CSE 4345 Andrew Socha
Source Code Vipul Kohli

JHalma

URLs:

Board 1: Tyler vs. Andrew

- Tyler: http://lyle.smu.edu/~tbgeorge/cse4345/a1/getMove.php
- Andrew: http://lyle.smu.edu/~sochaa/4345/FinalHalma/finalHalmaWithDamage.php

Board 2: Collider vs. Invalidity

- Invalidity: http://lyle.smu.edu/~aaloqla/halmagame/WebService.php
- Collider: http://lyle.smu.edu/~tbgeorge/cse4345/a1/getMove.php

Program

```
package ShowNTell;

/**
 * @(#)Program.java
 * Creates one or more Halma games.
 *
 * Includes GameBoard class,
 * which represents the board UI.
 *
 * @author Vipul Kohli
 * @author Andrew Socha
 * @version 12-4-2014
 */

import com.grack.nanojson.*;
import java.awt.*;
import java.util.*;
import javax.swing.*;

public class Program{

    public static void main(String[] args){
        String player1, player2,
               collisionPlayer1, collisionPlayer2,
               collision1Name, collision2Name,
               player1Name, player2Name,
               tieURL, tieName;

        //default players
        tieURL = "http://lyle.smu.edu/~jyeh/4345/api/index.php/
                  getMultiplayerMove";
        tieName = "Ty";
        player1 = "http://lyle.smu.edu/~tbgeorge/cse4345/a1/getMove.php";
        player2 = "http://lyle.smu.edu/~sochaa/4345/FinalHalma/
                   finalHalmaWithDamage.php";
        collisionPlayer1 = "http://lyle.smu.edu/~aaloqla/halmagame/
                            WebService.php";
        collisionPlayer2 = "http://lyle.smu.edu/~tbgeorge/cse4345/a1/
                            getMove.php";
        collision1Name = "Invalidity";
        collision2Name = "Collider";
        player1Name = "Tyler";
        player2Name = "Andrew";
        //text fields
        JTextField pfield1 = new JTextField(35);
        JTextField pfield2 = new JTextField(35);
        pfield1.setText(player1);
        pfield2.setText(player2);
        JTextField nfield1 = new JTextField(35);
        JTextField nfield2 = new JTextField(35);
        nfield1.setText(player1Name);
        nfield2.setText(player2Name);

        //create and set up the panel
        JPanel myPanel = new JPanel();
        myPanel.setLayout(new GridBagLayout());
        GridBagConstraints c = new GridBagConstraints();
        myPanel.add(new JLabel("Player 1 URL:    "));
        myPanel.add(pfield1);
        myPanel.add(new JLabel("Player 2 URL:  "));
        myPanel.add(pfield2);
        c.gridy = 1; //next row
        myPanel.add(new JLabel("Player 1 Name: "), c);
        myPanel.add(nfield1, c);
        myPanel.add(new JLabel("  Player 2 Name: "), c);
        myPanel.add(nfield2, c);

        //display the panel
        JOptionPane.showConfirmDialog(null, myPanel,
                "Please Enter Player Info", JOptionPane.DEFAULT_OPTION);

        //read user input
        player1 = pfield1.getText();
        player2 = pfield2.getText();
        player1Name = nfield1.getText();
        player2Name = nfield2.getText();

        //start the games
        HalmaGame [] tournament = {
            //new HalmaGame( tieURL, tieURL, tieName, tieName ),
            new HalmaGame( player1, player2, player1Name, player2Name ),
            new HalmaGame( collisionPlayer1, collisionPlayer2,
                    collision1Name, collision2Name )
        };
    }

}

class GameBoard extends OfficialObserver{

    @Override
    protected void handleUpdate(){
        if( !super.checkRecipient( MY_EMAIL ) )
            return;
        if(mTimer == TIMER_START)
            this.startGame();
        ALL_MOVES.add( super.getMessage() );
    }

    private static final Color
```

```
        TEAM_A_COLOR = new Color(204,0,153),
        TEAM_B_COLOR = new Color(0,102,153),
        TEXT_BGCOLOR = Color.white,
        TEXT_SELECTION_COLOR = Color.red;

    private static final int
        NUM_SPLITS = 2,
        BOARD_FRAME_WIDTH = (int) Toolkit.getDefaultToolkit().getScreenSize()
                .getWidth() / NUM_SPLITS,
        BOARD_FRAME_HEIGHT = (int) (Toolkit.getDefaultToolkit()
                .getScreenSize().getHeight() * 0.98),
        CELL_SIZE = BOARD_FRAME_WIDTH / 25,
        TIMER_START = 0;

    private static final Font
        FONT = new Font("Times New Roman", Font.BOLD, 20);

    private static final String
        MY_EMAIL = "g",
        TIMER = "Move: ",
        HALMATE = "HALMATE!  ",
        TEAM_A_WINS = "Red Team Victory!",
        TEAM_B_WINS = "Blue Team Victory!",
        TEAM_A_REPLACE = "Red",
        TEAM_B_REPLACE = "Blue",
        START_MESSAGE = "Click 'Step' or 'Run' to Continue | ",
        SPLIT_PHRASE = "SPLITSPLIT";

    private final HalmaWorld
        mWorld = new HalmaWorld(this);

    private static Integer numInstances;

    private final String
        mWorldMessage,
        mTeamA,
        mTeamB;

    private String mStart;

    private int
        mTimer;

    private final ArrayList<String>
        ALL_MOVES = new ArrayList<String>();

    public GameBoard(String teamA, String teamB){
        if(numInstances == null)
            numInstances = 0;
        else
            numInstances++;
        mTeamA = teamA;
        mTeamB = teamB;
        mWorldMessage = "Press \"Step\" to begin: " + teamA + " vs. " + teamB
            + "\n\nCheck internet connection. Starting move may be illegal.";
        mWorld.setMessage( mWorldMessage );
        mWorld.setGrid( new HalmaGrid("") );
        mWorld.show( BOARD_FRAME_WIDTH, BOARD_FRAME_HEIGHT );
        this.setTitle( mWorld, "HalmaWorld - " + teamA + " vs. " + teamB );
        this.centerWorldOnScreen( mWorld, numInstances);
        this.setTextArea( mWorld, FONT);
        mTimer = TIMER_START;
        this.setCellSize( mWorld, CELL_SIZE );
    }

    protected void startGame(){
        mStart = super.getMessage();
        drawBoard( mStart );
        mWorld.setMessage( START_MESSAGE + mWorld.getMessage() );
    }

    /**
     * The following methods are
     * derived from WorldFrame.java
     * or GridPanel.java
     */

    public static void setTitle(HalmaWorld inWorld, String title){
        inWorld.getFrame().setTitle( title );
    }

    public static void setCellSize(HalmaWorld inWorld, int size){
        inWorld.getFrame().getGridPanel().setCellSize( size );
    }

    public static void setZoom(HalmaWorld inWorld, double inFactor){
        inWorld.getFrame().getGridPanel().zoom(inFactor);
    }

    public static void setTextArea(HalmaWorld inWorld, Font inFont){
        JTextArea messageArea = inWorld.getFrame().getMessageArea();
        messageArea.setFont( inFont );
        messageArea.setEditable( true );
        messageArea.setFocusable( true );
        messageArea.setBackground( TEXT_BGCOLOR );
        messageArea.setSelectionColor( TEXT_SELECTION_COLOR );
    }

    private static void centerWorldOnScreen(HalmaWorld inWorld, int
            numInstances){
        inWorld.getFrame().setLocation( BOARD_FRAME_WIDTH * (numInstances %
                NUM_SPLITS ), 0 );
    }


    protected void stepAhead(){
        if( mTimer < ALL_MOVES.size() && !mWorld.getMessage().substring(0,
                HALMATE.length()).equals(HALMATE))
            drawBoard( ALL_MOVES.get( mTimer ) );
    }

    protected void rewindMove(){
        if( mTimer < TIMER_START + 2)
            return;
        mTimer-=2;
        clearBoard( mWorld );
        clearFlowers( mWorld );
        drawBoard( ALL_MOVES.get( mTimer ) );
    }

    protected void restartGame(){
        mTimer = TIMER_START;
        clearBoard( mWorld );
        clearFlowers( mWorld );
        drawBoard( mStart );
    }

    @Override
    public boolean equals(Object o){
        boolean out = super.equals(o);
        if( ALL_MOVES.size() <= 0 )
            return out;
        if( "step".equals( o ) )
            this.stepAhead();
        if( "restart".equals( o ) )
            this.restartGame();
        if( "rewind".equals( o ) )
            this.rewindMove();
        return out;
    }

    private static ArrayList<Piece> toPieceList(String officialData, boolean
            isPlayerMove){
        ArrayList<Piece> list = new ArrayList<Piece>();
        JsonArray array;
        try{ array = JsonParser.array().from(officialData);  }
        catch(JsonParserException e){ return null; }
        if (!isPlayerMove){
            int offset = 4;
            for(int k = 0; k < array.size(); k += offset)
                list.add( new Piece(
                    array.getInt(k),
                    array.getInt(k + 1),
                    array.getInt(k + 2),
                    array.getInt(k + 3)
                ) );
        }
        else{
            int offset = 2;
            list.add( new Piece(    //from piece
                    array.getInt(0),
                    array.getInt(1),
                    0,
                    0
                ) );
            for(int k = 3; k < array.size(); k += offset) //jumps
                list.add( new Piece(
                    array.getInt(k),
                    array.getInt(k + 1),
                    0,
                    0
                ) );
        }
        return list;
    }

    public static void highlightDestinations( HalmaWorld world ){
        for(int x = 0; x < 3; x++){
            for(int y = 0; y < 3; y++){
                Glitter g = new Glitter();
                g.setColor( TEAM_B_COLOR );
                world.add(new Location( y, x ), g);
            }
        }
        for(int row = 0; row < 3; row++){
            for(int col = BOARD_SIZE - 1; col >= BOARD_SIZE - 3; col--){
                Glitter g = new Glitter();
                g.setColor( TEAM_A_COLOR );
                world.add(new Location( row , col ), g);
            }
        }
    }

    public static void clearFlowers( HalmaWorld world ){
        for(int x = 0; x < BOARD_SIZE; x++){
            for(int y = 0; y < BOARD_SIZE; y++){
                Object obj = world.getGrid().get( new Location(y,x) );
                if(obj instanceof Flower){
                    world.remove( new Location(y,x) );
                }
            }
        }
    }

    public static void clearBoard( HalmaWorld world ){
        for(int x = 0; x < BOARD_SIZE; x++){
            for(int y = 0; y < BOARD_SIZE; y++){
                Object obj = world.remove( new Location(y,x) );
                if(obj instanceof Piece){
                    Piece p = (Piece) obj;
                    Flower a = new Flower();
                    a.setColor( p.getColor() );
                    world.add(new Location(y,x), a);
                }
            }
        }
    }

    //Determine the winner by counting the number of remaining highlighted
    //victory locations
    public static int getWinner( HalmaWorld world, Object marker ){
        Grid grid = world.getGrid();
        int blues = 0, reds = 0;
        for(int x = 0; x < grid.getNumCols(); x++){
            for(int y = 0; y < grid.getNumRows(); y++){
                Object o = grid.get( new Location(y, x) );
                if( o != null && marker.getClass().equals( o.getClass() )
                        && x < 3)
                    blues++;
                else if( o != null && marker.getClass().equals( o.getClass()
                        ) && x > 3)
                    reds++;
            }
        }

        if(reds == 0 && blues == 0)
            return 3;
        if(reds == 0)
            return 1;
        if(blues == 0)
            return 2;
        return 0;
    }

    private String upTimer(){
        mTimer++;
        return "" + mTimer;
    }

    private static Location getToLocation(String move){
        ArrayList<Location> moveLocs = toLocationList(move);
        Location target = moveLocs.get( moveLocs.size() - 1 );
        return new Location(target.getRow(), target.getCol());
    }

    private static void addToPieces(String team1Move, String team2Move,
            HalmaWorld world, boolean collision0, boolean collision1){
        Location
            redLoc = getToLocation( team1Move ),
            blueLoc = getToLocation( team2Move );
        XPiece
            redPiece = new XPiece(),
            bluePiece = new XPiece();
        redPiece.setColor( TEAM_A_COLOR );
        bluePiece.setColor( TEAM_B_COLOR );

        //don't draw the pieces if there was a collision, so the damaged
        //piece is drawn instead
        if (collision0 == false) world.add(redLoc, redPiece);
        if (collision1 == false) world.add(blueLoc, bluePiece);
    }

    private static ArrayList<Location> toLocationList(String move){
        JsonArray array;
        ArrayList<Location> locs = new ArrayList<Location>();
        try{ array = JsonParser.array().from(move); }
        catch(JsonParserException e){
            return null;
        }
        int x;
        ArrayList<Integer> coordList = new ArrayList<Integer>();
        for(int k = 0; k < array.size(); k++)
            coordList.add( array.getInt(k)  );
        Iterator<Integer> itr = coordList.iterator();
        if( !itr.hasNext() )
```

```
            return locs;
        x = itr.next();
        locs.add( new Location(itr.next(), x) );
        itr.next(); //skip damage
        while(itr.hasNext()){
            x = itr.next();
            locs.add( new Location(itr.next(), x) );
        }
        return locs;
    }

    private static String formatMove(String move){
        JsonArray array;
        try{
            array = JsonParser.array().from(move);
        }
        catch(JsonParserException e){
            return move;
        }
        int x;
        ArrayList<Integer> coordList = new ArrayList<Integer>();
        for(int k = 0; k < array.size(); k++)
            coordList.add( array.getInt(k)  );
        Iterator<Integer> itr = coordList.iterator();
        ArrayList<Location> locs = new ArrayList<Location>();

        x = itr.next();
        locs.add( new Location(itr.next(), x) );
        itr.next(); //skip damage
        while(itr.hasNext()){
            x = itr.next();
            locs.add( new Location(itr.next(), x) );
        }
        return locs.toString();
    }

    public static Piece createDamagedPiece(int damage, Color color){
        Piece [] damageCounts ={
            new One(),
            new Two(),
            new Three(),
            new Four(),
            new Five()
        };
        if(damage < 5)
            damageCounts[ damage - 1 ].setColor(color);
        return damageCounts[ damage - 1 ];
    }

    /*
     * clears board, highlights destinations, declares move/winner
     */
    protected void drawBoard(String inData){
        String onMessageField, p1Move, p2Move, pieceStr;
        int winner;
        ArrayList<Piece> pieces;
        String [] data = inData.split( SPLIT_PHRASE );
```

```
pieceStr = data[0];
boolean isValid = (pieceStr.charAt(0) == 'a');
char invalidPlayer = pieceStr.charAt(0);
pieceStr = pieceStr.substring(1);
p1Move = data[1];
p2Move = data[2];

onMessageField = TIMER + upTimer() + "\n" + mTeamA + ": "
    + formatMove(p1Move) + "\n" + mTeamB + ": " + formatMove(p2Move);

Location
    final0 = new Location(-1, -1),
    final1 = new Location(-1, -1);
if (isValid){
    //add player 1 move track
    pieces = toPieceList( p1Move, true ) ;
    for (Piece p : pieces){
        p.setColor( TEAM_A_COLOR );
        mWorld.add(p.getXYLocation(), p);
        final0 = p.getXYLocation();
    }

    //add player 2 move track
    pieces = toPieceList( p2Move, true ) ;
    for (Piece p : pieces){
        p.setColor( TEAM_B_COLOR );
        mWorld.add(p.getXYLocation(), p);
        final1 = p.getXYLocation();
    }
}

clearBoard( mWorld );
highlightDestinations( mWorld );

//add all the pieces
pieces = toPieceList( pieceStr, false ) ;
print( pieces.toString() );
boolean collision0 = false, collision1 = false;
boolean skipPiece; //because we already displayed a piece there with
                   //higher damage
for (Piece p : pieces){
    //make sure the pieces are the correct team color
    skipPiece = false;
    if(p.getTeam() == 0){
        p.setColor( TEAM_A_COLOR );
        if(p.getDamage() == 5){
            if (final0.equals(p.getXYLocation()))
                collision0 = true;
            else
                collision1 = true;
        }
    }
    else{
        p.setColor( TEAM_B_COLOR );
        if(p.getDamage() == 5){
            if (final1.equals(p.getXYLocation()))
                collision1 = true;
```

```
                    else
                        collision0 = true;
                }
            }

            //color overlapping pieces black
            if (mWorld.getGrid().get(p.getXYLocation()) instanceof Piece){
                p.setColor("black");
                if (p.getDamage() == 0){
                    skipPiece = true;
                    mWorld.getGrid().get(p.getXYLocation())
                            .setColor(Color.BLACK);
                }
            }

            //draw the pieces
            if (!skipPiece){
                if(p.getDamage() > 0)
                    mWorld.add(p.getXYLocation(), this.createDamagedPiece(
                            p.getDamage(), p.getColor() ));
                else
                    mWorld.add(p.getXYLocation(), p);
            }
        }//end for loop

        //check if the moves were valid, and display the moved pieces if so
        if (isValid)
            addToPieces(p1Move, p2Move, mWorld, collision0, collision1);
        else{
            if (invalidPlayer == '0' || invalidPlayer == '2') onMessageField
                    = "Invalid Move by " + mTeamA + " | " + onMessageField;
            if (invalidPlayer == '1' || invalidPlayer == '2') onMessageField
                    = "Invalid Move by " + mTeamB + " | " + onMessageField;
        }

        //check for victory
        winner = getWinner( mWorld, new Glitter() );
        if( winner == 1)
            onMessageField = HALMATE + TEAM_A_WINS.replace( TEAM_A_REPLACE ,
                    mTeamA);
        else if( winner == 2 )
            onMessageField = HALMATE + TEAM_B_WINS.replace( TEAM_B_REPLACE ,
                    mTeamB);
        else if( winner == 3 ) //tie
            onMessageField = HALMATE + "It's a tie!";

        mWorld.setMessage( onMessageField );
    }

}
```

HalmaGame

```
package ShowNTell;

/**
 * HalmaGame
 * Instantiates the components of the game, and runs the game.
 */

import java.util.*;

public class HalmaGame extends Thread{

    private final Official o;

    public HalmaGame(String url1, String url2, String name1, String name2){
        o = new Official();
        OfficialObserver [] array =
        {
            new HalmaMessenger( url1, url2 ),
            new CollisionAnalyst( new MoveValidator() ),
            new GameBoard( name1, name2 )
        };
        for( Observer keeper : array )
            o.addObserver(keeper);
        this.start();
    }

    @Override
    public void run(){
        o.startGame();
    }

}
```

Official

```
package ShowNTell;

/**
 * Official
 * Manages the game. Relays communication between the components of the game.
 */

import java.util.*;

public class Official extends Observable{

    private String mBoard, mMove;
    private int mCount;
    private boolean VICTORY;
    private static final double
        DELAY_DEFAULT = 0,
        RUN_COUNT = 200; //maximum moves before aborting game

    public static final int
        BOARD_SIZE = 18;

    private static final String
        SPLIT_PHRASE = "SPLITSPLIT",
        SUPER_SPLIT = "SPLITSPLITSPLIT",
        AI_RELAY = "m",
        COLLISIONS = "c",
        GRID = "g";

    public Official(){
        mBoard = getDefaultStartBoard();
        mCount = 0;
        VICTORY = false;
    }

    public void setVictory(boolean vict){
        VICTORY = vict;
    }

    public String getDefaultStartBoard(){
        ArrayList<Integer> iBoard = new ArrayList<>();
        //build teams
        int size = BOARD_SIZE;
        for(int x = 0; x < 3; x++){
            for(int y = 0; y < 4; y++){
                Piece red = new Piece(x, size-1-y, 0, 0);
                Piece blue = new Piece(size-1-x, size-1-y, 0, 1);
                iBoard.addAll( red.toIntList() );
                iBoard.addAll( blue.toIntList() );
            }
        }
        return iBoard.toString();
    }

    public void startBoard(String board){
        mBoard = board;
    }
```

```
public void startGame(){
    getRemoteAIMoves( mBoard );
}

//tell the Halma Messenger to request moves
private Official getRemoteAIMoves(String inBoard){
    send(AI_RELAY, inBoard);
    return this;
}

private Official setBoard(String inBoard){
    mBoard = inBoard;
    mCount++;
    return this;
}

public Official delay(double seconds){
    try{
        Thread.sleep( (int) (seconds * 1000) );
    }
    catch(InterruptedException e){}
    return this;
}

private Official output(String message){
    System.out.println(message);
    return this;
}

private Official setMove(String inMove){
    mMove = inMove;
    return this;
}

//receive a reply from an observer, and act accordingly
public void reply(String sender, String message){
    if( AI_RELAY.equals(sender) )
        output("From M: " + message)
        .setMove(message)
        .send( COLLISIONS , composeForCollisions(message) );
    else if ( COLLISIONS.equals(sender) && mCount < RUN_COUNT){
        output("From C: " + message)
        .setBoard(message.substring(1))
        .send( GRID, composeForGameBoard(message, mMove))
        .delay(DELAY_DEFAULT);
        if (!VICTORY) this.getRemoteAIMoves( message.substring(1) );
    }
}

private static String concat(String inFront, String inTail){
    return inFront + SPLIT_PHRASE + inTail;
}

private String composeForGameBoard(String inBoard, String inMoves){
    return concat(inBoard, inMoves);
}
```

```
    private String composeForCollisions(String AIMoves){
        return concat(mBoard, AIMoves);
    }

    protected Official send(String recipient, String message){
        setChanged();
        notifyObservers(recipient + SUPER_SPLIT + message);
        return this;
    }

}
```

OfficialObserver

```
package ShowNTell;

/**
 * OfficialObserver
 * Abstract class, representing a component of the game.
 * Implements communication between the Official and OfficialObservers
 */

import java.util.*;

public abstract class OfficialObserver implements Observer{

    protected abstract void handleUpdate();

    protected static final int BOARD_SIZE = Official.BOARD_SIZE;

    protected static boolean VICTORY = false;

    private static final String
        SPLIT_PHRASE = "SPLITSPLITSPLIT";
    private Official m_official;
    private String m_message, m_recipient;

    @Override
    public void update(Observable o, Object arg){
        if(arg instanceof String && o instanceof Official)
            processOfficialUpdate( (Official) o, arg.toString() );
    }

    private void processOfficialUpdate( Official o, String arg ){
        String[] parts = arg.split(SPLIT_PHRASE);
        if(parts.length != 2) return;
        m_official = (Official) o;
        m_recipient = parts[0];
        m_message = parts[1];
        handleUpdate();
        o.setVictory(VICTORY);
    }

    public static void print(String message){
        System.out.println(message);
    }

    protected void replyToOfficial(String sender, String message){
        m_official.reply(sender, message);
    }

    protected Official getOfficial(){
        return m_official;
    }

    protected String getMessageRecipient(){
        return m_recipient;
    }

    protected String getMessage(){
        return m_message;
    }

    protected boolean checkRecipient( String inCode ){
        return inCode.equals(m_recipient);
    }
}
```

HalmaMessenger

```
package ShowNTell;

/**
 * HalmaMessenger
 * Sends and receives messages to/from the AIs that are playing the game.
 */

import com.grack.nanojson.*;
import java.net.*;
import java.util.*;
import java.io.BufferedReader;
import java.io.DataOutputStream;
import java.io.IOException;
import java.io.InputStreamReader;
import java.net.HttpURLConnection;
import java.net.URL;
import java.util.logging.Level;
import java.util.logging.Logger;

public class HalmaMessenger extends OfficialObserver{

    @Override
    //called whenever an update is received from the observable
    public void handleUpdate(){
        if( super.checkRecipient( MY_EMAIL ) )
            super.replyToOfficial( MY_EMAIL , respondWithAIMoves(
                    super.getMessage() ) );
    }

    private static final String
        MY_EMAIL = "m",
        ROW_INDEX = "y",
        COLUMN_INDEX = "x",
        DAMAGE_INDEX = "damage",
        FROM_KEY = "from",
        TO_KEY = "to";

    private final String m_url1, m_url2;

    public HalmaMessenger(String inPlayer1addy, String inPlayer2addy){
        m_url1 = inPlayer1addy;
        m_url2 = inPlayer2addy;
    }

    private static ArrayList<String> toJSONList(String [] jsons){
        ArrayList<String> list = new ArrayList<String>();
        for(String str : jsons)
            list.add(str);
        return list;
    }

    private static String concat(String a, String b){
        return a + "SPLITSPLIT" + b;
    }

    private String respondWithAIMoves(String message){
```

```
    ArrayList<String> moves = getRemoteAIMoves(message);
    Iterator<String> jsons = moves.iterator();
    try{
        return concat(toSequence(jsons.next()), toSequence(
                jsons.next()) );
    }
    catch( NullPointerException n){
        return n.toString();
    }
}

public ArrayList<String> getRemoteAIMoves(String message){
    String [] moveArray =
        {
            getRemoteData(m_url1, message, 0),
            getRemoteData(m_url2, message, 1)
        };
    return toJSONList(moveArray);
}

public static String toSequence(String json){
    ArrayList<Integer> sequence = new ArrayList<Integer>();
    JsonObject obj;
    try{ obj = JsonParser.object().from(json); }
    catch(JsonParserException e){ return ""; }
    JsonObject fromObj = obj.getObject( FROM_KEY );
    JsonArray toArray = obj.getArray( TO_KEY );

    if (fromObj == null){
        sequence.add(-1);
        sequence.add(-1);
        sequence.add(-1);
    }
    else{
        int fromRow = fromObj.getInt( COLUMN_INDEX );
        int fromColumn = fromObj.getInt( ROW_INDEX );
        int fromDamage = fromObj.getInt( DAMAGE_INDEX );
        sequence.add(fromRow);
        sequence.add(fromColumn);
        sequence.add(fromDamage);
    }

    if (toArray == null){
        sequence.add(-1);
        sequence.add(-1);
    }
    else{
        for(Object o : toArray){
            obj = (JsonObject) o;
            if (obj == null){
                sequence.add(-1);
                sequence.add(-1);
            }
            else{
                sequence.add( obj.getInt( COLUMN_INDEX ) );
                sequence.add( obj.getInt( ROW_INDEX ) );
            }
        }
    }

    return sequence.toString();
}

//send JSON as a POST request to an AI and receive a JSON response
public static String getRemoteData(String address, String board, int
        playerNum){
    try {
        URL obj = new URL(address);
        HttpURLConnection con = (HttpURLConnection) obj.openConnection();

        con.setRequestMethod("POST");
        con.setRequestProperty("User-Agent", "Mozilla/5.0");
        con.setRequestProperty("Accept-Language", "en-US,en;q=0.5");

        //board data
        ArrayList<XYDLocation> boardList =
                CollisionAnalyst.getXYDList(board);
        String urlParameters = convertBoardToJSON(boardList, playerNum);
        print("From Messenger to AI: " + urlParameters);

        // Send post request
        con.setDoOutput(true);
        DataOutputStream wr = new
                DataOutputStream(con.getOutputStream());
        wr.writeBytes(urlParameters);
        wr.flush();
        wr.close();

        BufferedReader in = new BufferedReader(
                new InputStreamReader(con.getInputStream()));
        String inputLine;
        StringBuilder response = new StringBuilder();

        while ((inputLine = in.readLine()) != null)
            response.append(inputLine);
        in.close();
        print("AI Response: " + response.toString());

        return response.toString();

    } catch (MalformedURLException ex) {
        Logger.getLogger(HalmaMessenger.class.getName()).log(
                Level.SEVERE, null, ex);
    } catch (IOException ex) {
        Logger.getLogger(HalmaMessenger.class.getName()).log(
                Level.SEVERE, null, ex);
    }
    return "";
}

private static JsonObject toJSONObj(XYDLocation piece){
    try{
        return JsonParser.object().from( piece.toJSONString() );
    } catch(JsonParserException ex){
        Logger.getLogger(HalmaMessenger.class.getName()).log(
                Level.SEVERE, null, ex);
        return null;
    }
}

private static JsonObject toJSONObj(int x, int y){
    String json = JsonWriter.string()
        .object()
        .value("x", x)
        .value("y", y)
        .end()
        .done();
    try{
        return JsonParser.object().from(json);
    }
    catch(JsonParserException ex){
        Logger.getLogger(HalmaMessenger.class.getName()).log(
                Level.SEVERE, null, ex);
        return null;
    }
}

private static JsonStringWriter range(JsonStringWriter writer, int xMin,
        int xMax, int yMin, int yMax){
    for(int x = xMin; x <= xMax; x++)
        for(int y = yMin; y <= yMax; y++)
            writer = writer.value( toJSONObj(x, y) );
    return writer;
}

private static String convertBoardToJSON(ArrayList<XYDLocation>
        boardList, int playerNum){
    JsonStringWriter writer = JsonWriter.string().object()
        .value("boardSize", 18)
        .array("pieces");
    for (XYDLocation piece : boardList)
        if (piece.getTeam() == playerNum)
            writer = writer.value( toJSONObj( piece ) );

    writer = writer.end()
        .array("enemy");
    for (XYDLocation piece : boardList)
        if (piece.getTeam() != playerNum)
            writer = writer.value( toJSONObj( piece ) );

    writer = writer.end()
        .array("destinations");
    switch(playerNum){
        case 0:
        writer = range(writer, 17, 17, 0, 2);
        writer = range(writer, 16, 16, 0, 2);
        writer = range(writer, 15, 15, 0, 2);
        writer = writer.end().array("enemydestinations");
        writer = range(writer, 0, 2, 0, 2);
        writer = writer.end();
        break;
        default:
        writer = range(writer, 0, 2, 0, 2);
        writer = writer.end().array("enemydestinations");
        writer = range(writer, 17, 17, 0, 2);
        writer = range(writer, 16, 16, 0, 2);
        writer = range(writer, 15, 15, 0, 2);
        writer = writer.end();
    }//end playerNum switch
    return writer.end().done();
}
}
```

CollisionAnalyst

```
package ShowNTell;

/**
 * CollisionAnalyst
 * Enforces collisions between pieces. Also validates that moves follow the
 * rules.
 */

import java.util.*;

public class CollisionAnalyst extends OfficialObserver{

    private final Object VALIDATOR;

    public CollisionAnalyst( Object inValidator ){
        super();
        if(inValidator == null)
            VALIDATOR = false;
        else
            VALIDATOR = inValidator;
    }

    @Override
    //called whenever an update is received from the observable
    protected void handleUpdate(){
        if( !super.checkRecipient( MY_EMAIL ) )
            return;
        super.replyToOfficial( MY_EMAIL, getNewBoardPosition(
                super.getMessage() ) );
    }

    //|||||||||||||MEMBER DATA||||||||||||
    private static final String
        SPLIT_PHRASE = "SPLITSPLIT",
        MY_EMAIL = "c";
    private static final int
        DAMAGE_START = 5,
        DAMAGE_LITE = 5;
    //||||||||||||||||||||||||||||||||||||

    public String getNewBoardPosition(String twoPlayerMoveData){
        ArrayList<String> playerMoves = new ArrayList<String>();
        String [] data = toStrArray( twoPlayerMoveData );
        String board = data[0];
        playerMoves.add(data[1]);
        playerMoves.add(data[2]);
        String outBoard = getNewPieceData( board, playerMoves );
        return outBoard;
    }

    private static String [] toStrArray(String multiData){
        return multiData.replace(" ", "").split(SPLIT_PHRASE);
    }

    private static boolean isOwnCollision(Location toLoc0, Location toLoc1,
            XYDLocation xyd){
```

```
    return (xyd.equals(toLoc0) && xyd.getTeam() == 0)
            || (xyd.equals(toLoc1) && xyd.getTeam() == 1);
}

private static boolean isEnemyCollision(Location toLoc0, Location toLoc1,
        XYDLocation xyd){
    return (xyd.equals(toLoc0) && xyd.getTeam() == 1)
            || (xyd.equals(toLoc1) && xyd.getTeam() == 0);
}

/*
 * returns new board piece locations
 * WARNING: only supports 2 player game
 */
public String getNewPieceData( String oldBoard, ArrayList<String>
        movesList ){
    Location
        fromLoc0 = getFromLocation( movesList.get(0) ),
        fromLoc1 = getFromLocation( movesList.get(1) ),
        toLoc0 = getToLocation( movesList.get(0) ),
        toLoc1 = getToLocation( movesList.get(1) );
    ArrayList<XYDLocation>
        nextBoard = getXYDList(oldBoard);
    ArrayList<Location>
        toLocArray0 = getToLocationArray( movesList.get(0) ),
        toLocArray1 = getToLocationArray( movesList.get(1) );

    //Verify move is valid
    Integer damage0 = toIntArray(movesList.get(0))[2];
    Integer damage1 = toIntArray(movesList.get(1))[2];
    ArrayList<Object> params = new ArrayList<Object> ();
    Object [] test = {  damage0, fromLoc0, 0, toLocArray0, nextBoard,
            BOARD_SIZE };
    for(Object param : test)
        params.add(param);
    ArrayList<Object> params2 = new ArrayList<Object> ();
    Object [] test2 = { damage1, fromLoc1, 1, toLocArray1, nextBoard,
            BOARD_SIZE };
    for(Object param : test2)
        params2.add(param);
    boolean isValid0 = !VALIDATOR.equals( params );
    boolean isValid1 = !VALIDATOR.equals( params2 );
    if (!isValid0 && !isValid1){
        return "2"+nextBoard.toString().replace(" ", "");
    }
    if (!isValid0){
        return "0"+nextBoard.toString().replace(" ", "");
    }
    if (!isValid1){
        return "1"+nextBoard.toString().replace(" ", "");
    }

    //Check if there was a collision and update the board
    boolean isHeadOnCollision = toLoc0.equals(toLoc1);
    boolean movedPiece0 = false, movedPiece1 = false;
    for( XYDLocation xyd : nextBoard ){
        if(!isHeadOnCollision){
            if( !movedPiece0 && xyd.equals( fromLoc0, 0, damage0) ){
                xyd.setXY( toLoc0 );
                movedPiece0 = true;
            }
            else if( !movedPiece1 && xyd.equals(fromLoc1, 1, damage1) ){
                xyd.setXY( toLoc1 );
                movedPiece1 = true;
            }
            else if( isOwnCollision( toLoc0, toLoc1, xyd ) )
                xyd.setD( DAMAGE_START + 1);
            else if( isEnemyCollision( toLoc0, toLoc1, xyd ) )
                xyd.setD( DAMAGE_LITE + 1);
        }
        else{
            if( !movedPiece0 && xyd.equals( fromLoc0, 0, damage0 ) ){
                xyd.setXYD( toLoc0, DAMAGE_START + 1);
                movedPiece0 = true;
            }
            else if( !movedPiece1 && xyd.equals(fromLoc1, 1, damage1) ){
                xyd.setXYD( toLoc1, DAMAGE_START + 1);
                movedPiece1 = true;
            }
        }
        xyd.heal(); //heals all pieces
    }

    //of superclass
    VICTORY = checkVictory(nextBoard);

    return "a"+nextBoard.toString().replace(" ", "");
}

private static boolean checkVictory(ArrayList<XYDLocation> nextBoard){
    boolean isAtFinish;
    boolean redVict = true, blueVict = true;

    //create lists of where pieces will be if someone has won
    ArrayList<XYDLocation> redVictory = new ArrayList<>();
    for (int x = BOARD_SIZE-3; x < BOARD_SIZE; x++){
        for (int y = 0; y < 3; y++){
            redVictory.add(new XYDLocation(x, y, 0, 0));
        }
    }
    ArrayList<XYDLocation> blueVictory = new ArrayList<>();
    for (int x = 0; x < 3; x++){
        for (int y = 0; y < 3; y++){
            blueVictory.add(new XYDLocation(x, y, 0, 0));
        }
    }

    //compare those lists to the current board
    for ( XYDLocation victory : redVictory ){
        isAtFinish = false;
        for ( XYDLocation xyd : nextBoard ){
            if (xyd.getX() == victory.getX() && xyd.getY() ==
                    victory.getY()){
                isAtFinish = true;
                break;
            }
        }
        if (isAtFinish == false){
            redVict = false;
            break;
        }
    }
    for ( XYDLocation victory : blueVictory ){
        isAtFinish = false;
        for ( XYDLocation xyd : nextBoard ){
            if (xyd.getX() == victory.getX() && xyd.getY() ==
                    victory.getY()){
                isAtFinish = true;
                break;
            }
        }
        if (isAtFinish == false){
            blueVict = false;
            break;
        }
    }
    return (redVict || blueVict);
}

private static ArrayList<Integer> toIntList(int [] coords){
    ArrayList<Integer> coordList = new ArrayList<Integer>();
    for(int coordinate : coords)
        coordList.add( coordinate );
    return coordList;
}

public static ArrayList<XYDLocation> getXYDList( String inBoard ){
    ArrayList<Integer> coordList = toIntList( toIntArray(inBoard) );
    ArrayList<XYDLocation> xydlist = new ArrayList<XYDLocation>();
    Iterator<Integer> itr = coordList.iterator();
    while( itr.hasNext() )
        xydlist.add( new XYDLocation( itr.next(), itr.next(), itr.next(),
                itr.next() ) );
    return xydlist;
}

public static int [] toIntArray(String inStr){
    String [] nums
        = inStr.replace("[", "").replace("]", "")
            .replace(" ", "").split(",");
    ArrayList<Integer>coords = new ArrayList<Integer>();
    for(String num : nums)
        coords.add(Integer.parseInt(num, 10));
    int [] outArray = new int[coords.size()];
    Iterator <Integer> itr = coords.iterator();
    for(int k = 0; k < outArray.length; k++)
        outArray[k] = itr.next();
    return outArray;
}

public static Location getFromLocation(String move){
    int [] moveArray = toIntArray( move );
    return new Location( moveArray[1] , moveArray[0]);
}

public static Location getToLocation(String move){
    int [] moveArray = toIntArray( move );
    return new Location( moveArray[moveArray.length - 1] ,
            moveArray[moveArray.length - 2]);
}

public static ArrayList<Location> getToLocationArray(String move){
    int [] moveArray = toIntArray( move );
    ArrayList<Location> moveArrayList = new ArrayList<>();
    for (int i = 3; i < moveArray.length; i+=2){
        moveArrayList.add(new Location(moveArray[i+1], moveArray[i]));
    }
    return moveArrayList;
}
}
```

MoveValidator

```
package ShowNTell;

/**
 * Enforces halma game rules
 * Based on the JavaScript version by Dr. Coyle
 */

import java.util.*;

public class MoveValidator{

    private static final int IN_COUNT = 6;

    /* entry point for move validator */
    @Override
    public boolean equals( Object o ){
        if( isValidInput( o ) )
            return processInput( o );
        else{
            System.out.println("Validator Bug!");
        }
        return super.equals(o);
    }

    private boolean processInput( Object o ){
        ArrayList<Object> list = ( ArrayList<Object> ) o;
        Iterator<Object>itr = list.iterator();
        return !isValidMoveRequest((int) itr.next(), (Location) itr.next(),
                (int) itr.next(), (ArrayList<Location>) itr.next(),
                (ArrayList<XYDLocation>) itr.next(), (int) itr.next() );
    }

    public boolean isValidInput( Object o ){
        if(o instanceof ArrayList == false)
            return false;
        ArrayList list = ( ArrayList ) o;
        if( list.size() != IN_COUNT )
            return false;
        Iterator itr = list.iterator();
        Object [] checkTypes = {
            new Integer(5),
            new Location(0,0),
            new Integer(5),
            new ArrayList(),
            new ArrayList(),
            new Integer(5)
        };
        for( Object obj : checkTypes ){
            if( !itr.next().getClass().isAssignableFrom( obj.getClass() ) )
                return false;
        }
        return true;
    }
    private static boolean isThereAPieceBetween(Location cell1, Location
            cell2, ArrayList<XYDLocation> gPieces) {
        /* note: assumes cell1 and cell2 are 2 squares away
         either vertically, horizontally, or diagonally */
        int rowBetween = (cell1.getRow() + cell2.getRow()) / 2;
        int columnBetween = (cell1.getCol() + cell2.getCol()) / 2;
        for (int i = 0; i < gPieces.size(); i++) {
            if ((gPieces.get(i).getY() == rowBetween) &&
                    (gPieces.get(i).getX() == columnBetween)) {
                return true;
            }
        }
        return false;
    }

    private static boolean isOneSpaceAway(Location c1, Location c2) {
        int diffx = Math.abs(c1.getCol() - c2.getCol());
        int diffy = Math.abs(c1.getRow() - c2.getRow());
        int diffxy = diffx + diffy;
        if (diffxy == 1) return true;  // x y axis
        if (diffx==1 && diffy==1) return true; // diagonal
        return false;  // not linear or diagonal
    }

    private static boolean isTwoSpacesAway(Location c1, Location c2) {
        int diffx = Math.abs(c1.getCol() - c2.getCol());
        int diffy = Math.abs(c1.getRow() - c2.getRow());
        // check x and y
        if  ((diffx == 2 && diffy == 0) || (diffx == 0 && diffy == 2)  )
            return true;  // x y axis
        // check diagonal
        if (diffx==2 && diffy==2)
            return true;
        return false;  // not linear or diagonal
    }

    // checks that src & dest are one cell apart and dest is free
    private static boolean isLegalOneSquareMove(Location src, Location dest,
            ArrayList<XYDLocation> gPiecesArr) {
        return isOneSpaceAway(src,dest);
    }

    // checks that 1) src & dest are two cells apart 2) dest is free
    //             3) there exists a piece between src and dest
    private static boolean isLegalTwoSquareJump(int damage, Location src,
            Location dest, ArrayList<XYDLocation> gPiecesArr) {
        return (isTwoSpacesAway(src,dest) &&
                isThereAPieceBetween(src, dest, gPiecesArr) &&
                damage == 0);
    }

    // jumpArr will have original source piece followed by jump locations
    private static boolean isArrayOfValidJumps(int damage, Location src,
            ArrayList<Location> jumpArr, ArrayList<XYDLocation> gPiecesArr) {
        // add src cell to array
        jumpArr.add(0, src);
        while (jumpArr.size() > 1) {
            // check first two cells for jump
            if ( !isLegalTwoSquareJump(damage, jumpArr.get(0),
                    jumpArr.get(1), gPiecesArr) ) {
                System.out.print("Illegal jump from " +
                    jumpArr.get(0).toString() + " to: " +
                    jumpArr.get(1).toString());
                return false;
            }
            // remove first jump
            jumpArr.remove(0);
        }
        // all valid jumps
        return true;
    }

    // checks if piece is holding its position
    private static boolean isPieceHoldingPosition(Location src, Location
            dest) {
        return (src.getCol() == dest.getCol() && src.getRow() ==
                dest.getRow());
    }

    //checks if a piece is at a location
    private static boolean isPieceAt(Location src, int team, int damage,
            ArrayList<XYDLocation> gPieces){
        for (XYDLocation piece : gPieces){
            if (piece.getX() == src.getCol() && piece.getY() == src.getRow()
                    && team == piece.getTeam() && damage == piece.getD())
                return true;
        }
        return false;
    }

    // checks that array of requested moves is valid.
    // if only one move in array, check either non-jump or one jump
    // else check if all move pairs are jumping over some piece
    private static boolean isValidMoveRequest(int damage, Location src, int
            team, ArrayList<Location> moveArr, ArrayList<XYDLocation>
            gPieces, int BOARD_SIZE) {
        if(moveArr.isEmpty())
            return false;

        //ensure there is a piece at the location we are trying to move
        if(!isPieceAt(src, team, damage, gPieces))
            return false;

        //check if the AI tries to move outside the board
        Location finalMove = moveArr.get(moveArr.size()-1);
        if (finalMove.getCol() >= BOARD_SIZE || finalMove.getCol() < 0 ||
                finalMove.getRow() >= BOARD_SIZE || finalMove.getRow() < 0)
            return false;

        //check that a single move is valid
        if(moveArr.size() == 1) {
            Location dest = moveArr.get(0);  // only one
            return (isLegalOneSquareMove(src, dest, gPieces)  ||
                isLegalTwoSquareJump(damage, src, dest, gPieces)  ||
                isPieceHoldingPosition(src,dest) );
        }

        //check that a jump-chain is valid
        return isArrayOfValidJumps(damage, src, moveArr, gPieces);
    }
}
```

XYDLocation

```
package ShowNTell;

/**
 * XYDLocation
 * Represents a piece of the game board, composing of x and y coordinates,
 * damage, and a team.
 */

import com.grack.nanojson.*;

public class XYDLocation{

    int mDamage;
    Location mLoc;
    int mTeam;

    public XYDLocation(int x, int y, int d, int t){
        mLoc = new Location( y , x );
        mDamage = d;
        mTeam = t;
    }

    public XYDLocation setTeam(int t){
        mTeam = t;
        return this;
    }

    public int getTeam(){
        return mTeam;
    }

    public XYDLocation heal(){
        if (mDamage > 0)
            mDamage--;
        return this;
    }

    public XYDLocation setD(int d){
        mDamage = d;
        return this;
    }

    public XYDLocation setXYD(Location moveLoc, int d){
        setD(d);
        setXY(moveLoc);
        return this;
    }
    public XYDLocation setXY(Location moveLoc){
        setXY(moveLoc.getCol(), moveLoc.getRow());
        return this;
    }

    public XYDLocation setXY(int x, int y){
        setX(x);
        setY(y);
        return this;
    }

    public void setX(int x){
        int y = getY();
        mLoc = new Location(y, x);
    }

    public void setY(int y){
        int x = getX();
        mLoc = new Location(y, x);
    }

    public int getX(){
        return mLoc.getCol();
    }

    public int getY(){
        return mLoc.getRow();
    }

    public int getD(){
        return mDamage;
    }

    public boolean equals( Location other  ){
        return mLoc.equals( other );
    }

    public boolean equals(Location other, int otherTeam){
        return mLoc.equals(other) && otherTeam == mTeam;
    }

    public boolean equals(Location other, int otherTeam, int damage){
        return mLoc.equals(other) && otherTeam == mTeam && mDamage == damage;
    }

    @Override
    public String toString(){
        return getX() + "," + getY() + "," + getD() + "," + getTeam();
    }

    public String toJSONString(){
        return JsonWriter.string()
            .object()
            .value("x", getX())
            .value("y", getY())
            .value("damage", getD())
            .value("team", getTeam())
            .end()
            .done();
    }
}
```

Piece

```
package ShowNTell;

/**
 * Piece
 * Contains the data to draw a halma piece.
 * This file includes damage count pieces and XPiece.
 */

import java.util.*;
import java.awt.Color;

public class Piece extends Rock{

    private int x, y, damage, team;
    private String mColor;
    private static final String DEFAULT_COLOR_STRING = "default";

    public Piece(){
    }

    public Piece(int x, int y, int d, int t){
        this.x = x;
        this.y = y;
        this.damage = d;
        this.team = t;
        mColor = DEFAULT_COLOR_STRING;
    }

    public String setColor(Color c, boolean local){
        super.setColor(c);
        return c.toString();
    }
    public String setColor(String inColor){
        if( "red".equalsIgnoreCase(inColor) )
            return this.setColor( Color.red, true );
        if( "blue".equalsIgnoreCase(inColor) )
            return this.setColor( Color.blue, true );
        if( "cyan".equalsIgnoreCase(inColor) )
            return this.setColor( Color.cyan, true );
        if( "magenta".equalsIgnoreCase(inColor) )
            return this.setColor( Color.magenta, true );
        if( "black".equalsIgnoreCase(inColor) )
            return this.setColor( Color.black, true );
        return mColor;
    }

    public ArrayList<Integer> toIntList(){
        int [] arr = { x, y, damage, team};
        ArrayList<Integer> list = new ArrayList<Integer>();
        for (int i : arr)
            list.add( i );
        return list;
    }

    public Location getXYLocation(){
        return new Location(y, x);
    }

    public int getDamage(){
        return damage;
    }

    public int getTeam(){
        return team;
    }

    @Override
    public String toString(){
        return "P(" + x + "," + y + "," + damage + "," + team + ")";
    }
}

class Five extends Piece{}
class Four extends Piece{}
class Three extends Piece{}
class Two extends Piece{}
class One extends Piece{}
class XPiece extends Piece{}
```