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

written by Judy Mak ([jmak@sdsu.edu](mailto:jmak@sdsu.edu)) for the model at <http://github.com/jrmak/FNNR-ABM>

Glossary

FNNR – Fanjingshan National Nature Reserve  
ABM – Agent-based Model

GTGP – Grain-to-Green Program

IDE – Integrated Development Environment

OS – Operating System

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, openpyxl, matplotlib).

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

5. GOALS – Understand the project goals, Excel data file, and pseudo-code.

6. READCODE - Understand the Python code and imported libraries.

7. RUNCODE – Run the model, and understand which variables to edit.

8. THANKS – Contact project developers for more information.

In order to access and download this project (for Python beginners):

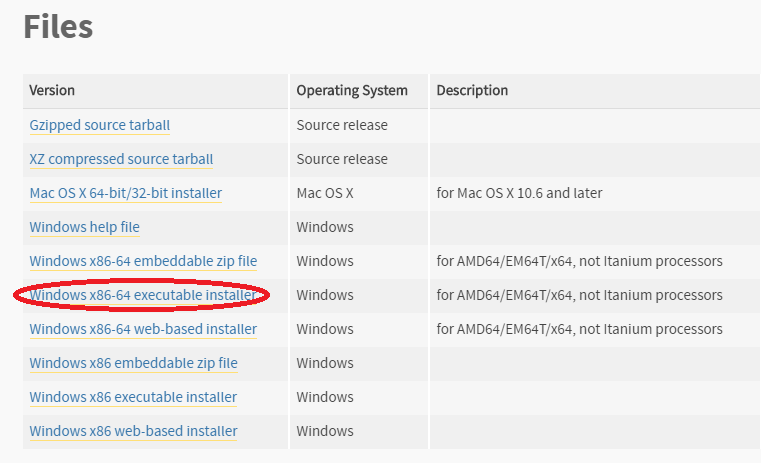
1. INSTALLATION - Have Python 3+ installed on your computer.

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 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

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), 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.

Openpyxl – helps import data from cells in Microsoft Excel or Open/LibreOffice Calc files

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), or a similar terminal on whatever OS you’re using, and type in:

pip install mesa

and when that’s done,

pip install openpyxl

If you are using conda or miniconda (or another environment/package manager), replace ‘pip’ with ‘conda’ in the above commands.

**Troubleshooting**

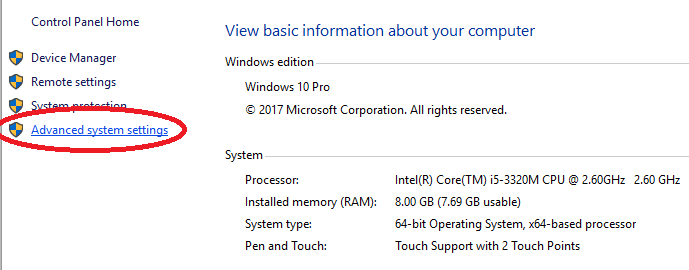
There are a number of possible error messages you can get. The instructions below diagnose them based on Windows 10.

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

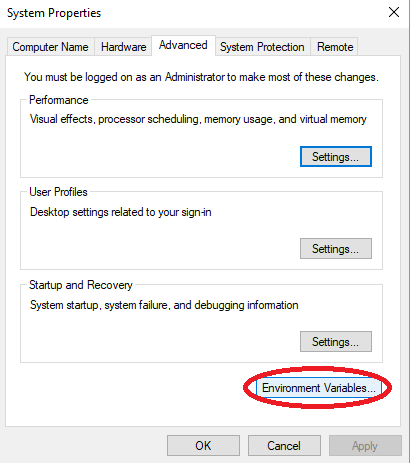
1. Set the Environment Path.

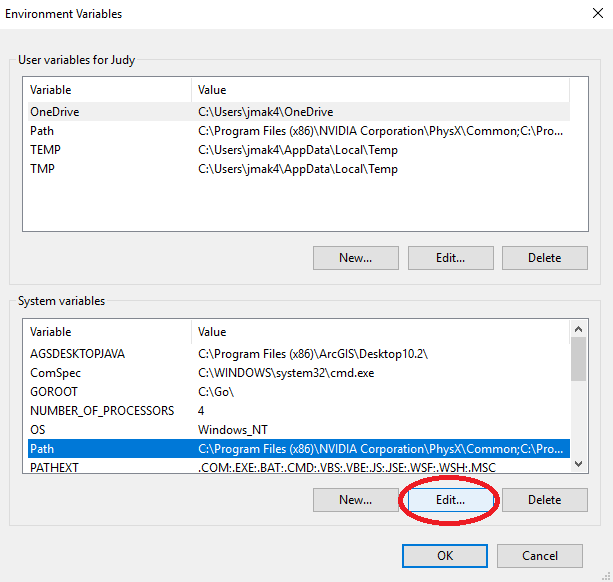
a. This PC or My Computer > Right-Click > Properties

b. Click on ‘Advanced system settings’ on the left tab.



c. Select ‘Environment Variables...’.

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

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 ← most 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 old ones in the current list.

\*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.

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:\Users\

in order for cmd.exe to look for pip in the right drive.

*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

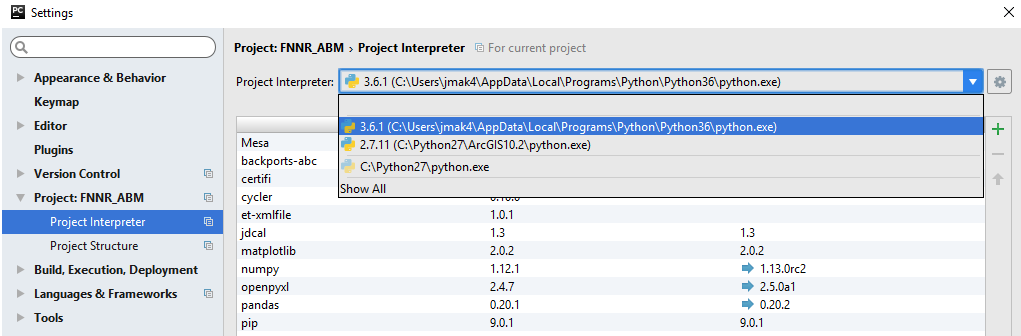
Note: ‘3point6’ here can be any name you wish, and ‘3.6’ can be changed to another version of Python.

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.



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’ve 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).

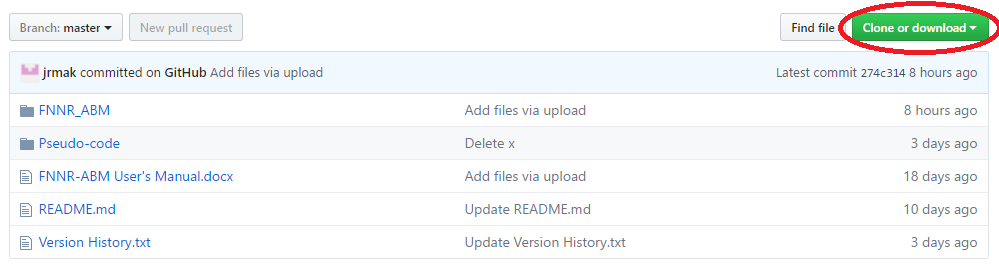
Finally, if your libraries appear to have successfully installed to the same directory that runs the desired version of Python as configured in your IDE, but you still get import errors, you may need to restart your computer.

4. GITHUB – Download and unzip the FNNR-ABM 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. While Github is a powerful suite that comes with its own commands, 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>.

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.



Make sure you are downloading the correct branch/version of the code. 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-master.’

Within this folder (also shown on the image above), the following files and folders exist:

FNNR\_ABM – the main folder containing Python code for the project. Code is written by Judy.

In this folder: agents.py, model.py, server.py, excel\_import.py, SimpleContinuousModule.py, simple\_continuous\_canvas.js, and the latest zipped version of the source data files.

\*simple\_continuous\_canvas.js and SimpleContinuousModule.py (which uses the Javascript file simple\_continuous\_canvas.js) are part of the Mesa project and are not typically edited throughout the project.

These files are explained in readme.md on the main page of the repository, directly below the file listing.

Ask Dr. An or Judy for the password to the zipped data files, or ask Shuang for the unzipped version (which should be identical to the latest uploaded version on Github).

Pseudo-code – a folder containing various versions, arranged by month/day in 2017, of pseudocode that the main code is based off of. Pseudocode is written by Shuang.

FNNR-ABM User’s Manual.docx – this Word document.

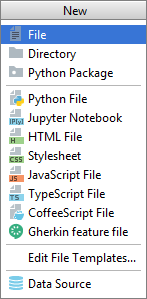
README.md – contains the project description. Initializes with each Github repository.

Version History.txt – contains dated project updates and version descriptions. Also details Judy’s reported working hours, and a to-do list for coding.

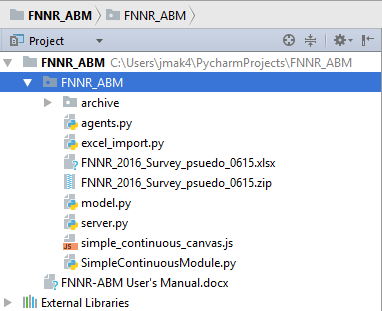
Once these files are downloaded to your computer (you probably only need the code files within FNNR\_ABM) and unzipped, move them to the appropriate location on your hard drive. If you are running a very basic setup with IDLE (the default Python IDE) it is possible to keep these files in your Downloads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

Using PyCharm, create a new PyCharm project by File > New Project… in the upper left corner. Name the project however you want (I used ‘FNNR\_ABM’) and place the project however 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, even if you’ve already named your project that.

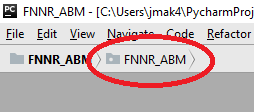
(Note that in general, if you’d like to start a new .py file to code and save, select ‘Python File’ and not the generic ‘File’ type at the top of the menu.)



From the FNNR-ABM 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:



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:



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 specifically) and press Shift + F10 or run it manually under the Run… option at the top toolbar. Afterwards, server (or server (1) here) should be the default module set to run next to the green buttons on the top right corner, as pictured below; you can also click the green triangle to run.



(Note: Demo to be shown once simulation is working again)

5. GOALS - Understand the project goals, Excel data file, and pseudo-code.

The goals of the FNNR project are as follows:

- Simulate GTGP conversion of land parcels;

- Simulate how GTGP conversion will affect household demographics;

and visualize the above components through step-by-step interactive models, graphs, and text/tables.

The Excel data file is mainly important for the variables it contains for each household that can be used in the model. Important variables found in the data file (named in the format “FNNR\_2016\_Survey\_psuedo\_XXXX.xlsx”, with XXXX representing a month and day) include:

Global atttributes – year (represented by each ‘step’ in the model)

Household attributes – unique id (part of Mesa, not used here), model (not directly referred to), household id (separate from household row of Excel file), administrative village (placeholder attribute used to draw it in the web simulation), step counter

Land parcel attributes – unique id, model, household id, household row, land parcel position (coordinates), GTGP status, age of head of household, gender of head of household, educational status of head of household, area of dry GTGP land, area of GTGP rice paddies, total dry land area, total rice paddie area, land type, travel time from household to land parcel, plant type, non-GTGP output, pre-GTGP output, step counter

Individual attributes – unique id, model, individual id (separate from unique id, which is unused), age, gender, educational status, marital status, past household id (if applicable), non-GTGP area (a mistake, but it doesn’t affect the rest of the model), step counter, initial age, local off-farm income, birth rate, interval since last time they gave birth (remains static if it doesn’t apply to that individual), minimum birth interval between births (pre-set in pseudocode, remains static), death rate, death flag, marriage, flag, match probability (pre-set in pseudocode, remains static), immigration marriage rate (pre-set in pseudocode, remains static), migration status, work flag, retirement flag, husband’s individual id (remains static if it doesn’t apply to that individual), and past individual id in case of marriage (remains static if it doesn’t apply to that individual)

6. READCODE - Understand the Python code and imported libraries.

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

The way the model works is that every step represents one year, starting in 2016 (or 2014, depending on what version of the data you are using.) The model then “runs;” what is really happening is that each step, a multitude of functions execute. The function parameters are assigned by the pseudo-code (which in turn is based on logistic regression); they also utilize some random generators in limited situations

This model runs on the Mesa 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)

Green variables are imported directly from the Excel data file or pre-set in the pseudocode

Red variables are calculated within that function

Purple variables have been calculated earlier in the model

A good module to look at the code with first is agents.py, which is most similar to the pseudocode.

**agents.py**

This module contains many empty lists at the top of the code. As the model runs, certain functions will fill these lists with data. The data will then be collected by other parts of the model.

class HouseholdAgent – this class simply acts to update itself every step (year) In the early stages of the code, it served as a superclass to Individual and Land agents, and was much more active; most of this code has since then been moved to the IndividualAgent class.

class LandParcelAgent – this class tracks all the land parcels in each household.

function “output” – sets unit prices depending on plant types, and calculates total land income and GTGP net income from land area (determined by adding up area subsets) and unit prices

function “gtgp\_participation” – calculates probability of each house participating on GTGP based on factors such as head of household demographics and travel time to get to land parcels

function “non\_gtgp\_count”/“gtgp\_count” – updates # of GTGP vs. non-GTGP land parcels in the reserve every step using the gtgp\_enrolled flag

function “step” – ties the other functions together and executes yearly (that is, every step), as well as compiling the data to update the Excel file. It uses landpos to differentiate between 2014 and 2016 datasets; 2014 land parcel objects do not have a set of coordinates assigned to them, since they are not featured in the web simulation, whereas 2016 ones do.

class IndividualAgent – this class concerns each individual in each household, as well as functions that create, destroy, or move individuals between or out of households. If a variable is assigned to 0 in \_\_init\_\_, its value will be defined later; otherwise, values are either using calculated estimates or are from the pseudocode. Values may be changed to suit desired inputs. This class’s functions are easy to track because they are the same functions that are largely pre-defined in the pseudocode.

It is important to differentiate ‘hh\_row’ from ‘hh\_id’ here. There are 94 households. ‘hh\_row’ refers to either 0-93 (the default indexing for each household, starting from 0), 1-94 (the default indexing for each household, if starting from 1), or 3-96 (the default indexing for each household, if looking at the rows in which they are found in the Excel file). This is why many references to ‘hh\_row’ may contain a +/- 1, 2, or 3, depending on the situation.

‘hh\_id’, on the other hand, refers to assigned household numbers from 1-169 (some numbers are skipped; there are only 94 households total) as defined in the Excel file.

function “create\_initial\_migrant\_list” – this is called before the rest of the model runs. It sets up the initial migration for the reserve, so that at step 0, there are approximately 90 pre-existing out-migrants (1 for most households, as pre-defined in the Excel file).

function “match\_female” – this loops through both single females and single males, and is the “marriage” function.

function “immigration\_marriage” – in case of a marriage, there is a 3% chance that an additional marriage will take place that includes an immigrant. This effectively adds an additional individual to the reserve.

function “birth” – this creates additional individuals to add to the reserve according to the last time the mother gave birth. These individuals are assigned a new unique ID, and will later go on to age, change work status, out-migrate, re-migrate, possibly marry, possibly give birth or help give birth, and die.

function “death” – this removes certain aged individuals from the reserve according to a pre-defined general death rate.

function “youth\_education” – this “enters” and “exits” individuals between the age of 5 and 19 in and out of school by setting their work status as they age.

function “aging” – this is similar to youth\_education; it “enters” and “exits” individuals between the ages of 15 and 65 from the workforce, and updates their age individually (as opposed to the model’s step\_counter, which ages the model itself).

function “out\_migration” – this function calculates (and executes, if it meets a pre-defined threshold) the probability of migration each step from each household (though households may only have one migrant out at a time). It also sets certain income parameters such as income\_local\_off\_farm and remittances for each immigrant.

function “re\_migration” – this function calculates the chances of each household’s immigrant returning based on the number of years they have been out and the immigrant’s age.

function “step” – ties all the functions together and executes them yearly (each step); also prepares data for exporting to the excel\_export\_household output file.

**excel\_import.py/excel\_import\_2014.py**

Reads data from the original input data Excel (.xlsx) file for the model to use

**model.py**

Whereas agents.py defines agent behavior, this module actually creates the agents and runs the model.

All functions defined before class ABM is defined (lines 1-178 of the module) are for setting up the graphs in server.py.

class ABM – Adds all agent objects to the model, sets up the model, and runs it.

function “make\_birth\_agents” – ensures that new agent objects created by birth are added to the model correctly

functions “return\_x”, “return\_y”, “return\_lp\_pos\_list”, “determine\_hhpos”, “determine\_landpos” – calculates the coordinates (landpos) of, and adds, the land parcels to the web simulation

function “make\_household\_agents\_20XX” – XX can be 14 or 16; sets a household and its attributes according to the Excel data input file for each household

function “make\_individual\_agents\_20XX” – XX can be 14 or 16; sets an individual and its attributes according to the Excel data input file for each individual

function “make\_land\_agents\_20XX” – XX can be 14 or 16; – sets a land parcel and its attributes according to the Excel data input file for each parcel

function “step” – each step, this collects the data for the model to update itself with as it runs.

**server.py**

\*This is the main module to execute/run for output data!

This module mostly creates the graphs using the matplotlib Python library, and writes the Excel reserve summary output files.

Enable or disable the line ‘plt.show()’ to show or hide the graphs (since there are many, and you may not need to see them every time you want to run the model).

**simulation.py – uses legend.png image**

\*This is the main module to execute/run for the web simulation!

This module sets up and runs the web simulation, which shows the changes in non-GTGP 🡪 GTGP land parcels over time. The land parcels (and households in the background) are represented by dots, which are explained by a legend. These dots are accurate in relation to each other’s positions.

**SimpleContinuousModule.py**

This module is part of Mesa, and was not edited by me at all. However, it must be included in local files in order for server.py to run.

**simple\_continuous\_canvas.js**

This file is part of Mesa, and was not edited by me at all. However, it must be included in local files in order for server.py to run.

**excel\_export\_household (+\_2014).py/excel\_export\_summary** **(+\_2014).py**

Manages writing and labeling for exported data files; see below

**FNNR-ABM\_export\_household\_20XX.csv** - XX can be 14 or 16 (2014 or 2016)

This Is the Excel export file for the reserve, and contains mean data from all households, such as average land parcels

**FNNR-ABM\_export\_summary.csv** - XX can be 14 or 16 (2014 or 2016)

This is the Excel export file for households, and contains data such as each household’s.

7. 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 need graphs and exported Excel data summary files (excel\_export\_summary.csv for the entire reserve, or excel\_export\_household.csv for each household).

\* To edit the amount of steps the model runs for, edit the line that currently reads “for i in range(81)” in server.py (line 25 in PyCharm).

\* If you want excel\_export\_household.csv to display data from each household for more than 5 steps, please edit the ‘step’ function in agents.py (line 906 in PyCharm) and change the ‘6’ to the number of steps you want plus one (so if you want it to run for 20 steps or years, change the ‘6’ to a ‘21’). Please note that this means the model will take longer to run, because that Excel file must write 94 additional rows of data (for 94 households) for each step that you add.

\* If you do not want the graphs to pop up when you run the model, disable the line “plt.show()” in server.py (line 215 in PyCharm). These are the same graphs that would display if you created them in Excel based on excel\_export\_summary.csv.

\* If you would like to change a single attribute in the model—such as birth rate or death rate—this can usually be found in the \_\_init\_\_ function within the appropriate class (IndividualAgent or LandParcelAgent) in agents.py.

**simulation.py** – run this module if you need to access the graphic web simulation for non-GTGP 🡪 GTGP land parcel conversion. This will launch a web browser that will show you an interactive step-by-step GTGP-status progression for the land parcels.

\* Click slowly to advance the steps, or run several years at once (much faster).

\* Do not use the ‘reset’ button within the web browser while running the simulation; it will not probably reset the label. Instead, re-run simulation.py instead.

8. THANKS – Contact project developers for more information.

If you run into an error while running the code or have other questions and this document does not help, contact project developers for more information.

Project Faculty Supervisor – Dr. Li An ([lan@mail.sdsu.edu](mailto:lan@mail.sdsu.edu))

Project Lead, Data File Keeper, and Pseudo-code Developer – Shuang Yang ([shuangyang@gmail.com](mailto:shuangyang@gmail.com))

Code Developer and Documentation Writer – Judy Mak ([jmak@sdsu.edu](mailto:jmak@sdsu.edu))

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

Special thanks in this part of the project go to the rest of the Geography Department at SDSU, SDSU undergraduate recorders who collected data and compiled the data files, as well as those at FNNR who helped complete the surveys.