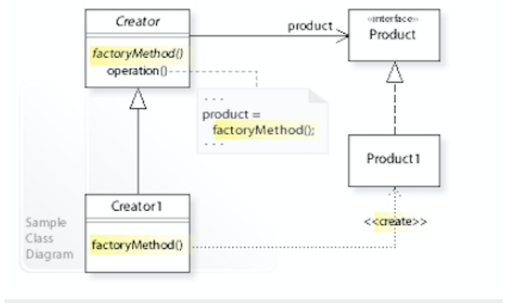
COMS 10009 Object-Oriented Programming

Scotland Yard Project Report

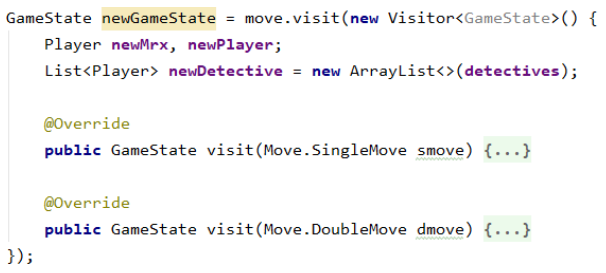
2nd May 2020

When hearing the interesting project, I was excited to accomplish this game. I know it's quite hard to finish the project on my own, so I planned to ask TAs for help. However, due to the outbreak of the COVID-19, it's not easy to find TAs and not convenience to describe my problems fully clear to them. Even though, I still tried my best to work on Scotland Yard Project, passing all CW-Model tests and managed to work on CW-Ai part. We believe we learn a lot through the questions we met in our work. Here are some achievement we reached.

Firstly, we built the skeleton of the whole project. During this step, we got stuck on the "Factory Pattern" that we were confused how to implement this pattern in the "MyFactoryModel". To solve the problem we asked TAs, searched on Google for relative information and discussed with our friends. Finally, we took the most basic steps and understood that Pattern is an assess point of Java regular expression API. When we need to deal with regular expressions in Java, we start with pattern class. ***Basically explaining is that we create object without exposing the code to the client and refer to newly created object on different inputs using a common interface.*** The picture on the left shows the function and relationships of Pattern class.

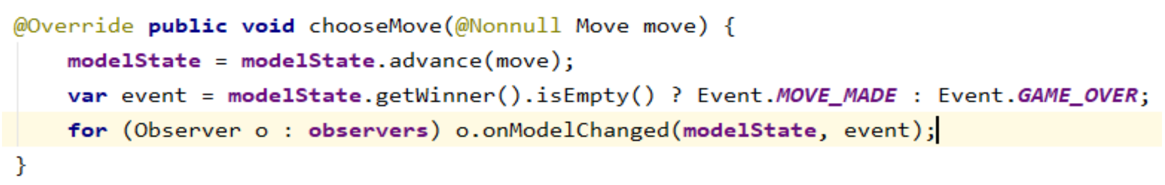


Secondly, we stepped in another problem when we work on "Advance" part. The first matter was that we were a little puzzled about the "visit pattern" so we went back to the PPT and the course Replay. Factually, this was not hard and we worked out quickly: "visit pattern" allows us to create a new operation which will not introduce the modifications to an already existing object structure. ***Here is our visit pattern code. We didn't change the functions of the move, but we can achieve some changes and update the state***. The other little matter is the double dispatch method. ***We used visit function twice but with different arguments, one take in "Move.SingleMove smove" another is "Move. DoubleMove dmove". Though we only access visit method but the program can recognize the argument and use the proper functions.***



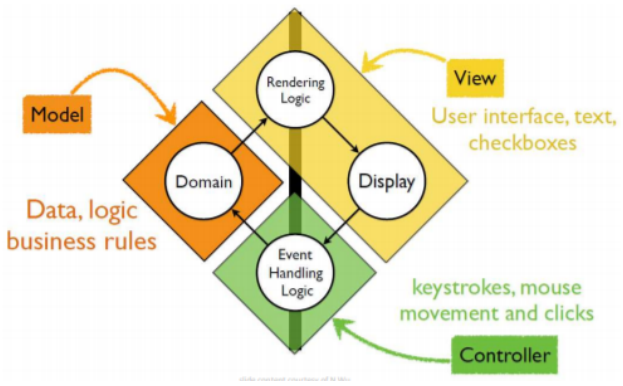
```
GameState newGameState = move.visit(new Visitor<GameState>() {
    Player newMrx, newPlayer;
    List<Player> newDetective = new ArrayList<>(detectives);

    @Override
    public GameState visit(Move.SingleMove smove) {...}

    @Override
    public GameState visit(Move.DoubleMove dmove) {...}
});
```

Thirdly, we have our last part, observer pattern, which is a one-to-many connection, let all the registered objects notify changes updated. It is implemented by creating an abstract observer class first and then let all the observer overload the update function. In our cases, the model is observer and game state is observable. ***Game state will inform model whether a move has been made or a game has ended, and model will update to its own state.***



```
@Override public void chooseMove(@Nonnull Move move) {
    modelState = modelState.advance(move);
    var event = modelState.getWinner().isEmpty() ? Event.MOVE_MADE : Event.GAME_OVER;
    for (Observer o : observers) o.onModelChanged(modelState, event);
}
```

As you can see through this project, our first java game program, we learnt something, applied what we got in class and searched additional information to support us finishing it. However, there are some additional codes supporting the graphical user interface, GUI. It is implemented through the following Model-View Controller pattern. Observer pattern also appears there, and its function is to create a layered structure that separates GUI and code. In addition, it has a composite pattern between Rendering Logic and Display. For example, GUI window contains panes, which contain buttons and text. All of these need to implement drawn function. There is also a strategy pattern between Controller and View in order to select from different input method like keyboard and mouse.


Model
Data, logic business rules
Domain
Rendering Logic
Display
Event Handling Logic
View
User interface, text, checkboxes
keystrokes, mouse movement and clicks
Controller

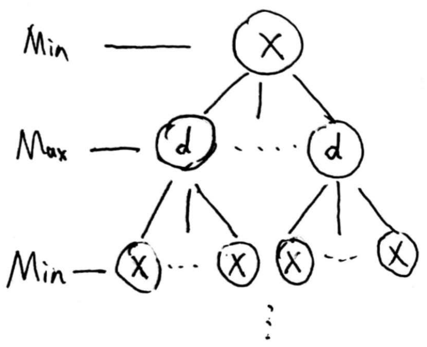
We only finish the Dijkstra short path algorithm in the code. This text is describe my understanding of the whole AI code after reading some article.

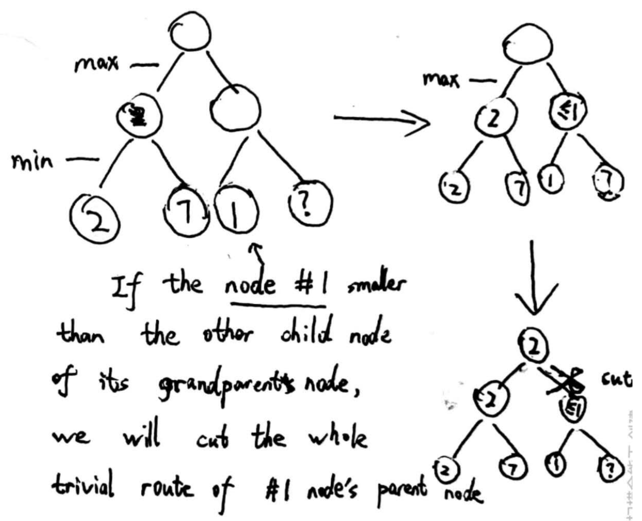
SCORE

The score in each layer of minimax algorithm is based on the distance between detectives and Mr. X. The distance is calculated by Dijkstra's short path algorithm in a directed graph. In this step we add weight on each path, for instance, the distance between two bus stops is assigned to 2 and the distance between two underground station is assigned to 4. In the end we can get an integer score which can be used in the following minimax algorithm.

MINIMAX

The basic idea of minimax algorithm is to minimize opponents' score and maximize its own score. Firstly, the detectives will always want to make the score as small as possible (catch Mr. X). As a result, detectives read the travel log of X and calculate scores of all Mr. X's possible position, choosing the move can make total score decrease dramatically. On the other hand, Mr. X will anticipate the next few rounds based on detectives' moving principle mentioned above. So, what X will get is an AI tree that calculated from bottom to up, to





If the node #1 smaller than the other child node of its grandparent's node, we will cut the whole trivial route of #1 node's parent node

decide the best move at current round.