. To do this we:

1. Set the sliders to a density of 95 and %-similar-wanted to 30.
2. Go to the menu and choose `Tools` > `BehaviorSpace`.
3. From the window that pops up choose `New`.
4. We'll keep things simple this time around so set the following options:

   Vary variables as follows…:

   ```
   ["density" 95]
   ["%-similar-wanted" 30]
   ["visualization" "square-x"]
   ```

   Repetitions:

   ```
   100
   ```

   Measure runs using these reporters:

   ```
   percent-similar
   percent-unhappy
   ```

   Measure runs at every step:

   ```
   Off
   ```

   Setup commands:

   ```
   setup
   ```

   Go commands:

   ```
   go
   ```

   Time limit:

   ```
   100
   ```

5. Click "OK" and you'll be returned to the original BehaviorSpace window. Click "Run" from this window and the Run options window will open.

6. Choose both “Spreadsheet output” and “Table output” in order to see what each does.

7. There will also be an option to set how many runs should be done in parallel. This defaults to the number of cores on your system. If you don’t want NetLogo to take over all the cores then drop the value (but it will take longer). For now the default will be fine since we’re not asking for much. Click “OK”.

8. You will be asked where to save each of the two output file types. choose the locations and BehaviorSpace will go to work. You’ll see ¼ of the simulations running in the window.

9. When complete (less than a minute?) open the files is program that can handle comma separated value files and examine the output.

> What is the difference between “Spreadsheet” and “Table”? Which would you use when?
>
> The runs didn’t aren’t in order, why is this?
>
> What does a 30% preference for similarity produce on average?
>
> Now, what is the relationship is between preference and final similarity.?

To answer this question we’ll need to return to BehaviorSpace and scan a *range* of values for similarity preference, as follows:

1. Bring up the BehaviorSpace window and choose to edit the original set of instructions.
2. Modify the instructions as follows:

   Vary variables as follows…:

   ```
   ["density" 95]
   ["%-similar-wanted" [0 10 100] ]
   ["visualization" "square-x"]
   ```

   Repetitions:

   ```
   10
   ```

   We’re dropping the number of repetitions because we don’t have enough time in the workshop to wait for 1100 runs to finish! 10 runs at 11 points will be 110 runs total and take about 3 minutes.

3. Click through to running the simulation and choosing where to save the results.

4. This will take a fair bit longer than the first set because we’re doing 550 runs in total rather than 100. Further, some of these runs won’t settle as they did in the first case meaning that they’ll go to the cut off.

You can speed things up by unchecking the boxes for Updating view and updating the plots and monitors. You can also move the slider to “faster”.

5. When complete open the files is program that can handle comma separated value files and examine the output.

With the output from the BehaviorSpace run a curve can be plotted using other statistical tools to show the relationship between similarity preference and final similarity achieved.

> Decide in your teams what the “lesson” from the segregation model is.

If it has not already been discovered already, show participants how the slider can be moved during the simulation. This is particularly interesting for starting at lower preferences and then moving it up to produce higher preferences and vice versa.

BehaviorSpace lets us vary model parameters to see the range of outputs that result. We could use it to try and determine what outputs are needed to produce an outcome, say happiness in this model, but there is another tool in the NetLogo kit that is specifically designed for this: BehaviorSearch.

BehaviorSearch was originally built outside of NetLogo and had to be acquired separately but it is now included with NetLogo. BehaviorSearch still maintains its own separate websited behaviorsearch.org. We’ll use BehaviorSearch to explore the space looking to see what variation of similarity and density produce the highest overall % similar.

> We’ll be moving quickly through the setup here and providing minimal explanation of the options as compared to earlier. We just won’t have time to walk through each and the point is to show what is possible. The BehaviorSearch Tutorial is an excellent place to get a full explanation of all the features we won’t be spending time on.

To do this we:

1. Look back in the original folder that was created when NetLogo was installed and open BehaviorSearch.
2. Click the “Browse for models” button and navigate to the segregation model:

   `NetLogo 6.0.3` > `models` > `Sample Models` > `Social Science` > `Segregation.nlogo`

3. Open `Segregation.nlogo`.

4. At the bottom of the BehaviorSearch setup window is a button that says “Load parameter ranges from model interface”. Click it.

5. Modify the parameter specifications window to read:

```
["%-similar-wanted" [30 10 100]]
["density" [49 5 99]]
```

Setup: `setup` Step: `go` Measure: `percent-similar` Measure if: `true` Stop if: (leave blank) Step limit: `100`

6. Set the “Search Objective” panel as follows:

   Goal: `Maximize Fitness` Collected Measure: `AT_FINAL_STEP` Fixed Sampling: `10` Combine Replicates: `MEAN`

7. Set the “Search Algorithm” panel as follows:

   Search Method Configuration: `MutationHillClimber` mutation-rate: `0.5` restart-after-stall-count: `0` Use Fitness Caching: `On` Evaluation limit: `100` Best-Checking Replicates: `5` Search Encoding Representation: `MixedTypeChromosome`

8. Choose to “Run BehaviorSearch”.

9. Set the “Run Options Dialog” window values as follows:

   Output File Stem: `Point this wherever you'd like to store the output` Number of Searches: `5` Starting at search ID: `1` Initial Random Seed: `0` Number of Threads: `4` (Use however many cores your machine has) Brief Output: `Unchecked`

10. Click “Start Search”.

11. When BehaviorSearch finishes click “Done” and then navigate to wherever you told it to put the output and look through the files.

Take Aways:

1. BehaviorSpace is useful tool for testing expectations about models and for coming to understand the overall shape/space of the simulation, hence its name.
2. BehaviourSearch is a useful tool for finding model parameters that produce specific outcomes.

Independent Exploration

Participants will be given a choice between exploring other models within their teams, working on the challenges below, *or* getting started building their own model.

Challenges:

1. Modify Wealth Distribution so that the agents die and new ones are put in their place rather than agent recycling.
2. Modify segregation to handle three types of agents
3. Modify segregation to take the number of agent types as an input
4. Modify segregation to allow for the similarity preference of each agent type to be different

Final Things to Point Out

1. Open a `.nlogo` file in a text editor to see what's there: code at the top, interface setup below.

2. There are script/shell/command line ways of running Behavior Search and Behavior Space. This means that you can run LOTS of simulations on research clusters.