Alireza
Rezaei

Advanced Program Structures Project

Scotland Yard

This document has two parts, user manual and development manual for the game " Scotland Yard".

Fifth Semester-winter Semester 2019-2020
ITE102772
Alireza Rezaei

Table of Contents

1- User Manual

1-1 Requirements:

The City Domino game has less graphic, so it needs basic operating computer.

As, it is implemented in java, it is platform free and should be possible to play it using any desktop operating systems.

The computer should have at least 512 MB memory and a single processor with more than 2 GHz clock to perform it faster.

Dedicated graphic card/memory is not necessary, as it has less graphical features.

The game does not need any specific I/O devices to play it. It just needs ordinary mouse or keypad to play, but the screen resolution of the monitor should be at least 1500 (width) to 1000 (height) that the game fits in it.

1-2 Program installation/start:

The game is provided in JAR file format and it does need to be installed on the computer. User just needs to store the JAR file plus lib folder, which contains the GSON library to read the json file, into their fixed storage, opens it, and starts to play it.

1-3 Operating instructions:

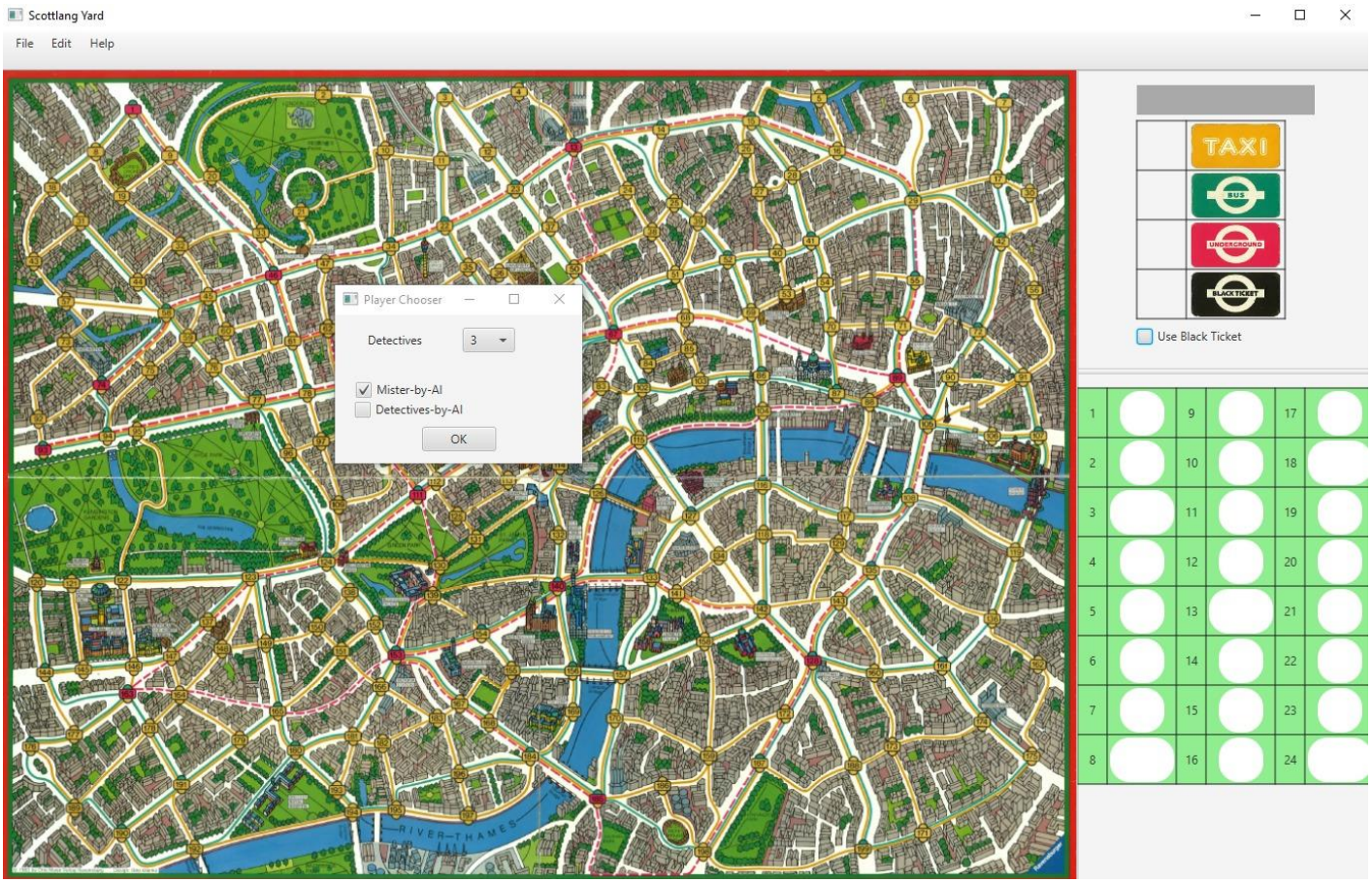

Figure 1: the main window of the game

GUI:

The GUI part of Scotland Yard game consists of the many components to control the game. these components can be seen as above.

The main map at the center is the map of London city. The city is connected with 199 stations. There different routes and means of transport from one station to another station. Furthermore, from some of the stations, player can use several means of transport to adjacent station. On the map taxi route is shown by yellow, bus route is presented by green color; train route is indicated by red, boat route by black lines.

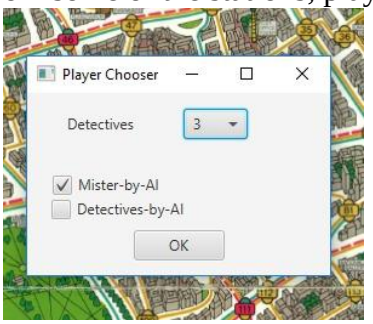

Figure 2: player chooser

When program starts and small window is displayed and waits until player select number of detectives and check/uncheck the

boxes to decide if the MisterX and/or Detectives are controlled by AI.

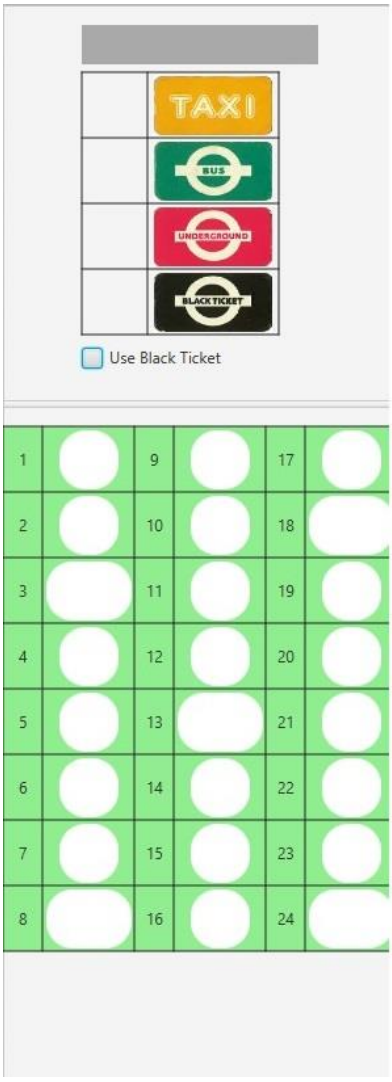

Figure 3: controls on the right side of the main window. It shows player's name, current player's ticket and journey board of MisterX

On the right side and at the top, there is a label that shows the player's name whose turn is to play. Then right below it there are boxes to show the amount of different tickets of each player while they are doing their turn. Then below the tickets, a check box exists and it is for the MisterX to select it if he/she wants to use black ticket instead of other means of transport.

At the bottom right, there are boxes to show journey of the MisterX. When MisterX plays, its used tickets are shown on this area. Some of the shapes in this area are bigger oval and they are that if the MisterX is controlled by AI and is hidden, it will be shown on the board.

At the top of the window there is a menu bar which contains some controls for the game.

Figure 4: menu bar

New game, save, load and close items can be seen at first menu (File). User can start a new game

Figure 7: File menu

Figure 5: Edit menu

Figure 6: help menu

Any time during the game, they also can save an already started game or continue to play a saved game using save and load controls. The close item in the File menu will close the game.

How to Play:

If you are MisterX, you must stay undercover to escape from your pursuers until the detectives can no longer move However if you are a detective, your purpose if to catch MisterX by moving onto the station where he is currently hiding

preparation:

-> One player becomes MisterX. The remaining players are detectives.

-> Share out the tickets;

MisterX gets 4 Taxi, 3 bus, 3 underground and as many black tickets as there are detectives. Each detective gets 10 taxis, 8 buses, and 4 underground.

-> MisterX and detectives are placed on random starting stations.

Playing the Game:

-> When MisterX is controlled by human the figur will be shown on the map. But when it is controlled by AI, its location will not be revealed until it reached to round 3, 8, 13, 18. Its used tickets are placed on travel log.

-> The detectives should work individually in choosing their individual means of transport. They then give up a ticket and move their game figures to the next stop. MisterX gets the used tickets.

End of the Game:

-> MisterX wins, if:

a) The detectives are no longer to move, or

b) MisterX reaches the last space in the travel log.

-> All detectives win, if one of them succeeds in moving onto MisterX station.

Black tickets:

MisterX can use black ticket in place of other tickets. The black tickets stand for any means of transport and it is indeed a black day for the detectives when one appears, for it prevents them from knowing what means of transport Mr.X has used. Even harder on the detectives is that Mr.X can use black ticket to travel with boat from point 194 to 157, from 157 to 115, from 115 to 108, or vice versa.)

Is more than on figure allowed to stand on one location?

-> No Two detectives can never stand on the same location.

Can player move back and forth?

-> Yes, it is possible to retrace your steps in subsequent move. Ofcourse you must hand in a ticket.

Are the detectives allowed to exchange tickets?

-> No, each detevtive can only use his\her own tickets.

What if a detective is unable to move?

-> The game goes on and other detectives try to catch MisterX. (III, 2000)

1-4 here are all error messages:

Figure 8: error during reading the json file. One of the station id is not ok.

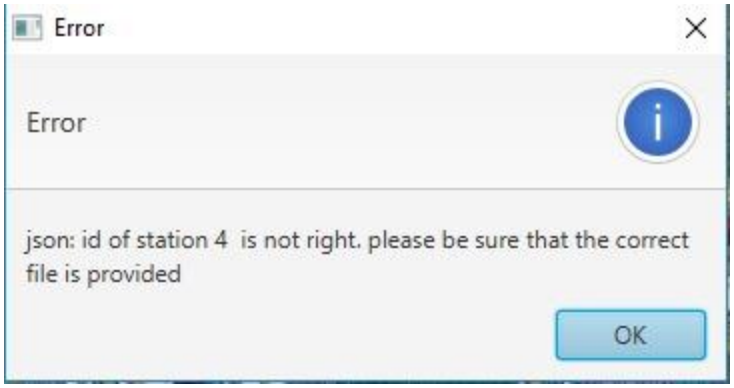

Figure 9: error during reading the json file. One of the stations id is not ok.

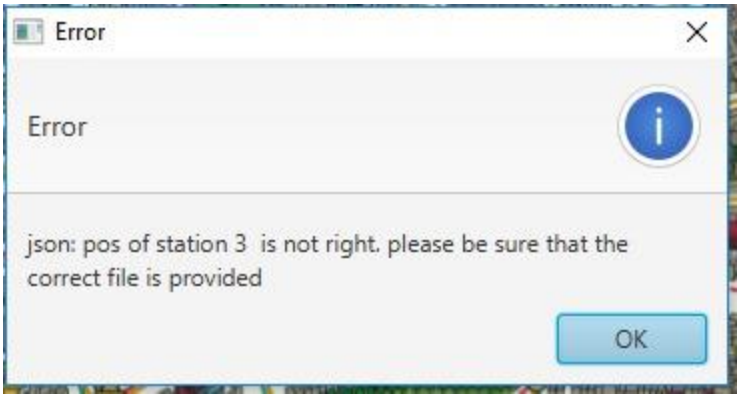

Figure 10: error during reading the json file. The position of one of the stations is not OK.

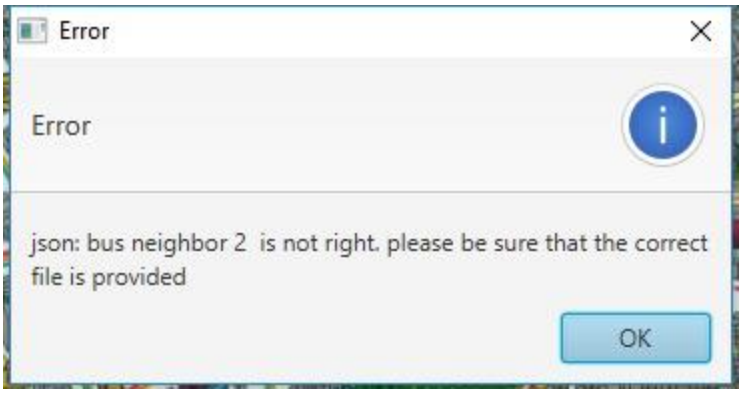

Figure 11: error during reading the json file. A neighbor id of one of the stations is not OK.

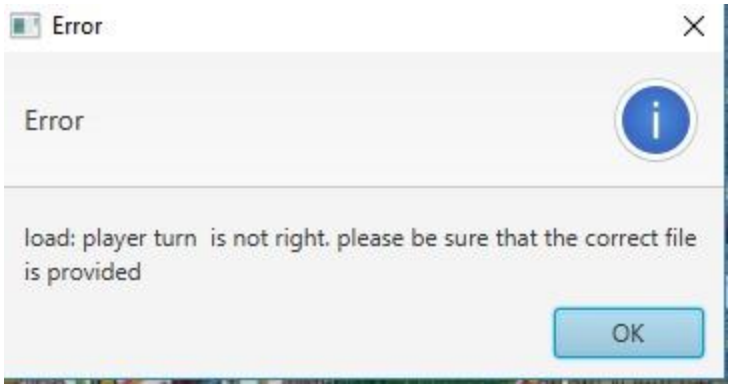

Figure 12: error during reading the saved file. Player turn is not OK.

Figure 13: error during reading the saved file. One of the detectives's position is not OK.

Figure 14: error during reading the saved file. Array of detectives is not OK.

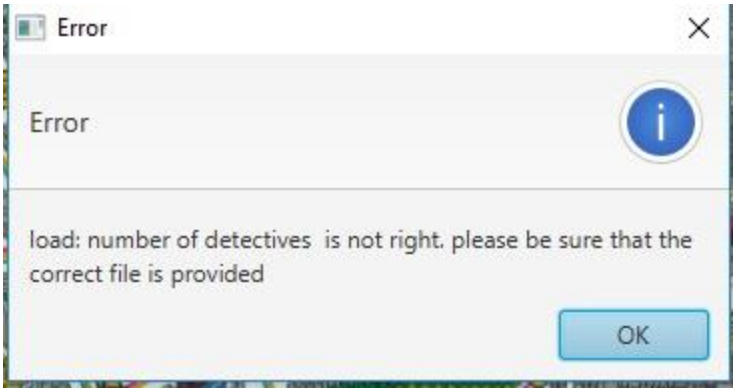

Figure 15: error during reading the saved file. Number of detectives is not OK.

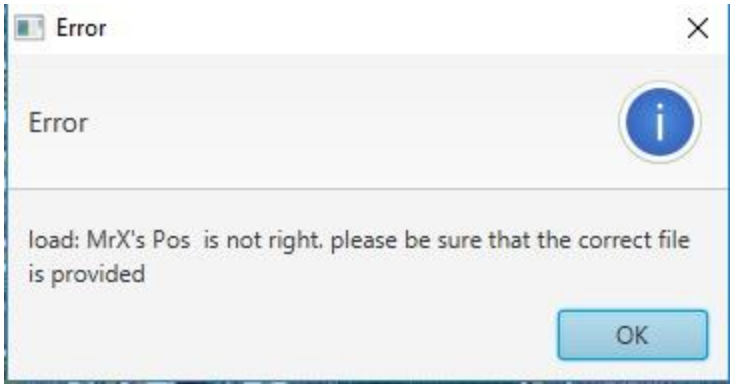

Figure 16: error during reading the saved file. MisterX's position is not OK.

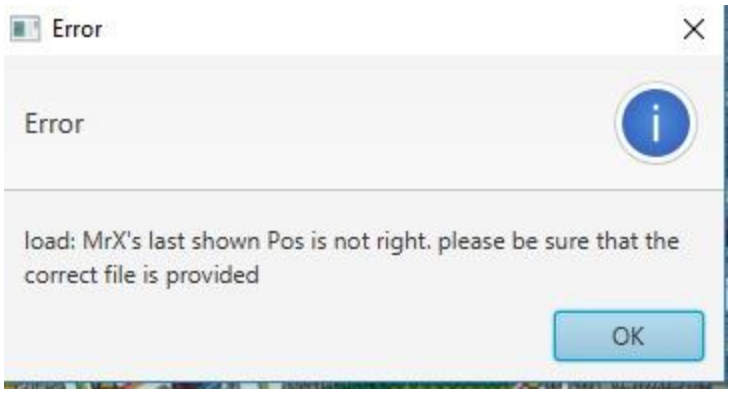

Figure 17: error during reading the saved file. MisterX's last shown position is not OK.

2-5-1 during the JSON parsing:

Error message	cause	Corrective action
One of the stations id not ok (Figure 8)	Identifiers should be from 1- 199 in the json file. If it has different number, it causes a problem.	The program is not starting will close automatically.
One of the stations id not ok (Figure 9)	Identifiers should be from 1- 199 in the json file. If it has different number, it causes a problem.	The program is not starting will close automatically.
One of the stations Position is not ok (Figure 10)	Positions should be double values between 00 to 1.0. if it has different value, it causes an error.	The program is not starting will close automatically.
One of the stations neighbor is not ok (Figure 11)	Same as identifiers, the neighbors of a station should be between 0 and 1.	The program is not starting will close automatically.

Table 1: json errors

2-5-2 during save/load:

Error message	cause	Corrective action
Player turn is not ok (Figure	Player's turn should from 0 to 1	The file is not loaded and user

12)	less than number of players.	can load correct file.
Detective position if not ok (Figure 13)	Position of the players should be from 1 to 199 station.	The file is not loaded and user can load correct file.
The length of array of detectives is not ok. (Figure 14)	The game can have 3 to 5 detectives. If less than 3 or more than 5 detectives are stored in the saved file, there will be an error.	The file is not loaded and user can load correct file.
Number of detectives is not ok (Figure 15)	It caused if the number of detectives stored in the file is not ok or the number of detectives is not the same as length of detectives array	The file is not loaded and user can load correct file.
MisterX position is not ok (Figure 16)	The position is a station on the map and should be from 1-199.	The file is not loaded and user can load correct file.
Mister X last shown position is not ok(Figure 17)	The position is a station on the map and should be from 1-199.	The file is not loaded and user can load correct file.

Table 2: load errors

2 Developer manual

2-1 Development configuration:

NetBeans IDE version 8.2 was used for program development and also java JDK was used for compiling and implementation. The code writing was done on a laptop with windows computer installed on it.

Junit version 6 was used for test of the program.

The project was saved into repository to access it from each computer at home or computers at lab of university.

For drawing the diagram of the classes and their connections, Microsoft Visio is used. It has more options to draw than Microsoft word.

Also as the map is provided as a json file including all stations and their attributes, GSON library was needed to parse and validated the json file.

2-2 Problem analysis and realization:

GUI:

The main board of the game is the map of London city and has the resolution from 1081(width) to 814(height). In the provided JSON file the stations' position is determined in scale of 0 to 1 (double value). When the user click on a point on the map it will be checked on which station player has clicked and its figure can be moved to that station. As the map of London is fixed for the game there is no other solution for the game board. Also the size of map is set big enough to read the stations number and see the connections between two stations. Consequently, the game may not be correctly seen on computer with a screen resolution less than 1500 to 1000.

On the right side, there are two grid panes for showing current player's tickets and MisterX journey log. Inside the grid panes are labels and inside each label, an image view component is added to show the picture. There is no better solution of using grid pane for this. For MisterX to use black ticket a checked box is added on right side. Another solution could be to hide this check box in a menu but as MisterX needs it at each round, it is better to put it somewhere that player can see and easily to use it. Moreover, as it is related to use of ticket, it is good idea to have it besides remaining tickets area.

Other things could be used differently were the controls inside the menus. For example for new game, save and load in File menu, it could be possible to use buttons on the main window but as they are not frequently used, it would make our main window messy with many item on it.

Also the cheat MisterX button in Edit item is better to be there as it is not frequently used like other items of menu. How To Play and About controls in help menu is useful for players to find useful information for how to play the game and what is needed to play the game.

Logic:

The station class was first made to have neighbors as integer values, but later it was changed to add other station objects as neighbor of a station. In this way access to methods of the station would be easier without changing to integer value to station object. Furthermore, station class

can find if a given station is neighbor to this station and also the given station has more route than one route to this station.

In player class, the tickets of the players were saved as separate integer values and getting them was not very easy. Then it was realized to save the number of tickets as a map of type and number. In this way, player can access to ticket easier especially using the type of the ticket.

Also each subclass of player has its own AI method to perform the turn when the player is controlled by AI. It would be possible to have it in game class but it would make the game class very long and complex to read. Players also store theirs current and previous station instead to store all of this information in game class.

The JSON parser class is reading information from a JSON file and validates it to store all stations information into a list of stations. The structural validation is being automatically done by GSON library and it needs only to have classes and like JSON file's structure. Then the logical validation is manually done by ErrorHandling class to check if data which provided in JSON file has no error. The parsing could be placed in IO class, but it would make the IO class long and hard to read. The JSON parsing in needed to read the files one time at the start of the new game. The other thing is that for reading from the JSON file, FileReader instance was used but later it was realized that if user make a jar file form the game the JSON file will be packed inside the jar file and FileReader cannot read from it. So with InputStream and InputStreamReader, the problem was solved.

IO class do three functions, save, load and log. To save into a JSON file, the data from game objects are provided to helper classes to make the structure and then using the GSON library the file can be made. Loading from a file is almost like JSON parsing procedure. So the GSON will validate the structure of the file and logical validation will be done using the ErrorHandling class. For save and load, there is no problem regarding to path as we store/read file in/from users directory. But the problem is with log as the user current directory should be found first and it should be added to file name.

Board class has only instance of the stations in a list and allow other classes to access to appropriate station from the list. For working on station it calls the proper methods from station class. One of the main functions that board is doing is to find suitable station which closest to clicked station. The check neighbor and more route methods from one station to another one

were implemented in board but they were moved to station class for better object oriented approach.

In ErrorHandling class all the logical errors for the JSON file and load file are checked. There are different possible ways to find and pass the errors. One way is to have separate classes for each error and pass them to GUI. This is readable by user and not easy to read by developers. Also it is possible to code them by integer values and it means each value represents an error. In program a small piece of string was used to pass the error to GUI, but the main message is in GUI.

AI:

AI strategies:

Both the detectives and Mister X can optionally be controlled by an AI. It should also be possible for the AI to control both "parties" in a game. The AI then determines the optimal next move and the detectives or Mister X move accordingly to the new positions. Of course, only moves to "neighboring" stations should be possible and only means of transport for which a ticket is still available can be used (the rules of the game must therefore be observed).

Figure 18: example of ai turn

In order to determine an optimal detective move, the following procedure should be implemented (making game situations comparable and comprehensible):

- the following different tactics are all played through one after the other (Attention: The examples with --> here always refer to exactly ONE tactic. In reality ALL of them should be tried and the best one should be taken!)
    - move to a possible target position (see below, circled light green in the picture on the right). So Mister X drove from the last pointing position at 116 out of 1x taxi).

If there are more than one available, choose the one with the smallest station number --> The blue detective would go to station 118.

- move to a directly adjacent subway station (to quickly get to other locations). If there are several accessible free stations, select the one with the lowest station number. --> The yellow detective would go to Station 185.
- move towards the last position where Mister X was shown (see below). Use the shortest (currently) free path for this. --> The red detective would go to station 70 (and following to 87, 86, 116), because this is the first one on the shortest way and it has a smaller number than the 89 (see below).
- move to the directly adjacent free position with the smallest station number (quasi as fallback, if nothing else works)

- After each attempted move, the new game situation ("detective moves to station x") is assessed using the evaluation function described below. The best rated move is carried out (if there are several moves with the same rating, the station with the smaller number is taken). The evaluation function consists of these single values and weights, which are added together. A larger value is better:
  - How many possible target positions can be reached by all detectives in the next turn divided by the total number of all possible target positions (for the detectives not yet drawn, their old stations should still be taken into account) Weighting: Number of achievable target positions by total number of target positions * 10 --> after the blue detective has moved to station 118, there are still 3 target positions left (104, 117, 127), but all of them cannot be reached --> 0 / 3 * 10 = 0
  - How far away is the " average possible target position " (see below) (so that the detectives who are further away from Mister X move roughly in his direction and don't run around stupidly in the area) Weighting: if 10 stations or more away = 0, otherwise (10 - number of stations on the shortest way to the " average possible target position ") --> After the blue detective moved to station 118, 3 target positions remained (104, 117, 127). Their coordinates are 765/341, 848/447 and 693/447, averaged 769/412. The closest station is 116, which is exactly 1 station away from 118. --> (10 - 1) = 9
  - How many stations can be reached from the current position with the existing tickets in one turn? Weighting: Number of stations divided by 13 (as these are the most different stations that can be reached at station 67) * 4 --> assuming the blue

detective still has 3 subway tickets, 4 bus tickets and 3 taxi tickets, he can reach 4 other stations (all by taxi) from station 118 --> 4 / 13 * 4 = 1,23

- What is the lowest number of tickets for a means of transport (if you don't have a ticket for one any more, this significantly restricts mobility) Weighting: If there are more than 2 tickets, the rating is 3, otherwise number of tickets. --> The lowest number of tickets at the blue detective (see above) is 3. --> = 3 --> the total rating would be 0 + 9 + 1,23 + 3 = 13,23

The AI for Mister X works in the same way, but uses a different tactic and another evaluation function. However, some functions of the detective AI can be reused here!

- Tactics of Mister X
  - move to a station that can be reached by as few players as possible on the next turn. For this purpose, Mister X simply tests all accessible stations.
- Evaluation function of Mister X
  - how many detectives can reach Mister X's position on the next turn? Weighting: (total number of detectives - number of detectives who can reach Mister X) * 10 Here you have to consider which tickets the detectives still have (Mister X also knows this in the real game). If necessary, a detective cannot reach Mister X on the direct neighboring station, because he lacks the matching ticket...
  - how many stations can Mister X reach from the current position with the available tickets in one turn, on which no detective is standing? Weighting: Number of stations divided by 13 (as these are the most different stations that can be reached at station 67) * 4
  - what is the lowest number of tickets for a means of transport (if you don't have a ticket for one any more, this significantly restricts mobility) Weighting: If there are still more than 2 tickets, the rating is 3, otherwise number of tickets. (Kan)

Shortest path finding:

One solution is to solve in O(VE) time using Bellman–Ford. If there are no negative weight cycles, then we can solve in O(E + VLogV) time using Dijkstra’s algorithm.

Since the graph is unweighted, we can solve this problem in O(V + E) time. The idea is to use a modified version of Breadth-first search in which we keep storing a given vertex in route while doing the breadth first search. (ab_gupta)
two stations, as source and destination, are provided and the source station is added to start of the route and it continuous to add to each child station to the separate route. Each route, which has been done and added, will be in visited list added to avoid cycle in the graph. In this approach all the routes to destination are created and shortest one is compared and is chosen. The route should be free of detectives. It good in term of process but as it makes all the routes, more memory is used.

Another approach is to have predecessor map and each current node is predecessor of the child until it reaches to the destination. When destination is found, it will traverse back to the source station to find the start of route. This has better memory usage as the paths are not made beforehand but the process needs one more step to make the route.

Furthermore, instead of the simply travel from A to B, a tree complete tree can be used by adding all the routes from parent to leafs. In this approach, the cycle is not concerned but it cost to make the tree and find all route and compare them to choose shortest one.

2-3 Program organization plan:

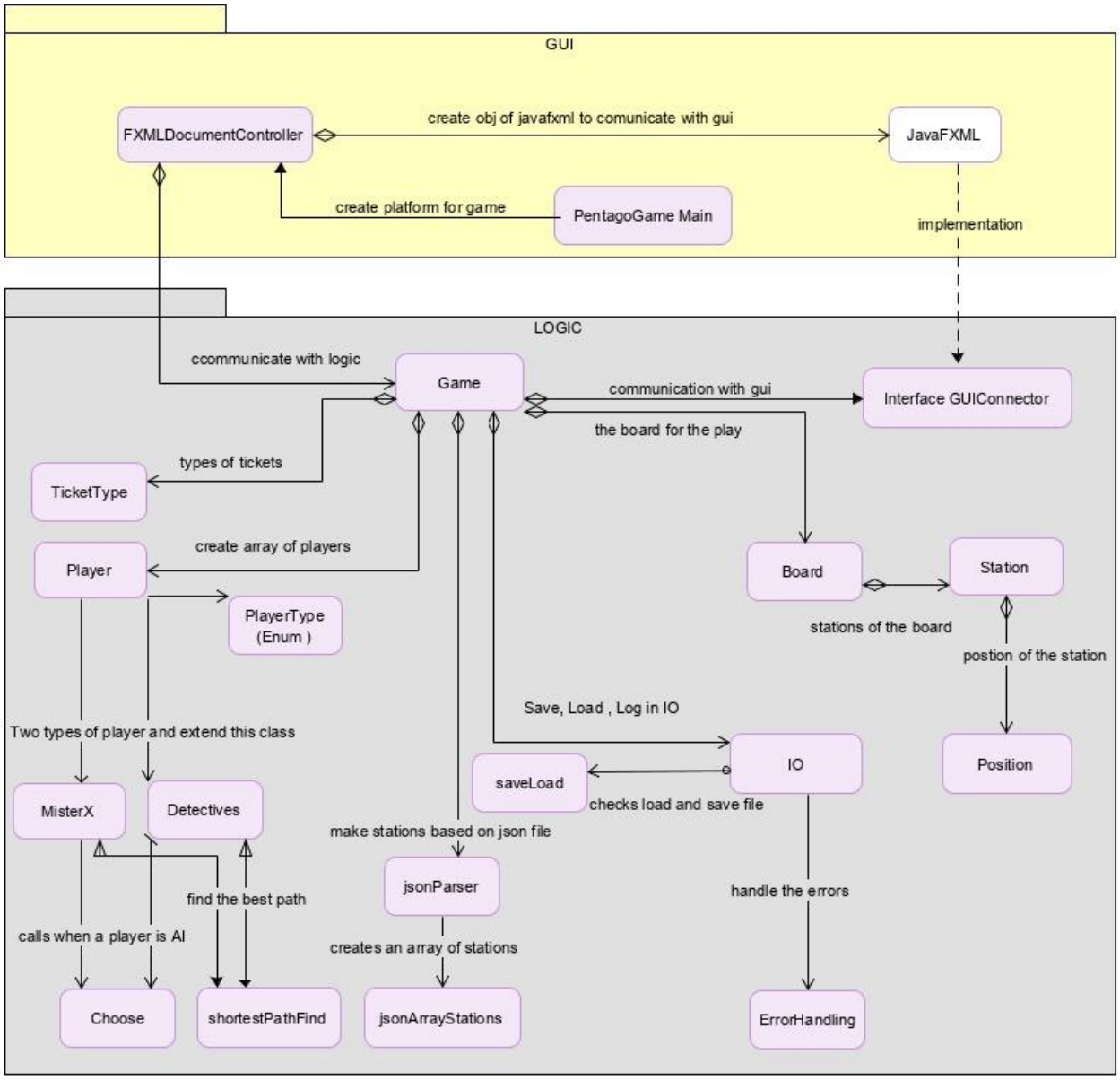

Figure 19: UML diagram of the program

2-4 Description of basic classes:

2-4-1 GUI

GameGUI class:

Game class in GUI package is the main starting point of the game. It creates the stage and loads the components which are made in FXMLDocument. Then it shows the graphics components to the user.

FXMLDocumentController:

In this class all the components of the gui were created and game will be initialized here and then it pass them to two objects of game in logic and JavaFXGUI. It also get users events and actions, and pass them to game in logic to update the logic and gui respectively.

JavaFXGUI:

This class is for updating any changes of the game in the logic package. An instance of JavaFXGUI is created in FXMLDocumentController and all necessary components are passed to JavaFXGUI constructor. When the changes happened in game of logic package, the updates are being sent to here via GUIConnector interface as JavaFXGUI class implements abstract methods of GUIConnector interface.

Colors class:

This is an Enum class containing the colors which are needed to present and circles of each player on the board.

2-4-2 Logic

Game class:

Game class is the contact point in logic and gui knows. Two main constructors in game are to make the main instance of the game or making a game from parameters which validated in from the load file. Also the game class controls all things in backend of the total game. it consider the turns and if the player is controlled by AI, it will send information to appropriate class and method to perform player turn.

Board:

Board class contains the graph of stations and it works between the game and the station class. The main functions that this class is doing are to check if two stations are neighbor, find the station close to clicked coordinated using Euclidean distance --> Pythagorean Theorem.

Station:

A station is a point on the map with id, coordinates, and different type of neighbors. In the station class, two neighbors can be checked if they are neighbors. All the neighbors (taxi, bus, train and boat) of a station can be accessed by a method in station class.

JSON parser class:

The Json parser class is checking if the JSON file is ok in term of structure and logic. The structure check will be done using jsonArrayStation class and JsonStation class which have the same structure (identifier, position, tube neighbors, bus neighbors, cab neighbors and boat neighbors) as the JSON file. After structural check, the logical check will be conducted to see if the content of the file is ok.

Players class:

Player class has attributes and methods which are needed for a player of the game. two other subclasses(MisterX and Detective) have inherited the general attributes of the player and beside that they have their own attributes and methods, especially AI method to find best move based on weighting of each neighbors of the player's position.

IO class:

IO class is for saving the current game, load a game from a saved file and log the game during play. For saving and loading another class named SaveLoad class is used to make save object and also perform structural check of the file for loading. Also the logical validation should be carried on to see if correct information is saved in the file. Furthermore logger method in this class is continuously writing a the moves of each player in a log file.

ShortestPathFind class:

This class is a used to find the shortest path between two stations and return the list of possible shortest path that there is no detective standing on it. The algorithm that is used here is the Dijkstra for path finding.

Choose class:

The instance of this class is used by AI method of each kind of players. It has three attributes which are weight, type of ticket and station. AI method stores best move as instance of Choose class based on calculation of the weight of different strategies.

ErrorHandling class:

The logical errors of the map JSON file and save file are controlled and handled here. It has two methods, jsonError method checks the logical errors of the JSON file and the loadError method controls the errors of saved file which supposed to be loaded. If an error is found, the methods show it appropriately on GUI and return false value to upper class otherwise everything is ok and true value is return for further action.

2-5 Program testing:

Action	expected	result
Click on a station	If the station is neighboring with the current station of the player nothing should be happened.	As expected
Click on a point	The point on which is clicked is far away from a station. Nothing should be happened.	As expected
Click on a station	Click on an adjacent station that player does not have ticket for it. Nothing should	As expected
Click on cheat MisterX	When MisterX is controlled by AI, it will hide / show MisterX figure.	As expected
Click on save	The proper save file should be stored	As expected
Click on load	Only proper saved file will be loaded	As expected
Check the black ticket box	MisterX can use black ticket instead of other means of transport	As expected
Click on new game	If everything is ok. New game will be started	As expected

Table 3: possible tests

Works Cited

ab_gupta. (n.d.). *www.geeksforgeeks.org*. Retrieved Feb 28, 2020, from https://www.geeksforgeeks.org/shortest-path-unweighted-graph/

III, P. T. (2000). *Revensburger*. Retrieved Feb 25, 2020, from www.ravensburger.de: https://www.ravensburger.com/spielanleitungen/ecm/Spielanleitungen/Scotland_Yard_W_And_B_GB.pdf

Kan, N. R. (n.d.). *lms.rechnernetze.fh-wedel.de*. Retrieved October 30, 2019, from https://lms.rechnernetze.fh-wedel.de:888/mod/page/view.php?id=1443