COMP 1451 - Assignment #3 Report

Alex Dai
A00937175

1. Design Decisions

From a designer's perspective, we want all classes loosely coupled. In this project, I want all manipulations and operations related to game pieces be included in the Board class. The reason and benefit is obvious. In the real world, the true status is only determined by the locations of pieces on the board. It is unnecessary to deal with pieces within the Player class. The player can make decisions of move, and input source and destination. After that, the main class Game, servers as a coordinator to pass the input locations to board object to complete the move. This design prevents extra operations of game piece objects in multiple classes. Meanwhile, each class or method is relatively cohesive and focuses on a specific task only.

Adding other types of game pieces are easy to implement. The GamePiece class serves as an abstract class of all pieces. We can create new piece classes and extend the GamePiece class. In each of these piece classes, we can define a different piece string to represent this piece on the board, and also override the isLegalMove method according to moving rules.

2. Description

(1) GamePiece
This is an abstract class representing the general piece objects for the game. There are two private fields: pieceString and owner. There is an abstract method isLegalMove.

(2) BallPiece
This class represents the Ball piece in the game. There is a method overriding here for the isLegalMove method to determine if the move is valid for this piece.

(3) TorchPiece
Similar to BallPiece class, the TorchPiece class represents the Torch piece in the game.

(4) Board
This class represents the game board. It has a field which is a two-dimensional array of game pieces. A few methods are provided:

a. check if a player loses the game by observing the game pieces on the board.
b. initialize the pieces on the board with default status.
c. display the board and locations of pieces on it.
d. move a piece from source to destination after validating the move.
e. a supportive method to determine if the path is clear for a single piece.

(5) Location
This class represents the piece location on the board. It has two fields for row and column coordinates.

(6) Player

This class represents the player of the game. The enterMove method allow the player to input source and destination of the intended move.

(7) Game
This is the main class of the game. Objects such as board, players, locationMapper are declared in this class. The play method is the entrance point to start the game.

(8) InputReader
This class provides methods to allow user input from the keyboard.

(9) LocationMapper
This functionality of this class is to map user input location such as “1a” to Location object to be used in other places.

(10) InvalidMoveException
This class is for invalid move exception. The constructor defines actions to be done when pre-defined exception occurs.

3. Proof of Testing

Battle of the Jugging Object Rules:
1. The pieces belonging to player One are shown as uppercase letters, "T" for torches and "O" for balls.
2. The pieces belonging to player Two are shown as lowercase letters, "t" for torches and "o" for balls.
3. Empty locations are shown as *.
4. The player who "captures" all the opponent’s pieces is the winner.
5. The players type their moves, specifying the source and destination of a move, e.g. "2b 3b" to move the ball at location 2b to empty location 3b.
6. Ball - A ball can move either up or down, but not sideways. It can move only one space per turn.
7. Torch - A torch can move up or down, and also side-to-side. A torch can move any number of spaces, but only one direction per turn.

Game start.

```
  a b c d e
1 * T T T *
2 * O O O *
3 * * * * *
4 * o o o *
5 * t t t *
```

Player 1 enter your move: > 6a 1b // source is out of bound
Invalid input for source location.

```
  a b c d e
1 * T T T *
2 * O O O *
```

3 * * * * *
4 * o o o *
5 * t t t *

Player 1 enter your move: > 1b 1f // destination is out of bound
Invalid input for destination location.

  a b c d e
1 * T T T *
2 * O O O *
3 * * * * *
4 * o o o *
5 * t t t *

Player 1 enter your move: > 1a 1b // source is empty
No piece at the source location.

  a b c d e
1 * T T T *
2 * O O O *
3 * * * * *
4 * o o o *
5 * t t t *

Player 1 enter your move: > 4b 3b // wrong piece
That's not your piece.

  a b c d e
1 * T T T *
2 * O O O *
3 * * * * *
4 * o o o *
5 * t t t *

Player 1 enter your move: > 1b 1b // source and destination are same
Invalid move, source and destination cannot be the same.

  a b c d e
1 * T T T *
2 * O O O *
3 * * * * *
4 * o o o *
5 * t t t *

Player 1 enter your move: > 2b 2a // ball cannot move sideways
Invalid move for this piece.

  a b c d e
1 * T T T *

2 * O O O *
3 * * * * *
4 * o o o *
5 * t t t *

Player 1 enter your move: > 1b 2a // torch cannot move along diagonal
Invalid move for this piece.

  a b c d e
1 * T T T *
2 * O O O *
3 * * * * *
4 * o o o *
5 * t t t *

Player 1 enter your move: > 1b 3b // piece cannot jump over another piece
Path is not clear.

  a b c d e
1 * T T T *
2 * O O O *
3 * * * * *
4 * o o o *
5 * t t t *

Player 1 enter your move: > 1b 2b // cannot capture own piece
You can't capture your own piece.

  a b c d e
1 * T T T *
2 * O O O *
3 * * * * *
4 * o o o *
5 * t t t *

Player 1 enter your move: > 10a 2b // input format is wrong, source or destination is of size 2
Invalid input format of source or destination.

  a b c d e
1 * T T T *
2 * O O O *
3 * * * * *
4 * o o o *
5 * t t t *

Player 1 enter your move: > 1b 1a // valid move

  a b c d e
1 T * T T *

```
2 * O O O *
3 * * * * *
4 * o o o *
5 * t t t *
```

Player 2 enter your move: > 4b 3b // valid move

```
  a b c d e
1 T * T T *
2 * O O O *
3 * o * * *
4 * * o o *
5 * t t t *
```

Player 1 enter your move: > 2b 3b // valid move

```
  a b c d e
1 T * T T *
2 * * O O *
3 * O * * *
4 * * o o *
5 * t t t *
```

Player 2 enter your move: > 5d 5e // valid move

```
  a b c d e
1 T * T T *
2 * * O O *
3 * O * * *
4 * * o o *
5 * t t * t
```

Player 1 enter your move: > 1a 4a // valid move

```
  a b c d e
1 * * T T *
2 * * O O *
3 * O * * *
4 T * o o *
5 * t t * t
```

Player 2 enter your move: > 5b 3b // valid move

```
  a b c d e
1 * * T T *
2 * * O O *
3 * t * * *
4 T * o o *
5 * * t * t
```

Player 1 enter your move: > 4a 4c // valid move

```
  a b c d e
1 * * T T *
2 * * O O *
3 * t * * *
4 * * T o *
5 * * t * t
```

Player 2 enter your move: > 5c 4c // valid move

```
  a b c d e
1 * * T T *
2 * * O O *
3 * t * * *
4 * * t o *
5 * * * * t
```

Player 1 enter your move: > 2c 3c // valid move

```
  a b c d e
1 * * T T *
2 * * * O *
3 * t O * *
4 * * t o *
5 * * * * t
```

Player 2 enter your move: > 4c 3c // valid move

```
  a b c d e
1 * * T T *
2 * * * O *
3 * t t * *
4 * * * o *
5 * * * * t
```

Player 1 enter your move: > 1c 3c // valid move

```
  a b c d e
1 * * * T *
2 * * * O *
3 * t T * *
4 * * * o *
5 * * * * t
```

Player 2 enter your move: > 3b 3c // valid move

```
  a b c d e
```

```
1 * * * T *
2 * * * O *
3 * * t * *
4 * * * o *
5 * * * * t
```

Player 1 enter your move: > 2d 3d // valid move

```
  a b c d e
1 * * * T *
2 * * * * *
3 * * t O *
4 * * * o *
5 * * * * t
```

Player 2 enter your move: > 4d 3d // valid move

```
  a b c d e
1 * * * T *
2 * * * * *
3 * * t o *
4 * * * * *
5 * * * * t
```

Player 1 enter your move: > 1d 3d // valid move

```
  a b c d e
1 * * * * *
2 * * * * *
3 * * t T *
4 * * * * *
5 * * * * t
```

Player 2 enter your move: > 3c 3d // valid move, player two won.

```
  a b c d e
1 * * * * *
2 * * * * *
3 * * * t *
4 * * * * *
5 * * * * t
```

Player 2 won!

4. Bug Report

(1) You cannot quit the game half way. The program is terminated only if one player wins.
(2) It does not support game board with large dimensions. The row and column should be less than 10.