Running Guide

The *pdpython* simulation comes in two variants; one that **produces a visual demonstration** (slower, step by step output), and one that runs **simulation parameters in batches** (no visualisation, much faster and larger-scale). Henceforth these are referred to as the **Visual** variant and the **Batchrunner** variant. There are also two versions of each of these variants, one for the simulations that involve random partner changes on a graph, and one that involves agents on a static 2D grid. ***We will outline how to run the latter of these two first, and then the former.***

UPDATE 25/01/23 – *Setup Notes*

Compatibility has been tested with *Python 3.10.4* and the packages listed in *requirements.txt*. There may be some initial errors printed to the terminal when *fixed_random_run.py* or *fixed_random_batchrun.py* are used (re: usage of literals and the depreciation of the Mesa Batchrunner class), but the code should still run as of January 2023. These may need to be replaced in future. To download and run any of the following files from the terminal, follow these steps to set up a virtual environment (Windows only: see [Documentation] for macOS alternative):

- Download the code folder and extract
- Cd into the code folder at the level that contains the python scripts (within *pdpython_model*)
- Create a virtual environment to run the code in by using each of these commands in turn:

  py -m venv env

  .\env\Scripts\activate

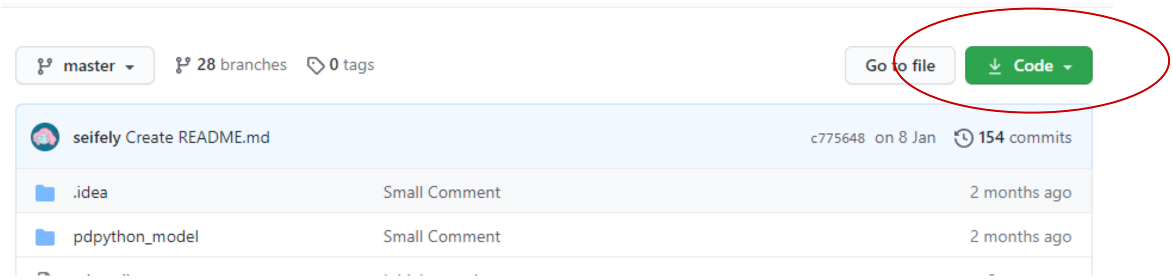
  py -m pip install -r requirements.txt

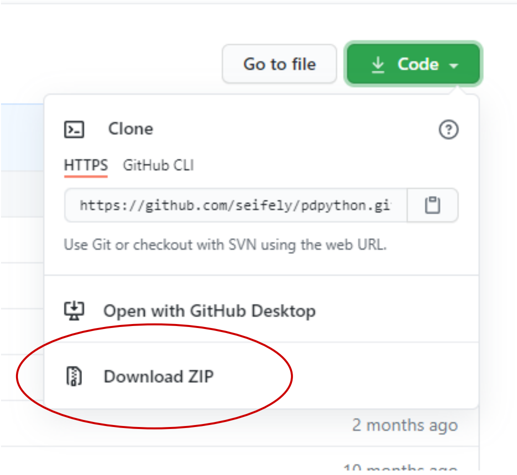
- Then run each python script using:

  python [filename.py]

To download:

You can download the code to run from: https://github.com/seifely/pdpythonRunAtHome



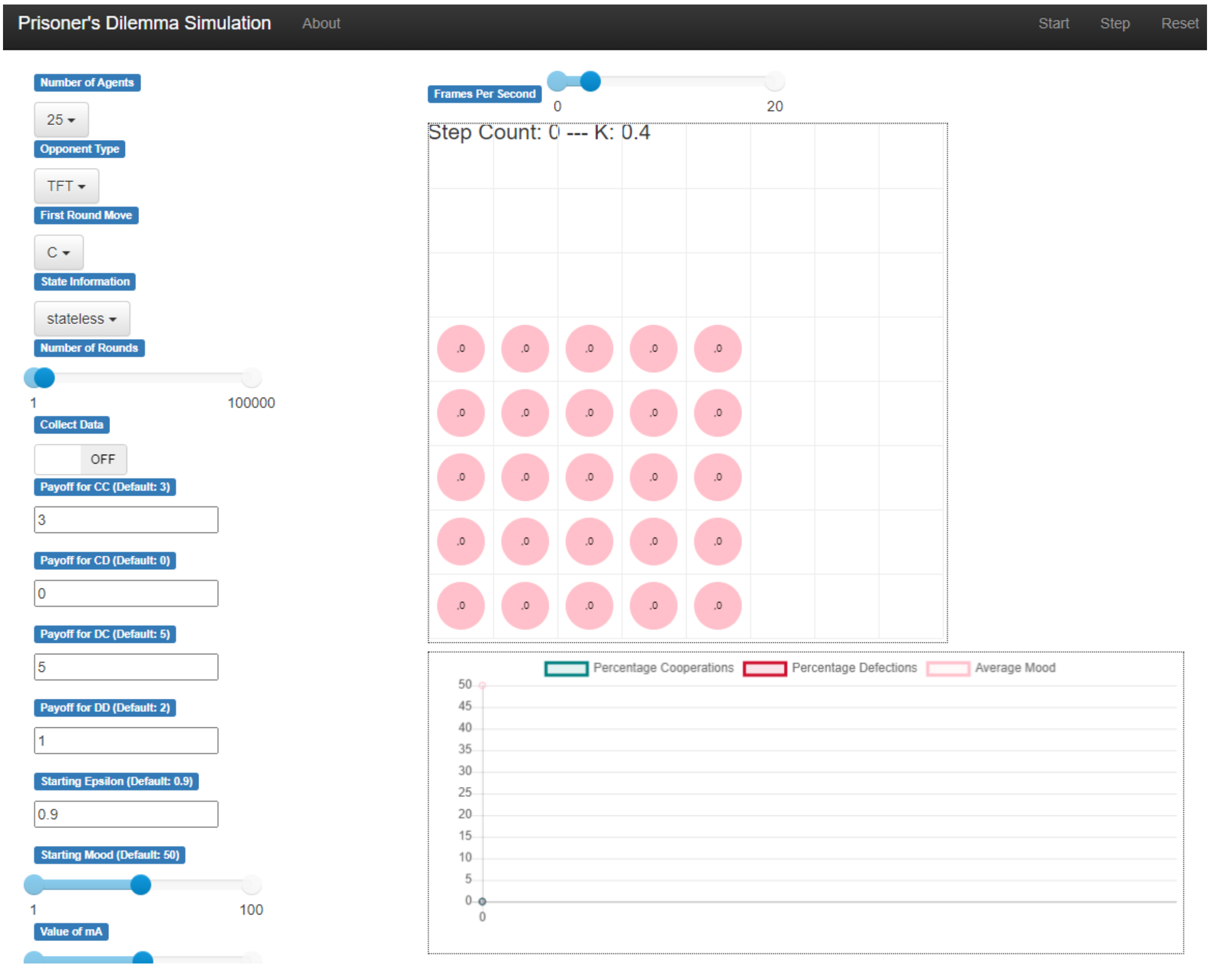


Running the 2D Lattice Simulation Version

The visual version of the code does **not** require inputs to be manually inputted into the code itself but can take inputs after it has already been run. This is the simpler, but less powerful version of the code as it currently stands.

Whether running from the command line or from your IDE such as PyCharm, simply run the file 'moody_run.py'. Your OS may prompt you to select a browser window to display the visual server on; both Google Chrome and Microsoft Edge have previously worked successfully, other browsers may display differently.

You will then be presented with this window.



All input options are displayed on the left-hand side, with **Start**, **Step**, and **Reset** in the upper right corner. Output displays are central. ***WARNING:*** Changes to inputs will not take effect until the ***Reset*** button in the upper right corner has been clicked, which will cause a refresh of the simulation. ***WARNING:*** No data is permanently saved to an outputted .CSV file in this variant of the simulation unless the 'Collect Data' input slider is set to 'ON'.

Once your input options have been selected and the sim refreshed, either click **Step** to make the simulation take a single timestep forwards, or click **Start** for the simulation to begin running timesteps at the speed approximately dictated by the *Frames Per Second* input option in the top middle of the screen. Data output updates as appropriate.

Central data outputs show the *K* (Cooperation Index) of the current payoffs given, and the step count as it is currently. The central grid shows agents in their grid positions – once *Step* has been pressed for the first time/*Start* has been clicked, agents will display their individual strategies as text within each agent body (circle) and their current score. The colour coding of agents displays their most commonly played move that turn: Green for Cooperation, Pink for an Equal Balance of Both Behaviours, and Red for Defection. **By default, agents will spawn in a checkerboard pattern, with 'black' tile equivalents spawning mSARSA agents, and 'white' tile equivalents spawning whichever opponent is given by the *Opponent Type* input variable.**

New input parameters can be added to this screen in the 'moody_server.py' file: for more on how to do this, please consult the original Mesa Visualisation documentation here.

Some of the input parameters for the simulation, such as memory size for learning agents and whether this memory has paired 'chunks' in it or single values, are not available to be altered in the visualiser inputs; this is because they are linked, and changing one without changing another suitably will cause the simulation to crash or simply not run altogether.

To run the Batchrunner variant:

The Batchrunner variant relies on hard-coded variables that can be added in as lists, which it iterates through in all possible permutations to produce larger datasets with different combinations of input parameters. For example, with the payoff for *R* being delineated as "CC": [3, 3.5, 4], the Batchrunner will run *x* iterations of the simulation, outputting data to .csv files, per each of those *R* payoffs. If no particular parameter/s are specified in the batchrun section, the simulation will use whatever is hard-coded as the default for that parameter. Within these lists, commas separate each parameter value.

These parameters can be found at the very bottom of the file 'batch_run.py' within the dictionary named 'br_params'. You can see that I have commented out many past parameter variations; any input parameter that is specified as an input variable to *'def __init__('* at the very top of the code in this file can be altered in the batch runner (as well as any new ones you might add).

Descriptions of each parameter alterable here can be found in the **Appendix**.

The other variables that must be altered for the **Batchrunner** are the number of times you want each parameter combination to be simulated (*x*), and the time step length of each simulation! These can be set under the 'br = BatchRunner(PDModel,…' section of code, as seen in the screenshot below. The value *x* is set in ‘iterations’, and the value for the timesteps is set in ‘max_steps’. Currently these are set to 5 and 10,000 respectively.

```
br = BatchRunner(PDModel,
                 br_params,
                 iterations=5,
                 max_steps=10000,
                 model_reporters={"Data Collector": lambda m: m.datacollector})
```

The output .csv files use the file name specified in the ***filename input***, which is hard-coded on approximately line 587. The default format for this filename outputs (concatenated):

‘moodygrid_ + *the PD payoffs used* + ‘start’_*the initial agent behaviour* + ‘mood’_*the starting mSARSA mood value* + ‘eps’_*the value of moody_epsilon* + *number of agents* + ‘mA’_*the mSARSA mA value* + *the mSARSA statemode* + *the opponent type* + ‘msarsa’ + *the simulation iteration number*[1]

Running the Random Graph Partner Changes Version

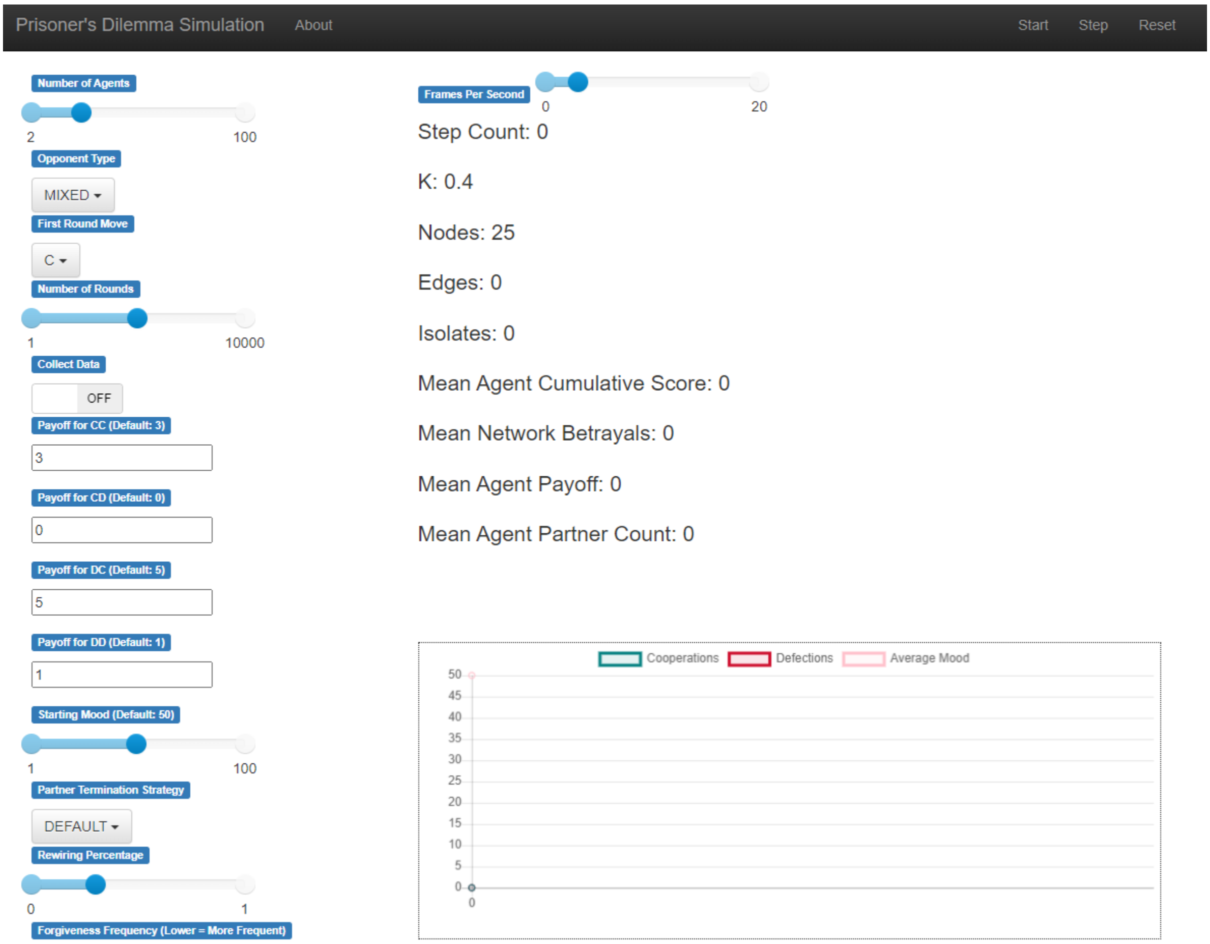
To run the Visual Variant:

The visual version of the code does **not** require inputs to be manually inputted into the code itself but can take inputs after it has already been run. This is the simpler, but less powerful version of the code as it currently stands.

Whether running from the command line or from your IDE such as PyCharm, simply run the file ‘fixed_random_run.py’. Your OS may prompt you to select a browser window to display the visual server on; both Google Chrome and Microsoft Edge have previously worked successfully, other browsers may display differently.

You will then be presented with this window.

[1] This number increases by 1 each time the simulation is run to prevent the outputted files from overwriting each other. It can be changed/reset to zero within the accompanying .csv file, ‘filename_number.csv’, within the code folder.



All input options are displayed on the left-hand side, with **Start**, **Step**, and **Reset** in the upper right corner. Output displays are central. ***WARNING:*** Changes to inputs will not take effect until the ***Reset*** button in the upper right corner has been clicked, which will cause a refresh of the simulation. ***WARNING:*** No data is permanently saved to an outputted .CSV file in this variant of the simulation unless the 'Collect Data' input slider is set to 'ON'.

Once your input options have been selected and the sim refreshed, either click **Step** to make the simulation take a single timestep forwards, or click **Start** for the simulation to begin running timesteps at the speed approximately dictated by the *Frames Per Second* input option in the top middle of the screen. Data output updates as appropriate.

Whenever the graph updates in this version of the simulation, a new window will be created displaying the new graph state, at the interval dictated by the 'Restructuring Frequency' input option. ***WARNING:*** Allowing too many of these popups to remain open over time may cause the simulation/your computer to crash due to all the physics rendering involved with the graph display. Regardless of whether you have turned 'Collect Data' on or not, the simulation will output the initialisation and end graphs to the current folder as both .PNG and .HTML files (using the file name specified in the ***filename input***, which is hard-coded on approximately line 655. The default format for this filename outputs (concatenated):

'restructure'_ *the graph restructure frequency* + *the PD payoffs used* + 'round'_*the graph restructure frequency* + 'mood'_*the starting mSARSA mood value* + 'graphprob'_*the initial graph* *connectivity likelihood* + *number of agents* + 'mA'_*the mSARSA mA value* + *the mSARSA statemode* + *the opponent type* + *the simulation iteration number*[2]

WARNING: Repeatedly clicking the 'Step' button too fast will often cause lag and potential crashes/errors when the network size is anything over around 5 agents. Please be gentle.

WARNING: Do not open any of the output .CSV files whilst the simulation is still running, as it will cause the sim to crash – the sim requires to read/write to each file as each simulation step occurs, and this is not possible if the file is open.

New input parameters can be added to this screen in the 'fixed_random_server.py' file: for more on how to do this, please consult the original Mesa Visualisation documentation here.

Some of the input parameters for the simulation, such as memory size for learning agents and whether this memory has paired 'chunks' in it or single values, are not available to be altered in the visualiser inputs; this is because they are linked, and changing one without changing another suitably will cause the simulation to crash or simply not run altogether.

To run the Batchrunner variant:

The Batchrunner variant relies on hard-coded variables that can be added in as lists, which it iterates through in all possible permutations to produce larger datasets with different combinations of input parameters. For example, with the payoff for *R* being delineated as "CC": [3, 3.5, 4], the Batchrunner will run *x* iterations of the simulation, outputting data to .csv files, per each of those *R* payoffs. If no particular parameter/s are specified in the batchrun section, the simulation will use whatever is hard-coded as the default for that parameter. Within these lists, commas separate each parameter value.

These parameters can be found at the very bottom of the file 'fixed_random_batchrun.py' within the dictionary named 'br_params'. You can see that I have commented out many past parameter variations; any input parameter that is specified as an input variable to *'def __init__('* at the very top of the code in this file can be altered in the batch runner (as well as any new ones you might add).

Descriptions of each parameter alterable here can be found in the **Appendix**.

The other variables that must be altered for the **Batchrunner** are the number of times you want each parameter combination to be simulated (*x*), and the time step length of each simulation! These can be set under the 'br = BatchRunner(PDModel,…' section of code, as seen in the screenshot below. The value *x* is set in 'iterations', and the value for the timesteps is set in 'max_steps'. Currently these are set to 1 and 25,000 respectively.

[2] This number increases by 1 each time the simulation is run to prevent the outputted files from overwriting each other. It can be changed/reset to zero within the accompanying .csv file, 'filename_number.csv', within the code folder.

```


br = BatchRunner(PDModel,
                 br_params,
                 iterations=1,
                 max_steps=25000,
                 model_reporters={"Data Collector": lambda m: m.datacollector})

```

Once you have set these parameters, run the ‘fixed_random_batchrun.py’ file. Data will output in the form of a) .CSV files for each agent in each simulation, and b) .PNG and .HTML files for the graph visualisation records. Both of these will automatically save to the current folder as they are generated. ***WARNING: Do not open any of the .CSV files whilst the simulation is still running, as it will cause the sim to crash – the sim requires to read/write to each file as each simulation step occurs, and this is not possible if the file is open.***

Appendix

Parameter input descriptions, possible values, and warnings:

Name	Description	Possible Values	Warnings/Notes
Number_of_agents	Number of agents in the simulation	2D Grid: Square values only Random Graph: any whole number	For the 2D grid lattice, this will not alter the number of agents spawned in of itself; it will likely cause a crash if changed in isolation. **Please only enter a square number and alter the 'height' and 'width' variables at the VERY top of the code to fit accordingly.**
DC	The T payoff	Any integer or float	Negative values may cause a crash
CC	The R payoff	Any integer or float	Negative values may cause a crash
DD	The P payoff	Any integer or float	Negative values may cause a crash
CD	The S payoff	Any integer or float	Negative values may cause a crash
Moody_alpha	The moody algorithm learning rate	Floating point value between 0 and 1	
Moody_gamma	The moody algorithm discount factor	Floating point value between 0 and 1	
Moody_epsilon	The value used in epsilon-greedy action selection	Floating point value between 0 and 1	
Moody_sarsa_oppo	The opponent type faced by mSARSA	"TFT", "LEARN", "MOODYLEARN", "ANGEL", "DEVIL", "VPP", "RANDOM", "WSLS", "iWSLS", "MIXED",	Mixed includes an equal likelihood of any of the strategies listed of being selected each time an 'opponent' agent picks its game-playing strategy. **If using fixed_random_batchrun.py, please ensure the list of opponents is provided as a tuple within the overall variable list, otherwise the sim will break.**
Moody_statemode	The mode for the amount of state information mSARSA utilises	'stateless', 'agentstate', 'moodstate'	
Moody_startmood	The starting mood	Value between 0 and 100	
Moody_MA	The value of mA for mSARSA agents	Values between 0 and <1	

Moody_opponents	Whether non-mSARSA agents also possess a mood value, even if they do not utilise it	True, False	
startingBehav	The behaviour that all agents should perform on the first round of the simulation	'C', 'D'	
Only available in the random graph version:			
changeFrequency	Every *x* rounds, agents in the network will change their partners	Any whole positive integer above 0	Set to be larger than the experiment timestep if no restructuring is to occur.
selectionStrategy	The strategy by which agents should select their new partners, as determined in the 'partnerDecision' function within *random_network_functions.py*	"DEFAULT", "SCORE", "REP"	The DEFAULT option makes partner decisions at random.
rewirePercentage	The proportion of the total possible agent pairings that are selected at random for consultation when the network restructures	Floating point value between (and including) 0 and 1, to two decimal places	
forgivenessPeriod	Every *x* instances of restructuring, agents in the network will reset several values related to who has behaved poorly against them in the past	Any whole positive integer above 0.	Set to be larger than the experiment timestep if no forgiveness is to occur.