Appendix 1 - Scripts

```
/**
 * A luzhanqi piece.
 *
 * @author Tony Liang
 * @version 1.1 2015-01-10
 */
public class Piece implements Comparable
{
    //constants
    private static final String EXCEPTION_CAST_ERROR = "otherPiece must be a Piece";
    private static final String EXCEPTION_NULL_POINTER_ERROR = "otherPiece may not be null";

    private final String IMAGE_BOMB = "Bomb.png";
    private final String IMAGE_CAPTAIN = "Captain.png";
    private final String IMAGE_COLONEL = "Colonel.png";
    private final String IMAGE_ENGINEER = "Engineer.png";
    private final String IMAGE_FIELD_MARSHALL = "Field Marshall.png";
    private final String IMAGE_FLAG = "Flag.png";
    private final String IMAGE_GENERAL = "General.png";
    private final String IMAGE_LANDMINE = "Landmine.png";
    private final String IMAGE_LIEUTENANT_GENERAL = "Lieutenant General.png";
    private final String IMAGE_LIEUTENANT = "Lieutenant.png";
    private final String IMAGE_MAJOR_GENERAL = "Major General.png";
    private final String IMAGE_MAJOR = "Major.png";

    //ranks of the pieces
    private static final int RANK_BOMB = -1;
    private static final int RANK_CAPTAIN = 2;
    private static final int RANK_COLONEL = 4;
    private static final int RANK_ENGINEER = 0;
    private static final int RANK_FIELD_MARSHALL = 8;
    private static final int RANK_FLAG = -2;
    private static final int RANK_GENERAL = 7;
    private static final int RANK_LANDMINE = 9;
    private static final int RANK_LIEUTENANT = 1;
    private static final int RANK_LIEUTENANT_GENERAL = 6;
    private static final int RANK_MAJOR = 3;
    private static final int RANK_MAJOR_GENERAL = 5;

    //instance fields
    private int rank;
    private String image_location;
    private int side;
    private String name;
    private int uniqueID;


    /**
     * Constucts a piece with the specified rank and posisition on the screen where it is to be displayed.
     *
     * @param side the side this piece belongs to. 0 for left side and 1 for the right side of the board.
     * @param rank the rank of the piece
     */
    public Piece(int side, int rank, int uniqueID)
    {
        if (rank <= RANK_LANDMINE && rank >= RANK_FLAG)
        {
            this.rank = rank;
            //Determine by rank, the image this piece will be represented by
            if (rank == RANK_ENGINEER)
            {
                image_location = IMAGE_ENGINEER;
                name = "Engineer";
            }// end of if (rank == 0)
            else if (rank == RANK_LIEUTENANT)
            {
                image_location = IMAGE_LIEUTENANT;
                name = "Lieutenant";
```

```
        }// end of else if (rank == 1)
        else if (rank == RANK_CAPTAIN)
        {
            name = "Captain";
            image_location = IMAGE_CAPTAIN;
        }// end of else if (rank == 2)
        else if (rank == RANK_MAJOR)
        {
            image_location = IMAGE_MAJOR;
            name = "Major";
        }// end of else if (rank == 3)
        else if (rank == RANK_COLONEL)
        {
            image_location = IMAGE_COLONEL;
            name = "Colonel";
        }// end of else if (rank == 4)
        else if (rank == RANK_MAJOR_GENERAL)
        {
            image_location = IMAGE_MAJOR_GENERAL;
            name = "Major General";
        }// end of else if (rank == 5)
        else if (rank == RANK_LIEUTENANT_GENERAL)
        {
            image_location = IMAGE_LIEUTENANT_GENERAL;
            name = "Lieutenant General";
        }// end of else if (rank == 6)
        else if (rank == RANK_GENERAL)
        {
            image_location = IMAGE_GENERAL;
            name = "General";
        }// end of else if (rank == 7)
        else if (rank == RANK_FIELD_MARSHALL)
        {
            image_location = IMAGE_FIELD_MARSHALL;
            name = "Field Marshall";
        }// end of else if (rank == 8)
        else if (rank == RANK_BOMB)
        {
            name = "Bomb";
            image_location = IMAGE_BOMB;
        }// end of else if (rank == -1)
        else if (rank == 9)
        {
            image_location = IMAGE_LANDMINE;
            name = "Landmine";
        }// end of else if (rank == 9)
        else if (rank == -2)
        {
            image_location = IMAGE_FLAG;
            name = "Flag";
        }// end of else if (rank == -2)

        this.uniqueID = uniqueID;

        if (side == 1 || side == 0) this.side = side;
    }//end of if (rank < 12 && rank > -4)
}//end of constructor Piece(int rank, int x_coordinate, int y_coordinate, int width, int height)

/*
 * Accessors
 */

/**
 * Compares another piece to this piece. Returns 1 if the collision between the pieces results in this piece victorious,
 * 0 if both pieces are eliminated and -1 otherwise.
 *
 * @param otherPiece the piece to be comapared to this piece
 * @return 1 if the collision between the pieces results in this piece victorious, 0 if both pieces are eliminated and -1 otherwise
```

```
 */
public int compareTo(Object otherPiece)
{
    //Specified by the Comparable interface
    if (otherPiece == null) throw new NullPointerException(EXCEPTION_NULL_POINTER_ERROR);
    //is otherPiece a reference to this entry?
    if (this == otherPiece) return 0;
    //Is the other entry a Piece?
    if (otherPiece.getClass() != this.getClass()) throw new ClassCastException(EXCEPTION_CAST_ERROR);

    Piece other = (Piece) otherPiece;

    //Bombs are eliminated with any piece upon a collision
    if (other.getRank() == RANK_BOMB || this.getRank() == RANK_BOMB) return 0;
    //Engineers can remove landmines
    if (other.getRank() == RANK_ENGINEER && this.getRank() == RANK_LANDMINE) return -1;
    if (other.getRank() == RANK_LANDMINE && this.getRank() == RANK_ENGINEER) return 1;
    //otherwise, normal order of ranks are followed
    if (other.getRank() == this.getRank()) return 0;
    if (other.getRank() > this.getRank()) return -1;
    return 1;
}//end of method compareTo(Piece otherPiece)

/**
 * Indicates whether another piece, when collided with this piece, will result in both pieces being eliminated.
 * Returns true if both pieces are eliminated and false otherwise.
 *
 * @param otherPiece the piece to be compared with this piece
 * @return true if both pieces are eliminated and false otherwise
 */
public boolean equals(Object otherPiece)
{
    //is otherPiece a pointer to null?
    if (otherPiece == null) return false;
    //is otherPiece actually a piece?
    if (otherPiece.getClass() != this.getClass()) return false;
    //is otherPiece a reference to this entry?
    if (otherPiece == this) return true;

    Piece other = (Piece) otherPiece;

    //What happens if a piece is a bomb?
    if (other.getRank() == RANK_BOMB || this.getRank() == RANK_BOMB) return true;
    return other.getRank() == this.getRank();
}//end of method equals(Object otherPiece)

/**
 * Returns the unique ID number of this Piece.
 *
 * @return the unique ID number of this piece
 */
public int getID()
{
    return uniqueID;
}//end of method getID()

/**
 * Returns the name of the image file.
 *
 * @return name of the image file
 */
public String getImageLocation()
{
    return image_location;
}//end of method getImageLocation()
```

```java
    /**
     * Returns the name of this piece.
     *
     * @return the name of this piece
     */
    public String getName()
    {
        return name;
    }// end of method getName()

    /**
     * Returns the rank of this piece.
     *
     * @return the rank of this piece
     */
    public int getRank()
    {
        return rank;
    }//end of method getRank()

    /**
     * Returns the side of this piece.
     *
     * @return the side of this piece
     */
    public int getSide()
    {
        return side;
    }//end of method getSide()
    /**
     * Returns a string representation of this piece.
     *
     * @return a string representation of this piece
     */
    public String toString()
    {
        String pieceSide = "";
        if (side == 0) pieceSide = "red";
        if (side == 1) pieceSide = "blue";

        return "["
        + "Rank: " + name
        + "Side: " + pieceSide
        + "]";
    }//end of method toString()
}//end of class Piece()
```

```java
/**
 * A luzhanqi board that stores the position of each piece.
 *
 * @author Tony Liang
 * @version 1.1 2015-01-03
 */
public class Board
{
    //constants
    private static final int COLUMN_B = 1;
    private static final int COLUMN_C = 2;
    private static final int COLUMN_D = 3;

    private static final int NUMBER_OF_COLUMNS = 5;
    private static final int NUMBER_OF_ROWS = 12;

    //ranks of the pieces
    private static final int RANK_BOMB = -1;
    private static final int RANK_CAPTAIN = 2;
    private static final int RANK_COLONEL = 4;
    private static final int RANK_ENGINEER = 0;
    private static final int RANK_FIELD_MARSHALL = 8;
    private static final int RANK_FLAG = -2;
    private static final int RANK_GENERAL = 7;
    private static final int RANK_LANDMINE = 9;
    private static final int RANK_LIEUTENANT = 1;
    private static final int RANK_LIEUTENANT_GENERAL = 6;
    private static final int RANK_MAJOR = 3;
    private static final int RANK_MAJOR_GENERAL = 5;

    private static final int ROW_3 = 2;
    private static final int ROW_4 = 3;
    private static final int ROW_5 = 4;
    private static final int ROW_12 = 7;
    private static final int ROW_13 = 8;
    private static final int ROW_14 = 9;

    //instance fields
    private Piece[][] board = new Piece [NUMBER_OF_COLUMNS][NUMBER_OF_ROWS];
    /**
     * Constucts a board with default characteristics.
     */
    public Board()
    {
        for (int columnNumber = 0; columnNumber < NUMBER_OF_COLUMNS; columnNumber++)
        {
            for (int rowNumber = 0; rowNumber < NUMBER_OF_ROWS; rowNumber++)
            {
                board[columnNumber][rowNumber] = null;
            }// end of for (int rowNumber = 0; rowNumber < NUMBER_OF_ROWS; rowNumber++)
        }// end of for (int columnNumber = 0; columnNumber < NUMBER_OF_COLUMNS; columnNumber++)
    }// end of constructor Board()

    /**
     * Constructs a board that is identical to the another board.
     *
     * @param board the other board to be made identical to
     */
    public Board(Board board)
    {
        Piece[][] otherBoard = board.getBoard();
        for (int columnNumber = 0; columnNumber < NUMBER_OF_COLUMNS; columnNumber++)
        {
            for (int rowNumber = 0; rowNumber < NUMBER_OF_ROWS; rowNumber++)
            {
                this.board[columnNumber][rowNumber] = otherBoard[columnNumber][rowNumber];
            }// end of for (int rowNumber = 0; rowNumber < NUMBER_OF_ROWS; rowNumber++)
        }// for (int columnNumber = 0; columnNumber < NUMBER_OF_COLUMNS; columnNumber++)
```

```
}// end of constructor Board(Board boa)

/*
 * Accessors
 */

/**
 * Returns the board of this Board.
 *
 * @return a 2D array of Pieces representing the board of this Luzhanqi game
 */
public Piece[][] getBoard()
{
    return board;
}// end of method getBoard()

/**
 * Returns the piece at the specified x and y coordinates.
 *
 * @param x the x coordinate of the piece
 * @param y the y coordinate of the piece
 * @return the Piece at board location x and y
 */
public Piece getPiece(int x, int y)
{
    return board[x][y];
}// end of method getPiece(int x, int y)

/*
 * Mutators
 */

/**
 * Adds a piece to the board, returns true if the piece is added correctly, false otherwise.
 *
 * @param piece the piece to be added
 * @param locationx the x coordinate of the location the piece is to be added in
 * @param locationy the y coordinate of the location the piece is to be added in
 * @return true if the piece is added correctly, otherwise false
 */
public boolean addPiece(Piece piece, int locationx, int locationy)
{
    //add the piece if there is no existing piece at that location
    if (board[locationx][locationy] == null)
    {
        board[locationx][locationy] = piece;
        return true;
    }// end of if (board[locationx][locationy] == null)
    return false;
}// end of addPiece(Piece piece, int locationx, int locationy)

/**
 * Moves piece from <code>locationx1</code>,<code>locationy1</code> to <code>locationx2</code>,
 * <code>locationy2</code> and collides with the piece (if existing) at <code>locationx2</code>,
 * <code>locationy2</code>. Returns true if the piece was moved successfully, false otherwise.
 *
 * @param locationx1 the x coordinate of the location of the piece to be moved
 * @param locationy1 the y coordiante of the location of the piece to be moved
 * @param locationx2 the x coordinate of the destination location
 * @param locationy2 the y coordinate of the destination location
 * @return true if the piece is moved successfully, false otherwise.
 */
public boolean movePiece(int locationx1, int locationy1, int locationx2, int locationy2)
{
    //is there a piece at the destination location?
    if (board[locationx2][locationy2] != null)
```

```
        {
            //can you collide with pieces of the same side?
            if (board[locationx1][locationy1].getSide() != board[locationx2][locationy2].getSide())
            {
                //can a piece collide with another piece at a campsite?
                if ((locationx2 == COLUMN_B || locationx2 == COLUMN_D) && (locationy2 == ROW_3 || locationy2 == ROW_5
                    || locationy2 == ROW_12 || locationy2 == ROW_14)) return false;
                else if (locationx2 == COLUMN_C && (locationy2 == ROW_4 || locationy2 == ROW_13)) return false;
                //find which piece is greater and make adjustments to the board accordingly
                if (board[locationx1][locationy1].compareTo(board[locationx2][locationy2]) > 0)
                {
                    //if the selected piece is greater than the destination piece, then the selected piece moves to destination
                    removePiece(locationx2, locationy2);
                    addPiece(board[locationx1][locationy1], locationx2, locationy2);
                    removePiece(locationx1, locationy1);
                }// end of if (board[locationx1][locationy1].compareTo(board[locationx2][locationy2]) > 0)
                else if (board[locationx1][locationy1].compareTo(board[locationx2][locationy2]) < 0)
                {
                    //if the selected piece is smaller, then the selected piece is eliminated
                    removePiece(locationx1, locationy1);
                }// end of else if (board[locationx1][locationy1].compareTo(board[locationx2][locationy2]) < 0)
                else if (board[locationx1][locationy1].compareTo(board[locationx2][locationy2]) == 0)
                {
                    //if both pieces are equal, then both pieces are removed
                    removePiece(locationx1, locationy1);
                    removePiece(locationx2, locationy2);
                }// end of else if (board[locationx1][locationy1].compareTo(board[locationx2][locationy2]) == 0)
            }
            else
            {
                //if both pieces are from the same side
                return false;
            }// end of if (board[locationx1][locationy1].getSide() != board[locationx2][locationy2].getSide())
        }
        else
        {
            //add piece to destination
            addPiece(board[locationx1][locationy1], locationx2, locationy2);
            removePiece(locationx1, locationy1);
        }// end of if (board[locationx2][locationy2] != null)
        return true;
    }// end of method movePiece(int locationx1, int locationy1, int locationx2, int locationy2)

    /**
     * Removes the piece at the specified location from the board
     *
     * @param locationx the x coordinate of the location where the piece is to be removed
     * @param locationy the y coordinate of the location where the piece is to be removed
     */
    public void removePiece(int locationx, int locationy)
    {
        if (board[locationx][locationy] != null) board[locationx][locationy] = null;
    }// end of removePiece(int locationx, int locationy)
}// end of class Board()
```