DA SILVEIRA Lucas
NOURRY Mathys
B2

Algorithm comparison

Algorithm N°1

```
String[] jouerIA1(char couleur) {
    String[] mvt = new String[2];
    Random rand = new Random();

    List<int[]> casesLibres = new ArrayList<>();
    for (int i = 0; i < state.length; i++) {
        for (int j = 0; j < state[i].length; j++) {
            if (state[i][j] == '.') {
                // Recherche des pions de la couleur donnée autour de la case libre
                if (i > 0 && state[i-1][j] == couleur) {
                    casesLibres.add(new int[] { i-1, j });
                }
                if (i < state.length-1 && state[i+1][j] == couleur) {
                    casesLibres.add(new int[] { i+1, j });
                }
                if (j > 0 && state[i][j-1] == couleur) {
                    casesLibres.add(new int[] { i, j-1 });
                }
                if (j < state[i].length-1 && state[i][j+1] == couleur) {
                    casesLibres.add(new int[] { i, j+1 });
                }
            }
        }
    }

    if(!casesLibres.isEmpty()){
        int index = rand.nextInt(casesLibres.size());
        int[] pion = casesLibres.get(index);
        int i = pion[0];
        int j = pion[1];

        // Recherche des destinations possibles pour la pièce en utilisant la méthode existante
        String[] destinationsPossibles = possibleDests(couleur, i, j);
        if (destinationsPossibles.length > 0) {
            int indexDest = rand.nextInt(destinationsPossibles.length);
            String destination = destinationsPossibles[indexDest];
            mvt[0] = (char)(i+'A') + "" + j;
            mvt[1] = destination;
        }
    }
    return mvt;
}
```

Explanation

The "jouerIA" method is used to implement an artificial intelligence in a game using a random strategy. It takes as input a variable "colour" of type char which represents the colour of the player for whom the AI is playing. It initializes an array of strings "String[] mvt" which will contain the coordinates of the selected move. It also uses a "Random rand" object to generate random numbers to select the move.

The method then declares a list "freeboxes" which will contain the coordinates of the freeboxes around the counters of the given colour. It uses a nested loop to go through all the squares in the current state of the game (state). For each square, if it is empty (state[i] [j] == '.'), it checks the adjacent squares to see if they contain pieces of the given colour. If it finds any, it adds the co-ordinates of these squares to the "free squares" list.

Once the loop has finished going through all the squares, it uses the "Random rand" object to generate a random number which corresponds to an index of the "free squares" list. It then uses the coordinates of this square to play the movement. It then returns the move as an array of strings "String[] mvt" which contains the coordinates of the move.

In summary, this method implements an artificial intelligence that plays randomly by looking for free squares around the pawns of the given colour to choose a move.

Pros and Cons

With this method, the AI player goes through all the free squares on the board (state) and looks for pieces of the given colour (colour) that are around these free squares. It then adds these pieces to a list (free squares) of pieces that can be moved. If the list is not empty, it randomly chooses a piece from the list and uses an existing method (possibleDests) to find the possible destinations for this piece. It randomly chooses a destination from these possible destinations and returns the start and destination coordinates of the chosen part using an array of strings (mvt).

The disadvantage of this method is that it does not take into account the potential moves of the opponents or the long-term objectives, it does not guarantee an optimal performance because it does not take into account the positions of the pieces or the strategies of play. It does not take into account the future state of the game, which can cause mistakes in the game, such as blocking important pieces. It is therefore important to combine it with other AI techniques to improve the performance of the AI.

Algorithm N°2

```
String[] jouerIA2(char couleur) {
    Random rand = new Random();
    String[] mvt = new String[2];
    boolean found = false;
    while(!found) {
        int idLettre = rand.nextInt(6);
        int idCol = rand.nextInt(6);
        if(state[idLettre][idCol] == couleur) {
            String[] dests = possibleDests(couleur, idLettre, idCol);
            for(int i = 0; i < dests.length; i++) {
                if(state[dests[i].charAt(0) - 'A'][dests[i].charAt(1) - '0'] == '.') {
                    mvt[0] = "" + (char)(idLettre + 'A') + idCol;
                    mvt[1] = dests[i];
                    found = true;
                    break;
                }
            }
        }
    }
    return mvt;
}
```

The method jouerIA2(char couleur) is a function that plays a random move for a player of the given colour in a board game. It uses a game state (state) which is an array of characters to determine the pieces of the given colour (colour) on the board. It also uses an existing method (possibleDests) to find the possible destinations for these pieces.

The method starts by creating a random object (rand) to generate random numbers. It also creates an array of strings (mvt) to store the start and destination coordinates of the chosen room. Then, the method enters a loop that stops only when a valid part is found.

In the loop, the method randomly generates row and column coordinates (idLetter and idCol) for a coin on the board. If the room matches the given colour (colour) and the room exists at these coordinates, the method uses the possibleDests method to find the possible destinations for this room. Then it scans the possible destinations to find a destination that is empty
(state[destinations[i].charAt(0) - 'A'][destinations[i].charAt(1) - '0'] == '.'). If such a destination is found, the method stores the start and destination coordinates of the chosen part in the mvt array and modifies the found variable to exit the loop.

Once the loop is complete, the method returns the array mvt containing the start and destination coordinates of the chosen part.

The disadvantages of this method are :
- It does not take into account the potential moves of the opponents or the long term objectives
- It does not guarantee optimal performance because it does not take into account the positions of the pieces or the game strategies.
- Randomness can lead to mistakes in play, such as blocking important pieces.
- It can enter an infinite loop if no valid moves are found.

Comparison:

The two methods playIA1(char colour) and playIA2(char colour) are intended to play a random move for a player of the given colour in a board game. However, they work in a slightly different way:

- The method jouerIA1(char colour) traverses all free squares on the board (state) and searches for pieces of the given colour (colour) that are around these free squares. It then adds these pieces to a list (freeboxes) of pieces that can be moved. If the list is not empty, it randomly chooses a piece from the list and uses an existing method (possibleDests) to find the possible destinations for this piece. It randomly chooses a destination from these possible destinations and returns the start and destination coordinates of the chosen piece.

- The method jouerIA2(char colour) randomly generates row and column coordinates (idLetter and idCol) for a piece on the board. If the piece matches the given colour (colour) and if the piece exists at these coordinates, it uses the possibleDests method to find the possible destinations for this piece. Then it scans the possible destinations to find a destination that is empty. If such a destination is found, it stores the starting and destination coordinates of the chosen room in an array and returns this array.
The two methods are quite similar in that they use random numbers to select rooms and destinations. The main difference is that the first method scans all free squares on the board to find pieces that can be moved, whereas the second method randomly generates the coordinates of a piece to find a valid destination. However, both methods do not take into account the potential moves of the opponents or the long term objectives, they do not guarantee an optimal performance because they do not take into account the positions of the pieces or the game strategies.

The best algorithm is the first, jouerIA1().