CS1006 P3 - Animal Chess

190022658

1 Overview

In this practical, we were asked to create a program on which / against which you could play a game called Animal Chess.

```
   a   b   c
 -----------------
1| G | L | E |
 -----------------
2|   | C |   |
 -----------------
3|   | c |   |
 -----------------
4| e | l | g |
 -----------------
```

Player 1 to move

Piece(s) on hand for player 1:
Piece(s) on hand for player 2:

Press M for menu

Example 1: Opening screen

2 Design

The program is divided into five classes - **Animal, Board, Minimax, Piece, Player**. The purpose of each class is as follows.

Animal - This class only contains an enum of all the animals.

Piece -

This class contains an animal and side (boolean) as its attributes. Other than these the important methods are capture() and promote(). capture() changes the side of the piece and if the piece is a Rooster, it gets converted into a Chick. Similarly promote() changes a Chick to a Rooster.

Board -

This class has a board (Piece[][]), player1 (Player), player2 (Player), player3 (Minimax), turn (boolean) and gameOver (boolean).

Here, I first tried using two players and when the user wanted to play against AI, I tried *Player player2 = new Minimax(*arguments*);* but this did not work as now I could not use methods from Minimax.java. Therefore the decision to make 3 players. At a point in the runtime, only two of these are instantiated as the game is a two player game.

There are methods in all the classes which creates a duplicate of an instance of the class. This was made because change to a reference type changes the original object otherwise. This will prove to be useful in Minimax. I am sure an alternate way of doing this could have proved to be better.

Then there are methods canMove() and move() which allow a user to move a piece and control if wrong input is entered.

reset() resets the board according to the mode one wants to use. The modes currently are player vs player, and player vs engine.

drop() controls dropping of a piece.

showCurrentBoard() prints out the board and pieces in a player's hand.

play() controls player vs player games.

playVE() controls player vs engine games.

parseInput() parses the input given by the user and moves/drops a piece.

The rest of the methods simply control the UI.

Player -

A player has side (boolean) as its main attribute. There, again, is a duplicatePlayer() method here.

canMove() here controls canMove() in Board. The method goes through all possible moves of the specific piece (without drops - they are controlled differently) and checks if the move entered by the user is one of them. If it is, then returns true. Otherwise, false.

move() moves the pieces, promotes and captures if required.

drop() controls all legal drops.

Minimax -

This class is a bit messier than I would have liked, but it works.

positionEval() evaluates position of a board on the basis of position and pieces. According to me, only a chick should get a point advantage as it approaches the last rank

because that is when it almost becomes a rooster which is a powerful piece. And so I have implemented that.

The most confusing method - possibleStates() - this method returns all the possible states of a board state after a move/drop. Firstly, a nested loop is used so that it iterates through all the pieces. We *continue;* when the piece's side is not equal to turn because that piece cannot move out of turn. Then we have another loop for all the possible moves for the piece. Here, we use the duplicate method to store the value of a board state to a temporary variable. For all the possible moves we execute the move in the temporary instance of board and then add it to a list of Boards. We store all the possible drops too.

changeState() is the method where the actual Minimax algorithm is implemented. This method goes to the leaves of the tree without actually creating a storage structure. This method is recursive and so applies the algorithm. After the algorithm comes back up to the original depth we store the instance of the board that is supposed to be the next move.

We finally use the method above in useEngine() which is not the best name, but is all that my brain could come up with. We accept depth here.

3 Testing

A fun sacrificial game with minimax depth=1

```
   a   b   c
 -----------------
1| G | L | E |
 -----------------
2|   | C |   |
 -----------------
3|   | c |   |
 -----------------
4| e | l | g |
 -----------------
```

Player 1 to move

Piece(s) on hand for player 1:
Piece(s) on hand for player 2:

Press M for menu

M <enter>

What do you want to do?

1. Play a new game
2. Continue
3. Exit

Enter a number:

1 <enter>

1. PvP (player vs player)
2. PvE (player vs engine)

Enter a number:

2 <enter>

```
    a   b   c
  -------------
1| G | L | E |
  -------------
2|   | C |   |
  -------------
3|   | c |   |
  -------------
4| e | l | g |
  -------------
```

Player 1 to move

Piece(s) on hand for player 1:
Piece(s) on hand for player 2:

b3 b2 <enter>

```
    a   b   c
  -------------
1| G | L | E |
  -------------
2|   | c |   |
  -------------
3|   |   |   |
  -------------
4| e | l | g |
  -------------
```

Player 2 to move

Piece(s) on hand for player 1: CHICK
Piece(s) on hand for player 2:

```
    a   b   c
  -------------
1| G |   | E |
  -------------
2|   | L |   |
  -------------
3|   |   |   |
  -------------
4| e | l | g |
  -------------
```

Player 1 to move

Piece(s) on hand for player 1: CHICK
Piece(s) on hand for player 2: CHICK

c4 c3 <enter>

```
    a   b   c
  -------------
1| G |   | E |
  -------------
2|   | L |   |
  -------------
3|   |   | g |
  -------------
4| e | l |   |
  -------------
```

Player 2 to move

Piece(s) on hand for player 1: CHICK
Piece(s) on hand for player 2: CHICK

```
    a   b   c
  -------------
1| G |   | E |
  -------------
2|   |   |   |
  -------------
3|   |   | L |
  -------------
4| e | l |   |
  -------------
```

Player 1 to move

Piece(s) on hand for player 1: CHICK
Piece(s) on hand for player 2: CHICK GIRAFFE

b4 b3 <enter>
Illegal Move!

```
    a   b   c
  -------------
1| G |   | E |
  -------------
2|   |   |   |
  -------------
3|   |   | L |
  -------------
4| e | I |   |
  -------------
```

Player 1 to move

Piece(s) on hand for player 1: CHICK
Piece(s) on hand for player 2: CHICK GIRAFFE

b4 a3 <enter>

```
    a   b   c
  -------------
1| G |   | E |
  -------------
2|   |   |   |
  -------------
3| I |   | L |
  -------------
4| e |   |   |
  -------------
```

Player 2 to move

Piece(s) on hand for player 1: CHICK
Piece(s) on hand for player 2: CHICK GIRAFFE

```
    a   b   c
  -------------
1| G |   | E |
  -------------
2|   |   |   |
  -------------
```

```
3|  l  |  C  | L |
  ------------------
4|  e |     |     |
  ------------------
```

Player 1 to move

Piece(s) on hand for player 1: CHICK
Piece(s) on hand for player 2: GIRAFFE

a4 b3 <enter>

```
    a    b    c
  ------------------
1| G |     | E |
  ------------------
2|     |     |    |
  ------------------
3|  l  |  e | L |
  ------------------
4|     |     |    |
  ------------------
```

Player 2 to move

Piece(s) on hand for player 1: CHICK CHICK
Piece(s) on hand for player 2: GIRAFFE

```
    a    b    c
  ------------------
1| G |     | E |
  ------------------
2|     |     |    |
  ------------------
3|  l  |  L |    |
  ------------------
4|     |     |    |
  ------------------
```

Player 1 to move

Piece(s) on hand for player 1: CHICK CHICK
Piece(s) on hand for player 2: GIRAFFE ELEPHANT

a3 b3 <enter>

```
    a    b    c
  ------------------
```

```
1|  G  |     |  E  |
  ------------------
2|     |     |     |
  ------------------
3|     |  I  |     |
  ------------------
4|     |     |     |
  ------------------
```

Player 2 to move

Piece(s) on hand for player 1: CHICK CHICK LION
Piece(s) on hand for player 2: GIRAFFE ELEPHANT

GAME OVER

What do you want to do?

1. Play a new game
2. Continue
3. Exit

3 <enter>

Testing 3: A sample game

This game shows that these functionalities work -
move, drop, minimax, game over
which is a basic game.

4 Conclusion

This practical was not the best I could do. Given more time I would have implemented history, save, load, minimax vs minimax (which can be done with the current programs but setup needed), a better UI, levels (which can also be done), an interface which allows game log to be entered and continue game from there on, undo, different ways of calling a piece's position, and different modes (for example chess has king of the hill). Maybe even timed.

The current submission would also have been better given more time. The implementation is not the best in terms of logic and presentation.

I did have some problems working from, which I thought were not worth asking for an extension for, but I do hope that the project will be marked leniently. Thanks a lot.

Random Reading

- https://docs.oracle.com/javase/7/docs/ (java docs)