```
/**
 * @author Harley Phung
 * Project 4 - create a game called Tsuro for 2 players.
 */

import java.lang.Exception;
import javafx.application.Application;
import javafx.stage.Stage;
import javafx.scene.Scene;
import javafx.scene.layout.GridPane;
import javafx.scene.control.Button;
import javafx.scene.paint.Color;
import javafx.event.EventHandler;
import javafx.event.ActionEvent;
import javafx.scene.text.Text;

public class Tsuro extends Application {

    /** The default side of the board */
    private int defaultSide = 50;

    /** Number of row in the board */
    private static int numRow = 6;

    /** Number of column in the board */
    private static int numCol = 6;

    /** Number of card in player's hand */
    private static int handCard = 3;

    /** Number of players */
    private static int numPlayers = 2;

    /** Call the instance of TsuroButton, the button will be rememebered when
implements buttons */
    private TsuroButton[] button1 = null;

    /** Call the instance of TsuroButton, the button will be rememebered when
implements buttons */
    private TsuroButton[] button2 = null;

    /** Call the instance of TsuroButton, the button will be implemented with
highlighted buttons */
    private TsuroButton[][] bButton = null;

    /** The chosen button for player 1 to implement to the board */
    private TsuroButton chosenButton1 = null;

    /** The chosen button for player 2 to implement to the board */
    private TsuroButton chosenButton2 = null;

    /** The selected board button to be implemented by the player 1 */
    private TsuroButton selectedBoard1 = null;

    /** The selected board button to be implemented by the player 2 */
    private TsuroButton selectedBoard2 = null;

    /** The turn number */
    private int numTurn = 1;
```

```
/** current position of the blue stone */
private int currentBlue = -1;

/** Current position of the green stone */
private int currentGreen = -1;

/**
 * Method that return the number of row
 * @return numRow  the number of row in board
 */
public int getNumRow() {
    return this.numRow;
}

/**
 * Method that returns the numebr of columns
 * @retun numCol  the number of columns in the board
 */
public int getNumCol() {
    return this.numCol;
}
/**
 * Method that return the number of player's handsize
 * @return handCard  the number of player's hand size.
 */
public int getHandCard() {
    return this.handCard;
}

/**
 * Method that retun the number of players
 * @return numPlayers  the number of players
 */
public int getNumPlayers() {
    return this.numPlayers;
}

/**
 * Method that return the chosen button for player 1
 * @return chosenButton1  the chosen button for player 1
 */
public TsuroButton getChosenButton1() {
    return this.chosenButton1;
}

/**
 * Method that changes the chosen button for player 1
 * @param chosenButton1  the chosen button for player 1
 */
public void setChosenButton1(TsuroButton chosenButton1) {
    this.chosenButton1 = chosenButton1;
}

/**
 * Method that return the chosen button for player 2
 * @return chosenButton1  the chosen button for player 2
 */
public TsuroButton getChosenButton2() {
```

```
    return this.chosenButton2;
}

/**
 * Method that changes the chosen button for player 2
 * @param chosenButton2  the chosen button for player 2
 */
public void setChosenButton2(TsuroButton chosenButton2) {
    this.chosenButton2 = chosenButton2;
}

/**
 * Method that return the selected button in the board for player 1
 * @return selectedBoard1  the selected button in the board for player 1
 */
public TsuroButton getSelectedBoard1() {
    return this.selectedBoard1;
}

/**
 * Method that changes the selected button in the board for player 1
 * @param selectedBoard1  the selected button in the board for player 1
 */
public void setSelectedBoard1(TsuroButton selectedBoard1) {
    this.selectedBoard1 = selectedBoard1;
}

/**
 * Method that return the selected button in the board for player 2
 * @return selectedBoard2  the selected button in the board for player 2
 */
public TsuroButton getSelectedBoard2() {
    return this.selectedBoard2;
}

/**
 * Method that changes the selected button in the board for player 1
 * @param selectedBoard2  the selected button in the board for player 1
 */
public void setSelectedBoard2(TsuroButton selectedBoard2) {
    this.selectedBoard2 = selectedBoard2;
}

/**
 * Method that returns the current position of the blue stone
 * @return currentBlue  the current position of the blue stone
 */
public int getCurrentBlue() {
    return this.currentBlue;
}

/**
 * Method that changes the current position of the blue stone
 * @param currentBlue  the current position of the blue stone
 */
public void setCurrentBlue(int currentBlue) {
    this.currentBlue = currentBlue;
}
```

```
    /**
     * Method that returns the current position of the gree stone
     * @return currentBlue  the current position of the green stone
     */
    public int getCurrentGreen() {
        return this.currentGreen;
    }

    /**
     * Method that changes the current position of the green stone
     * @param currentGreen the current position of the green stone
     */
    public void setCurrentGreen(int currentGreen) {
        this.currentGreen = currentGreen;
    }

    /**
     * Method that finds the next tile
     * @param boardClicked  the button will be checked for the location in the
board.
     * @param position  the array of position of the button clicked on the board
     */
    public int[] thisPosition(TsuroButton boardClicked) {
        int[] position = new int[2];
        int outBreak = 0;
        for(int i = 0; i < this.numRow; i++) {
            for(int j = 0; j < this.numCol; j++) {
                if(boardClicked.equals(this.bButton[i][j])) {
                    position[0] = i;
                    position[1] = j;
                    outBreak = 1;
                }
            }
        }
        return position;
    }

    /**
     * Method that changed the location of the pathway so that they rotate 90
degrees clockwise
     * @param player  if player = 0 then this is player 1, else is player 2.
     * @param b  the button that is chosed to be rotated
     */
    public void rotateChosenButton(int player, TsuroButton b) {
        int[] newConnections = new int[8];
        for(int i = 0; i < 8; i++) {
            int j = i + 2;
            if (i == 2) j = 5;
            else if (i == 3) j = 4;
            else if (i == 6) j = 1;
            else if (i == 7) j = 0;

            if (b.getConnections()[i] == 2) newConnections[j] = 5;
            else if(b.getConnections()[i] == 3) newConnections[j] = 4;
            else if(b.getConnections()[i] == 6) newConnections[j] = 1;
            else if(b.getConnections()[i] == 7) newConnections[j] = 0;
            else newConnections[j] = b.getConnections()[i] + 2;
        }
        b.setConnections(newConnections);
```

```
    if(player == 0) this.setChosenButton1(b);
    else if (player == 1) this.setChosenButton2(b);
}

/**
 * Method that removes all the stone.
 * @param button  the button will be removed all the current stones in it.
 * @return true  if all the button is removed.
 */
public boolean removeAllStones(TsuroButton button) {
    button.removeStone(0);;
    button.removeStone(1);
    button.removeStone(2);
    button.removeStone(3);
    button.removeStone(4);
    button.removeStone(5);
    button.removeStone(6);
    button.removeStone(7);
    return true;
}

/**
 * Method that reset the chosenButton
 * @param player  the player in this turn.
 * @param chosenButton  the button that need to be reset
 */
public void resetChosenButton(int player, TsuroButton chosenButton) {
    chosenButton.setBackgroundColor(Color.WHITE);
    chosenButton.setRotate(0);
    chosenButton.setConnections(chosenButton.makeRandomConnectionArray());
    this.removeAllStones(chosenButton);
    if (player == 0) {
        chosenButton.addStone(Color.BLUE, 6);
    }
    else {
        chosenButton.addStone(Color.GREEN, 2);
    }
}

/**
 * Method that check if the stones collided
 * @return true  stone collided
 * @return false  stone did not collide
 */
public boolean stoneCollide() {
    int blueTileRow = thisPosition(this.selectedBoard1)[0];
    int blueTileCol = thisPosition(this.selectedBoard1)[1];
    int greenTileRow = thisPosition(this.selectedBoard2)[0];
    int greenTileCol = thisPosition(this.selectedBoard2)[1];

    if(blueTileRow == greenTileRow + 1 && blueTileCol == greenTileCol) {
        if(this.getCurrentBlue() == 0 && this.getCurrentGreen() == 4) {
            return true;
        }
        else if(this.getCurrentBlue() == 1 && this.getCurrentGreen() == 5) {
            return true;
        }
        else {
            return false;
```

```
        }
    }

    else if(blueTileRow == greenTileRow && blueTileCol == greenTileCol - 1) {
        if(this.getCurrentBlue() == 2 && this.getCurrentGreen() == 6) {
            return true;
        }
        else if(this.getCurrentBlue() == 3 && this.getCurrentGreen() == 7) {
            return true;
        }
        else {
            return false;
        }
    }

    else if(blueTileRow == greenTileRow - 1 && blueTileCol == greenTileCol) {
        if(this.getCurrentBlue() == 4 && this.getCurrentGreen() == 0) {
            return true;
        }
        else if(this.getCurrentBlue() == 5 && this.getCurrentGreen() == 1) {
            return true;
        }
        else {
            return false;
        }
    }

    else if(blueTileRow == greenTileRow && blueTileCol == greenTileCol + 1) {
        if(this.getCurrentBlue() == 6 && this.getCurrentGreen() == 2) {
            return true;
        }
        else if(this.getCurrentBlue() == 7 && this.getCurrentGreen() == 3) {
            return true;
        }
        else {
            return false;
        }
    }
    return false;
}

/**
 * Method that check if the stone will go out of board
 * @param player  if player = 0 then that is player 1, else is player 2
 * @return true  the stone go out
 * @return false  the stone on the board
 */
public boolean outOfBoard(int player) {
    TsuroButton currentTile = null;
    int currentStonePos = -1;
    if (player == 0) {
        currentTile = this.getSelectedBoard1();
        currentStonePos = this.getCurrentBlue();
    }
    else if(player == 1) {
        currentTile = this.getSelectedBoard2();
        currentStonePos = this.getCurrentGreen();
    }
```

```
        int currentTileRow = this.thisPosition(currentTile)[0];
        int currentTileCol = this.thisPosition(currentTile)[1];

        if(currentStonePos == 0 || currentStonePos == 1) {
            if(currentTileRow == 0) {
                return true;
            }
            else {
                return false;
            }
        }

        else if(currentStonePos == 2 || currentStonePos == 3) {
            if(currentTileCol == this.numCol - 1) {
                return true;
            }
            else {
                return false;
            }
        }

        else if(currentStonePos == 4 || currentStonePos == 5) {
            if(currentTileRow == this.numRow - 1) {
                return true;
            }
            else {
                return false;
            }
        }
        else if(currentStonePos == 6 || currentStonePos == 7) {
            if(currentTileCol == 0) {
                return true;
            }
            else {
                return false;
            }
        }
        else {
            return false;
        }
    }

    /**
     * Method that add the chosen button to the board
     * @param player  if player = 0 then that is player 1, else then that is player
2
     * @param boardClicked  the button in the board where the chosen button from
player's hand will be implemented
     */
    public void addChosenButtonToBoard(int player, TsuroButton boardClicked) {
        TsuroButton currentTile = null;
        int currentStonePos = -1;
        TsuroButton handTile = null;
        Color color = null;
        int nextStonePos = -1;
        int gameStatus = -1;
        int[] nextStoneMap = new int[]{4, 5, 6, 7, 0, 1, 2, 3};

        //if the player is player 1
```

```
        if (player == 0) {
            currentTile = this.getSelectedBoard1();
            currentStonePos = this.getCurrentBlue();
            handTile = this.chosenButton1;
            color = Color.BLUE;
            gameStatus = 1; //player 2 win
        }
        //if the player is player 2
        else if(player == 1) {
            currentTile = this.getSelectedBoard2();
            currentStonePos = this.getCurrentGreen();
            handTile = this.chosenButton2;
            color = Color.GREEN;
            gameStatus = 0; //player 1 win
        }

        int clickedRow = this.thisPosition(boardClicked)[0];
        int clickedCol = this.thisPosition(boardClicked)[1];
        int currentTileRow = this.thisPosition(currentTile)[0];
        int currentTileCol = this.thisPosition(currentTile)[1];

        if(currentStonePos == 0 || currentStonePos == 1) {
            if(currentTileRow == 0) {
                this.gameOver(gameStatus);
            }
            else if(clickedRow == currentTileRow - 1 && clickedCol ==
currentTileCol) {
                if(boardClicked.getConnections() != null) {
                    return;  //do nothing
                }
                else {
                    ;  //add chosen button to boardClicked
                }
            }
            else { //if click other than the button that are supposed to click then
let the player do it again
                return;
            }
        }
        else if (currentStonePos == 2 || currentStonePos == 3) {
            if(currentTileCol == this.numCol - 1) {
                this.gameOver(gameStatus);
            }
            else if(clickedRow == currentTileRow && clickedCol == currentTileCol +
1) {
                if(boardClicked.getConnections() != null) {
                    return;  //do nothing
                }
                else {
                    ;  //add chosen button to boardClicked
                }
            }
            else {
                return;
            }
        }
        else if (currentStonePos == 4 || currentStonePos == 5) {
            if(currentTileRow == this.numRow - 1) {
                this.gameOver(gameStatus);
```

```
            }
            else if(clickedRow == currentTileRow + 1 && clickedCol ==
currentTileCol) {
                if(boardClicked.getConnections() != null) {
                    return;  //do nothing
                }
                else {
                    ;  //add chosen button to boardClicked
                }
            }
            else {
                return;
            }
        }
        else if (currentStonePos == 6 || currentStonePos == 7) {
            if(currentTileCol == 0) {
                this.gameOver(gameStatus);
            }
            else if(clickedRow == currentTileRow && clickedCol == currentTileCol -
1) {
                if(boardClicked.getConnections() != null) {
                    return; //do nothing
                }
                else {
                    ;  //add chosen button to boardClicked
                }
            }
            else {
                return;
            }
        }

        nextStonePos = nextStoneMap[currentStonePos];
        this.removeAllStones(currentTile);
        boardClicked.setConnections(handTile.getConnections());
        boardClicked.addStone(color, handTile.getConnections()[nextStonePos]);
        boardClicked.setBackgroundColor(Color.WHITE);
        this.resetChosenButton(player, handTile);

        if(player == 0) {
            this.setCurrentBlue(boardClicked.getConnections()[nextStonePos]);
            this.chosenButton1 = null;
            this.setSelectedBoard1(boardClicked);
        }
        else if (player == 1) {
            this.setCurrentGreen(boardClicked.getConnections()[nextStonePos]);
            this.chosenButton2 = null;
            this.setSelectedBoard2(boardClicked);
        }

        if(this.stoneCollide() == true) {
            this.gameOver(2);
        }
        else if(this.outOfBoard(player) == true) {
            this.gameOver(gameStatus);
        }
        this.numTurn += 1;
    }
```

```
    /**
     * Find the next tile to see if the stone should be travel to the next tile
     * @param currentTileRow  the tile's row that the stone is currently on
     * @param currentTileCol  the tile's col that the stone is current on
     * @param currentStonePos  the current position of the stone
     * @return nextPos  the next position of the stone in the next tile
     */
    public int[] findNextTile(int currentTileRow, int currentTileCol, int
currentStonePos) {
        int[] nextPos = new int[2];
        if(currentStonePos == 0 || currentStonePos == 1) {
            nextPos[0] = currentTileRow - 1;
            nextPos[1] = currentTileCol;
        }
        else if(currentStonePos == 2 || currentStonePos == 3) {
            nextPos[0] = currentTileRow;
            nextPos[1] = currentTileCol + 1;
        }
        else if(currentStonePos == 4 || currentStonePos == 5) {
            nextPos[0] = currentTileRow + 1;
            nextPos[1] = currentTileCol;
        }
        else if(currentStonePos == 6 || currentStonePos == 7) {
            nextPos[0] = currentTileRow;
            nextPos[1] = currentTileCol - 1;
        }
        return nextPos;
    }

    /**
     * Method travel stone that will make the stone always go to the next non-null
path
     * @param player  if the player = 0, that is player 1, else then that is player
2
     */
    public void travelStone(int player) {
        int breakLoop = 0;
        TsuroButton currentTile = null;
        int currentStonePos = -1;
        int nextTileRow = -1;
        int nextTileCol = -1;
        int gameStatus = -1;
        TsuroButton nextTile = null;
        int[] nextStoneMap = new int[]{4, 5, 6, 7, 0, 1, 2, 3};
        int nextStonePos = -1;
        if(player == 0) {
            currentTile = this.selectedBoard1;
            currentStonePos = this.currentBlue;
            gameStatus = 1;
        }
        else if (player == 1) {
            currentTile = this.selectedBoard2;
            currentStonePos = this.currentGreen;
            gameStatus = 0;
        }

        int currentTileRow = this.thisPosition(currentTile)[0];
        int currentTileCol = this.thisPosition(currentTile)[1];
```

```
        while(breakLoop == 0) {
            if(this.outOfBoard(player) == true) {
                this.gameOver(gameStatus);
            }

            nextTileRow = this.findNextTile(currentTileRow, currentTileCol,
currentStonePos)[0];
            nextTileCol = this.findNextTile(currentTileRow, currentTileCol,
currentStonePos)[1];
            nextTile = this.bButton[nextTileRow][nextTileCol];
            if(nextTile.getConnections() == null) {
                breakLoop = 1;
            }
            else {
                nextStonePos = nextStoneMap[currentStonePos];
                if(player == 0) {
                    this.setSelectedBoard1(nextTile);
                    this.setCurrentBlue(nextTile.getConnections()[nextStonePos]);
                    nextTile.addStone(Color.BLUE, nextTile.getConnections()
[nextStonePos]);
                    this.removeAllStones(currentTile);
                }
                else if (player == 1){
                    this.setSelectedBoard2(nextTile);
                    this.setCurrentGreen(nextTile.getConnections()[nextStonePos]);
                    nextTile.addStone(Color.GREEN, nextTile.getConnections()
[nextStonePos]);
                    this.removeAllStones(currentTile);
                }
                currentTileRow = nextTileRow;
                currentTileCol = nextTileCol;
                currentTile = nextTile;
                currentStonePos = currentTile.getConnections()[nextStonePos];
                if(this.stoneCollide() == true) {
                    this.gameOver(2);
                }
                else if(this.outOfBoard(player) == true) {
                    this.gameOver(gameStatus);
                }
                else {
                    ;
                }
            }
        }
    }

    /**
     * Method that determines that the game is over
     * @param gameStatus  the game status when the game is over, indicating who win
the game
     */
    public void gameOver(int gameStatus) {
        this.setChosenButton1(null);
        this.setChosenButton2(null);
        this.setSelectedBoard1(null);
        this.setSelectedBoard2(null);
        this.setCurrentGreen(-1);
        this.setCurrentBlue(-1);
        System.out.println("The game is over");
```

```
        if(gameStatus == 0) System.out.println("Player 1 win");
        else if(gameStatus == 1) System.out.println("Player 2 win");
        else if(gameStatus == 2) System.out.println("Tie");
        System.exit(0);
    }

    /**
     * Method that create the player 1 hand and add to gridpane on a new stage
     */
    public void player1(){
        Stage stage1 = new Stage();
        GridPane p1GridPane = new GridPane();
        int player1Hand = 0;
        button1 = new TsuroButton[handCard];
        for ( ; player1Hand < this.handCard; player1Hand++) {
            Tsuro.this.button1[player1Hand] = new
TsuroButton(Tsuro.this.defaultSide, Tsuro.this.defaultSide);

Tsuro.this.button1[player1Hand].setConnections(Tsuro.this.button1[player1Hand].make
RandomConnectionArray());
            Tsuro.this.button1[player1Hand].addStone(Color.BLUE, 6);
            EventHandler<ActionEvent> myDefaultEvent = new PlayerAction();
            Tsuro.this.button1[player1Hand].setOnAction(myDefaultEvent);
            p1GridPane.add(Tsuro.this.button1[player1Hand], player1Hand, 1);
        }
        Scene scene1 = new Scene(p1GridPane);
        stage1.setScene(scene1);
        stage1.setTitle("Player 1");
        stage1.setX(300);
        stage1.setY(250);
        stage1.show();
    }

    /**
     * Create the player 2 hand and add to gridpane on a new stage
     */
    public void player2(){
        Stage stage2 = new Stage();
        GridPane p2GridPane = new GridPane();
        int player2Hand = 0;
        button2 = new TsuroButton[handCard];
        for ( ; player2Hand < this.handCard; player2Hand++) {
            Tsuro.this.button2[player2Hand] = new
TsuroButton(Tsuro.this.defaultSide, Tsuro.this.defaultSide);

Tsuro.this.button2[player2Hand].setConnections(Tsuro.this.button2[player2Hand].make
RandomConnectionArray());
            //For each button, you have to have the position of the stone
            Tsuro.this.button2[player2Hand].addStone(Color.GREEN, 2);
            EventHandler<ActionEvent> myDefaultEvent = new PlayerAction();
            Tsuro.this.button2[player2Hand].setOnAction(myDefaultEvent);
            p2GridPane.add(Tsuro.this.button2[player2Hand], player2Hand, 1);
        }
        Scene scene2 = new Scene(p2GridPane);
        stage2.setScene(scene2);
        stage2.setTitle("Player 2");
        stage2.setX(300);
        stage2.setY(350);
        stage2.show();
```

```
    }

    /**
     * Create a stage where the board can be placed in
     * @param primaryStage the stage where boards, players, and scene placed on
     */
    @Override
    public void start(Stage primaryStage) {
        int boardRow = 0;
        int boardCol = 0;
        GridPane gridPane = new GridPane();
        TsuroButton button;
        bButton = new TsuroButton[numRow][numCol];
        for (boardRow = 0; boardRow < Tsuro.this.numRow; boardRow++) {
            for (boardCol = 0; boardCol < Tsuro.this.numCol; boardCol++) {
                Tsuro.this.bButton[boardRow][boardCol] = new
TsuroButton(Tsuro.this.defaultSide, Tsuro.this.defaultSide);
                EventHandler<ActionEvent> boardEvent = new BoardAction();
                Tsuro.this.bButton[boardRow][boardCol].setOnAction(boardEvent);
                gridPane.add(Tsuro.this.bButton[boardRow][boardCol], boardCol,
boardRow);
            }
        }

        Scene scene = new Scene(gridPane);
        primaryStage.setTitle("Tsuro");
        primaryStage.setScene(scene);
        primaryStage.show();
        this.player1();
        this.player2();
    }

    /**
     * The inner class that help the player's click on the button and show the
required action on each button
     */
    public class PlayerAction implements EventHandler<ActionEvent> {
        public void handle(ActionEvent e) {
            TsuroButton b = (TsuroButton)e.getSource();
            for (int index = 0; index < Tsuro.this.handCard; index++) {
                if (b.equals(Tsuro.this.button1[index]) ||
b.equals(Tsuro.this.button2[index])) {

                    //For player 1
                    if(Tsuro.this.numTurn % 2 == 1) {
                        if (b.equals(Tsuro.this.button1[index])) {
                            if(Tsuro.this.getChosenButton1() == null) {
                                b.setBackgroundColor(Color.YELLOW);
                                Tsuro.this.setChosenButton1(b);
                            }
                            else {
                                if(Tsuro.this.getChosenButton1().equals(b)) {
                                    Tsuro.this.rotateChosenButton(0, b);
                                }
                                else { //change to other button, the chosen button
has to change to the newly selected one
                                    removeAllStones(Tsuro.this.chosenButton1);

Tsuro.this.getChosenButton1().setBackgroundColor(Color.WHITE);
```

```
                                    Tsuro.this.getChosenButton1().setRotate(0);

Tsuro.this.getChosenButton1().addStone(Color.BLUE, 6);
                                    b.setBackgroundColor(Color.YELLOW);
                                    Tsuro.this.setChosenButton1(b);
                                }
                            }
                        }
                    }

                    //For player 2
                    if (Tsuro.this.numTurn % 2 == 0) {
                        if (b.equals(Tsuro.this.button2[index])) {
                            if(Tsuro.this.getChosenButton2() == null) {
                                b.setBackgroundColor(Color.YELLOW);
                                Tsuro.this.setChosenButton2(b);
                            }
                            else {
                                if(Tsuro.this.getChosenButton2().equals(b)) {
                                    Tsuro.this.rotateChosenButton(1, b);
                                }

                                else {
                                    //change to other button, the choseen button
has to change to the newly selected one
                                    removeAllStones(Tsuro.this.getChosenButton2());

Tsuro.this.getChosenButton2().setBackgroundColor(Color.WHITE);
                                    Tsuro.this.getChosenButton2().setRotate(0);

Tsuro.this.getChosenButton2().addStone(Color.GREEN, 2);
                                    b.setBackgroundColor(Color.YELLOW);
                                    Tsuro.this.setChosenButton2(b);
                                }
                            }
                        }
                    }
                }
            }
        }
    }

    /**
     * An inner class handle the events happening on the board.
     */

    public class BoardAction implements EventHandler<ActionEvent> {

        public void handle(ActionEvent event) {
            try {
                TsuroButton boardClicked = (TsuroButton)event.getSource();
                if(Tsuro.this.numTurn == 1) { //The first turn of the player 1
                    if (Tsuro.this.getChosenButton1() == null) { //this means the
button is in board, do nothing, have to choose player's button first
                    }

                    else if (Tsuro.this.getChosenButton1() != null) {
                        for(int index = 0; index < Tsuro.this.handCard; index++) {
                            //for player 1
```

```
if(Tsuro.this.getChosenButton1().equals(Tsuro.this.button1[index])) {
                                for (int boardRow = 0; boardRow <
Tsuro.this.numRow; boardRow++) {
                                    int boardCol = 0;
                                    if(Tsuro.this.bButton[boardRow]
[boardCol].equals(boardClicked)) {
                                        Tsuro.this.bButton[boardRow]
[boardCol].setConnections(Tsuro.this.getChosenButton1().getConnections());
                                        Tsuro.this.bButton[boardRow]
[boardCol].addStone(Color.BLUE, Tsuro.this.getChosenButton1().getConnections()[6]);

Tsuro.this.setCurrentBlue(Tsuro.this.getChosenButton1().getConnections()[6]);

Tsuro.this.getChosenButton1().setConnections(Tsuro.this.getChosenButton1().makeRand
omConnectionArray());

Tsuro.this.getChosenButton1().setBackgroundColor(Color.WHITE);

removeAllStones(Tsuro.this.getChosenButton1());


Tsuro.this.getChosenButton1().addStone(Color.BLUE, 6);
                                        Tsuro.this.button1[index] =
Tsuro.this.getChosenButton1();
                                        Tsuro.this.setChosenButton1(null);
                                        Tsuro.this.numTurn = Tsuro.this.numTurn +
1;

Tsuro.this.setSelectedBoard1(Tsuro.this.bButton[boardRow][boardCol]);
                                        if(Tsuro.this.outOfBoard(0) == true) {
                                            Tsuro.this.gameOver(1);
                                        }
                                    }
                                }
                            }
                        }
                    }
                }
                //For player 2
                if(Tsuro.this.numTurn == 2) { //The first turn of player 2
                    if(Tsuro.this.getChosenButton2() == null) {
                        ;
                    }
                    else if(Tsuro.this.getChosenButton2() != null) {
                        for(int index = 0; index < Tsuro.this.handCard; index++) {

if(Tsuro.this.getChosenButton2().equals(Tsuro.this.button2[index])) {
                                for(int boardRow = 0; boardRow < Tsuro.this.numRow;
boardRow++) {
                                    int boardCol = Tsuro.this.numCol - 1;
                                    if(Tsuro.this.bButton[boardRow]
[boardCol].equals(boardClicked)) {
                                        Tsuro.this.bButton[boardRow]
[boardCol].setConnections(Tsuro.this.getChosenButton2().getConnections());
                                        Tsuro.this.bButton[boardRow]
[boardCol].addStone(Color.GREEN, Tsuro.this.getChosenButton2().getConnections()
[2]);
```

```
Tsuro.this.setCurrentGreen(Tsuro.this.getChosenButton2().getConnections()[2]);

Tsuro.this.getChosenButton2().setConnections(Tsuro.this.getChosenButton2().makeRand
omConnectionArray());

Tsuro.this.getChosenButton2().setBackgroundColor(Color.WHITE);

removeAllStones(Tsuro.this.getChosenButton2());
                                        Tsuro.this.getChosenButton2().setRotate(0);

Tsuro.this.getChosenButton2().addStone(Color.GREEN, 2);
                                        Tsuro.this.button2[index] =
Tsuro.this.getChosenButton2();
                                        Tsuro.this.setChosenButton2(null);
                                        Tsuro.this.numTurn = Tsuro.this.numTurn +
1;

Tsuro.this.setSelectedBoard2(Tsuro.this.bButton[boardRow][boardCol]);
                                        if(Tsuro.this.outOfBoard(1) == true) {
                                            Tsuro.this.gameOver(0);
                                        }
                                    }
                                }
                            }
                        }
                    }

                    else {
                        int player = -1;
                        if(Tsuro.this.numTurn % 2 == 1) {
                            player = 0;
                        }

                        else {
                            player = 1;
                        }
                        Tsuro.this.addChosenButtonToBoard(player, boardClicked);
                        Tsuro.this.travelStone(player);
                    }
                }

                catch(NullPointerException e) {
                    ;
                }
            }
        }

        /**
         * The main method that allows user to input the board size and handsize
         * @param args  the command line arguments
         * @exception Exception  if there's exception in the method
         */
        public static void main (String[] args) {
            try {
                if (args.length == 0) {
                    numRow = 6;
                    numCol = 6;
                }
```

```
            else if (args.length == 2) {
                numRow = Integer.parseInt(args[0]);
                numCol = Integer.parseInt(args[1]);
            }
            else if (args.length == 3) {
                numRow = Integer.parseInt(args[0]);
                numCol = Integer.parseInt(args[1]);
                handCard = Integer.parseInt(args[2]);
            }
            else if (args.length == 4) {
                numRow = Integer.parseInt(args[0]);
                numCol = Integer.parseInt(args[1]);
                handCard = Integer.parseInt(args[2]);
                numPlayers = Integer.parseInt(args[3]);
            }
        }

        catch(Exception e) {
            System.out.println("The input is inappropriate");
        }
        Application.launch(args);
    }
}
```