The Game: Face 2 Face - AI Interface Reference

November 2019

1 Running a Game and Interpreting the History

Running a game is a simple as the following code snippet shows:

```
1 Game game = new Game(new RandomPlayer("a"), new RandomPlayer("b"));
2 Player winner = game.simulate();
3 game.getHistory().printHistory();
```

The first line initializes a game with two `RandomPlayer`s which get different names for identifiability. The second line starts the acutal simulation of the game and returns the player winning the game. In order to visualize the process of the game, one can obtain a `GameHistory` (see l. 3) and print it. The output should look similar to the following

```
1  Game ended after 5 moves and was won by random_player_a (player 1)
2  Initial game state:
3  Handcards                                  A   D
4  player 1 (52) : [10, 28, 58, 41, 53, 38]  1 60
5  player 2 (52) : [22, 6, 47, 59, 46, 44]  1 60
6  --------------------------------
7  move 0 (player 1) : 58->OWN_DESCENDING_DISCARD_PILE,
       28->OWN_ASCENDING_DISCARD_PILE, 41->OWN_ASCENDING_DISCARD_PILE,
       10->OWN_DESCENDING_DISCARD_PILE, 53->OWN_ASCENDING_DISCARD_PILE
8  Handcards                                  A   D
9  player 1 (50) : [38, 35, 19]          53  10
10 player 2 (52) : [22, 6, 47, 59, 46, 44]  1 60
11 --------------------------------
12 move 1 (player 2) : 46->OWN_DESCENDING_DISCARD_PILE,
       22->OPPONENTS_ASCENDING_DISCARD_PILE,
       44->OWN_ASCENDING_DISCARD_PILE, 6->OWN_DESCENDING_DISCARD_PILE,
       47->OWN_ASCENDING_DISCARD_PILE, 59->OWN_ASCENDING_DISCARD_PILE
13 Handcards                                  A   D
14 player 1 (50) : [38, 35, 19]          22  10
15 player 2 (46) : [21, 58, 49, 37, 29, 53]  59 6
16 --------------------------------
17 move 2 (player 1) : 38->OPPONENTS_ASCENDING_DISCARD_PILE,
       35->OWN_ASCENDING_DISCARD_PILE
```

```
Handcards                                    A   D
player 1 (45) : [19, 39, 20, 9, 46, 23]  35 10
player 2 (46) : [21, 58, 49, 37, 29, 53]  38 6
---------------------------------
move 3 (player 2) : 21->OPPONENTS_DESCENDING_DISCARD_PILE,
    49->OWN_ASCENDING_DISCARD_PILE, 53->OWN_ASCENDING_DISCARD_PILE,
    58->OWN_ASCENDING_DISCARD_PILE
Handcards                                    A   D
player 1 (45) : [19, 39, 20, 9, 46, 23]  35 21
player 2 (42) : [37, 29, 55, 33, 20, 16]  58 6
---------------------------------
move 4 (player 1) : 9->OPPONENTS_DESCENDING_DISCARD_PILE,
    20->OWN_DESCENDING_DISCARD_PILE, 19->OWN_DESCENDING_DISCARD_PILE,
    39->OWN_ASCENDING_DISCARD_PILE, 46->OWN_ASCENDING_DISCARD_PILE
Handcards                                    A   D
player 1 (40) : [23, 8, 59, 16, 37, 5]  46 19
player 2 (42) : [37, 29, 55, 33, 20, 16]  58 9
---------------------------------
Winner: random_player_a (player 1)
```

Firstly, note that you can see the winner of the game both at the top and at the end of the history. Secondly, apart from the initial game state, the history is constructed of elements giving information about a move of the form

```
1 move 3 (player 2) : 21->OPPONENTS_DESCENDING_DISCARD_PILE,
      49->OWN_ASCENDING_DISCARD_PILE, 53->OWN_ASCENDING_DISCARD_PILE,
      58->OWN_ASCENDING_DISCARD_PILE
2 Handcards                                    A   D
3 player 1 (45) : [19, 39, 20, 9, 46, 23]  35 21
4 player 2 (42) : [37, 29, 55, 33, 20, 16]  58 6
5 ---------------------------------
```

The first line of such an element gives you the number of the move which is visualized and which player made the associated move. Furthermore, you can see the placements of the move. In the element above, for example, the placement `21->OPPONENTS_DESCENDING_DISCARD_PILE` means that player 2 placed his hand card with the value 21 on the opponent's (i.e. player 1's) descending discard pile. Lines 3-4 give the state of the game after the move was made. In this case, line 3 gives the part of the game state associated with player 1. The number in brackets behind "player 1", i.e. 45, is the amount of cards on the draw pile of the player while the numbers in the list after the player name, i.e. `[19, 39, 20, 9, 46, 23]`, give the hand cards of player 1. The number following next, i.e. 35, (with the "A" above) gives the top card of the ascending discard pile of player 1, whereas the next number, i.e. 21, (with the "D" above) gives the top card of the descending discard pile.

Note that by default, the game engine will run the game with a fixed random seed and thus all games run will look identical. If you want to see how your AI behaves under different random seeds, you can initialize the game in the

following:

```
//random seed is 82356482
Game game = new Game(new RandomPlayer("a"), new RandomPlayer("b"),
    82356482)
```

2 Implementing your own AI

Your AI has to implement the following Java interface:

```
/**
 * This interface defines the functionality required by all AIs which
     should be
 * able to play the game.
 *
 * @author Alexander Tornede
 *
 */
public interface Player {

   /**
    * Initializes this player with the given random seed. If your AI
        uses any kind
    * of randomness, make sure to initialize the underlying random
        generator with
    * this seed.
    *
    * @param randomSeed The seed given to your agent by the game engine.
    */
   public void initialize(long randomSeed);

   /**
    * Computes the AIs next move given the game state.
    *
    * @param gameState The current state of the game
    * @return The next move of the AI based on the given game state.
    */
   public Move computeMove(GameState gameState);

   /**
    * Returns the name of this AI.
    *
    * @return The name of this AI.
    */
   public default String getName() {
      return getClass().getSimpleName();
   }
```

```
35 }
```

The `void initialize(long randomSeed)` method will be called by the game engine during the initialization phase and pass the random seed it uses for this run to your agent. If your agent uses any form of randomization, make sure that your random processes are based on the passed random seed. This is extremely important for the final evaluation as it guarantees reproducibility and thus a fair comparison of your approaches.

The `Move computeMove(GameState gameState)` method should contain the core of your approach. In each turn, your AI will be given a `GameState` object describing the current state of the board and your hand cards. Based on this information your AI needs to return a valid `Move` object describing the move your AI performs. A `Move` consists of `Placement`s which model that a given card is placed onto a given discard pile. Make sure that the `Move` which is returned by your AI is a valid one, i.e. adheres to all game rules, as it will be rejected by the engine otherwise. This will result in your AI loosing the game.

Please refer to the reference implementation of a random AI included in the course material for more details on how to create a `Move` and a `Placement`.