**Getting Started with the FNNR-ABM-Primate Project**

Note: The FNNR-ABM-Primate project (this project) 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.

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

8. THANKS – Contact project developers for more information.

[Note that Sections 1 through 4 are prepared for ABM/ Python beginners; experienced Python modelers can download the Mesa and Matplotlib libraries and then jump to Section 5]:

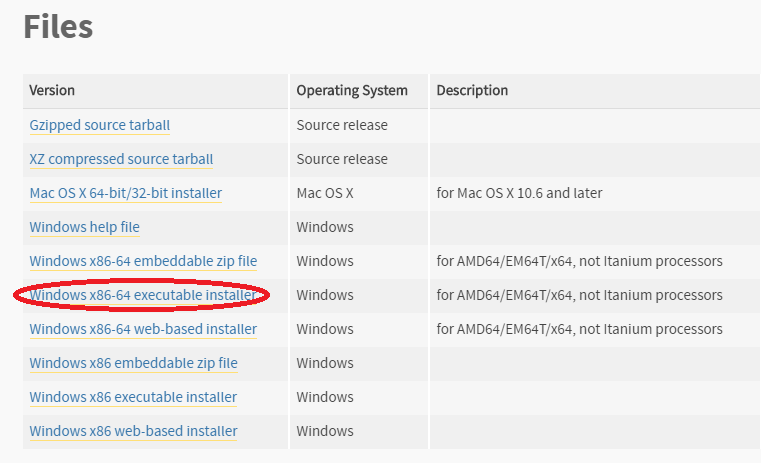
1. INSTALLATION - Have Python 3+ installed on your computer.

To download the latest version of Python, visit <https://www.python.org/>. At time of writing, [Python 3.6.1](https://www.python.org/downloads/release/python-361/) is the latest version, though again, 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.



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, openpyxl, 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

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), Numpy (for a variety of numerical expressions or generations), Six (for wrapping over differences between Python 2 and 3), Tqdm (progress meter/Jupyter Notebook-related), Matplotlib (for plotting, and more). The user will likely not directly access these libraries when working with Mesa, but they should be aware of what the libraries do.

Matplotlib – this may come with Mesa, but just in case you get an error related to this, you may need to download matplotlib separately, since it is used explicitly and separately from Mesa in the model. It helps build graphs to display Python data.

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

pip install mesa

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.

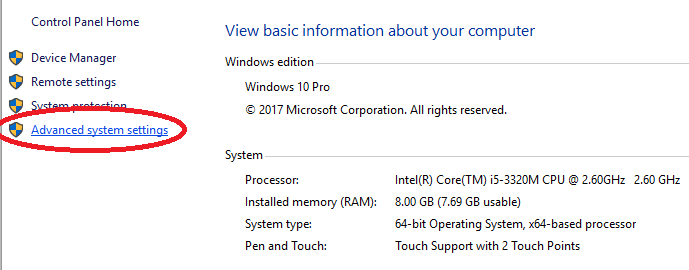
*If ‘pip’ is not recognized in the command window:*

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

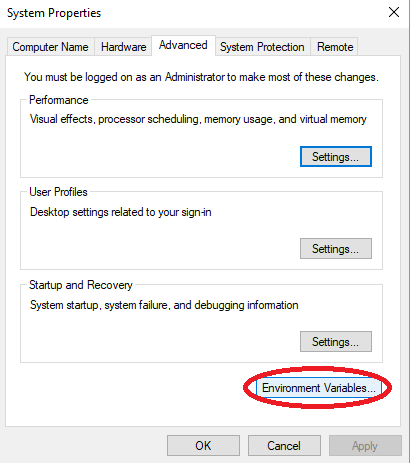
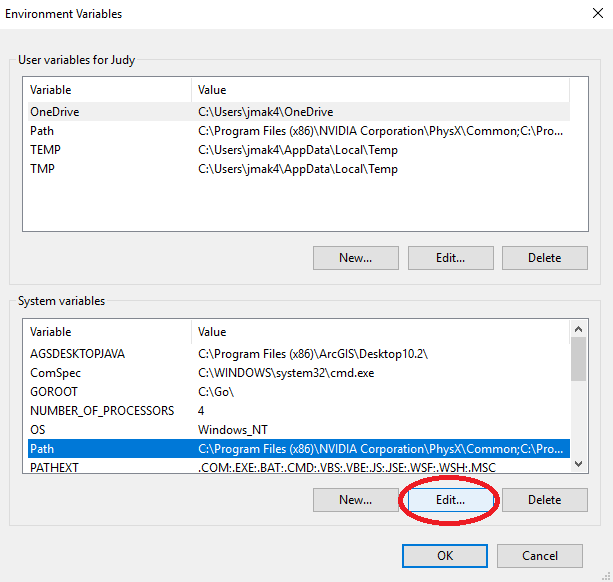
d. Select ‘Path’ under ‘System Variables’ (near bottom of the window, not the first ‘Path’ near the top), then ‘Edit...’.

Figure 3.3 – System Variables

e. Select ‘New’, then ‘Browse...’ to find where your Python installation is. Common file paths 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

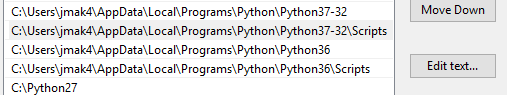
C:\Users\<YOUR USERNAME>\AppData\Local\Programs\Python\Python36\Scripts ← likely

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 file paths, 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.

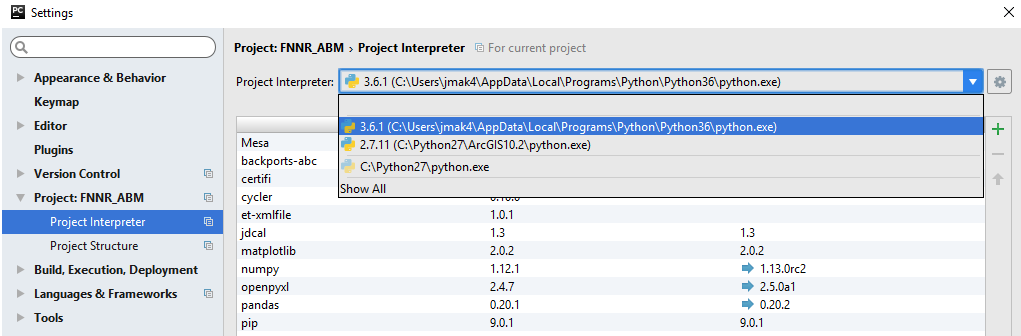
Then activate the env in the Command window:

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, openpyxl, pandas, Mesa, and more (assuming that you have successfully used pip or conda in the command line to install the libraries at this point).

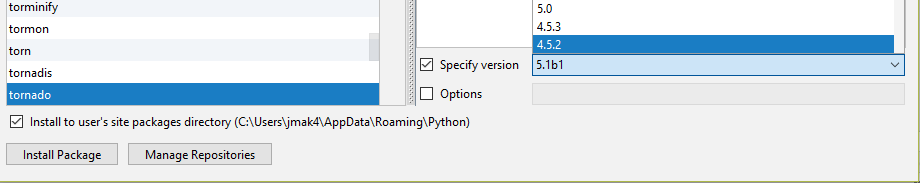
**Updating Libraries**

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: 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 (may not be needed)



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.

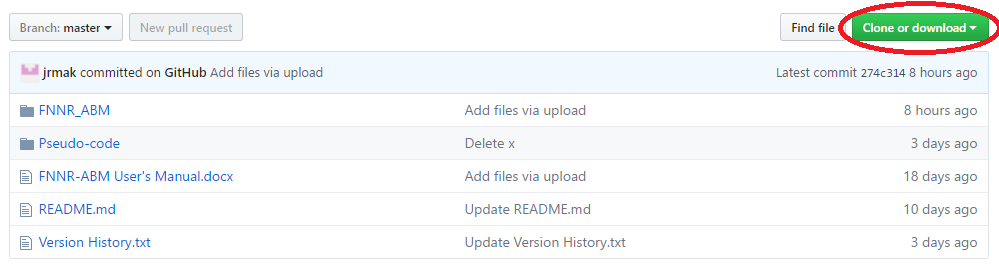
4. GITHUB – Download and unzip the FNNR-ABM-Primate project files from Github.

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 7zip. The name of the folder should resemble ‘FNNR-ABM-Primate-master.’

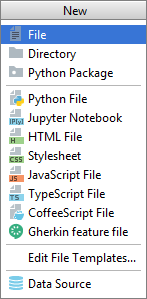
Once these files are downloaded to your computer (you probably only need the code files within FNNR\_ABM-Primate) and unzipped, move them to the appropriate location on your hard drive. If you are running a very basic setup with IDLE (the default Python IDE), it is possible to keep these files in your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

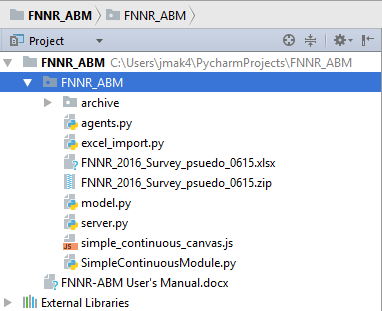
Note that in general, if you would like to start a new .py file to code and save, select ‘Python File’ and not the generic ‘File’ type at the top of the menu (see below).

Figure 4.2 – New ‘Python File’ (not the selected ‘File’)



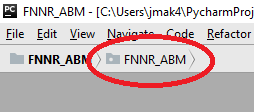
From the FNNR-ABM-Primate file you have on your computer, copy and paste your code (.py) files into PyCharm under your Python Package folder directory. Also paste the latest copy of the unzipped .xlsx Excel data file. The result on the left-hand window should look similar to this:

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.

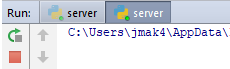
The code should now be visible whenever a module is opened from the left-hand side directory. To run the code, open server.py (it must be this module for this project specifically) and press Shift + F10 or run it manually under the Run… option at the top toolbar, or right-click the tab at the top where 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. xx) 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 through slider inputs; and

- Simulate and compare agent movement in a visualization given different input environmental grids.

- Understand monkey habitat use behavior from a bottom-up approach

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 both submodels actually run under both modules), 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.

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); they 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)

Below are the modules that comprise the model’s code.

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. The equivalent of running server.py, especially if you have a slow computer, is to visit the web application, currently hosted at: <http://abm-webapp-abm.193b.starter-ca-central-1.openshiftapps.com/>

**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 uploaded to GitHub based on probability categories (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 in the same directory.

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 in the visualization.

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

Each monkey agent goes through the following events, in addition to aging and potentially switching to new age categories in each time step:

1) Both male and female – birth or death; death occurs according to mortality probabilities for each age category, as set in the pseudocode with original data from Yang, Lei, and Yang (2002);

2) Female only – possibility of giving birth every 3 years in April or September, or giving birth in the year when a child has died recently;

3) Male only – possibility of breaking off into an all-male group when enough males over the age of 10 are flagged.

**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, humans move 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. 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.

This behavior varies only with respect to the resource choice. Different resources have different frequency rates for collection and different distances to each human’s home location.

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.

**excel\_export\_summary.py** – exports age/gender structure results per step of the population sub-model to an Excel (.csv) file when graph.py is run.

**abm\_excel\_export\_summary.csv** – accompanying output file. Is appended to (expands) when graph.py is run again. This can be changed by adding or removing the erase\_summary\_() function call in graph.py.

**\*difference.py** – Creates the difference output in a .csv file (from which a heatmap or point density map can be generated) between any two given trials or averaged output of “With Humans” vs. “Without Humans” scenarios. Since it is computationally intensive to average all movement outputs from all trials, one can also choose the trials whose kappa results for each threshold most closely match the averaged kappa statistics for each threshold from multiple runs.

**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.

**excel\_export\_density\_plot\_<suffix depends on trial run>.csv** – accompanying output file. Is overwritten (refreshes) when graph.py is run again, but there are usually multiple output files (one for each trial run of each scenario). The overwrite setting can be changed by adding or removing the erase\_summary\_plot() function call in graph.py.

**\*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 agent’s walk path for each model run. To run this, you must run graph.py while the # of families attribute in model.py is set to 1 and use the excel\_export\_density\_plot\_<suffix>.csv output of that with this module; in other words, it can only follow one family-group agent at a time, or else multiple agents’ movements will become falsely correlated.

**\*trim\_grid35.py** – Once a heatmap .csv file is generated in graph.py (excel\_export\_density\_plot\_<suffix>.csv), 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 run through kappa\_batch.py.

**\*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) first.

**agg\_dem87100.txt** – An ASCII file that contains the elevation data extracted from Cindy Tsai’s DEM, which forms part of the environmental grid that the monkeys move upon. Resolution (grid size) is 87 x 100.

**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 approximately 87 x 100, but may vary slightly.

**hh\_survey.csv** – Contains the locations of resources for human agents to gather, as well as the frequency of said resource. The frequency formula in humans.py is slightly arbitrary and can be modified.

**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’; these are pre-loaded files used in the model when its setting is ‘normal\_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, and understand which variables to edit.

The code can be run in multiple ways, depending on what data you need to access. You may execute (run) one of the following modules:

**server.py** – run this module if you would like to see a visualization of monkeys’ families (small-group in terms of Yang, Lei, and Yang 2002) movement through the FNNR in a given year (73 time-steps).

**graph.py** – run this module to see graphs of the population changes 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.

To create an Excel Heatmap from the data, 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).

7. 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 400m or 800m, 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 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 on the selected layer, 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.

3. Run the Dissolve tool on the buffered layer. Leave all fields at default values.

4. Run the Polygon to Raster tool on the buffered layer.

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 (30m resolution) to 87 x 100 (300m resolution). 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 1000m 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”.)

Please 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, rather than generating a new grid each time and saving it (as is done on “first\_run”).

8. 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.

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.