```
 1 import javax.swing.*;
 2 import java.awt.*;
 3 import java.awt.event.ActionEvent;
 4 import java.awt.event.ActionListener;
 5
 6 public class GUI extends JFrame{
 7
 8     GridBagConstraints c;
 9     GridBagLayout layout;
10     boolean RIGHT_TO_LEFT = false;
11     char[] action;
12     JLabel[] p1cards;
13     JLabel[] p2cards;
14     JButton[] buttons;
15     JLabel p1points;
16     JLabel p2points;
17     JLabel msg;
18     JLabel round;
19     boolean turn;
20
21     GUI(){
22         layout = new GridBagLayout();
23         c = new GridBagConstraints();
24         action = new char[2];
25         p1cards = new JLabel[6];
26         p2cards = new JLabel[6];
27         buttons = new JButton[13];
28         turn = false;
29
30         this.setDefaultCloseOperation(JFrame.
   EXIT_ON_CLOSE);
31         this.setTitle("CoExistence");
32         this.setLocationRelativeTo(null);
33         this.setLayout(layout);
34
35         populateLayout();
36
37         this.setVisible(true);
38         this.setSize(600,400);
39     }
40
```

```
41      public void updateLayout(char[][] gameboard){
42          updateCards(gameboard);
43          updateMsg(gameboard);
44
45          p2points.setText(gameboard[7][38] + "");
46          p1points.setText(gameboard[9][38] + "");
47          round.setText("R: " + gameboard[8][38]);
48      }
49
50      public void updateMsg(char[][] gameboard){
51          String msg = "";
52          int limit = 38;
53          int i = 17;
54          int j = 1;
55
56          while(gameboard[i][limit] == ' ') limit--;
57
58          while(j != limit + 1) {
59              msg += gameboard[i][j];
60              j++;
61          }
62
63          this.msg.setText(msg);
64      }
65
66      public void updateCards(char[][] gameboard){
67          for(int i = 0; i < 6; i++){
68              p2cards[i].setText(checkCard(3, 3 + i*5
    , gameboard));
69          }
70
71          for(int i = 0; i < 6; i++){
72              p1cards[i].setText(checkCard(11, 3 + i*
    5, gameboard));
73          }
74      }
75
76      public String checkCard(int i, int j, char[][]
    gameboard){
77          String card = "";
78
```

```
 79          if(gameboard[i][j] == '=' && gameboard[i][
    j + 1] == ']') card = "Hammer";
 80          else if(gameboard[i][j] == '^' &&
    gameboard[i][j + 1] == ' ') card = "Arrow";
 81          else if(gameboard[i][j] == ' ' &&
    gameboard[i][j + 1] == '/') card = "Sword";
 82          else if(gameboard[i][j] == '7' &&
    gameboard[i][j + 1] == '>') card = "Axe";
 83
 84          return card;
 85      }
 86
 87      /**
 88       * Create the GUI and show it.  For thread
    safety,
 89       * this method should be invoked from the
 90       * event-dispatching thread.
 91       */
 92      public void populateLayout(){
 93
 94          if (RIGHT_TO_LEFT) {
 95              this.setComponentOrientation(
    ComponentOrientation.RIGHT_TO_LEFT);
 96          }
 97
 98          JButton button;
 99          JLabel label;
100          JPanel panel;
101
102          resetContraints();
103          // 1st Row
104          button = new JButton("A");
105          button.addActionListener(new
    ActionListener() {
106              @Override
107              public void actionPerformed(
    ActionEvent actionEvent) {
108                  action[1] = actionEvent.
    getActionCommand().toCharArray()[0];
109
110              }
```

```
111         });
112         c.gridx = 0;
113         this.add(button, c);
114 
115         this.buttons[0] = button;
116 
117         button = new JButton("B");
118         button.addActionListener(new
    ActionListener() {
119             @Override
120             public void actionPerformed(
    ActionEvent actionEvent) {
121                 action[1] = actionEvent.
    getActionCommand().toCharArray()[0];
122 
123             }
124         });
125         c.gridx = 1;
126         this.add(button, c);
127 
128         this.buttons[1] = button;
129 
130         button = new JButton("C");
131         button.addActionListener(new
    ActionListener() {
132             @Override
133             public void actionPerformed(
    ActionEvent actionEvent) {
134                 action[1] = actionEvent.
    getActionCommand().toCharArray()[0];
135 
136             }
137         });
138         c.gridx = 2;
139         this.add(button, c);
140 
141         this.buttons[2] = button;
142 
143         button = new JButton("D");
144         button.addActionListener(new
    ActionListener() {
```

```
145             @Override
146             public void actionPerformed(
    ActionEvent actionEvent) {
147                 action[1] = actionEvent.
    getActionCommand().toCharArray()[0];
148 
149             }
150         });
151         c.gridx = 3;
152         this.add(button, c);
153 
154         this.buttons[3] = button;
155 
156         button = new JButton("E");
157         button.addActionListener(new
    ActionListener() {
158             @Override
159             public void actionPerformed(
    ActionEvent actionEvent) {
160                 action[1] = actionEvent.
    getActionCommand().toCharArray()[0];
161 
162             }
163         });
164         c.gridx = 4;
165         this.add(button, c);
166 
167         this.buttons[4] = button;
168 
169         button = new JButton("F");
170         button.addActionListener(new
    ActionListener() {
171             @Override
172             public void actionPerformed(
    ActionEvent actionEvent) {
173                 action[1] = actionEvent.
    getActionCommand().toCharArray()[0];
174 
175             }
176         });
177         c.gridx = 5;
```

```
178         this.add(button, c);
179 
180         this.buttons[5] = button;
181 
182         label = new JLabel("0");
183         c.gridx = 6;
184         this.add(label,c);
185 
186         p2points = label; // Save this label so we
     can update it later
187 
188         resetContraints();
189         // 2nd row
190 
191         c.gridx = 0;
192         c.gridy = 2;
193         c.gridheight = 2;
194         c.gridwidth = 1;
195         c.weighty = 1.0;
196 
197         for(int i = 0; i < 6; i++){
198             c.gridx = i;
199             label = new JLabel("Card");
200             panel = new JPanel();
201             panel.add(label);
202             panel.setBackground(new Color(255, 255
    , 255));
203 
204             // Add player's card to corresponding
    array
205             p2cards[i] = label;
206 
207             this.add(panel, c);
208         }
209 
210         c.weighty = 0;
211         c.gridheight = 1;
212         button = new JButton("PS");
213         c.gridx = 6;
214         this.add(button, c);
215         button.addActionListener(new
```

```
215 ActionListener() {
216             @Override
217             public void actionPerformed(
    ActionEvent actionEvent) {
218                 action[0] = 'P';
219                 action[1] = 'S';
220 
221             }
222         });
223 
224         this.buttons[6] = button;
225 
226         resetContraints();
227         // Third Row
228         c.gridx = 0;
229         c.gridy = 4;
230         c.gridheight = 2;
231         c.gridwidth = 1;
232         c.weighty = 1.0;
233 
234         for(int i = 0; i < 6; i++){
235             c.gridx = i;
236             label = new JLabel("Card");
237             panel = new JPanel();
238             panel.add(label);
239             panel.setBackground(new Color(255, 255
    , 255));
240 
241             // Add player's card to corresponding
    array
242             p1cards[i] = label;
243 
244             this.add(panel, c);
245         }
246 
247         c.weighty = 0;
248         c.gridheight = 1;
249         label = new JLabel("R: 1");
250         c.gridx = 6;
251         this.add(label, c);
252 
```

```
253         round = label;  // Save this label so we
    can update it later
254 
255         resetContraints();
256         // Fourth Row
257         c.gridy = 6;
258 
259         button = new JButton("A");
260         button.addActionListener(new
    ActionListener() {
261             @Override
262             public void actionPerformed(
    ActionEvent actionEvent) {
263                 action[0] = actionEvent.
    getActionCommand().toCharArray()[0];
264 
265             }
266         });
267         c.gridx = 0;
268         this.add(button, c);
269 
270         this.buttons[7] = button;
271 
272         button = new JButton("B");
273         button.addActionListener(new
    ActionListener() {
274             @Override
275             public void actionPerformed(
    ActionEvent actionEvent) {
276                 action[0] = actionEvent.
    getActionCommand().toCharArray()[0];
277 
278             }
279         });
280         c.gridx = 1;
281         this.add(button, c);
282 
283         this.buttons[8] = button;
284 
285         button = new JButton("C");
286         button.addActionListener(new
```

```
286 ActionListener() {
287             @Override
288             public void actionPerformed(
    ActionEvent actionEvent) {
289                 action[0] = actionEvent.
    getActionCommand().toCharArray()[0];
290 
291             }
292         });
293         c.gridx = 2;
294         this.add(button, c);
295 
296         this.buttons[9] = button;
297 
298         button = new JButton("D");
299         button.addActionListener(new
    ActionListener() {
300             @Override
301             public void actionPerformed(
    ActionEvent actionEvent) {
302                 action[0] = actionEvent.
    getActionCommand().toCharArray()[0];
303 
304             }
305         });
306         c.gridx = 3;
307         this.add(button, c);
308 
309         this.buttons[10] = button;
310 
311         button = new JButton("E");
312         button.addActionListener(new
    ActionListener() {
313             @Override
314             public void actionPerformed(
    ActionEvent actionEvent) {
315                 action[0] = actionEvent.
    getActionCommand().toCharArray()[0];
316 
317             }
318         });
```

```
319         c.gridx = 4;
320         this.add(button, c);
321 
322         this.buttons[11] = button;
323 
324         button = new JButton("F");
325         button.addActionListener(new
    ActionListener() {
326             @Override
327             public void actionPerformed(
    ActionEvent actionEvent) {
328                 action[0] = actionEvent.
    getActionCommand().toCharArray()[0];
329 
330             }
331         });
332         c.gridx = 5;
333         this.add(button, c);
334 
335         this.buttons[12] = button;
336 
337         label = new JLabel("0");
338         c.gridx = 6;
339         this.add(label, c);
340 
341         p1points = label; // Save this label so we
     can update it later
342 
343         resetContraints();
344         // Fifth row
345         c.gridy = 7;
346         c.gridwidth = 7;
347         label = new JLabel("Message Log",
    SwingConstants.LEFT);
348         panel = new JPanel();
349         panel.add(label);
350         panel.setBackground(new Color(255, 255,
    255));
351 
352         this.add(panel, c);
353 
```

```
354         msg = label; // Save this label so we can
    update it later
355     }
356 
357     public String checkForMove() {
358         String returnValue = "";
359 
360         if (action[0] != '\0' && action[1] != '\0'
    ) {
361 
362             returnValue = new String(action);
363 
364             // Reset actions after processing
365             action[0] = '\0';
366             action[1] = '\0';
367         }
368 
369         return returnValue;
370     }
371 
372     public void resetContraints(){
373         c.gridx = 0;
374         c.gridy = 0;
375         c.gridheight = 1;
376         c.gridwidth = 1;
377         c.weightx = 0.0;
378         c.weighty = 0.0;
379         c.fill = GridBagConstraints.BOTH;
380     }
381 
382     public void updateButtons(){
383         if(turn) enableButtons();
384         else disableButtons();
385     }
386 
387     public void disableButtons(){
388         for(JButton button : buttons) button.
    setEnabled(false);
389     }
390 
391     public void enableButtons(){
```

```
392             for(JButton button : buttons) button.
    setEnabled(true);
393         }
394 }
```

```
 1 import javax.swing.*;
 2 import java.io.BufferedReader;
 3 import java.io.IOException;
 4 import java.io.PrintWriter;
 5 import java.util.ArrayList;
 6 import java.util.Collections;
 7
 8 public class Game {
 9     private char[][] gameboard;
10     int round;
11     Player p1;
12     Player p2;
13     int currPlayer;
14
15     /**
16      * Game class which simulates a game between 2
   players
17      * @param in1 input stream for player 1
18      * @param in2 input stream for player 2
19      * @param out1 output stream for player 1
20      * @param out2 output stream for player 2
21      */
22     Game(BufferedReader in1, BufferedReader in2,
   PrintWriter out1, PrintWriter out2) throws
   IOException {
23         currPlayer = (int) Math.floor(Math.random
   () * 2);
24         p1 = new Player();
25         p2 = new Player();
26         boolean properInp;
27         String inputLine;
28
29         // Initialize a default game board to
   simulate all the round on
30         initializeBoard();
31         printMsg("NEW GAME");
32
33         // Simulate 5 round
34         game:
35         for(round = 1; round < 5; round++){
36             dealCards(p1, p2);
```

```
37              printGameBoards(out1, out2);
38
39              // Simulate a round
40              round:
41              while (true) {
42
43                  // The if condition is to check if
   the starting player is player1
44                  if (currPlayer == 0) {
45                      // Player 1 Turn
46                      do {
47                          inputLine = in1.readLine().
   toUpperCase();
48                          if (inputLine == null)
   break game;
49                          // Make changes to the
   board based on p1
50                          properInp = updateState(
   inputLine);
51
52                          if(properInp){
53                              // Change the player,
   print the new board for both parties and check if
   the round is over
54                              updatePlayer();
55                              if(roundEnd(p1, p2))
   break round;
56                          }
57                          printGameBoards(out1, out2
   );
58                      } while (!properInp);
59                  }
60
61                  // Player 2 Turn
62                  do{
63                      inputLine=in2.readLine().
   toUpperCase();
64                      if (inputLine==null) break game
   ;
65                      // Make changes to the board
   based on p2
```

```
66                      properInp = updateState(
   inputLine);
67
68                      if(properInp){
69                          // Change the player,
   print the new board for both parties and check if
   the round is over
70                          updatePlayer();
71                          if(roundEnd(p1, p2)) break
    round;
72                      }
73                      printGameBoards(out1, out2);
74                  } while(!properInp);
75              }
76
77              // Check if the game is over and reset
    the passes from both variables
78              if(p1.points == 9 || p2.points == 9)
   break game;
79              p1.pass = false;
80              p2.pass = false;
81              printMsg("NEW ROUND");
82          }
83          printGameBoards(out1, out2);    // Print a
    gameboard indicating the game is over for both
   players
84      }
85
86      /**
87       * Changes the arrows to reflect that the game
    is over
88       */
89      public void updateArrowsForGameOver(){
90          gameboard[2][38] = '-';
91          gameboard[4][38] = '-';
92          gameboard[12][38] = '-';
93          gameboard[14][38] = '-';
94      }
95
96      /**
97       * Prints a loss screen in a relative 5*5
```

```
 97 array starting from (i, j)
 98      */
 99     public void lossScreen(int i, int j){
100 
101         clear5By5(i, j);
102 
103         gameboard[i][j] = '[';
104         gameboard[i][j+1] = '=';
105         gameboard[i][j+2] = 'X';
106         gameboard[i][j+3] = '=';
107         gameboard[i][j+4] = ']';
108 
109         gameboard[i+1][j+1] = 'o';
110         gameboard[i+1][j+3] = 'O';
111         gameboard[i+2][j+2] = 'v';
112 
113         gameboard[i+3][j+1] = '-';
114         gameboard[i+3][j+2] = '-';
115         gameboard[i+3][j+3] = '-';
116 
117         gameboard[i + 4][j] = '[';
118         gameboard[i + 4][j+1] = '=';
119         gameboard[i + 4][j+2] = 'X';
120         gameboard[i + 4][j+3] = '=';
121         gameboard[i + 4][j+4] = ']';
122     }
123 
124     /**
125      * Prints a win screen in a relative 5*5 array
    starting from (i, j)
126      */
127     public void winScreen(int i, int j){
128         char[] arr = "Player Won".toCharArray();
129         clear5By5(i, j);
130 
131         int x = i + 2;
132         int y = j;
133 
134         for(char c : arr){
135             gameboard[x][y] = c;
136             y++;
```

```
137             if(y > j + 5){
138                 y = j;
139                 x++;
140             }
141         }
142     }
143 
144     /**
145      * Clear a 5*5 relative array starting from (i
    , j)
146      */
147     public void clear5By5(int i, int j){
148         int x = i;
149         int y = j;
150 
151         // Reset the space first
152         for(x=i; x < i + 5; x++){
153             for(y=j; y < j + 5; y++) gameboard[x][
    y] = ' ';
154         }
155     }
156 
157     /**
158      * @return true if the round is over, false
    otherwise
159      */
160     public boolean roundEnd(Player p1, Player p2){
161         if(p1.points == 9){
162             return true;
163         } else if(p2.points == 9){
164             return true;
165         } else if(p1.pass && p2.pass){
166             return true;
167         }
168         return false;
169     }
170 
171     public void updatePlayer(){
172         currPlayer = 1 - currPlayer;
173     }
174 
```

```
175     /**
176      * Update Player state based on a player move
177      * @param inputLine the player input
178      * @return true if the move was valid, false
    otherwise
179      */
180     public boolean updateState(String inputLine){
181         char[] inpArray = inputLine.toCharArray();
182         Player p1;
183         Player p2;
184 
185         // Decide who's turn it is and assign p1
    to the attacking player and p2 to the defending
    player
186         if(currPlayer == 0){
187             p1 = this.p1;
188             p2 = this.p2;
189         } else{
190             p1 = this.p2;
191             p2 = this.p1;
192         }
193 
194         // Check is the player passed
195         if(inputLine.equals("PS")) {
196             p1.pass = true;
197             printMsg("PLAYER PASSED");
198             return true;
199         }
200 
201         // Check if the input is of correct length
202         if(inpArray.length != 2) {
203             printMsg("SYNTAX ERROR");
204             return false;
205         }
206 
207         // Check if the selection is out of bounds
     or not allowed and Assign the cards corresponding
     to the attacking and defending player
208         if (!(inpArray[0] >= 'A' && inpArray[0] <
    'G' && inpArray[1] >= 'A' && inpArray[1] < 'G')) {
209             printMsg("SYNTAX ERROR");
```

```
210             return false;
211         }
212 
213         String attacker = p1.cards.get(((int)
    inpArray[0] - 65));
214         String receiver = p2.cards.get(((int)
    inpArray[1] - 65));
215 
216         // Check if the move is valid
217         if(!validateMove(attacker, receiver)){
218             printMsg("INVALID MOVE");
219             return false;
220         }
221 
222         p2.cards.set(((int)inpArray[1] - 65), " "
    );
223 
224         // Only give points if the move does not
    involve arrows as attacking/receiving card
225         if(!(attacker.equals("arrow") || receiver.
    equals("arrow"))) p1.points++;
226 
227         if(p1.points == 9) printMsg(attacker.
    toUpperCase() + " takes " + receiver.toUpperCase
    () + "! Game Over");
228         else printMsg(attacker.toUpperCase() + "
    takes " + receiver.toUpperCase());
229 
230         return true;
231     }
232 
233     /**
234      * Check if a move is valid
235      * @param attacker attacking card
236      * @param receiver defending card
237      * @return true if it valid, false otherwise
238      */
239     public boolean validateMove(String attacker,
    String receiver){
240         boolean valide = false;
241 
```

```
242         if (attacker.equals("axe") && receiver.
    equals("hammer")) valide = true;
243         else if (attacker.equals("hammer") &&
    receiver.equals("sword")) valide = true;
244         else if (attacker.equals("sword") &&
    receiver.equals("axe")) valide = true;
245         else if (attacker.equals("arrow") ||
    receiver.equals("arrow")) valide = true;
246 
247         if (attacker.equals(" ") || receiver.
    equals(" ")) valide = false;
248 
249         return valide;
250     }
251 
252     /**
253      * Print gameboards for both players
254      * @param out1 output stream for player 1
255      * @param out2 output stream for player 2
256      */
257     public void printGameBoards(PrintWriter out1,
    PrintWriter out2){
258         String printable;
259 
260         // Print for the player 1
261         printable = "";
262         char[][] p1Board = getGameboard(p1, p2);
263 
264         for(int i = 0; i < 19; i++){
265             printable += new String(p1Board[i]) +
    "\n";
266         }
267         out1.println(printable.substring(0,
    printable.length() - 1));
268 
269         // print for player 2
270         printable = "";
271         char[][] p2Board = getGameboard(p2, p1);
272 
273         for(int i = 0; i < 19; i++){
274             printable += new String(p2Board[i]) +
```

```
274 "\n";
275         }
276         out2.println(printable.substring(0,
    printable.length() - 1));
277     }
278 
279     /**
280      * Print a message in the message log
281      * @param msg the message to be printed
282      */
283     public void printMsg(String msg){
284 
285         char[] text = msg.toCharArray();
286 
287         if(text.length > 38) throw new Error("The
    message is too long");
288 
289         int i = 17;
290         int j = 1;
291 
292         for(int x = 1; x < 39; x++){
293             gameboard[i][x] = ' ';
294         }
295 
296         for(; j < msg.length() + 1; j++){
297             gameboard[i][j] = text[j - 1];
298         }
299     }
300 
301     /**
302      * Reads the cards both players have and,
    populates the board from player p1's prespective
303      * @param p1 relative player 1 whose board we
    are printing
304      * @param p2 the opponent player
305      * @return a populated gameboard
306      */
307     public char[][] getGameboard(Player p1, Player
     p2) {
308 
309         resetCards();
```

```
310
311         // Set all the cards for p1 and p2
312         setCard(3, 2, p2.cards.get(0));
313         setCard(3, 7, p2.cards.get(1));
314         setCard(3, 12, p2.cards.get(2));
315         setCard(3, 17, p2.cards.get(3));
316         setCard(3, 22, p2.cards.get(4));
317         setCard(3, 27, p2.cards.get(5));
318
319         setCard(11, 2, p1.cards.get(0));
320         setCard(11, 7, p1.cards.get(1));
321         setCard(11, 12, p1.cards.get(2));
322         setCard(11, 17, p1.cards.get(3));
323         setCard(11, 22, p1.cards.get(4));
324         setCard(11, 27, p1.cards.get(5));
325
326         // Check whose turn it is, whose's
    prespective it is and initialize the arrows
327         if(p1.equals(this.p1)){
328             if(currPlayer != 0){
329                 gameboard[2][38] = '^';
330                 gameboard[4][38] = '-';
331
332                 gameboard[12][38] = '^';
333                 gameboard[14][38] = '-';
334             } else {
335                 gameboard[2][38] = '-';
336                 gameboard[4][38] = 'v';
337
338                 gameboard[12][38] = '-';
339                 gameboard[14][38] = 'v';
340             }
341         }else{
342             if(currPlayer != 0 && !p1.equals(this.
    p1)){
343                 gameboard[2][38] = '-';
344                 gameboard[4][38] = 'v';
345
346                 gameboard[12][38] = '-';
347                 gameboard[14][38] = 'v';
348             } else{
```

```
349                 gameboard[2][38] = '^';
350                 gameboard[4][38] = '-';
351 
352                 gameboard[12][38] = '^';
353                 gameboard[14][38] = '-';
354             }
355         }
356 
357         // Print the points and the current round
358         gameboard[7][38] = (char) (p2.points + '0'
    );
359         gameboard[8][38] = (char) (round + '0');
360         gameboard[9][38] = (char) (p1.points + '0'
    );
361 
362         // Print a win/loss message if needed
363         if(p1.points == 9){
364             updateArrowsForGameOver();
365             printMsg("GAME OVER. You Win");
366             winScreen(10, 31);
367             lossScreen(2, 31);
368         } else if(p2.points == 9){
369             updateArrowsForGameOver();
370             printMsg("GAME OVER. You Lose");
371             winScreen(2, 31);
372             lossScreen(10, 31);
373         } else if(round > 4){
374 
375             // When we reach round 6 we exit the
    for loop but it's not really round 6. The game
    ends at round 5
376             // so to reflect that we update the
    round counter in the returned board and reset the
    round back to 6
377             round--;
378             gameboard[8][38] = (char) (round + '0'
    );
379             round++;
380 
381             updateArrowsForGameOver();
382             printMsg("BOTH PLAYERS LOSE");
```

```
383             lossScreen(2, 31);
384             lossScreen(10, 31);
385         }
386 
387         return gameboard;
388     }
389 
390     /**
391      * Sets a card contents in a relative 3*3
    relative array starting from (i, j)
392      * @param card the card we are setting
393      */
394     public void setCard(int i, int j, String card
    ) {
395         if (card.equals("axe")) {
396             setAxe(i, j);
397         } else if (card.equals("hammer")) {
398             setHammer(i, j);
399         } else if (card.equals("arrow")) {
400             setArrow(i, j);
401         } else if (card.equals("sword")) {
402             setSword(i, j);
403         }
404     }
405 
406     /**
407      * Create a hammer in a 3x3 relative array
    starting from (i, j)
408      */
409     public void setHammer(int i, int j) {
410         gameboard[i][j] = '[';
411         gameboard[i][j + 1] = '=';
412         gameboard[i][j + 2] = ']';
413         gameboard[i + 1][j + 1] = '|';
414         gameboard[i + 2][j + 1] = '|';
415     }
416 
417     /**
418      * Create an arrow in a 3x3 relative array
    starting from (i, j)
419      */
```

```
420     public void setArrow(int i, int j) {
421         gameboard[i][j + 1] = '^';
422         gameboard[i + 1][j + 1] = '|';
423         gameboard[i + 2][j + 1] = '^';
424         gameboard[i + 2][j] = '/';
425         gameboard[i + 2][j + 2] = '\\';
426     }
427 
428     /**
429      * Create a sword in a 3x3 relative array
    starting from (i, j)
430      */
431     public void setSword(int i, int j) {
432         gameboard[i][j + 2] = '/';
433         gameboard[i + 1][j + 1] = '/';
434         gameboard[i + 2][j] = 'X';
435     }
436 
437     /**
438      * Create an axe in a 3x3 relative array
    starting from (i, j)
439      */
440     public void setAxe(int i, int j) {
441         gameboard[i][j] = '<';
442         gameboard[i][j + 1] = '7';
443         gameboard[i][j + 2] = '>';
444         gameboard[i + 1][j + 1] = '|';
445         gameboard[i + 2][j + 1] = 'L';
446     }
447 
448 
449     /**
450      * Reset all the card spaces to empty
451      */
452     public void resetCards(){
453         // Set empty inner cards for the top row
454         emptyInnerCard(3, 2);
455         emptyInnerCard(3, 7);
456         emptyInnerCard(3, 12);
457         emptyInnerCard(3, 17);
458         emptyInnerCard(3, 22);
```

```
459         emptyInnerCard(3, 27);
460 
461         // Set empty inner cards for the bottom
    row
462         emptyInnerCard(11, 2);
463         emptyInnerCard(11, 7);
464         emptyInnerCard(11, 12);
465         emptyInnerCard(11, 17);
466         emptyInnerCard(11, 22);
467         emptyInnerCard(11, 27);
468 
469     }
470 
471     /**
472      * Empty the card in a 3*3 relative array
    space starting from (i, j)
473      */
474     public void emptyInnerCard(int i, int j){
475         for(int x = i; x < i + 3; x++){
476             for(int y = j; y < j + 3; y++){
477                 gameboard[x][y] = ' ';
478             }
479         }
480     }
481 
482     /**
483      * Initialize an empty 19 * 40 array and set
    up the gameboard on it
484      */
485     public void initializeBoard() {
486         gameboard = new char[19][40];
487 
488         // Initialize the array
489         for (int i = 0; i < 19; i++) {
490             for (int j = 0; j < 40; j++) {
491                 gameboard[i][j] = ' ';
492             }
493         }
494         setBorders();
495         initializeEmptyCards();
496         initializeIdentifiers();
```

```
497
498         // Set up some more miscellaneous
    constants.
499         gameboard[3][38] = '|';
500         gameboard[13][38] = '|';
501
502         gameboard[7][37] = '[';
503         gameboard[7][39] = ']';
504         gameboard[8][37] = 'R';
505         gameboard[8][39] = '<';
506         gameboard[9][37] = '[';
507         gameboard[9][39] = ']';
508     }
509
510     public void initializeIdentifiers(){
511         int j = 0;
512         char[] letters = new char[]{'A', 'B', 'C'
    , 'D', 'E', 'F'};
513         for(int i = 3; i < 31; i+=5){
514             gameboard[1][i] = letters[j];
515             gameboard[15][i] = letters[j];
516             j++;
517         }
518     }
519
520     public void initializeEmptyCards(){
521
522         // Set empty cards for the top row
523         createEmptyCard(2, 1);
524         createEmptyCard(2, 6);
525         createEmptyCard(2, 11);
526         createEmptyCard(2, 16);
527         createEmptyCard(2, 21);
528         createEmptyCard(2, 26);
529
530         // Set empty cards for the bottom row
531         createEmptyCard(10, 1);
532         createEmptyCard(10, 6);
533         createEmptyCard(10, 11);
534         createEmptyCard(10, 16);
535         createEmptyCard(10, 21);
```

```
536         createEmptyCard(10, 26);
537     }
538 
539     /**
540      * Creates an empty card in a relative 3*3
    space starting from (i, j)
541      */
542     public void createEmptyCard(int i, int j){
543         gameboard[i][j] = '/';
544         gameboard[i][j + 4] = '\\';
545         gameboard[i+4][j] = '\\';
546         gameboard[i+4][j+4] = '/';
547         for(int k = i + 1; k < i + 4; k++){
548             gameboard[k][j] = '|';
549             gameboard[k][j + 4] = '|';
550         }
551         for(int k = j + 1; k < j + 4; k++){
552             gameboard[i][k] = '-';
553             gameboard[i + 4][k] = '-';
554         }
555     }
556 
557     /**
558      * Set the top and bottom borders. Also set
    the border for the message log and the player
    separator.
559      */
560     public void setBorders(){
561         gameboard[0][0] = '/';
562         gameboard[0][39] = '\\';
563         for(int i = 1; i < 39; i++){
564             gameboard[0][i] = '-';
565             gameboard[18][i] = '-';
566         }
567         gameboard[18][0] = '\\';
568         gameboard[18][39] = '/';
569 
570         gameboard[17][0] = '|';
571         gameboard[17][39] = '|';
572 
573         gameboard[16][0] = '|';
```

```
574         gameboard[16][39] = '|';
575         for(int i = 1; i < 39; i++) gameboard[16][
    i] = '-';
576 
577 
578         gameboard[8][0] = '<';
579         gameboard[8][36] = '>';
580         for(int i = 1; i < 36; i++) gameboard[8][i
    ] = '=';
581     }
582 
583     /**
584      * Deals new cards to both players from a
    randomly shuffled deck
585      */
586     public void dealCards(Player p1, Player p2){
587         ArrayList<String> deck = getDeck();
588         p1.cards.clear();
589         p2.cards.clear();
590 
591         for(int i = 0; i < 6; i++){
592             p1.cards.add(deck.get(i));
593             p2.cards.add(deck.get(i + 6));
594         }
595     }
596 
597     /**
598      * @return a randomly shuffled deck of cards
599      */
600     public ArrayList<String> getDeck(){
601         ArrayList<String> deck = new ArrayList
    <>();
602         deck.add("axe");
603         deck.add("axe");
604         deck.add("axe");
605         deck.add("hammer");
606         deck.add("hammer");
607         deck.add("hammer");
608         deck.add("sword");
609         deck.add("sword");
610         deck.add("sword");
```

```
611             deck.add("arrow");
612             deck.add("arrow");
613             deck.add("arrow");
614 
615             Collections.shuffle(deck);
616 
617             return deck;
618         }
619 }
620 
```

```
 1
 2 import java.io.BufferedReader;
 3 import java.io.IOException;
 4 import java.io.InputStreamReader;
 5 import java.io.PrintWriter;
 6 import java.net.Socket;
 7 import java.net.UnknownHostException;
 8
 9 /**
10  * Class that manages the GUI client
11  */
12 public class Client {
13
14     public static void main(String[] args) {
15
16         GUI gui = new GUI();
17         String returnValue;
18
19         String hostName="localhost";
20         int portNumber=35754;
21         try (
22                 Socket conn=new Socket(hostName,
   portNumber);
23                 PrintWriter sockOut=new PrintWriter
   (conn.getOutputStream(),true);
24                 BufferedReader sockIn=new
   BufferedReader(new InputStreamReader(conn.
   getInputStream()));
25         ) {
26             String fromServer;
27             char[][] gameboard = new char[19][40];
28
29             game:
30             while (true) {
31
32                 gui.updateButtons();
33                 if(gameboard[4][38] == '-' &&
   gameboard[2][38] == '-') break;
34
35                 for(int i = 0; i < 19; i++) {
36                     fromServer = sockIn.readLine();
```

```
37
38                      // Incase if the other player
   leaves the game
39                      if(fromServer == null) break
   game;
40
41                      if(fromServer.equals(
   "/----------------------------------------\\") && i
    != 0){
42                          i = 0;
43                          continue;
44                      }
45
46                      gameboard[i] = fromServer.
   toCharArray();
47                  }
48
49                  // If it's not our turn then wait
   for input by going back to the start of the while
   loop
50                  if(gameboard[4][38] != 'v') {
51                      gui.turn = false;
52                      gui.updateLayout(gameboard);
53                      continue;
54                  }
55
56                  gui.turn = true;
57                  gui.updateButtons();
58                  gui.updateLayout(gameboard);
59
60                  while(true) {
61
62                      sockOut.flush();
63
64                      if((returnValue = gui.
   checkForMove()) != ""){
65                          sockOut.println(returnValue
   );
66                          break;
67                      }
68                  }
```

```
69              }
70          } catch (UnknownHostException e) {
71              System.out.println("I think there's a
    problem with the host name.");
72          } catch (IOException e) {
73              System.out.println("Had an IO error
    for the connection.");
74          }
75      }
76 }
77
```

```java
import java.util.ArrayList;

/**
 * Player class to keep information about a player in one unit
 */
public class Player {
    int points;
    ArrayList<String> cards;
    boolean pass;

    Player(){
        points = 0;
        cards = new ArrayList<>(5);
        pass = false;
    }
}

```

```
 1 import java.net.*;
 2 import java.io.*;
 3
 4 /**
 5  * A Threaded server class which pairs people in a
   game.
 6  */
 7 class ThreadedServer {
 8     public static void main(String[] args) {
 9         int portNumber = 35754;
10         try (
11                 ServerSocket serverSocket = new
   ServerSocket(portNumber);
12         ) {
13             while (true) {
14                 Socket client1=serverSocket.accept
   ();
15                 Socket client2=serverSocket.accept
   ();
16                 new ConnectionThread(client1,
   client2).start();
17             }
18         } catch (IOException e) {
19             System.out.println("Exception caught
   when trying to listen on port " + portNumber + " or
    listening for a connection");
20             System.out.println(e.getMessage());
21         }
22     }
23 }
```

```java
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.PrintWriter;
import java.net.Socket;

/**
 * Thread which facilitates a game between 2 people
 */
public class ConnectionThread extends Thread{
    private Socket client1,client2;

    public ConnectionThread(Socket c1, Socket c2) {
        client1=c1; client2=c2;
    }

    public void run() {
        try (
                // Set up inputs and outputs for both clients
                PrintWriter out1 = new PrintWriter(client1.getOutputStream(), true);
                PrintWriter out2 = new PrintWriter(client2.getOutputStream(), true);
                BufferedReader in1 = new BufferedReader(new InputStreamReader(client1.getInputStream()));
                BufferedReader in2 = new BufferedReader(new InputStreamReader(client2.getInputStream()));
        ) {
            new Game(in1, in2, out1, out2);
        } catch (IOException e) {
            System.out.println("Ah nertz.");
        }
    }
}
```