Agent Based Models Part 2

review

- what are agent based model?

review

- what are agent based model?
  - kind of a vague term
  - *Agent-based models* are computational simulation models that involve discrete agents.

review

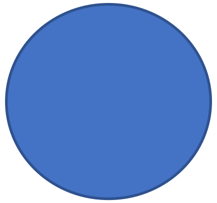
- what are agent based model?
  - kind of a vague term
  - *Agent-based models* are computational simulation models that involve discrete agents.

ABMs are usually implemented as simulation models in a computer, where each agent's behavioral rules are described in an algorithmic fashion rather than a purely mathematical way

review

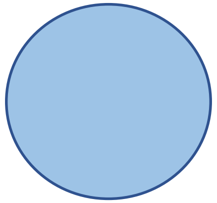
- what are agent based model?
  - kind of a vague term
  - *Agent-based models* are computational simulation models that involve discrete agents.

ABMs are usually implemented as simulation models in a computer, where each agent's behavioral rules are described in an algorithmic fashion rather than a purely mathematical way



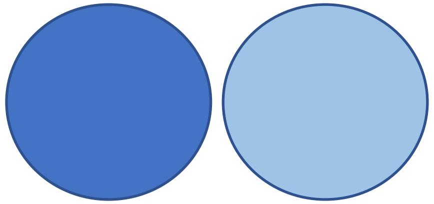
greedy = 0
wealth = 1

toy problem where each agent is defined by its wealth level and its generosity



greedy = 1
wealth = 0

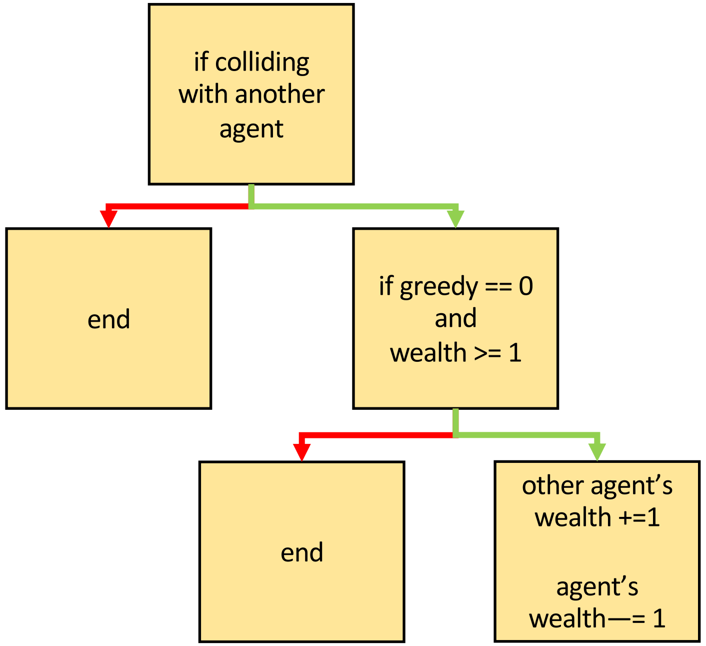
- toy problem where each agent is defined by its wealth level and its generosity
- the agents move around the field and when the agents interact, colloid into each other, agents that are not greedy will give one wealth to another agent

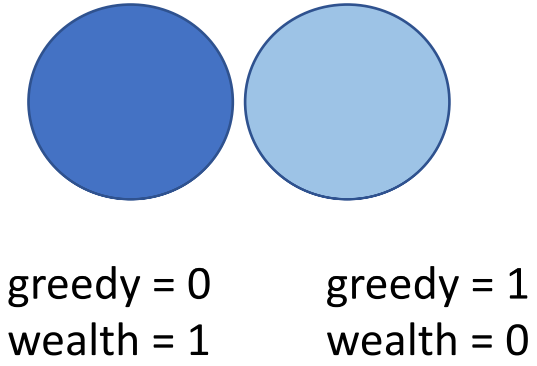


greedy = 0
wealth = 1

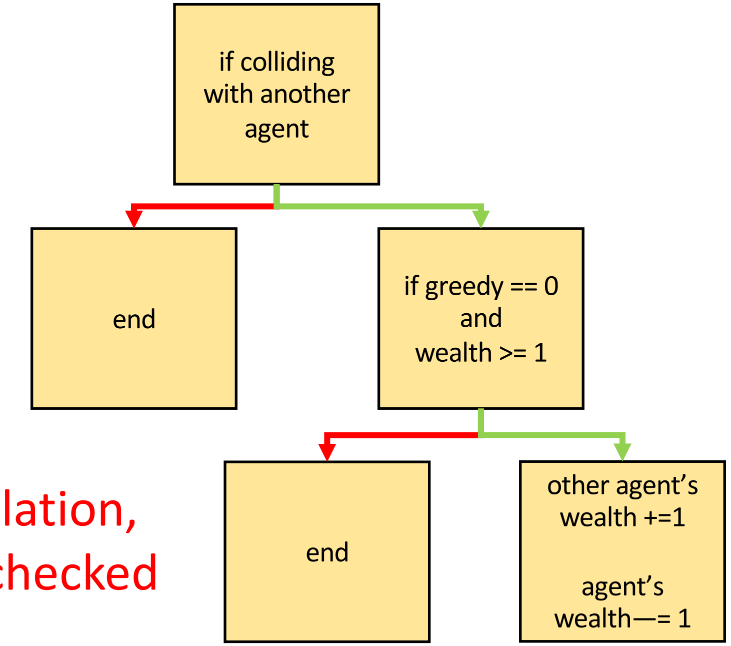
greedy = 1
wealth = 0

- toy problem where each agent is defined by its wealth level and its generosity
- the agents move around the field and when the agents interact, colloid into each other, agents that are not greedy will give one wealth to another agent

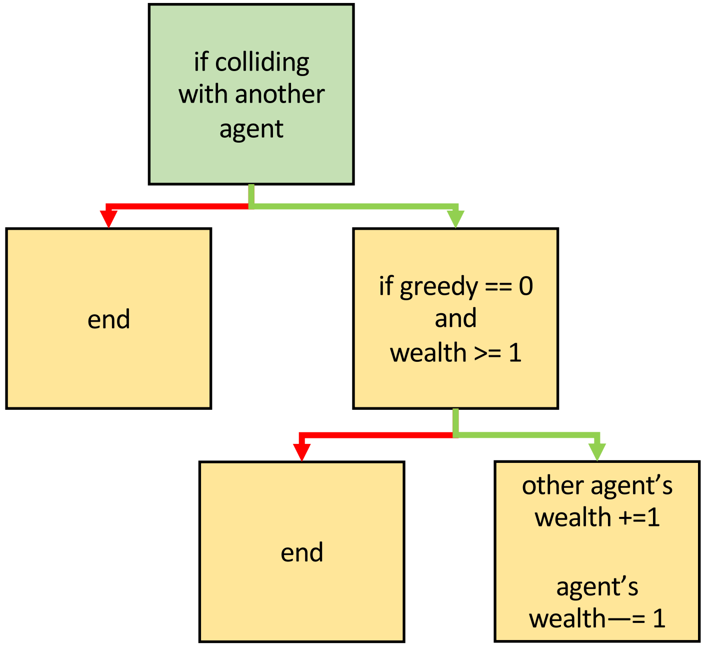


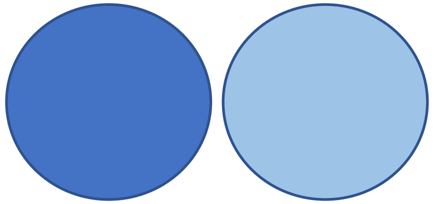


every time unit in the simulation, the following algorithm is checked for each agent



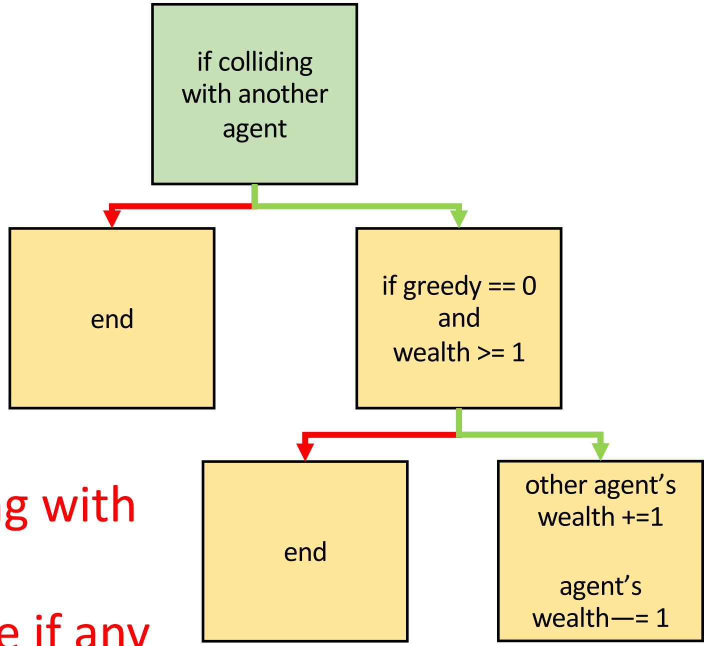
- toy problem where each agent is defined by its wealth level and its generosity
- when the agents interact, colloid into each other, agents that are not greedy will give one wealth to another agent





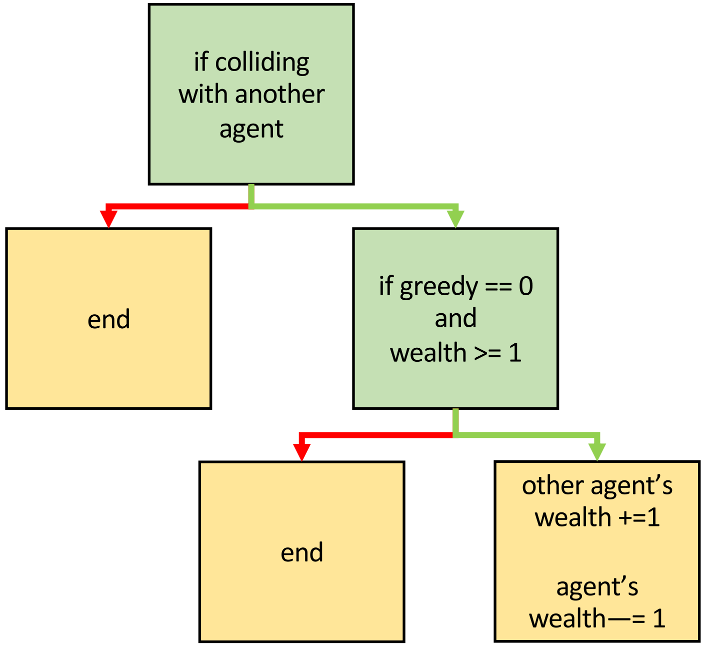
greedy = 0
wealth = 1

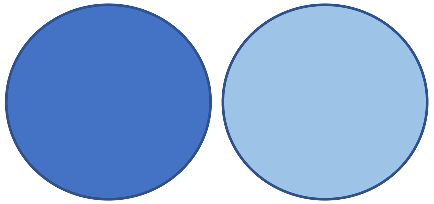
greedy = 1
wealth = 0



first we check if they are colliding with another agent,
if they are we need to determine if any wealth is traded
if there is no collision, then the algorithm is done
in this case the dark blue and light blue agent are colliding with each other.

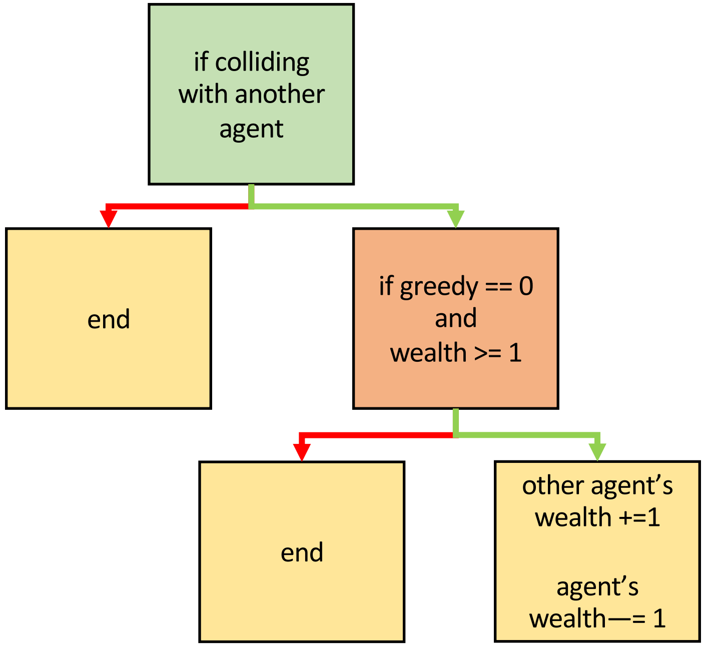
- toy problem where each agent is defined by its wealth level and its generosity
- the agents move around the field and when the agents interact, colloid into each other, agents that are not greedy will give one wealth to another agent





greedy = 0
wealth = 1

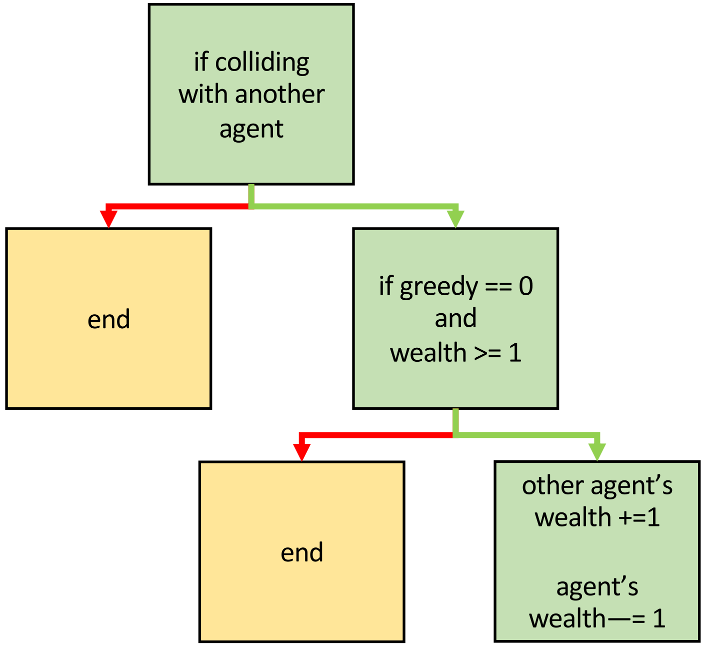
greedy = 1
wealth = 0

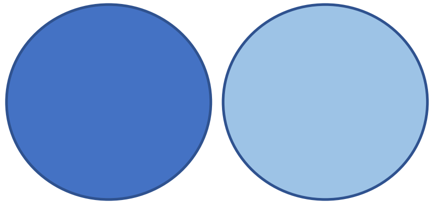


Next we should next determine whether the agent of interest is greedy.

- if they are greedy, they will not even consider giving money
- if they are not greedy, then they will

- toy problem where each agent is defined by its wealth level and its generosity
- the agents move around the field and when the agents interact, colloid into each other, agents that are not greedy will give one wealth to another agent

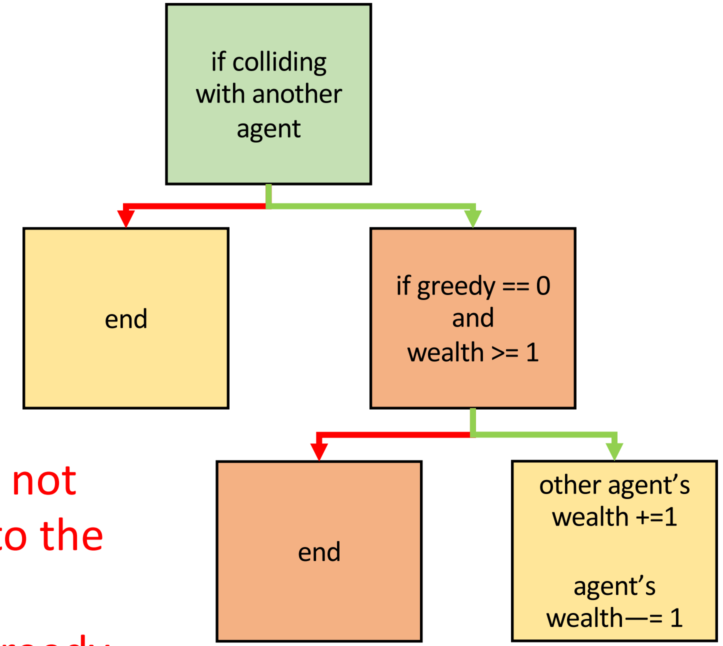


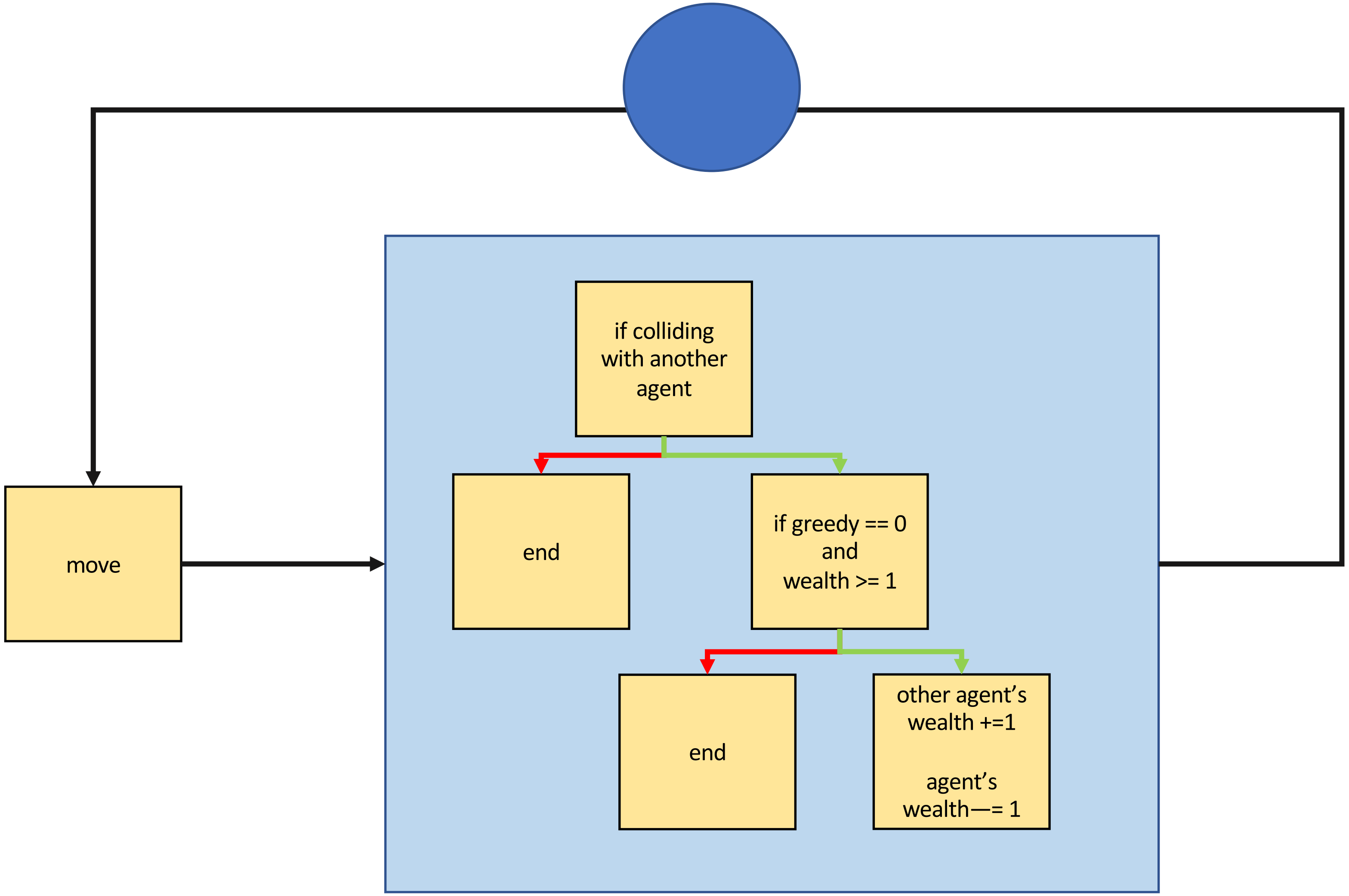


greedy = 0
wealth = 0

greedy = 1
wealth = 1

- In this case the dark blue agent is not greedy so it will give one wealth to the other agent, the light blue one
- However, the light blue agent is greedy, and it will not do the same

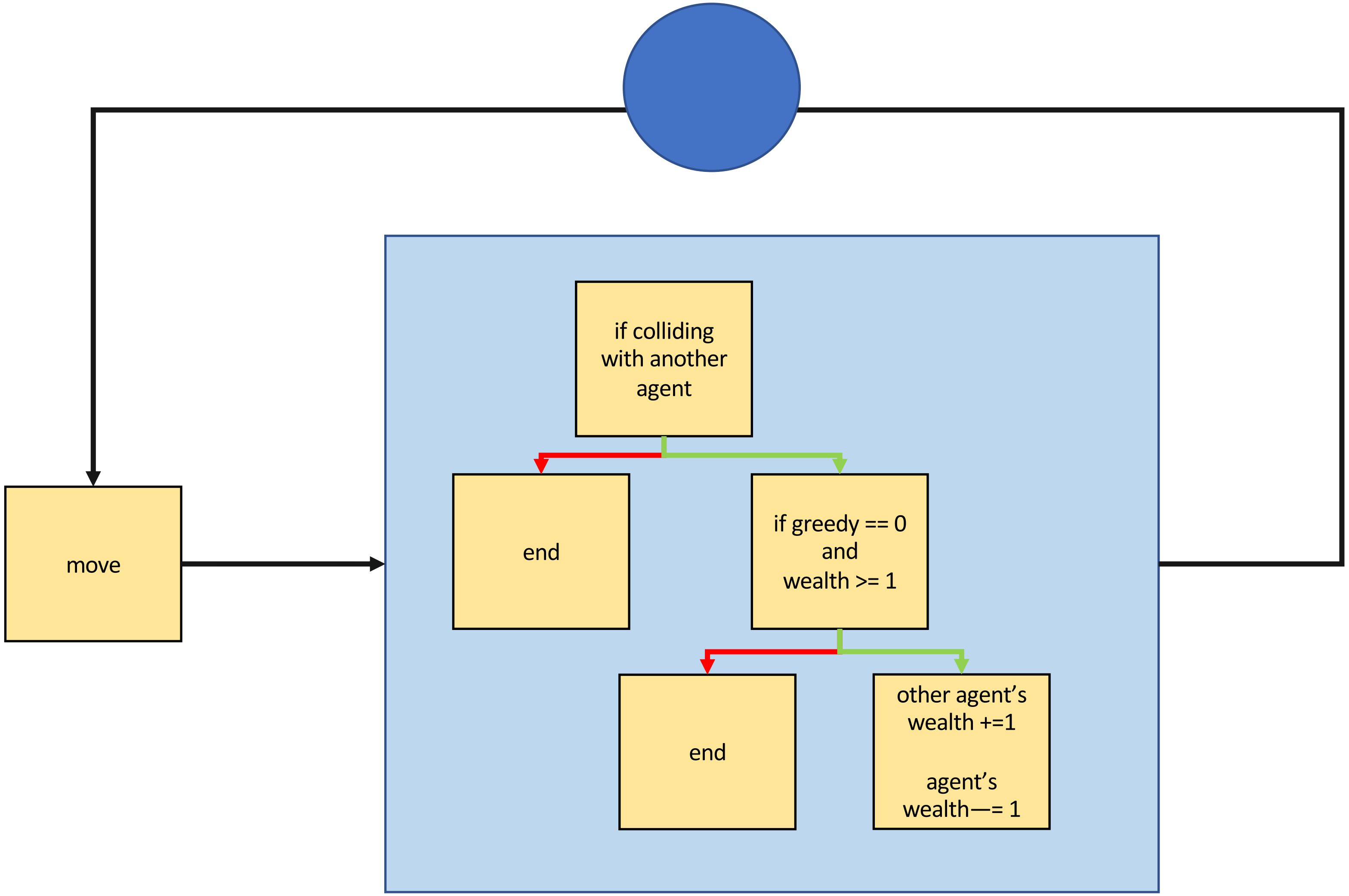




this algorithm is repeated for each time unit for each individual agent

- first the agent will move by one unit on the field
- second it will run the algorithm to determine if it can donate wealth to another agent

this is repeated over and over until the program is terminated by the user or a user defined stopping point is reached



this algorithm is repeated for each time unit for each individual agent

- first the agent will move by one unit on the field
- second it will run the algorithm to determine if it can donate wealth to another agent

this is repeated over and over until the program is terminated by the user or a user defined stopping point is reached

this ability to capture complex individual behavior is enticing to us a researchers

however, this comes at a cost

The complexity of agent behavior means that mathematical analysis does not work.

Instead, standard statistical analysis are used instead for output analysis

-25 degrees

starting charge	rep 1	rep 2	rep 3	rep 4	rep 5	rep 6	rep 7	rep 8	rep 9	rep 10
48.5	0	0	0	0	0	0	0	0	0	0
47	0	0	0	0	0	0	0	0	0	0
45.5	0	0	0	0	0	0	0	0	0	0
44	0	0	0	0	0	0	0	0	0	0
42.5	0	0	0	0	0.00672143	0	0	0	0	0.01941748
41	0.05377147	0.00522778	0.04929052	0.05675878	0.01344287	0	0.0029873	0.00448096	0.04929052	0.00896191
39.5	0.04705004	0.09260642	0.09634055	0.09932786	0.06945482	0.08289768	0.01643017	0.05302465	0.04480956	0.07617625
38	0.07617625	0.10231516	0.09335325	0.16131441	0.10828977	0.12546677	0.14712472	0.1120239	0.11127707	0.16355489
36.5	0.15758028	0.16056759	0.13965646	0.1351755	0.16131441	0.10978342	0.13592233	0.09559373	0.21433906	0.13293503
35	0.19417476	0.16056759	0.21359223	0.2300224	0.16504854	0.16206124	0.17849141	0.19342793	0.17401046	0.23525019

Average	standard. Dev.	P(x<=.1)	Starting Charge
0	0	1	48.5
0	0	1	47
0	0	1	45.5
0	0	1	44
0.00261389	0.006270671	1	42.5
0.02442121	0.024325213	0.99905506	41
0.0678118	0.026958936	0.88375544	39.5
0.12008962	0.029039487	0.24453036	38
0.14428678	0.032495847	0.08646579	36.5
0.19066468	0.027665174	0.00052419	35

The complexity of agent behavior means that mathematical analysis does not work.

Instead, standard statistical analysis are used instead for output analysis.
you run the simulation many times and obtain a distribution you can then use for statistical testing.

- T paired tests
- standard comparison tests
- Ranking and Selection techniques

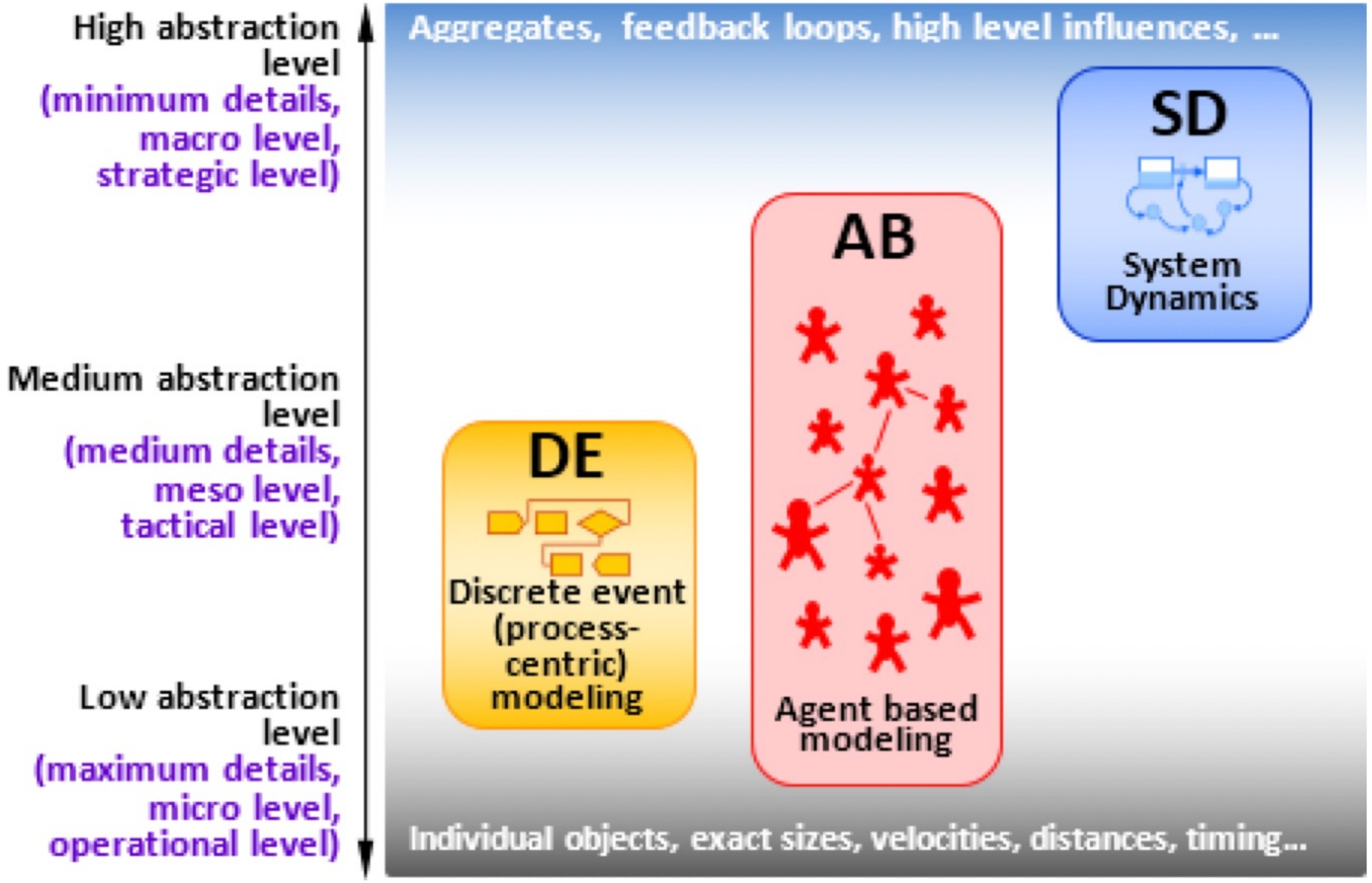
-25 degrees starting charge	rep 1	rep 2	rep 3	rep 4	rep 5	rep 6	rep 7	rep 8	rep 9	rep 10
48.5	0	0	0	0	0	0	0	0	0	0
47	0	0	0	0	0	0	0	0	0	0
45.5	0	0	0	0	0	0	0	0	0	0
44	0	0	0	0	0	0	0	0	0	0
42.5	0	0	0	0	0.00672143	0	0	0	0	0.01941748
41	0.05377147	0.00522778	0.04929052	0.05675878	0.01344287	0	0.0029873	0.00448096	0.04929052	0.00896191
39.5	0.04705004	0.09260642	0.09634055	0.09932786	0.06945482	0.08289768	0.01643017	0.05302465	0.04480956	0.07617625
38	0.07617625	0.10231516	0.09335325	0.16131441	0.10828977	0.12546677	0.14712472	0.1120239	0.11127707	0.16355489
36.5	0.15758028	0.16056759	0.13965646	0.1351755	0.16131441	0.10978342	0.13592233	0.09559373	0.21433906	0.13293503
35	0.19417476	0.16056759	0.21359223	0.2300224	0.16504854	0.16206124	0.17849141	0.19342793	0.17401046	0.23525019

Average	standard. Dev.	P(x<=.1)	Starting Charge
0	0	1	48.5
0	0	1	47
0	0	1	45.5
0	0	1	44
0.00261389	0.006270671	1	42.5
0.02442121	0.024325213	0.99905506	41
0.0678118	0.026958936	0.88375544	39.5
0.12008962	0.029039487	0.24453036	38
0.14428678	0.032495847	0.08646579	36.5
0.19066468	0.027665174	0.00052419	35

Agent based models are a great way to play around with ideas in a risk free environment

Building an agent-based model is a balancing act of:

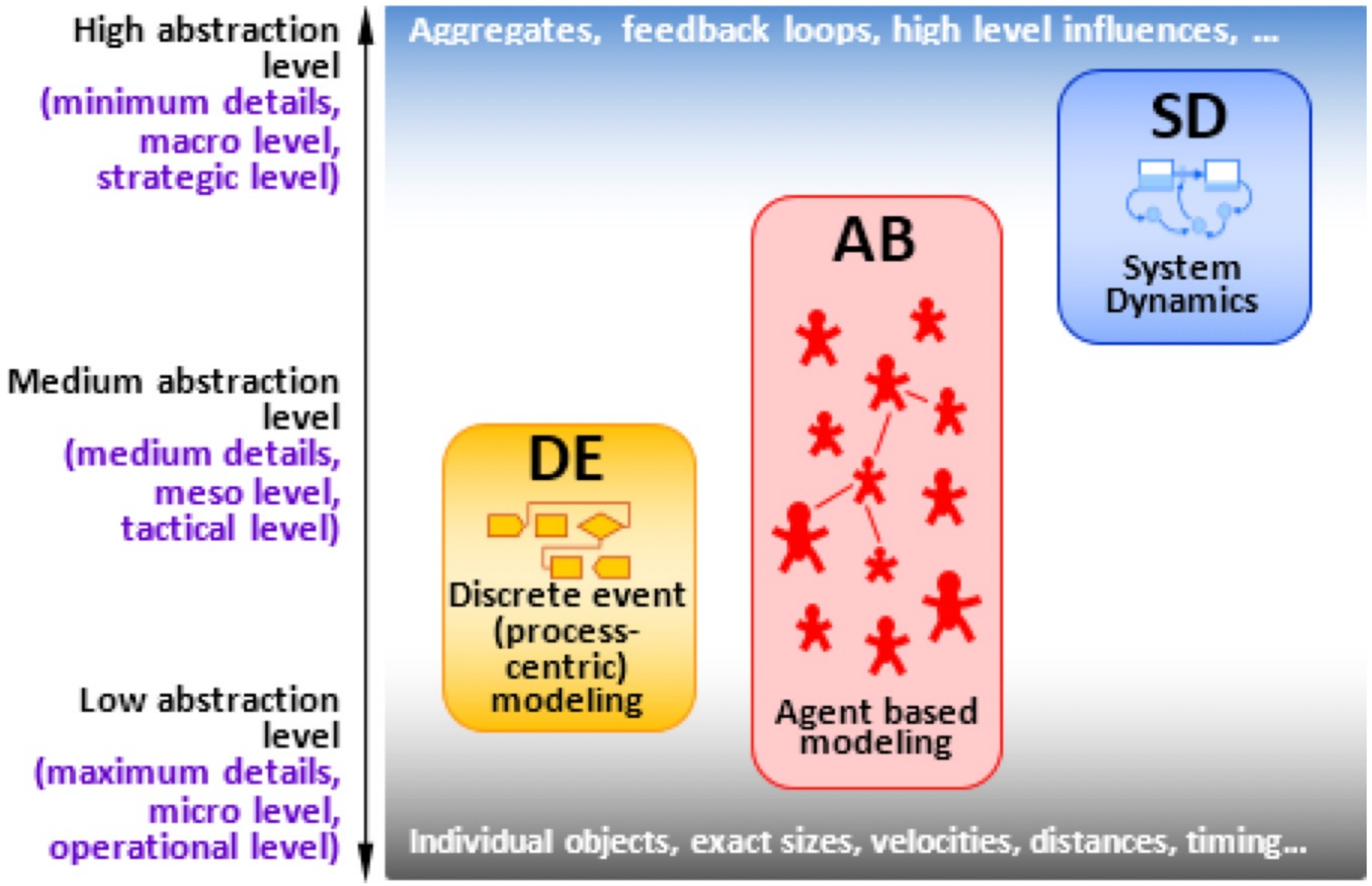
- simplicity
- validity
- robustness



Agent based models are a great way to play around with ideas in a risk free environment

Building an agent-based model is a balancing act of:

- simplicity
- validity
- robustness



It is very tempting to keep adding complexity to the model to try to approach a more realistic setting, however, with more complexity you increase the difficulty of analyzing or justifying your model

group work!

- https://www.netlogoweb.org/launch#https://www.netlogoweb.org/assets/modelslib/Sample%20Models/Biology/Wolf%20Sheep%20Predation.nlogo
- In the “Models Library” open the model “Wolf Sheep Predation” (see the category “Biology”) and turn on the “grass?” switch.
- Use the model to determine why the wolf population goes to 0 fairly quickly when the grass regrowth time is set to 10
- Use the model to determine how much is much for the wolf population

what are python libraries exactly

- code you don’t have to write!

what are python libraries exactly

- code you don’t have to write!

- How do we use python libraries?

what are python libraries exactly

- code you don’t have to write!

- How do we use python libraries?
- At the top of the program we will type the following

  import <Library Name>
- although this assumes that the library is loaded already

what are python libraries exactly

- code you don’t have to write!

- How do we use python libraries?
- At the top of the program we will type the following

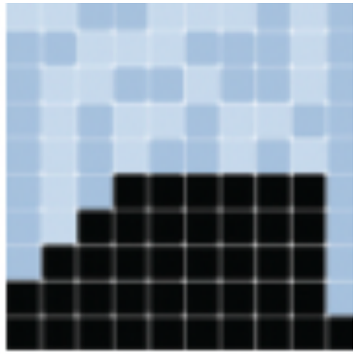
  import <Library Name>
- although this assumes that the library is loaded already

- If the library is not loaded we will have to install it using the following code

  pip install <Library Name>

How do we start to program an agent based model?

- Agent based models are coding intense
  - requires organized and modular coding to help prevent bugs



For that reason, we will be learning the python library MESA
this Library allows us to save time on building core components such as spatial grids and agent schedulers from scratch

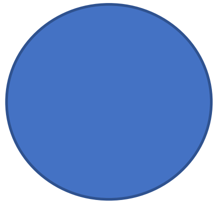
Mesa also contains modules that help with
- Modeling
- Analysis
- Data visualization

Lets first look at how we can use the modeling modules from MESA to create agents

It is convenient to create object for agents using classes to define each agent's individual behaviors and parameters

very basic agent object

```
import random
import mesa
class agent(mesa.Agent):
        def __init__(self,model,name,wealth,greed):
                super().__init__(name,model)
                self.name = name
                self.wealth = wealth
                self.greed = greed
```



Lets first look at how we can use the modeling modules from MESA to create agents

It is convenient to create object for agents using classes to define each agent's individual behaviors and parameters

very basic agent object

```
import random
import mesa

class agent(mesa.Agent):
        def __init__(self,model,name,wealth,greed):
                super().__init__(name,model)
                self.name = name
                self.wealth = wealth
                self.greed = greed
```

defines the class name and the parent class from the mesa library we are reading in.
Reading in mesa.Agent allows us to use the functions from that file

Lets first look at how we can use the modeling modules from MESA to create agents

It is convenient to create object for agents using classes to define each agent's individual behaviors and parameters

very basic agent object

```
import random
import mesa

class agent(mesa.Agent):
    def __init__(self,model,name,wealth,greed):
        super().__init__(name,model)
        self.name = name
        self.wealth = wealth
        self.greed = greed
```

Initialization function, a special function that is read by default when ever we create an agent object

- self represents the instance of the object and is used to access variables and functions belonging to the class
- model is the model object that holds all the agent objects
- name refers to a unique ID for each agent
- wealth and greed are user defined values

you must pass in the model, ID, wealth level and greedy value, you do not need to pass in self, that is created when you call the class

Lets first look at how we can use the modeling modules from MESA to create agents

It is convenient to create object for agents using classes to define each agent's individual behaviors and parameters

very basic agent object

```
import random
import mesa

class agent(mesa.Agent):
        def __init__(self,model,name,wealth,greed):
            super().__init__(name,model)
            self.name = name
            self.wealth = wealth
            self.greed = greed
```

calls the initialization method of the parent class

```
def __init__(self, unique_id: int, model: Model) -> None:
    """
    Create a new agent.

    Args:
        unique_id (int): A unique identifier for this agent.
        model (Model): The model instance in which the agent exists.
    """
    self.unique_id = unique_id
    self.model = model
    self.pos: Position | None = None

    # register agent
    try:
        self.model.agents_[type(self)][self] = None
    except AttributeError:
        # model super has not been called
        self.model.agents_ = defaultdict(dict)
        self.model.agents_[type(self)][self] = None
        self.model.agentset_experimental_warning_given = False

        warnings.warn(
            "The Mesa Model class was not initialized. In the future, you need to explicitly initialize the Model by calling s
            FutureWarning,
            stacklevel=2,
        )
```

Lets first look at how we can use the modeling modules from MESA to create agents

It is convenient to create object for agents using classes to define each agent’s individual behaviors and parameters

very basic agent object

```
import random
import mesa

class agent(mesa.Agent):
    def __init__(self,model,name,wealth,greed):
        super().__init__(name,model)
        self.name = name
        self.wealth = wealth
        self.greed = greed
```

sets the name, wealth, and greed parameters

self.<name> specifies that specific instance of that agent.

<__main__.Agent object at 0x169a8d600>

<__main__.Agent object at 0x13f42ba60>

Lets first look at how we can use the modeling modules from MESA to create agents

It is convenient to create object for agents using classes to define each agent's individual behaviors and parameters

very basic agent object

```
import random
import mesa

class agent(mesa.Agent):
    def __init__(self,model,name,wealth,greed):
        super().__init__(name,model)
        self.name = name
        self.wealth = wealth
        self.greed = greed
    def step(self):
        print(self.wealth)
```

function that defines the behavior that an agent can take each time step

group tasks (15 mins)

- each group is assigned a code template to comment
- your task is to comment the code after each block where it says “#comment here 1”
- then run each block
- we will share our understandings at (12:15 – 12:30)

links

- group A
- https://colab.research.google.com/drive/1ASYMwJq0-790jKTV2RQClbooVJp1PI-Q?usp=sharing
- group B
- https://colab.research.google.com/drive/1BwFP_CqCuy2DGiPPxpbMEqrEAFHL_4Xb?usp=sharing
- group C
- https://colab.research.google.com/drive/1JBxEytzosYkTZBrXVOp6ajbNRgU3pFyJ?usp=sharing
- group D
- https://colab.research.google.com/drive/1JBxEytzosYkTZBrXVOp6ajbNRgU3pFyJ?usp=sharing

Agent based modeling Part 3

Lets next look at how we can use the modeling modules from MESA to create the model

very basic model object

model objects that reads in the parent class mesa.Model

```
class MyModel(mesa.Model):
        def __init__(self, n_agent):
                super().__init__()
                self.schedule = mesa.time.RandomActivation(self)
                self.grid = mesa.space.MultiGrid(10,10,torus =true)
                for i in range(n_agents):
                        wealth = random.randint(1,6)
                        greed = random.randint(0,1)
                        a =  agent(self, i, wealth, greed)
                        self.schedule.add(a)
                        coords = (self.random.randrange(0,10),self.random.randrange(0,10))
                        self.grid.place_agent(a,coords)
        def step(self):
                self.schedule.step()
```

Lets next look at how we can use the modeling modules from MESA to create the model

very basic model object

```
class MyModel(mesa.Model):
    def __init__(self, n_agent):
        super().__init__()
        self.schedule = mesa.time.RandomActivation(self)
        self.grid = mesa.space.MultiGrid(10,10,torus =true)
        for i in range(n_agents):
            wealth = random.randint(1,6)
            greed = random.randint(0,1)
            a =  agent(self, i, wealth, greed)
            self.schedule.add(a)
            coords = (self.random.randrange(0,10),self.random.randrange(0,10))
            self.grid.place_agent(a,coords)
    def step(self):
        self.schedule.step()
```

creates schedule object

- The scheduler controls the order in which agents are activated, it is responsible for when the step() function in agent is called.
- The scheduler is also responsible for advancing the model by one step.
- RandomActication means that each agent in the schedule is activated randomly in a time step

Lets next look at how we can use the modeling modules from MESA to create the model

very basic model object

```
class MyModel(mesa.Model):
        def __init__(self, n_agent):
                super().__init__()
                self.schedule = mesa.time.RandomActivation(self)
                self.grid = mesa.space.MultiGrid(10,10,torus =true)
                for i in range(n_agents):
                        wealth = random.randint(1,6)
                        greed = random.randint(0,1)
                        a =  agent(self, i, wealth, greed)
                        self.schedule.add(a)
                        coords = (self.random.randrange(0,10),self.random.randrange(0,10))
                        self.grid.place_agent(a,coords)
        def step(self):
                self.schedule.step()
```

creates grid object,
torus means we treat the grid like a torus (we allow looping)

for loop that creates the specified number of agent objects.
for each agent:
- generates a random wealth and greed value
- adds the agent object to the schedule
- adds the agent object to a randomly generated location in the grid

Lets next look at how we can use the modeling modules from MESA to create the model

very basic model object

```
class MyModel(mesa.Model):
    def __init__(self, n_agent):
        super().__init__()
        self.schedule = mesa.time.RandomActivation(self)
        self.grid = mesa.space.MultiGrid(10,10,torus =true)
        for i in range(n_agents):
            wealth = random.randint(1,6)
            greed = random.randint(0,1)
            a =  agent(self, i, wealth, greed)
            self.schedule.add(a)
            coords = (self.random.randrange(0,10),self.random.randrange(0,10))
            self.grid.place_agent(a,coords)
    def step(self):
        self.schedule.step()
```

```
class agent(mesa.Agent):
    def __init__(self, model,name,wealth,greed):
        super().__init__(name,model)
        self.name = name
```

Lets next look at how we can use the modeling modules from MESA to create the model

very basic model object

```
class MyModel(mesa.Model):
        def __init__(self, n_agent):
                super().__init__()
                self.schedule = mesa.time.RandomActivation(self)
                self.grid = mesa.space.MultiGrid(10,10,torus =true)
                for i in range(n_agents):
                        wealth = random.randint(1,6)
                        greed = random.randint(0,1)
                        a =  agent(self, i, wealth, greed)
                        self.schedule.add(a)
                        coords = (self.random.randrange(0,10),self.random.randrange(0,10))
                        self.grid.place_agent(a,coords)
        def step(self):
                self.schedule.step()
```

This class activates all the agents once per step, in random order
Calls the scheduler's step() function which then call's the agent's step function.

Lets next look at how we can use the modeling modules from MESA to create the model

very basic model object

```
class MyModel(mesa.Model):
    def __init__(self, n_agent):
        super().__init__()
        self.schedule = mesa.time.RandomActivation(self)
        self.grid = mesa.space.MultiGrid(10,10,torus =true)
        for i in range(n_agents):
            wealth = random.randint(1,6)
            greed = random.randint(0,1)
            a =  agent(self, i, wealth, greed)
            self.schedule.add(a)
            coords = (self.random.randrange(0,10),self.random.randrange(0,10))
            self.grid.place_agent(a,coords)
    def step(self):
        self.schedule.step()
```

This class activates all the agents once per step, in random order Calls the scheduler's step() function which then call's the agent's step function.

Now lets go to collab for a quick demo!

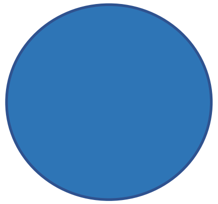
Right now, our model can't really do much……

In order to add more detail to our model, we first need to determine the order we want each agent to do things

Right now, our model can’t really do much……

In order to add more detail to our model, we first need to determine the order we want each agent to do things

for each time step



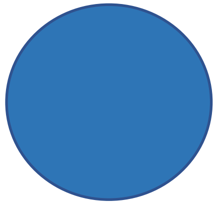
we need the agent to:

- move
- take/give wealth if applicable

Right now, our model can’t really do much……

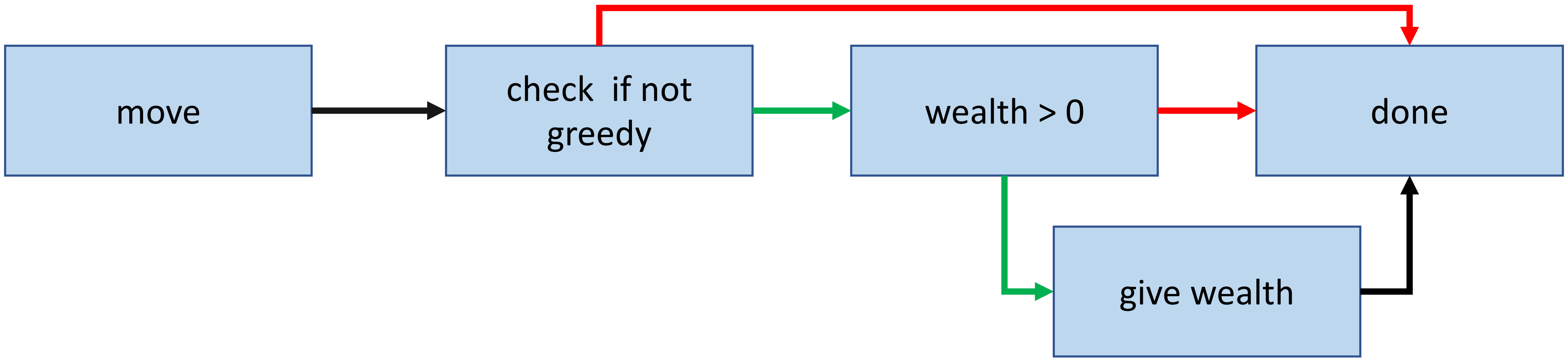
In order to add more detail to our model, we first need to determine the order we want each agent to do things

for each time step



we need the agent to:

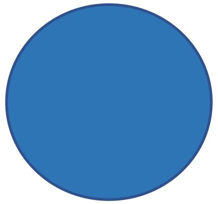
- move
- give wealth if applicable



Right now, our model can't really do much……

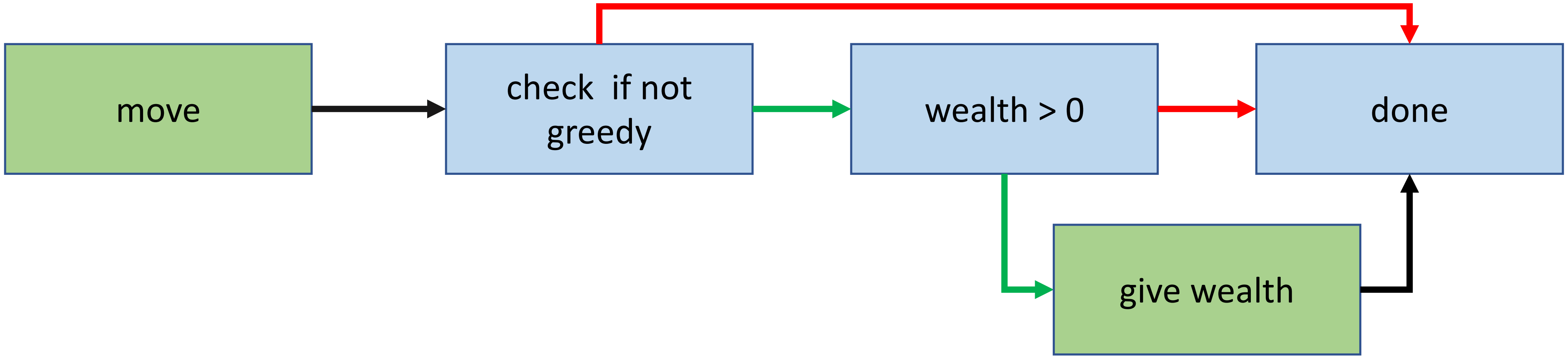
In order to add more detail to our model, we first need to determine the order we want each agent to do things

for each time step



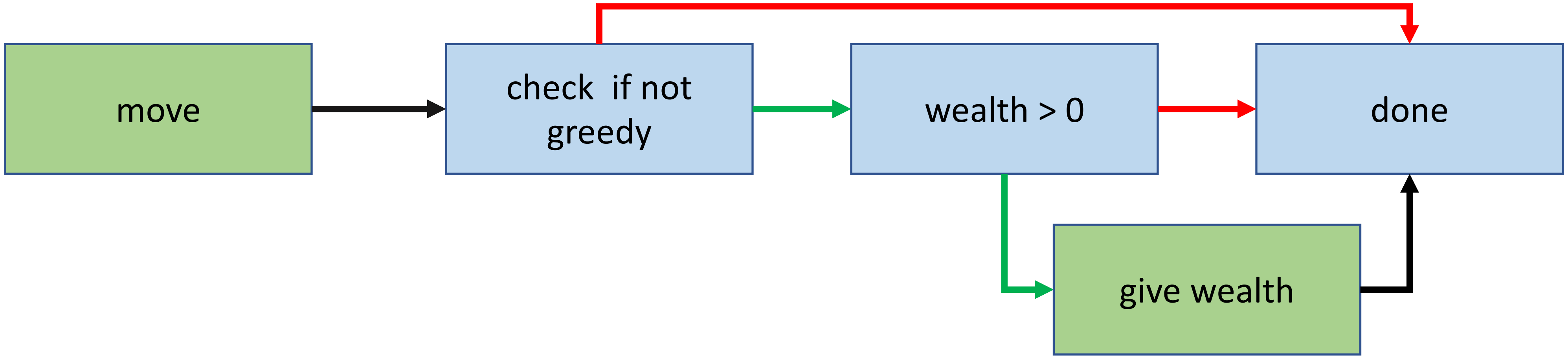
we need the agent to:

- move
- give wealth if applicable



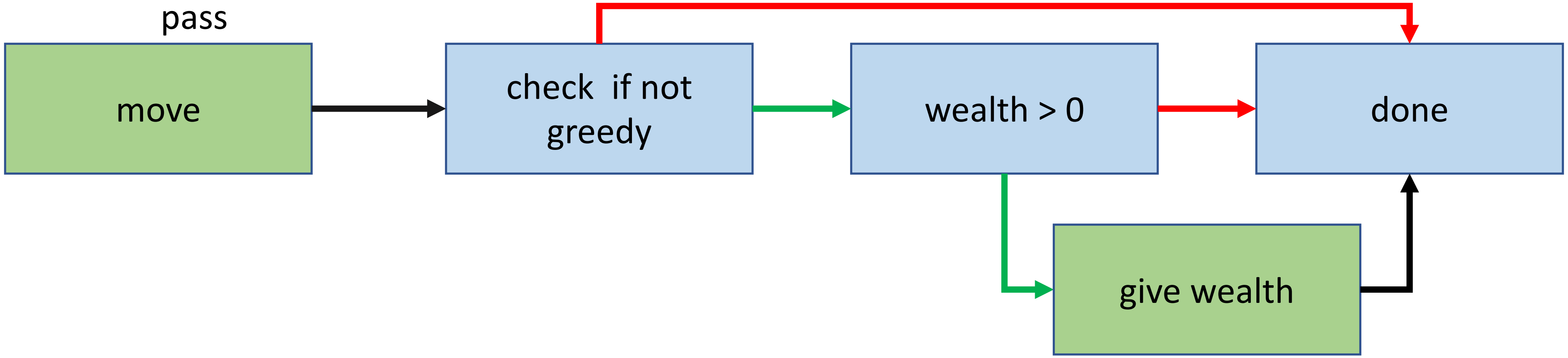
```
class agent(mesa.Agent):
    def __init__(self,model,name,wealth,greed):
        super().__init__(name,model)
        self.name = name
        self.wealth = wealth
        self.greed = greed
    def step(self):
        print(self.wealth)
    def move(self):
        pass
    def giveWealth(self):
        pass
```

place holder functions in the agent class



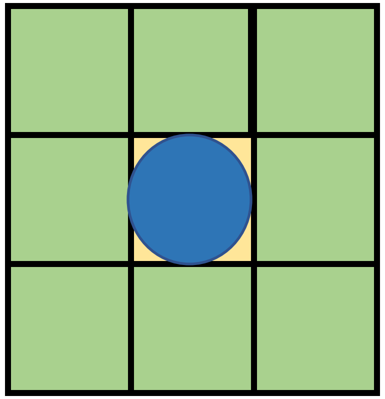
```
class agent(mesa.Agent):
        def __init__(self,model,name,wealth,greed):
                super().__init__(name,model)
                self.name = name
                self.wealth = wealth
                self.greed = greed
        def step(self):
                self.move()
                if self.greed == 0:
                        if self.wealth > 0:
                                self.giveWealth()
        def move(self):
                pass
        def giveWealth(self):
                pass
```

place holder functions in the agent class



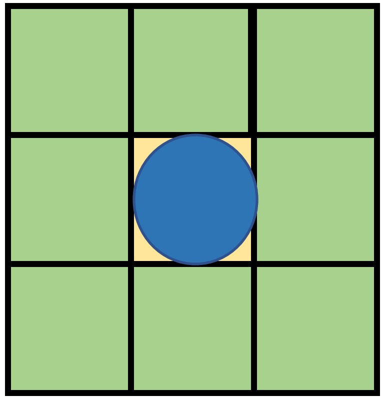
Lets add in movement

- we want our agent to consider all tiles on the grid that are adjacent to it
- the agent should move to one of the adjacent tiles randomly



Lets add in movement

- we want our agent to consider all tiles on the grid that are adjacent to it
- the agent should move to one of the adjacent tiles randomly



def move(self):

make a list of all possible locations you can move to

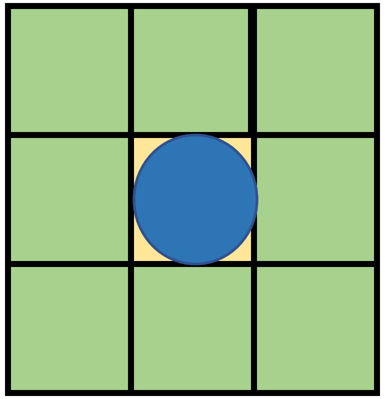
pick one at random

move to that location

```
def move(self):
possible_steps = self.model.grid.get_neighborhood(self.pos, moore = True,
include_center = False)
new_position = self.random.choice(possible_steps)
self.model.grid.move_agent(self, new_position)
```

Lets add in movement

- we want our agent to consider all tiles on the grid that are adjacent to it
- the agent should move to one of the adjacent tiles randomly



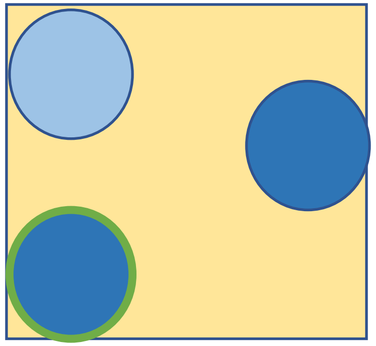
def move(self):

make a list of all possible locations you can move to
pick one at random
move to that location

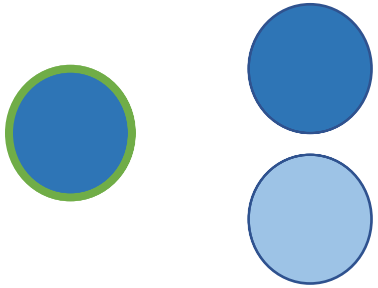
```
def move(self):
possible_steps = self.model.grid.get_neighborhood(self.pos, moore = True, include_center = False)
new_position = self.random.choice(possible_steps)
self.model.grid.move_agent(self, new_position)
```

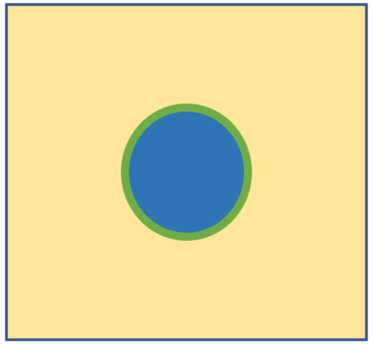
now let's add the giveWealth function

- we want to have a non-greedy agent with at least 1 wealth consider the other agents on the same tile
- if there are other agents on the same tile, the non-greedy agent should give 1 wealth to one of the other agents on the same tile
- if there are no other agents, nothing happens

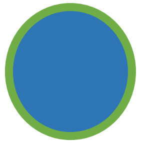


pick one agent to give to





nobody to give to



now let’s add the giveWealth function

- we want to have a non-greedy agent with at least 1 wealth consider the other agents on the same tile
- if there are other agents on the same tile, the non-greedy agent should give 1 wealth to one of the other agents on the same tile
- if there are no other agents, nothing happens

```
def giveWealth(self):
    make a list of all potential recipients in the current position
    remove your self from contention (you can’t donate to yourself)
    if there are possible recipients to donate to
        pick a random agent from the list of potential recipients
        give one wealth to them
```

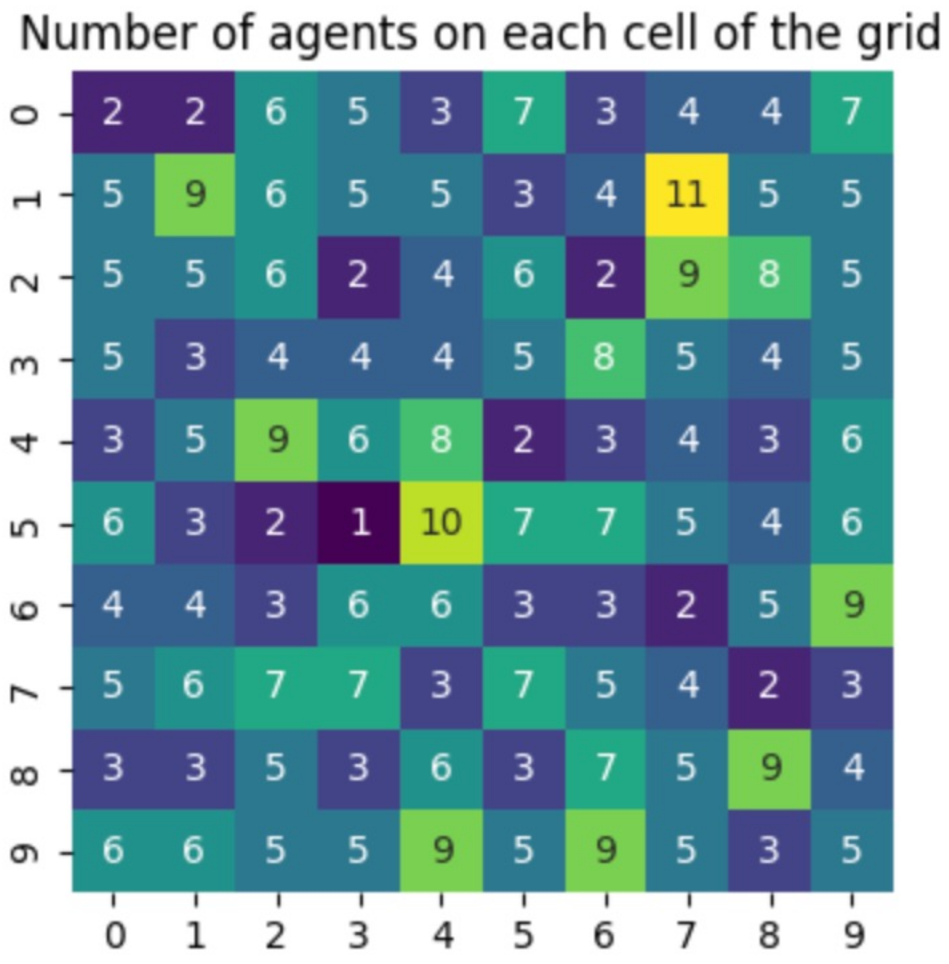
def giveWealth(self):
    make a list of all potential recipients in the current position
    remove your self from contention (you can’t donate to yourself)
    if there are possible recipients to donate to
        pick a random agent from the list of potential recipients
        give one wealth to them

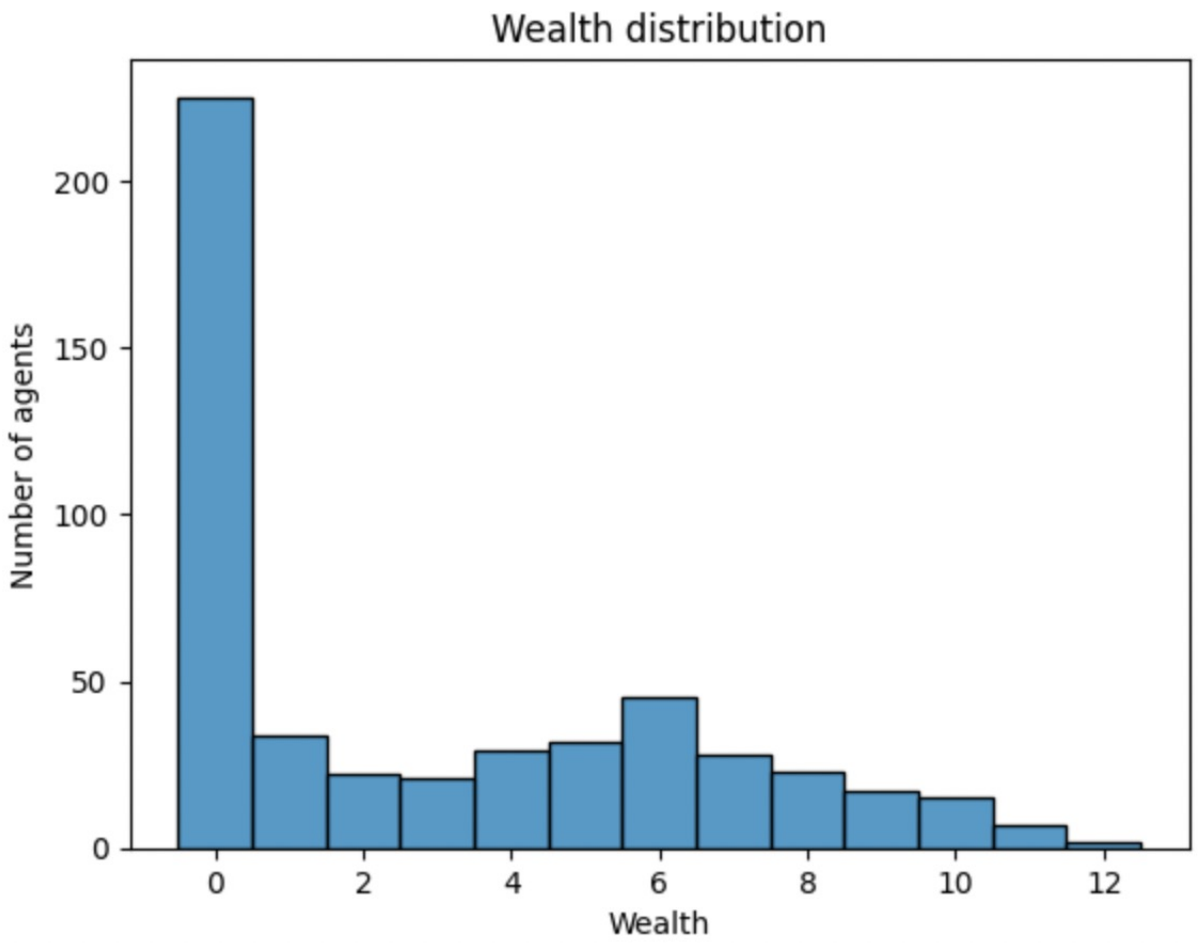
```
def giveWealth(self):
    cellmates = self.model.grid.get_cell_list_contents([self.pos])
    cellmates.pop(cellmates.index(self))
    if len(cellmates) > 1:
        other = self.random.choice(cellmates)
        other.wealth += 1
        self.wealth -= 1
```

Last thing! data collection

- we will go over how to do this more in depth tommorow with the mesa datacollection libraries
- for now, we will do a simple procedure

Last thing! data collection





group tasks (5 mins)

- each group is assigned a code template to comment
- your task is to comment the code after each block where it says “#comment here 2”
- then run each block
- we will share our understandings after

links

- group A
- https://colab.research.google.com/drive/1Yv3jEewnYH0FVz8u_Tj5DURN0xadIZA2?usp=sharing
- group B
- https://colab.research.google.com/drive/16ZsVHpW0m53qT-ujTP3ksgEk-Vm1JvjT?usp=sharing
- group C
- https://colab.research.google.com/drive/1w3UUNi9KUkCxvZBcdafprxdFTcKHl1tC?usp=sharing
- group D
- https://colab.research.google.com/drive/1Tyl89DtZ1gmC4C1hxoQ-wfm8HFWGnGYY?usp=sharing

Individual work

- save a copy of the code
- try to implement at least one of following
  - add a new parameter
  - add a new branch in the algorithm
  - plot the new parameter

  - add some stochastic decision making:
  - add an option for a non greedy agent to decide not to give wealth to a greedy agent
- Share code with me!