panaxea Documentation

Release 1

Dario Panada

Feb 20, 2020

Contents:

- genindex
- modindex
- search

CHAPTER 1

Welcome to panaxea's documentation!

CHAPTER 2

Framework Core

2.1 Environment

class `Environment.`**`Environment`**(*name*, *model*)

Bases: `object`

Initializes a generic environment instance. Assigns the name and binds it to the model instance.

This class would **not** be instantiated itself. It would be extended by a concrete environment extension.

name [string] The name of the environment, this will be used when referring to it throughout the code. (Eg: AgentEnv, OxygenEnv, etc.)

model [model] The instance of the model class to which the environment will be attached.

class `Environment.`**`Grid2D`**(*name*, *xsize*, *ysize*, *model*)

Bases: *`Environment.Environment`*, `object`

Initializes a 2D Grid object. Assigns the name, size and binds it to a model instance.

This class would **not** be instantiated itself as it lacks any concrete extension of the underlying grid. Rather, it is used to check whether position in the grid is valid and provide moore neighbourhoods and additional information based on the geometry of a 2D grid.

name [string] The name of the environment, this will be used when referring to it throughout the code. (Eg: AgentEnv, OxygenEnv, etc.)

xsize [int] The number of positions along the x-axis. In other words, the width of the environment.

ysize [int] The number of positions along the y-axis. In other words, the height of the environment.

model [model] The instance of the model class to which the environment will be attached.

`get_moore_neighbourhood`(*position*, *shuffle_neigh=True*)

Returns a list of moore neighbours for a given position.

A moore neighbourhood is intended as all positions immediately adjacent to a target one.

position [tuple] A tuple consisting of exactly two element, each of them being an integer.

shuffle_neigh [bool] Optional, if set to true the list of neighbours will be shuffled and returned in a random order.

list A list of moore neighbours

valid_position(*position*)
Checks whether a coordinate is valid with regards to the size of the grid instance.

Checks if none of the coordinate values are negative or out of bounds.

position [tuple] A tuple consisting of exactly two element, each of them being an integer.

bool True if the position is a valid one, false otherwise.

class Environment.**Grid3D**(*name*, *xsize*, *ysize*, *zsize*, *model*)
Bases: *Environment.Environment*, object

Initializes a 3D Grid object. Assigns the name, size and binds it to a model instance.

This class would **not** be instantiated itself as it lacks any concrete extension of the underlying grid. Rather, it is used to check whether position in the grid is valid and provide moore neighbourhoods and additional information based on the geometry of a 3D grid.

name [string] The name of the environment, this will be used when referring to it throughout the code. (Eg: AgentEnv, OxygenEnv, etc.)

xsize [int] The number of positions along the x-axis. In other words, the width of the environment.

ysize [int] The number of positions along the y-axis. In other words, the height of the environment.

zsize [int] The number of positions along the z-axis. In other words, the depth of the environment.

model [model] The instance of the model class to which the environment will be attached.

get_moore_neighbourhood(*position*, *shuffle_neigh=True*)
Returns a list of moore neighbours for a given position.

A moore neighbourhood is intended as all positions immediately adjacent to a target one.

position [tuple] A tuple consisting of exactly three element, each of them being an integer.

shuffle_neigh [bool] Optional, if set to true the list of neighbours will be shuffled and returned in a random order.

list A list of moore neighbours

valid_position(*position*)
Checks whether a coordinate is valid with regards to the size of the grid instance.

Checks if none of the coordinate values are negative or out of bounds.

position [tuple] A tuple consisting of exactly three element, each of them being an integer.

bool True if the position is a valid one, false otherwise.

class Environment.**NumericalGrid**
Bases: object

Initializes a NumericalGrid object. A NumericalGrid holds a single number value at each position.

This class exposes methods to explore a position's neighbourhood.

This class would **not** be itself instantiated, but would be extended by another class that would implement it.

get_least_in_neigh(*position*)

Gets the coordinates of the moore neighbour with the smallest value. If multiple neighbours meet the criteria, any may be returned.

A moore neighbourhood is defined as all positions immediately adjacent the one we are searching. It does not include the target position itself.

Positions should be given and are returned as tuples of two or three values, depending if this object grid is associated to a 2D or 3D environment.

position: tuple The position whose neighbourhood we wish to search.

tuple The moore position with the smallest value.

get_max_in_neigh(*position*)

Gets the coordinates of the moore neighbour with the largest value. If multiple neighbours meet the criteria, any may be returned.

A moore neighbourhood is defined as all positions immediately adjacent the one we are searching. It does not include the target position itself.

Positions should be given and are returned as tuples of two or three values, depending if this object grid is associated to a 2D or 3D environment.

position: tuple The position whose neighbourhood we wish to search.

tuple The moore position with the largest value.

class `Environment.`**NumericalGrid2D**(*name*, *xsize*, *ysize*, *model*)

Bases: *`Environment.Grid2D`*, *`Environment.NumericalGrid`*, `object`

Instantiates a 2D Numerical Grid. This extends Grid2D and NumericalGrid, allowing to store values in a two-dimensional grid exposing all methods offered by the aforementioned classes.

name [string] The name of the environment, this will be used when referring to it throughout the code. (Eg: AgentEnv, OxygenEnv, etc.)

xsize [int] The number of positions along the x-axis. In other words, the width of the environment.

ysize [int] The number of positions along the y-axis. In other words, the height of the environment.

model [model] The instance of the model class to which the environment will be attached.

class `Environment.`**NumericalGrid3D**(*name*, *xsize*, *ysize*, *zsize*, *model*)

Bases: *`Environment.Grid3D`*, *`Environment.NumericalGrid`*, `object`

Instantiates a 3D Numerical Grid. This extends Grid3D and NumericalGrid, allowing to store values in a three-dimensional grid exposing all methods offered by the aforementioned classes.

name [string] The name of the environment, this will be used when referring to it throughout the code. (Eg: AgentEnv, OxygenEnv, etc.)

xsize [int] The number of positions along the x-axis. In other words, the width of the environment.

ysize [int] The number of positions along the y-axis. In other words, the height of the environment.

model [model] The instance of the model class to which the environment will be attached.

class `Environment.`**ObjectGrid**

Bases: `object`

Initializes an ObjectGrid. Object grids at each position hold collections of objects. Most likely these would be instances of agents.

This class provides common methods to and, move and remove agents from the grid. It also exposes methods to get agent densities at various positions.

This class would **not** be itself instantiated, but would be extended by another class that would implement it.

`add_agent` (*agent*, *position*)

Adds an agent to a position in the environment.

This does *not* check if an agent already exists at another position. This should be checked separately.

This class does *not* update the internal state of the agent. So, if the agent also keeps its own record of its position in the grid, this should be updated separately.

If an invalid position is provided, then the agent will not be added.

agent [Agent] The instance of the agent we wish to update.

position: tuple The position to which we wish to add the agent.

`get_least_populated_moore_neigh` (*position*)

Gets the coordinates of the moore neighbour with the fewest agents. If multiple neighbours meet the criteria, any may be returned.

A moore neighbourhood is defined as all positions immediately adjacent the one we are searching. It does not include the target position itself.

Positions should be given and are returned as tuples of two or three values, depending if this object grid is associated to a 2D or 3D environment.

position: tuple The position whose neighbourhood we wish to search.

tuple The moore position with the fewest number of agents.

`get_most_populated_moore_neigh` (*position*)

Gets the coordinates of the moore neighbour with the most agents. If multiple neighbours meet the criteria, any may be returned.

A moore neighbourhood is defined as all positions immediately adjacent the one we are searching. It does not include the target position itself.

Positions should be given and are returned as tuples of two or three values, depending if this object grid is associated to a 2D or 3D environment.

position: tuple The position whose neighbourhood we wish to search.

tuple The moore position with the highest number of agents.

`move_agent` (*agent*, *position_old*, *position_new*)

Moves an agent from a grid position to another grid position.

This class does *not* update the internal state of the agent. So, if the agent also keeps its own record of its position in the grid, this should be updated separately.

It is up to the developer to check that the old position did indeed contain the agent. If an invalid position is provided as a new position, the agent will not be moved.

Positions should be given as tuples of two or three values, depending if this object grid is associated to a 2D or 3D environment.

agent [Agent] The instance of the agent we wish to update.

position_old [tuple] The old position of the agent.

position_new [tuple] The new position of the agent.

remove_agent(*agent*, *position*)
Removes an agent from a position.

This class does *not* update the internal state of the agent. So, if the agent also keeps its own record of its position in the grid, this should be updated separately.

Positions should be given as tuples of two or three values, depending if this object grid is associated to a 2D or 3D environment.

agent [Agent] The instance of the agent we wish to update.

position: tuple The position from which we wish to remove the agent.

class Environment.**ObjectGrid2D**(*name*, *xsize*, *ysize*, *model*)
Bases: *Environment.Grid2D*, *Environment.ObjectGrid*, object

Instantiates a 2D Object Grid. This extends Grid2D and ObjectGrid, allowing to place objects in a twp-dimensional grid exposing all methods offered by the aforementioned classes.

name [string] The name of the environment, this will be used when referring to it throughout the code. (Eg: AgentEnv, OxygenEnv, etc.)

xsize [int] The number of positions along the x-axis. In other words, the width of the environment.

ysize [int] The number of positions along the y-axis. In other words, the height of the environment.

model [model] The instance of the model class to which the environment will be attached.

class Environment.**ObjectGrid3D**(*name*, *xsize*, *ysize*, *zsize*, *model*)
Bases: *Environment.Grid3D*, *Environment.ObjectGrid*, object

Instantiates a 3D Object Grid. This extends Grid3D and ObjectGrid, allowing to place objects in a three-dimensional grid exposing all methods offered by the aforementioned classes.

name [string] The name of the environment, this will be used when referring to it throughout the code. (Eg: AgentEnv, OxygenEnv, etc.)

xsize [int] The number of positions along the x-axis. In other words, the width of the environment.

ysize [int] The number of positions along the y-axis. In other words, the height of the environment.

zsize [int] The number of positions along the z-axis. In other words, the depth of the environment.

model [model] The instance of the model class to which the environment will be attached.

2.2 Model

class Model.**Model**(*epochs*, *verbose=True*, *properties={}*)
Bases: object

Initializes a model object. The model object is the primary component of each simulation, holding the schedule, the environments, model properties, and the current progress in the simulation.

Essentially, the model holds a snapshot of the simulation world at any point in progress.

epochs [int] The number of epochs the simulation should run for.

verbose [bool, optional] If set to true, output is sent to standard output. If set to false, output (Ie: print statements) is disabled. Defaults to true.

properties: dict, optional Specifies a dictionary of property values. This can follow any format he developers need and should be adapted to the simulation's needs. Defaults to an empty dictionary.

run()
Runs the simulation for the number of epochs configured or until an the exit flag is set to true.

Note that the state of the schedule, environments etc. will result altered after the model runs. If you wish to run the same model multiple times, you should first copy the original instance to a backup variable.

2.3 Schedule

class `Schedule.`**Schedule**
Bases: `object`

Holds all simulation steppables and provides methods to progress simulation epochs.

Agents should be added to *agentsToSchedule* and not to *agents*, they will then be copied to the schedule before the start of the next epoch.

Similarly, agents which are to be removed should be added ot *agentsToRemove* and will be removed before the start of the next epoch.

The list *agents* should not be accessed directly.

The list *helpers* should be set during simulation setup.

step_epilogues(*model*)
Executes the StepEpilogue method of all helpers first and all agents after.

It is unlikely that this method would be called directly, but rather would be called as part of StepSchedule.

model [Model] The instance of the model to which the schedule is bound.

step_mains(*model*)
Executes the StepMan method of all helpers first and all agents after.

It is unlikely that this method would be called directly, but rather would be called as part of StepSchedule.

model [Model] The instance of the model to which the schedule is bound.

step_prologues(*model*)
Executes the StepPrologue method of all helpers first and all agents after.

It is unlikely that this method would be called directly, but rather would be called as part of StepSchedule.

model [Model] The instance of the model to which the schedule is bound.

step_schedule(*model*)
Adds and removes agents from the schedule as appropriate and executes all step methods of agents and helpers as appropriate.

model [Model] The instance of the model to which the schedule is bound.

2.4 Steppables

class `Steppables.`**Agent**
Bases: *`Steppables.Steppable`*

An agent represents a self-contained unit in the simulation with a state, a beahaviour and which may interact with, be affected by and affect the environment and other agents. A simulation may have multiple agent classes.

Examples of agent classes may include people, tissue cells, etc.

add_agent_to_grid(*environment_name*, *position*, *model*)

Adds an agent to a position in a grid. This both updates the state of the grid (via API calls to the grid object) **and** the internal state of the agent.

This does **not** check if the agent already exists in the grid. So, it potentially allows for the agent to be added to the same grid multiple times.

This method does check whether a position is valid, and where that is not the case the agent is not added and a warning is printed to screen.

environment_name [string] The name of the environment to which the agent should be added. This should match the name property in the environment object.

position [tuple] The position to which the agent will be added. This should be given as a tuple of two or three values, depending if this object grid is associated to a 2D or 3D environment.

model [Model] The instance of the model on which the simulation is based.

move_agent(*environment_name*, *position_new*, *model*)

Moves an agent from a position to another in an environment.

This both updates the state of the grid (via API calls to the grid object) **and** the internal state of the agent.

This method does check whether a position is valid, and where that is not the case the agent is not moved and a warning is printed to screen.

environment_name [string] The name of the environment to which the agent should be added. This should match the name property in the environment object.

position_new [tuple] The position to which the agent will be moved to. This should be given as a tuple of two or three values, depending if this object grid is associated to a 2D or 3D environment.

model [Model] The instance of the model on which the simulation is based.

remove_agent(*model*)

Removes an agent from the simulation. This removes the agent all environments and from the schedule.

In removing an agent from all environments, the internal state of the agent is also updated.

In practice, the agent is not immediately removed from the schedule but is added to the list of agents to remove and will be removed at the following epcoh.

model [Model] The instance of the model on which the simulation is based.

remove_agent_from_grid(*environment_name*, *model*)

Removes an agent from an environment.

This both updates the state of the grid (via API calls to the grid object) **and** the internal state of the agent.

This method does not check if an agent exists in an environment, but if it doesn't the method is trivially void.

environment_name [string] The name of the environment to which the agent should be added. This should match the name property in the environment object.

model [Model] The instance of the model on which the simulation is based.

class Steppables.**Helper**

Bases: *Steppables.Steppable*

A placeholder class to allow for helper steppables to have their own data-type.

A helper allows to encapsulate logic that should be executed at each time-step, but which does not belong to any agent. Common examples could include functions to save text or graphics to an output file, record model properties, update environment properties, etc.

This class does not provide methods or attributes, but allows for the "Helper" data-type to exist which is a bit more informative than simply having objects of 'Steppable' type.

class `Steppables.`**`Steppable`**

Bases: `object`

Represents generic steppable object. Steppables represent entities which can be added to the schedule and whose logic is excuted once per epoch.

Steppables implement a prologue, main and epilogue. All prologues for all steppables are executed first, followed by all mains followed by all epilogues.

`step_epilogue`(*model*)

Placeholder to enforce that all steppables implement a stepEpilogue method with the correct signature. Per se, this method does nothing but can be overridden by child classes.

model [Model] The instance of the model to which the schedule to which the agent belong is bound.

`step_main`(*model*)

Placeholder to enforce that all steppables implement a stepMain method with the correct signature. Per se, this method does nothing but can be overridden by child classes.

model [Model] The instance of the model to which the schedule to which the agent belong is bound.

`step_prologue`(*model*)

Placeholder to enforce that all steppables implement a stepPrologue method with the correct signature. Per se, this method does nothing but can be overridden by child classes.

model [Model] The instance of the model to which the schedule to which the agent belong is bound.

CHAPTER 3

Famework Tookit

class Toolkit.**AgentSummary**(*record_every=1*)
Bases: `panaxea.core.Steppables.Helper`, `object`

Helper class which, in the epilogue of each epoch, returns a summary of the count of agents belonging to each class considering the union of the schedule and agentsToSchedule.

recordEvery [int, optional] Defines every how many epochs an agent summary is returned. Defaults to 1. (Ie: Every epoch)

step_epilogue(*model*)
Builds a summary of number of agents per agent class in the model.

model [Model] An instance of the model on which the current simulation is based.

Counter A counter object where keys correspond to agent classes and values to agent counts.

class Toolkit.**ModelPickler**(*out_dir*)
Bases: `panaxea.core.Steppables.Helper`, `object`

At each epoch, outputs a serialized copy of the model in its current state.

outDir [string] The directory where pickle files should be outputted. This should be specified as relative to the script from which the simulation is launched

step_epilogue(*model*)
Creates and saves the pickle file.

model [Model] An instance of the model on which the current simulation is based.

class Toolkit.**ModelPicklerLite**(*out_dir*, *prefix=None*, *pickle_every=1*, *pickle_schedule=False*, *pickle_envs=False*)
Bases: `panaxea.core.Steppables.Helper`, `object`

Creates a lighter version of the pickle allowing to include or exclude specific elements.

outDir [string] The directory where pickle files should be outputted. This should be specified as relative to the script from which the simulation is launched

prefix [string, optional] A prefix that will be given to the name of each output file. Eg: For a prefix "my_model" a sample output file would be my_model_epoch_0.pickle Defaults to None

pickleEvery: number, optional Determines the frequency of model serializing. A value of 1 will create one pickle per epoch, a value of 2 will create a pickle every other epoch, etc. Defaults to 1.

pickleSchedule [bool, optional] If set to true, the schedule object will be included. This will also include all agents on the schedule. Defaults to false.

pickleEnvs [bool, optional] If set to true, all environment objects will be included. This also includes all agents in every environment. Defaults to false.

`pickle_model`(*model*)

Creates and serializes the pickleLight object based on previously defined properties.

model [Model] An instance of the model on which the current simulation is based.

`step_epilogue`(*model*)

Makes a call to the pickleModel method. No special logic here, just delegating to the method.

model [Model] An instance of the model on which the current simulation is based.

`Toolkit.`**`depickle_from_lite`**(*picklePath*)

Given a path to a pickle light file, recreates the corresponding object with all available properties

This is **not** a helper and **should not be added to the schedule**. It is useful to recreate (partial) model objects.

This model may or may not be runnable when recreated depending on whether all properties (schedule, environments…) were retained.

picklePath [string] The path to the pickle file relative to where the function is being called from.

Model A (potentially incomplete) instance of a model derived from the pickle file.

Python Module Index

Index