Mesa Time Module

Objects for handling the time component of a model. In particular, this module contains Schedulers, which handle agent activation. A Scheduler is an object which controls when agents are called upon to act, and when.

The activation order can have a serious impact on model behavior, so it's important to specify it explicitly. Example simple activation regimes include activating all agents in the same order every step, shuffling the activation order every time, activating each agent *on average* once per step, and more.

Key concepts:

Step: Many models advance in 'steps'. A step may involve the activation of all agents, or a random (or selected) subset of them. Each agent in turn may have their own step() method.

Time: Some models may simulate a continuous 'clock' instead of discrete steps. However, by default, the Time is equal to the number of steps the model has taken.

class **BaseScheduler(*model: Model*)** [source]

Simplest scheduler; activates agents one at a time, in the order they were added.

Assumes that each agent added has a *step* method which takes no arguments.

(This is explicitly meant to replicate the scheduler in MASON).

Create a new, empty BaseScheduler.

add(*agent: Agent*) → None [source]

Add an Agent object to the schedule.

Args:

agent: An Agent to be added to the schedule. NOTE: The agent must have a step() method.

remove(*agent: Agent*) → None [source]

Remove all instances of a given agent from the schedule.

Args:

agent: An agent object.

step() → None [source]

Execute the step of all the agents, one at a time.

get_agent_count() → int [source]

Returns the current number of agents in the queue.

agent_buffer(*shuffled: bool = False*) → Iterator[Agent] [source]

Simple generator that yields the agents while letting the user remove and/or add agents during stepping.

---

class **RandomActivation(*model: Model*)** [source]

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

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

Assumes that all agents have a step(model) method.

Create a new, empty BaseScheduler.

step() → None [source]

Executes the step of all agents, one at a time, in random order.

add(*agent: Agent*) → None

Add an Agent object to the schedule.

Args:

agent: An Agent to be added to the schedule. NOTE: The agent must have a step() method.

agent_buffer(*shuffled: bool = False*) → Iterator[Agent]

Simple generator that yields the agents while letting the user remove and/or add agents during stepping.

get_agent_count() → int

Returns the current number of agents in the queue.

remove(*agent: Agent*) → None

Remove all instances of a given agent from the schedule.

Args:

agent: An agent object.

---

class **SimultaneousActivation(*model: Model*)** [source]

A scheduler to simulate the simultaneous activation of all the agents.

This scheduler requires that each agent have two methods: step and advance. step() activates the agent and stages any necessary changes, but does not apply them yet. advance() then applies the changes.

Create a new, empty BaseScheduler.

step() → None [source]

Step all agents, then advance them.

add(*agent: Agent*) → None

Add an Agent object to the schedule.

Args:

agent: An Agent to be added to the schedule. NOTE: The agent must have a step() method.

agent_buffer(*shuffled: bool = False*) → Iterator[Agent]

Simple generator that yields the agents while letting the user remove and/or add agents during stepping.

get_agent_count() → int

Returns the current number of agents in the queue.

remove(*agent: Agent*) → None

Remove all instances of a given agent from the schedule.

Args:

agent: An agent object.

---

class **StagedActivation(*model: Model, stage_list: list[str] | None = None, shuffle: bool = False, shuffle_between_stages: bool = False*)** [source]

A scheduler which allows agent activation to be divided into several stages instead of a single *step* method. All agents execute one stage before moving on to the next.

Agents must have all the stage methods implemented. Stage methods take a model object as their only argument.

This schedule tracks steps and time separately. Time advances in fractional increments of 1 / (# of stages), meaning that 1 step = 1 unit of time.

Create an empty Staged Activation schedule.

Args:

model: Model object associated with the schedule. stage_list: List of strings of names of stages to run, in the

order to run them in.

shuffle: If True, shuffle the order of agents each step. shuffle_between_stages: If True, shuffle the agents after each

stage; otherwise, only shuffle at the start of each step.

step() → None [source]

Executes all the stages for all agents.

add(*agent: Agent*) → None

Add an Agent object to the schedule.

Args:

agent: An Agent to be added to the schedule. NOTE: The agent must have a step() method.

agent_buffer(*shuffled: bool = False*) → Iterator[Agent]

Simple generator that yields the agents while letting the user remove and/or add agents during stepping.

get_agent_count() → int

Returns the current number of agents in the queue.

remove(*agent: Agent*) → None

Remove all instances of a given agent from the schedule.

Args:

agent: An agent object.

---

class **RandomActivationByType(*model: Model*)** [source]

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

The *step_type* method is equivalent to the NetLogo 'ask [breed]...' and is generally the default behavior for an ABM. The *step* method performs *step_type* for each of the agent types.

Assumes that all agents have a step() method.

This implementation assumes that the type of an agent doesn't change throughout the simulation.

If you want to do some computations / data collections specific to an agent type, you can either: - loop through all agents, and filter by their type - access via *your_model.scheduler.agents_by_type[your_type_class]*

Create a new, empty BaseScheduler.

add(*agent: Agent*) → None [source]

Add an Agent object to the schedule

Args:

agent: An Agent to be added to the schedule.

remove(*agent: Agent*) → None [source]

Remove all instances of a given agent from the schedule.

step(*shuffle_types: bool = True, shuffle_agents: bool = True*) → None [source]

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

Args:

shuffle_types: If True, the order of execution of each types is

shuffled.

shuffle_agents: If True, the order of execution of each agents in a

type group is shuffled.

step_type(*type_class: type[Agent], shuffle_agents: bool = True*) → None [source]

Shuffle order and run all agents of a given type. This method is equivalent to the NetLogo

'ask [breed]...'.

Args:

type_class: Class object of the type to run.

get_type_count(*type_class: type[Agent]*) → int [source]

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

agent_buffer(*shuffled: bool = False*) → Iterator[Agent]

Simple generator that yields the agents while letting the user remove and/or add agents during stepping.

get_agent_count() → int

Returns the current number of agents in the queue.