COMP 512: Multiagent Systems Course
Building an agent for the board game *Pandemic*

Persons Responsible: Antonios Vogiatzis, Errikos Streviniotis, Georgios Chalkiadakis

November 2020

*******Deadline: 5 January 2021, 23:55*******
Note: there will be no deadline extension granted.

1 Introduction

Pandemic is a cooperative board game, where the players have to work with each other in order to save the world from 4 fast-spreading diseases. The game was designed by *Matt Leacock* and published by *Z-Man Games* in 2008; and it consists a role-playing game with its primary goal being the players' cooperation rather than creating competition among them.

In this term project, we provide the students with a game platform for the Pandemic game (implemented at ECE's InteLLigence Lab by Athina Georgara, Antonis Vogiatzis, and Konstantinos Voloudakis). The ultimate goal is to allow autonomous rational agents to play the game, using the already built functional framework provided; and to exploit the framework to also evaluate the agents' performance. In other words, we provide a framework which anyone interested on building agents for this cooperative game can use, and can also exploit in order to test his/her agent's strategy.

2 Gameplay

The goal of the game is for the players to collaborate with each other to treat four (4) diseases, and ultimately eliminate them before a *pandemic* occurs. That is, each player is to form an actions' plan, i.e., a strategy/policy, taking into consideration his/her role and current state, along with the partially observed role and current state of her/his co-players. In more details, a different role is assigned to each player randomly; this different role entitles the player with some *special ability*, i.e. with some extra actions that only the player with this role can perform.

To begin, there is a board like the one illustrated in Figure 1. The board depicts a world map, and points out the 48 cities of interest in the game. Each city is has two attributes: *(a)* a disease type, and *(b)* the neighbouring cities. The former attribute is characterised by a colour (blue, yellow, red, and black), and essentially indicates which one of the 4 diseases can infect this specific city. Each city can be infected only by the disease that characterises it (unless an outbreak occurs, see Rules below), and the *level of infection* in this city is defined by the number of *cubes* lying on it on the board. The second attribute is a list of cities, which are connected to this specific city, depicted by white lines on the board. Moreover, a city may have a research center; initially there is only one research center in Atlanta, which coincides with the starting point of the game.

There are two types of cards placed on the board, the *Infection Deck* and the *Player Deck*.

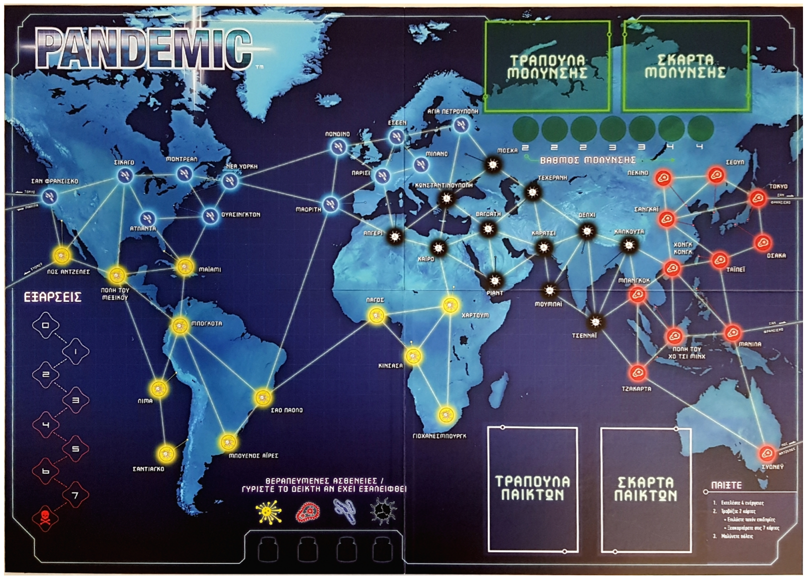

Figure 1: Pandemic board

- *Infection Deck*: there are 48 infection cards, one per city. The infection cards indicate in which city a 'disease spread' (a contamination) should happen, i.e., in which city we should add cubes.
- *Player Deck*: there are 52 player cards, which contain 48 city-cards, and 4 epidemic-cards:
  - city-card: a city-card is related to a city on the board;
  - epidemic-card: an epidemic-card starts an epidemic. If an epidemic card is played the following happen:
    (a) move the infection rate marker forward 1 space on the Infection Rate Track;
    (b) draw the bottom card from the infection deck. Unless its disease colour has been eradicated, put 3 disease cubes of that colour on the named city; if the city already has cubes of this colour, do not add 3 cubes to it. Instead, add just enough cubes so that it has 3 cubes of this colour and then an *outbreak* of this disease occurs in the city. Discard this card to the Infection Discard Pile;
    (c) Reshuffle just the cards in the Infection Discard Pile and place them on top of the Infection Deck.

Each player draws 2 player cards and $2-4$ (infection rate) infection cards during his/her turn. Now let us first present the universal actions (i.e., the actions that any player can perform), and then proceed on describing the roles and their special abilities.

2.1 Actions

There are seven (7) universal (common to all players) actions, as enlisted below:

- **Drive/Ferry**: Move to a city connected by a white line to the one you are in.

- **Direct Flight**: Discard a city-card to move to the city named on the card.
- **Charter Flight**: Discard the city-card that matches the city you are in to move to any city
- **Shuttle Flight**: Move from a city with a research station to any other city that has a research station.
- **Build a Research Station**: Discard the city-card that matches the city you are in to place a research station there. Take the research station from the pile next to the board. If all 6 research stations have been built, take a research station from anywhere on the board.
- **Treat disease** : Remove 1 disease cube from the city you are in, placing it in the cube supply next to the board. If this disease colour has been cured (see Discover a Cure below), remove all cubes of that colour from the city you are in.
  If the last cube of a cured disease is removed from the board, this disease is eradicated.
- **Discover a Cure**: At any research station, discard 4 City cards of the same colour from your hand to cure the disease of that colour. Move the disease's cure marker to its Cure Indicator.

2.2 Roles

Each player has a role with special ability to improve your team's chances.

- **MEDIC**: The Medic removes all cubes, not only 1, of the same colour when doing the Treat Disease action. If a disease has been cured, he automatically removes all cubes of that colour from a city, simply by entering it or being there. This does not take an action.
  The Medic's automatic removal of cubes can occur on other players' turns, if he is moved by the Dispatcher or the Airlift Event.
- **OPERATIONS EXPERT** : The Operations Expert may, as an action, either
  - build a research station in his current city without discarding (or using) a City card
  - once per turn, move from a research station to any city by discarding any City card.
- **QUARANTINE SPECIALIST**: The Quarantine Specialist prevents both outbreaks and the placement of disease cubes in the city she is in and all cities connected to that city. She does not affect cubes placed during setup.
- **SCIENTIST** : The Scientist needs only 3 (not 4) City cards of the same disease colour to Discover a Cure for that disease.

2.3 Rules

The game requires 2 to 4 players. Each player is assigned, randomly, to one of the roles described above. Each player use the pawn indicated by his/her role, and all players are placed in the city of Atlanta, which is the starting point of the game. Each player draws k player-cards and holds them (cards in hand); the number k depends on the number of players as shown below:

Number of Players	Initial Number of Cards in Hand
2	4
3	3
4	2

A research center is place in Atlanta city. The *outbreak marker* is set to zero, and the four *cure markers* are set with the vial side up. The *infection rate marker* is set to 2.

3 infection cards are initially drawn from the Infection Deck. The cities indicated by the infection cards are infected with the corresponding disease, and their initial infection rate is 3 (there are 3 cubes on each infected city). These infection cards are placed in the Infection Discard Pile. 3 more infection cards are drawn from the Infection Deck, and the cities' infection rate is 2 (there are 2 cubes on each of these infected cities). These infection cards are placed in the Infection Discard Pile. Lastly, we draw 3 more infection cards are drawn from the Infection Deck, and the cities' infection rate is 1 (there is 1 cube on each of these infected cities). These infection cards are placed in the Infection Discard Pile.

We can regulate the game's difficulty level, by using different numbers of epidemic cards.

Difficulty Level	Number of Epidemic Cards
Introductory	4
Standard	5
Heroic	6

Each game turn is composed by 3 phases, that is each player in a game turn follows these phases:

1. Perform 4 actions. An action can be either a universal one, or the action described by the player's role (i.e., the player's special ability). A player can perform the same action more than once within a turn.

2. The player draws 2 player cards. If the player draws an epidemic-card, this card should immediately do the following steps:

   - Increase the infection rate (move the infection rate marker forward 1 space on the Infection Rate Track.
   - Draw the bottom card from the Infection Deck. In case the disease that corresponds to the city has not been eradicated, we infect this city with infection level 3. If the city has an infection level $0 < k \leq 3$, increase the infection level so that it reaches level 3 (after the infection there are 3 cube on this city). The infection card is placed on the Infection Discard Pile.
   - Shuffle the cards of the Infection Discard Pile, and place them on top of the Infection Deck.

   If the player has more than 7 player cards in hand, the player must discard the extra cards (i.e., discard as many cards as needed in order for the player to have 7 cards in hand).

3. Draw as many infection cards from the top of the Infection Deck as the current infection rate. These cards are placed on the Infection Discard Pile.

Outbreaks: When a city with infection level 3 must be infected, an outbreak occurs. That is, we move the outbreak marker forward 1 space on the Outbreak Track. Therefore, all neighbour cities are infected **by the city's occurring the outbreak disease** (i.e., increase the disease infection level by 1). To clarify this, consider an outbreak occurs in Algiers (black disease), which is neighbour to Cairo (black disease), Istanbul (black disease), Madrid (blue disease), and Paris (blue disease); as such, Madrid and Paris are infected by the black disease, even though their primary disease is the blue one.

If a neighbouring city has an infection level 3 this causes a chain reaction outbreak. When a chain reaction outbreak occurs, the cities where an an outbreak has already occurred, are not included in the neighbour cities to be infected.

Game End: There are 4 final states of the game.

- As soon as all four Disease-Cures are discovered, the game ends, the players **win** the game.
- If the outbreak marker reaches the last space on the the Outbreak Track, that is, when 8 outbreaks have occurred the game ends, and the players **lose** the game.
- If we are unable to place the required number of disease cubes on the board, the game ends, and the players **lose** the game.
- If the player cannot draw 2 player-cards on his/her turn after performing his/her 4 actions, the game ends, and the players **lose** the game.

3 Game Implementation

Having discussed the game play, now we will present our implementation. For our implementation we worked with the Java language. The following discussion pertains to the server.package, which includes the City.class, Board.class and Server.class. For further discussion regarding the packages of this project, see section 6.

3.1 Cities

We created a class named *City*, and 48 instances of this initialize in the *Board* class, one per each city of interest in our game. The City class contains the following:

- Variables

```
String name;
String colour;
int neighboursNumber;
String[] neighbourCities;
int redCubes;
int blueCubes;
int blackCubes;
int yellowCubes;
boolean hasReseachStation;
```

- Methods

```
public getNeighbors(index)
```

returns a specific neighbour city.

```
int getMaxCube()
```

returns the highest count of a cube of any colour.

```
int getMaxCubeColor()
```

returns the colour of highest count of a cube.

```
int getCubes(String colour)
```

returns the number of cubes of a specific colour from the city.

```
void removeCube(String cubeColour)
```

removes a cube of colour 'cubeColour' from the city.

```
public addCube(String cubeColour, Player quarantinespecialist)
```

adds a cube of colour 'cubeColour' to the city if the current location of Quarantine Specialist is not the city. After adding the cube, the method checks for outbreaks and return *true*, if an outbreak occurs, and *false* otherwise.

```
public boolean checkOutBreaks()
```

checks if an outbreak is to occur, if yes the method adds the necessary cubes to the neighbour cities. This method, recursively performs chain reaction outbreaks, if they occur.

All getters and setters methods are also implemented.

Each one of the 48 cities of interest, is initialized in *Board* class.

3.2 Actions

We have implemented 7 out of the 8 overall global actions defined in the game. These actions are implemented as methods in the class *Board*:

- DriveCity:
  - Variables

    ```
    int playerID;
    String destination;
    ```

    Drive / Ferry to the desired city.
- DirectFlight:
  - Variables

    ```
    int playerID;
    String destination;
    ```

    Direct flight to desired city.

- CharterFlight:
  - Variables

    ```
    int playerID;
    String destination;
    ```

    Charter flight to desired city.
- ShuttleFlight:
  - Variables

    ```
    int playerID;
    String destination;
    ```

    Shuttle flight to desired city.
- BuildResearchStation:
  - Variables

    ```
    int playerID;
    String cityToBuild;
    ```

    Build Research Station to specific city.
- TreatDisease:
  - Variables

    ```
    int playerID;
    String cityToTreat;
    String color;
    ```

    Treat a disease.
- DiscoverCure:
  - Variables

    ```
    int playerID;
    String colorToCure;
    ```

    Cure disease action.

The global action ‘Share Knowledge’ was not implemented as it requires a communication protocol between the players.
All actions above implemented inside *Board* class in *server* package.

3.3 Roles and Special Abilities

We have implemented 4 out of the 7 roles defined in the game. These roles are:

- Medic
- Scientist
- Quarantine Specialist
- Operations Expert

Their special abilities are implemented as methods, and are the following:

- MedicSpecialAbility :
  The MEDIC special ability is implemented as a method in Board class updateMedicProtectionList(cubeColor);
- ScientistSpecialAbility:
  The SCIENTIST special ability is implemented as check condition (if statement) in the classic cure disease method check condition in method checkDiscoverCure();
- QuarantineSpecialistSpecialAbility:
  The QUARANTINE_SPECIALIST special ability included as a method in Board class updateQuarantineSpecialistProtectionList();
- OperationsExpertSpecialAbility:
  The OPERATIONS_EXPERT special ability is implemented as method in Board class operationsExpertTravel(int playerID, String destination, String cardToThrow);. Also the second special ability of this role is implemented as a check condition (if statement) in the buildRS(int, String) function.

The methods of the special abilities (except the Quarantine Specialist's and Scientist) implemented in the class Board.

3.4 Board class

The Board class consist of the following:

- Infection Deck
  For the Infection Cards we use two lists:

  ```
  ArrayList<String> infectDeck;
  ArrayList<String> discardedPile;
  ```

- Player Deck
  The Player Deck is also implemented with a list consists of 48 cities cards and 4 epidemic cards.

```
ArrayList<String> playerDeck;
```

and there is the same methods regarding the player deck as Infect Deck.

Infection Deck consists of 48 city cards which are used to contaminated the city network graph. Player Deck consists of 48 city cards and 4-6 Epidemic Cards which are used to provide cards for users per turn.

- Methods for aggregate statistics/info
  - printCitiesAndCubes()
    Printing all cities with current cubes in them
  - printRemainingCubesAndDiseaseStatus()
    Printing remaining cubes of each disease and cured / eradicated status
  - ArrayList< *String* > getHandOf(int playerID)
    Searches for the players and returns the hands of specific player
  - printRemainingCubesAndDiseaseStatus()
    Printing remaining cubes of each disease and cured / eradicated status

4 Potential Choices of Algorithms for Player implementation

- Minimax (and other tree pruning methods, alpha beta, Negamax, SSS*, etc.)
- Heuristic/Evaluation functions
- Monte Carlo Tree Search (MCTS)
- Markov Chain Monte Carlo (MCMC)
- ...

5 Requirements for the project

1. Implementing an agent for Pandemic Board Game
2. Opponent modelling
3. Evaluation of the agent's strategy
4. Evaluation of the report delivered by each group, taking into account the code of the executed agent
5. Run several games and export tables of analytic metrics such as
   (a) Run test games with 2-4 players.
   (b) Measure the win/lose ratio after many games (approximate 100 games) using lose reasons.

*****Implementing only a functional agent without some kind of opponent modelling analysis and/or without some way to actively combine and take into account the suggestions of other agents, will not provide a passing grade.*****

6 Server and Client packages

The *Server* package contains the following classes:

- **Server.java:** In this class we are defining the number of players and the port that the *Server* is listens to. We are creating the *Server*, the output and input streams for each *Client* and also a thread using the *Client's* socket. We are creating the main instance of the Board that we are going to use throughout the game. Also, in this class we are controlling the flow of the game (who is playing, when the game ends, etc). Last but not least, we are decoding the messages that each *Client* sends as the action or the suggestion for each round.
- **Board.java:** This class contains the whole information about the game board. It's used from both the *Server* and the *Clients*. The *Server* uses it as main board where everything happens and sends an instance of the game board to each *Client* after each turn. The *Clients* are using it as they want in order to predict or test their actions. Every change that happens in the board of a *Client* does not affects the board of the *Server*.
- **City.java:** The class of the City object that we are going to hold all the info about each city.

The *Client* package contains the following class:

- **Client.java:** This is going to be the class of your client. It should be used as a simple client to build your strategy logic. The Client is connecting to the *Server* and uses two separate threads to communicate with the *Server*. The write tread runs only when it's time for the client to send his action or his suggestion to the *Server*. The read thread is constantly running to renew the board with all the necessary info. A dummy player is also implemented in this class.
- **citiesWithDistancesObj.java**: Utility class which is used by clients in order to compute the distance between two cities.

7 Running the project

In this section we describe the ways to import, export and test your project.

*****You are allowed to modify the code in the Client class only.*****

7.1 Import the project

You will be given a .zip file. In order to import the project do the following steps:

1. Open up eclipse.
2. Select File.
3. Select Import.
4. Select Existing Projects into Workspace.
5. Select the project and click Finish.

*****You should not change anything in the Server package and in the way that the Client.class connects and communicates with the server. Feel free to take a look in the Server package for functions that are going to help you throughout the project.*****

7.2 Export your project

You should end up with a **.jar file** of your project. In order to do that you should perform the following steps:

1. Right click to your project.
2. Select Export.
3. Select Java.
4. Select Jar file.
5. Select the project you want to export and click Finish.

7.3 Running the game

Before exporting your project define all the game variables in the Server.class and Board.class. After the export, open up one cmd window for the *Server* and one more cmd window for each *Client* that is going to join the game.
Use the command **java -cp jarFileLocation PLH512.server.Server** to open up the server replacing "jarFileLocation" with the location of your .jar file. Use the command **java -cp jarFileLocation PLH512.client.Client** to open up each client replacing "jarFileLocation" with the location of your .jar file and the "clientClass" with the name of the class that your agent is implemented.

Example of commands for the *Server* and the *Clients* of the game:

- Server: java -cp `C:\Users\PLH512\Desktop\pandemic_V4.1.jar` PLH512.server.Server
- Client: java -cp `C:\Users\PLH512\Desktop\pandemic_V4.1.jar` PLH512.client.Client

8 How to achieve a high grade

In order for a team to score well in its evaluation, the report must cover the following elements:

1. Description of the main difficulties and challenges you encountered in designing and implementing your agent (given the difficulties and challenges your agent had to face).
2. Reference all the bibliographic information you took into account when developing the agent, and provide a detailed description of how the bibliographic references influenced the planning of your agent's strategy.
3. Provide a detailed, coherent, easy-to-understand description of your agent's strategy, preferably described in as a mathematical manner as possible.
4. Discuss whether the implementation choices you made are consistent with the theoretical concepts introduced in the course; they do not necessarily have to be and may well not be consistent, but, regardless, you must justify your relevant choices—and to show that you have taken into consideration basic relevant theoretical concepts, even if you chose to focus on using heuristics rather than theory.

Top grade (10/10) at the project corresponds to 40% of the course grade. The agent that will rank first in the tournament that will be held, will receive an additional 10% bonus (i.e. 0.4 points will be added to its designers' total final course grade). The agent ranking second receives a 5% bonus. A team can achieve the top project grade without ranking first or second in the tournament, while it is not necessarily true that a team receiving a bonus will be also receiving a high grade in the project.

9 Report Instructions

Each team must submit

1. An eight-page report.
2. zip-file with the code.
3. jar-file of your project.

Important note: Your project must be tested before its submission.
For any question please contact the teaching assistants at the following emails:
a) anvogiatzis@intelligence.tuc.gr b) estreviniotis@isc.tuc.gr

*******Deadline: 5 January 2021, 23:55*******
Note: there will be no deadline extension granted.