The Journal of Open Source Software

AgentPy: A package for agent-based modeling in Python

Joël Foramitti[1, 2]

1 Institute of Environmental Science and Technology, Universitat Autònoma de Barcelona, Spain **2** Institute for Environmental Studies, Vrije Universiteit Amsterdam, The Netherlands

DOI: 10.21105/joss.03065

Software

- Review
- Repository
- Archive

Editor: Sebastian Benthall

Reviewers:

- @jamesdamillington
- @martibosch

Submitted: 16 January 2021
Published: 22 June 2021

Introduction

Agent-based models allow for computer simulations based on the autonomous behavior of heterogeneous agents. They are used to generate and understand the emergent dynamics of complex systems, with applications in fields like ecology (DeAngelis & Diaz, 2019), cognitive sciences (Madsen et al., 2019), management (North & Macal, 2007), policy analysis (Castro et al., 2020), economics (Arthur, 2021; Farmer & Foley, 2009), and sociology (Bianchi & Squazzoni, 2015).

AgentPy is an open-source library for the development and analysis of agent-based models. It aims to provide an intuitive syntax for the creation of models together with advanced tools for scientific applications. The framework is written in Python 3, and optimized for interactive computing with IPython and Jupyter. A reference of all features as well as a model library with tutorials and examples can be found in the documentation.[1]

Statement of Need

There are numerous modeling and simulation tools for agent-based models, each with their own particular focus and style (Abar et al., 2017). Notable examples are NetLogo (Wilensky, 1999), which is written in Scala/Java and has become the most established tool in the field; and Mesa (Masad & Kazil, 2015), a more recent framework that has popularized the development of agent-based models in Python.

AgentPy's main distinguishing feature is that it integrates the many different tasks of agent-based modeling within a single environment for interactive computing. This includes the creation of custom agent and model types, interactive simulations (Figure 1) similar to the traditional NetLogo interface, numeric experiments over multiple runs, and the subsequent data analysis of the output. All of these can be performed within a Jupyter Notebook.

The software is further designed for scientific applications, and includes tools for parameter sampling (similar to NetLogo's BehaviorSpace), Monte Carlo experiments, random number generation, parallel computing, and sensitivity analysis. Beyond these built-in features, AgentPy is also designed for compatibility with established Python libraries like EMA Workbench, NetworkX, NumPy, pandas, SALib, SciPy, and seaborn.

[1] Link to the AgentPy documentation: https://agentpy.readthedocs.io

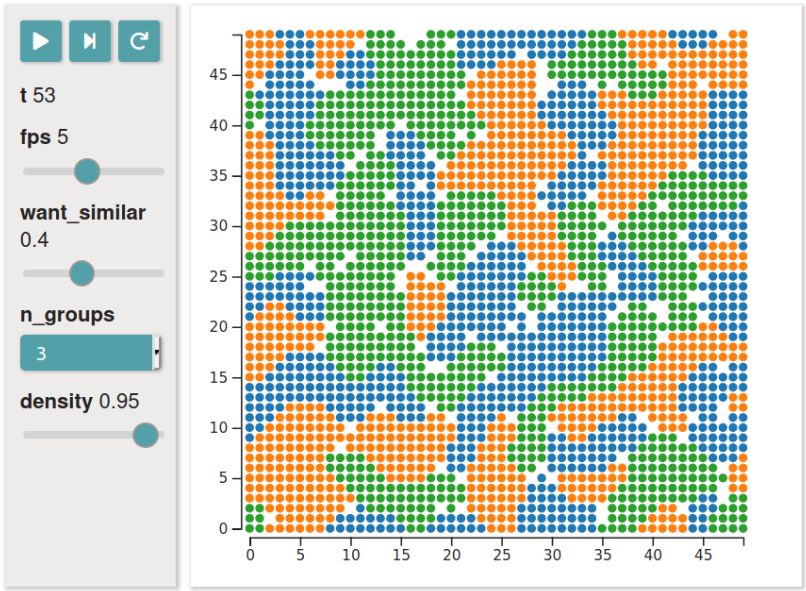

Figure 1: An interactive simulation of Schelling's segregation model in a Jupyter Notebook.

Basic structure

The AgentPy framework follows a nested structure that is illustrated in Figure 2. The basic building blocks are the agents, which can be placed within (multiple) environments with different topologies such as a network, a spatial grid, or a continuous space. Models are used to initiate these objects, perform a simulation, and record data. Experiments can run a model over multiple iterations and parameter combinations. The resulting output data can then be saved and re-arranged for analysis and visualization.

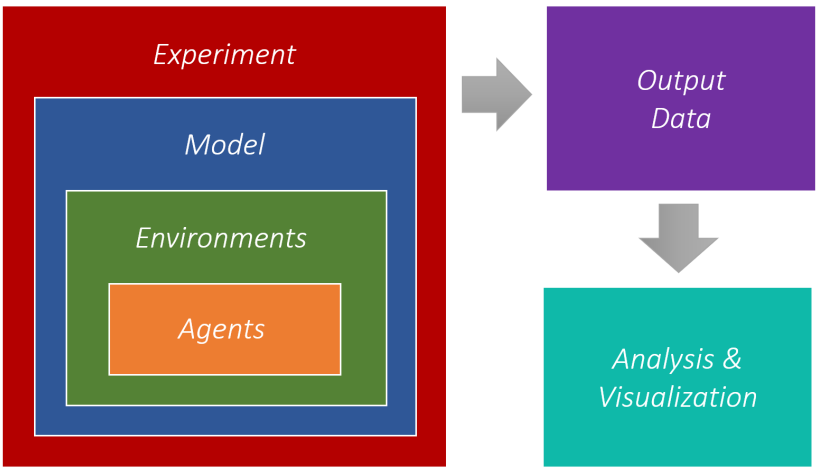

Figure 2: Nested structure of the AgentPy framework.

Model example

The following code shows an example of a simple model that explores the distribution of wealth under a randomly trading population of agents. The original version of this model was written in Mesa, allowing for a comparison of the syntax between the two frameworks.[2] To start, we import the AgentPy library as follows:

```
import agentpy as ap
```

We then define a new type of `Agent`. The method `setup` will be called automatically at the agent's creation. Each agent starts with one unit of wealth. `wealth_transfer` will be called by the model during each time-step. When called, the agent randomly selects a trading partner and hands them one unit of their wealth, given that they have one to spare.

```
class WealthAgent(ap.Agent):

    def setup(self):
        self.wealth = 1

    def wealth_transfer(self):
        if self.wealth > 0:
            partner = self.model.agents.random()
            partner.wealth += 1
            self.wealth -= 1
```

Next, we define a `Model`. The method `setup` is called at the beginning the simulation, `step` is called during each time-step, and `end` is called after the simulation has finished. An `AgentList` is used to create a set of agents that can then be accessed as a group. The attribute `p` is used to access the model's parameters. And the method `record` is used to store data for later analysis.

```
class WealthModel(ap.Model):

    def setup(self):
        self.agents = ap.AgentList(self, self.p.n, WealthAgent)

    def step(self):
        self.agents.wealth_transfer()

    def end(self):
        self.agents.record('wealth')
```

To run a simulation, a new instance of the model is created with a dictionary of parameters. While the parameter `n` is used in the model's setup, the parameter `steps` automatically defines the maximum number of time-steps. Alternatively, the simulation could also be stopped with `Model.stop`. To perform the actual simulation, one can use `Model.run`.

```
parameters = {'n': 100, 'steps': 100}
model = MoneyModel(parameters)
results = model.run()
```

[2] For a direct comparison, see: https://agentpy.readthedocs.io/en/stable/comparison.html

Parameters can also be defined as ranges and used to generate a `Sample`. This sample can then be used to initiate an `Experiment` that can repeatedly run the model over multiple parameter combinations and iterations. In the following example, the parameter `n` is varied from 1 to 100 and the simulation is repeated 10 times for each value of `n`.

```
parameters = {'n': ap.IntRange(1, 100), 'steps': 100}
sample = ap.Sample(parameters, n=10)
exp = ap.Experiment(MoneyModel, sample, iterations=10, record=True)
results = exp.run()
```

The output of both models and experiments is given as a `DataDict` with tools to save, arrange, and analyse data. Here, we use the seaborn library to display a histogram of the experiment's output. The plot is presented in Figure 3. It shows that the random interaction of the agents creates an inequality of wealth that resembles a Boltzmann-Gibbs distribution.

```
import seaborn as sns
sns.histplot(data=results.variables.MoneyAgent, binwidth=1)
```

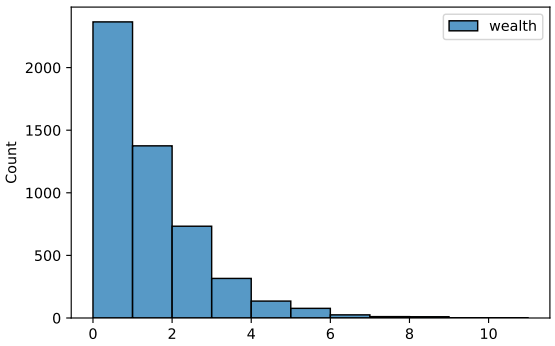

Figure 3: Histogram of the agents' wealth in the model example.

More examples - including spatial environments, networks, stochastic processes, interactive simulations (see Figure 1), animations, and sensitivity analysis - can be found in the model library and user guides of the documentation. For questions and ideas, please visit the discussion forum.[3]

Acknowledgements

This study has received funding through an ERC Advanced Grant from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement n° 741087). I thank Jeroen C.J.M van den Berg, Ivan Savin, Martí Bosch, and James Millington for their helpful comments.

[3]Link to the AgentPy dicussion forum: https://github.com/JoelForamitti/agentpy/discussions

References

Abar, S., Theodoropoulos, G. K., Lemarinier, P., & O'Hare, G. M. P. (2017). Agent based modelling and simulation tools: A review of the state-of-art software. *Computer Science Review*, *24*, 13–33. https://doi.org/10.1016/j.cosrev.2017.03.001

Arthur, W. B. (2021). Foundations of complexity economics. *Nature Reviews Physics*, *3*, 136–145. https://doi.org/10.1038/s42254-020-00273-3

Bianchi, F., & Squazzoni, F. (2015). Agent-based models in sociology. *WIREs Computational Statistics*, *7*(4), 284–306. https://doi.org/10.1002/wics.1356

Castro, J., Drews, S., Exadaktylos, F., Foramitti, J., Klein, F., Konc, T., Savin, I., & Bergh, J. van den. (2020). A review of agent-based modeling of climate-energy policy. *WIREs Climate Change*, *11*(4), e647. https://doi.org/10.1002/wcc.647

DeAngelis, D. L., & Diaz, S. G. (2019). Decision-making in agent-based modeling: A current review and future prospectus. *Frontiers in Ecology and Evolution*, *6*, 237. https://doi.org/10.3389/fevo.2018.00237

Farmer, J. D., & Foley, D. (2009). The economy needs agent-based modelling. *Nature*, *460*(7256), 685–686. https://doi.org/10.1038/460685a

Madsen, J. K., Bailey, R., Carrella, E., & Koralus, P. (2019). Analytic versus computational cognitive models: Agent-based modeling as a tool in cognitive sciences. *Current Directions in Psychological Science*, *28*(3), 299–305. https://doi.org/10.1177/0963721419834547

Masad, David, & Kazil, Jacqueline. (2015). Mesa: An Agent-Based Modeling Framework. In Kathryn Huff & James Bergstra (Eds.), *Proceedings of the 14th Python in Science Conference* (pp. 51–58). https://doi.org/10.25080/Majora-7b98e3ed-009

North, M. J., & Macal, C. M. (2007). *Managing business complexity: Discovering strategic solutions with agent-based modeling and simulation*. Oxford University Press. https://doi.org/10.1093/acprof:oso/9780195172119.001.0001

Wilensky, U. (1999). *NetLogo*. Center for Connected Learning; Computer-Based Modeling, Northwestern University. Evanston, IL. http://ccl.northwestern.edu/netlogo/