This is an overview of how to make basic changes to the core and significant parts of the framework.

Below are examples of possible changes to the framework that could be made by researchers. The changes that could be made to the framework are limitless; however, this might serve to give a starting point for researchers to begin modifying and understanding the model and framework. This document aims to provide explanations of core function interactions and an overview of potential changes that can be made to each section.

A pdf was chosen over something like a Jupyter notebook to be as inclusive as possible of researchers with little to no coding or Python experience.

<u>Changing basic agent behavior</u>

```python
# if - 1. other non-sugar agents within vision
# 2. random.random() < trauma level
# 3. agent is starving
# 4. the max sugar on the visible canvas < this agent's metabolism
# then engage in possible trauma influenced behaviors
if len(agent_neighbors) > 0 and self.random.random() < self.trauma and self.starvation > -1 \
    and max_sugar < self.metabolism and trauma_influenced_behavior:

    # pick random non-sugar agent and move to them
    self.random.shuffle(agent_neighbors)
    pos = agent_neighbors[0]

    this_cell = self.model.grid.get_cell_list_contents(pos)
    for agent in this_cell:
        if isinstance(agent,SsAgent):
            break


    # trauma influenced behavior #
    # the starvation level and trauma level affects what the agent is
    # capable of doing to other agents
    if self.starvation > 15 and self.random.random() < self.trauma * .1:
        self.sugar += agent.is_cannibalized()
        # this agent becomes more traumatized by cannibalizing another
        self.trauma += .05
    elif self.starvation > 5 and self.random.random() < self.trauma * .5:
        self.sugar += agent.is_killed()
    elif self.starvation <= 5 and self.random.random() < self.trauma * 1.0:
        self.sugar += agent.is_mugged()

    self.model.grid.move_agent(self,pos)
# else if there is no sugar on the visible canvas and no non-sugar agents,
# just move somewhere random within vision
elif max_sugar == 0:
    self.random.shuffle(neighbors)
    self.model.grid.move_agent(self, neighbors[0])

# else, move to cell with the most sugar that is nearest to consume it
# when the "eat" function is called
else:

    # Look for location with the most sugar
    candidates = [
        pos for pos in neighbors if self.get_sugar(pos).amount == max_sugar
    ]
    # Narrow down to the nearest ones
    min_dist = min(get_distance(self.pos, pos) for pos in candidates)
    final_candidates = [
        pos for pos in candidates if get_distance(self.pos, pos) == min_dist
    ]

    self.random.shuffle(final_candidates)
    self.model.grid.move_agent(self, final_candidates[0])
```

We see 3 major if-statements in this above pictured section of code (which is in the move function in agents.py). The first if-statement is testing for five things to be true before continuing to execute code that determines the action of the agent based on the influence trauma exerts:

1. There must be at least one other agent in its vicinity of movement (neighborhood)
    a. There must be another agent to act on in order to see the effects of trauma influence
2. A probability draw succeeds against the trauma level
    a. This just means that the probability of an agent acting because of their trauma is proportional to the amount of trauma they have. A more traumatized agent is linearly more probable to act due to trauma.
3. This agent is currently starving
    a. This and #4 determine if an agent is in a dire state of needing resources for survival.
4. The maximum sugar on the canvas the agent sees in its neighborhood is less than its metabolism
    a. If an agent isn't able to obtain enough food to last even another time step, then the agent may turn to obtaining food from other agents.
5. The variable trauma_influenced_behavior is set to True
    a. This variable is just a setting allowing a researcher to easily turn off agents acting on trauma for control runs

Modifying or adding constraints to this if-statement can allow researchers to adjust when agents will show the influence of trauma through their actions. The checks can be probabilistic (2) or regular variable checks (1,3,4,5); both types are seen in this example.

Within this first if-statement between lines 254 and 264 contains the functionality of trauma-influenced behavior. Each if-statement has a both a higher starvation threshold and lower probability from a random draw for increasingly extreme actions taken by agents, from mugging, to killing another agent, to cannibalizing other agents. By adjusting the if-statements and adding or adjusting the functions seen here, researchers can adjust how and when trauma-influenced behavior based on the current state of the agent. There is a possibility that the agent will do nothing besides move when acting from trauma, since it is probability based.

The second if-statement determines what the current agent will do if there are no other agents nearby and there is no sugar on the landscape to consume.

The third if-statement catches all the other cases and just makes the current agent move to the cell with the most sugar. If there is a tie, it will choose a random cell that is tied for the most sugar.

Adjusting any of these if-statements and the code within adjusts the core behavior of the agents in the simulation.

Extending and customizing epigenetic features

```
def prenatal_trauma_create(self,eld):
    '''
    Example of prenatal trauma creating transgenerational epigenetic effects

    According to the literature, life expectancy decreases
    if the offspring are the same sex 2 generations later, so that is included
    in the trigger criteria dictionary.

    The trigger age is 0 so that this effect is present starting at birth

    Parameters
    ----------
    eld (expected life decrease): float (0-1)
        Percentage of the decreased lifespan of the descendant of this agent
        in 2 generations.

    Returns
    -------
    None.

    '''
    self.epigenetic_lifespan_decrease_prenatal += eld
    self.epigenetic_lifespan_decrease_prenatal = min(self.epigenetic_lifespan_decrease_prenatal,0.2)
    # trigger is (gen + 3) b/c it is 2 generations after the agent currently about to be birthed (prenatal)
    # also 'sex' of the next child is chosen now, since the epigenetic effects are based on that
    self.next_birth_sex = self.random.choice(['m','f'])
    triggers = {'generation':self.generation+3,'age':0,'sex':self.next_birth_sex}
    expression = ['prenatal_trauma_express',self.epigenetic_lifespan_decrease_prenatal]
    self.future_epigenetic_symptoms['prenatal1'] = [triggers,expression]

    return None
```

Let's take an example of prenatal trauma from starvation, as this serves as an excellent semi-complex example of employing the customizable epigenetic features of this framework. Prenatal starvation trauma affects descendants who are the same sex and are two generations later than the person who is in utero at the time of starvation.

Below is a code block in the agent init function. We can see here that we created an object variable that is named “epigenetic_lifespan_decrease_prenatal” which we made specifically for this function, since it models an epigenetic feature that reduces the lifespan of particular descendants.

```
# epigenetic
self.future_epigenetic_symptoms = dict()
self.epigenetic_lifespan_decrease_prenatal = 0
self.epigenetic_lifespan_increase_prepubecent = 0
```

This variable is the first edited in the function. Since it is an object variable, modifying the variable within the scope of the function is easily done.

```
# pre-natal traumas have transgenerational epigenetic effects
if self.pregnant:
    self.prenatal_trauma_create(.04)
```

When the function is invoke in the framework, the eld parameter is .04, meaning that the object variable "epigenetic_lifespan_decrease_prenatal" will have .04 added to it (this will be used to effectively reduce the lifespan of the affected descendants by 4% each time this is called).

Line 548 limits the lifespan reduction to a maximum of 20%.

Self.next_birth_sex was created specifically for this epigenetic modification, since the epigenetic effects are dependent on the agent *in utero*. Normally, the sex of the agent is determined at birth, but adding this selection in the function, in addition to the code blocks below, allows the simulation to know the sex of the next agent birth ahead of time.

```
        if sex is None:
            self.sex = self.random.choice(['m','f'])
        else:
            self.sex = sex
```

```
            if epigenetic_effects:
                ssa = SsAgent(self.model.agent_id, self.model, False,
                               sugar = int(self.sugar*.5), trauma = self.trauma,
                                trauma_min = tm_offspring, cortisol = cortisol_offspring,
                              generation = self.generation, family = self.family,
                              sex = self.next_birth_sex,
                                epigenetic_symptoms = eg_for_birth
                              )
                self.next_birth_sex = None
```

Lines 552-554 in the first screenshot of this section showcase the important part of the code example for understanding how the customization and extensibility operates in the framework.

The variable "triggers" must be a dictionary where each key is an agent attribute that every agent will be initialized with. For each of these attributes, the writer of the code must specify what the values of each attribute must be in order for the epigenetic modification to take effect. All trigger parameters must be met for this to occur. The triggers dictionary can use nearly any combination of universal agent attributes; however, the generation agent attribute must be specified as a trigger parameter.

By setting 'age' to 0, we effectively only apply this epigenetic effect one time to the agent that inherits this epigenetic mutation. Without the age parameter being set in the triggers dictionary, the epigenetic mutation would be applied every timestep to said descendant agent.

Line 554 shows the way to model the epigenetic effects themselves. The expression variable should always be a list with at least one element. The first element needs to be the name of the function that expresses the trauma in the agent as a string. All subsequent elements must be the input parameter values for that function.

```python
    def prenatal_trauma_express(self,dec_perc):
        '''
        Expression of the transgenerational epigentic effects of prenatal trauma.

        For every epigenetic symptom, there needs to be a function used to express
        that effect. This is an example of proper implementation.

        Parameters
        ----------
        dec_perc : float (0-1)
            Percentage that the lifespan of this agent will be decreased by
            (affects only death by old age value)

        Returns
        -------
        None.

        '''

        self.death *= 1 - dec_perc

        return None
```

In this case, when the descendant agent experiences this epigenetic mutation, this function will be called at age = 0 (birth), where the lifespan variable is decreased by the amount specified generations earlier when the trauma occurred. **For every epigenetic expression created, you must have a corresponding function that expresses the trauma.**

Line 555 in the screenshot shows the boring, yet vital step of simple creating a unique key for the epigenetic creation function in the "self.future_epigenetic_symptoms" variable, with the values as a list with the triggers dictionary and the expression list. The key these values are assigned to makes no difference, as long as it is unique per epigenetic creation function.

Using the triggers dictionary to customize the necessary conditions for epigenetic effects to appear in later generations and using the expression list combined with custom epigenetic effect functions allows researchers to fully customize and extend the use of this framework to their own need. If researchers email us with other questions about changing/extending/customizing other parts of the framework, this document will be updated to reflect those additional explanations.

Email of the primary developer: [REDACTED FOR ANNSIM BLIND REVIEW]