Course: Principals of Software Development – ENSF 409

Lab 4

Instructor: M. Moshirpour

Student Name: Mitchell Sawatzky and Connor Newman

Date Submitted: Feb 12, 2016

Exercise B

Geometry.java

```
import java.util.Iterator;
import java.util.TreeSet;
public class Geometry{
        private TreeSet <Shape> shapes;
    public Geometry() {
        shapes = new TreeSet <Shape>();
    }
        public static void main(String[] args) {
                Rectangle r1 = new Rectangle(3.0, 4.0, 5.0, 6.0, "R1", new Colour("Black"));
        Circle c1 = new Circle (13.0, 14.0, 15.0, "C1", new Colour ("Green"));
        System.out.println(r1);
        System.out.println(c1);
                Rectangle r2 = new Rectangle(23.0, 24.0, 25.0, 26.0, "R2", new Colour("Black"));
        Circle c2 = new Circle (33.0, 34.0, 35.0, "C2", new Colour("Yellow"));
        System.out.println(r2);
        System.out.println(c2);
                Prism p1 = new Prism(43.0, 44.0, 45.0, 46.0, 47.0, "P1", new Colour("White"));
        Prism p2 = new Prism(53.0, 54.0, 55.0, 56.0, 57.0, "P2", new Colour("Gray"));
        System.out.println(p1);
        System.out.println(p2);
        Geometry demo = new Geometry();
        System.out.println("\nAdding Rectangle, Circle, and Prism objects to the list... ");
        demo.add(r1);
        demo.add(r2);
        demo.add(c1);
        demo.add(c2);
        demo.add(p1);
        demo.add(p2);
        System.out.println("\nShowing information about objects added to the list:");
        demo.showAll();
        System.out.println("\nShowing area, perimeter, and volume of objects in the list:");
```

```
Iterator <Shape> it = demo.shapes.iterator();
       while(it.hasNext()){
            demo.calculator(it.next());
       }
        }
    public void add(Shape sh) {
        shapes.add(sh);
    }
    public void showAll() {
        Iterator <Shape> it = shapes.iterator();
       while (it.hasNext()) {
            System.out.println(it.next());
       }
    }
    public void calculator(Shape sh) {
        System.out.printf("The area, perimeter, and volume of %s are %.2f, %.2f, %.2f.\n", sh.name,
sh.area(), sh.perimeter(), sh.volume());
    }
}
```

Shape.java

```
abstract class Shape implements Comparable<Shape> {
    protected Point origin;
    protected Text name;
    abstract protected Double area();
    abstract protected Double perimeter();
    abstract protected Double volume();

    protected Shape(Double x_origin, Double y_origin, String name, Colour colour){
        origin = new Point(x_origin,y_origin, colour);
        this.name = new Text(name);
    }

    protected Point getOrigin() {
        return origin;
}
```

```
protected String getName() {
           return name.getText();
    }
    protected Double distance( Shape other){
           return origin.distance(other.origin);
    }
    protected Double distance(Shape a, Shape b){
            return Point.distance(a.origin, b.origin);
    }
    protected void move(Double dx, Double dy){
           origin.setx(origin.getx()+dx);
           origin.sety(origin.gety()+dy);
    }
public int compareTo(Shape other) {
   return name.compareTo(other.name);
}
    @Override
    public String toString(){
           String s = "\nShape name: " + name + "\nOrigin: " + origin;
           return s;
    }
```

Text.java

```
public class Text implements Comparable<Text> {
    private String text;

    public Text(String text) {
        this.text = text;
     }

    public void setText(String newText){
        text = newText;
}
```

```
public String getText(){
        return text;
}

public int compareTo(Text other) {
    return text.compareTo(other.text);
}

@Override
    public String toString(){
        return (text);
}
```

Exercise C

BlockingPlayer.java

```
* Provides a tic tac toe robot Player that attempts to block every move it's opponent makes.
 * @version 1.0
 * @author Mitchell Sawatzky and Connor Newman
 * @since Feb 2016
 */
public class BlockingPlayer extends RandomPlayer {
     * Constructs a BlockingPlayer object with the specified name, mark, and board.
     * @param name the Player's name
     * @param mark the Player's mark
     * @param board the Player's board
     */
    public BlockingPlayer(String name, char mark, Board board) {
        super (name, mark, board);
    }
     * Detects whether or not the opponent is about to win, and blocks it if necesarry, otherwise it makes
a random move.
    protected void makeMove() {
```

```
for (int i = 0; i < 3; i++) {
            for (int j = 0; j < 3; j++) {
                if ((board.getMark(i, j) == SPACE_CHAR) && testForBlocking(i, j)) {
                    board.addMark(i, j, mark);
                    return;
                }
            }
       }
        super.makeMove();
    }
     * Tests wether or not the specified board space would win the game for the opponent if they played
there on the next turn.
     ^{st} @param row the row of the board to test
     st @param col the column of the board to test
     * @return true if the space needs to be blocked, false otherwise
    protected boolean testForBlocking(int row, int col) {
        char oM = opponent.mark();
       boolean res = true;
       // row
        for (int i = 0; i < 3; i++) {
            if ((i != col) && (board.getMark(row, i) != oM)) {
                res = false;
                break;
            }
        }
        if (res)
            return true;
       // col
       res = true;
        for (int i = 0; i < 3; i++) {
            if ((i != row) && (board.getMark(i, col) != oM)) {
                res = false;
                break;
            }
       }
        if (res)
            return true;
```

```
// can't be diagonal
    if ((row + col) % 2 != 0)
        return false;
   // diagonal
    switch (row) {
        case 0:
            if (col != 0 && board.getMark(2, 0) == oM && board.getMark(1, 1) == oM)
                return true;
            else if (board.getMark(2, 2) == oM && board.getMark(1, 1) == oM)
                return true;
            break;
        case 1:
            if ((board.getMark(0, 0) == oM && board.getMark(2, 2) == oM) ||
                (board.getMark(0, 2) == oM && board.getMark(2, 0) == oM))
                return true;
            break;
        case 2:
            if (col != 0 && board.getMark(0, 0) == oM && board.getMark(1, 1) == oM)
                return true;
            else if (board.getMark(0, 2) == oM && board.getMark(1, 1) == oM)
                return true;
            break;
   }
   return false;
}
```

Board.java

```
// Board.java
// ENSF 409 - LAB 3 - Ex. C
// This file was originally written for ENGG 335 in fall 2001, and was
// adapted for ENSF 409 in 2014
//

/**

* Provides a tic-tac-toe board and logic to fill, empty, and test if a player has won.

* @author Originally written by Mahmood Moussavi, modified by Mitchell Sawatzky and Connor Newman

* @version 1.0

* @since Originally written in fall 2001, adapted in 2014, modified in 2016

*/
```

```
public class Board implements Constants {
     * Two-Dimensional char array to hold the values of each slot on the board
        private char theBoard[][];
    /**
     * The total number of slots filled in on the board.
        private int markCount;
     * Constructs a Board object without any spaces filled in.
        public Board() {
                markCount = 0;
                theBoard = new char[3][];
                for (int i = 0; i < 3; i++) {
                         theBoard[i] = new char[3];
                         for (int j = 0; j < 3; j++)
                                 theBoard[i][j] = SPACE_CHAR;
                }
        }
     * Returns the value of a board slot at a given row and column.
     * @param row the row to retrieve the board slot from
     \ensuremath{^{*}} @param col the column to retrieve the board slot from
     * @return the Character value of the board slot
        public char getMark(int row, int col) {
                return theBoard[row][col];
        }
     * Returns whether or not the board has values in all 9 slots.
     * @return True if all 9 slots are full, False otherwise
        public boolean isFull() {
                return markCount == 9;
        }
```

```
* Checks whether or not the letter X has won on the current board.
 * @return 0 if X has not won, 1 otherwise
    public int xWins() {
            return checkWinner(LETTER_X);
    }
/**
 * Checks whether or not the letter O has won on the current board.
 st @return 0 if 0 has not won, 1 otherwise
    public int oWins() {
            return checkWinner(LETTER_0);
    }
 * Prints the board to stdout.
    public void display() {
            displayColumnHeaders();
            addHyphens();
            for (int row = 0; row < 3; row++) {</pre>
                    addSpaces();
                    System.out.print("
                                         row " + row + ' ');
                    for (int col = 0; col < 3; col++)
                             System.out.print("| " + getMark(row, col) + " ");
                    System.out.println("|");
                    addSpaces();
                    addHyphens();
            }
    }
 * Sets the value of the board slot at a given row and column.
 st @param row the row to set the slot value
 * @param col the column to set the slot value
 ^{st} @param mark the Character to set the slot to
    public void addMark(int row, int col, char mark) {
            theBoard[row][col] = mark;
            markCount++;
```

```
}
/**
* Resets every value on the board to SPACE_CHAR.
    public void clear() {
            for (int i = 0; i < 3; i++)
                    for (int j = 0; j < 3; j++)
                             theBoard[i][j] = SPACE_CHAR;
            markCount = 0;
    }
* Uses tic-tac-toe logic to determine if a specific player has won.
* @param mark the player to check, either LETTER_X or LETTER_O
* @return 0 if the player has lost, 1 otherwise
    int checkWinner(char mark) {
            int row, col;
            int result = 0;
            for (row = 0; result == 0 && row < 3; row++) {
                    int row_result = 1;
                    for (col = 0; row_result == 1 && col < 3; col++)</pre>
                             if (theBoard[row][col] != mark)
                                     row_result = 0;
                    if (row_result != 0)
                             result = 1;
            }
            for (col = 0; result == 0 && col < 3; col++) {
                    int col_result = 1;
                    for (row = 0; col_result != 0 && row < 3; row++)</pre>
                             if (theBoard[row][col] != mark)
                                     col_result = 0;
                    if (col_result != 0)
                             result = 1;
            }
            if (result == 0) {
                    int diag1Result = 1;
```

```
for (row = 0; diag1Result != 0 && row < 3; row++)</pre>
                            if (theBoard[row][row] != mark)
                                    diag1Result = 0;
                   if (diag1Result != 0)
                            result = 1;
           }
           if (result == 0) {
                   int diag2Result = 1;
                   for (row = 0; diag2Result != 0 && row < 3; row++)</pre>
                            if (theBoard[row][3 - 1 - row] != mark)
                                    diag2Result = 0;
                   if (diag2Result != 0)
                           result = 1;
           }
           return result;
   }
* Print the board's column headers to stdout.
   void displayColumnHeaders() {
           System.out.print("
                                       ");
           for (int j = 0; j < 3; j++)
                   System.out.print("|col " + j);
           System.out.println();
   }
* Adds a line to separate the board's rows.
   void addHyphens() {
           System.out.print("
                                       ");
           for (int j = 0; j < 3; j++)
                   System.out.print("+----");
           System.out.println("+");
   }
* Adds spacing inside the board to correctly place the values of the slots.
   void addSpaces() {
           System.out.print("
                                        ");
```

Constants.java

Game.java

```
//Game.java
import java.io.*;

/**
    * @author Started by: M. Moussavi
    * Completed by: Mitchell Sawatzky and Connor Newman
    * Asks the user to select a player type, creates the player, creates the board,
    * assigns a referee to the game, then initiates the game.
```

```
public class Game implements Constants {
        private Board theBoard;
        private Referee theRef;
        /**
         * creates a board for the game
         */
    public Game( ) {
        theBoard = new Board();
        }
     * calls the referee method runTheGame
     ^{st} @param r refers to the appointed referee for the game
     * @throws IOException
    public void appointReferee(Referee r) throws IOException {
        theRef = r;
        theRef.runTheGame();
    }
        public static void main(String[] args) throws IOException {
                Referee theRef;
                Player xPlayer, oPlayer;
                BufferedReader stdin;
                Game theGame = new Game();
                stdin = new BufferedReader(new InputStreamReader(System.in));
                System.out.print("\nPlease enter the name of the \'X\' player: ");
                String name= stdin.readLine();
                while (name == null) {
                        System.out.print("Please try again: ");
                        name = stdin.readLine();
                }
                xPlayer = create_player (name, LETTER_X, theGame.theBoard, stdin);
                System.out.print("\nPlease enter the name of the \'0\' player: ");
```

```
name = stdin.readLine();
        while (name == null) {
                System.out.print("Please try again: ");
                name = stdin.readLine();
        }
        oPlayer = create_player (name, LETTER_O, theGame.theBoard, stdin);
        theRef = new Referee(theGame.theBoard, xPlayer, oPlayer);
theGame.appointReferee(theRef);
}
/**
 * Creates the specified type of player indicated by the user.
 * @param name player's name
 * @param mark player's mark (X or 0)
 ^{st} @param board refers to the game board
 * @param stdin refers to an input stream
 * @return a newly created player
 * @throws IOException
 */
static public Player create_player(String name, char mark, Board board,
                BufferedReader stdin)throws IOException {
        // Get the player type.
        final int NUMBER_OF_TYPES = 4;
        System.out.print ( "\nWhat type of player is " + name + "?\n");
        System.out.print(" 1: human\n" + " 2: Random Player\n"
        + " 3: Blocking Player\n" + " 4: Smart Player\n");
        System.out.print( "Please enter a number in the range 1-" + NUMBER_OF_TYPES + ": ");
        int player_type = 0;
        String input;
        stdin = new BufferedReader(new InputStreamReader(System.in));
        input= stdin.readLine();
        player_type = Integer.parseInt(input);
        while (player_type < 1 || player_type > NUMBER_OF_TYPES) {
                System.out.print( "Please try again.\n");
                System.out.print ( "Enter a number in the range 1-" +NUMBER_OF_TYPES + ": ");
                input= stdin.readLine();
                player_type = Integer.parseInt(input);
```

```
}
                // Create a specific type of Player
                Player result = null;
                switch(player_type) {
                        case 1:
                                 result = new HumanPlayer(name, mark, board);
                                 break;
                        case 2:
                                 result = new RandomPlayer(name, mark, board);
                                 break;
                        case 3:
                                 result = new BlockingPlayer(name, mark, board);
                                 break;
                        case 4:
                                 result = new SmartPlayer(name, mark, board);
                                 break;
                        default:
                                 System.out.print ( "\nDefault case in switch should not be reached.\n"
                                 + " Program terminated.\n");
                                 System.exit(0);
                return result;
        }
}
```

HumanPlayer.java

```
import java.util.Scanner;

/**

* Provides methods to gather input from stdin in order to play a game of tic tac toe.

* @author Mitchell Sawatzky and Connor Newman

* @version 1.0

* @since Feb 2016

*/

public class HumanPlayer extends Player {

    /**

    * Constructs a HumanPlayer object with the specified name, mark, and board.

    * @param name the Player's name

    * @param board the Board to play the game on
```

```
*/
public HumanPlayer(String name, char mark, Board board) {
    super(name, mark, board);
}
 \ensuremath{^{*}} Starts a game of tic tac toe with this player as player X.
public void play() {
    String winner;
    Player p = this;
    while (true) {
        if (board.isFull()) {
            winner = "Nobody";
            break;
        } else if (board.xWins() == 1) {
            winner = name;
            break;
        } else if (board.oWins() == 1) {
            winner = opponent.name();
            break;
        p.makeMove();
        board.display();
        p = p.opponent;
    }
    System.out.printf("\nTHE GAME IS OVER: %s is the winner!\n", winner);
}
 * Prompts the user via stdout to make a move on the tic tac toe Baord.
public void makeMove() {
    int row, col;
    Player p = this;
    while (true) {
        while (true) {
            System.out.printf("%s, what row should your next %c be placed in? ", p.name, p.mark);
            Scanner input = new Scanner(System.in);
            row = input.nextInt();
            if (row < 0 || row > 2)
```

```
System.out.printf("\nInvalid row: %d, please try again.\n", row);
                else
                    break;
            }
            while (true) {
                System.out.printf("%s, what column should your next %c be placed in? ", name, mark);
                Scanner input = new Scanner(System.in);
                col = input.nextInt();
                if (col < 0 || col > 2)
                    System.out.printf("\nInvalid column: %d, please try again.\n", col);
                else
                    break;
            }
            if (board.getMark(row, col) == SPACE_CHAR) {
                board.addMark(row, col, mark);
                break;
            } else {
                System.out.printf("\nThe coordinate (%d, %d) has already been used.\n", row, col);
            }
        }
    }
}
```

Player.java

```
//Player.java

/**
 * Provides a container to hold a Player's name and preferred mark (X or 0), as well as logic prototypes.
 * @author Mitchell Sawatzky and Connor Newman
 * @version 1.0
 * @since Feb 5, 2016
 */
abstract class Player implements Constants {
    /**
    * The name of the player.
    */
    protected String name;

    /**
    * The player's mark, either 'X' or '0'.
```

```
*/
protected char mark;
* The player's opponent.
protected Player opponent;
/**
* The Board to play the game on.
*/
protected Board board;
/**
* Constructs a Player Object with a given name, mark, and Board.
* @param name the Player's name
 * @param mark the Player's mark, either 'X' or 'O'
 ^{st} @param b the Board to play the game on
public Player(String name, char mark, Board b) {
   this.name = name;
   this.mark = mark;
   this.board = b;
}
/**
* Getter function for the Player's name.
* @return the String name of the Player
*/
protected String name() {
   return name;
}
^{st} Getter function for the Player's mark
* @return the char mark of the Player
*/
protected char mark() {
   return mark;
}
* Sets the opponent of a given Player to another Player.
```

```
* @param opp the Player opponent
*/
protected void setOpponent(Player other) {
    this.opponent = other;
}

/**
    * Initiate a game of tic-tac-toe with the opponent player.
    */
abstract protected void play();

/**
    * Prompt the user to place their mark on a given board slot retrieved through stdin.
    */
abstract protected void makeMove();
}
```

RandomGenerator.java

```
// RandomGenerator.java
import java.util.Random;
/**
 * Provides a method to spawn a random integer.
 * @author M. Moussavi
 */
class RandomGenerator {
/**
 * creates a random number ranging between lo and hi,
 ^{st} @param lo the lower bound of the random integer
 \ensuremath{^*} @param hi the upper bound of the random integer
 * @return the random integer
 */
        int discrete(int lo, int hi)
        {
                 if(lo >= hi){}
                         System.out.println("Error discrete, lo >= hi");
                         System.exit(0);
                 }
```

```
Random r = new Random();
int d = r.nextInt(hi - lo + 1) + lo;
return d;
}
```

RandomPlayer.java

```
* Provides a tic tac toe robot that randomly chooses a space on every move.
* @author Mitchell Sawatzky and Connor Newman
 * @version 1.0
* @since Feb 2016
*/
public class RandomPlayer extends Player {
    /**
     st Constructs a RandomPlayer object with the specified name, mark, and board.
     */
    public RandomPlayer(String name, char mark, Board board) {
        super(name, mark, board);
    }
    /**
     * Starts a game of tic tac toe with this player as player X.
     */
    protected void play() {
       String winner;
       Player p = this;
       while (true) {
            if (board.isFull()) {
                winner = "Nobody";
                break;
            } else if (board.xWins() == 1) {
                winner = p.name;
                break;
            } else if (board.oWins() == 1) {
                winner = p.opponent.name();
                break;
            }
            p.makeMove();
            board.display();
```

```
p = p.opponent;
}
System.out.printf("\nTHE GAME IS OVER: %s is the winner!\n", winner);
}

/**
 * Picks a random board slot and makes a move there.
 */
protected void makeMove() {
    RandomGenerator rand = new RandomGenerator();
    int row, col;

    do {
        row = rand.discrete(0, 2);
        col = rand.discrete(0, 2);
    } while (board.getMark(row, col) != SPACE_CHAR);

    board.addMark(row, col, mark);
}
```

Referee.java

```
//Referee.java

/**

* Mediates and controls a game of Tic Tac Toe.

* Begins the game by printing the board, and then asks Player X to choose

* @author Mitchell Sawatzky and Connor Newman

* @version 1.0

* @since Feb 5, 2016

*/
public class Referee {
    /**

    * Player X of the game.

    */
    private Player x;

/**

    * Player 0 of the game.

    */
    private Player o;
```

```
^{st} The board to play on.
    private Board b;
    /**
     * Construct a Referee object from Players and a Board.
     * @param board the Board for the referee to control
     * @param xPlayer the player with the mark 'X'
     * @param oPlayer the player with the mark '0'
     */
    public Referee(Board board, Player xPlayer, Player oPlayer) {
        this.b = board;
        this.x = xPlayer;
        this.o = oPlayer;
    }
     * Initiate a game with Player X as the starting player.
    public void runTheGame() {
        x.setOpponent(o);
        o.setOpponent(x);
        b.display();
        x.play();
        System.out.println("\033[1mGame ended ...\033[0m");
    }
}
```

SmartPlayer.java

```
/**

* Provides a tic tac toe robot that first checks it it can win, and then checks whether or not it can block the opponent from winning.

* @author Mitchell Sawatzky and Connor Newman

* @version 1.0

* @since Feb 2016

*/
public class SmartPlayer extends BlockingPlayer {
```

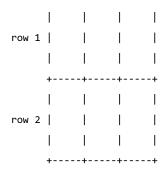
```
/**
     * Constructs a SmartPlayer object with the specified name, mark, and board.
     * @param name the Player's name
     * @param mark the Player's mark
     \ensuremath{^*} @param board the board to play the game on
     */
    public SmartPlayer(String name, char mark, Board board) {
        super(name, mark, board);
    }
     * First checks whether it can win the game, and then falls back to BlockingPlayer's logic to block
the opponent.
    protected void makeMove() {
        for (int i = 0; i < 3; i++) {
            for (int j = 0; j < 3; j++) {
                if ((board.getMark(i, j) == SPACE_CHAR) && testForWinning(i, j)) {
                    board.addMark(i, j, mark);
                    return;
                }
            }
        }
        super.makeMove();
    }
     * Decides whether placing a mark in the specified row and column will win the getName
     * @param row the row to place the mark in
     * @param col the column to place the mark in
     ^{st} @return true if placing the mark wins the game, false otherwise
    public boolean testForWinning(int row, int col) {
        boolean res = true;
        // row
        for (int i = 0; i < 3; i++) {
            if ((i != col) && (board.getMark(row, i) != mark)) {
                res = false;
                break;
            }
```

```
}
   if (res)
       return true;
   // col
   res = true;
   for (int i = 0; i < 3; i++) {
       if ((i != row) && (board.getMark(i, col) != mark)) {
            res = false;
           break;
       }
   }
   if (res)
       return true;
   // can't be diagonal
   if ((row + col) % 2 != 0)
        return false;
   // diagonal
    switch (row) {
       case 0:
            if (col != 0 && board.getMark(2, 0) == mark && board.getMark(1, 1) == mark)
                return true;
           else if (board.getMark(2, 2) == mark && board.getMark(1, 1) == mark)
                return true;
            break;
       case 1:
            if ((board.getMark(0, 0) == mark && board.getMark(2, 2) == mark) ||
                (board.getMark(0, 2) == mark && board.getMark(2, 0) == mark))
                return true;
            break;
       case 2:
            if (col != 0 \&\& board.getMark(0, 0) == mark \&\& board.getMark(1, 1) == mark)
                return true;
            else if (board.getMark(0, 2) == mark && board.getMark(1, 1) == mark)
                return true;
            break;
   }
   return false;
}
```

}

Terminal output for a Human vs Human game:

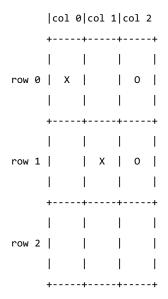
```
Please enter the name of the 'X' player: John
What type of player is John?
 1: human
 2: Random Player
 3: Blocking Player
 4: Smart Player
Please enter a number in the range 1-4: 1
Please enter the name of the 'O' player: Sandy
What type of player is Sandy?
 1: human
 2: Random Player
 3: Blocking Player
 4: Smart Player
Please enter a number in the range 1-4: 1
         |col 0|col 1|col 2
         +----+
   row 0
   row 1
   row 2
John, what row should your next X be placed in? 0
John, what column should your next X be placed in? 0
         |col 0|col 1|col 2
         +----+
             row 0 | X |
             +----+
```



Sandy, what row should your next 0 be placed in? 0 Sandy, what column should your next 0 be placed in? 2

John, what row should your next X be placed in? 1 John, what column should your next X be placed in? 1

Sandy, what row should your next 0 be placed in? 1
Sandy, what column should your next 0 be placed in? 2



John, what row should your next X be placed in? 2

John, what column should your next X be placed in? 2

THE GAME IS OVER: John is the winner! Game ended \dots

Terminal Output for a Human vs RandomPlayer Game:

Please enter the name of the 'X' player: John

What type of player is John?

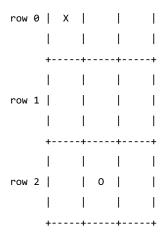
- 1: human
- 2: Random Player
- 3: Blocking Player
- 4: Smart Player

Please enter a number in the range 1-4: 1 $\,$

Please enter the name of the 'O' player: Sandy

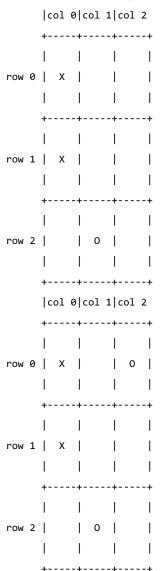
What type of player is Sandy? 1: human 2: Random Player 3: Blocking Player 4: Smart Player Please enter a number in the range 1-4: 2 |col 0|col 1|col 2 +----+ row 0 row 1 row 2 | John, what row should your next X be placed in? 0 John, what column should your next \boldsymbol{X} be placed in? 0 |col 0|col 1|col 2 +----+ row 0 | X | row 1 row 2

+----+
|col 0|col 1|col 2
+----+



John, what row should your next X be placed in? 1

John, what column should your next X be placed in? 0



John, what row should your next \boldsymbol{X} be placed in? 2

```
John, what column should your next \boldsymbol{X} be placed in? 0
               |col 0|col 1|col 2
               +----+
               row 0 | X | | 0 |
               row 1 | X | |
               +----+
               row 2 | X | 0 |
               +----+
      THE GAME IS OVER: John is the winner!
      Game ended ...
Terminal output for a Human vs BlockingPlayer game:
       Please enter the name of the 'X' player: John
      What type of player is John?
        1: human
        2: Random Player
        3: Blocking Player
        4: Smart Player
      Please enter a number in the range 1-4: 1
      Please enter the name of the 'O' player: Sandy
      What type of player is Sandy?
        1: human
        2: Random Player
        3: Blocking Player
        4: Smart Player
      Please enter a number in the range 1-4: 3
               |col 0|col 1|col 2
               +----+
          row 0
               +----+
```

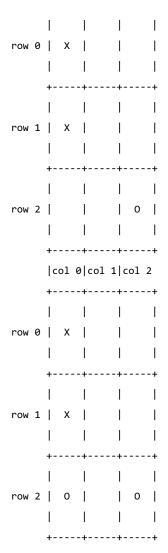
		1			
row	1	1			
		1	I	1	
		+	+	+	+
			1	1	I
row	2				
			1	1	١
		+	+	+	+

John, what row should your next X be placed in? 0 John, what column should your next X be placed in? 0 $\,$

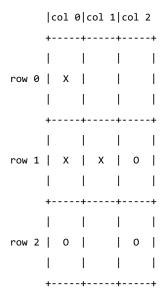
|col 0|col 1|col 2 +----+ row 0 | X | row 1 row 2 |col 0|col 1|col 2 row 0 | X | row 1 | row 2

John, what row should your next X be placed in? 1 John, what column should your next X be placed in? 0 $\,$

|col 0|col 1|col 2



John, what row should your next X be placed in? 1 John, what column should your next X be placed in? 1



John, what row should your next X be placed in? 0

John, what column should your next X be placed in? 2

|col 0|col 1|col 2 +----+ row 0 | X | +----+ row 1 | X | X | 0 | row 2 | 0 | 0 | +----+ |col 0|col 1|col 2 +----+ row 0 | X | 0 | X | +----+ row 1 | X | X | 0 |

row 2 | 0 | | 0 |

+----+ John, what row should your next X be placed in? 2 John, what column should your next X be placed in? 1 |col 0|col 1|col 2 +----+ row 0 | X | 0 | X | +----+ row 1 | X | X | 0 | +----+ row 2 | 0 | X | 0 | +----+ THE GAME IS OVER: Nobody is the winner! Game ended ... Terminal Output for a Human vs SmartPlayer game: Please enter the name of the 'X' player: John What type of player is John? 1: human 2: Random Player 3: Blocking Player 4: Smart Player Please enter a number in the range 1-4: 1 Please enter the name of the 'O' player: Sandy What type of player is Sandy? 1: human 2: Random Player 3: Blocking Player 4: Smart Player Please enter a number in the range 1-4: 4 |col 0|col 1|col 2 +----+

row	0				
			l	l	
		+	+	+	+
		1			
row	1	1			
		1			
		+	+	+	+
		1			
row	2	1			
		1			١
		+	+	+	+

John, what row should your next X be placed in? 0

John, what column should your next X be placed in? 0

|col 0|col 1|col 2 row 0 | X | row 1 | row 2 |col 0|col 1|col 2 row 0 | X | row 1 | 0 | row 2

John, what row should your next \boldsymbol{X} be placed in? 0

John, what column should your next \boldsymbol{X} be placed in? 2

|col 0|col 1|col 2 +----+ row 0 | X | row 1 | 0 | row 2 +----+ |col 0|col 1|col 2 +----+ row 0 | X | 0 | X | row 1 | 0 | row 2

John, what row should your next X be placed in? 0

John, what column should your next X be placed in? 2

The coordinate (0, 2) has already been used.

John, what row should your next X be placed in? 2

John, what column should your next X be placed in? 0

|col 0|col 1|col 2

row	1		0	1		-		
				1				
		+-		-+-		-+-		-+
				1				1
row	2		Χ					
		+-		-+-		-+-		-+
		c	ol (∂ c	ol	1 c	ol	2
		+-		-+-		-+-		-+
				1				1
row	0		Χ	1	0		Χ	1
				1				1
		+-		-+-		-+-		-+
				1				1
row	1		0	1	0			1
				1				1
		+-		-+-		-+-		-+
		1		1		1		1
row	2		Х					1
								1
		+-		-+-		-+-		-+

John, what row should your next X be placed in? 2

John, what column should your next X be placed in? 2

|col 0|col 1|col 2 +----+ row 0 | X | 0 | X | +----+ row 1 | 0 | 0 | +----+ row 2 | X | | X | +----+ |col 0|col 1|col 2 +----+ 1 1 1 row 0 | X | 0 | X |

		+-		+-		+-		+
		1		1				I
row	1			(0		0	
		1		1				
		+-		+-		+-		+
				1				I
row	2		Х				Х	
		+-		+-		+-		+

THE GAME IS OVER: Sandy is the winner! $\label{eq:Game_sample} \mbox{Game ended } \dots$