Open ABM

Release 0.0.1

Conor M. Artman

Dec 09, 2019

CONTENTS

CHAPTER

ONE

OPEN ABM TOOLS AND UTILITIES

Open ABM runs on MESA and wraps basic MESA implementations with easy-to-use helper functions. All current models were built by the MESA team. Users are encouraged to explroe MESA and its tools after getting started with Open ABM here: https://mesa.readthedocs.io/en/master/tutorials/intro_tutorial.html

For an interactive introduction, please see our Jupyter notebook walkthrough.

oabm_tools.**check_models**()
Returns the current list of available models one can plug-and-play with Open ABM

Returns Returns a list of strings of valid names one can use with Open ABM's make() function for building models.

Return type String

oabm_tools.**export_agent_data**(*model*)
Helper function for export_data, but accessible to user if preferred.

Parameters **model** – An Open ABM supported model / environment instance

Returns Agent-level data

Return type Pandas Dataframe

oabm_tools.**export_data**(*model*, *agent_level*, *model_level*)
Exports data associated with the datacollector attribute of a model instance.

Parameters

- **model** – An instance of an Open ABM-supported model / environment
- **agent_level** (*Boolean*) – Tells export_data whether to return agent-level data
- **model_level** (*Boolean*) – Tells export_data whether to return model-level data

Raises **AssertionError** – Model and agent must be Booleans, and must both be specified.

Returns Dataset of model- and/or agent-level data depending on user input

Return type Pandas Dataframe

oabm_tools.**export_model_data**(*model*)
Helper function for export_data, but accessible to user if preferred.

Parameters **model** – An Open ABM supported model / environment instance

Returns Model-level data

Return type Pandas Dataframe

oabm_tools.**get_agent_parameters**(*model*)
Returns Python dictionary of all agents' model parameters.

Parameters **model** – An Open ABM supported model / environment instance.

Returns Nested dictionary of each agent's dictionary of parameters/attributes.

oabm_tools.**get_agent_step**(*agent_list*, *verbose=True*, *distinct=True*)
Returns step functions of agents in a list.

Parameters

- **agent_list** – A list of agents whose step functions we want to see
- **verbose** – Boolean specifying if the function should print each agent's step function
- **distinct** – Boolean. If true, then when verbose is True will print each agent's step function. If false with verbose set to true, will only print distinct step function per type of agent.

Returns List of agent step functions in the form of individual Python dictionaries.

oabm_tools.**get_model_parameters**(*model*)
Returns the full dictionary of all model-level parameters / attributes.

Parameters **model** – An Open ABM supported model / environment instance

Returns Python dictionary of model-level parameters

oabm_tools.**get_model_step**(*model*, *verbose=True*)
Returns step functions of model instance.

Parameters

- **model** – Open ABM supported model instance.
- **verbose** – Boolean specifying if the function should print step function

Returns String of source code for the model's step function.

oabm_tools.**make**(*model_name*, *server=True*)
Makes a model instantiation of the requestion ABM, where the ABM name is a string taken from the list generated by check_models()

Parameters

- **model_name** (*String*) – String name of the model, as taken from check_models() output
- **server** (*Boolean*) – Boolean-valued parameter. If true, a pop-out interactive window should appear in your browser for the default visualizations of the requested model.

Raises **AssertionError** – model_name only accepts strings.

Returns Returns an instance of the requested ABM

oabm_tools.**nested_dict**()
Helper function for creating nested dictionaries.

oabm_tools.**set_agent_parameters**(*model*, *new_params*)
Takes a model object and uses the dictionary of new parameters to update agents' parameters.

Parameters **model** – An Open ABM supported model /environment instance.

New_params A python dictionary with unique IDs of agents as keys and dictionaries of new parameter values as values.

Returns Returns an updated model object containing updated agent parameters.

oabm_tools.**set_agent_step**(*agent_dict*, *model*)
Set step functions based on a dict of unique IDs with associated new step function.

Parameters

- **agent_dict** – Python dictionary of unique ID / step function keys/values
- **model** – Open ABM supported model instance

Returns Updated model instance

oabm_tools.**set_data_collection**(*model*, *agent_level*, *model_level*, *agent_data_to_collect=None*, *model_data_to_collect=None*)

Sets the data to be collected by the model for agent and model leveld data.

data_to_collect can be a dict of string_names the user chooses and then attributes the user wants, or it can be string_names the user chooses and functions of the model that the user wants

Note that if only specifying data collection for agent or model level, then data_to_collect automatically fills that slot; otherwise, user must specify separate dictionary

Also note that this assumes that the given model has a datacollector attribute already!

Parameters

- **model** – Open ABM supported model instance
- **agent_level** – Boolean specifying whether to collect agent level data
- **model_level** – Boolean specifying whether to collect model level data

Returns Updated instance of the model set to collect the requested data.

oabm_tools.**set_model_parameters**(*model*, *new_params*)

Adjusts the dictionary of model-level parameters / attributes.

Parameters

- **model** – An Open ABM supported model / environment instance
- **new_params** – A python dictionary of parameters / attributes replacing old parameter values.

Returns A model instance

oabm_tools.**set_model_step**(*model*, *new_step_func*)

Sets step function for the model, and returns the updated model instance.

Parameters

- **model** – An Open ABM supported model instance.
- **new_step_func** – A method acting as the new step function for the model.

Returns An updated model instance.

CHAPTER

TWO

BANK RESERVES MODEL

The following code was adapted from the Bank Reserves model included in Netlogo Model information can be found at: http://ccl.northwestern.edu/netlogo/models/BankReserves Accessed on: November 2, 2017 Author of NetLogo code: Wilensky, U. (1998). NetLogo Bank Reserves model. http://ccl.northwestern.edu/netlogo/models/BankReserves. Center for Connected Learning and Computer-Based Modeling, Northwestern University, Evanston, IL.

class examples.bank_reserves.bank_reserves.model.**BankReserves**(*height=20*, *width=20*, *init_people=2*, *rich_threshold=10*, *reserve_percent=50*, *server=True*, *num_steps=100*)

This model is a Mesa implementation of the Bank Reserves model from NetLogo. It is a highly abstracted, simplified model of an economy, with only one type of agent and a single bank representing all banks in an economy. People (represented by circles) move randomly within the grid. If two or more people are on the same grid location, there is a 50% chance that they will trade with each other. If they trade, there is an equal chance of giving the other agent $5 or $2. A positive trade balance will be deposited in the bank as savings. If trading results in a negative balance, the agent will try to withdraw from its savings to cover the balance. If it does not have enough savings to cover the negative balance, it will take out a loan from the bank to cover the difference. The bank is required to keep a certain percentage of deposits as reserves and the bank's ability to loan at any given time is a function of the amount of deposits, its reserves, and its current total outstanding loan amount.

grid_w = 20

init parameters "init_people", "rich_threshold", and "reserve_percent" are all UserSettableParameters

run_model(*n=None*)

Run the model until the end condition is reached. Overload as needed.

step()

A single step. Fill in here.

examples.bank_reserves.bank_reserves.model.**get_num_mid_agents**(*model*)

return number of middle class agents

examples.bank_reserves.bank_reserves.model.**get_num_poor_agents**(*model*)

return number of poor agents

examples.bank_reserves.bank_reserves.model.**get_num_rich_agents**(*model*)

return number of rich agents

examples.bank_reserves.bank_reserves.model.**get_total_savings**(*model*)

sum of all agents' savings

examples.bank_reserves.bank_reserves.model.**get_total_wallets**(*model*)
sum of amounts of all agents' wallets

The following code was adapted from the Bank Reserves model included in Netlogo Model information can be found at: http://ccl.northwestern.edu/netlogo/models/BankReserves Accessed on: November 2, 2017 Author of NetLogo code: Wilensky, U. (1998). NetLogo Bank Reserves model. http://ccl.northwestern.edu/netlogo/models/BankReserves. Center for Connected Learning and Computer-Based Modeling, Northwestern University, Evanston, IL.

class examples.bank_reserves.bank_reserves.agents.**Bank**(*unique_id*, *model*, *reserve_percent=50*)

bank_loans = None
percent of deposits the bank must keep in reserves - this is a UserSettableParameter in server.py

class examples.bank_reserves.bank_reserves.agents.**Person**(*unique_id*, *pos*, *model*, *moore*, *bank*, *rich_threshold*)

do_business()
check if person has any savings, any money in wallet, or if the bank can loan them any money

loans = None
start everyone off with a random amount in their wallet from 1 to a user settable rich threshold amount

step()
A single step of the agent.

take_out_loan(*amount*)
borrow from the bank to put money in my wallet, and increase my outstanding loans

Citation: The following code is a copy from random_walk.py at https://github.com/projectmesa/mesa/blob/master/examples/wolf_sheep/wolf_sheep/random_walk.py Accessed on: November 2, 2017 Original Author: Jackie Kazil

Generalized behavior for random walking, one grid cell at a time.

class examples.bank_reserves.bank_reserves.random_walk.**RandomWalker**(*unique_id*, *pos*, *model*, *moore=True*)
Class implementing random walker methods in a generalized manner. Not indended to be used on its own, but to inherit its methods to multiple other agents.

random_move()
Step one cell in any allowable direction.

class examples.boid_flockers.boid_flockers.boid.**Boid**(*unique_id*, *model*, *pos*, *speed*, *velocity*, *vision*, *separation*, *cohere=0.025*, *separate=0.25*, *match=0.04*)
A Boid-style flocker agent.

The agent follows three behaviors to flock:

- Cohesion: steering towards neighboring agents.
- Separation: avoiding getting too close to any other agent.
- Alignment: try to fly in the same direction as the neighbors.

Boids have a vision that defines the radius in which they look for their neighbors to flock with. Their speed (a scalar) and velocity (a vector) define their movement. Separation is their desired minimum distance from any other Boid.

cohere(*neighbors*)
Return the vector toward the center of mass of the local neighbors.

match_heading(*neighbors*)
Return a vector of the neighbors' average heading.

separate(*neighbors*)
Return a vector away from any neighbors closer than separation dist.

step()
Get the Boid's neighbors, compute the new vector, and move accordingly.

CHAPTER

THREE

FLOCKERS

A Mesa implementation of Craig Reynolds's Boids flocker model. Uses numpy arrays to represent vectors.

class examples.boid_flockers.boid_flockers.model.**BoidFlockers**(*population=100*, *width=100*, *height=100*, *speed=1*, *vision=10*, *separation=2*, *cohere=0.025*, *separate=0.25*, *match=0.04*, *server=True*, *num_steps=1000*)

Flocker model class. Handles agent creation, placement and scheduling.

make_agents()
Create self.population agents, with random positions and starting headings.

run_model(*n=None*)
Run the model until the end condition is reached. Overload as needed.

step()
A single step. Fill in here.

class examples.boid_flockers.boid_flockers.SimpleContinuousModule.**SimpleCanvas**(*portrayal_metho*
can-
vas_height=500,
can-
vas_width=500)

render(*model*)
Build visualization data from a model object.

Parameters **model** – A model object

Returns A JSON-ready object.

CHAPTER

FOUR

BOLTZMANN WEALTH MODEL

This example created by MESA's team

class examples.boltzmann_wealth_model.boltzmann_wealth_model.model.**BoltzmannWealthModel**(*N=*
wid
hei
num
ser

A simple model of an economy where agents exchange currency at random.

All the agents begin with one unit of currency, and each time step can give a unit of currency to another agent. Note how, over time, this produces a highly skewed distribution of wealth.

run_model(*n=None*)
Run the model until the end condition is reached. Overload as needed.

step()
A single step. Fill in here.

class examples.boltzmann_wealth_model.boltzmann_wealth_model.model.**MoneyAgent**(*unique_id*, *model*)

An agent with fixed initial wealth.

step()
A single step of the agent.

CHAPTER

FIVE

BOLTZMANN WEALTH NETWORK MODEL

This example created by MESA's team

class examples.boltzmann_wealth_model_network.boltzmann_wealth_model_network.model.**Boltzma**

A model with some number of agents.

run_model(*n=None*)
Run the model until the end condition is reached. Overload as needed.

step()
A single step. Fill in here.

class examples.boltzmann_wealth_model_network.boltzmann_wealth_model_network.model.**MoneyAge**

An agent with fixed initial wealth.

step()
A single step of the agent.

class examples.conways_game_of_life.conways_game_of_life.cell.**Cell**(*pos*, *model*, *init_state=0*)

Represents a single ALIVE or DEAD cell in the simulation.

advance()
Set the state to the new computed state – computed in step().

step()
Compute if the cell will be dead or alive at the next tick. This is based on the number of alive or dead neighbors. The state is not changed here, but is just computed and stored in self._nextState, because our current state may still be necessary for our neighbors to calculate their next state.

CHAPTER

SIX

CONWAYS GAME OF LIFE MODEL

This example created by MESA's team.

class examples.conways_game_of_life.conways_game_of_life.model.**ConwaysGameOfLife**(*height=50*, *width=50*, *server=True*)

Represents the 2-dimensional array of cells in Conway's Game of Life.

run_model(*n=None*)
Run the model until the end condition is reached. Overload as needed.

step()
Have the scheduler advance each cell by one step

examples.conways_game_of_life.conways_game_of_life.portrayal.**portrayCell**(*cell*)
This function is registered with the visualization server to be called each tick to indicate how to draw the cell in its current state. :param cell: the cell in the simulation :return: the portrayal dictionary.

class examples.epstein_civil_violence.epstein_civil_violence.agent.**Citizen**(*unique_id*, *model*, *pos*, *hardship*, *regime_legitimacy*, *risk_aversion*, *threshold*, *vision*)

A member of the general population, may or may not be in active rebellion. Summary of rule: If grievance - risk > threshold, rebel.

unique_id
unique int

x, y
Grid coordinates

hardship
Agent's 'perceived hardship (i.e., physical or economic privation).' Exogenous, drawn from U(0,1).

regime_legitimacy
Agent's perception of regime legitimacy, equal across agents. Exogenous.

risk_aversion
Exogenous, drawn from U(0,1).

threshold
if (grievance - (risk_aversion * arrest_probability)) > threshold, go/remain Active

vision
number of cells in each direction (N, S, E and W) that agent can inspect

condition
Can be "Quiescent" or "Active;" deterministic function of greivance, perceived risk, and

grievance
deterministic function of hardship and regime_legitimacy; how aggrieved is agent at the regime?

arrest_probability
agent's assessment of arrest probability, given rebellion

step()
Decide whether to activate, then move if applicable.

update_estimated_arrest_probability()
Based on the ratio of cops to actives in my neighborhood, estimate the p(Arrest | I go active).

update_neighbors()
Look around and see who my neighbors are

class examples.epstein_civil_violence.epstein_civil_violence.agent.**Cop**(*unique_id*, *model*, *pos*, *vision*)
A cop for life. No defection. Summary of rule: Inspect local vision and arrest a random active agent.

unique_id
unique int

x, y
Grid coordinates

vision
number of cells in each direction (N, S, E and W) that cop is able to inspect

step()
Inspect local vision and arrest a random active agent. Move if applicable.

update_neighbors()
Look around and see who my neighbors are.

CHAPTER

SEVEN

EPSTEIN CIVIL VIOLENCE MODEL

This example created by MESA's team.

class examples.epstein_civil_violence.epstein_civil_violence.model.**EpsteinCivilViolence**(*height*, *width*, *citizen_density*, *cop_density*, *citizen_vision*, *cop_vision*, *legitimacy*, *max_jail_term*, *active_threshold*, *arrest_probability*, *movement*, *max_iters*, *seed*, *num_agents*)

Model 1 from "Modeling civil violence: An agent-based computational approach," by Joshua Epstein. http://www.pnas.org/content/99/suppl_3/7243.full .. attribute:: height

grid height

width
grid width

citizen_density
approximate % of cells occupied by citizens.

cop_density
approximate % of calles occupied by cops.

citizen_vision
number of cells in each direction (N, S, E and W) that citizen can inspect

cop_vision
number of cells in each direction (N, S, E and W) that cop can inspect

legitimacy
(L) citizens' perception of regime legitimacy, equal across all citizens

max_jail_term
(J_max)

active_threshold
if (grievance - (risk_aversion * arrest_probability)) > threshold, citizen rebels

arrest_prob_constant
set to ensure agents make plausible arrest probability estimates

movement
binary, whether agents try to move at step end

max_iters
model may not have a natural stopping point, so we set a max.

static count_jailed(*model*)
Helper method to count jailed agents.

static count_type_citizens(*model*, *condition*, *exclude_jailed=True*)
Helper method to count agents by Quiescent/Active.

run_model(*n=None*)
Run the model until the end condition is reached. Overload as needed.

step()
Advance the model by one step and collect data.

class examples.forest_fire.forest_fire.agent.**TreeCell**(*pos*, *model*)
A tree cell.

x, y
Grid coordinates

condition
Can be "Fine", "On Fire", or "Burned Out"

unique_id
(x,y) tuple.

unique_id isn't strictly necessary here, but it's good practice to give one to each agent anyway.

step()
If the tree is on fire, spread it to fine trees nearby.

CHAPTER

EIGHT

FOREST FIRE MODEL

This example created by MESA's team.

class examples.forest_fire.forest_fire.model.**ForestFire**(*height=100*, *width=100*, *density=0.65*, *server=True*, *num_steps=1000*)

Simple Forest Fire model.

static count_type(*model*, *tree_condition*)

Helper method to count trees in a given condition in a given model.

run_model(*n=None*, *export_agent_data=False*, *export_model_data=False*)

Run the model until the end condition is reached. Overload as needed.

step()

Advance the model by one step.

class examples.hex_snowflake.hex_snowflake.cell.**Cell**(*pos*, *model*, *init_state=0*)

Represents a single ALIVE or DEAD cell in the simulation.

advance()

Set the state to the new computed state – computed in step().

step()

Compute if the cell will be dead or alive at the next tick. A dead cell will become alive if it has only one neighbor. The state is not changed here, but is just computed and stored in self._nextState, because our current state may still be necessary for our neighbors to calculate their next state. When a cell is made alive, its neighbors are able to be considered in the next step. Only cells that are considered check their neighbors for performance reasons.

CHAPTER

NINE

HEX SNOWFLAKE MODEL

This example created by MESA's team

class examples.hex_snowflake.hex_snowflake.model.**HexSnowflake**(*height=50*, *width=50*, *server=True*, *num_steps=1000*)

Represents the hex grid of cells. The grid is represented by a 2-dimensional array of cells with adjacency rules specific to hexagons.

run_model(*n=None*)

Run the model until the end condition is reached. Overload as needed.

step()

Have the scheduler advance each cell by one step

examples.hex_snowflake.hex_snowflake.portrayal.**portrayCell**(*cell*)

This function is registered with the visualization server to be called each tick to indicate how to draw the cell in its current state. :param cell: the cell in the simulation :return: the portrayal dictionary.

class examples.pd_grid.pd_grid.agent.**PDAgent**(*pos*, *model*, *starting_move=None*)

Agent member of the iterated, spatial prisoner's dilemma model.

step()

Get the neighbors' moves, and change own move accordingly.

CHAPTER

TEN

PRISONER'S DILEMMA GRID MODEL

This example created by MESA's team

class examples.pd_grid.pd_grid.model.**PdGrid**(*height=50*, *width=50*, *schedule_type='Random'*, *payoffs=None*, *seed=None*, *server=True*, *num_steps=1000*)

Model class for iterated, spatial prisoner's dilemma model.

run_model(*n=None*)

Run the model until the end condition is reached. Overload as needed.

step()

A single step. Fill in here.

examples.pd_grid.pd_grid.portrayal.**portrayPDAgent**(*agent*)

This function is registered with the visualization server to be called each tick to indicate how to draw the agent in its current state. :param agent: the agent in the simulation :return: the portrayal dictionary

CHAPTER

ELEVEN

SCHELLING MODEL

This example created by MESA's team.

class examples.schelling.model.**Schelling**(*height=20*, *width=20*, *density=0.8*, *minority_pc=0.2*, *homophily=3*, *num_steps=1000*, *server=True*)

Model class for the Schelling segregation model.

run_model(*n=None*)

Run the model until the end condition is reached. Overload as needed.

step()

Run one step of the model. If All agents are happy, halt the model.

class examples.schelling.model.**SchellingAgent**(*pos*, *model*, *agent_type*)

Schelling segregation agent

step()

A single step of the agent.

class examples.schelling.server.**HappyElement**

Display a text count of how many happy agents there are.

render(*model*)

Build visualization data from a model object.

Parameters **model** – A model object

Returns A JSON-ready object.

examples.schelling.server.**schelling_draw**(*agent*)

Portrayal Method for canvas

CHAPTER

TWELVE

SUGARSCAPE CONSTANT GROWBACK MODEL

Replication of the model found in Netlogo: Li, J. and Wilensky, U. (2009). NetLogo Sugarscape 2 Constant Growback model. http://ccl.northwestern.edu/netlogo/models/Sugarscape2ConstantGrowback. Center for Connected Learning and Computer-Based Modeling, Northwestern University, Evanston, IL.

class examples.sugarscape_cg.sugarscape_cg.model.**SugarscapeCg**(*height=50*, *width=50*, *initial_population=100*, *server=True*)

Sugarscape 2 Constant Growback

run_model(*step_count=200*)

Run the model until the end condition is reached. Overload as needed.

step()

A single step. Fill in here.

class examples.sugarscape_cg.sugarscape_cg.agents.**SsAgent**(*pos*, *model*, *moore=False*, *sugar=0*, *metabolism=0*, *vision=0*)

step()

A single step of the agent.

class examples.sugarscape_cg.sugarscape_cg.agents.**Sugar**(*pos*, *model*, *max_sugar*)

step()

A single step of the agent.

examples.sugarscape_cg.sugarscape_cg.agents.**get_distance**(*pos_1*, *pos_2*)

Get the distance between two point

Parameters **pos_2** (*pos_1,*) – Coordinate tuples for both points.

class examples.sugarscape_cg.sugarscape_cg.schedule.**RandomActivationByBreed**(*model*)

A scheduler which activates each type of agent once per step, in random order, with the order reshuffled every step.

This is equivalent to the NetLogo 'ask breed...' and is generally the default behavior for an ABM.

Assumes that all agents have a step() method.

add(*agent*)

Add an Agent object to the schedule

Parameters **agent** – An Agent to be added to the schedule.

get_breed_count(*breed_class*)
Returns the current number of agents of certain breed in the queue.

remove(*agent*)
Remove all instances of a given agent from the schedule.

step(*by_breed=True*)
Executes the step of each agent breed, one at a time, in random order.

Parameters **by_breed** – If True, run all agents of a single breed before running the next one.

step_breed(*breed*)
Shuffle order and run all agents of a given breed.

Parameters **breed** – Class object of the breed to run.

CHAPTER

THIRTEEN

VIRUS-ON-NETWORK MODEL

This example created by MESA's team.

class examples.virus_on_network.virus_on_network.model.**State**
An enumeration.

class examples.virus_on_network.virus_on_network.model.**VirusAgent**(*unique_id*, *model*, *initial_state*, *virus_spread_chance*, *virus_check_frequency*, *recovery_chance*, *gain_resistance_chance*)

step()
A single step of the agent.

class examples.virus_on_network.virus_on_network.model.**VirusOnNetwork**(*num_nodes=10*, *avg_node_degree=3*, *initial_outbreak_size=1*, *virus_spread_chance=0.4*, *virus_check_frequency=0.4*, *recovery_chance=0.3*, *gain_resistance_chance=0.5*, *num_steps=1000*, *server=True*)

A virus model with some number of agents

run_model(*n=None*)
Run the model until the end condition is reached. Overload as needed.

step()
A single step. Fill in here.

class examples.virus_on_network.virus_on_network.server.**MyTextElement**

render(*model*)
Build visualization data from a model object.

Parameters **model** – A model object

Returns A JSON-ready object.

CHAPTER

FOURTEEN

WOLF-SHEEP PREDATION MODEL

Replication of the model found in NetLogo: Wilensky, U. (1997). NetLogo Wolf Sheep Predation model. http://ccl.northwestern.edu/netlogo/models/WolfSheepPredation. Center for Connected Learning and Computer-Based Modeling, Northwestern University, Evanston, IL.

class examples.wolf_sheep.wolf_sheep.model.**WolfSheep**(*height=20*, *width=20*, *initial_sheep=100*, *initial_wolves=50*, *sheep_reproduce=0.04*, *wolf_reproduce=0.05*, *wolf_gain_from_food=20*, *grass=False*, *grass_regrowth_time=30*, *sheep_gain_from_food=4*, *num_steps=1000*, *server=True*)

Wolf-Sheep Predation Model

run_model(*n=None*)

Run the model until the end condition is reached. Overload as needed.

step()

A single step. Fill in here.

class examples.wolf_sheep.wolf_sheep.agents.**GrassPatch**(*unique_id*, *pos*, *model*, *fully_grown*, *countdown*)

A patch of grass that grows at a fixed rate and it is eaten by sheep

step()

A single step of the agent.

class examples.wolf_sheep.wolf_sheep.agents.**Sheep**(*unique_id*, *pos*, *model*, *moore*, *energy=None*)

A sheep that walks around, reproduces (asexually) and gets eaten.

The init is the same as the RandomWalker.

step()

A model step. Move, then eat grass and reproduce.

class examples.wolf_sheep.wolf_sheep.agents.**Wolf**(*unique_id*, *pos*, *model*, *moore*, *energy=None*)

A wolf that walks around, reproduces (asexually) and eats sheep.

step()

A single step of the agent.

Generalized behavior for random walking, one grid cell at a time.

class examples.wolf_sheep.wolf_sheep.random_walk.**RandomWalker**(*unique_id*, *pos*, *model*, *moore=True*)

Class implementing random walker methods in a generalized manner.

Not indended to be used on its own, but to inherit its methods to multiple other agents.

random_move()

Step one cell in any allowable direction.

class examples.wolf_sheep.wolf_sheep.schedule.**RandomActivationByBreed**(*model*)

A scheduler which activates each type of agent once per step, in random order, with the order reshuffled every step.

This is equivalent to the NetLogo 'ask breed...' and is generally the default behavior for an ABM.

Assumes that all agents have a step() method.

add(*agent*)

Add an Agent object to the schedule

Parameters **agent** – An Agent to be added to the schedule.

get_breed_count(*breed_class*)

Returns the current number of agents of certain breed in the queue.

remove(*agent*)

Remove all instances of a given agent from the schedule.

step(*by_breed=True*)

Executes the step of each agent breed, one at a time, in random order.

Parameters **by_breed** – If True, run all agents of a single breed before running the next one.

step_breed(*breed*)

Shuffle order and run all agents of a given breed.

Parameters **breed** – Class object of the breed to run.

CHAPTER

FIFTEEN

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

e

o