University of Toronto
CSC 207, Fall 2021

Assignment 1: Three Musketeers

Due on **October 22nd, 2021, 11:59PM EST**

Description

This assignment's major learning goal is object oriented design in Java. In this assignment, you will be:

- Designing basic classes
- Ensuring classes are small and do one thing (Single Responsibility Principle)
- Creating class abstractions (Dependency Inversion)
- Using inheritance, composition, and polymorphism

We will be encoding the Three Musketeers Board game. Feel free to play against an AI to gain some intuition about this game.

THE GAME

This game is played on a 5x5 square board with this starting position:

- Starting Position:

```
  | A B C D E
--+----------
1 | O O O O X
2 | O O O O O
3 | O O X O O
4 | O O O O O
5 | X O O O O
```

RULES

- TURNS - The Musketeers always go first. Each player makes one move before passing to their opponent.
- MUSKETEER (X) - On the Musketeers' turn, a musketeer can move to an adjacent orthogonal (neighboring, non-diagonal) cell occupied by a guard, capturing it and occupying that cell.
- GUARD (O) - On the Guards' turn, a guard can move to an adjacent orthogonal empty cell.
- GOALS -
  - The Musketeers win if they cannot capture any guards AND if they are **not** on the same row or column.
  - The guards win if all the musketeers are on the same row or column at any point in the game.

An Example

The Guards' turn:

```
  | A B C D E
--+----------
1 |   X
2 |       O
3 |   X   O O
4 |       O
5 | O   O X
```

- Guard moves $D4 \rightarrow E4$

```
  | A B C D E
--+----------
1 |   X
2 |       O
3 |   X   O O
4 |         O
5 | O   O X
```

- Musketeers' only move is to capture the guard on $C5$

```
  | A B C D E
--+----------
1 |   X
2 |       O
3 |   X   O O
4 |         O
5 | O   X
```

- Guard moves $A5 \rightarrow B5$

```
  | A B C D E
--+----------
1 |   X
2 |       O
3 |   X   O O
4 |         O
5 |   O X
```

- Musketeers can only capture the guard on $B5$

```
  | A B C D E
--+----------
1 |   X
2 |       O
3 |   X   O O
4 |         O
5 |   X
```

- Guards win because 3 musketeers are all on the B column

TASKS

You will be given starter code to implement classes and methods.

Your implementation will include a collection of features, including:

- implement Human VS Human play
- implement Human VS Computer (with both Greedy and Random strategies)
- investigate the effectiveness of your Greedy and Random strategies.

(There is a glossary on some of these terms at the very end of this handout if you are interested in what some of the words mean.)

In your implementation you are allowed/required to:

- may add more private methods (no removal of the provided methods though!)
- may add additional classes
- use static and instance variables/methods as appropriate, with appropriate protections
- appropriately document your code
- make use of a stack to undo moves (or really any basic data structure as you see fit)
- use inheritance, composition, and polymorphism where possible

There should be no need to make any new files of your own.

TODO

ThreeMusketeers.java

- **move(Agent agent): (*Line 139*)**

  Gets a valid move from the given agent (Human/Random/Greedy), adds a copy of the move using the Move copy constructor to the moves stack for undoing later, then does the move on the board.

- **undoMove():**

  Removes a move from the top of the moves stack and undoes the move on the board.

Board.java

- **getCell(Coordinate coordinate): (*Line 61*)**

  Gets the cell on the board at the given coordinate.

- **getMusketeerCells():**

  Gets all the musketeer cells on the board.

- **getGuardCells():**

  Gets all the guard cells on the board.

- **move(Move move):**

  Executes the given valid move on the board.

- **undoMove(Move move):**

  Undo the given move on the board. The given move is a copy of the original move. So the cells in the copy are not the same as the cells on the board and the copy has the pieces that were originally in the fromCell and toCell.

- **isValidMove(Move move):**

  Checks if the given move is valid according to the rules of the game. (1) the toCell is next to the fromCell. (2) the fromCell piece can move onto the toCell piece.

- **getPossibleCells():**

  Gets all the cells that have pieces that can be moved this turn. Needs to check if the cell has at least one possible destination.

- **getPossibleDestinations(Cell fromCell):**

  Gets all the cells that the piece in the given fromCell can move to.

- **getPossibleMoves():**

  Gets all the possible moves that can happen this turn. This function can use the getPossibleCells and getPossibleDestinations functions to help get the list of possible moves.

- **isGameOver():**

  Checks if the game is over and if it is, sets the winner attribute in the Board class to the winner Piece Type.

BoardEvaluatorImpl.java

- **evaluateBoard(Board board)**

  : Returns a score for the given board. This function is used by the GreedyAgent. A higher score means the Musketeer side is winning and a lower score means the Guard side is winning. Add heuristics to generate a score given a board. The GreedyAgent will go through possible moves and pick the best move for the current side based on this score.

Guard.java

- **canMoveOnto(Cell cell):**

  Returns true if the Guard can move onto the given cell according to the rules of the game, false otherwise.

Musketeer.java

- **canMoveOnto(Cell cell):**

  Returns true if the Musketeer can move onto the given cell according to the rules of the game, false otherwise.

HumanAgent.java

- **getMove():**

  Asks the human for a move to be done. This function needs to validate the human input and make sure the move is valid for the piece type that is moving.

RandomAgent.java

- **getMove():**

  Gets a valid random move that can be done on the board.

Testing

Unit tests can be written in the `src/assignment1.testing` folder to test each class and the functions you implement. Unit tests created for this assignment will not be marked, but will be useful for you to ensure your code works for various cases.

`BoardTest.java` is an example test file, given to help you create more tests for `Board.java`.

Getting Started

Starter Code

Accept the Github Classroom invitation, then clone the repo, and set up in Eclipse.

Github Classroom Invitation Link: https://classroom.github.com/a/GbXlGwSt

Important: Edit the utorid file with your utorid (all lowercase)

Playing the Game

The first thing you should do is to play the game, in order to get a better understanding of how it works and what you will be creating in this assignment. Here is a link where you can play a completed version of the game against an AI.

Implementation

Once you have loaded the `A1Starter` project into the your workspace, you can find the files in the `src/assignment1` directory.

To gain a better understanding of how the game is pieced together, try applying a ”top-down” approach to reading the files (this means start from an overview of how the game works and then look at the implementation details of the smaller components). Start by opening `ThreeMusketeers.java`. Running this file will launch the game itself in the console of your editor.

Once you have done so, delve into the components of the game, such as the board the game is played on (`Board.java`), then the game pieces that fill the board (`Guard.java` and `Musketeer.java`), and the `Agents`.

GLOSSARY

- Strategy: In game theory, a player's strategy is an algorithm of how they decide to play (Wikipedia, 2021). An example of this could be to copy whatever the opponent did on their turn in chess. This would be a strategy, but perhaps not the best strategy to win the game of chess.
- Random (Strategy): Moving randomly, "without a strategy", per se.
- Greedy (Strategy): Taking the best move at that moment in time during the player's turn.
- Heuristic: A gauge that does not necessarily lead to a victory, but could provide some insight towards victory. An example could be to mentally keep an amount of moves you need to check your opponent's king in chess.