**Project 4 Testing Report**

**Author: Tung Ho , Lin**

**Class Tsuro**

**Constructor with no input**

-tested in Interactions Pane

-input: Tsuro t = new Tsuro()

-output: a window with a 6x6 grid of empty Buttons appears, together with 2 windows each with 3 buttons with connections and stones

**Constructors with 2 inputs(row and column)**

-tested in Interactions Pane

-input: Tsuro t = new Tsuro(4, 4)

-output: a window with a 4x4 grid of empty Buttons appears, together with 2 windows each with 3 buttons with connections and stones

-intput: Tsuro t = new Tsuro(3, 6)

-output: a window with a 3x6 grid of empty Buttons appears, together with 2 windows each with 3 buttons with connections and stones

-input: Tsuro t = new Tsuro(4, 0)

-output: an IllegalArgumentException is thrown, saying that column cannot be 0 or negative

-input: Tsuro t = new Tsuro(0, 0)

-output: an IllegalArgumentException is thrown, saying that row cannot be 0 or negative

-input: Tsuro t = new Tsuro(-3, -5)

-output: an IllegalArgumentException is thrown, saying that row cannot be 0 or negative

**Constructor with 3 inputs(row, column and handsize)**

-tested in Interactions Pane

-input: Tsuro t = new Tsuro(4, 4, 3)

-output: a window with a 4x4 grid of empty Buttons appears, together with 2 windows each with 3 buttons with connections and stones

-input: Tsuro t = new Tsuro(5, 6, 2)

-output a window with a 5x6 grid of empty Buttons appears, together with 2 windows each with 2 buttons with connections and stones

-input: Tsuro t = new Tsuro(0, 0, 0)

-output: an IllegalArgumentException is thrown, saying that row cannot be 0 or negative

-input: Tsuro t = new Tsuro(3, -5, 0)

-output: an IllegalArgumentException is thrown, saying that column cannot be 0 or negative

-input: Tsuro t = new Tsuro(3, 5, 0)

-output: an IllegalArgumentException is thrown, saying that handsize cannot be 0 or negative

**Method main**

-tested in Interactions Pane

-input: java Tsuro

-output: a window with a 6x6 grid of empty Buttons appears, together with 2 windows each with 3 buttons with connections and stones

-input: java Tsuro 0 0

-output: an IllegalArgumentException is thrown, saying that row cannot be 0 or negative

-input: java Tsuro 3 5

-output: a window with a 3x5 grid of empty Buttons appears, together with 2 windows each with 3 buttons with connections and stones

-input: java Tsuro 3.0 5.0

-output: a NumberFormatException is thrown, saying the input is invalid

-input: java Tsuro -3 5

-output: an IllegalArgumentException is thrown, saying that row cannot be 0 or negative

-input: java Tsuro 5 -3 2

-output: an IllegalArgumentException is thrown, saying that column cannot be 0 or negative

-input: java Tsuro 5 3 6 2

-output: a statement saying that the input is invalid is printed

**Method getX()**

-tested in the game

-this method returns the x coordinate of the input tile

-when the game starts, player1 can only place his tile on the right edge of the game board, where x has a value of 0

-when the game runs, player1 can only start on the right edge

- when the game starts, player2 can only place his tile on the left edge of the game board, where x has a value of 5

-when the game runs, player2 can only start on the left edge

**Method getY()**

-tested in the game

-this method returns the Y coordinate of the input tile

-during the game when the stone hits the edge of the board, a getY() method is used to check if the tile containing a certain player ‘s stone is at the top edge of the game board(where y = 0)

-the game does stop and the player whose stone hits the top edge of the game board loses

**Method isPlayer1Turn()**

-tested in the game

-this method returns true if it is player1 ‘s turn to play

-player1 starts the game with a move and then player2 proceeds

-the game runs fluently with turn alternating between player1 and player2, only after player1 finishes his turn will player2 be able to play

**Method isPlayer2Turn()**

-tested in the game

-this method returns true if it is player2 ‘s turn to play

-player1 starts the game with a move and then player2 proceeds

-the game runs fluently with turn alternating between player1 and player2, only after player2 finishes his turn will player1 be able to play

**Method isAdjacent()**

-tested in the game

-this method returns true if the second input tile is adjacent to the first input tile

-a tile(tile1) is adjacent to another(tile2) only if:

1) tile1 is not tile 2

2) if the stone is at the top of tile1 and tile2 is on top of tile1

3) if the stone is on the right side of tile1 and tile2 is on the right of tile1

4) if the stone is at the bottom of tile1 and tile2 is below tile1

5) if the stone is on the left side of tile1 and tile2 is on the left of the tile1

-isAdjacent() is used to check if a player made a correct move, if the player clicks on the last tile he plays, then isAdjacent() will return false because Condition 1 is false, isAdajcent() will also return false if the player clicks on any other tiles that do not connect the stone side of the last tile because all other Conditions will check the move in different direction

-the game runs fluently while this method determines if the input tiles are next to each other

**Method getP1Adjacent()**

-tested in the game

-this method returns the adjacent tile of the input tile

-the adjacent tile of the input TsuroButton(tile) will be returned only if:

1) the stone is at the top of the input tile and getY(tile) is not 0

2) the stone is on the right of the input tile and getX(tile) is not the total columns

3) the stone is at the bottom of the input tile and getY(tile) is not the total rows

4) the stone is on the left of the input tile and getX(tile) is not 0

if the input tile is at the edge of the play board, null will be returned

-getP1Adjacent() is used in actionPerformed() to check if the adjacent tile is null and if the adjacent tile has connections in it.

-the game runs fluently while this method returns the adjacent tile of the input tile

**Method getP2Adjacent()**

-tested in the game

-this method works the same as getP1Adjacent but it checks the stone of player2 instead of player1

-the adjacent tile of the input TsuroButton(tile) will be returned only if:

1) the stone is at the top of the input tile and getY(tile) is not 0

2) the stone is on the right of the input tile and getX(tile) is not the total columns

3) the stone is at the bottom of the input tile and getY(tile) is not the total rows

4) the stone is on the left of the input tile and getX(tile) is not 0

if the input tile is at the edge of the play board, null will be returned

-getP2Adjacent() is used in actionPerformed() to check if the adjacent tile is null and if the adjacent tile has connections in it.

-the game runs fluently while this method returns the adjacent tile of the input tile

**Method getNextEndPoint()**

-tested with JUnit

-this method returns the endpoint after 90 CW rotation of a tile

-test 0 : test if the input endpoint is 0, it returns 2

-test 1: test if the input endpoint is 1, it returns 3

-test many: test if the input endpoint is 5, it returns 7

-negative numbers are not tested because if a negative number is inputted, the whole game will crash because there is no negative endpoints, the game runs fluently

-the game runs fluently while this method returns the endpoint after 90 CW rotation of a tile

**Method rotate90()**

-tested in the game

-this method will rotate everything in an input tile 90 CW, including the connections but excluding the stone

-it used the getConnections() from class TsuroButton to get the original connections and set the tile ‘s connection to the new rotated values by using getNextEndPoint() method

-this method is used in the player boards to rotate the hand tiles if it is highlighted and clicked

-the hand tiles, together with the connections, rotate with no error in the game

**Method getStoneStop()**

-tested with JUnit

-this method returns the stopping endpoint of a stone

-it retrieves the input button ‘s connection paths and uses the input starting endpoint to determine where will the stone goes

-in the JUnit test, a TsuroButton with a random connection is generated and used to test the method, therefore, the method will return different values every time we test it, but it doesn ‘t matter as long as it equals to the stopping endpoint of the path

-test 0: test if the input point is 0, it returns the stopping endpoint of the stone

-test 1: test if the input point is 1, it returns the stopping endpoint of the stone

-test many: test if the input point is 0, it returns the stopping endpoint of the stone

**Method getConnectEndpt()**

-tested with JUnit

-this method returns the connecting endpoint of the adjacent tile

-it is used in actionPerformed to generate the location of the stopping endpoint of the stone; every time a stone travels across a tile, this method has to be called to “climb over” the border between the tiles

-test 0: test if the input endpoint is 0, it returns 4

-test 1: test if the input endpoint is 1, it returns 5

-test many: test if the input endpoint is 7, it returns 3

-the game runs fluently while this method returns the connecting endpoint of the adjacent tile

**Method removeAllStone()**

-tested in the game

-although the removeStone() method exists already in the class TsuroButton, this method can remove all the stones quicker if there is only 1 stone in the input tile

-it is used in the player boards to remove all the stone after the player has made its move and the stone has a new position

-when the game runs, the player boards show only 1 stone at the endpoint of the current stone position on the board each time, proves that all the stones are removed and a new stone is added every time a player makes a move

**Method isCorrectMove()**

-tested in the game

-return true if the player has chosen a correct location to place his tile on the board

-a move is correct only if the below conditions are ALL met

1) the player board of the player whose turn it is has a highlighted button

if it is the FIRST move of this player, then:

2) the tile has to be on the right or left edge of the game board

if it is the later moves of his player, then:

2) the clicked button has to be empty

3) isAdjacent(clickedbutton, player1 ‘s stone last stopped) is true

-if not all of the above Conditions are met then it is not a correct move, the highlighted button will not be placed at the location of the clicked button.

-to test this method, I tried to click the player1 ‘s last button and all other buttons that are not adjacent to the player1 ‘s last button, nothing happened and it is still my turn to make a correct move

-the game runs fluently while this method determines whether a player makes a correct move

**Method p1GameOver()**

-tested in the game

-it returns true if player 1 has lost the game

-player1 will lose the game only if :

1) the stone has reached the edge of the game board

-although if two stones collide into each other, it will also cause player1 to lose, I will deal with that situation later in the actionPerformed() method

-the game runs fluently and ends if p1GameOver() is true

**Method p2GameOver()**

-tested in the game

-it returns true if player 2 has lost the game

-different from p1GameOver because it checks player2 ‘s stone

-player2 will lose the game only if :

1) the stone has reached the edge of the game board

-although if two stones collide into each other, it will also cause player2 to lose, I will deal with that situation later in the actionPerformed() method

-the game runs fluently and ends if p2GameOver() is true

**Method actionPerformed()**

-tested in the game

-this method will detect any mouse clicks on any of the button on the game board and make a decision according to all the Conditions at that moment

-the following lists are paths of actions that a mouse click can lead to:

1a) if isPlayer1Turn() is true; else 1b)

2a) if both players have not lost the game; else 2b)

3a) if isCorrectMove() is true; else do nothing

4a) if it is player1 ‘s first move; else 4b)

-then the clicked button on the game board will be replaced with the selected button in player1 ‘s player board and a stone will be added at the stopping endpoint according to the button ‘s connection; the location of the clicked button will be stored as player1 ‘s last button and the position of the stone will be stored

4b) if it is NOT player1 ‘s first move

-then the clicked button on the game board will be replaced with the selected button in player1 ‘s player board and a stone will be added at the stopping endpoint according to the button ‘s connection; the location of the clicked button will be stored as player1 ‘s last button and the position of the stone will be stored

5a) and if player1 ‘s last button is adjacent to player2 ‘s last button; else do nothing

6a) and if the two stones do not collide; else do nothing

-the stone will keep on moving along the path until it reaches the edge of the game board or if the next(adjacent tile) is empty

1b) if isPlayer2Turn() is true

if both players have not lost the game;

if isCorrectMove() is true; else

if it is player2 ‘s first move

-then the clicked button on the game board will be replaced with the selected button in player2 ‘s player board and a stone will be added at the stopping endpoint according to the button ‘s connection; the location of the clicked button will be stored as player2 ‘s last button and the position of the stone will be stored

if it is NOT player2 ‘s first move

-then the clicked button on the game board will be replaced with the selected button in player2 ‘s player board and a stone will be added at the stopping endpoint according to the button ‘s connection; the location of the clicked button will be stored as player2 ‘s last button and the position of the stone will be stored

and if player2 ‘s last button is adjacent to player1 ‘s last button

and if the two stones do not collide

-the stone will keep on moving along the path until it reaches the edge of the game board or if the next(adjacent tile) is empty

2b) if either of the player lost, the winning player will receive a message in the player board saying he won, the other one will receive one that says he lost; else 2bb)

2bb) if both player lost, both player receives a message in their player boards saying that they lost

-tests that I did: what happens

-clicked on the game board when there is no highlighted button: nothing happens

-clicked on the wrong location on the game board for my tile to go to: nothing happens

-lost on purpose on the first move: game declare me as the loser and the other player as the winner

-place a tile that connects the last tile with other tiles: the stone moves until it reaches the edge or it reaches an empty tile

-place a tile that will affect both stones and cause them both to move: they move along their own path

-kill the other player on purpose: the game declares me the winner and the other player as the loser

-collides with the other player ‘s stone: the game declares both the players losers

-this method constructs most of the game ‘s logic and therefore will crash if there is an error. So far after 50 games and testing every possible bug it can have, the game runs fluently

**Inner Class Player**

**Constructor with 1 input**

-tested in the game

-the constructor will create a player with a window containing 3 or the input number of hand buttons from the constructor of Tsuro

**Method isHighlighted()**

-tested in the game

-this method will return true if the input button is highlighted

-a button is highlighted if the background is yellow

-this method is used in isCorrectMove() to check if there is a selected button in the player board

-the game runs fluently and players can add their selected button to the board while this method checks if the input button is highlighted

**Method anyHighlighted()**

-tested in the game

-this method will return true if there is a highlighted button in the player’s board

-this method is used to check if there is a highlighted button before the player clicks on the game board, this method will prevent an error

-the game starts with no error

**Method getHighlighted()**

-tested in the game

-this method will return an array of int from the highlighted button

-this method is used to retrieve the connections of the players ‘ selected button in the player board

-the game runs fluently and after the player selected a correct location on the game board, the selected and highlighted button is copied and placed on the game board

**Method actionPerformed()**

-tested in the game

-this method will detect any mouse clicks on any of the button on the player board and make a decision according to all the Conditions at that moment

1a) if the clicked button is in Player1 ‘s board; else 1b)

2a) isPlayer1Turn() is true; else 2b)

3a) if the clicked button is not highlighted; else 3b)

-highlight the button

3b) if the clicked button is already highlighted

-rotate the tile 90 CW

1b) if the clicked button is in Player2 ‘s board

2b) isPlayer2Turn() is true

if the clicked button is not highlighted

-highlight the button

else

rotate the tile 90 CW

-test that I did: what happens

-click on one of the hand tiles when there are none of the tiles are highlighted: the clicked tile is highlighted in yellow

-clicked on another hand tiles while there is already on highlighted tile: the new clicked tile is highlighted and the originally highlighted tile is unhighlighted

-clicked on an already highlighted tile: the tile rotates 90CW

-keep clicking on an already highlighted tile: the tile keeps rotating 90CW

-the hand tiles are highlighted and unhighlighted when clicked and are able to rotate with no error