# Source Code

```
# Project 2017 ATC - Flu Transmission
# Built on Mesa Agent Library from George Mason Univ.
# Written by Ben Thorp
# ATC-3 Ben Thorp, Ben Sheffer, Alex Baten, Teddy Gonzales
# Version 1.1
#   Added start and stop of contagiousness
# Version 1.2
#   Added HealthCareAccess and HealthLifeStyle

from mesa import Agent, Model
from mesa.time import RandomActivation
import random
from mesa.space import MultiGrid
from mesa.datacollection import DataCollector
import matplotlib.pyplot as plt

# Function that computs the number of infections for graphing
def compute_infections(model):
    total_inf_count = 0
    for cell in model.grid.coord_iter():
        cell_content, x, y = cell
        for human in cell_content:
            if human.infected:
                total_inf_count += 1
    return total_inf_count

# Function that computes the number of infections in community 1 for graphing
def compute_infections_c1(model):
    total_inf_count = 0
    for cell in model.grid.coord_iter():
        cell_content, x, y = cell
        for human in cell_content:
            if human.infected:
                if human.community1 is True:
                    total_inf_count += 1
    return total_inf_count
```

```
# Function that computs the number of infections in community 2 for graphing
def compute_infections_c2(model):
    total_inf_count = 0
    for cell in model.grid.coord_iter():
        cell_content, x, y = cell
        for human in cell_content:
            if human.infected:
                if ( human.community1 is False ):
                    total_inf_count += 1
    return total_inf_count

# Function that computs the immunity for everyone for graphing
def compute_immunity(model):
    total_im_count = 0
    for cell in model.grid.coord_iter():
        cell_content, x, y = cell
        for human in cell_content:
            if human.immunity:
                total_im_count += 1
    return total_im_count

class InfectModel(Model):
    """A Mesa Model to simulate the spread of disease through a home and work
environment"""

    # __init__ creates the model
    def __init__(self, N, h,
                 hls_com1, hls_com2, lowL, highL,
                 con_start, con_end_short, con_end_long,
                 com1_hca, com2_hca, vac_com1, vac_com2):
        # N - the total number of humans
        # h - the number of houses or rooms in each environment
        # hls_com1 - the percentage of people in community1
        #     that are living a healthy lifestyle
        # hls_com2 - the percentage of people in communtiy2
        #     that are living a healthy lifestyle
        # lowL - a low likelyhood to catch the disease (
        #     assosiated with healthy lifestyle)
```

```
# highL - a high likelyhood to catch the disease (
#     assosiated with healthy lifestlye)
# con_start - when the disease starts to be contagious
# con_end_short - when the disease stops being contagious with health care
# con_end_long - when the disease stops being contagious without health care
# com1_hca - the health care access in community1
# com2_hca - the health care access in community2
# vac_com1 - percentage of vaccinated humans in community1
# vac_com2 - percentage of vaccinated humans in community2
self.num_agents = N
self.grid = MultiGrid(4, h, True)
self.schedule = RandomActivation(self)
self.community1_hca = com1_hca
self.community2_hca = com2_hca
self.con_end_short = con_end_short
self.con_end_long = con_end_long
self.vaccinated_com1 = vac_com1
self.vaccinated_com2 = vac_com2
self.healthy_lifestyle_com1 = hls_com1
self.healthy_lifestyle_com2 = hls_com2
self.low_likelyhood = lowL
self.high_likelyhood = highL

# Create N humans for the model
for i in range(self.num_agents):
    a = Human(i, self, con_start)
    self.schedule.add(a)

    # Add the agent to a random grid cell
    if (a.community1 is True):
        x = 0
    else:
        x = 1
    y = random.randrange(self.grid.height)
    self.grid.place_agent(a, (x, y))

# Initialize timestep
self.timestep=0
self.day=True
```

```
        # Initialize the software that collects the data each timestep
        self.datacollector = DataCollector(
            model_reporters={"Community2": compute_infections_c2,
                             "Community1": compute_infections_c1,
                             "Immunity": compute_immunity,
                             "Infections": compute_infections,
                             }
            )

    def step(self):
        # Collects the data for this timestep
        self.datacollector.collect(self)

        self.schedule.step()
        self.timestep+=1
        if (self.timestep % 2 == 0 ):
            # Day time
            self.day=True
        else:
            # Night time
            self.day=False

    # run_steps steps the model forword for "steps" (days/nights)
    def run_steps(self, steps):
        for i in range(steps*2):
            # Doubling steps makes correct number of day/night cycles
            # Because a day/night cycle takes two timesteps
            self.step()
        inf_data = self.datacollector.get_model_vars_dataframe()
        inf_data.plot()
        plt.show()

class Human(Agent):
    """An agent that represents one human in the model"""
    def __init__(self, unique_id, model, con_start):
        # unique_id - the human's id number
        # model - the Mesa simulation class
        # likelihood - the chance that the infection spreads from one human to another
```

```
# con_start - when the disease starts to be contagious

# Call the Mesa agent setup
super().__init__(unique_id, model)

# Initialize the human variables
self.community1 = random.choice([True, False])
self.adult=random.choice([True, False])
self.household=random.randrange(model.grid.height)
self.schoolroom=random.randrange(model.grid.height)
self.workroom=random.randrange(model.grid.height)

# Initialize disease variables
# self.likelihood = likelihood
if self.community1 == True:
    if (random.random() <= model.healthy_lifestyle_com1):
        self.likelihood = model.low_likelyhood
    else:
        self.likelihood = model.high_likelyhood
else:
    if (random.random() <= model.healthy_lifestyle_com2):
        self.likelihood = model.low_likelyhood
    else:
        self.likelihood = model.high_likelyhood

self.con_timer = 0
self.con_start = con_start

# change length of disease based on community health care access
if self.community1 == True:
    if model.community1_hca == True:
        self.con_end = model.con_end_short
    else :
        self.con_end = model.con_end_long
else :
    if model.community2_hca == True:
        self.con_end = model.con_end_short
    else :
        self.con_end = model.con_end_long
```

```
        self.immunity = False
        if self.community1 == True:
            if (random.random() <= model.vaccinated_com1):
                self.immunity = True

        else:
             if (random.random() <= model.vaccinated_com2):
                self.immunity = True

        self.infected = False
        if unique_id==1:
            self.infected=True
        if unique_id==2:
            self.infected=True

    def move(self):
        community0_row=0
        community1_row=1
        schoolroom_row=2
        workroom_row=3
        if (self.model.day): # Day
            if (self.adult): # Adults go to work
                # workroom_row - the work environment
                # self.workroom - the room in the work environment
                new_position = (workroom_row, self.workroom)
            else : # Kids go to school
                new_position = (schoolroom_row, self.schoolroom)
        else : # Night
            if (self.community1): # Assigns the human to its correct community
                new_position = (community0_row, self.household)
            else :
                new_position = (community1_row, self.household)
        self.model.grid.move_agent(self, new_position)

    def infect_others(self):
        cellmates = self.model.grid.get_cell_list_contents([self.pos])
        if len(cellmates) > 1:
            for other in cellmates:
```

```
            if other.immunity is False:
                if other.infected is False:
                    if(random.random() <= self.likelihood):
                        other.infected = True

def step(self):
    self.move()
    if self.infected:
        self.con_timer +=1
        if (( self.con_timer >= self.con_start) and (
          self.con_timer <= self.con_end)):
                self.infect_others()
        else:
           if ( self.con_timer > self.con_end):
                self.immunity = False
                if self.community1 == True :
                    if self.model.community1_hca == True:
                        self.immunity = True
                else:
                    if self.model.community2_hca == True:
                        self.immunity = True

                self.infected = False
```