Complete User's Manual: FNNR-ABM-Primate Project

Note: The FNNR-ABM-Primate project (this project as of 8/15/2018) is J. Mak's thesis, involving modeling GGM population dynamics and movement. The FNNR-ABM project, by comparison, is led by the PES project team (http://complexities.org/pes/), and involves household PES enrollment, which is modeled in this project as of 2019.

Glossary/Terms

FNNR – Fanjingshan National Nature Reserve, our study site from which we have collected data (http://complexities.org/pes/research/recent-updates/).
GGM – Guizhou "Golden" Monkey
ABM – Agent-based Model
IDE – Integrated Development Environment
OS – Operating System
3.X.X – Version of Python; for example, 3.7.0 is the latest as of July 2018.

Table of Contents

(Press Ctrl+F and type in section keyword in capital letters to jump to section)

1. INSTALLATION - Have Python 3+ installed on your computer.
2. IDE – (optional but recommended) Download a Python IDE.
3. LIBRARY – Download the Python libraries needed for this project (Mesa, matplotlib).
4. GITHUB – Download and unzip the FNNR-ABM project files from Github.
5. READCODE - Understand the project goals, Python code, and imported libraries.
6. RUNCODE – Run the model, and understand which variables to edit.
7. PLOTS – Edit the exported model out for use in analysis.
8. HEATMAPPING – Generate heatmaps in Excel or ArcMap/ArcGIS Pro.
9. PREPARE (optional) – Learn how to prepare household buffer layers in ArcMap/ArcGIS Pro.
10. THANKS – Contact project developers for more information.

[Note that Sections 1 through 4 are prepared for ABM/ Python beginners; experienced Python users or modelers can download the Mesa v0.8.3, Pyshp, and Matplotlib libraries and then jump to Section 5. Make sure to download Mesa 0.8.3 specifically, and if running server.py, downgrade tornado to 4.5.2.]

1. INSTALLATION - Have Python 3+ installed on your computer.

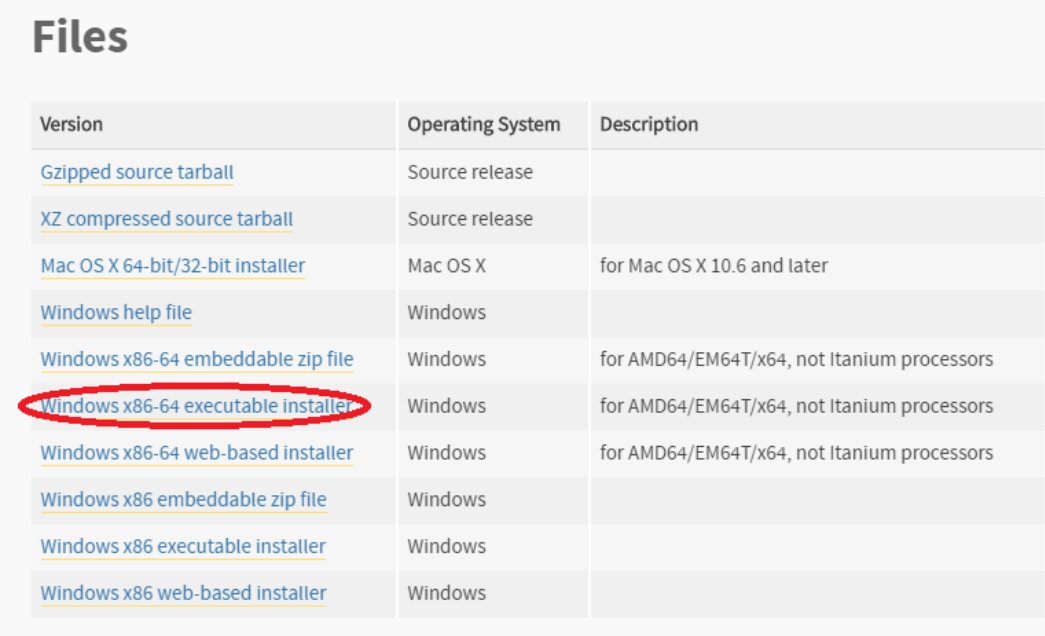
To download the latest version of Python, visit https://www.python.org/. Any version of Python 3.X.X should work. Python 2.X.X is more stable for use with older systems, but it differs in syntax from Python 3.X.X, so it is not compatible with code from the imported libraries we will use here (such as Mesa).

On the Python download page, scroll to the bottom and select the option that is best for you. For the most common configuration, refer to Figure 1.1; however, it may not apply to you. First, find out if you have a Mac, Linux or Windows OS, then figure out if your OS is 32-bit (x86) or 64-bit (x64). To find this out, view your computer properties (on Windows 10, search or find 'This PC' in File Explorer,

right-click, and select ‘Properties’ from the menu; other versions of Windows might need you to right-click ‘My Computer’). Most standard newer computers will have the 64-bit version of Windows.

Once you download and run the installer (or configure the zip file/tarball; the installer is recommended), follow the installation steps to install Python 3.X.X onto your computer. If you are not sure what options to pick, do not change the default options. Keep note of where Python is installed on your computer. If it is convenient and fast to do so, restart your computer afterwards.

Figure 1.1 – The most common option. This option may not be right for you if you are not using a 64-bit version of Windows.



Files

Version	Operating System	Description
Gzipped source tarball	Source release	
XZ compressed source tarball	Source release	
Mac OS X 64-bit/32-bit installer	Mac OS X	for Mac OS X 10.6 and later
Windows help file	Windows	
Windows x86-64 embeddable zip file	Windows	for AMD64/EM64T/x64, not Itanium processors
Windows x86-64 executable installer	Windows	for AMD64/EM64T/x64, not Itanium processors
Windows x86-64 web-based installer	Windows	for AMD64/EM64T/x64, not Itanium processors
Windows x86 embeddable zip file	Windows	
Windows x86 executable installer	Windows	
Windows x86 web-based installer	Windows	

2. IDE - (optional but recommended) Download a Python IDE.

There are many different software programs that will run your Python code. IDEs are optional to download because Python comes with a default one named IDLE (and for shorter python functions, one can even run code straight from the command line). However, downloading a more sophisticated IDE will handle different versions of Python and different libraries more seamlessly, as well as provide debugging/testing tools and more detailed error messages. They may also provide other tools such as a built-in file system to manage multiple Python modules (files) more easily, the ability to open non-Python files, and more.

Once you have found an IDE (google to find different ones available; the one used in this tutorial is PyCharm), follow installation instructions, unzipping/extracting any files with 7zip (a free program) or Winzip as needed. If the IDE you downloaded is PyCharm, make sure to create a new Python Project (only do this once, not every time you run the code) and place all of the FNNR-ABM-Private files inside the main Project Package.

3. LIBRARY - Download the Python libraries needed for the project (Mesa, pyshp, possibly matplotlib).

Python has many built-in frameworks and libraries (collections of pre-written functions and modules) that save users time and effort, as well as many more libraries available on the web to download; most common projects will use at least one external library (as opposed to being coded entirely from scratch). The two libraries we must download for the project are:

Mesa – a Python 3+ framework for working with agent-based models
Note: You must use Mesa v0.8.3 for this project!
It "allows users to quickly create agent-based models using built-in core components (such as spatial grids and agent schedulers) or customized implementations; visualize them using a browser-based interface; and analyze their results using Python's data analysis tools. Its goal is to be the Python 3-based counterpart to NetLogo, Repast, or MASON."
Documentation can be found online at https://mesa.readthedocs.io/en/master/
Or by downloading the .pdf at https://media.readthedocs.org/pdf/mesa/latest/mesa.pdf
If you install Mesa through pip (covered later here), it will come installed along with the other libraries it depends on, such as Tornado (web framework), Pandas (data structure library), and Numpy (for a variety of numerical expressions or generations). The user will likely not directly access these libraries when working with Mesa.

Matplotlib –This library helps build graphs to display Python data, but it may not be necessary to download this if the plot_setting is set to False in fnnr_config_file.py, especially if you are planning to create plots/graphs for data analysis manually using Microsoft Excel.

Pyshp – This helps convert your .csv file output to .shp for use in creating a point density map in ArcMap (see Section 7 of this User's Manual). It may not be necessary to download this; you can also open the output .csv density coordinate file the model generates in ArcMap and generate the XY data layer from that.

The most common (and Pythonic) way to install external libraries is to open the Command Prompt[1] on Windows (also known as cmd.exe, which is a terminal/shell for users to run administrative system commands) and type in:

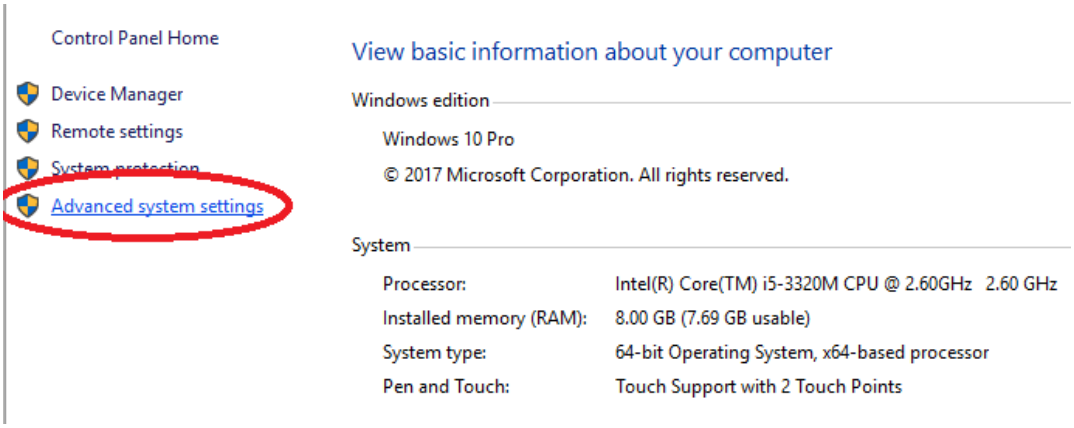
pip install mesa==0.8.3

Before you do this, read the troubleshooting section to see if you have set up your environment correctly; if you have, your current directory should not matter. If you are using conda or miniconda (or another environment/package manager like PyCharm), replace 'pip' with 'conda' in the above commands. Ensure that you are in the right directory while running commands.

Troubleshooting
There are a number of possible error messages you can get. The instructions below diagnose them based on Windows 10.

If 'pip' is not recognized in the command window:
1. Set the Environment Path.
a. This PC or My Computer > Right-Click > Properties
b. Click on 'Advanced system settings' on the left tab.

Figure 3.1 – Setting System Environments Before Using Pip

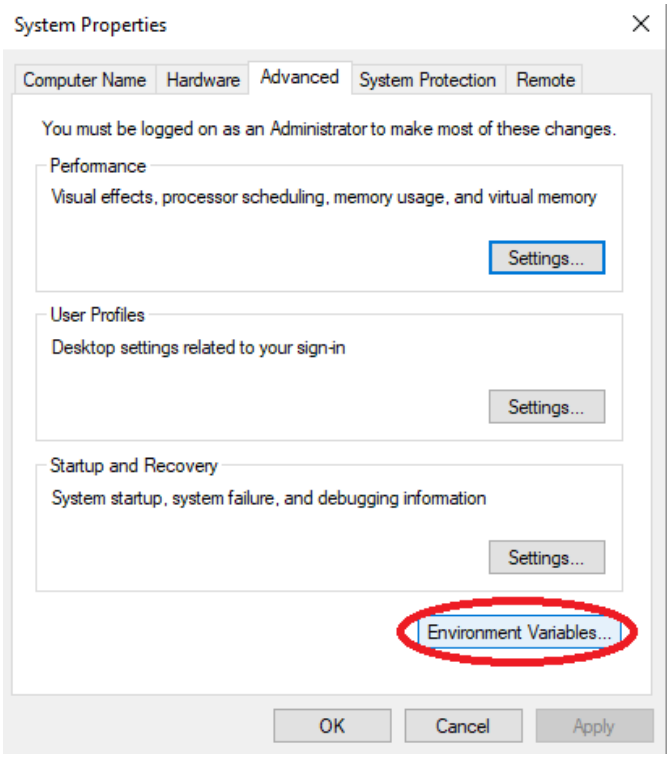


c. Select 'Environment Variables...'.
(An alternative to steps a-c is to search for 'environment variables' from the Windows main menu search tool without submitting and select the auto-completed suggestion, or to access it via the control panel.)

See Figure 3.2.

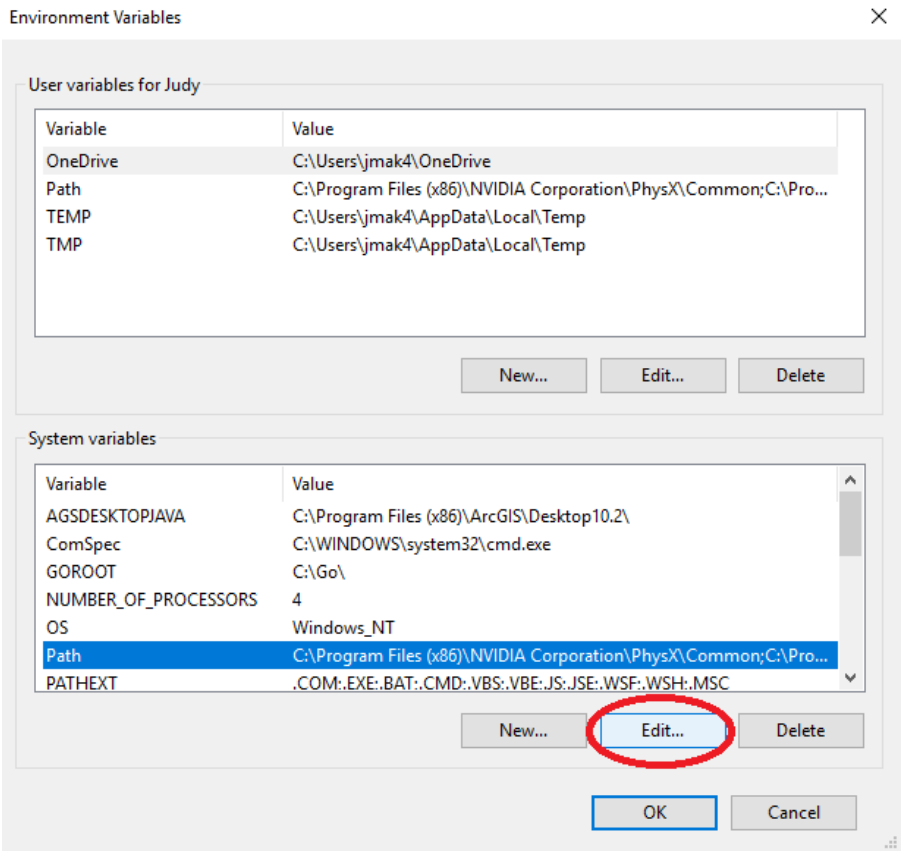
Figure 3.2 –Advanced System Settings

[1] In Windows 10, click on the Windows icon on the lower-left corner of your computer monitor, type any letter, and you will see the search window. Type command or cmd in the search line, and you will see the icon for the Command Prompt. cmd.exe is the name of the Command Prompt executable program.



d. Select 'Path' under 'System Variables' (near bottom of the window, not the first 'Path' near the top), then 'Edit...'.

Figure 3.3 – System Variables

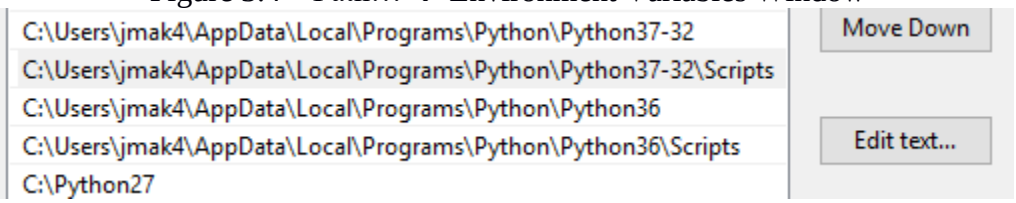


e. Select 'New', then 'Browse...' to find where your Python installation is. Common filepaths to add here include (depending on where you've installed Python):

C:\Python27 ← likely
C:\Users\<YOUR USERNAME>
C:\Users\<YOUR USERNAME>\Downloads
C:\Users\<YOUR USERNAME>\AppData\Local\Programs\Python\Python36 ←likely
C:\Users\<YOUR USERNAME>\AppData\Local\Programs\Python\Python36\Scripts ←likely
C:\Users\<YOUR USERNAME>\AppData\Local\Programs\Python\Python37-32 ← likely
C:\Users\<YOUR USERNAME>\AppData\Local\Programs\Python\Python37-32\Scripts ← likely

Basically, add the directory that contains the same version of python.exe that you want to run, and maybe others, such as your Downloads folder, to be safe. Make sure that when adding new filepaths, you do not overwrite or delete old ones in the current list. Refer to Figure 3.4.

Figure 3.4 – Path… → Environment Variables Window



Also add the \Scripts\ folder after the Python folder as seen in Figure 3.4, just in case (this may or may not be strictly needed).

*NOTE: If you have multiple versions of Python installed, make sure that the Python version you want is moved up above the other version(s). To do this, select the 'Move Up' button in the Path > Edit… window. Now you should be able to run pip in the command line to install the necessary libraries.

2. Change the CMD Directory.

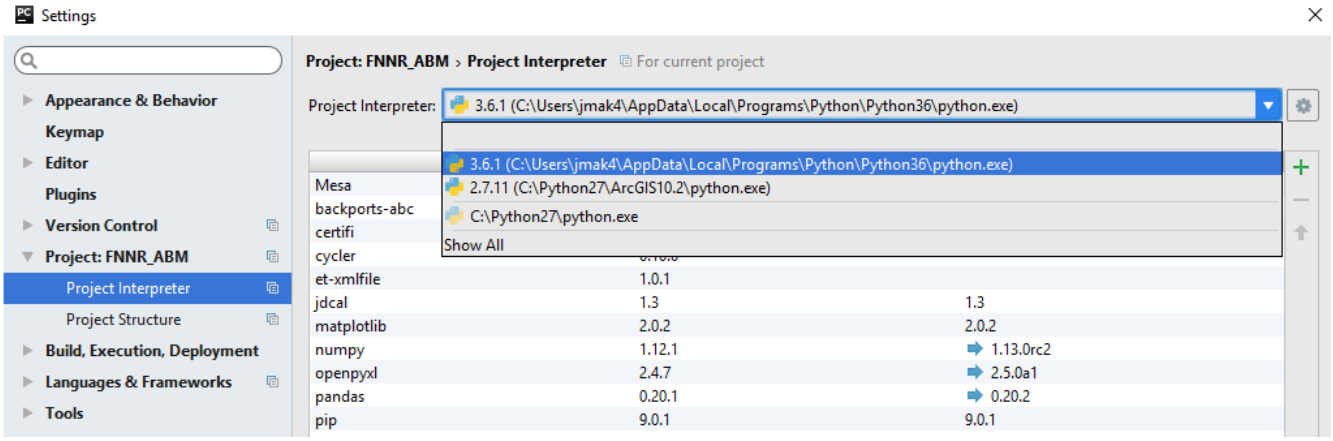
'cd' is a command that changes directories to what the user types.
Once you are in the correct parent directory, your system should take care of the rest.
For example, if you have Python installed under C:\Users\<YOUR USERNAME>\AppData\Local\Programs\Python\Python36, then in cmd.exe, you may want to type:
cd C:
then
cd\Users\
in order for cmd.exe to change its directory and look for pip in the right drive.
Another useful shell command is dir, which shows folders/files in current directory
Do not type 'python' directly into the cmd window unless you want to access the Python shell directly, but if you do (for example, to check the Python version), you can type 'exit()' to return to being able to deliver typical commands.

If it installs successfully in the wrong directory, or if your IDE does not recognize the library after installation:
3. If you are using Anaconda/Miniconda and the library has installed the library in the wrong environment (such as one for a version of Python 2.X.X), set up a new environment; otherwise, skip to Step 4.
To set up a new environment, type the following into the Command window:
conda create --name 3point6 python 3.6

Note: ‘3point6’ here can be any name you wish, and ‘3.6’ can be changed to another version of Python. Then activate the env in the Command window:
activate 3point6 (or whatever you named it)
You should be able to use the pip command to install the needed libraries under this new environment now. After you do so in the command line, proceed to Step 4.

4. Change your IDE/project interpreter.

Figure 3.5 – Setting Project Interpreters in PyCharm



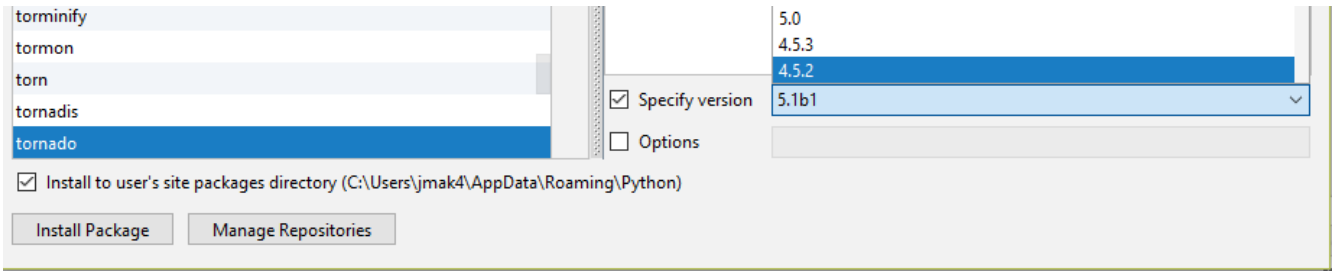
This varies per IDE, but in PyCharm, you will want to go to File > Settings to set the Project Interpreter. The Project Interpreter should either be wherever you have the desired version of Python installed, or in a Conda environment that has the desired version of Python installed (see Step 3). You will know you have selected the correct environment when:
- the Python version shown is 3.X.X, and
- the libraries shown in the table under the Interpreter selection dropdown box include pip, matplotlib, pandas, Mesa, and more (assuming that you have successfully used pip or conda in the command line to install the libraries at this point).

Updating Libraries
There are multiple ways to ensure you have a specific version of a library. Pip and conda are two ways, but I prefer to use PyCharm’s Project Interpreter under File > Settings (see Figure 3.4) to install libraries.

Note: The version of Mesa must be 0.8.3!
Note: tornado should be version 4.5.2 – the latest version (5.X) may be bugged with Mesa.

Double-click a library name in the Project Interpreter to install, update, or revert a library. In the case of a new user, they will probably want to revert their tornado library. The screenshot below shows how to select version 4.5.2—even if you have a later version—and install that package in PyCharm.

Figure 3.6 – Reverting to tornado version 4.5.2 or Mesa 0.8.3



Finally, if your libraries appear to have successfully installed to the same directory that runs the desired version of Python as configured in your IDE, but you still get import errors, you may need to restart your computer.

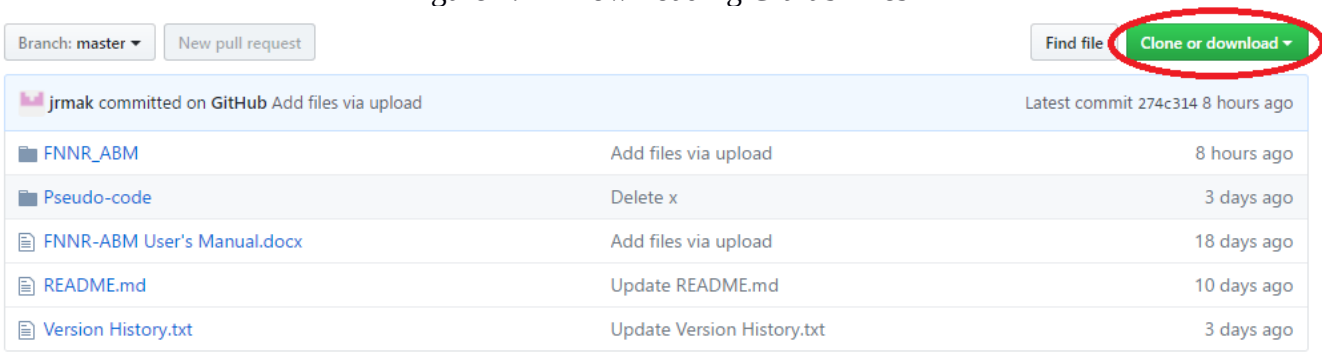
4. GITHUB–Download and unzip the FNNR-ABM-Primate project files from Github.

Github is a file hosting, sharing, and version-control website and tool, particularly for code files of various computer languages. It functions similarly to Google Drive, but for code. For the purposes of this project, it is useful for sharing and storing different versions of .py files in an easily-readable format for others to download, edit, share, and track version histories of file changes. In that respect, it resembles an advanced version of Google Drive or Google Docs, but specifically meant to be used for collaborative code sharing. While Github is a powerful suite that comes with its own commands and software, named Git, we will only cover how to navigate the website's basic functions here.

This project's Github repository is located at https://github.com/jrmak/FNNR-ABM-Primate.

Once you navigate to the page, the files from the repository will be shown; the organization is similar to Window's file system. To download the files, find the green button to the right that says "Clone or download." This can only be done from the main repository page, and not individual files, though one may also copy and paste code directly from viewed raw files.

Figure 4.1 – Downloading Github Files



Change the above figure (using a graph from your site at https://github.com/jrmak/FNNR-ABM-Primate)
Shown above is an example from another project, but the green 'download' button should be in the same place.

Make sure you are downloading the correct branch/version of the code (usually 'master'). In this example, the master branch (see the leftmost grey button at the top of the image) is selected by default; it is usually the latest stable version of the project available. Once you click the green button, select the option on the right (download as .zip), unless you have Github for desktop installed and specifically

want to edit the files from there (not be covered in this tutorial). The repository's files will then be compiled into a .zip file; unzip it using WinZip or 7-Zip. The name of the folder should resemble 'FNNR-ABM-Primate-master.'

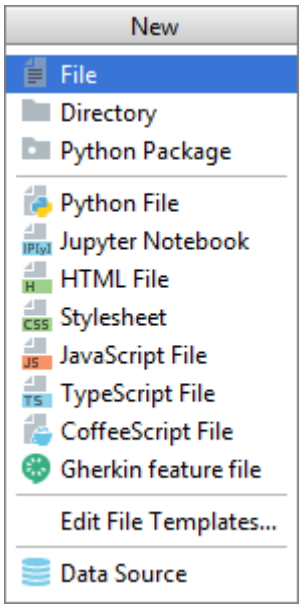
Once these files are downloaded to your computer (you probably only need the code files within FNNR_ABM-Primate) and unzipped, move them to the appropriate location on your hard drive. If you are running a very basic setup with IDLE (the default Python IDE), it is possible to keep these files in your Downloads folder (in Windows), but the steps below are recommended.

You only need to do the following the first time you download the files:

Using PyCharm, create a new PyCharm project by File > New Project… in the upper left corner. Name the project whatever you want (I used 'FNNR_ABM_Primate') and place the project wherever you want (I prefer to keep the default under PycharmProjects, then once the project has been created, create the Python package using File > New… as shown in the figure below (Select 'Python Package' from the drop-down list). Name the Python Package FNNR_ABM_Primate, even if you have already named your project that.

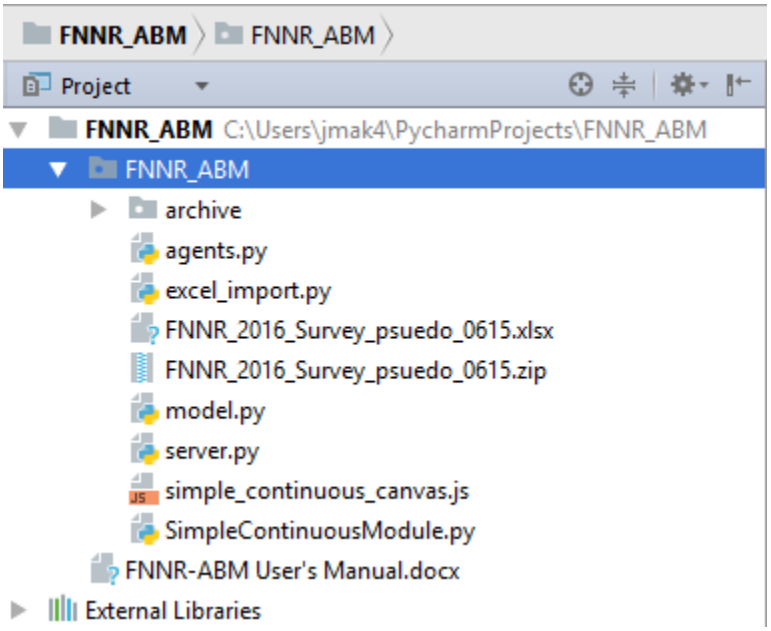
Note that in general, if you would like to start a new .py file to code and save, select 'Python File' and not the generic 'File' type at the top of the menu (see below).

Figure 4.2 – New 'Python File' (not the selected 'File')



From the FNNR-ABM-Primate file you have on your computer, copy and paste your code (.py) files into PyCharm under your Python Package folder directory. Also paste the latest copy of the unzipped .xlsx Excel data file. The result on the left-hand window should look similar to this:

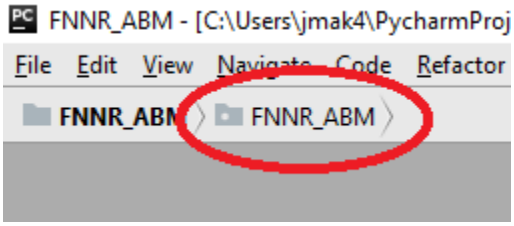
Figure 4.3 – Project Structure (Only create a new project once)



(Again, replace 'FNNR-ABM' in Figure 4.3 with 'FNNR-ABM-Primate'. Please note that the above screenshot is an example and does not reflect the current state of this project—it was taken from FNNR-ABM, not FNNR-ABM-Primate.)

Note that if this window is not visible, you may need to toggle it to show within PyCharm. One way is to double-click the Python package's tab:

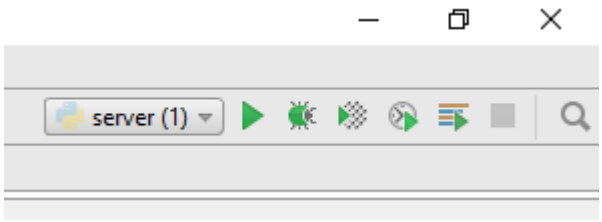
Figure 4.4 – Showing Visible Windows



Note: even though the example in Figure 4.4 shows 'FNNR-ABM', you would actually be showing the 'FNNR-ABM-Primate' project.

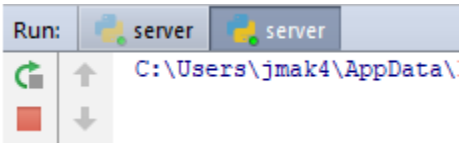
The code should now be visible whenever a module is opened from the left-hand side directory. To run the code, open server.py (it must be this module for this project specifically) and press Shift + F10 or run it manually under the Run… option at the top toolbar, or right-click the tab at the top where the module name is visible (e.g. 'server.py'). Afterwards, server or graph should be the default module set to run next to the green buttons (Fig. 4.5) on the top right corner, as pictured below; you can also click the green triangle to run.

Figure 4.5 – Run 'server' (Despite the screenshot, it should not say 'server (1)' unless you have stored your files in multiple directories)



*Once you run the code, especially for server.py, make sure you exit the instance afterwards. You can do this by either closing any windows the code opens (whether it is a web browser tab or all matplotlib graph windows) or by clicking the red rectangle (symbolizing 'STOP') in PyCharm next to the bottom-left or top-right portions of the screen. Make sure you end all instances of "server" running. When in doubt, you may also press Ctrl + F2 to terminate all processes running. If you are looking at a graph or running the visualization in the web browser, this will close the windows.

Figure 4.6 – Stopping All Processes



5. READCODE - Understand the project goals, Python code, and imported libraries.

The goals of the FNNR-ABM-Primate project are as follows:

- Simulate and record changes to monkey population structures and dynamics over time;
- Simulate agent movement in a visualization given different input environmental grids;
- Understand monkey group habitat use and behavior from a bottom-up approach; and- Incorporate human household and gathering data into the model to simulate their impact on monkey group movement.

This section is comprised of a detailed description of each module, its functions, and how the Mesa framework supports the code.

There are two parts to the model: the visualization of agent movement and a demographic population structure model. Both parts run at the same time regardless of which module you choose to run (graph.py and server.py export different outputs, but the full model runs regardless of which you pick since both submodels interact with each other), and share the same agents, but only one result (either visualization or population graphs) can be calculated and shown. The other modules are shared between them. For example, in both models, human individuals age, populate, die, and gather resources; in the visualization, their resource gathering affects monkey movement by drawing them away from certain areas of their breeding habitat and limiting their range, whereas in the demographic submodel, the results of changes in human demographic structure in the FNNR are recorded and exported.

The model is run in steps; every step represents five days in a year (which means that 73 model steps represent one year). During each step, a multitude of functions execute. The function parameters are determined based on real-world data (Yang, Lei, and Yang 2002); these parameters also utilize some

random generators in limited situations where accurate information is not available or uncertainty exists.

This model runs on the Mesa library framework, which is specifically designed to use Python to create agent-based models. Mesa contains the following tools (as well as more not covered at this time):

- Defined Agent and Model superclasses that allow an ABM-style relationship between the two, as illustrated by time-steps
- Visualization and data-graphing components (web simulation, data collector).

Other libraries include pyshp, which is only used in convert_csv_to_shapefile.py, and matplotlib, which is only used in a commented-out section of graph.py.
Below are the modules that comprise the model's code. Some of these are found in Data/ folder rather than the ABM/ folder; both are available on the model's Github repository.

An asterisk (*) before the module name (such as *example.py) indicates that the module is executable (runnable).

***server.py and *graph.py** – Described in the next section; runs the submodels. To clarify, graph.py is what should be run if you want data output; server.py is simply a web browser simulation.

model.py – contains the main model structure (movement class); initializes agents.
First, monkey families are randomly generated, then members within the families are generated. The starting population structure models the pseudocode rules—see 'Pseudo-code-GMMs-06-27.docx' on Github under the 'Data' folder (for example, monkeys aged 7-10 comprise 20% of the population, so if a random number generator from 0 to 1 brings up a number between 0.42 and 0.62—a range of 0.20—the resulting age is a random number from 7 to 10; Yang, Lei, and Yang 2002).

After the families are generated, they are placed on the grid to move in the visualization model. The grid values themselves (attributes of these grids such as elevation, slope)—which will be weighted to determine movement direction each step—are read from a .txt file (file name) in the same directory (included in the Github files as agg_veg60.txt and agg_dem87100.txt).

Every step, the model runs, and each agent (family agents for the visualization and monkey agents for the population model) follows their rules as set in families.py, monkeys.py, and humans.py.

families.py - The Family agent class determines the behavior of the pixels—that is, family groups of 20-40 monkeys—in the visualization. 'Family' in 'families.py' refers to a monkey family only. The family agent class is only utilized in the visualization, and does not affect the data output generated by graph.py.

Every step, family (monkey group) agents:

- Move towards the Yangaoping region in the northeastern FNNR in the month or two before April and September (their mating seasons);
- Move around the Yangaoping region during April and September; or
- Move away from the Yangaoping region in the month or two after April and September;
- They also always move away from humans gathering resources or from low elevations, regardless of the time of the year.

- Finally, if they are not moving towards or away from the Yangaoping region (either because it is not close to mating season or they are already within the Yangaoping region during mating season), and other human/elevation rules do not apply, they move according to the surrounding vegetation of their 8-neighbor cell block (that is, north/east/west/south and diagonals). They are most likely to move to coniferous, broadleaf, mixed, or deciduous regions; somewhat likely to move to shrubland, lichen and bamboo regions; and least likely to move to PES, managed forest, or farmland regions. They cannot move into human settlements.

monkeys.py - the Monkey agent class determines the behavior of each individual (e.g., birth, death, growth) in the population / demographic structure model.

Every step, monkey agents:

- Age, increase the amount of time passed since the last birth, and check the age category; upgrade the age category if the monkey moves onto the next one
- Check if the monkey has recently lost an infant (if a mother)
- Check if the male family needs to break off of a main group (if in an all-male family); if so, creates that family and moves other agents into it
- Splits a large family into two if the family size is too big; also stops migrating out family members once it reaches a small size again
- Gives birth if the gender (1 = female), birth interval, random chance, and time of year allows
- Dies if the random chance for their age group for a 5-day time-step allows

environment.py – contains the environmental grid class, which contributes trivially to understanding the model, but is necessary code-wise.

humans.py – contains the Human agent class.
Every step, the head of each human household, as determined by either their order in the imported FNNR data or the agent's age, moves towards a randomly-chosen resource from a list of resources their household gathers. Once they have the resource, they move back home and repeat the process again, depending on the current resource's listed frequency attribute—the monthly frequency represents the amount of times per month they are "exposed" to monkey agents, or otherwise able to block monkey movement. There is one moving human collector agent per household. Human households are situated accurately in terms of geographic relative position to each other, clustered in two distinct villages near the Yangaoping region. Different resources have different frequency rates for collection and different distances to each human's home location.

Most human collector agents have other human agents who stay at home. Even though their points do not move on the map, the human model still models human demographics every step. Data from FNNR residents from the two villages near Yangaoping were used, though future generations (once the model starts generating newborn humans) are generated. Finally, households are not their own agent, but each human agent is tied to a household ID, which updates household attributes through global lists.

Every step, human agents:

- Set work and education status and death rates based on their age category
- Age and upgrade age categories if they have aged to the next one
- Migrate, whether temporarily due to S. Yang's formula to possibly send remittances, or permanently due to college (rare)

- Return from migration, if applicable, and open up a spot for a new migrant (since each household is only allowed one migrant)
- Gather resources, which moves them on the grid in the visualization (seen if server.py is run, otherwise only serves to affect monkey movement in graph.py)
- Determine if they should be named the head of the household (if the past one has died or migrated); usually, only those of working status/of age can be named the head of the household
- Marry, if of age and a certain random chance has been met
- Die, according to a random chance as determined by their age category
- Give birth, if of the correct age/gender (2 = female) and the time interval for births is high enough; each female can only give birth a certain amount before her life plan maximum threshold of # of children is reached, and only if she is of birth-giving age and married

This module also contains the Resource agent class, even though it does not currently need to be an 'agent' in the sense that they do not dynamically change.

land.py – contains the Land agent class, which represents each individual land parcel owned by all households. The land parcel comes from S. Yang's data and do not change in total quantity during the model run. These agents are only represented in the demographic run (graph.py), not in the visualization (server.py).

Every step, land agents:

- Have a small chance of converting to GTGP if non-GTGP, or vice-versa (the original formula a yearly chance, but each step represents 5 days; this is accounted for)
- Update their crop prices, unit outputs, and other attributes based on their GTGP status

fnnr_config_file.py – Edit the settings in this file to change parameters in your model run. These parameters include GTGP compensation type and amount, setting the simulation to run with or without humans, the number of monkey families/population of monkeys in the reserve, and more.
The settings are explained as follows, with the necessary Python variable type in parentheses, though there are also notes in the code comments:

- run_setting (string): by default, run the model as a "normal_run;" first_run is an advanced for when you have expanded the model or have just loaded your own buffer files (see section 9 of this documentation)
- plot_setting (string): by default, it is False. True may generate errors because this is legacy code; it may be worked on later.
- family_setting (integer): 20 by default. Represents the # of monkey families in the reserve.
- year_setting (integer): 10 by default. The number of years the model runs for. 1 year = 73 steps; 10 years = 730 steps; 20 years = 1460 steps.
- human_setting (string): "with_humans" by default. Turning off the human setting will run the model (whether you run graph.py or the simulation in server.py) without any human activity.
- PES_span (integer): how long the GTGP lasts, in years.
- no_pay_part (integer): the chances a household will remain in the GTGP after it ends.
- min_threshold (integer): very similar to no_pay_part; see the pseudocode.
- scenario (string): "flat" by default; can also be "land_type" or "time." Defines the type of GTGP compensation scenario that occurs; 'flat' means the same amount is paid the whole time. The 'land_type' setting means that compensation varies between dry land and rice, and 'time'

means that compensation rises or is lowered after a certain amount of time (in the variable 'time_breakpoint') has passed.

- unit_comp_<various> determines the unit compensation depending on the 'scenario' variable. Only the unit_comp that applies to the set scenario variable will apply.
- land_step_measure: default is 6; do not change unless the entire model's time resolution changes. Only included as a variable because of the pseudocode.
- random_walk_graph_setting: default is False; only set to True if family_setting = 1 and you want to generate a line plot of random-walk movement in the FNNR over a year by a single monkey family.
- model_exported_density_plot_file: legacy variable; will be removed in a future update.

excel_export_summary_monkeys.py– exports monkey age/gender structure results per step of the monkey population sub-model to an Excel (.csv) file when graph.py is run.
abm_export_summary_monkeys_<suffix depends on trial run>.csv – accompanying exported output file; explained more in section 7.

excel_export_summary_humans.py – exports population results per step of the human population sub-model to an Excel (.csv) file when graph.py is run; also exports human demographics.
abm_excel_export_summary_humans_<suffix depends on trial run>.csv – accompanying exported output file; explained more in section 7.
abm_excel_export_summary_human_demographic_<suffix depends on trial run>.csv – accompanying exported output file; explained more in section 7.

excel_export_density_plot.py – exports complete movement results for all agents at all cells of the movement/visualization sub-model to an Excel (.csv) file when graph.py is run.
abm_export_density_plot_with_humans_<suffix depends on trial run>.csv – accompanying output file; explained more in section 7.

excel_export_summary_households.py - exports household GTGP data when graph.py is run.
abm_excel_export_summary_household<suffix depends on trial run>.csv – accompanying exported output file; explained more in section 7.

***difference.py** – Creates the difference output in a .csv file (from which a heatmap or point density map can be generated, though this is much less important than the main heatmap) between any two given trials of "With Humans" vs. "Without Humans" scenarios. (To save outputs from different trials, either rename the output files manually each time after each run or edit the "movement_session_id" variable near the top of graph.py.) If one wants to see a "typical" result: since it is computationally intensive to average all movement outputs from all trials, one would then choose the trials whose kappa results for each threshold most closely match the averaged kappa statistics for each threshold from multiple runs. In other words, difference heatmaps can only be created from single trials compared against each other.
Example: if the average kappa statistic between two runs—that is, a landscape set of "With Humans" vs. "Without Humans" scenarios—for a threshold of some X number of sightings at each coordinate over Y number of trials was 0.644, and the kappa statistic for a threshold of Z number of sightings for a given range within the same set of Y trials was 0.350, pick a trial that exhibits kappa statistics close to those numbers for X and Z thresholds for a "typical" result—that is, a result whose buffer ranges would resemble those of the average. Actually, one could randomly choose the trial pairs, and the difference

image would not change much (it would show that the humans' greatest impact on monkey movement were at the location of the settlements).

***kappa_batch.py –** Batch-processes kappa statistic results from multiple trials to be averaged manually in Excel (generates an Excel file with multiple rows of kappa coefficients that can be easily handled). Inputs must be trimmed "with_humans" and "without_human" .csv files (excel_export_density_plot_<suffix>.csv), or the trimmed .csv output of write_maxent.pv and the averaged difference of all trials of the "with_humans" and "without_humans" .csv files (the latter is computationally intensive to calculate—you can also choose the single trial whose kappa values are closest to the average).

***randomwalk.py**–This module can show you a graphic of a single family-group agent's walk path for a model run. To run this, you must run graph.py while the # of families attribute in model.py is set to 1 and then use the excel_export_density_plot_<suffix>.csv output of that model run with this module; in other words, this module can only follow one family-group agent at a time, or else multiple agents' movement paths will become falsely correlated with each other, and the paths will entangle.

***trim_grid35.py**– Once a heatmap .csv file is generated in graph.py (excel_export_density_plot_<suffix>.csv—change the suffix by changing graph.py's "movement_session_id" variable in between runs), run that file with this module to only extract the Yangaoping region. Then a heatmap can be created in Excel from the output (select all data, excluding headers such as 'x' and 'y' column labels → create an X-Y scatterplot → set points to 99% transparent, no outline) or via ArcMap (instructions later in this tutorial—see next section), or run through kappa_batch.py for statistical analysis.

***write_maxent.py** – Translates a Maxent ASCII (.asc) file (generated when a Maxent suitability analysis is run by the Maxent software program) into a heatmap in .csv using a percentile-based "translation" function. The Maxent ASCII file is usually manually trimmed to the Yangaoping region (bottom 65%, or 65, rows truncated from the text) first.

***convert_csv_to_shapefile.py** – Converts a .csv file full of X, Y coordinates (with X being column 1 and Y being column 2) to a shapefile and adds a 'Counts' column that counts all occurrences of that particular X, Y coordinate (that is, the row) in the entire .csv file. This is written to work with the coordinates provided by the model's ~85 x 100 (or 85 x 35 once trimmed) grid (which, in turn, represents a 300m spatial resolution where the visualization submodel can run smoothly). The shapefile can then be opened in ArcMap; see section 8 of this document.

***30_trials_step6_732_1458.py** – After a set number of trials are run for the model for a given set of settings, this script is used to help compile all of the step 6 (0 years), step 732 (10 years), and step 1458 (20 years) outputs into one folder. Once this script achieves this, it is trivial to find the average output for each of the steps between multiple model runs. This assumes constant settings and must also be repeated for each type of output .csv folder the model generates (e.g. human demographics, monkey demographics, etc.).

agg_dem87100.txt, agg_veg60.txt – ASCII files that contain the elevation and vegetation data extracted from C. Tsai's DEM and vegetation maps, respectively, which both form the environmental grid that the monkeys move upon. The space is 87 x 100 (adjusted to fit an 85 x 100 grid) with a resolution of 300 m (grid size).

pes_ascii200.txt, pes_ascii400.txt, farm_ascii300.txt, farm_ascii600.txt, hh_ascii300.txt, hh_ascii600.txt, forest_ascii200.txt, forest_ascii400.txt - See Section 7 of this document for how to create a .txt layer with a custom buffer distance (the distance is in meters). This represents the human settlement area in which monkeys either will not enter or have a low chance of entering. By default, the buffers are the lower of the two numbers provided for each category (PES, farm, household, and managed forest); you can switch to the higher-distance buffer and observe model changes if needed. Resolution (grid size) is 300 m, in which the study site is represented approximately as an 87 x 100 lattice, but may vary slightly.

hh_survey.csv – Contains the locations of resources for human agents to gather, as well as the frequency of each said resource. The frequency formula can be found in humans.py, and bases the amount of human movement on the frequency of the selected resource collection; however, due to the relatively low time resolution of five days per step, it is a proportional/symbolic, rather than a direct, visualization of human resource gathering.

hh_citizens.csv – Contains data for the households surveyed in the FNNR and their residents, taken from S. Yang's research.

hh_land.csv – Contains data for the households surveyed in the FNNR and their land parcles, taken from S. Yang's research.

household.csv – Contains locations for human agents to call "home." Also determines the amount of human agents—in the August 2018 version of the model, there is one human agent (resource collector) representing each household.

grid_elevation, grid_veg, grid_without_humans, masterdict_elevation, rest_of_reserve_dict, etc. (no filetype suffix) – created by the Python 'pickle' tool when the model type is set to 'first_run' in the parameters at the beginning of model.py (near "def __init__"); these are pre-loaded files used in the model when its setting is 'normal_run' (set in the same place as 'first_run') to speed up processes.

CanvasGridVisualization.py, GridDraw.js, CanvasModule2.js – edited parts of the Mesa library stored in the local directory; not written by me, only slightly modified. If these are deleted, the import statements must reroute to the default versions and location stored in the Mesa library (same names, except replace 'CanvasModule2' with 'CanvasModule'). These were included for personal aesthetic changes that were not possible to make otherwise (such as erasing the visible lattice outlines from the grid in the web browser visualization).

Legacy files associated with the project include agents.py, kappa.py, heatmap.py, resource_dict.py, and maxent.py. If you have downloaded an older version of this project in the past then updated the same directory with newer downloads from Github, you should delete these legacy files; they are no longer used for the project.

6. RUNCODE – Run the model.

The code can be run in multiple ways, depending on what data you need to access.
Before running, you may want to edit the configurations in **fnnr_config_file.py**.
Then you may execute (run) one of the following modules:

server.py – run this module if you would like to see a visualization of monkeys' family (small-group in terms of Yang, Lei, and Yang 2002) movement, as well as human resource gathering, through the FNNR in a given year (73 time-steps/year).

graph.py– run this module to generate date of the population changes for humans and monkeys over years, including births, deaths, and at the end of the model run, the demographic structure. This also updates (or creates) a .csv file listing the age and sex structure of the monkeys in the reserve, and updates or creates a .csv file containing total agent movements that can have Excel heatmaps created from the data.

Once the model is run in an IDE, it will take several minutes to complete (730 steps = 10 years at 73 5-day "ticks" a year; the model can be stopped before finishing, but the data written will not be complete). Make sure the model is set to "normal_run" from the model parameters near the beginning of model.py (near def __init__) if it is not the first time you are running it, and that the number_of_families (for monkey family groups) is set to 20 to accurately reflect a fully-sized GGM population. Results can be seen immediately (as in, with no further analysis work needed to generate output data) after the first run (that is, after a few dozen or hundred steps of the model are run) with the following .csv files generated: abm_export_summary_monkeys.csv (for monkey populations) and abm_export_summary_humans.csv (for human populations). Further actions can be taken to analyze model results more deeply; this is described in Step 7 (Export) of this document and also in J. Mak's thesis (searchable/downloadable through the SDSU library).

If you would like to run the legacy version of the model that primarily focuses on monkey demographics and movement (no new information, but runs faster), as studied in J. Mak's thesis, download the 8/2/18 version found on Github (though it does not come with fnnr_config_file.py; you would have to edit the settings within the code). This version includes randomly-generated head-of-household gatherers instead of actual human data in a live population model from the FNNR.

7. PLOTS – Edit the exported model output for use in analysis.

The following files (with suffixes) should be generated once a model has fully run (that is, all steps have run; by default, the model runs for 730 steps, or 10 years):

- abm_export_density_plot.csv
- abm_export_summary_monkeys.csv
- abm_export_summary_household_(varies).csv*
- abm_export_summary_humans.csv
- abm_export_summary_human_demographics.csv

* The name of the household file will vary depending on the settings in fnnr_config_file.py.

abm_export_density_plot.csv details a full list of all coordinates (in the 85 x 100 grid of the model) that the monkeys have stepped on. You may use trim_grid35.py to restrict the region to the Yangaoping area (that is, 'trim' the data to an 85 x 35 grid to represent the top 35% of the reserve). Afterwards, it is ready to either have difference.py run on it (to generate a difference heatmap between that and another abm_export_density_plot.csv from a different run), or convert_csv_to_shapefile.py run on it in order to generate a heatmap from it in ArcMap (see section 8).

The other output .csv files should be processed through Excel, and basic line or bar graphs/plots can be made from the data directly.

abm_export_summary_monkeys.csv outlines changes in monkey population, total births, total deaths, and the demographics every 6 time-steps (that is, every 30 days, or approximately every month) for the entirety of the model run. It also lists the female-to-male ratio as a decimal point, and the number of females of reproductive age. The monkey population formulas are derived from data gathered by Yang, Lei, & Yang (2002).

abm_export_summary_household_(varies).csv is named according to the GTGP compensation settings in fnnr_config_file.py. An example may be abm_export_summary_household_flat_270.csv (with suffixes), which indicates that households are compensated a flat amount for GTGP land parcel enrollment. This file lists the total number of non-GTGP and GTGP land parcels in the two studied villages during the model run, the average number of non-GTGP and GTGP parcels for each household in those villages during the model run, and the average non-GTGP and GTGP parcel area (in mu) per household in the villages during the model run.

abm_export_summary_humans.csv outlines changed in human population, total births, total deaths, marriages, total number of laborers, single males, married males, and current number of migrants in the two villages studied near the Yangaoping region in the FNNR. Starting human data are gathered from Yang et al. (2014).

abm_export_summary_human_demographics.csv outlines the age breakdowns of the human populations for males and females for ages 0-10, 10-20, 20-30, 30-40, 40-50, 50-60, 60-70, 70-80, 80-90, and 90+. This file is used only to create population pyramids. A tutorial on creating those is not included here because it can easily be found on Google; it also depends on what version of Excel you use.

8. HEATMAPPING - Generate heatmaps in Excel or ArcMap.

Optional: To create an Excel Heatmap from the .csv file, make an X-Y scatterplot from the data (exclude the column headers such as ‘x’ and ‘y’), and set each point to 99% transparent with no outline, and adjust the axes accordingly (X-axis range: 0 to 85; Y-axis range: 0 to 100 untrimmed to represent the entire FNNR, or 65 to 100 trimmed to represent the Yangaoping region). This is extremely processor-intensive and may cause your computer to lag with every action while creating the plot. The next section teaches how to create a point density map in ArcMap, which will yield higher-quality results.

a. First, make a copy of your ‘excel_export_density_plot.csv’ file (the file should have a suffix number after “plot”). Edit ‘trim_grid35.py’ in the ABM files to have your current ‘excel_export_density_plot<suffix>.csv’ selected. Then run ‘trim_grid35.py’. This will trim all monkey movements (as generated by the model) to the Yangaoping area. Make sure your .csv output file used for trim_grid35.py is in the same folder as trim_grid35.py.

b. Next, run ‘convert_csv_to_shapefile.py’ with your trimmed .csv selected (edit the desired input file’s name in ‘convert_csv_to_shapefile.py’ first). You may also edit the output shapefile’s name.

c. Then, to create a point density map and/or analyze the density plot of the points moved by the monkeys in the Yangaoping area, open ArcGIS for Desktop 9.X/10.X or ArcGIS Pro 1.X/2.X (referred to as 'ArcMap' from now on) and select the 'Add Data' button. 
Add the shapefile created in step 7b. You may receive a warning about the shapefile not having a map projection; ignore it. Your shapefile should look like Figure 8.1 (colors may vary).

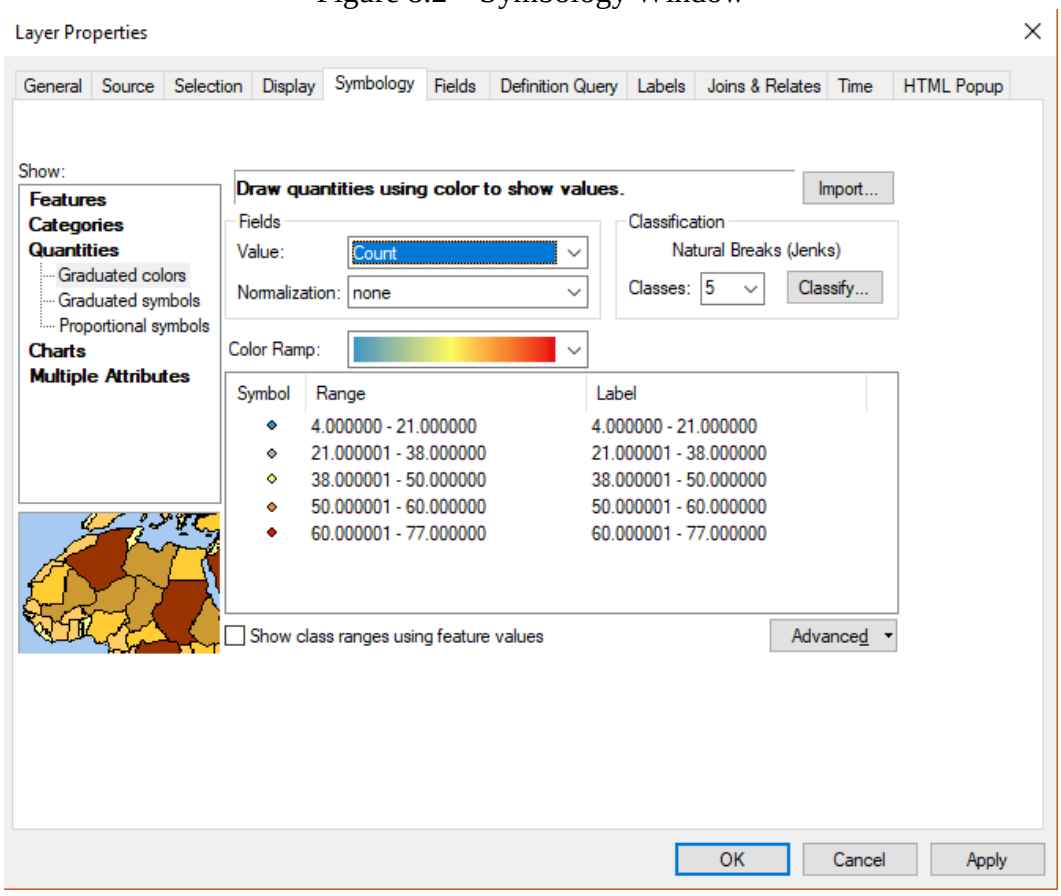
Figure 8.1 – Monkey Movement Model Shapefile Output, Unsymbolized

c. Your next step in creating a point density map is to symbolize the layer. Right-click the layer in the 'Table of Contents' and select 'Properties…' and then the 'Symbology' tab. Under the 'Quantities' category on the left, select 'Graduated Colors'. The menus on the right side of the window will change. Under the 'Fields' section near the top of the Symbology tab (not the 'Fields' tab—don't switch tabs at this point), for 'Value,' select 'Counts' from the drop down list. This means that you are coloring your map according to the different value ranges of 'Counts;' in turn, 'Counts' represents where monkey movement is most concentrated. Under the 'Color Ramp,' choose the color scheme you like best, but a graduated color scheme from dark to light or a rainbow (blue to red) is a suggestion. See Figure 8.2 for an example.

You may receive a warning about too many records—this is fine, since you have tens of thousands of records. The first few ten thousand (the points are recorded in chronological order) will suffice in creating an accurately scaled picture (Your point density map will not change by too much over multiple runs). However, if you have a fast computer and would like to use the entirety of your model run, you can change the maximum number of records handled in ArcMap's settings.

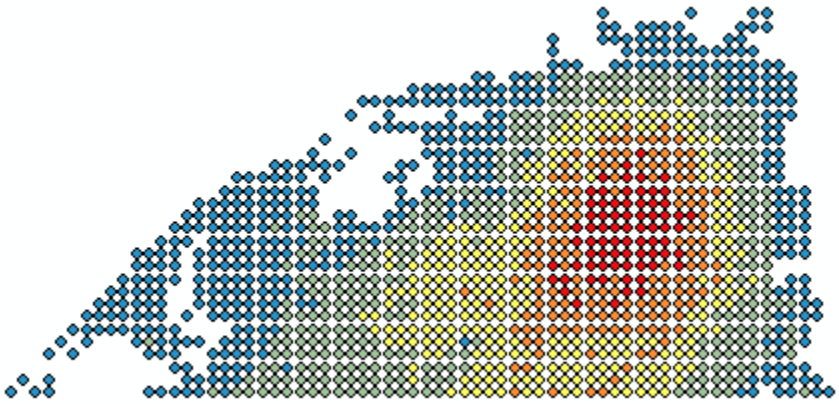
Leave the breaks however you wish. By default, there will be 5 classes in the Natural Breaks (Jenks) classification.

Figure 8.2 – Symbology Window



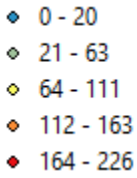
d. Click ‘OK’. Your map should look like this:

Figure 8.3 – Initial Point Density Map of Monkey Movement



And the legend should look like this (note that the numbers you get may be different from the numbers shown below, particularly if you have set the model to run for longer than a few years or have more than ten to twenty monkey families set in fnnr_config_file.py):

Figure 8.4 – Initial Legend in Table of Contents



These numbers represent how many times monkeys have "stepped" on that coordinate (remember that this is the top 35% of an 85 x 100 grid, or in other words, an 85 x 35 grid, to represent the top part of the reserve—the Yangaoping region). This legend is slightly off; 0 - 20 should be shown as 1 - 20, since 0 is really white space. Use the symbology tab (edit the 'Label' field by double-clicking '1 – 20' on the right) to change this.

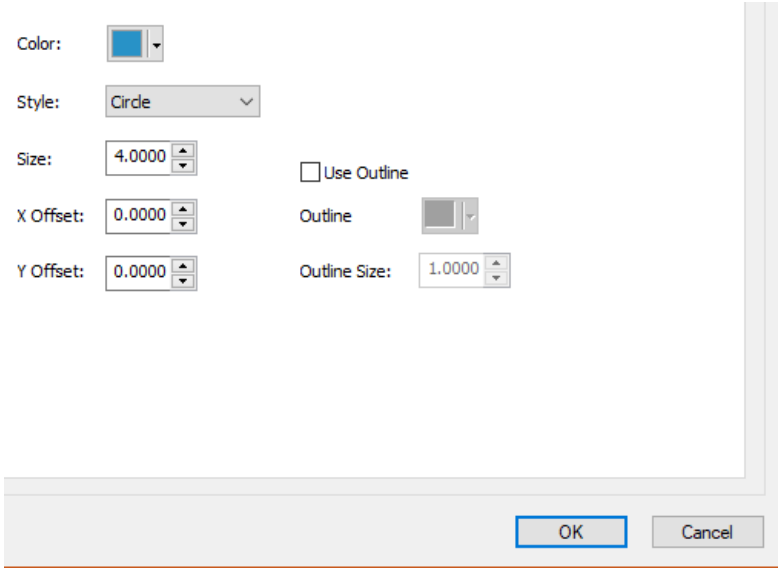
Finally, you may change the look of the point density map to be much brighter and more cohesive.

Figure 8.5 – Final Point Density Map Appearance

e. To edit the appearance of the map, double-click on a symbol, such as , in the legend in the Table of Contents (see Figure 8.4). A window titled 'Symbol Selector' will pop up. Select 'Edit Symbol…' and another window, the 'Symbol Property Editor,' will appear.

Uncheck the 'Use Outline' box as shown in Figure 8.6 and press 'OK'. Do not change the color scheme. This will generate something that looks like Figure 8.5.

Figure 8.6 – Outline



9. PREPARE (optional) – Learn how to prepare new household buffer layers in ArcMap/ArcGIS Pro.

The model can be run as is, and two different .txt versions of each buffer for a human settlement land type are included. For example, one can use either 'hh_ascii400.txt' or 'hh_ascii800.txt' in model.py, which each loads a region on the FNNR of human settlements near Yangaoping buffered at 400 m or 800 m, respectively. The four human settlement land types are household (hh), PES (payments for ecosystem services areas), farm (farmland), and forest (managed forests).

These human settlement point layers were originally taken from a 2016 household survey conducted by Yang et al. (2016). Contact Shuang Yang for further details. The 2016 survey only contains the point locations of the households of two villages—these are the two villages closest to the Yangaoping region, and the same households for which we have resource-gathering data (which in turn are used in the visualization model).

Using ArcMap, buffers can be created around these households and imported into the model. The buffered regions for 400m and 800m around these households have already been imported. However, if you would like to create custom buffer sizes, or are interested in the process used to create the ASCII files imported into the model, the process is as follows:

1. Select Relevant Point Data – Only the two villages closest to Yangaoping from the 2016 household survey were imported. The 2015 household surveys contain the locations for many more villages around the boundary of the FNNR; however, we do not have resource collection data for these villages. Furthermore, this can be done with the household, PES, farm, or forest (managed forest) layers.
2. Run the Buffer tool in ArcMap on the selected point .shp layer that portrays all of the households, etc. (files are found under Data on github), and choose the distance in meters (The distances used for existing layers are 400 and 800 meters, respectively). Leave other fields with their default values. The buffer tool creates a polygon that outlines a given radial distance around points in a layer.
3. Run the Dissolve tool on the buffered layer to merge the buffers for each point into one large buffer. Leave all fields at default values.

4. Run the Polygon to Raster tool on the buffered layer to convert the buffers into rasters.
5. Run the Extract by Mask tool, using a raster shapefile cut to the same size as the FNNR boundaries as the mask; examples you can use for this are the same files you use for Maxent analysis, such as the DEM or slope. Also, before you run it, change the Processing Extent of the layer (leave the Snap Raster option alone) and Raster Analysis (both cell size and mask) of the later to match the mask layer you chose. Do this by clicking the "Environments…" button in the main Extract By Mask tool window, next to the run button.
6. Aggregate (tool name) the layer by a factor of 10. This changes the effective ASCII width and height (or spatial resolution) from approximately 870 x 1000 (30 m resolution) to 87 x 100 (300 m resolution—this is the final spatial resolution of the model represented). This is done because the visualization will run much more slowly if the resolution is any higher.
9. Use the Raster to ASCII tool to convert your file into a .txt layer that can be imported into the model. If converting a buffer of 1000 m from the PES layer from the 2016 survey, for example, try to follow the naming convention of 'pes_ascii500.txt'. If you did this correctly, when opening the layer, you should see the dimensions as approximately 87 x 100 (slightly off is okay). Make sure the .txt file is saved in the same directory you have your other model Python project (.py) files in.

After this, you can see your new layer by changing the variable file near the top of model.py (In this case, you might change the variable pes_file from "pes_ascii200.txt" to "pes_ascii500.txt").

The model should run as a "normal_run" out of the box, but if any environment input files (such as aggveg60.txt) are replaced or changed, be aware that the model must be run as a "first_run" every time an environmental layer is changed or updated. This is because subsequent "normal_run"s of the model will use a grid loaded from memory in order to run faster, rather than generating a new grid each time and saving it (as is done on "first_run").

10. THANKS – Contact project developers for more information.

If you run into an error while running the code or have other questions beyond this document, contact project developers for more information. Also, the 8/12/18 version of the code is the most extensively tested version and the version published for J. Mak's thesis, but it uses dummy human data instead of live human data from the FNNR, and does not include GTGP simulation.

Project Faculty Supervisor – Dr. Li An
Project Developer – J.Mak(email via GitHub)

You may also directly comment on the Github repository at https://github.com/jrmak/FNNR-ABM-Primate.

Special thanks to all involved in the FNNR Project and who contributed data (available at http://complexities.org/pes/people/).

References:

Yang, Y. Q., X. P. Lei, and C. D. Yang. 2002. *Fanjingshan Research: Ecology of the wild Guizhou snub-nosed monkey (Rhinopithecus bieti)*. Guiyang: Guizhou Science Press.

Yang, Shuang. et al. (2014). “Evaluating the Impacts and Feedbacks of Payments for Ecosystem Services.” *AAG Annual Meeting*. Presentation.