Scotland Yard Assessment Report

COMS10017-2022

Frederick Zou & Zhengyang Xie

**CW-Model Summary:**

For our CW-Model of the Scotland Yard, we have completed and passed all 83 tests. And we use the Implementation Guide through both CW-Model and CW-AI.

Factory

Firstly, we create the private class which implement the GameState, by called MyGameState. And it pops out the skeleton frame shows what we need to implement. And our constructor creates a few essential fields such as the game graph, a list of rounds, and the list of player objects as well as checking that the players, tickets and the game have been initialized correctly. Then, we used Getters to complete several classes such as getPlayers, GetDetectiveLocation and getPlayerTickets and so on.

Visitor pattern

The Advance part is the core of the whole game. We used Visitor Pattern through this part, here, we can integration what we have and verify the player’s behavior by adding some small methods. That helps us to get information about a specific move, and the state of Mr.X of reveal and hidden by using DoubleMove or SingleMove.

Observer

We mostly used this method at MyModelFactory, which can notify all the observers when the state of the observed object changes, such as the player winning or losing a game or use SingleMove or DoubleMove ticket to move. The subject can notify its observer objects without knowing them. It just calls their update method as prescribed in the Observer interface.

**CW-AI Summary:**

For our CW-AI of the Scotland Yard, we have completed all function we wrote and running normally. Our AI is not only concise and comprehensive, but also can take the optimal solution at every step and stay away from the detective as far as possible. At this part of the code, we used Dijkstra, Minimax, Visitor Pattern and Alpha-Beta pruning algorithm.

According to the guide, we determined our scoring method, which is calculate the maximum distance from Mr.X to every detective in the next step. And at first, we used Dijkstra to calculate the shortest distance from detectives to Mr.X, the stored it into scores. In this part of the AI, we also formulate different weight for each ticket (Taxi<Bus<Underground<Secret).

We called some functions (board.getPlayers and board.GameState) given in the file to obtain detective and Mr.X information. And we get Mr.X and detective location and them information and used visitor Pattern to obtain the Location of Mr.X after using each SingleMove and DoubleMove ticket. After finishing those code, Mr.X has chances to escape the pursuit of detectives.

After that, to make our AI more intelligent, we added Minimax to the code. It uses a simple Recursive Algorithm to calculate from top to bottom to the leaf node of the tree, and then sends back the Minimax through recursive backtracking. In this way, we can get the optimal solution or minimize the possibility of AI going out of the worst case in the next step. However, because it must check the number of game states when running, which will increase exponentially with the progress of the game, increasing the amount of calculation of the program. Although exponential growth cannot be eliminated, we can used AlphaBeta to effectively reduce it and subtract those branches that do not affect the decision, to get the results as soon as possible.

And importantly, we added additionalscore. To ensure that when the game has just started, Mr.X can give priority to the tickets with shorter travel distance (Taxi or bus). When the critical moment, such as detective approaching Mr.X, he will use the doubleMove ticket.

**Reflection on achievement:**

In the close task, we can successfully complete the tests of 83 tests, and use methods including observer, visitor pattern, etc. In the part of getWinner, we divide the victory conditions very carefully: Mr.X winning scenarios: there are no remaining rounds and if detectives are out of tickets or stuck; detectives winning scenarios: Mr.X is captured or Mr.X is stuck or has no tickets. In the advance move, by using double ticket, we separate it to several detail situations when double ticket is used during reveal rounds, and each is treated separately. In MyModelFactory, we just simply finish all test by using observer and called advance move in MyGameStateFactory.

In the open task, we made Mr.X to pick move based on the score of it. The scoring systems mainly focused on the distance between Mr.X and detectives, however, we have also assigned different transportation with different weights and assign scores for Double and Secret ticket. We used Dijkstra algorithm to calculate the shortest distance from detectives to Mr.X, get Mr.X and detective location and information and used visitor Pattern to obtain the Location of Mr.X. after using each Move. And to make our AI more intelligent, we added Minimax to the code, and used Alpha-Beta pruning to effectively reduce it and subtract those branches that do not affect the decision to get the results as soon as possible. When building the game tree, we are able to set the number of iteration depth to any desired value (The larger the number, the deeper the number of depth). When the number of Iteration depth (count) is set to 7, the calculation can be completed almost instantaneously. When the number of Iteration depth (count) is set to 9-11, it will reduce the calculation speed, but Mr.X have a greater chance to pick a better move since it considered more steps forward.

**Reflection on strengths, limitations, and improvements:**

Our codes passed all tests with high readability and well organized. The AI is concise and comprehensive with only around 240 lines in length. It can pick the optimum move at every step and try to stay away from detectives with consider few steps further. With our scoring system, Mr.X is expected to use Taxi or Bus ticket with high priority and only use Double ticket under emergency situation (additional score). We can easily manipulate the depth of the game tree to influence Mr.X’s ability on decision making, however, with more level in the game tree, more calculation and more time is needed.

There are several limitations we encounter in AI now. The first one is that secret tickets are often used at the beginning of each game. We have tried to change the value of additional score, but it doesn’t seem to have a major impact on the performance of the AI. The second limitation we encounter is that when the number of Iteration depth rises from 8 to 10 or even more, the AI probability will not be able to calculate the next steps within the given time due to a massive amount of calculation needed for all the possible moves. and the third one is that our basic logic is to let Mr.X run in the opposite direction of detectives, and detectives will easily put Mr. X is forced to the boundary of the game map, which will lead Mr. X to be surrounded by detectives.

Theoretically, there is a method to solve this problem, because even if Alpha-Beta is used, it still needs to search part of the space until the termination state. For more in-depth nodes, this method is unrealistic to calculate all the time in the effective time. Therefore, the heuristic evaluation function can be added to effectively transform the non-terminating node into the terminating node at the end of time and use the truncation test to replace the terminating test to obtain its optimal solution. Or we can use multi-threading to double the amount of computation of AI at the same time. To solve Mr.X has been pushed to the boundary, we can partition each point of the game map, divide the outermost part into a region, and adjust the scores of these points to a low level to make Mr.X do not go near the boundaries of the map unless necessary.