```python
#Classes Challenge 40:  Epidemic Outbreak GUI App
import math
import random
import tkinter

class Simulation():
    """A class to control a simulation and facilitate the spread of a disease."""

    def __init__(self):
        """Initialize attributes"""
        self.day_number = 1

        #Get simulation initial conditions from the user
        #Population size must be a perfect square for this program
        print("To simulate an epidemic outbreak, we must know the population size.")
        self.population_size = int(input("---Enter the population size: "))
        #Convert users population size to nearest perfect square for visual purposes
        root = math.sqrt(self.population_size) #For example, if population_size is 79, root = 8.8881

        #User did not enter a perfect square for the population
        if int(root + .5)**2 != self.population_size: # int(8.881 +.5)**2 = int(9.3881)**2 = 9**2 = 81 != 79
            root = round(root, 0) # round(8.881, 0) = 9.0
            self.grid_size = int(root) #grid_size = 9
            self.population_size = self.grid_size**2 #population_size = 9*9 = 81 the closest PERFECT SQUARE TO 79
            print("Rounding population size to " + str(self.population_size) + " for visual purposes.")
        #The user did enter a perfect square for the population
        else:
            self.grid_size = int(math.sqrt(self.population_size))

        print("\nWe must first start by infecting a portion of the population.")
        self.infection_percent = float(input("---Enter the percentage (0-100) of the population to initially infect: "))
        self.infection_percent /= 100

        print("\nWe must know the risk a person has to contract the disease when exposed.")
        self.infection_probability = float(input("---Enter the probability (0-100) that a person gets infected when exposed to the disease: "))

        print("\nWe must know how long the infection will last when exposed.")
        self.infection_duration = int(input("---Enter the duration (in days) of the infection: "))

        print("\nWe must know the mortality rate of those infected.")
        self.mortality_rate = float(input("---Enter the mortality rate (0-100) of the infection: "))

        print("\nWe must know how long to run the simulation.")
        self.sim_days = int(input("---Enter the number of days to simulate: "))


class Person():
    """A class to model an individual person."""

    def __init__(self):
        """Initialize attributes"""
        self.is_infected = False #Person starts healthy, not infected
        self.is_dead = False #Person starts ALIVE
```

```
54          self.days_infected = 0 #Keeps track of days infected for individual
    person
55
56
57      def infect(self, simulation):
58          """Infect a person based on sim conditions"""
59          #random number generated must be less than infection_probability to
    infect
60          if random.randint(0, 100) < simulation.infection_probability:
61              self.is_infected = True
62
63
64      def heal(self):
65          """Heals a person from an infection"""
66          self.is_infected = False
67          self.days_infected = 0
68
69
70      def die(self):
71          """Kill a person"""
72          self.is_dead = True
73
74
75      def update(self, simulation):
76          """Update an individual person if the person is not dead.  Check if they
    are infected
77              If they are, increase the days infected count, then check if they
    should die or be healed."""
78          #Check if the person is not dead before updating
79          if not self.is_dead:
80              #Check if the person is infected
81              if self.is_infected:
82                  self.days_infected += 1
83                  #Check to see if the person will die
84                  if random.randint(0, 100) < simulation.mortality_rate:
85                      self.die()
86                  #Check if the infection is over, if it is, heal the person
87                  elif self.days_infected == simulation.infection_duration:
88                      self.heal()
89
90
91  class Population():
92      """A class to model a whole population of Person objects"""
93
94      def __init__(self, simulation):
95          """Initialize attributes"""
96          #This will be a list of N lists, where N is the simulation grid size.
97          #Each list within the list will represent a row in an NxN grid.
98          #Each element of the row will represent an individual Person object.
99          #Each of these lists will hold N Person objects and there will be N
    lists.
100         self.population = [] #A list to hold all Persons in the population.
101
102         #Loop through the needed number of rows
103         for i in range(simulation.grid_size):
104             row = []
105             #Loop through the needed number of Person objects for each row
106             for j in range(simulation.grid_size):
107                 person = Person()
108                 row.append(person)
109             #The entire row is complete, append it to the population
110             self.population.append(row)
111
112
```

```
113     def initial_infection(self, simulation):
114         """Infect an initial portion of the population based on initial
    conditions of the sim"""
115         #Infect the infection_percent*population_size gives the total number to
    infect
116         #Round to 0 decimals and cast to int so it can be used in a loop.
117         infected_count =
    int(round(simulation.infection_percent*simulation.population_size, 0))
118 
119         infections = 0
120         #Infect the population until you have infected the correct starting
    amount
121         while infections < infected_count:
122             #x is a random row in the population, y is a random person in the
    random row
123             #self.population[x][y] represents a random person in the population
    list
124             x = random.randint(0, simulation.grid_size - 1)
125             y = random.randint(0, simulation.grid_size - 1)
126 
127             #If the person is not infected, infect them!
128             if not self.population[x][y].is_infected:
129                 self.population[x][y].is_infected = True
130                 self.population[x][y].days_infected = 1
131                 infections += 1
132 
133 
134     def spread_infection(self, simulation):
135         """Spread the infection in a 2D array to all adjacent people to a given
    person.
136             A given person in the population attribute is referenced as
    self.population[i][j]
137             A person to the right of the given person is referenced as
    self.population[i][j+1]
138             A person to the left of the given person is referenced as
    self.population[i][j-1]
139             A person below the given person is referenced as self.population[i+1]
    [j]
140             A person above the given person is referenced as self.population[i-1]
    [j]"""
141 
142         #Loop through all rows of the population
143         for i in range(simulation.grid_size):
144             #Loop through all of the Person objects in a given row
145             for j in range(simulation.grid_size):
146                 #Check to see if this given person self.population[i][j] is not
    dead
147                 if self.population[i][j].is_dead == False:
148                     #Check to see if we need to infect this person.
149                     #We will try infect the given person if an adjacent person
    is already infected
150                     #If i == 0, we are in the first row so, we can't look above
151                     if i == 0:
152                         #If j == 0, we are in the first column, so we can't look
    left.
153                         if j == 0:
154                             if self.population[i][j+1].is_infected or
    self.population[i+1][j].is_infected:
155                                 self.population[i][j].infect(simulation)
156                         #If we are in the last column, we can't look right
157                         elif j == simulation.grid_size-1:
158                             if self.population[i][j-1].is_infected or
    self.population[i+1][j].is_infected:
159                                 self.population[i][j].infect(simulation)
```

```
                        #If we are in any other column, we can look left, right, or below
                        else:
                            if self.population[i][j-1].is_infected or self.population[i][j+1].is_infected or self.population[i+1][j].is_infected:
                                self.population[i][j].infect(simulation)
                    #If i == simulation.grid_size -1 we are in the last row, so we can't look below
                    elif i == simulation.grid_size-1:
                        #If j == 0, we are in the first column, so we can't look left.
                        if j == 0:
                            if self.population[i][j+1].is_infected or self.population[i-1][j].is_infected:
                                self.population[i][j].infect(simulation)
                        #If we are in the last column, we can't look right
                        elif j == simulation.grid_size-1:
                            if self.population[i][j-1].is_infected or self.population[i-1][j].is_infected:
                                self.population[i][j].infect(simulation)
                        #If we are in any other column, we can look left, right, or above
                        else:
                            if self.population[i][j-1].is_infected or self.population[i][j+1].is_infected or self.population[i-1][j].is_infected:
                                self.population[i][j].infect(simulation)
                    #Otherwise, we are in a row in between, we can look left, right, below or above
                    else:
                        #If j == 0, we are in the first column, so we can't look left.
                        if j == 0:
                            if self.population[i][j+1].is_infected or self.population[i+1][j].is_infected or self.population[i-1][j].is_infected:
                                self.population[i][j].infect(simulation)
                        #If we are in the last column, we can't look right
                        elif j == simulation.grid_size-1:
                            if self.population[i][j-1].is_infected or self.population[i+1][j].is_infected or self.population[i-1][j].is_infected:
                                self.population[i][j].infect(simulation)
                        #If we are in any other column, we can look left, right, below, or above
                        else:
                            if self.population[i][j-1].is_infected or self.population[i][j+1].is_infected or self.population[i+1][j].is_infected or self.population[i-1][j].is_infected:
                                self.population[i][j].infect(simulation)


    def update(self, simulation):
        """Update the whole population by updating each individual Person"""
        simulation.day_number += 1
        #Loop through the population to access each row
        for row in self.population:
            #Loop through the row to update each Person
            for person in row:
                person.update(simulation)


    def display_statistics(self, simulation):
        """Display the statistics of the population"""
        #Initialize values
        total_infected_count = 0
        total_death_count = 0
```

```
209
210         #Loop through the population to access each row
211         for row in self.population:
212             #Loop through the row to access each person
213             for person in row:
214                 #Person is infected
215                 if person.is_infected:
216                     total_infected_count += 1
217                     #Person is dead
218                     if person.is_dead:
219                         total_death_count += 1
220
221         #Calculate percentage of population that is infected and dead
222         infected_percent = round(100*(total_infected_count/
    simulation.population_size), 4)
223         death_percent = round(100*(total_death_count/
    simulation.population_size), 4)
224
225         #Statistics summary
226         print("\n-----Day # " + str(simulation.day_number) + "-----")
227         print("Percentage of Population Infected: " + str(infected_percent) +
    "%")
228         print("Percentage of Population Dead: " + str(death_percent) + "%")
229         print("Total People Infected: " + str(total_infected_count) + " / " +
    str(simulation.population_size))
230         print("Total Deaths: " + str(total_death_count) + " / " +
    str(simulation.population_size))
231
232
233 #A helper function to create graphics
234 def graphics(simulation, population, canvas):
235     """A helper function to update the tkinter display."""
236     #Get the dimensions of an individual square in a grid
237     #Each square represents a person in the population
238     #Use 600 for a GUI window that is 600x600, change if desired.
239     #To get the dimensions of a square, take the dimensions of the window and
    divide by total number of squares in a row
240     square_dimension = 600//simulation.grid_size
241
242     #Loop through all rows in the population
243     for i in range(simulation.grid_size):
244         #y is the starting index of where a given square should be drawn
245         y = i*square_dimension
246         #Loop through all persons in the row
247         for j in range(simulation.grid_size):
248             #x is the starting index of where a given square should be drawn.
249             x = j*square_dimension
250
251             #Check to see if the given person is dead
252             if population.population[i][j].is_dead:
253                 #Create a red square at the correct location
254                 canvas.create_rectangle(x, y, x+square_dimension,
    y+square_dimension, fill='red')
255             #The current person is not dead, check if they are infected or
    healthy
256             else:
257                 if population.population[i][j].is_infected:
258                     #Create a yellow square at the correct location
259                     canvas.create_rectangle(x, y, x+square_dimension,
    y+square_dimension, fill='yellow')
260                 else:
261                     canvas.create_rectangle(x, y, x+square_dimension,
    y+square_dimension, fill='green')
262
```

```
263
264 #The main code
265 #Create a simulation object
266 sim = Simulation()
267
268 #Set constant variables for window size
269 WINDOW_WIDTH = 600
270 WINDOW_HEIGHT = 600
271
272 #Create the tkinter window and canvas
273 sim_window = tkinter.Tk()
274 sim_window.title("Epidemic Outbreak")
275 sim_canvas = tkinter.Canvas(sim_window, width=WINDOW_WIDTH,
    height=WINDOW_HEIGHT, bg='lightblue')
276 sim_canvas.pack(side=tkinter.LEFT)
277
278 #Create a population object
279 pop = Population(sim)
280
281 #Set the initial conditions of the population
282 pop.initial_infection(sim)
283 pop.display_statistics(sim)
284 input("Press Enter to begin simulation.")
285
286 #Run the simulation
287 for i in range(1, sim.sim_days):
288     pop.spread_infection(sim)
289     pop.update(sim)
290     pop.display_statistics(sim)
291     graphics(sim, pop, sim_canvas)
292
293     #Update the tkinter window to reflect the graphics change
294     sim_window.update()
295
296     #If we are currently not on the last day of the simulation, wipe the canvas
    clean
297     if i != sim.sim_days-1:
298         sim_canvas.delete('all')
```