

Agent-based modeling of the spread of a disease

Bioinformation Technology, CS-A1121

Casper Tillander, 907116, 20.4.2023

Introduction

This document provides a comprehensive overview of a disease simulation project, including a project description, user instructions, external libraries, program structure, algorithms, data structures, file handling, testing, and evaluation. It serves as a guide to understanding the project's development process, implementation, and functionality, as well as its strengths and weaknesses. The document also highlights changes from the original plan, the development timeline, self-assessment, references, and attachments, offering valuable insights into the project's success and areas for improvement.






Table of Contents

1. General description	3
2. Instructions for the user	4
3. External libraries.....	4
4. Structure of the program	4
5. Algorithms	5
5.1 Manhattan distance computation.....	5
5.2 Pathfinding	5
5.3 Artificial intelligence.....	6
5.4 Random attribute generation.....	6
5.5 Attribute-based susceptibility calculation.....	6
5.6 Disease status updates	6
5.7 Alternative solutions	6
6. Data structures	7
6.1 Dictionaries.....	7
6.2 Lists	7
6.3 Custom Classes	7
6.4 Integers, Floats, and Booleans	7
6.5 Objects.....	7
8.6 Alternative data structures.....	7
7. Files.....	8
8. Testing	8
9. Known shortcomings and flaws of the program	8
10. Three best and three worst areas	9
11. Changes to the original plan.....	9
12. Realized order and schedule	10
13. Assessment of the final result	10
14. References	12
15. Attachments	13

1. General description

This disease simulation project aims to provide a graphical representation of the spread of a disease within a confined population. The simulation displays the interactions between different types of individuals in a grid-based environment, providing a visual insight into how various factors influence the progression of a disease outbreak. This project fulfills, and in some areas, exceeds the requirements for a hard project.

There are five types of artificial intelligence (AI) in the simulation. SmartAI agents, when healthy, interacts with other healthy AI agents in the world. However, when they fall ill, they self-quarantine in the world's top left corner. AvoidingAI agents constantly tries to distance themselves from other AI agents to minimize infection risk, and they move toward the world's center whenever possible. Doctors actively seek out sick AI agents to provide medical treatment, and vaccinators identify susceptible individuals and administer vaccinations. The builder's objective is to divide the world in half by constructing an impassable wall, segregating the AI population. The AI agents also have individual attributes, such as age, gender and preexisting conditions which all affect the likelihood of them getting infected. The different AI agents are visualized by different shapes in the simulation, as can be seen in the table. The disease status is visualized by different colors: the susceptible AI agents are green, the infected AI agents in the incubation period are pink, the sick ones are red, and the recovered ones are blue.

Type of AI	Symbol
SmartAI	
AvoidingAI	
Doctor	
Vaccinator	
Builder	

The randomly generated attributes are the unique feature of this project. My initial unique feature, which aimed to make the simulation resemble a shop, had to be altered because it was not feasible to implement while maintaining a realistic disease simulation.

In the simulation, the disease can spread through direct contact or via airborne transmission within a range of one grid square in the grid-based environment. AI agents may be susceptible to the disease, infected, sick, recovered and immune, or deceased. Deceased AI agents are removed from the simulation.

The user has control over various aspects of the simulation, including the number of AI agents, disease parameters (infection probability, mortality rate, transmission mode, incubation period, and recovery time), among others. Upon completion, the program presents the user with relevant statistics and exports the data to a CSV file. The simulation ends when there are no more sick AI agents in the world or when all AI agents are dead.

2. Instructions for the user

Before running the program, ensure that the necessary dependencies are installed. PyQt6 is required for the program to function. To launch the program, run the main.py file. The program can be exited at any time without the program crashing by closing the current window (input, simulation, or statistics).

Upon launching the program, an input window will be displayed. This window allows users to set the desired parameters for the simulation. Enter the appropriate values for each category, such as the number of individuals in various roles (Smart Healthy, Avoiding Healthy, Smart Sick, Avoiding Sick, Doctors, Vaccinators, and Builders), the spread type, infection chance, death chance, incubation duration, recovery duration, and vaccination rate. After entering the desired parameters, click the submit button to initiate the simulation.

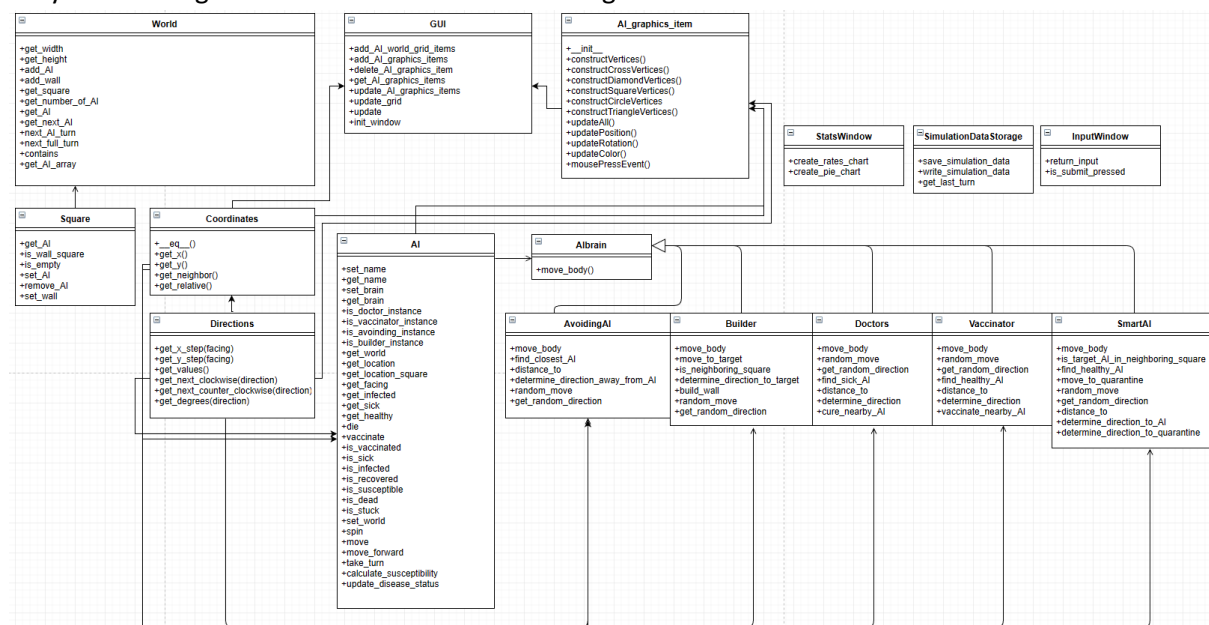
The simulation continues as long as there are sick AI agents in the world, and the user doesn't close the simulation window. When the simulation concludes, a separate window will display the statistics, providing insights into the disease's impact on the population and the individual attributes of the AI agents. The statistics will also be saved to the file simulation_data.csv for further analysis. An example of running the program can be found in the attachments.

3. External libraries

The only external library used in this program is PyQt6. In this project it is used for building the graphical user interface, including the simulation graphics, the input window, and the statistics window.

4. Structure of the program

The program is divided into seventeen different classes and a test file containing multiple classes used for testing the program. The structure of the program, including the methods of the classes and how they all work together can be seen in the UML diagram below.



The AI class serves as the core of the program, representing individual AI agents within the simulation. Each instance of the class embodies an AI agent's disease status, role (e.g. builder, doctor), and specific attributes. The AI agents' movements and roles are governed by the Albrain class, which acts as a superclass for specialized subclasses representing different agent types (doctors, vaccinators, builders, etc.). The World class defines the grid-based environment AI agents inhabit, while the Square, Coordinates, and Directions classes provide supplementary functionality to facilitate implementation. The graphical aspects of the simulation are managed by the GUI and AI_graphics_item classes. The AI_graphics_item class visually distinguishes agents based on role and disease status, while the GUI class is responsible for rendering the world and updating the grid to reflect changes, such as newly added walls.

The StatsWindow, SimulationDataStorage, and InputWindow classes operate independently from the main simulation classes and are invoked directly in the main function. These classes manage the display of input and statistics windows, as well as the writing of data to a CSV file. The test file consists of multiple classes, each with their own methods for testing the various components of the program.

```

Test
-TestDiseaseLogic
+create_test_environment
+test_infection_spread
+create_test_environment_2
+test_infection_spread_2
-TestDataStorage
+test_data_storage
-TestDoctors
+setUp
+test_find_sick_AI
+test_distance_to
+test_determine_direction
+test_move_body
+test_cure_nearby_AI
-TestAvoidingAI
+setUp
+test_find_closest_AI
+test_determine_direction_away_from_AI
+test_move_body
-TestSmartAI
+setUp
+test_find_healthy_AI
+test_move_body
+test_move_to_quarantine
+test_determine_direction_to_AI
+test_determine_direction_to_quarantine
+test_is_target_AI_in_neighboring_square
-TestVaccinator
+setUp
+test_move_body
+test_vaccinate_nearby_AI
-TestBuilder
+setUp
+test_move_body
+test_build_wall_after_1000_turns

```

The current class structure is effective and allows for easy expansion of the program. However, introducing a dedicated class to manage the disease could improve code organization, as the AI class is already quite lengthy, and would further facilitate future development.

5. Algorithms

A key part of the simulation is the movement of the AI agents. Several algorithms and mathematical computations are involved in this. Some algorithms are also used in the AI class, where they handle the updating of the disease status.

5.1 Manhattan distance computation

One of the essential algorithms used in the simulation is the Manhattan distance computation. This formula calculates the distance between two AI agents on the grid. By taking the absolute difference of their x and y coordinates, the Manhattan distance provides a useful estimate of the steps required to move from one agent to another. This calculation plays a crucial role in finding the closest sick or healthy AI agents for different AI roles, such as doctors or vaccinators. The Manhattan distance is computed as:

$$\text{Manhattan distance} = |x_1 - x_2| + |y_1 - y_2|$$

5.2 Pathfinding

The pathfinding process in the game relies on a simple greedy algorithm that chooses the direction in which an AI agent should move based on its distance to the target AI agent. Rather than meticulously evaluating every possible path, this algorithm prioritizes the direction with the larger difference

between the x or y coordinates. In essence, the AI agent moves primarily in the x direction if the target is further horizontally and primarily in the y direction if the target is further vertically.

5.3 Artificial intelligence

Artificial intelligence is at the core of the game, with AI agents designed to perform various tasks depending on their roles, such as curing, vaccinating, or avoiding infection. These tasks are executed using different algorithms that allow the AI agents to adapt their behavior intelligently according to the world's state.

The methods used in the different classes are in some cases similar but implemented in different ways to achieve the required task. For example, Doctors search for the closest sick AI, while Vaccinators look for the closest healthy AI. SmartAI agents use the same pathfinding algorithms to interact with other healthy AI, or to self-quarantine when sick. AvoidingAI agents calculate the Manhattan distance the same way as the other classes, but then use it to avoid the other AI agents.

5.4 Random attribute generation

When creating a new AI instance, it randomly assigns attributes like age, gender, smoker status, pre-existing conditions, and lifestyle. Each attribute is assigned by randomly choosing from a range or list of possible values.

5.5 Attribute-based susceptibility calculation

The `calculate_susceptibility()` function computes the susceptibility of an AI to infection based on its individual attributes. It multiplies the base susceptibility by various factors depending on the AI agent's age, gender, smoker status, pre-existing conditions, and lifestyle. This results in a susceptibility value that represents the AI agents vulnerability to infection and is then used in the `update_disease_status()` function.

5.6 Disease status updates

The `update_disease_status()` function handles the progression of the disease in the AI and manages its interactions with other AI agents based on their health status and the spread type (direct contact or within a distance of two squares). The function checks the AI agent's infection and recovery status and updates its state accordingly. It also handles the effects of incubation, recovery, death, and vaccination on infection chances. If an AI is susceptible, it calculates the probability of infection based on its susceptibility, vaccination status, and the health status of neighboring AI agents.

5.7 Alternative solutions

There are several alternative solutions that could have been implemented in the simulation, especially for pathfinding. More advanced pathfinding algorithms like A* or Dijkstra's algorithm could be used to find the shortest path between two AI agents. These algorithms consider the cost of traversing each cell in the world grid and find the optimal path to reach the destination. If one wanted to implement more walls into the world, a better pathfinding algorithm would be essential. However, the simple greedy algorithm was chosen in this case, as it is easier to implement and sufficient for the simulation's requirements.

New AI agents can easily be implemented in the simulation. However, more advanced AI agents would require more complex algorithms and decision-making strategies.

6. Data structures

In the simulation, the data structures used are integers, floats, booleans, dictionaries, lists, custom classes, and objects. These data structures are appropriate for the storage and handling of data required in the program because they allow efficient organization and access to the required information.

6.1 Dictionaries

Dictionaries are used to store counts of various attributes such as gender, smoker status, pre-existing conditions, and lifestyle. They are appropriate because they provide constant time access, insertion, and deletion operations, allowing for quick updates and lookups.

6.2 Lists

Lists are used in some parts of the code because they provide a simple way to store and iterate through a collection of items. The objects used in the simulation are also often stored in lists.

6.3 Custom Classes

Custom classes, such as the AI class, are used to encapsulate the properties and behaviors of the individuals in the simulation. This allows for an organized and modular approach to managing the AI agents and their interactions.

6.4 Integers, Floats, and Booleans

Integers, floats, and booleans are used to represent numerical and logical data in the simulation. They are used to store and update the age, income, infection status, and other attributes of the agents.

6.5 Objects

Objects, such as the AI class, are used to represent the individual agents in the simulation. They encapsulate the agents' properties and behaviors, allowing for easy manipulation and interaction between them. Other built-in Python objects such as strings, dates, and datetimes are also used to represent various attributes of the agents.

8.6 Alternative data structures

Other data structures that could have been considered are sets, tuples, and arrays. Sets could be used for storing unique elements, but in this case, dictionaries provide more functionality for the required tasks. Tuples could be used as an alternative to lists if immutability is desired, but lists are more suitable for the given code due to their flexibility. Arrays could be used for fixed-size collections, but lists provide more convenient dynamic resizing in Python. The data structures used in the code are mutable, such as dictionaries, lists, and objects. Mutable structures are suitable in this case because the simulation involves updating the state of agents, their attributes, and counts throughout the program's execution.

Python's predefined data structures like dictionaries and lists are used in the code, providing convenient and efficient ways to store and manipulate data without the need for creating custom data structures.

7. Files

The simulation saves its data to a CSV file after each run, with each turn written as a new row. The file format includes headers and simulation data such as turn number, counts of dead, infected, recovered, vaccinated, and cured individuals, total population, rates of mortality, infection, recovery, vaccination, and cure, counts of genders, smoker status, pre-existing conditions, and lifestyle. The program does not require any user-created settings files to function properly.

8. Testing

The simulation undergoes thorough testing using unittest. Multiple testing classes are utilized to test various aspects of the simulation. While not every part of the simulation requires testing, every function in each AI is tested to ensure its intended behavior. The disease logic is also tested as it can be difficult to confirm if the disease has transmitted to an AI in every square within a two-square distance. Most testing classes have the same structure: The testing world is established, one or more AI are placed within it, and the simulation is run several times to confirm if the AI behaves as expected, such as if the doctor finds and cures the infected AI. The SimulationDataStorage is also tested to confirm that it correctly writes values to the csv file. Supporting classes, like Coordinates and Directions, and user interface elements, such as InputWindow and StatsWindow, are not tested as they are fundamental to the program's functioning and are therefore easy to check without testing.

Although the program passes all written tests, test coverage is not complete for some classes and could be improved. Nonetheless, combined with repeated testing by running the simulation, the testing is sufficient to conclude that the simulation operates correctly.

Test
-TestDiseaseLogic
+create_test_environment
+test_infection_spread
+create_test_environment_2
+test_infection_spread_2
-TestDataStorage
+test_data_storage
-TestDoctors
+setUp
+test_find_sick_AI
+test_distance_to
+test_determine_direction
+test_move_body
+test_cure_nearby_AI
-TestAvoidingAI
+setUp
+test_find_closest_AI
+test_determine_direction_away_from_AI
+test_move_body
-TestSmartAI
+setUp
+test_find_healthy_AI
+test_move_body
+test_move_to_quarantine
+test_determine_direction_to_AI
+test_determine_direction_to_quarantine
+test_is_target_AI_in_neighboring_square
-TestVaccinator
+setUp
+test_move_body
+test_vaccinate_nearby_AI
-TestBuilder
+setUp
+test_move_body
+test_build_wall_after_1000_turns

9. Known shortcomings and flaws of the program

Although the simulation functions as intended, it has a few limitations and possible weaknesses that are worth mentioning. The main issue is the usage of a basic pathfinding algorithm, which occasionally result in AI agents getting trapped, especially in a world with a lot of AI agents or walls. To address this issue, a more sophisticated algorithm, such as A* or Dijkstra's, could be employed. However, implementing such an algorithm may have an impact on performance and will significantly increase the complexity of the program. The current pathfinding approach works adequately for the simulation's present complexity, but more intricate simulations will necessitate a more sophisticated pathfinding method.

Additionally, despite the comprehensive testing, there might still be undiscovered bugs or errors that could affect performance. Repeated running of the program with different inputs, as well as a more complete testing plan would ensure that the simulation works as intended.

Finally, there is room for improvement in the way the program keeps track of the disease status of all AI agents for statistics. Currently, the values are incremented by one each time the respective function is called. The problem occurs if, for instance, the `die()` function is called multiple times for the same AI. This issue could result in false statistics. Unfortunately, I have not discovered a better way to address this issue. The justification behind not spending significant time addressing the issue is that it seems to work sufficiently for most cases, and it doesn't affect the rest of the program.

10. Three best and three worst areas

The three worst areas of the simulation are addressed in more detail in the *Known shortcomings and flaws of the program* section. They are the lack of a sophisticated pathfinding algorithm, the lack of a complete testing plan and the way in which the statistics for the AI agent's disease status is kept.

The three best areas of the program are the creativity of the AI agents, the user's ability to affect the simulation, and the visually appealing graphics of the simulation.

The five unique AI agents in the simulation each have distinct movement logic and functions. Their different behaviors significantly impact how the simulation progresses, and the reliance on randomness is minimal. For instance, the doctor's movement is only random when there are no sick AI agents in the world. However, this scenario never occurs since the simulation ends when there are no sick AI agents left in the world. The simulation creates an engaging experience for the user since no two runs of the simulation are the same due to the randomized features of the AI agents.

The user has control of over 13 different aspects of the simulation, such as the number of AI agents, disease behavior, and disease parameters. With the diversity of the AI agents and the various user controls, the simulation is highly customizable, adding to the overall experience.

The visually appealing graphics is the strongest feature of the project. Each AI is represented by a unique shape, and each disease status assigned a specific color. Additionally, the statistics are presented using both pie charts and bar charts, exceeding the project requirements.

11. Changes to the original plan

Although the general idea of the project remained the same, some notable changes were made to improve it.

The most significant change was the unique feature. Initially, the simulation was meant to resemble a shop, but this was not practical to implement while maintaining a realistic disease simulation. Instead, the randomly generated attributes became the unique feature of the project.

Furthermore, more AI agents were implemented than originally planned, which led to changes in the program's class structure. The SimulationDataStorage class was added to enable the program to write statistics to a CSV file, and the disease logic was incorporated into the AI class rather than being in a separate Disease class. Additionally, minor changes were made to the testing plan since the original plan was deemed unrealistic.

12. Realized order and schedule

Some minor changes have been made to the schedule, but the general order of the implementation has remained the same.

Project schedule:

Week 8 (Feb 20 - Feb 26)

- Feb 24: Complete project plan
- Feb 25: Research and study PyQt6 graphical features (5 hours)

Week 9 (Feb 27 - Mar 5)

- Feb 27: Guidance meeting (11:00 - 11:15)
- Mar 1-5: Complete PyQt6 exercises (10 hours)

Week 10 (Mar 6 - Mar 12)

- Mar 6-12: Develop basic simulation using RobotWorld, improve simulation, and experiment with different AI and disease logic (10 hours)

Week 11 (Mar 13 - Mar 19)

- Mar 13-19: Build user input page for parameter changes and simulation launch (10 hours)

Week 12 (Mar 20 - Mar 26)

- Mar 24: Deadline 1 - Commit some code to Git.
- Mar 20-26: Refine simulation, add the window that displays the statistics (10 hours)

Week 13 (Mar 27 - Apr 2)

- Mar 27 - Apr 2: Complete all remaining code, implement the data storage and the testing of the program (15 hours)

Week 14 (Apr 3 - Apr 9)

- Apr 7: Deadline 2 - Commit some tests to Git (done in advance)
- Apr 5: Write the technical document for the project. (15h)

May 12, 2023, 14:00: Final deadline for the project.

13. Assessment of the final result

Overall, the program successfully satisfies the project's criteria. The program's strengths include the creative AI agents and their movement logic, and the ability for users to tailor almost all the simulation parameters, making it highly adaptable. The simulation's graphical component also contributes to its visual appeal and comprehensibility.

A limitation of the program is the pathfinding algorithm employed, which occasionally causes AI characters to become trapped. While integrating a more advanced pathfinding algorithm could resolve this issue, it might result in slower performance and increased complexity. Another potential concern is the method of tracking disease status for all AI agents for statistical purposes, which might produce

inaccurate statistics if a function is called repeatedly for the same AI. This factor should be carefully considered when expanding the project.

For future enhancements, one possibility is to incorporate more advanced AI behavior to create a more realistic simulation. The disease logic could also be expanded, further improving the simulation, and different types of diseases could be added to the same simulation. The program's class structure facilitates easy expansion and improvement of the simulation.

Although the program's structure is conducive to modifications and expansion, adopting a more modular code could further improve the ability to expand the program. An example of this would be implementing a separate disease class. Data structures employed in the program, such as lists and dictionaries, are appropriate for its needs and integrate well with the existing algorithms. Additionally, the well-commented code offers a clear understanding of its functions, paving the way for ongoing improvements.

In terms of self-evaluation, I feel that the final project fulfills and, in some areas, surpasses the project's requirements. Nonetheless, there is always room for improvement, and I could have dedicated more time to certain aspects of the project, such as testing and incorporating a more advanced pathfinding algorithm.

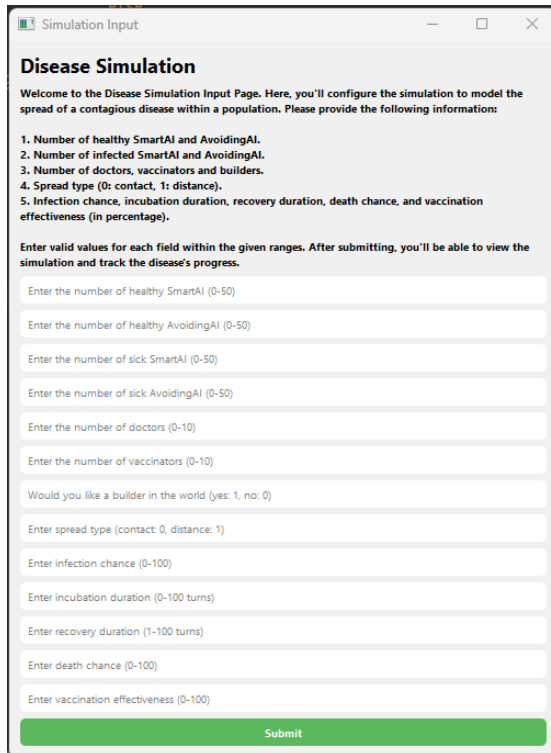
14. References

- [1] Fitzpatrick, M. (2023) *PyQt6 tutorial 2023, create python guis with Qt, Python GUIs*. Python GUIs. Available at: <https://www.pythonguis.com/pyqt6-tutorial/> (Accessed: April 5, 2023).
- [2] Foundation, O.G. (2021) *Manhattan distance [explained], OpenGenus IQ: Computing Expertise & Legacy*. OpenGenus IQ: Computing Expertise & Legacy. Available at: <https://iq.opengenus.org/manhattan-distance/> (Accessed: April 5, 2023).
- [3] Jamgade, P.S. (no date) *Path finding algorithms, OpenGenus IQ: Computing Expertise & Legacy*. OpenGenus IQ: Computing Expertise & Legacy. Available at: <https://iq.opengenus.org/path-finding-algorithms/> (Accessed: April 5, 2023).
- [4] *Reference guide¶* (no date) *Reference Guide - PyQt Documentation v6.4.1*. Available at: <https://www.riverbankcomputing.com/static/Docs/PyQt6/> (Accessed: April 5, 2023).
- [5] *PySide6.QtCharts#* (no date) *PySide6.QtCharts - Qt for Python*. Available at: <https://doc.qt.io/qtforpython/PySide6/QtCharts/index.html> (Accessed: April 5, 2023).
- [6] *CSV - csv file reading and writing* (no date) *Python documentation*. Available at: <https://docs.python.org/3/library/csv.html> (Accessed: April 5, 2023).
- [7] *Unittest.mock - mock object library* (no date) *Python documentation*. Available at: <https://docs.python.org/3/library/unittest.mock.html> (Accessed: April 5, 2023).
- [8] user1632861user1632861 et al. (1959) *Meaning of @classmethod and @staticmethod for beginner, Stack Overflow*. Available at: <https://stackoverflow.com/questions/12179271/meaning-of-classmethod-and-staticmethod-for-beginner> (Accessed: April 6, 2023).

15. Attachments

Example of running the program.

1. Run the main.py file.



Disease Simulation

Welcome to the Disease Simulation Input Page. Here, you'll configure the simulation to model the spread of a contagious disease within a population. Please provide the following information:

1. Number of healthy SmartAI and AvoidingAI.
2. Number of infected SmartAI and AvoidingAI.
3. Number of doctors, vaccinators and builders.
4. Spread type (0: contact, 1: distance).
5. Infection chance, incubation duration, recovery duration, death chance, and vaccination effectiveness (in percentage).

Enter valid values for each field within the given ranges. After submitting, you'll be able to view the simulation and track the disease's progress.

Enter the number of healthy SmartAI (0-50)

Enter the number of healthy AvoidingAI (0-50)

Enter the number of sick SmartAI (0-50)

Enter the number of sick AvoidingAI (0-50)

Enter the number of doctors (0-10)

Enter the number of vaccinators (0-10)

Would you like a builder in the world (yes: 1, no: 0)

Enter spread type (contact: 0, distance: 1)

Enter infection chance (0-100)

Enter incubation duration (0-100 turns)

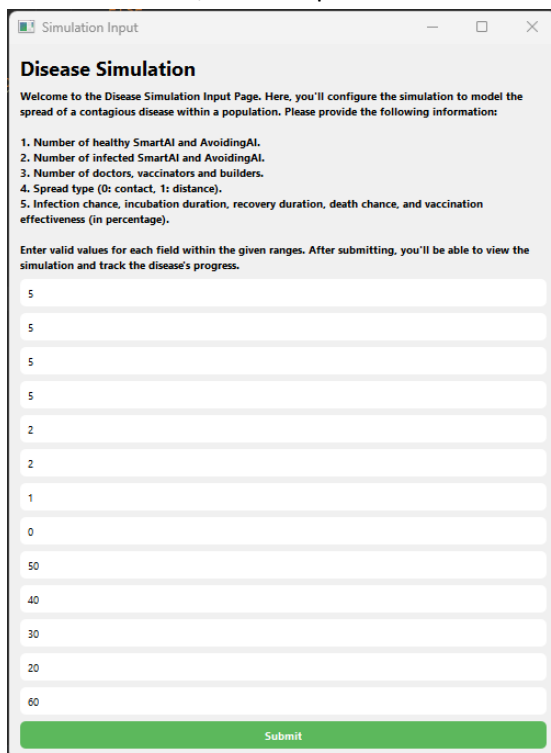
Enter recovery duration (1-100 turns)

Enter death chance (0-100)

Enter vaccination effectiveness (0-100)

Submit

2. Fill in the values, for example:



Disease Simulation

Welcome to the Disease Simulation Input Page. Here, you'll configure the simulation to model the spread of a contagious disease within a population. Please provide the following information:

1. Number of healthy SmartAI and AvoidingAI.
2. Number of infected SmartAI and AvoidingAI.
3. Number of doctors, vaccinators and builders.
4. Spread type (0: contact, 1: distance).
5. Infection chance, incubation duration, recovery duration, death chance, and vaccination effectiveness (in percentage).

Enter valid values for each field within the given ranges. After submitting, you'll be able to view the simulation and track the disease's progress.

5

5

5

5

2

2

1

0

50

40

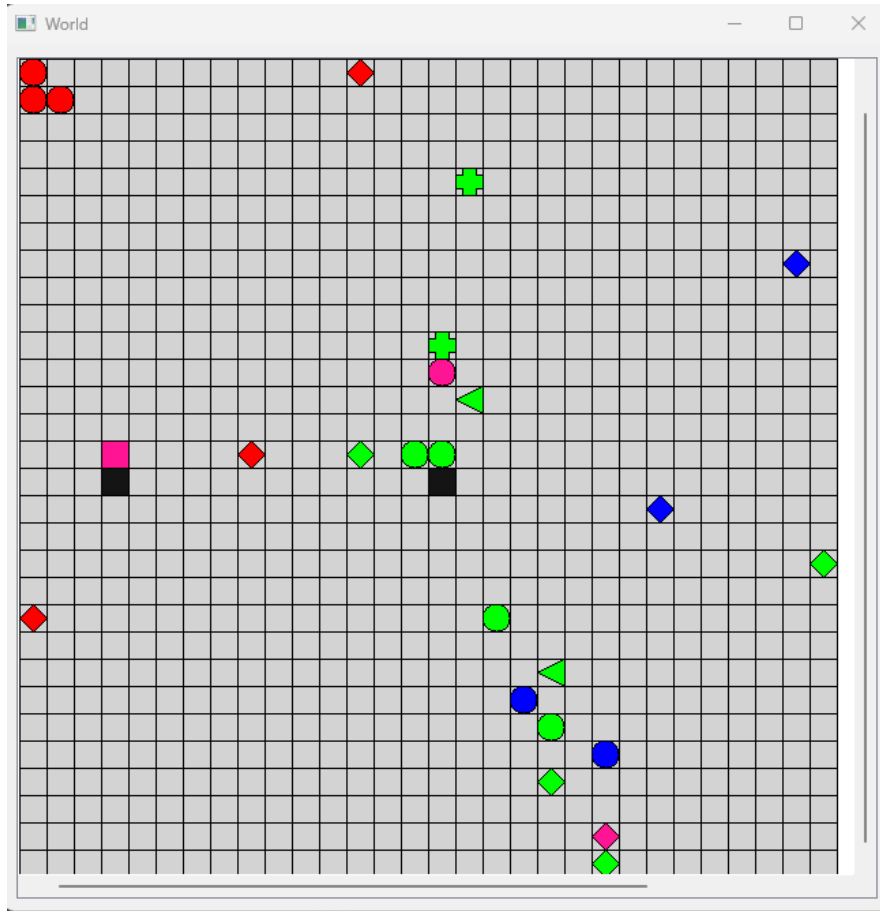
30

20

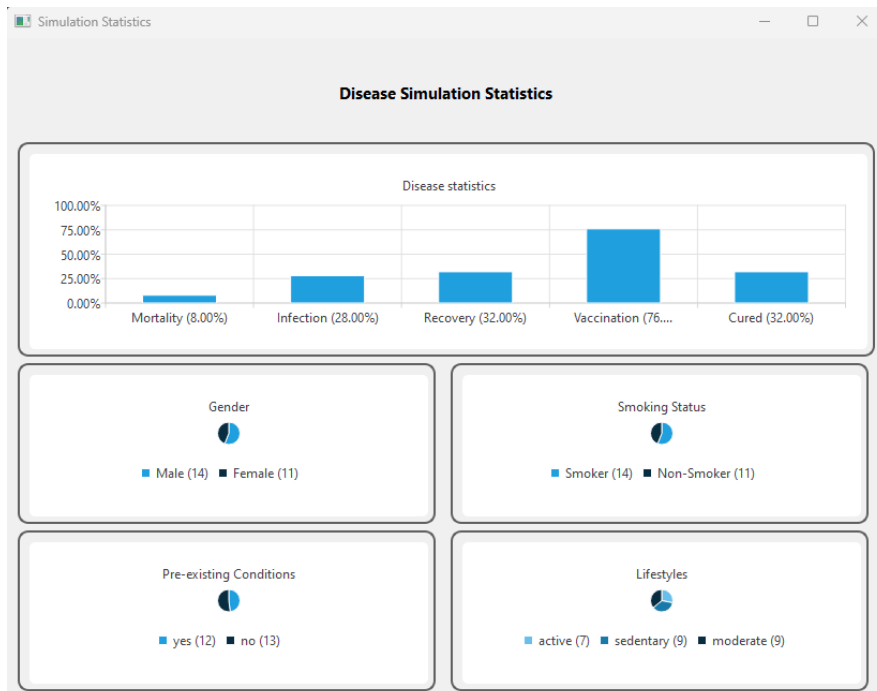
60

Submit

3. Press submit and watch the simulation.



4. After the simulation has ended, watch the statistics.



5. The statistics window can be closed at any time. The simulation statistics have also been saved to the csv file simulation_data.csv, along with statistics from previous runs.

Turn	Dead	Infected	Recovered	Vaccinated	Cured	Total_Population	Mortality_Rate	Infection_Rate	Recovery_Rate	Vaccination_Rate	Cured_Rate	Male	Female	Smoker	Non-Smoker
1	163	505	0	22	57	220	0.74090909	1.57727272	0.0	0.1	0.25909090	119	101	112	108
2	3	12	20	7	0	46	0.06521739	0.54621739	0.15217391	0.13043478	0.09090909	22	24	23	23
3	0	1	14	1	1	59	0.0	0.57627118	0.01694915	0.01694915	0.01694915	32	27	26	33
4	0	0	1	0	1	0	0.0	0.0	0.33333333	0.0	0.10000000	3	3	2	4
5	0	0	11	23	10	60	0.0	0.0	0.18535533	0.38333333	0.10000000	32	28	35	25
6	0	0	1	0	1	0	0.0	0.0	0.33333333	0.33333333	0.00000000	3	3	1	5
7	0	0	1	0	1	0	0.0	0.0	0.33333333	0.0	0.10000000	5	1	3	3
8	28	20	0	18	13	42	0.66666666	0.47619047	0.0	0.23809523	0.09523809	18	24	24	18
9	0	0	1	0	1	0	0.0	0.0	0.33333333	0.0	0.10000000	2	4	2	4
10	0	0	1	0	1	0	0.0	0.0	0.33333333	0.0	0.10000000	5	1	2	4
11	0	0	1	0	1	0	0.0	0.0	0.33333333	0.0	0.10000000	4	2	2	4
12	0	0	1	4	1	0	0.0	0.0	0.33333333	0.60000000	0.00000000	4	2	4	2
13	0	0	1	3	1	0	0.0	0.0	0.33333333	0.5	0.10000000	5	1	4	2
14	0	0	1	4	1	0	0.0	0.0	0.33333333	0.60000000	0.00000000	4	2	4	2
15	0	0	1	5	1	0	0.0	0.0	0.33333333	0.83333333	0.10000000	5	1	3	3

6. Run the simulation again with different values.