JASSS
THE JOURNAL OF ARTIFICIAL SOCIETIES AND SOCIAL SIMULATION

Supplementary Materials to:
Journal of Artificial Societies and Social Simulation **28** (4) 11 <https://www.jasss.org/28/4/11.html>
DOI: 10.18564/jasss.5831

AGENTBLOCKS: a community platform for sharing, comparing, and improving reusable building blocks for (agent-based) models

Tatiana Filatova[1], Liz Verbeek[1], Martin Warnier[1], Amineh Ghorbani[1], Igor Nikolic[1], Volker Grimm[2,3], Uta Berger[4], Michael Barton[5], Andrew Bell[6], Allen Lee[5], Nicholas Magliocca[7] and Thorid Wagenblast[1]

1 Delft University of Technology, Faculty of Technology Policy and Management, Department of Multi Actor Systems, Jaffalaan 5, 2628 BX, Delft The Netherlands
2 Helmholtz Centre for Environmental Research – UFZ, Department of Ecological Modelling, Permoserstr. 15, 04318 Leipzig, Germany
3 University of Potsdam, Plant Ecology and Nature Conservation, Zeppelinstraße 48A, 14471 Potsdam, Germany
4 TU Dresden, Faculty of Environmental Sciences, Department of Forest Sciences, 01062 Dresden, Germany
5 School of Complex Adaptive Systems, Arizona State University, 1031 S. Palm Walk
6 Tempe, AZ 85281-2701, USA
7 Department of Global Development, Cornell University, Warren 266, Cornell University, Ithaca, NY, 14850, USA
8 Department of Geography, The University of Alabama, Box 870322, Tuscaloosa, AL 35401, USA

Table of Contents

Appendix A: Templates for a Reusable Building Block and Diagrams

This template summarizes all components of an RBB, see also the online template: https://www.agentblocks.org/assets/frontend/media/RBB_template.pdf. Once you log in to contribute an RBB to the platform, you will be asked to describe these components for your RBB.

Every RBB consists of two parts:

Part 1: RBB 'Main Description' ('Main' in the online version) – is the general description of an RBB, independent of model-specific design choices or programming language. It provides information on the theoretical foundations, context, purpose and scale, and describes a common shape this RBB takes in an ABM: what (types) of agents does it concern, and how is it connected to the rest of the model in terms of input/output?

Part 2: RBB 'Implementation' ('Impl.N[Language]' in the online version) – is a model-specific description of an RBB, including its implementation in a specific programming language, so that various *RBB Implementations* can co-exist for a single RBB. For example, this applies to domain-specific alternatives representing the same process. Part 2 provides information on the process flow, the (model-specific) input and output parameters, the position of the RBB in the full ABM, and contains a working implementation in a specific programming language.

The 'Main Description' and 'Implementation' parts of the RBB consist of the obligatory and optional components, following the two-tier approach (Berger at al 2024). Besides programming language and discipline, the list of fields one needs to provide information for includes:

Part 1: RBB 'Main Description'	Fields	Function
Identifier	Title	Needed for the citable references
	Authors	
Background and purpose	Background & purpose	Describe the purpose of this RBB. Communicate the underlying problem and context where this process/action matters (in the real world and/or in the full ABM). Describe theoretical foundations of the modelled process/action.
	[* Optional] Overview figure	Provide a visual overview of the theory or process captured by this RBB. You may provide an existing figure, or create one using this template. Do not mix it with figure of the full ABM: please provide here only the conceptual figure describing the relevant RBB to help others understand what it models.
Key concepts and definitions	Key concepts and definitions	Communicate the underlying semantic ontology, including key concepts and underlying assumptions. How can this RBB parameterized? What kind of additional information or empirical data are needed to inform this RBB?
	[* Optional] Scale	If relevant, describe spatial and temporal scales/resolution at which this RBB applies.

Detailed specification	Input description	Provide a verbal description of a (common) input to this RBB (on a conceptual level, any implementation-specific inputs and numerical values may be provided in **Part 2**)
	Output description	Provide a verbal description of a (common) output of this RBB (on a conceptual level, any implementation-specific inputs and numerical values may be provided in **Part 2**)
	Level of interactions	Specify processes/actions at which level of interactions – just within a single agent, among agents or between agent and its environment – your RBB aims to capture. Choose from: • Single Agent • Multiple Agents(same type) • Multiple Agents (different types) • Agent(s) and the Environment
	Agent definition	For all types of agents - based on the level of interactions specified above – provide: 1. Agent type (e.g. household, farmer, government, ...) 2. Attributes of this agent type 3. Functions of this agent type

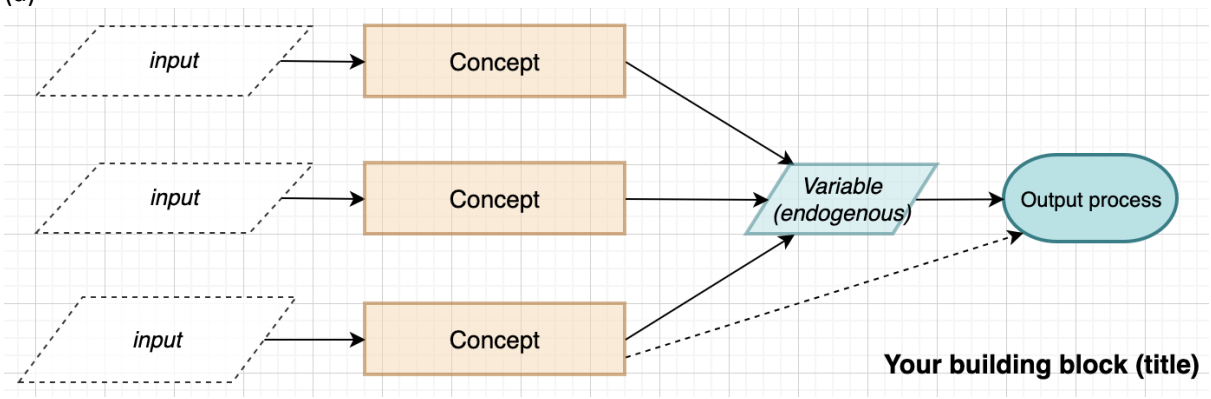
Part 2: RBB 'Implementation'	Fields	Function
Process flow	Flow chart	Visual overview of the process flow of this implementation of the RBB. You may provide an existing figure, or create one using this template
Input and output	Input	Provide the input parameters required for this implementation of the RBB
	Output	Provide the output produced by your implementation of the RBB (this may be output parameters, or changes to the model- or agent states)
Position in an ABM	Sequence diagram	Visual overview of the position of the RBB in an ABM. You may provide an existing figure, or create one using this template
Pseudocode	Pseudocode	Describe the process flow in the form of pseudocode
Reusable code block	Code	Provide a *working* example of the RBB as it is employed in your ABM in a specific programming language
Agent-based model	Model description	Provide a short description of the ABM in which you employ this RBB.

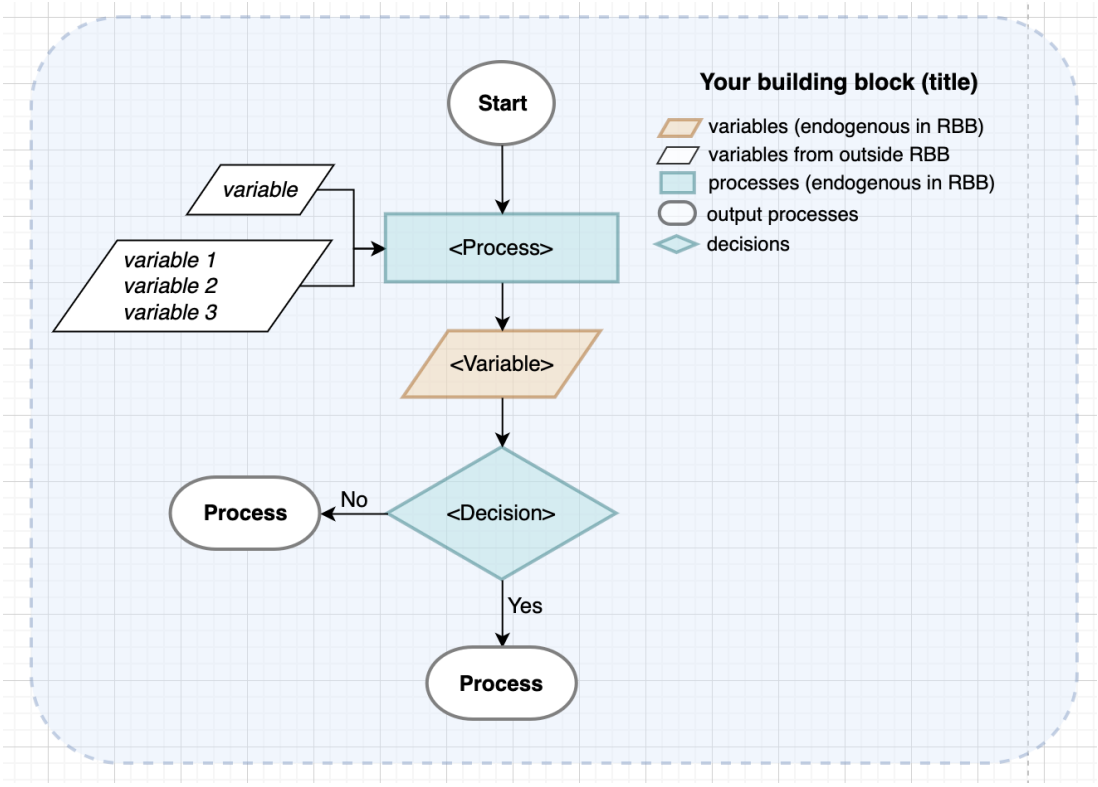
	Model code	Provide a link to the original model code
	[* Optional] Published material	Provide a link to published material describing the full model

Besides the Template for an RBB, we also offer Templates for the development of the diagrams. Often agent-based modelers already have a diagram for the full ABM but not always for the specific code snippet they are ready to make an RBB from. In case you do not have diagrams of the process flow of the targeted RBB or its position in the full ABM, you can create one using these open access online Diagram templates. To access the editable version, please select 'Edit' in the bottom panel and mind that there are three tabs depicting different diagrams:

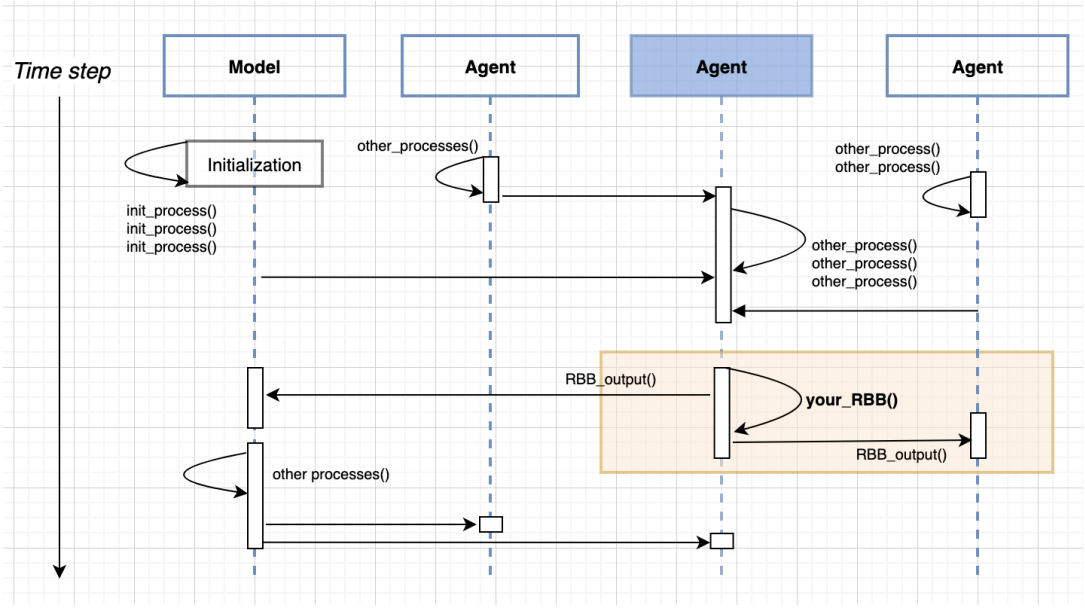
The AGENTBLOCKS platform currently offers three diagram templates with editable drag & drop elements (Figure A.1). If you want to offer another template, feel free to reach out to us or just add it using the "+" sign in the online Diagrams app.

(a)





(b)



(c)

Figure A.1: Examples of open access templates for diagrams to facilitate the meta data description of a Reusable Building Block. Panel (a) offers the template for the 'Conceptual overview' of an RBB meant to supplement Part 1 – RBB 'Main Description'. Panel (b) offers the template for the 'Flow chart' diagram for the core processes coded by an RBB. Panel (c) offers the template for the 'Sequence diagram' to indicate the position of the RBB in the full ABM. The last two diagrams are meant to support Part 2 – RBB 'Implementation'.

Appendix B: Architecture of the open access AGENTBLOCKS Platform for reusable building blocks

Figure B.1 provides the ontological structure of the backend of the AGENTBLOCKS repository of RBBs.

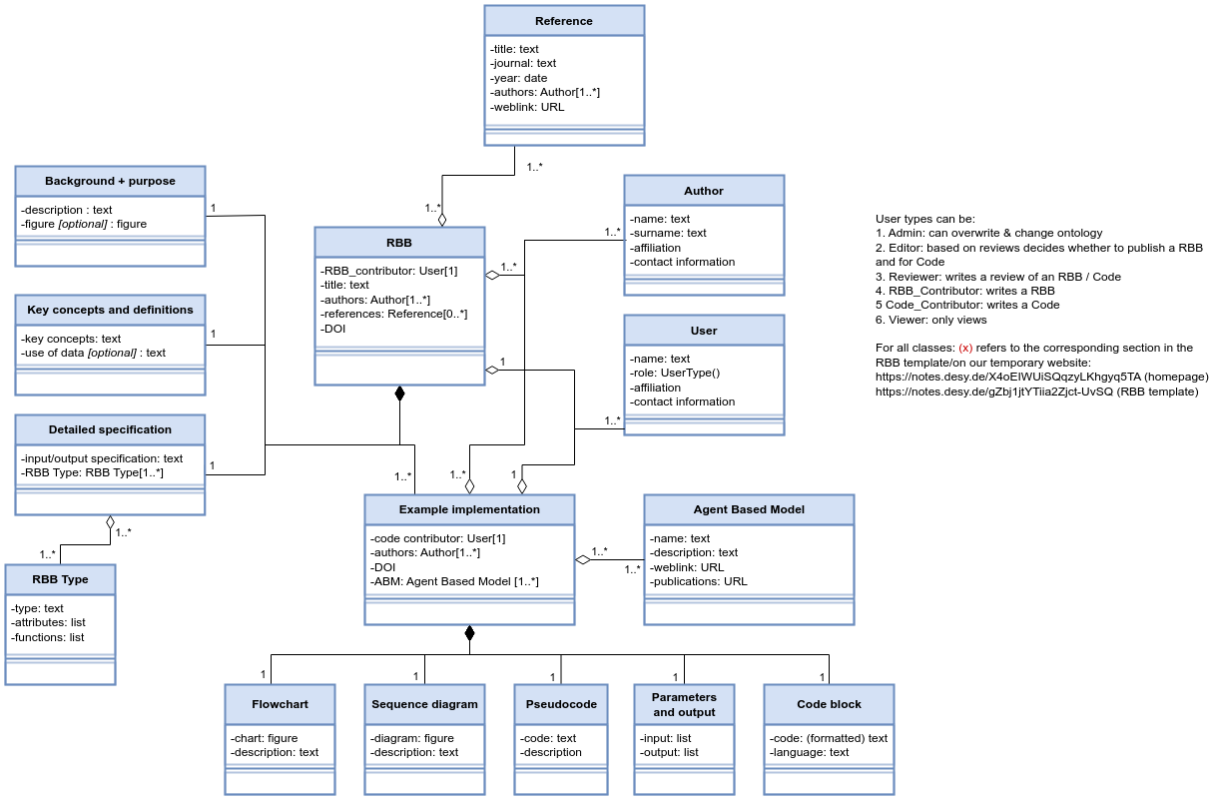


Figure B.1: Relational database in the backend of the AGENTBLOCKS platform for reusable building blocks

Appendix C: Examples of the first Reusable Building Blocks

Example 1: Theory of Planned Behavior

See the latest version online: https://www.agentblocks.org/rbb/under-development-theory-of-planned-behavior-for-agents-decision-making

Part 1: RBB 'Main Description'

Identifier: Title, Authors:

Theory of Planned Behavior for individual decision-making – Tatiana Filatova, Liz Verbeek

Background and purpose:

Background & purpose:

Developed in the early 1990s by Ajzen [1], the Theory of Planned Behavior is one of the most influential theories in social and health psychology, and has been used in many environmental studies to describe behavioral change. In ABMs, TPB is often used as an alternative to perfect rationality in individual decision-making processes.

TPB assumes that a person's intention to change their behavior is driven by: 1) their attitude towards the behavior, 2) subjective norms (perceived social pressure to perform the behavior), and 3) perceived behavioral control (the degree to which they believe they're able to perform the behavior).

The figure below shows these three core components of TPB.

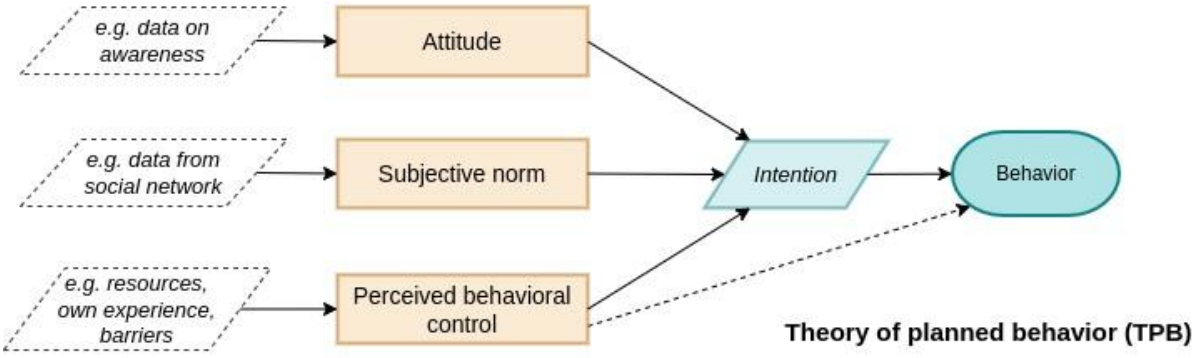
Empirical survey studies have demonstrated that TPB performs well in explaining patterns of real-world behavior changes across a wide range of applications, including actions like improving health, investing in new resource-efficient technologies, choosing travel modes, nature and resource conservation, choosing organic food, using bioplastics, adapting to climate change and so on. Consequently, in ABMs, TPB is used to study e.g. technology diffusion among households; migration; decision-making processes by farmers; waste recycling; urban development and travel behavior (see [2] for review of applications).

References

[1] Ajzen, I. (1991). The theory of planned behavior. Organizational behavior and human decision processes, 50(2), 179-211. https://doi.org/10.1016/0749-5978(91)90020-T

[2] Muelder, H. and Filatova, T. (2018). One theory-many formalizations: Testing different code implementations of the theory of planned behaviour in energy agent-based models. Journal of Artificial Societies and Social Simulation, 21(4), 5. https://doi.org/10.18564/jasss.3855

[* Optional] Overview figure:



Key concepts and definitions

Key concepts and definitions:

A change in behavior happens through pursuing certain actions. As mentioned above, TPB assumes that the decision to pursue a certain (behavior changing) action is driven by three groups of behavioral and socio-economic factors: **attitudes**, **subjective norms** and **perceived behavioral control** [1].

Attitudes describe an individuals own judgement of whether specific behavior is positive and to what degree it is favorable.

The subjective norm component in TPB captures the influence of external social pressure: one's perception of whether others expect them to perform certain actions.

Finally, person naturally should have control over their decision to perform certain behavior, as well as over opportunities and resources to pursue the 'planned' action. This is described by the perceived behavioral control (PBC) component, which indicates a subjective judgement about one's own ability to implement the behavior (based on e.g. general ease of implementation, past experience, perceived barriers, or self-efficacy).

Like in most behavioral theories, TPB assumes that behavior is predicted by **intentions** (i.e. the stronger the intention towards a specific behavior, the more likely the person will engage in it), but this intention does not necessarily result in immediate action (e.g. because of a time lag or prioritization of other measures) [2].

In some ABM implementations, intentions and actions are equated (having the intention to act directly leads to action). Since in reality having the intention to do something does not always (and usually does not) directly result in taking action, we recommend to model this so-called i**ntention-behavior gap** explicitly, for example by adding stochasticity serving as a probabilistic barrier between intention and action.

Use of data

To formalize TPB in an agent-based model, the above-mentioned components (attitude, subjective norm and PBC) need to be quantified in some way. Usually, this is achieved by collecting (or utilizing existing) survey data on the behavior of interest.

In such surveys, attitudes are typically parameterized by knowledge, awareness and overall evaluation of the topic, and can be quantified by measuring (dis)agreement with specific statements related to the topic.

The parameterization of subjective norm typically includes data on the number of people in a social circle (e.g. friends, neighbors, family) who have already implemented the behavior of interest, and individual perceptions of the expectations of these peers regarding the particular action. In, ABMs the subjective norm can be updated endogenously as simulation unfolds and the number of agents pursuing or abandoning the action changes.

Survey data informing the PBC component describes financial constraints (e.g. payback period; whether one has sufficient savings or income to fund an action; the benefit/cost ratio of (not) performing the action), information (e.g. objective vs. biased information about technical specification or costs of an action), questions on self-efficacy (to what extent one believes to be able to perform the action) and past experience. Although frequently measured in surveys, the latter two are rarely included in formalizations of TPB in agent-based models.

References

[1] Ajzen, I. (1991). The theory of planned behavior. Organizational behavior and human decision processes, 50(2), 179-211. https://doi.org/10.1016/0749-5978(91)90020-T

[2] Weinstein, N. D., Rothman, A. J., & Nicolich, M. (1998). Use of correlational data to examine the effects of risk perceptions on precautionary behavior. Psychology and Health, 13(3), 479-501. https://doi.org/10.1080/08870449808407305

[* Optional] Scale

As TPB describes a decision-making process, it may operate on any scale in which choices by individual agents are relevant. It is thus scale-agnostic for spatial scale.

For its temporal extent it is important to note that the interval between deliberation moments should be realistic (e.g. considering adopting energy-efficient technologies like solar panels on a daily basis does not make sense; one or several timesteps per year would be a more realistic time scale). Of course, the total temporal extent of the ABM should follow the same reasoning and should match the timestep intervals.

Detailed specification

Input description

Any implementation of TPB in an agent-based model includes (at least) the following input factors:

1. Attitude towards the behavior
2. Subjective norm
3. Perceived behavioral control (PBC)

In addition, for some applications, and if data is available, the following factors may be included:

1. Self-efficacy (which may be included in the PBC component)
2. Past experience or behavior

Output description

Level of interactions

Single Agent

Agent definition

Agent type: Household

Description: A typical TPB process in an ABM parameterizes its three core components (attitude, subjective norm and PBC) as agents' attributes, where the agents typically represent households or individual people. These three factors form the input for the TPB process that computes the agents' intention to take action, which is in turn translated to a binary decision to take action or not.

Attributes:

- attitude
- subjective norm
- PBC

Functions:

- compute intention
- take action (yes/no?)

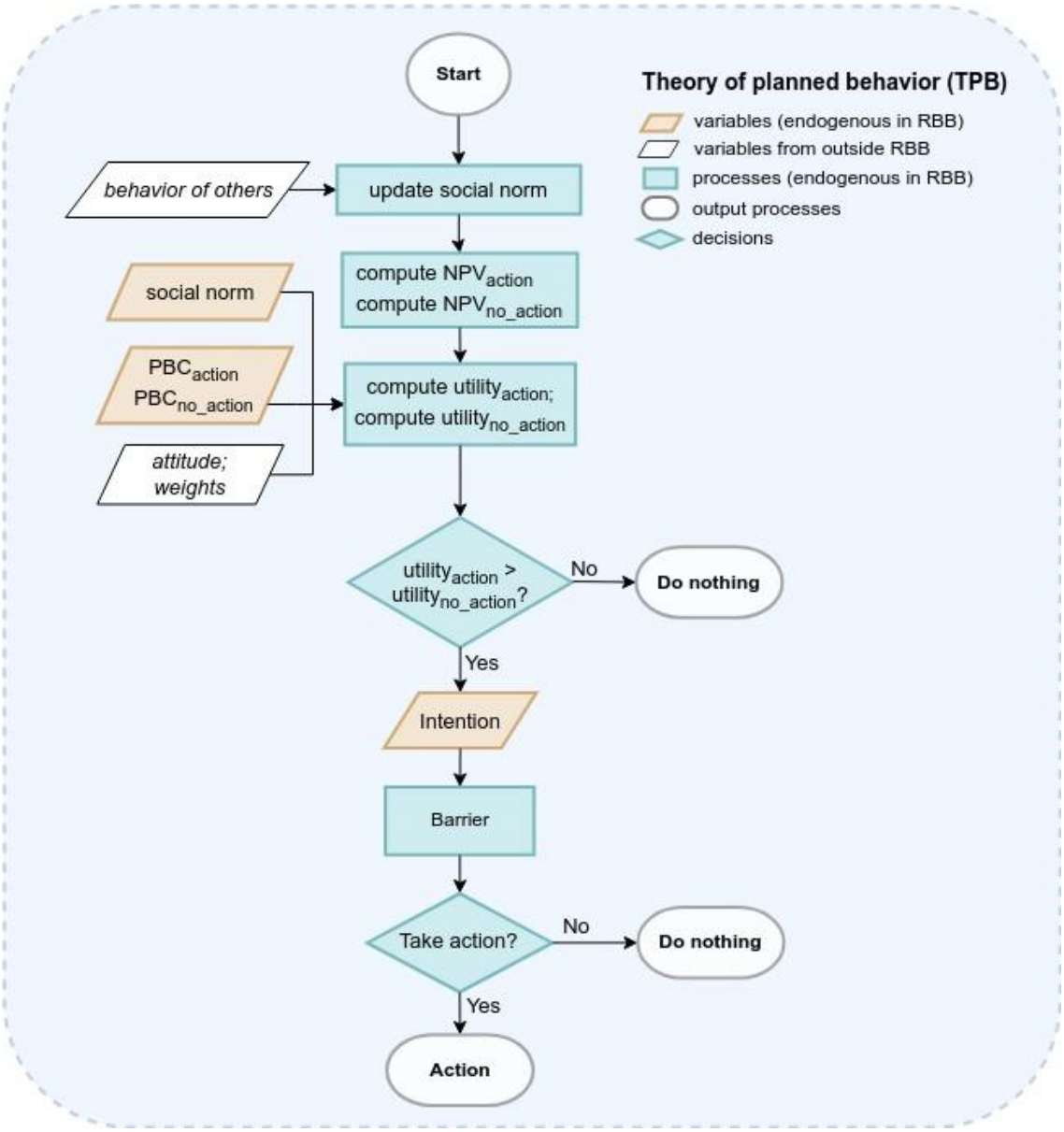
Part 2: RBB 'Implementation'

Description

This implementation of the TPB building block in the Energy ABM, where TPB represents the decision-making process of households in the Netherlands that consider installing PV panels. In this example implementation, TPB is formalized as a sequence of several processes. First, agents compare the (expected) utility of installing PV panels to the (expected) utility of not installing PV panels, which informs their intention to take action (or not). These utility functions are weighted averages of the TPB components - attitude, subjective norm and PBC.
The attitudes of the agents towards PV panels are parameterized using survey data on PV panel installation in the Netherlands (see link to the published material for more information on the survey).
The same survey data provides the weights for each of the input factors of the utility function(s).

It is assumed that subjective norms stem from social norms. The social norms in this model are formalized as the fraction of connections in an agent's social network (representing friends, family, neighbors) that have already installed PV panels. These social norms are dynamic: they are updated every timestep as PV uptake increases.
In the Energy ABM, it is assumed that PBC represents only financial constraints. It is formalized as an individual cost-benefit analysis of installing PV panels: every agent computes the Net Present Value (NPV) for installing PV panels (based on an expected payback time) as well as for doing nothing.
After computing an agent's intention to take action, a probabilistic barrier is added to represent the intention-behavior gap.

Process flow (Flow chart)



Notes

This flow chart represents the implementation of TPB in the Energy ABM. The subjective norms are formed by endogenous (dynamic) social norms arising from the

behavior of other agents. PBC is computed as the Net Present Value (cost-benefit analysis) of installing PV panels or doing nothing. Attitudes and weights are static and paramterized using survey data. Together, they contribute to the agent's (expected) utility of taking action or not.
After computing the utility for both taking action (installing PV panels) and doing nothing, the agent compares the two. If the utility of doing nothing is higher, the agent does not take action. If the utility of installing PV panels is higher, however, the agent considers - has the intention to - taking action. Before actually taking action, some stochasticity is added to represent the intention-behavior gap. Thus, some agents with the intention to take action actually do so, others don't.

Parameters and output

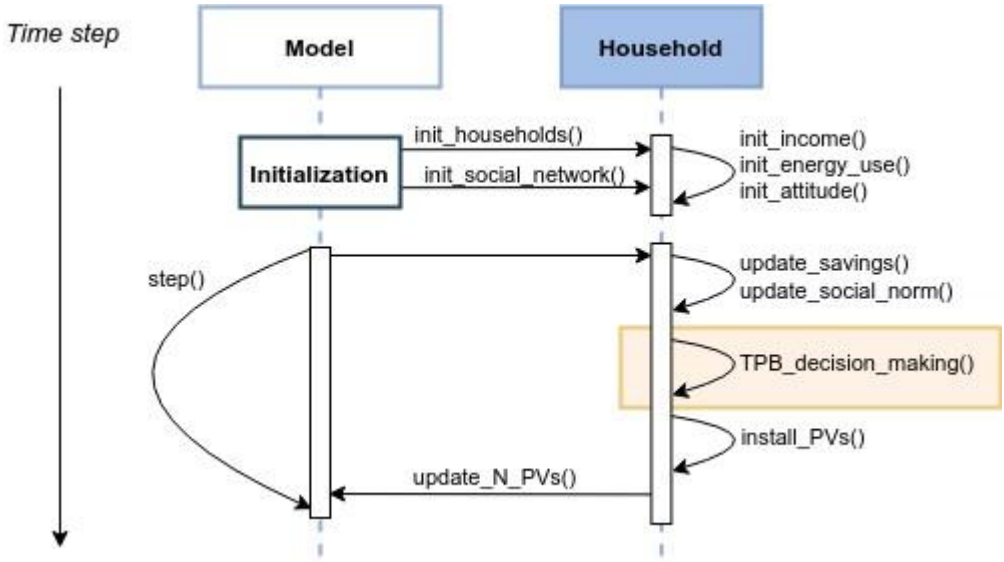
Input to RBB:

1. Behavior of others
2. Attitude
3. Weights

Output of RBB

1. Binary decision to take action

Position in an ABM



Pseudocode

function TPB(agent):
get attitude, social norm **from** agent
PBC_{action} ← NPV_{action}(expected costs, expected benefits)
PBC_{no_action} ← NPV_{no_action}(expected costs, expected benefits)
$utility_{action}$ ← weighted function of attitude, social_norm and PBC_{action}
$utility_{no\_action}$ ← weighted function of -attitude, social_norm and PBC_{no_action}
if $utility_{action}$ > $utility_{no\_action}$ **then**
barrier passed? ← Draw Bernoulli($p_{barrier}$)
if barrier passed? **then**
take action? = True

```
                    else
                        take action? = False
                    end if
                else
                    take action? = False
                end if
            return take action?
```

Reusable code block

Programming language: Python


```
# -*- coding: utf-8 -*-
"""
@author: e.verbeek@tudelft.nl
Python implementation of Theory of Planned Behavior for household decision-making. This code contains a
Household class employing the TPB decision-making module described by this building block. In addition, it
includes code for a Model class required to implement TPB decision-making in an agent-based model.

"""

import numpy as np

from scipy.stats import bernoulli
from mesa import Model
from mesa import Agent
from mesa.time import RandomActivation

class TPB_Model(Model):
    """Model class. """

    def __init__(self, n_households, n_connections, HH_TPB_weights, attitudes,
                 measure_benefits, measure_costs):
        """Initialization of an ABM employing TPB as decision-making module.

        Args:
            n_households (int)      : Number of households
            n_connections (int)     : Number of social connections per household
```

```
            HH_TPB_weights (list) : Homogeneous weights for TPB function
            attitudes (list)      : HH attitude towards taking action
            measure_benefits      : Estimated yearly benefits of taking action
            measure_costs         : Estimated yearly costs of taking action
        """

        # Keep track of agent IDs and number of households
        self.current_id = 0
        self.n_households = n_households
        # Add schedule. The type of schedule can be chosen by the user
        self.schedule = RandomActivation(self)

        # Save TPB weights (homogeneous)
        self.HH_TPB_weights = HH_TPB_weights
        # Initialize households
        for n in range(n_households):
            hh = Household(self, attitudes[n], measure_benefits[n], measure_costs)
            self.schedule.add(hh)
        # Add social connections (randomly select n)
        for hh in self.schedule.agents:
            others = [HH for HH in self.schedule.agents if HH != hh]
            hh.connections = np.random.choice(self.schedule.agents, n_connections)

        # Save maximum Net Present Value for normalization
        self.max_NPV = max(max(hh.compute_NPV() for hh in self.schedule.agents))

        # ... other model initialization functions ... #

    def step(self):
        """Model step"""
        self.schedule.step()

class Household(Agent):
    """Household agent class."""

    def __init__(self, model, attitude, measure_benefits, measure_costs):
        """Initialization of a household agent.

        Args:
            model (Model)     : Model containing the household agent
            attitude (list)   : Value for attitude towards taking action
        """
        super().__init__(model.next_id(), model)

        self.measure = False
        self.attitude = attitude
        # Parameters for Net Present Value estimation
        # Please note that for simplicity, the costs are homogeneous
        # in this example implementation, while in reality, of course, these
        # usually differ per household and depend on other factors
        self.measure_benefits = measure_benefits
        self.measure_costs = measure_costs

        # ... other Household initialization functions ...

    def TPB(self, TPB_weights, barrier=True, barrier_prob=0.5):
        """Theory of planned behavior for household decision-making.
```

```
        Args:
            TPB_weights  : Vector of predefined weights for PMT function.
                           Here, weights are estimated using logistic regression on survey data.
            barrier      : Boolean (True/False) to either apply or not apply probabilistic barrier between intention
and action
            barrier_prob : Probability to pass the barrier
        """

        social_norm_action, social_norm_no_action = self.get_social_norm()
        PBC_action, PBC_no_action = self.compute_NPV()
        # Normalize PBC
        PBC_action = PBC_action/self.model.max_NPV
        PBC_no_action = PBC_no_action/self.model.max_NPV
        # Get TPB input vectors
        TPB_vars_action = [self.attitude, social_norm_action, PBC_action]
        TPB_vars_no_action = [-self.attitude, social_norm_no_action, PBC_no_action]

        # Compute utility of taking action or not
        utility_action = TPB_weights @ TPB_vars_action
        utility_no_action = TPB_weights @ TPB_vars_no_action
        if utility_action > utility_no_action:
            # (Optional) Apply probabilistic barrier between intention and action
            if barrier:
                if bernoulli.rvs(barrier_prob) == 1:
                    self.measure = True
                    # ... other consequences of applying the measure ... #
            else:
                self.measure = True
                # ... other consequences of applying the measure ... #

    return

    def compute_NPV(self, timespan=10, interest_rate=0.05):
        """Compute Net Present Value (NPV) of taking action (or not).

        Args:
            timespan (int)      : Number of years to take into account
            interest_rate (float)  : Bank interest rate
        """

        # Yearly benefits (assumed these to not change over time)
        benefits = self.measure_benefits
        # Investment costs (only spend in first year)
        costs = np.zeros(timespan)
        costs[0] = self.measure_costs
        # Discount rates are typically based on bank interest rate
        timesteps = np.linspace(0, timespan - 1, timespan)
        discount_rates = (1 + interest_rate)**timesteps

        # Compute Net Present Value
        NPV_measure = sum((benefits - costs) / discount_rates)
        NPV_not_measure = sum((costs - benefits) / discount_rates)

        return NPV_measure, NPV_not_measure

    def get_social_norm(self):
```

```
        """Update household social norm based on actions of social connections"""

        # Get number of social connections that have taken measure or not
        taken_measure = sum(hh.measure for hh in self.connections)
        not_taken_measure = sum(not hh.measure for hh in self.connections)

        # Return as fraction of total social connections
        return (taken_measure/len(self.connections),     not_taken_measure/len(self.connections))

    def step(self):
        """Household agent step"""
        self.TPB(self.model.HH_TPB_weights)

# To run the model, please specify: n_households (int), HH_TPB weights (list of 3 weights)
# attitudes (list of attitude per household), measure_benefits (list), measure_costs (int)
# and run the following:
model = TPB_Model(n_households, n_connections, HH_TPB_weights, attitudes,
                          measure_beneasure_benefits, measure_costs)
n_steps = 100 for n in range(n_steps): model.step()
```

Code instructions:
To run this example code, specify the number of households, weights for the TPB building block (list of 3), attitude values for all households, a list of the financial benefit of implementing the measure per agent (this can also be determined endogenously in the model) and the costs of implementing the measure.

Agent-based model
Model name: Energy ABM

Model description:
The TPB building block is employed in the Energy ABM for household decision-making on PV panel installation. This model can be used to estimate the uptake of solar panels and resulting changes in greenhouse gas emissions (GHG) in the Netherlands.
In the model, three household decision-making modules can be compared: rational decision-making, behavioral decision-making based on the Theory of Planned Behavior, and behavioral decision-making with opinion diffusion.
Weblink to model code:
https://github.com/SC3-TUD/PNAS-Simulating-Institutional-Heterogeneity-in-Sustainability-Science
Link to the published material:
https://doi.org/10.1073/pnas.2215674121

Example 2: Opinion Dynamics Model of Social Influence

See the latest version online: https://www.agentblocks.org/rbb/degrootian-opinion-dynamics-model-of-social-influence

Part 1: RBB 'Main Description'

Identifier: Degrootian Opinion Dynamics Model of Social Influence, Thorid Wagenblast

Background and purpose: Often, agent-based modeling is used to model the interaction of people or households. The attributes of these agents can be factual (e.g., income or location) but can also contain perceptions and opinions regarding different matters. In the real world, people are influenced by those around them and change their opinions over time. There are numerous models describing how opinion transfer happens (see [1] for an overview). This building block can be used to model the opinion diffusion among agents based on the DeGroot opinion dynamics model [2]. It is an assimilative social influence model (i.e., opinion differences get reduced) and treats the individuals or agents exchanging opinions as nodes in a network. The links do not change over time and a weight is assigned to each link. In the ABM it can be used to model the exchange of continuous opinions or information of agents that interact within a network or based on proximity.

References:

[1] Flache, A., Mäs, M., Feliciani, T., Chattoe-Brown, E., Deffuant, G., Huet, S., & Lorenz, J. (2017). Models of Social Influence: Towards the Next Frontiers. Journal of Artificial Societies and Social Simulation, 20(4), 2. **https://doi.org/10.18564/jasss.3521**
[2] Degroot, M. H. (1974). Reaching a Consensus. Journal of the American Statistical Association, 69(345), 118–121. **https://doi.org/10.1080/01621459.1974.10480137**

Key concepts and definitions: A change in opinion occurs through the influence of those around us. In an ABM, this means that an agent's perception attributes are influenced by those it interacts with. These can be the closest neighbors or connecting nodes within a network.
The Degrootian opinion dynamics model calculates the opinion formation based on one's own and surrounding opinions, putting a weight on each of them [1]. It can be summarized through

$$F_{ij} = \sum_{j=1}^{k} F_j * p_{ij}$$

where F_{ij} is the new opinion of an agent based on their own opinion and those of others F_j. Each opinion (own and those of others) has a weight p_{ij} assigned to it, denoting the importance given to the own opinion and the opinions of neighbors or other connections in a network. k denotes the total number of connections this agent has in their network.
Since people are not expected to change their opinion entirely from one moment to the next, each agent as a certain **basic own trust** which represents how much agents trust their own opinion. Some individuals are more confident in their own opinion, while others attach more value to social expectations. The **social expectation** is an optional factor that can be included Together, these variables determine the weights p_{ij} assigned to each of the opinions (including one's own).
In order to implement this model of opinion dynamics in an agent-based model, the values for the **basic own trust** (and **social expectation**) variables should be quantified. To do so, there are two options:

1. You set the **basic own trust** (and **social expectation**) yourself. The **basic own trust** should be the same for all agents, the **social expectation**, if included, can vary (e.g., drawing from a normal distribution). Both values should be between 0 and 1.
2. You use empirical data to inform these variables. Then, the **basic own trust** is still the same for all agents, but their **social expectation** varies based on answers given in a survey. Both values should also be between 0 and 1.

Scale: As opinion dynamics is an inter-agent process, it can operate on any spatial scale. You might choose to let the interaction happen in relation to spatial scale (e.g., interaction based on agents being neighbors on a grid), but this does not need to be the case (agents can interact irrespective of spatial distance).
Regarding the temporal scale: The combination of **basic own trust** and **social expectation** determines how fast the opinions change from step to step. Aligning this with the topic of interaction makes sense. For example, if your step represents 1 minute, you might want to choose

a higher **basic own trust/social expectation** combination because it is unlikely that people radically change their opinion from one step to the next, whereas if your step corresponds to multiple years, a lower **basic own trust/social expectation** combination might make more sense. This might differ from opinion to opinion (e.g., political views might change slower than opinions on food preference).

Detailed specification

Input description: An implementation of the Degroot opinion dynamics model should consider – at least – the following input factors:

- Basic own trust: *How much of the opinion the agent always keeps. Determines to some degree how fast the opinions change.*
- Opinion: *What opinion of perception is changing?*

Optional inputs:

- Social expectation: Agent-specific variation of how much of their opinion they keep. Only makes sense to include when wanting different weights assigned between agents. Collected in survey through a question like "Do your family, friends and/or social network expect you to ...?"
- Other opinions: You need at least one opinion/perception that you want to update through social interaction but can opt for a multitude.

Output description: The output of the opinion dynamics implementation would be the adjusted opinion of the agents.

Level of interactions: Multiple agents (same type)

Agent type: Household, Individual

This RBB describes agents that exchange and adjust opinions or perceptions such as households or individuals.

Attributes:

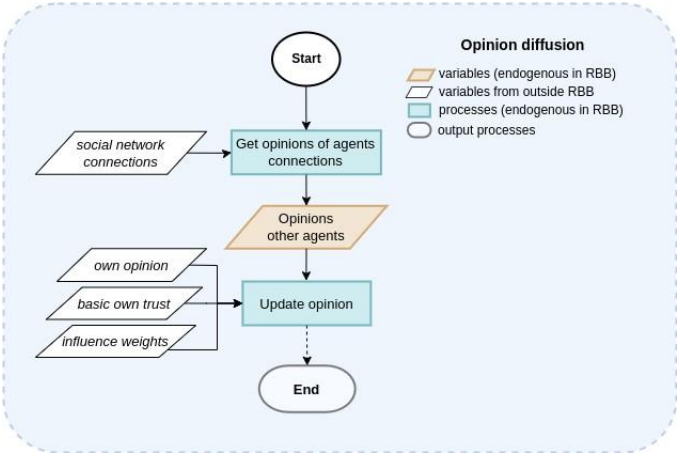
- Basic own trust, social expectation, perception/opinion value, interacting agents

Functions:

- Get other opinions, update opinion
-

Part 2: RBB 'Implementation'

Process flow (Flow chart)



Input and output

Input

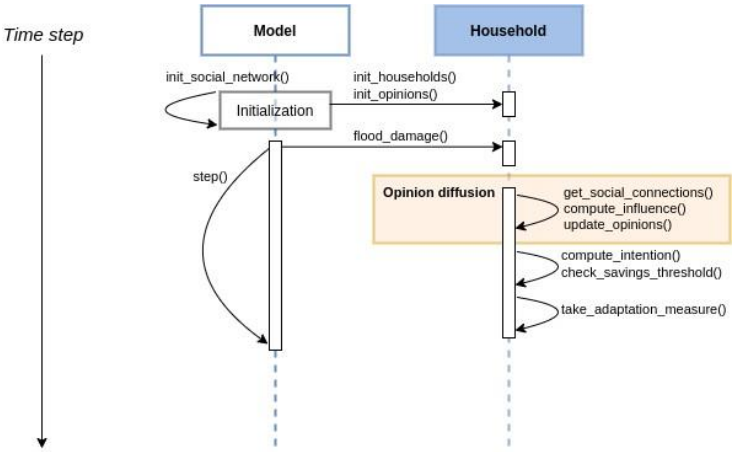
1. Basic own trust
2. Social expectation
3. Opinion/perception

Output

1. Updated opinion/perception

Position in an ABM

Sequence Diagram



Pseudocode

function UPDATE_OPINION():
    compute *influence weights* **from** *basic own trust* and *social expectation*
    get *agent connections*
    compute $worry_{others}$ **from** *agent connections*
    compute $\Delta worry$ **from** *influence weights*, $worry_{others}$
    update $worry_{own}$ **by** $\Delta worry$
end function

Reusable code block

Code: PYTHON


```
# -*- coding: utf-8 -*-
"""
@author: t.wagenblast@tudelft.nl
"""
import random
import numpy as np
from mesa import Model
from mesa import Agent
from mesa.time import RandomActivation

class OD_Model(Model):
    """Agent-based model employing opinion diffusion in a social network."""

    def __init__(self, n_households, n_connections):
        """Model initialization

        Args:
            n_households      : Number of households
            n_connections     : Number of social connections per household
        """

        # Keep track of agent IDs and number of households
        self.current_id = 0
        self.n_households = n_households
        # Add schedule; the type of schedule can be chosen by the user
        self.schedule = RandomActivation(self)

        # Initialize households
        for i in range(n_households):
            hh = Household(self,
                           worry=random.random(),
                           social_expectation=random.random(),
```

```
                    basic_own_trust=0.5)
            self.schedule.add(hh)

        # Create social network
        for hh in self.schedule.agents:
            others = [household for household in self.schedule.agents
                      if household != hh]
            hh.social_connections = np.random.choice(others, n_connections,
                                                     replace=False)

        # ... other model initialization functions ... #

    def step(self):
        """Model step."""
        self.schedule.step()


class Household(Agent):
    """Household agent."""

    def __init__(self, model, worry, social_expectation, basic_own_trust=0.5):
        """Initialization of a household agent.

        Args:
            model (Model)            : Model containing the household agent
            worry (float)            : Worry value, may change over time
            social_expectation (float)  : Weight assigned to opinion of others
            basic_own_trust (float)     : Weight assigned to own opinion
                                  (default = 0.5).
        """
        super().__init__(model.next_id(), model)

        # Perception value that changes through the opinion diffusion
        self.worry = worry

        # Calculate trust into other (i.e. weight assigned to opinion of others)
        self.trust_in_others = (social_expectation * (1 - basic_own_trust))
        self.trust_in_oneself = 1 - self.trust_in_others

        # Social connections are created after all households are initialized
        self.social_connections = []

    def get_other_opinions(self, attribute):
        """Get opinions of agents' social connections.

        Args:
            attribute (string)      : Attribute to collect opinions for
        """
        return [getattr(hh, attribute) for hh in self.social_connections]

    def update_opinion(self, attribute, other_opinions):
```

```
        """Get new opinion from influence of social network.

        Args:
          attribute (string)     : Attribute influenced by social network
          other_opinions (list)  : Opinions of others
        """

        # Update household's current opinion
        old_value = getattr(self, attribute)
        social_influence = sum((np.array(other_opinions) * self.trust_in_others)
                               / len(other_opinions))
        new_value = old_value * self.trust_in_oneself + social_influence

        # Update own opinion
        setattr(self, attribute, new_value)

    def step(self):
        """Household step."""

        # Update household worry value
        other_opinions = self.get_other_opinions("worry")
        self.update_opinion("worry", other_opinions)

        # ... other Household step functions ...

n_households = 1000
n_connections = 7
model = OD_Model(n_households, n_connections)

steps = 50
for i in range(steps):
    print("STEP", i+1)
    model.step()
```

Agent-based model: Social Influence in Private Adaptation ABM

Model description: This model explores the uptake of flood adaptation measures of households under the influence of social interaction.
The idea is to see how different network structures (Barabasi-Albert scale-free network, Watts-Strogatz small-world network, and Erdös-Renyi random network) and the exchange of different perceptions regarding flood adaptation impact the adaptation uptake and damage figures over a larger area.

Model code: https://github.com/thoridw/SIPAABM

Published material: http://dx.doi.org/10.2139/ssrn.4763672

Example 3: Agent Expectation Formation

See the latest version online: https://www.agentblocks.org/rbb/agent-expectation-formation

Part 1: RBB 'Main Description'

Identifier: "Agent Expectation Formation", Nicholas R. Magliocca

Background and purpose:

The purpose of the Agent Expectation Formation RBB is to represent a set of competing, simplified mental models individual agents can use to form expectations of future states of interest. This form of expectation formation is consistent with bounded rationality and relies on inductive reseasoning (Arthur, 1994). This mode of expectation formation assumes that agents can observe and learn from past states but may not have complete information of all past states. The conceptual basis for these expectation formation models was described in the El Farol Bar problem (Arthur, 1994) and was designed to predict the number of attendees at a future time based on past attendance information. This was subsequently adapted to form expectations of future stock prices in applications of the Santa Fe Institute's artifical stock market (LeBaron et al., 1999). The generalized formalization of this RBB futher modifies these expectation models for broader applicability, such as housing prices (Magliocca et al., 2011, 2014b) and crop prices and yields (Magliocca et al., 2013, 2014a).

[Optional] Overview figure

Key concepts and definitions

Expectations of future states are often used to inform decisions in the current time period. There are many ways to represent expectation formation ranging from implicit (i.e., assuming the future will replicate the past; e.g., habitual behavior or reinforcement learning) to explicit (e.g., forecasting; Macal and North, 2005). Evidence from experimental psychology suggests that expectation formation more closely resembles bounded rationality (Kahneman, 2003) and relies on heuristics and/or simple mental models that reside somewhere between the implicit and explicit extremes (Tversky and Kahneman, 1974). The conceptual structure proposed by Arthur (1994) uses a set of diverse mental models, or 'ecology of beliefs', in which no single mental model can remain the most accurate indefinitely because of stochasticity, emergent dynamics, and/or strategic behaviors inherent in complex adaptive systems.

Scale: In principle, the spatial and temporal scales of applicability of this RBB are not constrained, because an agent may have access to historical information that was not directly observed. However, if the implementation considers only direct observation/experience with whatever is being predicted, there are implicit assumptions about the cognitive capacities of decision-makers that would likely limit the temporal extent of implementation to those that decision-makers could be plausibly recalled.

Detailed specification

Input description: Inputs to this RBB include: 1) directly sensed or otherwise observed past states of the metric being predicted; 2) time horizons for considering past information; and 3) stored and evaluated performances of individual prediction models to select the best performing prediction model at each time step.

Output description: The output of the RBB is an expectation of the future state of a metric of interest in time t+1.

Level of interactions: Agent(s) and the Environment

Agent definition: Individual or household

An agent is given a set of twenty prediction models. Each prediction model may use one of six different prediction methods, and there may be more than one model applying the same prediction method in the agent's set of twenty models. Some of these prediction methods map past and present metrics into the next period using various extrapolation methods. Other methods translate changes from only last period's metrics to next period's metrics.

Attributes:

1. Time horizon (i.e., memory)
2. unique set of prediction models
3. model performance update rate

Functions:

1. Observe past information
2. evaluate prediction model performance

select most successful model

Part 2: RBB 'Implementation'

Description: In this implementation, housing developers and farmers make pricing decisions informed by expectations of future housing and land prices, respectively. Adapted from price expectation models used in agent-based financial literature (e.g. Arthur, 1994, 2006; Axtell, 2005), agents try to predict next period's price based on current and past price information. An agent is given a set of twenty prediction models. Each prediction model may use one of six different prediction methods, and there may be more than one model applying the same prediction method in the agent's set of twenty models. Some of these prediction methods map past and present prices (P) into the next period using various extrapolation methods.

1. Mean model: predicts that *P(t+1)* will be the mean price of the last *x* periods of the agent's memory.

$P(t+1) = P(t-x:t)/x;$

2. Cycle model: predicts that *P(t+1)* will be the same as *x* periods ago (cycle detector).

$P(t+1) = P(t-x)$;

3. Projection model: predicts that *P(t+1)* will be the least-squares, non-linear trend over the last *x* periods.

$P(t+1)= aP(t_s)^2 + bPt_s + c;$ where t_s is the time span of *t-x* to *t*, and *a*, *b*, and *c* are coefficients of fit.

Other methods translate changes from only last period's price to next period's price.

4. Mirror model: predicts that *P(t+1)* will be a given fraction ξ of the difference in this period's price, *P(t)*, from last period's price, *P(t-1)*, from the mirror image around half of *P(t)*.

$$P(t+1) = 0.5P(t) + [0.5P(t) - (1 - \xi)(P(t) - P(t-1))];$$

5. Re-scale model: predicts that *P(t+1)* will be a given factor ζ of this period's price bounded by [0,2].

$$P(t+1) = \zeta P(t);$$

6. Regional model: predicts that *P(t+1)* is influenced by regional price information coming from neighboring agents.

For farmers, land prices are a function land scarcity as measured by the number of remaining farmers, N_f, in the region at time t.

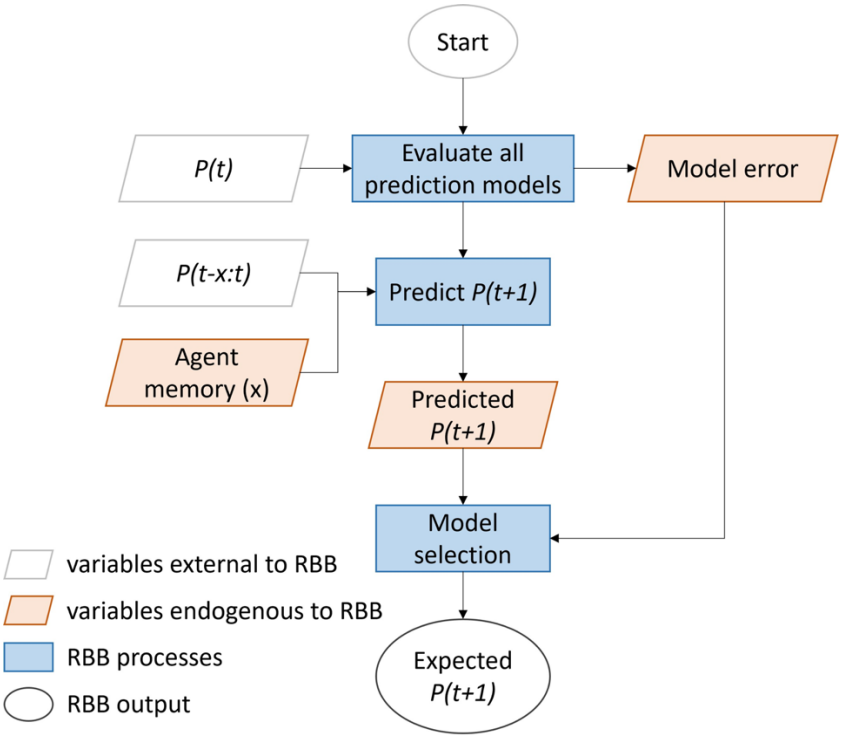
$$P(t+1) = P(t)(1 - 1/N_f);$$

For developers, the expected price of house types with size, *h*, on lot size, *l*, in a given neighborhood, N_b, is the mean of prices of house and lots of the same sizesP in adjacent neighborhoods, N_{nei}. N_{nei} are neighbors in the cardinal directions.

$$P(N_{b,h,l},t+1) = \text{mean}\{P(N_{nei,h,l},t)\};$$

All models in the agent's set of prediction models are used to predict the price in the next time period (*P(t+1)*). In time *t+1*, the actual price is known and an error squared is calculated for each model by squaring the difference between the predicted price and the actual price. The prediction model with the least error is used to make the agent's pricing decisions in the current period. This same process of prediction and evaluation is used every period so that the most successful prediction model is used every time.

Process flow (Flow chart)



Input and output

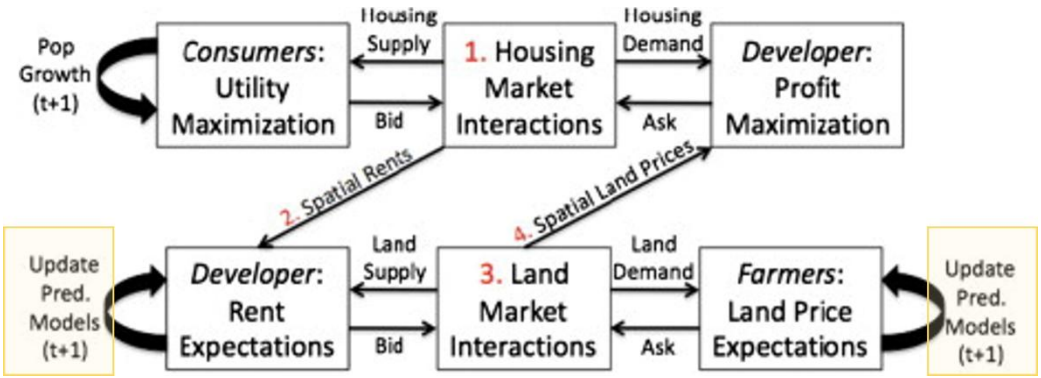
Input

- Current price
- Past prices

Output:

- Expected price

Position in an ABM



Pseudocode

function assign price prediction models( ):

number of price prediction models ← 20

assign type of each price prediction model from random number 1:6

assign agent memory from random number 1:10

end

```
function predict price( ):
        compute current model errors from current price compared to expected price
        model error ← current model error + past model error
        compute expected prices from price prediction models and agent memory
        select prediction of model with lowest model error
        expected price ← prediction of best performing price prediction model
end
```

Reusable code block

```
%%%%%%%%%%%%% Agent Expectation Formation %%%%%%%%%%%%%%%
Nagents=100; %example agent population
TSTART=10; %end of spin-up period
TMAX=30; %total number of time steps
NUMMODEL=20; %Number of prediction models per agent
PRODCLASS=6; %number of yield prediction model types
PRICECLASS=5; %number of price prediction model types
MAXMEANMODEL=10; %maximum time steps into the past to calculate mean
MAXCYCLEMODEL=10; %maximum time steps into the past to calculate cycle
MAXPROJECT=10; %maximum time steps into the past to calculate trend
DELTA=0.6;

priceproj=zeros(Nagents,TMAX); %{[iown,NUMMMODEL] Nuse}
priceerror=zeros(Nagents,TMAX); %cumulative model error
ipricebestSAVE=zeros(Nagents,TMAX); %index of best performing model
ExptPrice=zeros(Nagents,TMAX); %prediction of best performing model
pricemodelSAVE=zeros(Nagents,TMAX); %best performing model

%%% Price EXPECTATIONS
pricemodel = ceil(PRICECLASS*rand(Nagents,NUMMODEL)); %full heterogeneity
priceclass1=find(pricemodel == 1);
priceclass2=find(pricemodel == 2);
priceclass3=find(pricemodel == 3);
priceclass4=find(pricemodel == 4);
priceclass5=find(pricemodel == 5);
priceclass6=find(pricemodel == 6);

% Initialization
bb = zeros(Nagents,NUMMODEL,'single');

%Price models
for i = 1:PRICECLASS
    if i == 1 % mirror model
        bb(priceclass1) = rand(1); % fraction that pred is away from 1/2 from mirror image
```

```
    elseif i == 2 % mean model
        bb(priceclass2) = ceil(MAXMEANMODEL*rand(length(priceclass2),1));
    elseif i == 3 %cycle model
        bb(priceclass3) = ceil(MAXCYCLEMODELrand(length(priceclass3),1));
    elseif i == 4 % projection model
        bb(priceclass4) = ceil(2+((MAXPROJECT-1)-2)rand(length(priceclass4),1));
    elseif i == 5 % rescale model
        bb(priceclass5) = 2rand(length(priceclass5),1);
    elseif i == 6 %regional trends
        bb(prodclass6) = ceil(MAXMEANMODEL*rand(length(priceclass6),1));
    end
end

% Dynamics
for t=TSTART+1:TMAX
    for n=1:Nagents
        ipriceclass1=find(pricemodel(n,:)==1);
        ipriceclass2=find(pricemodel(n,:)==2);
        ipriceclass3=find(pricemodel(n,:)==3);
        ipriceclass4=find(pricemodel(n,:)==4);
        ipriceclass5=find(pricemodel(n,:)==5);
        ipriceclass6=find(pricemodel(n,:)==6);
        for i = 1:PRICECLASS
            if i == 1 % mirror models
                priceproj(n,ipriceclass1) = priceproj(n,t)+(1-bb(n,ipriceclass1)).*
                    (0.5*priceproj(n,t)-(priceproj(n,t)-priceproj(n,t-1)));
            elseif i == 2 % mean model
                for jl = 1:length(ipriceclass2)
                    priceproj(n,ipriceclass2(jl)) = mean(priceproj(n,t:-1:(t-bb(n,ipriceclass2(jl)))));
                end
            elseif i == 3   %cycle model
                priceproj(n,ipriceclass3) = priceproj(n,t-round(max(1,bb(n,ipriceclass3))));
            elseif i == 4 % projection model
                for jl = 1:length(ipriceclass4)    %Nonlinear Forecast
                    indata=priceproj(n,t-(1+bb(n,ipriceclass4(jl))):t);
                    pcoef=polyfit(1:length(indata),indata,1);
                    pline=pcoef(1).*(1:length(indata)+1)+pcoef(2);
                    priceproj(n,ipriceclass4(jl))=pline(length(pline));
                end
            elseif i == 5 % rescale model
                priceproj(n,ipriceclass5) = bb(n,ipriceclass5)*priceproj(n,t);
            elseif i == 6  % local(0) or regional(1) trends
                ipricelocal=(bb(n,ipriceclass6)==0);
                ipricereg=(bb(n,ipriceclass6)==1);
                if isempty(Nagents)==1
                    break
                end
                priceproj(n,ipriceclass6(ipricelocal)) = priceproj(n,t).*(1+1/length(Nagents));
```

```
            end
          end
        end
        priceerror(n,:) = (1-DELTA)*priceerror(n,:)+DELTA*abs(priceproj(n,t)-priceproj(n,:));
        [pricebest,ipricebest] = min(priceerror(n,:),[],2);
        ipricebestSAVE(n,t) = ipricebest;
        pricemodelSAVE(n,t) = pricemodel(n,ipricebest);
         ExptPrice(n,t+1) = priceproj(n,ipricebest);
    end
```

Agent-based model

Model description: The overall purpose of this model is to explore how feedbacks between housing and land markets influence the conversion of undeveloped land (e.g., agriculture) to residential housing. The agent-based model (ABM) presented here is a version of the CHALMS model (Magliocca et al., 2011, 2012) that has been adapted to a coastal landscape subject to uncertain impacts from coastal storms (C-CHALMS). The goal of the model is not to simulate the development patterns and market dynamics of any particular location. Rather, the aim is to isolate psychological and perceptual factors that influence location and adaptation decisions and their effects on key interactions between housing and land markets (particularly the timing and proximity to the coast of land conversion. The model investigates how agent-level decisions and interactions through markets link to market- and landscape-level outcomes, such as housing and land prices and extent and configuration of residential development, respectively. Further, the goal is to understand how residential housing consumers make trade-offs between amenities and risks of damages from storms given location near the coast, and how those trade-offs do or do not influence adaptive decisions in response to storms, such as purchasing insurance and/or relocating to less risky areas.

Model code: https://github.com/nickmags13/CHALMS_coastal_simple

Published material: https://doi.org/10.1016/j.compenvurbsys.2018.03.009

Appendix D: Criteria for RBB review and instructions for reviewers

RBB 'Description'	**RBB 'Implementation'**

Please provide any specific feedback on the Reusable Building Block. Consider using the Reviewer Guide below as an optional guide and provide your professional reflections. Your general recommendations for the improvement of the description of the RBB are very welcome. Thank you very much for your time and remember:
 	
• **Clarity of Description:** Please reflect briefly whether the author has clearly explained the motivation behind the RBB, its theoretical underpinning and its purpose.
• **Clarity of Assumptions:** Do the authors sufficiently explain the assumptions that underlie this RBB?
• **Potential Reusability:** From your point of view, can this RBB be used in other models and/or in other application domains? Feel free to note which ones.
• **Novelty:** Does the RBB describe an original (not yet existing in the database) component of an ABM?
	Please provide any specific feedback on the Reusable Building Block. Consider using the Reviewer Guide below as an optional guide and provide your professional reflections. Your general recommendations for the improvement of the implementation of the RBB are very welcome. Thank you very much for your time and remember:
	
	• **Modularity:** Briefly explain whether the described RBB is sufficiently modular. Would it be more usable if its scope were narrower or broader, or is the current level of modularity appropriate?
	• **Clarity and Logic:** Can this RBB be used in other (agent-based) models based on the provided implementation (in terms of algorithm, pseudo-code, and/or access to the simulation program itself). Specifically:
	  ○ Are the inputs/outputs clearly specified?
	  ○ Is the provided algorithm/pseudo-code organized and logical?
	• **Reusability in other ABMs:** Please assess whether the provided implementation code is usable as a part in other (agent-based) models.
	  ○ Is the provided code implementation organized and logical?
	  ○ To what extent does the implementation code help understand how the RBB can be used in an ABM?
	  ○ Check if the diagram describing the place of RBB in the full ABM is easy to follow.
	• **Weblinks:** Is the link to the full ABM using this RBB provided? Does it work?