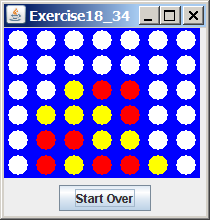
**Final Project: Connect Four**

Problem Description:

The applet in the figure below enables two players to play the connect-four game. The applet enables two players to place red and yellow discs in turn. To place a disk, the player needs to click on an available cell. An *available cell* is unoccupied and whose downward neighbor is occupied. The applet flashes the four winning cells if a player wins and reports no winners if all cells are occupied with no winners.



Your task is to modify the given program to enable a user to play with the computer. Things need to do with projects:

1. Comprehend the given program. The source code for the program is given below named Exercise18\_34.
2. Modify it to enable a user to play with the computer. Let the computer play as intelligently as you can. The bottom line is to make a legal play.
3. Let the user start first.
4. Name your PConnectFour.

Submit the following items:

1. Compile and Submit to LiveLab (you must submit the program regardless whether it complete or incomplete, correct or incorrect)

Here is the source code for Exercise18\_34:

**import** java.awt.\*;

**import** java.awt.event.\*;

**import** javax.swing.\*;

**public** **class** Exercise18\_34 **extends** JApplet {

**private** Cell[][] cells = **new** Cell[6][7];

**private** **char** nextDisc = 'R';

**private** Timer timer = **new** Timer(100, **new** FlashingCells());

**private** **int**[][] result;

**private** JButton jbtStartOver = **new** JButton("Start Over");

**class** FlashingCells **implements** ActionListener {

**public** **void** actionPerformed(ActionEvent e) {

**if** (cells[result[0][0]][result[0][1]].token == ' ') {

cells[result[0][0]][result[0][1]].token = nextDisc;

cells[result[1][0]][result[1][1]].token = nextDisc;

cells[result[2][0]][result[2][1]].token = nextDisc;

cells[result[3][0]][result[3][1]].token = nextDisc;

}

**else** {

cells[result[0][0]][result[0][1]].token = ' ';

cells[result[1][0]][result[1][1]].token = ' ';

cells[result[2][0]][result[2][1]].token = ' ';

cells[result[3][0]][result[3][1]].token = ' ';

}

repaint();

}

}

**public** Exercise18\_34() {

JPanel panel1 = **new** JPanel(**new** GridLayout(6, 7));

**for** (**int** i = 0; i < cells.length; i++)

**for** (**int** j = 0; j < cells[i].length; j++)

panel1.add(cells[i][j] = **new** Cell(i, j));

add(panel1, BorderLayout.*CENTER*);

JPanel panel2 = **new** JPanel();

panel2.add(jbtStartOver);

add(panel2, BorderLayout.*SOUTH*);

jbtStartOver.addActionListener(**new** ActionListener() {

**public** **void** actionPerformed(ActionEvent e) {

timer.stop();

nextDisc = 'R';

result = **null**;

**for** (**int** i = 0; i < cells.length; i++)

**for** (**int** j = 0; j < cells[i].length; j++)

cells[i][j].token = ' ';

repaint();

}

});

}

**class** Cell **extends** JPanel {

**char** token = ' ';

**int** i, j;

**boolean** isFlashing;

**private** **boolean** available() {

**return** (token == ' ' && (i == 5 || cells[i + 1][j].token != ' '));

}

**public** Cell(**int** i, **int** j) {

**this**.i = i; **this**.j = j;

**this**.addMouseListener(**new** MouseAdapter() {

**public** **void** mousePressed(MouseEvent e) {

**if** (result != **null**)

**return**; // Game is over

**if** (available()) {

token = nextDisc;

result = *isConsecutiveFour*(cells);

**if** (result != **null**) {

timer.start();

}

**else** **if** (nextDisc == 'R')

nextDisc = 'Y';

**else**

nextDisc = 'R';

repaint();

}

}

});

}

**protected** **void** paintComponent(Graphics g) {

**int** radius = Math.*min*(getWidth(), getHeight()) \* 4 / 10;

g.setColor(Color.*BLUE*);

g.fillRect(0, 0, getWidth(), getHeight());

**if** (token == ' ') {

g.setColor(Color.*WHITE*);

g.fillOval(getWidth() / 2 - radius, getHeight() / 2 - radius,

radius \* 2, radius \* 2);

}

**else** **if** (token == 'R') {

g.setColor(Color.*RED*);

g.fillOval(getWidth() / 2 - radius, getHeight() / 2 - radius,

radius \* 2, radius \* 2);

}

**else** **if** (token == 'Y') {

g.setColor(Color.*YELLOW*);

g.fillOval(getWidth() / 2 - radius, getHeight() / 2 - radius,

radius \* 2, radius \* 2);

}

}

}

**public** **static** **int**[][] isConsecutiveFour(Cell[][] cells) {

**char**[][] values = **new** **char**[cells.length][cells[0].length];

**for** (**int** i = 0; i < cells.length; i++)

**for** (**int** j = 0; j < cells[i].length; j++)

values[i][j] = cells[i][j].token;

**return** *isConsecutiveFour*(values);

}

**public** **static** **int**[][] isConsecutiveFour(**char**[][] values) {

**int** numberOfRows = values.length;

**int** numberOfColumns = values[0].length;

// Check rows

**for** (**int** i = 0; i < numberOfRows; i++) {

**if** (*isConsecutiveFour*(values[i]) != **null**) {

**int**[][] result = **new** **int**[4][2];

result[0][0] = result[1][0] = result[2][0] = result[3][0] = i;

**int** k = *isConsecutiveFour*(values[i]);

result[0][1] = k; result[1][1] = k + 1;

result[2][1] = k + 2; result[3][1] = k + 3;

**return** result;

}

}

// Check columns

**for** (**int** j = 0; j < numberOfColumns; j++) {

**char**[] column = **new** **char**[numberOfRows];

// Get a column into an array

**for** (**int** i = 0; i < numberOfRows; i++)

column[i] = values[i][j];

**if** (*isConsecutiveFour*(column) != **null**) {

**int**[][] result = **new** **int**[4][2];

result[0][1] = result[1][1] = result[2][1] = result[3][1] = j;

**int** k = *isConsecutiveFour*(column);

result[0][0] = k; result[1][0] = k + 1;

result[2][0] = k + 2; result[3][0] = k + 3;

**return** result;

}

}

// Check major diagonal (lower part)

**for** (**int** i = 0; i < numberOfRows - 3; i++) {

**int** numberOfElementsInDiagonal

= Math.*min*(numberOfRows - i, numberOfColumns);

**char**[] diagonal = **new** **char**[numberOfElementsInDiagonal];

**for** (**int** k = 0; k < numberOfElementsInDiagonal; k++)

diagonal[k] = values[k + i][k];

**if** (*isConsecutiveFour*(diagonal) != **null**) {

**int**[][] result = **new** **int**[4][2];

**int** k = *isConsecutiveFour*(diagonal);

result[0][0] = k + i;

result[1][0] = k + 1 + i;

result[2][0] = k + 2 + i;

result[3][0] = k + 3 + i;

result[0][1] = k;

result[1][1] = k + 1;

result[2][1] = k + 2;

result[3][1] = k + 3;

**return** result;

}

}

// Check major diagonal (upper part)

**for** (**int** j = 1; j < numberOfColumns - 3; j++) {

**int** numberOfElementsInDiagonal

= Math.*min*(numberOfColumns - j, numberOfRows);

**char**[] diagonal = **new** **char**[numberOfElementsInDiagonal];

**for** (**int** k = 0; k < numberOfElementsInDiagonal; k++)

diagonal[k] = values[k][k + j];

**if** (*isConsecutiveFour*(diagonal) != **null**) {

**int**[][] result = **new** **int**[4][2];

**int** k = *isConsecutiveFour*(diagonal);

result[0][0] = k;

result[1][0] = k + 1;

result[2][0] = k + 2;

result[3][0] = k + 3;

result[0][1] = k + j;

result[1][1] = k + 1 + j;

result[2][1] = k + 2 + j;

result[3][1] = k + 3 + j;

**return** result;

}

}

// Check sub-diagonal (left part)

**for** (**int** j = 3; j < numberOfColumns; j++) {

**int** numberOfElementsInDiagonal

= Math.*min*(j + 1, numberOfRows);

**char**[] diagonal = **new** **char**[numberOfElementsInDiagonal];

**for** (**int** k = 0; k < numberOfElementsInDiagonal; k++)

diagonal[k] = values[k][j - k];

**if** (*isConsecutiveFour*(diagonal) != **null**) {

**int**[][] result = **new** **int**[4][2];

**int** k = *isConsecutiveFour*(diagonal);

result[0][0] = k;

result[1][0] = k + 1;

result[2][0] = k + 2;

result[3][0] = k + 3;

result[0][1] = j - k;

result[1][1] = j - k - 1;

result[2][1] = j - k - 2;

result[3][1] = j - k - 3;

**return** result;

}

}

// Check sub-diagonal (right part)

**for** (**int** i = 1; i < numberOfRows - 3; i++) {

**int** numberOfElementsInDiagonal

= Math.*min*(numberOfRows - i, numberOfColumns);

**char**[] diagonal = **new** **char**[numberOfElementsInDiagonal];

**for** (**int** k = 0; k < numberOfElementsInDiagonal; k++)

diagonal[k] = values[k + i][numberOfColumns - k - 1];

**if** (*isConsecutiveFour*(diagonal) != **null**) {

**int**[][] result = **new** **int**[4][2];

**int** k = *isConsecutiveFour*(diagonal);

result[0][0] = k + i;

result[1][0] = k + i + 1;

result[2][0] = k + i + 2;

result[3][0] = k + i + 3;

result[0][1] = numberOfColumns - k - 1;

result[1][1] = numberOfColumns - (k + 1) - 1;

result[2][1] = numberOfColumns - (k + 2) - 1;

result[3][1] = numberOfColumns - (k + 3) - 1;

**return** result;

}

}

**return** **null**;

}

**public** **static** Integer isConsecutiveFour(**char**[] values) {

**for** (**int** i = 0; i < values.length - 3; i++) {

**boolean** isEqual = **true**;

**for** (**int** j = i; j < i + 3; j++) {

**if** (values[j] == ' ' || values[j] != values[j + 1]) {

isEqual = **false**;

**break**;

}

}

**if** (isEqual) **return** **new** Integer(i);

}

**return** **null**;

}

**public** **static** **void** main(String[] args) {

JFrame frame = **new** JFrame("Exercise18\_34");

JApplet applet = **new** Exercise18\_34();

frame.add(applet);

frame.setSize(210, 220);

frame.setDefaultCloseOperation(JFrame.*EXIT\_ON\_CLOSE*);

frame.setLocationRelativeTo(**null**);

frame.setVisible(**true**);

}

}