I. Introduction

[Connor’s job]

II. Model

[Connor’s job - introduction]

a. MyGameStateFactory

MyGameStateFactory is an implementation of Factory with a single method: build, which itself return a new MyGameState object, a subclass of MyGameStateFactory that implements GameState. Most of the operations are done in the MyGameState class and as such this section of the report will be focusing on it.

i. Atributes

The atributes of MyGameState are as follow:

* setup, (of type GameSetup) holds the setup of the current game board,
* remaining, (of type ImmutableList<Piece>) holds the pieces left to move in the current round; it alternates between the detective pices and the mrX piece,
* log, (of type ImmutableList<LogEntry>) holds mrX’s travell log,
* mrX, (of type Player),
* detectives, (of type List<Player>),
* everyone, (of type ImmutableSet<Piece>) is a set of all the pieces present in the current game,
* moves, (of type ImmutableSet<Move>) holds all the legal moves that the pieces can make in their current positions on the game map,
* remainingRounds, (of type ImmutableList<Boolean>) lists the truth values of the reveal state of the remaining rounds in relation to mrX’s travell log.

All the atributes are initialised in MyGameState’s constructor, as they are passed as arguments when the object is created, or they are derived from the passed arguments.

ii.Methods

advance

- takes as argument a move and then it calls in turn it’s helper methods to advance the game to the next game state if the move is legal.

advance’s helper methods are:

updateRemaining - updates the remaining atribute to contain all the detectives present in the game if the move was commenced by MrX. If the move was commenced by a detective, remaining is updated to contain either the detectives who have not yet made a move and have at least one ticket, or, if the detective was the last one in remaining in the previous game state, remaining is updated to contain only MrX;

updateLocations – updates the location of the player who commenced the move. This method makes use of the visitor pattern to get the final destination of the move, so there is no different case for single or double moves;

updateTickets – updates the ticket that each player holds. This method also makes use of the visitor pattern as the ImmutableList<Ticket> tickets holds all the tickets used in the move. The tickets are all used and if the move was commenced by a detective, then mrX receives said tickets;

updateLog – updates mrX’s travell log. The visitor pattern is used twice, to get the destinations of the move and the tickets used as immutable lists. For each destination of the move, a new log entry is added to log in accordance to the reveal state of the round.

getDetectiveLocation

- takes a detective as argument and returns they’re loction as an Optional<Integer>

getPlayerTickets

- has a subclass MyTicketBoard that implements TicketBoard. The function then returns an Optional of MyTicketBoard with the player’s piece (which is passed as an argument to getPlayerTickets) as an argument is returned.

getWinner

- TO DO

getAvailableMoves

- return a immutable set of all possible moves if the game has ended. The state of the game is determined on the state of the list returned by the getWinners method; if the list is empty, then a list of moves is returnd; if the list is not empty, then an empty list is returned.

getAvailableMoves makes use of getMoves to get the list of all possible moves.

getMoves

- is a helper function for both getAvailableMoves and getWinner. It returns a immutable set of all possible moves of the pieces that are present in the remaining attribute.

If remaining contains detectives, then getSingleMoves is called for each detective and the set of all possible single moves for all detectives is returned. Otherwise, if remaining contains mrX, getSingleMoves and getDoubleMoves are called and the a set of all possible moves is returned.

getMove’s helper methods are:

getSingleMoves – goes through all adjacent nodes where a detective is not already present and if the player (which is passed as an argument) has the required ticket to travel to that location, the move is added to a list. If the player has a ‘secret’ ticket, than they can travel to any non-occupied adjacent node regardless of the required ticket type.

GetDoubleMove – calls getSingleMoves twice, once on to establish all possible single distance moves and one to determine all distance two moves. The method also checks for the pressence of a ‘double’ ticket.

iii. Getters

getSetup, getPlayers, getDetectiveLocations, and getMrXTravellLog all retrun atributes of MyGameState.

iv. Evaluation

TO DO

b. MyModelFactory

[Connor’s job]

III. AI

TO DO introduction to ScotFish

TO DO - replan AI section and write it

IV. Partner’s Contributions

TO DO

V. Workflow

TO DO (if there is time and space for it)

[Closing statement – Connor’s job]