1 import javax.swing.\*;  
 2 /\*\* The Driver of Snake, where it all begins.  
 3 \* IMPORTANT--> See Snake class and OptionPanel class for data structure  
 4 \* All of this was done by Raynor Kuang  
 5 \* Version: 6/1/2011\*/  
 6 public class SnakeDriver {  
 7 /\*\*Frame of the game.\*/  
 8 private static JFrame frame;  
 9 /\*\*Runs the game.\*/  
10 public static void main(String[] args) {  
11 initialize(); }  
12 /\*\*Actually initializes the program.\*/  
13 private static void initialize() {  
14 frame = new JFrame("Final Project: Snake!");  
15 frame.setSize(600, 650);  
16 frame.setLocation(200, 100);  
17 frame.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);  
18 MenuPanel m = new MenuPanel(true);  
19 frame.setContentPane(m);  
20 m.requestFocus();  
21 frame.setVisible(true); }  
22 /\*\*Reloads the frame with a new panel.  
23 \*@param j JPanel run in the game frame  
24 \*@param s Title of the panel \*/  
25 public static void reload(JPanel j, String s) {  
26 frame.remove(frame.getContentPane());  
27 frame.setTitle(s);  
28 frame.setContentPane(j);  
29 j.requestFocus();  
30 frame.setVisible(true); } }

1 import javax.swing.\*;  
 2 import java.awt.\*;  
 3 import java.awt.event.\*;  
 4 import java.util.\*;  
 5 /\*\*MenuPanel that opens when the game runs\*/  
 6 public class MenuPanel extends javax.swing.JPanel  
 7 {  
 8 /\*\*OptionPanel that opens options screen\*/   
 9 private OptionPanel optPan;  
 10 /\*\*SnakePanel that runs game screen\*/  
 11 private SnakePanel snakePan;  
 12 /\*\*Map of the possible input keys and their actual input keys\*/  
 13 public static Map<Integer, Integer> myKeys;  
 14 /\*\*Snake color\*/  
 15 public static Color sc = Color.green;  
 16 /\*\*Food color\*/  
 17 public static Color fc = Color.black;  
 18 /\*\*Background color\*/  
 19 public static Color bgc = Color.yellow.brighter().brighter();  
 20 /\*\* Creates new MainPanel with buttons\*/  
 21 public MenuPanel(boolean firstOpen) {  
 22 setLayout(new GridLayout(5,0));  
 23 JLabel title = new JLabel("SNAKE", JLabel.CENTER);  
 24 add(title);  
 25 JButton snakeButton = new javax.swing.JButton("SNAKE");  
 26 add(snakeButton);  
 27 JButton optButton = new javax.swing.JButton("Options");  
 28 add(optButton);  
 29 JButton tutorialButton = new JButton("Instruction");  
 30 add(tutorialButton);  
 31 JButton creditsButton = new JButton("Credits");  
 32 add(creditsButton);  
 33 optButton.addActionListener(  
 34 new java.awt.event.ActionListener() {  
 35 public void actionPerformed(java.awt.event.ActionEvent evt) {  
 36 optButtonActionPerformed(evt); } });  
 37 snakeButton.addActionListener(  
 38 new java.awt.event.ActionListener() {  
 39 public void actionPerformed(java.awt.event.ActionEvent evt) {  
 40 snakeButtonActionPerformed(evt); } });  
 41 tutorialButton.addActionListener(  
 42 new java.awt.event.ActionListener() {  
 43 public void actionPerformed(java.awt.event.ActionEvent evt) {  
 44 tutButtonActionPerformed(evt); } });  
 45 creditsButton.addActionListener(  
 46 new java.awt.event.ActionListener() {  
 47 public void actionPerformed(java.awt.event.ActionEvent evt) {  
 48 credButtonActionPerformed(evt); } });  
 49 if(firstOpen) {   
 50 myKeys = new HashMap<Integer, Integer>();  
 51 myKeys.put(KeyEvent.VK\_LEFT, KeyEvent.VK\_LEFT);  
 52 myKeys.put(KeyEvent.VK\_RIGHT, KeyEvent.VK\_RIGHT);  
 53 myKeys.put(KeyEvent.VK\_P, KeyEvent.VK\_P);  
 54 myKeys.put(KeyEvent.VK\_ESCAPE, KeyEvent.VK\_ESCAPE);  
 55 myKeys.put(KeyEvent.VK\_R, KeyEvent.VK\_R);  
 56 myKeys.put(KeyEvent.VK\_SPACE, KeyEvent.VK\_SPACE); }  
 57 }  
 58 /\*\*Actions of option button\*/  
 59 private void optButtonActionPerformed(java.awt.event.ActionEvent evt) {  
 60 optPan = new OptionPanel(myKeys, sc, fc, bgc);  
 61 SnakeDriver.reload(optPan, "Final Project: SNAKE!"); }  
 62 /\*\*Actions of game button\*/  
 63 private void snakeButtonActionPerformed(java.awt.event.ActionEvent evt) {  
 64 DiffPanel diff = new DiffPanel(myKeys, sc, fc, bgc);  
 65 SnakeDriver.reload(diff, "Final Project: SNAKE!"); }  
 66 /\*\*Displays tutorial\*/  
 67 private void tutButtonActionPerformed(java.awt.event.ActionEvent evt) {  
 68 String s = "Press the left arrow key to turn counter-clockwise and the right to turn clockwise;";  
 69 s+="\n you move forward automatically.";  
 70 s+="\n Press p to pause, r to reset the game, the escape key to return to the main menu, and the space bar to mute/unmute music.";  
 71 s+="\nThese keys can be changed in the option menu.";  
 72 s+="\nEat food (the circles) to increase your score and grow in length.";  
 73 s+="\nIf you hit the wall or hit yourself, it's game over!";  
 74 s+="\nIf you reach your level \* 3000 points an opening will appear in the wall.";  
 75 s+="\nGo through it to get to the next level.";  
 76 s+="\nThere are 4 levels in total, but you're not supposed to be able to beat them all.";   
 77 s+="\nThe faster you beat the levels, the more points you get.";  
 78 s+="\nGet as many points as you can and enter the highscores!";  
 79 s+="\nChoose your difficulty when you press the play button.";  
 80 JOptionPane.showMessageDialog(null, s, "Instructions", JOptionPane.INFORMATION\_MESSAGE); }   
 81 private void credButtonActionPerformed(java.awt.event.ActionEvent evt) {  
 82 String s = "I don't always spent copious amounts of time working on projects,";  
 83 s+="\nbut when I do it's to make Snake.";  
 84 s+="\nStay thirsty my friends.";  
 85 s+="\n--Le Raynor Kuang";   
 86 s+="\n(And many additional thanks to the Internet, for always being there for me,";  
 87 s+="\nto NetBeans for making my life easier with GUI's,";  
 88 s+="\nto Mr. Rose for being a great teacher,";  
 89 s+="\nand to Scott Pilgrim the Game, from which I took this awesome music)";  
 90 JOptionPane.showMessageDialog(null, s, "Instructions", JOptionPane.INFORMATION\_MESSAGE); }  
 91 public static void setKeys(Map<Integer, Integer> keys) {  
 92 myKeys = keys; }  
 93 public static void setColors(Color snakec, Color foodc, Color backc) {  
 94 sc = snakec;  
 95 fc = foodc;  
 96 bgc = backc; }  
 97 public void paint(Graphics g) {  
 98 ImageIcon snakeTitle = new ImageIcon("snaketitle.png");  
 99 super.paint(g);  
100 g.drawImage(snakeTitle.getImage(), 300 - (snakeTitle.getIconWidth() / 2), 10, getParent()); } }

1 import java.awt.\*;  
 2 import java.awt.event.\*;  
 3 import java.awt.image.\*;  
 4 import java.awt.geom.\*;  
 5 import javax.swing.\*;  
 6 import java.io.\*;  
 7 import java.util.Map;  
 8   
 9 public class OptionPanel extends javax.swing.JPanel {  
 10 /\*\*Map containing all the important actions and the respective keyboard locations\*/  
 11 private Map<Integer, Integer> myKeys;  
 12 /\*\*Current key being changed\*/  
 13 private Integer current;  
 14 /\*\*Timer to update color of Snake, Food, and Background\*/  
 15 private Timer timer;  
 16 /\*\*Colors of snake, food, and background\*/  
 17 private Color sc, fc, bgc;  
 18 /\*\*Examples of slider colors\*/  
 19 private javax.swing.JLabel backgroundLabel, snakeLabel, foodLabel;  
 20 /\*\*Sliders for background color\*/  
 21 private javax.swing.JSlider bgb, bgg, bgr;  
 22 /\*\*Buttons that can be clicked to change keys\*/  
 23 private javax.swing.JButton escape, left, right, menu, pause, reset, mute;  
 24 /\*\*Sliders for Food color\*/  
 25 private javax.swing.JSlider fb, fg, fr;  
 26 /\*\*Instructions on key changing\*/  
 27 private javax.swing.JLabel jLabel1;  
 28 /\*\*Panel used in GUI\*/  
 29 private javax.swing.JPanel jPanel1;  
 30 /\*\*Sliders for Snake color\*/  
 31 private javax.swing.JSlider sb, sg, sr;  
 32 /\*\* Creates new form OptionPanel \*/  
 33 public OptionPanel(Map<Integer, Integer> keys, Color snakec, Color foodc, Color backgroundc) {  
 34 myKeys = keys;  
 35 sc = snakec;  
 36 fc = foodc;  
 37 bgc = backgroundc;  
 38 timer = new Timer(1, new Listener());  
 39 timer.start();  
 40 initComponents(); }  
 41 /\*\*Action when left button is clicked\*/  
 42 private void leftActionPerformed(java.awt.event.ActionEvent evt) {  
 43 current = KeyEvent.VK\_LEFT;  
 44 requestFocus(); }  
 45 /\*\*Action when right button is clicked\*/   
 46 private void rightActionPerformed(java.awt.event.ActionEvent evt) {  
 47 current = KeyEvent.VK\_RIGHT;  
 48 requestFocus(); }  
 49 /\*\*Action when pause button is clicked\*/  
 50 private void pauseActionPerformed(java.awt.event.ActionEvent evt) {  
 51 current = KeyEvent.VK\_P;  
 52 requestFocus(); }  
 53 /\*\*Action when escape button is clicked\*/  
 54 private void escapeActionPerformed(java.awt.event.ActionEvent evt) {  
 55 current = KeyEvent.VK\_ESCAPE;  
 56 requestFocus(); }  
 57 /\*\*Action when reset button is clicked\*/  
 58 private void resetActionPerformed(java.awt.event.ActionEvent evt) {  
 59 current = KeyEvent.VK\_R;  
 60 requestFocus(); }  
 61 /\*\*Action when mute button is clicked\*/  
 62 private void muteActionPerformed(java.awt.event.ActionEvent evt) {  
 63 current = KeyEvent.VK\_SPACE;  
 64 requestFocus(); }  
 65 /\*\*Takes user back to the menu\*/  
 66 private void menuActionPerformed(java.awt.event.ActionEvent evt) {  
 67 MenuPanel.setKeys(myKeys);  
 68 MenuPanel.setColors(sc, fc, bgc);  
 69 SnakeDriver.reload(new MenuPanel(false), "Final Project: Snake!"); }  
 70 /\*\*Replaces current key with input\*/  
 71 private void formKeyPressed(java.awt.event.KeyEvent evt) {  
 72 myKeys.put(current, evt.getKeyCode()); }  
 73 /\*\*Events that occur every second\*/  
 74 public class Listener implements ActionListener {  
 75 public void actionPerformed(ActionEvent e) {  
 76 left.setText("Turn counter-clockwise: " + KeyEvent.getKeyText(myKeys.get(KeyEvent.VK\_LEFT)));  
 77 right.setText("Turn clockwise: "+ KeyEvent.getKeyText(myKeys.get(KeyEvent.VK\_RIGHT)));  
 78 pause.setText("Pause: " + KeyEvent.getKeyText(myKeys.get(KeyEvent.VK\_P)));  
 79 escape.setText("Return to main menu from game: " + KeyEvent.getKeyText(myKeys.get(KeyEvent.VK\_ESCAPE)));  
 80 reset.setText("Reset game: " + KeyEvent.getKeyText(myKeys.get(KeyEvent.VK\_R)));  
 81 mute.setText("Mute/Unmute: " + KeyEvent.getKeyText(myKeys.get(KeyEvent.VK\_SPACE)));  
 82 sc = new Color(sr.getValue(), sg.getValue(), sb.getValue());  
 83 fc = new Color(fr.getValue(), fg.getValue(), fb.getValue());  
 84 bgc = new Color(bgr.getValue(), bgg.getValue(), bgb.getValue());  
 85 snakeLabel.setBackground(sc);  
 86 foodLabel.setBackground(fc);  
 87 backgroundLabel.setBackground(bgc); } }  
 88 /\*\*Create all elements of GUI\*/  
 89 private void initComponents() {  
 90   
 91 jPanel1 = new javax.swing.JPanel();  
 92 left = new javax.swing.JButton();  
 93 right = new javax.swing.JButton();  
 94 jLabel1 = new javax.swing.JLabel();  
 95 sr = new javax.swing.JSlider();  
 96 bgr = new javax.swing.JSlider();  
 97 bgg = new javax.swing.JSlider();  
 98 bgb = new javax.swing.JSlider();  
 99 backgroundLabel = new javax.swing.JLabel();  
100 pause = new javax.swing.JButton();  
101 escape = new javax.swing.JButton();  
102 reset = new javax.swing.JButton();  
103 mute = new javax.swing.JButton();  
104 sb = new javax.swing.JSlider();  
105 sg = new javax.swing.JSlider();  
106 menu = new javax.swing.JButton();  
107 snakeLabel = new javax.swing.JLabel();  
108 foodLabel = new javax.swing.JLabel();  
109 fr = new javax.swing.JSlider();  
110 fg = new javax.swing.JSlider();  
111 fb = new javax.swing.JSlider();  
112   
113 javax.swing.GroupLayout jPanel1Layout = new javax.swing.GroupLayout(jPanel1);  
114 jPanel1.setLayout(jPanel1Layout);  
115 jPanel1Layout.setHorizontalGroup(  
116 jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)  
117 .addGap(0, 100, Short.MAX\_VALUE) );  
118 jPanel1Layout.setVerticalGroup(  
119 jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)  
120 .addGap(0, 100, Short.MAX\_VALUE) );  
121   
122 setBackground(new java.awt.Color(255, 255, 0));  
123 setPreferredSize(new java.awt.Dimension(600, 616));  
124 addKeyListener(  
125 new java.awt.event.KeyAdapter() {  
126 public void keyPressed(java.awt.event.KeyEvent evt) {  
127 formKeyPressed(evt); } });  
128   
129 left.setBackground(new java.awt.Color(255, 255, 0));  
130 left.setFont(new java.awt.Font("Tahoma", 1, 14)); // NOI18N  
131 left.setForeground(new java.awt.Color(0, 153, 0));  
132 left.setText("Turn counter-clockwise");  
133 left.addActionListener(  
134 new java.awt.event.ActionListener() {  
135 public void actionPerformed(java.awt.event.ActionEvent evt) {  
136 leftActionPerformed(evt); } });  
137   
138 right.setBackground(new java.awt.Color(255, 255, 0));  
139 right.setFont(new java.awt.Font("Tahoma", 1, 14)); // NOI18N  
140 right.setForeground(new java.awt.Color(0, 153, 0));  
141 right.setText("Turn clockwise");  
142 right.addActionListener(  
143 new java.awt.event.ActionListener() {  
144 public void actionPerformed(java.awt.event.ActionEvent evt) {  
145 rightActionPerformed(evt); } });  
146   
147 jLabel1.setBackground(new java.awt.Color(0, 255, 255));  
148 jLabel1.setFont(new java.awt.Font("Tahoma", 1, 14)); // NOI18N  
149 jLabel1.setHorizontalAlignment(javax.swing.SwingConstants.CENTER);  
150 jLabel1.setText("Click the desired button of change. Then press new key.");  
151   
152 sr.setBackground(new java.awt.Color(255, 255, 0));  
153 sr.setMaximum(255);  
154 sr.setValue(sc.getRed());  
155   
156 bgr.setBackground(new java.awt.Color(255, 255, 0));  
157 bgr.setMaximum(240);  
158 bgr.setValue(bgc.getRed());  
159   
160 bgg.setBackground(new java.awt.Color(255, 255, 0));  
161 bgg.setMaximum(240);  
162 bgg.setValue(bgc.getGreen());  
163   
164 bgb.setBackground(new java.awt.Color(255, 255, 0));  
165 bgb.setMaximum(240);  
166 bgb.setValue(bgc.getBlue());  
167   
168 backgroundLabel.setFont(new java.awt.Font("Tahoma", 1, 14)); // NOI18N  
169 backgroundLabel.setHorizontalAlignment(javax.swing.SwingConstants.CENTER);  
170 backgroundLabel.setText("Background Color");  
171 backgroundLabel.setBorder(javax.swing.BorderFactory.createEtchedBorder(new java.awt.Color(0, 0, 0), null));  
172 backgroundLabel.setOpaque(true);  
173   
174 pause.setBackground(new java.awt.Color(255, 255, 0));  
175 pause.setFont(new java.awt.Font("Tahoma", 1, 14)); // NOI18N  
176 pause.setForeground(new java.awt.Color(0, 153, 0));  
177 pause.setText("Pause");  
178 pause.addActionListener(  
179 new java.awt.event.ActionListener() {  
180 public void actionPerformed(java.awt.event.ActionEvent evt) {  
181 pauseActionPerformed(evt); } });  
182   
183 escape.setBackground(new java.awt.Color(255, 255, 0));  
184 escape.setFont(new java.awt.Font("Tahoma", 1, 14)); // NOI18N  
185 escape.setForeground(new java.awt.Color(0, 153, 0));  
186 escape.setText("Return to main menu");  
187 escape.addActionListener(  
188 new java.awt.event.ActionListener() {  
189 public void actionPerformed(java.awt.event.ActionEvent evt) {  
190 escapeActionPerformed(evt); } });  
191   
192 reset.setBackground(new java.awt.Color(255, 255, 0));  
193 reset.setFont(new java.awt.Font("Tahoma", 1, 14)); // NOI18N  
194 reset.setForeground(new java.awt.Color(0, 153, 0));  
195 reset.setText("Reset game");  
196 reset.addActionListener(  
197 new java.awt.event.ActionListener() {  
198 public void actionPerformed(java.awt.event.ActionEvent evt) {  
199 resetActionPerformed(evt); } });  
200   
201 mute.setBackground(new java.awt.Color(255, 255, 0));  
202 mute.setFont(new java.awt.Font("Tahoma", 1, 14)); // NOI18N  
203 mute.setForeground(new java.awt.Color(0, 153, 0));  
204 mute.setText("Mute/Unmute");  
205 mute.addActionListener(  
206 new java.awt.event.ActionListener() {  
207 public void actionPerformed(java.awt.event.ActionEvent evt) {  
208 muteActionPerformed(evt); } });  
209   
210 sb.setBackground(new java.awt.Color(255, 255, 0));  
211 sb.setMaximum(255);  
212 sb.setValue(sc.getBlue());  
213   
214 sg.setBackground(new java.awt.Color(255, 255, 0));  
215 sg.setMaximum(255);  
216 sg.setValue(sc.getGreen());  
217   
218 menu.setBackground(new java.awt.Color(255, 255, 0));  
219 menu.setFont(new java.awt.Font("Tahoma", 1, 14)); // NOI18N  
220 menu.setForeground(new java.awt.Color(0, 153, 0));  
221 menu.setText("Back to the main menu");  
222 menu.addActionListener(  
223 new java.awt.event.ActionListener() {  
224 public void actionPerformed(java.awt.event.ActionEvent evt) {  
225 menuActionPerformed(evt); } });  
226   
227 snakeLabel.setBackground(new java.awt.Color(0, 204, 0));  
228 snakeLabel.setFont(new java.awt.Font("Tahoma", 1, 14)); // NOI18N  
229 snakeLabel.setHorizontalAlignment(javax.swing.SwingConstants.CENTER);  
230 snakeLabel.setText("Snake Color");  
231 snakeLabel.setBorder(javax.swing.BorderFactory.createLineBorder(new java.awt.Color(0, 0, 0)));  
232 snakeLabel.setOpaque(true);  
233   
234 foodLabel.setBackground(new java.awt.Color(153, 153, 153));  
235 foodLabel.setFont(new java.awt.Font("Tahoma", 1, 14)); // NOI18N  
236 foodLabel.setHorizontalAlignment(javax.swing.SwingConstants.CENTER);  
237 foodLabel.setText("Food Color");  
238 foodLabel.setBorder(javax.swing.BorderFactory.createLineBorder(new java.awt.Color(0, 0, 0)));  
239 foodLabel.setOpaque(true);  
240   
241 fr.setBackground(new java.awt.Color(255, 255, 0));  
242 fr.setValue(fc.getRed());  
243   
244 fg.setBackground(new java.awt.Color(255, 255, 0));  
245 fg.setValue(fc.getGreen());  
246   
247 fb.setBackground(new java.awt.Color(255, 255, 0));  
248 fb.setValue(fc.getBlue());  
249   
250 javax.swing.GroupLayout layout = new javax.swing.GroupLayout(this);  
251 this.setLayout(layout);  
252 layout.setHorizontalGroup(  
253 layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)  
254 .addComponent(menu, javax.swing.GroupLayout.Alignment.TRAILING, javax.swing.GroupLayout.DEFAULT\_SIZE, 561, Short.MAX\_VALUE)  
255 .addGroup(layout.createSequentialGroup()  
256 .addContainerGap()  
257 .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)  
258 .addGroup(layout.createSequentialGroup()  
259 .addComponent(left, javax.swing.GroupLayout.DEFAULT\_SIZE, 264, Short.MAX\_VALUE)  
260 .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)  
261 .addComponent(right, javax.swing.GroupLayout.DEFAULT\_SIZE, 271, Short.MAX\_VALUE))  
262 .addGroup(layout.createSequentialGroup()  
263 .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)  
264 .addComponent(reset, javax.swing.GroupLayout.DEFAULT\_SIZE, 264, Short.MAX\_VALUE)  
265 .addComponent(pause, javax.swing.GroupLayout.DEFAULT\_SIZE, 264, Short.MAX\_VALUE))  
266 .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)  
267 .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)  
268 .addComponent(mute, javax.swing.GroupLayout.DEFAULT\_SIZE, 271, Short.MAX\_VALUE)  
269 .addComponent(escape, javax.swing.GroupLayout.DEFAULT\_SIZE, 271, Short.MAX\_VALUE)))  
270 .addComponent(jLabel1, javax.swing.GroupLayout.Alignment.TRAILING, javax.swing.GroupLayout.PREFERRED\_SIZE, 537, javax.swing.GroupLayout.PREFERRED\_SIZE)  
271 .addGroup(layout.createSequentialGroup()  
272 .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)  
273 .addComponent(sg, javax.swing.GroupLayout.DEFAULT\_SIZE, 263, Short.MAX\_VALUE)  
274 .addComponent(sb, javax.swing.GroupLayout.DEFAULT\_SIZE, 263, Short.MAX\_VALUE)  
275 .addComponent(sr, javax.swing.GroupLayout.DEFAULT\_SIZE, 263, Short.MAX\_VALUE))  
276 .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)  
277 .addComponent(snakeLabel, javax.swing.GroupLayout.PREFERRED\_SIZE, 274, javax.swing.GroupLayout.PREFERRED\_SIZE))  
278 .addGroup(javax.swing.GroupLayout.Alignment.TRAILING, layout.createSequentialGroup()  
279 .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.TRAILING)  
280 .addComponent(bgb, javax.swing.GroupLayout.Alignment.LEADING, javax.swing.GroupLayout.DEFAULT\_SIZE, 257, Short.MAX\_VALUE)  
281 .addComponent(bgr, javax.swing.GroupLayout.Alignment.LEADING, javax.swing.GroupLayout.DEFAULT\_SIZE, 257, Short.MAX\_VALUE)  
282 .addComponent(bgg, javax.swing.GroupLayout.Alignment.LEADING, javax.swing.GroupLayout.DEFAULT\_SIZE, 257, Short.MAX\_VALUE))  
283 .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.UNRELATED)  
284 .addComponent(backgroundLabel, javax.swing.GroupLayout.PREFERRED\_SIZE, 274, javax.swing.GroupLayout.PREFERRED\_SIZE)))  
285 .addContainerGap())  
286 .addGroup(layout.createSequentialGroup()  
287 .addContainerGap()  
288 .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)  
289 .addComponent(fr, javax.swing.GroupLayout.DEFAULT\_SIZE, 263, Short.MAX\_VALUE)  
290 .addComponent(fb, javax.swing.GroupLayout.DEFAULT\_SIZE, 263, Short.MAX\_VALUE)  
291 .addComponent(fg, javax.swing.GroupLayout.DEFAULT\_SIZE, 263, Short.MAX\_VALUE))  
292 .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)  
293 .addComponent(foodLabel, javax.swing.GroupLayout.DEFAULT\_SIZE, 274, Short.MAX\_VALUE)  
294 .addContainerGap())  
295 );  
296 layout.setVerticalGroup(  
297 layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)  
298 .addGroup(javax.swing.GroupLayout.Alignment.TRAILING, layout.createSequentialGroup()  
299 .addContainerGap()  
300 .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)  
301 .addComponent(left, javax.swing.GroupLayout.DEFAULT\_SIZE, 52, Short.MAX\_VALUE)  
302 .addComponent(right, javax.swing.GroupLayout.PREFERRED\_SIZE, 52, javax.swing.GroupLayout.PREFERRED\_SIZE))  
303 .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)  
304 .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING, false)  
305 .addComponent(escape, javax.swing.GroupLayout.DEFAULT\_SIZE, javax.swing.GroupLayout.DEFAULT\_SIZE, Short.MAX\_VALUE)  
306 .addComponent(pause, javax.swing.GroupLayout.PREFERRED\_SIZE, 49, javax.swing.GroupLayout.PREFERRED\_SIZE))  
307 .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)  
308 .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING, false)  
309 .addComponent(mute, javax.swing.GroupLayout.DEFAULT\_SIZE, javax.swing.GroupLayout.DEFAULT\_SIZE, Short.MAX\_VALUE)  
310 .addComponent(reset, javax.swing.GroupLayout.PREFERRED\_SIZE, 51, javax.swing.GroupLayout.PREFERRED\_SIZE))  
311 .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)  
312 .addComponent(jLabel1, javax.swing.GroupLayout.PREFERRED\_SIZE, 27, javax.swing.GroupLayout.PREFERRED\_SIZE)  
313 .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING, false)  
314 .addGroup(layout.createSequentialGroup()  
315 .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)  
316 .addComponent(sr, javax.swing.GroupLayout.PREFERRED\_SIZE, 31, javax.swing.GroupLayout.PREFERRED\_SIZE)  
317 .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)  
318 .addComponent(sg, javax.swing.GroupLayout.PREFERRED\_SIZE, 31, javax.swing.GroupLayout.PREFERRED\_SIZE)  
319 .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)  
320 .addComponent(sb, javax.swing.GroupLayout.PREFERRED\_SIZE, 31, javax.swing.GroupLayout.PREFERRED\_SIZE))  
321 .addGroup(layout.createSequentialGroup()  
322 .addGap(14, 14, 14)  
323 .addComponent(snakeLabel, javax.swing.GroupLayout.DEFAULT\_SIZE, javax.swing.GroupLayout.DEFAULT\_SIZE, Short.MAX\_VALUE)))  
324 .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)  
325 .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING, false)  
326 .addGroup(javax.swing.GroupLayout.Alignment.TRAILING, layout.createSequentialGroup()  
327 .addComponent(foodLabel, javax.swing.GroupLayout.PREFERRED\_SIZE, 110, javax.swing.GroupLayout.PREFERRED\_SIZE)  
328 .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED))  
329 .addGroup(layout.createSequentialGroup()  
330 .addComponent(fr, javax.swing.GroupLayout.PREFERRED\_SIZE, javax.swing.GroupLayout.DEFAULT\_SIZE, javax.swing.GroupLayout.PREFERRED\_SIZE)  
331 .addGap(18, 18, 18)  
332 .addComponent(fg, javax.swing.GroupLayout.PREFERRED\_SIZE, javax.swing.GroupLayout.DEFAULT\_SIZE, javax.swing.GroupLayout.PREFERRED\_SIZE)  
333 .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED, javax.swing.GroupLayout.DEFAULT\_SIZE, Short.MAX\_VALUE)  
334 .addComponent(fb, javax.swing.GroupLayout.PREFERRED\_SIZE, javax.swing.GroupLayout.DEFAULT\_SIZE, javax.swing.GroupLayout.PREFERRED\_SIZE)  
335 .addGap(15, 15, 15)))  
336 .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)  
337 .addGroup(layout.createSequentialGroup()  
338 .addComponent(bgr, javax.swing.GroupLayout.DEFAULT\_SIZE, javax.swing.GroupLayout.DEFAULT\_SIZE, Short.MAX\_VALUE)  
339 .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)  
340 .addComponent(bgg, javax.swing.GroupLayout.DEFAULT\_SIZE, javax.swing.GroupLayout.DEFAULT\_SIZE, Short.MAX\_VALUE)  
341 .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)  
342 .addComponent(bgb, javax.swing.GroupLayout.DEFAULT\_SIZE, javax.swing.GroupLayout.DEFAULT\_SIZE, Short.MAX\_VALUE)  
343 .addGap(32, 32, 32))  
344 .addGroup(layout.createSequentialGroup()  
345 .addComponent(backgroundLabel, javax.swing.GroupLayout.DEFAULT\_SIZE, 102, Short.MAX\_VALUE)  
346 .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.UNRELATED)))  
347 .addComponent(menu, javax.swing.GroupLayout.PREFERRED\_SIZE, 90, javax.swing.GroupLayout.PREFERRED\_SIZE))  
348 ); } }

1 import javax.swing.\*;  
 2 import java.util.Map;  
 3 import java.awt.\*;  
 4 import java.awt.event.\*;  
 5 public class DiffPanel extends JPanel {  
 6 /\*\*Buttons for difficulty choice\*/  
 7 JButton easy, medium, hard, insane;  
 8 /\*\*Description of each difficulty\*/  
 9 JTextArea description;  
 10 /\*\*subPanel with buttons\*/  
 11 JPanel cPanel;  
 12 /\*\*SnakePanel to be loaded for game\*/  
 13 SnakePanel snake;  
 14 /\*\*Map of the possible input keys and their actual input keys\*/  
 15 private Map<Integer, Integer> myKeys;  
 16 /\*\*Snake, Food, and background colors\*/  
 17 private Color sc, fc, bgc;  
 18 /\*\*Creates a difficulty panel where you can choose difficulty of the Snake game  
 19 \*@param keys The user chosen controls  
 20 \*@param snakec Color of snake  
 21 \*@param foodc Color of food  
 22 \*@param backc Color of background\*/  
 23 public DiffPanel(Map<Integer, Integer> keys, Color snakec, Color foodc, Color backc) {  
 24 myKeys = keys;  
 25 sc = snakec;  
 26 fc = foodc;  
 27 bgc = backc;  
 28 setLayout(new BorderLayout());   
 29 description = new JTextArea("");   
 30 add(description, BorderLayout.CENTER);   
 31 cPanel = new JPanel();  
 32 cPanel.setLayout(new GridLayout(4, 1));  
 33 add(cPanel, BorderLayout.WEST);   
 34 easy = new JButton("Easy");  
 35 easy.addActionListener(  
 36 new java.awt.event.ActionListener() {  
 37 public void actionPerformed(java.awt.event.ActionEvent evt) {  
 38 snake = new SnakePanel(40, myKeys, sc, fc, bgc);  
 39 SnakeDriver.reload(snake, "Final Project: Snake!"); } });  
 40 easy.addMouseListener(new EasyListener());  
 41 cPanel.add(easy);   
 42 medium = new JButton("Medium");  
 43 medium.addActionListener(  
 44 new java.awt.event.ActionListener() {  
 45 public void actionPerformed(java.awt.event.ActionEvent evt) {  
 46 snake = new SnakePanel(27, myKeys, sc, fc, bgc);  
 47 SnakeDriver.reload(snake, "Final Project: Snake!"); } });  
 48 medium.addMouseListener(new MediumListener());   
 49 cPanel.add(medium);  
 50   
 51 hard = new JButton("Hard");  
 52 hard.addActionListener(  
 53 new java.awt.event.ActionListener() {  
 54 public void actionPerformed(java.awt.event.ActionEvent evt) {  
 55 snake = new SnakePanel(20, myKeys, sc, fc, bgc);  
 56 SnakeDriver.reload(snake, "Final Project: Snake!"); } });  
 57 hard.addMouseListener(new HardListener());   
 58 cPanel.add(hard);  
 59   
 60 insane = new JButton("Insane");  
 61 insane.addActionListener(  
 62 new java.awt.event.ActionListener() {  
 63 public void actionPerformed(java.awt.event.ActionEvent evt) {  
 64 snake = new SnakePanel(10,myKeys, sc, fc, bgc);  
 65 SnakeDriver.reload(snake, "Final Project: Snake!"); } });  
 66 insane.addMouseListener(new InsaneListener());   
 67 cPanel.add(insane);  
 68 }  
 69 /\*\*Gives description of easy level\*/  
 70 public class EasyListener implements MouseListener {  
 71 public void mouseEntered(MouseEvent evt) {  
 72 String s = "Well, everyone needs to start somewhere.";  
 73 s+="\nPlaying this level is kind of a waste of all the time I spent making this game,";  
 74 s+="\nbut whatever, you're the player.";  
 75 s+="\nWuss.";  
 76 description.setText(s); }  
 77 public void mouseExited(MouseEvent evt) {  
 78 description.setText(""); }  
 79 public void mouseClicked(MouseEvent evt) {}  
 80 public void mousePressed(MouseEvent evt){}  
 81 public void mouseReleased(MouseEvent evt) {} }  
 82 /\*\*Gives description of medium level\*/  
 83 public class MediumListener implements MouseListener {  
 84 public void mouseEntered(MouseEvent evt) {  
 85 String s = "Good job.";  
 86 s+="\nYou are completely standard.";  
 87 s+="\nIn no way special, at all.";  
 88 s+="\nI totally expected you to pick this difficulty.";  
 89 s+="\nJust like everyone else.";  
 90 s+="\nI just, I dunno, thought you'd be more of a rebel.";  
 91 s+="\nBut it's okay, it's fine.";  
 92 s+="\nPlaying medium is completely fine.";  
 93 s+="\nIf you're happy with where you are in life.";  
 94 description.setText(s); }  
 95 public void mouseExited(MouseEvent evt) {  
 96 description.setText(""); }  
 97 public void mouseClicked(MouseEvent evt) {}  
 98 public void mousePressed(MouseEvent evt) {}  
 99 public void mouseReleased(MouseEvent evt) {} }  
100 /\*\*Gives description of hard level\*/  
101 public class HardListener implements MouseListener {  
102 public void mouseEntered(MouseEvent evt) {  
103 String s = "Now that's more like it.";  
104 s+="\nGoing for the gold,";  
105 s+="\nshooting for the stars.";  
106 s+="\nYou're a real champ.";  
107 s+="\nCruising through life, you like to take the fast lane.";  
108 s+="\nYou stop for nothing.";  
109 s+="\nThis is truly the difficulty kings play Snake on.";  
110 s+="\nBut what about gods?";  
111 s+="\nThere's still one more difficulty setting...";  
112 description.setText(s); }  
113 public void mouseExited(MouseEvent evt) {  
114 description.setText(""); }  
115 public void mouseClicked(MouseEvent evt) {}  
116 public void mousePressed(MouseEvent evt) {}  
117 public void mouseReleased(MouseEvent evt) {} }  
118 /\*\*Gives description of insane level\*/  
119 public class InsaneListener implements MouseListener {  
120 public void mouseEntered(MouseEvent evt) {  
121 String s = "I worship thee, O great one.";  
122 s+="\nThis isn't Sparta.";  
123 s+="\nThis is INSANITY.";  
124 s+="\nOf the legends, you rise above them all.";  
125 s+="\nBecause you are EPIC.";  
126 s+="\nYou're Evel Knievel, the Flash, Rambo, and a Ninja all rolled into one.";  
127 s+="\nBut at heart, to actually attempt this";  
128 s+="\nYou must be CRAZY.";  
129 s+="\nYou're more jittey than the Joker,";  
130 s+="\nmore loony than the Mad Hatter.";  
131 s+="\nI really wouldn't want to be trapped in a dark room with you.";  
132 s+="\nAnyways, good luck on this.";  
133 s+="\nYou'll need it.";  
134 description.setText(s); }  
135 public void mouseExited(MouseEvent evt) {  
136 description.setText(""); }  
137 public void mouseClicked(MouseEvent evt) {}  
138 public void mousePressed(MouseEvent evt) {}  
139 public void mouseReleased(MouseEvent evt) {} } }

1 import java.util.ArrayList;   
 2 import java.util.Map;  
 3 import java.util.Scanner;  
 4 import java.io.\*;  
 5 import java.awt.\*;  
 6 import java.awt.event.\*;  
 7 import java.awt.image.\*;  
 8 import java.awt.geom.\*;  
 9 import java.awt.RenderingHints.\*;  
 10 import javax.swing.\*;  
 11 import java.lang.\*;   
 12   
 13 /\*\* The JPanel that runs the main game\*/  
 14 public class SnakePanel extends JPanel {  
 15 /\*\*Size of frame\*/  
 16 private final int FRAME = 400;  
 17 /\*\*Color of background\*/  
 18 private Color BACKGROUND;  
 19 /\*\*Image that graphics are painted on to\*/  
 20 private BufferedImage myImage;  
 21 /\*\*Graphics class for graphics\*/  
 22 private Graphics2D myBuffer;  
 23 /\*\*Main game snake\*/  
 24 private Snake snake;  
 25 /\*\*Main game food; can be eaten and regenerated\*/  
 26 private Food mainFood;  
 27 /\*\*Keeps the Snake from turning too fast\*/  
 28 private long lastPressProcessed = 0;  
 29 /\*\*Is the game over?\*/  
 30 private boolean isGameOver = false;  
 31 /\*\*Size of in-game wall\*/  
 32 private int wall;  
 33 /\*\*Current score\*/  
 34 private double score = 0;  
 35 /\*\*Current level\*/  
 36 private int level = 1;  
 37 /\*\*Difficulty\*/  
 38 private String diff;  
 39 /\*\*List of highscore names\*/  
 40 private ArrayList<String> hsnames;  
 41 /\*\*List of highscore scores\*/  
 42 private ArrayList<Integer> hsscores;  
 43 /\*\*On a screen between levels?\*/  
 44 private boolean betweenLevels = false;  
 45 /\*\*Used as a stopwatch\*/  
 46 private long anyTime = 0;  
 47 /\*\*Start time\*/  
 48 private long startTime;  
 49 /\*\*Timers for animation and other time-related processes\*/  
 50 private Timer action, snakespeed, leftTimer, rightTimer;  
 51 /\*\*Is the game muted?\*/  
 52 private boolean isMuted = false;  
 53 /\*\*Music that plays during game\*/  
 54 public Sound scottwinter;  
 55 /\*\*Sound that plays on level up\*/  
 56 private Sound stageclear;  
 57 /\*\*Sound that plays when food is eaten\*/  
 58 private Sound chomp;  
 59 /\*\*Map of the possible input keys and their actual input keys\*/  
 60 private Map<Integer, Integer> myKeys;  
 61 /\*\*Constructor of the game panel\*/  
 62 public SnakePanel(int speed, Map<Integer, Integer> keys, Color sc, Color fc, Color bgc) {  
 63 myKeys = keys;   
 64 BACKGROUND = bgc;   
 65 myImage = new BufferedImage(FRAME, FRAME, BufferedImage.TYPE\_INT\_RGB);  
 66 myBuffer = myImage.createGraphics();  
 67 myBuffer.setColor(BACKGROUND);  
 68 myBuffer.fillRect(0, 0, FRAME, FRAME);   
 69 wall = FRAME;   
 70 snake = new Snake(sc);  
 71 for(int k = 0;k < 10;k++)  
 72 snake.addSegment(new Segment(200, 200 + 4 \* k, Color.green));   
 73 mainFood = new Food(200, 100, 12, fc);   
 74 action = new Timer(1, new Listener());  
 75 action.start();   
 76 snakespeed = new Timer(speed, new SpeedListener());  
 77 snakespeed.start();  
 78 leftTimer = new Timer(speed + 30, new LeftListener());  
 79 leftTimer.setInitialDelay(speed + 50);  
 80 rightTimer = new Timer(speed + 30, new RightListener());  
 81 rightTimer.setInitialDelay(speed + 50);   
 82 startTime = System.currentTimeMillis();   
 83 if(speed==40)  
 84 diff = "easy";  
 85 else if(speed==27)  
 86 diff = "medium";  
 87 else if(speed==20)  
 88 diff = "hard";  
 89 else  
 90 diff = "insane";  
 91 Scanner scoreScan = new Scanner(System.in);  
 92 hsnames = new ArrayList<String>();  
 93 hsscores = new ArrayList<Integer>();  
 94 try{  
 95 scoreScan = new Scanner(new File("HS"+diff+".txt"));}  
 96 catch(Exception e)  
 97 {}  
 98 while(scoreScan.hasNext()) {  
 99 hsnames.add(scoreScan.nextLine());  
100 hsscores.add(Integer.parseInt(scoreScan.nextLine())); }   
101 scottwinter = new Sound("Scott Pilgrim\_Another Winter.wav");  
102 stageclear = new Sound("Scott Pilgrim\_Stage Clear.wav");  
103 chomp = new Sound("chomp.wav");  
104 scottwinter.loop();   
105 addKeyListener(new Key());  
106 setFocusable(true); }  
107 /\*\*Paint the screen\*/  
108 public void paintComponent(Graphics g) {  
109 g.drawImage(myImage, 0, 0, getWidth(), getHeight(), null); }  
110 /\*\*Codes for keyboard input\*/  
111 public class Key extends KeyAdapter {  
112 public void keyPressed(KeyEvent e) {   
113 if(e.getKeyCode() == myKeys.get(KeyEvent.VK\_RIGHT) && action.isRunning() && !rightTimer.isRunning())  
114 snake.turnRight();  
115 if(e.getKeyCode() == myKeys.get(KeyEvent.VK\_LEFT) && action.isRunning() && !leftTimer.isRunning())  
116 snake.turnLeft();   
117 if(e.getKeyCode() == myKeys.get(KeyEvent.VK\_RIGHT) && action.isRunning())  
118 rightTimer.start();  
119 if(e.getKeyCode() == myKeys.get(KeyEvent.VK\_LEFT) && action.isRunning())  
120 leftTimer.start();   
121 if(e.getKeyCode() == myKeys.get(KeyEvent.VK\_ESCAPE)) {   
122 if(action.isRunning()) {  
123 action.stop();  
124 snakespeed.stop();  
125 if(JOptionPane.showConfirmDialog(null,"Are you sure you want to return to the menu?", "GAME PAUSED", JOptionPane.YES\_NO\_OPTION) == JOptionPane.OK\_OPTION)  
126 {  
127 scottwinter.exit();  
128 stageclear.exit();  
129 SnakeDriver.reload(new MenuPanel(false), "Finale Project: Snake!"); }  
130 else if(!isGameOver) {  
131 action.start();  
132 snakespeed.start(); } }  
133 else if(JOptionPane.showConfirmDialog(null,"Are you sure you want to return to the menu?", "GAME PAUSED", JOptionPane.YES\_NO\_OPTION) == JOptionPane.OK\_OPTION)  
134 {  
135 scottwinter.exit();   
136 stageclear.exit();   
137 SnakeDriver.reload(new MenuPanel(false), "Finale Project: Snake!"); } }  
138 if(e.getKeyCode() == myKeys.get(KeyEvent.VK\_P)) {  
139 myBuffer.setColor(Color.red);  
140 myBuffer.drawString("PAUSED", FRAME / 2 - 20, FRAME / 2);  
141 repaint();  
142 if(action.isRunning()) {  
143 action.stop();  
144 snakespeed.stop(); }  
145 else if(!isGameOver) {  
146 action.start();   
147 snakespeed.start(); } }  
148 if(e.getKeyCode() == myKeys.get(KeyEvent.VK\_R)) {  
149 if(action.isRunning()) {  
150 action.stop();  
151 snakespeed.stop();  
152 if(JOptionPane.showConfirmDialog(null,"Are you sure you want to reset?", "GAME PAUSED", JOptionPane.YES\_NO\_OPTION) == JOptionPane.OK\_OPTION)  
153 {  
154 scottwinter.exit();  
155 stageclear.exit();  
156 SnakeDriver.reload(new SnakePanel(snakespeed.getDelay(), myKeys, snake.getColor(), mainFood.getColor(), BACKGROUND), "Finale Project: Snake!");  
157 }  
158 else if(!isGameOver) {  
159 action.start();  
160 snakespeed.start(); } }  
161 else if(JOptionPane.showConfirmDialog(null,"Are you sure you want to reset?", "GAME PAUSED", JOptionPane.YES\_NO\_OPTION) == JOptionPane.OK\_OPTION)  
162 {   
163 scottwinter.exit();  
164 stageclear.exit();  
165 SnakeDriver.reload(new SnakePanel(snakespeed.getDelay(), myKeys, snake.getColor(), mainFood.getColor(), BACKGROUND), "Finale Project: Snake!");  
166 }  
167 }  
168 if(e.getKeyCode() == myKeys.get(KeyEvent.VK\_SPACE)) {  
169 isMuted = !isMuted;  
170 if(isMuted)  
171 scottwinter.stop();  
172 else  
173 scottwinter.loop(); } }  
174 public void keyReleased(KeyEvent e) {  
175 if(e.getKeyCode() == myKeys.get(KeyEvent.VK\_RIGHT) && action.isRunning())  
176 rightTimer.stop();  
177 if(e.getKeyCode() == myKeys.get(KeyEvent.VK\_LEFT) && action.isRunning())  
178 leftTimer.stop(); } }  
179 /\*\*Events that occur every second\*/  
180 public class Listener implements ActionListener {  
181 public void actionPerformed(ActionEvent e) {  
182 if(betweenLevels) {  
183 if(level > 4) {  
184 isGameOver = true;  
185 myBuffer.setColor(Color.black);  
186 myBuffer.fillRect(0, 0, FRAME, FRAME);  
187 myBuffer.setColor(Color.red);  
188 myBuffer.drawString("YOU WIN! Final Score: " + (int)score, FRAME / 2 - 40, FRAME / 2);  
189 snakespeed.stop();  
190 repaint();  
191 stageclear.play(); }  
192 else{  
193 myBuffer.setColor(Color.black);  
194 myBuffer.fillRect(0, 0, FRAME, FRAME);  
195 myBuffer.setColor(Color.red);  
196 myBuffer.drawString("Level: " + level, FRAME / 2 - 80, FRAME / 2);  
197 if(System.currentTimeMillis() - anyTime > 5000) {  
198 betweenLevels = false;  
199 snakespeed.start();  
200 scottwinter.loop(); } } }   
201 if(!isGameOver && !betweenLevels) {  
202 myBuffer.setColor(BACKGROUND);  
203 myBuffer.fillRect(0, 0, FRAME, FRAME);   
204 myBuffer.setColor(Color.black);  
205 myBuffer.setStroke(new BasicStroke(7));  
206 myBuffer.drawRect((FRAME - wall) / 2, (FRAME - wall) / 2, wall, wall);  
207 myBuffer.setStroke(new BasicStroke(1));   
208 drawFood();   
209 drawSnake();   
210 if(mainFood.getCircle().intersects(snake.getHead().getX(), snake.getHead().getY(), 4, 4))  
211 eatFood(mainFood);  
212 if(score > level \* 3000 - level \* 250) {  
213 myBuffer.setStroke(new BasicStroke(7));  
214 myBuffer.setColor(BACKGROUND);  
215 myBuffer.drawRect(FRAME / 2 - 10, (FRAME - wall) / 2, 20, 0);  
216 myBuffer.setStroke(new BasicStroke(1)); }   
217 if(score > level \* 3000 - level \* 250 && snake.getHead().getX() > FRAME / 2 - 10 && snake.getHead().getX() < FRAME / 2 + 10 && snake.getHead().getY() < (FRAME - wall) / 2 + 5)  
218 {  
219 anyTime = System.currentTimeMillis();  
220 increaseLevel();  
221 betweenLevels = true;  
222 snakespeed.stop();  
223 mainFood = new Food((int)(Math.random() \* (wall - 23) + ((FRAME - wall) / 2 + 5)), (int)(Math.random() \* (wall - 23) + ((FRAME - wall) / 2 + 5)), 12, mainFood.getColor());   
224 }   
225 isGameOver = isDead(); }   
226 else if(!betweenLevels)  
227 gameOver();  
228 myBuffer.setColor(Color.red);  
229 myBuffer.drawString("Score: "+(int)score, FRAME - 80, FRAME - 6);  
230 myBuffer.drawString("Level: "+level, 20, FRAME - 6);  
231 repaint(); } }  
232 /\*\*Moves snake and increments score; change in speed of timer changes speed of snake\*/  
233 public class SpeedListener implements ActionListener {  
234 public void actionPerformed(ActionEvent e) {  
235 snake.move();  
236 score+=0.2; } }  
237 /\*\*Turns snake left constantly after start by keyboard until stop by keyboard\*/  
238 public class LeftListener implements ActionListener {  
239 public void actionPerformed(ActionEvent e) {  
240 snake.turnLeft(); } }   
241 /\*\*Turns snake right constantly after start by keyboard until stop by keyboard\*/  
242 public class RightListener implements ActionListener {  
243 public void actionPerformed(ActionEvent e) {  
244 snake.turnRight(); } }   
245 /\*\*Draw the snake on the screen\*/   
246 public void drawSnake() {  
247 myBuffer.setColor(snake.getColor());   
248 myBuffer.setRenderingHint(RenderingHints.KEY\_RENDERING,RenderingHints.VALUE\_RENDER\_QUALITY);  
249 myBuffer.setRenderingHint(RenderingHints.KEY\_ANTIALIASING,RenderingHints.VALUE\_ANTIALIAS\_ON);   
250 for(Segment seg : snake.getSegments()) {  
251 myBuffer.fillRect(seg.getX(), seg.getY(), 4, 4);  
252 myBuffer.setColor(Color.black);  
253 myBuffer.drawRect(seg.getX(), seg.getY(), 4, 4);  
254 myBuffer.setColor(snake.getColor()); }  
255 myBuffer.setColor(Color.red);  
256 myBuffer.setStroke(new BasicStroke(1));   
257 myBuffer.drawLine((int)(snake.getSegment(0).getRect().getCenterX()), (int)(snake.getSegment(0).getRect().getCenterY()), (int)(10 \* Math.cos(snake.getDir()) + snake.getHead().getRect().getCenterX()), (int)(10 \* Math.sin(snake.getDir()) + snake.getHead().getRect().getCenterY()));  
258 }  
259 /\*\*Draw the food on the screen\*/  
260 public void drawFood() {  
261 myBuffer.setColor(mainFood.getColor());  
262 if(mainFood.getX()>=0)  
263 myBuffer.fillOval(mainFood.getX(), mainFood.getY(), mainFood.getWorth(), mainFood.getWorth()); }  
264 /\*\*Eat the food, update score, and generate new food  
265 \*@param f The Food to be eaten \*/  
266 public void eatFood(Food f) {  
267 score+=mainFood.getWorth() \* 10;  
268 Segment tail = snake.getSegments().get(snake.getLength() - 1);  
269 for(int k = 0;k < f.getWorth();k++)  
270 snake.addSegment(new Segment(tail.getX(), tail.getY(), snake.getColor()));  
271 if(score < level \* 3000 - level \* 250)  
272 mainFood = new Food((int)(Math.random() \* (wall - 23) + ((FRAME - wall) / 2 + 5)), (int)(Math.random() \* (wall - 23) + ((FRAME - wall) / 2 + 5)), 12, mainFood.getColor());  
273 else  
274 mainFood = new Food(-1, -1, 0, Color.black); chomp.play(); }  
275 /\*\*Carries out game over situation\*/  
276 public void gameOver() {  
277 myBuffer.setColor(Color.black);  
278 myBuffer.fillRect(0, 0, FRAME, FRAME);  
279 myBuffer.setColor(Color.red);  
280 myBuffer.drawString("GAME OVER", FRAME / 2 - 40, FRAME / 2);   
281 myBuffer.drawString("High Scores:", FRAME / 2 - 40, FRAME / 2 + 20);  
282 repaint();   
283 String name = JOptionPane.showInputDialog(null, "Name?", "Game over...", JOptionPane.QUESTION\_MESSAGE);   
284 int pos = 0;  
285 for(int k = 0;k < hsscores.size();k++)  
286 if(hsscores.get(k) > score)  
287 pos++;  
288 hsscores.add(pos, (int)score);  
289 hsnames.add(pos, name);   
290 for(int k = 0;k < 6;k++)  
291 myBuffer.drawString(hsnames.get(k), FRAME / 2 - 120, FRAME / 2 + 40 + k \* 20);  
292 for(int k = 0;k < 6;k++)  
293 myBuffer.drawString("" + hsscores.get(k), FRAME / 2 + 65, FRAME / 2 + 40 + k \* 20);   
294 saveScores(); repaint(); action.stop(); snakespeed.stop(); }  
295 /\*\*Tests if the snake is dead and the game is over  
296 \*@return Is the snake dead? \*/  
297 public boolean isDead() {  
298 if(snake.getHead().getX() < (FRAME - wall) / 2 + 4 || snake.getHead().getX() > FRAME - ((FRAME - wall) / 2) - 8 || snake.getHead().getY() < (FRAME - wall) / 2 + 4 || snake.getHead().getY() > FRAME - (FRAME - wall) / 2 - 8)   
299 return true;//4 is half-thickness of wall  
300 for(int k = 2;k < snake.getLength(); k++)  
301 if(snake.getSegment(0).getRect().intersects(snake.getSegment(k).getRect()))  
302 return true;  
303 return false; }  
304 /\*\*Increases the level\*/  
305 public void increaseLevel() {  
306 level++;  
307 score+=Math.max(0, level \* 200000 - (anyTime - startTime));  
308 wall-=60;  
309 while(snake.getLength() > 0)  
310 snake.removeSegment();  
311 for(int k = 0;k < 10;k++)  
312 snake.addSegment(new Segment(200, wall - 50 + 4 \* k, Color.black));  
313 snake.setDir(Math.PI \* 3.0 / 2.0);  
314 scottwinter.stop();  
315 stageclear.play(); }  
316 /\*\*Save high scores to file\*/  
317 public void saveScores() {  
318 PrintStream outfile = null;  
319 try{  
320 outfile = new PrintStream(new FileOutputStream("HS" + diff + ".txt")); }  
321 catch(FileNotFoundException e) {  
322 JOptionPane.showMessageDialog(null,"The file could not be created."); }  
323 System.setOut(outfile);  
324 for(int k = 0;k < hsnames.size(); k++) {  
325 System.out.println(hsnames.get(k));  
326 System.out.println(hsscores.get(k)); } outfile.close(); } }

1 import java.io.File;  
 2 import java.io.IOException;  
 3 import javax.sound.sampled.\*;  
 4 import javax.sound.sampled.AudioFormat;  
 5 import javax.sound.sampled.AudioInputStream;  
 6 import javax.sound.sampled.AudioSystem;  
 7 import javax.sound.sampled.DataLine;  
 8 import javax.sound.sampled.LineUnavailableException;  
 9 import javax.sound.sampled.SourceDataLine;  
10 /\*\*A Sound class that plays music and other sounds\*/ public class Sound  
11 {  
12 /\*\*Buffer rate of the sound\*/  
13 private static final int EXTERNAL\_BUFFER\_SIZE = 128000;  
14 /\*\*The Clip that plays the sound\*/  
15 Clip line = null;  
16 /\*\*Has the sound been played at least once?\*/  
17 boolean oncePlayed = false;  
18 AudioInputStream ais = null;  
19 /\*\*The constructor of SoundTest. The filename is used to be read in and the audio is later gotten from it and used.  
20 \*@param filename The filename of the file containing the sound\*/  
21 public Sound(String filename) {  
22 try {  
23 File soundname = new File(filename);  
24 ais = AudioSystem.getAudioInputStream(soundname);  
25 line = AudioSystem.getClip();  
26 line.open(ais); }   
27 catch (Exception ex) {  
28 line = null; }  
29   
30 }  
31 /\*\*Plays the music.\*/  
32 public void play() {  
33 FloatControl gainControl = (FloatControl)line.getControl(FloatControl.Type.MASTER\_GAIN);  
34 gainControl.setValue(6);  
35 if (line != null) {   
36 if (oncePlayed == false) {  
37 line.loop(0);  
38 oncePlayed = true; }  
39 else  
40 line.loop(1); } }  
41 /\*\*Loops the music\*/  
42 public void loop() {  
43 line.loop(Clip.LOOP\_CONTINUOUSLY); }  
44 /\*\*Pauses the music\*/  
45 public void stop() {  
46 line.stop(); }  
47 /\*\*Close and exit the file\*/  
48 public void exit() {  
49 line.stop();  
50 line.flush();  
51 line.close();  
52 line = null;  
53 ais = null; } }

1 import java.awt.\*;  
 2 import java.lang.\*;  
 3 import java.util.\*;  
 4 /\*\*A Snake class. SEE segments FIELD FOR DATA STRUCTURE\*/  
 5 public class Snake {  
 6 /\*\*The segments that make up the snake.  
 7 \* Segments can be added, removed, and gotten from index using this data structure \*/  
 8 public ArrayList<Segment> segments;  
 9 /\*\*Speed factor of snake\*/  
10 public double mySpeed;  
11 /\*\*Direction of snake\*/  
12 public double myDir;  
13 /\*\*Color of snake\*/  
14 public Color myColor;  
15 /\*\*Length of snake i.e. number of segments\*/  
16 public int myLength;  
17 /\*\*Creates a snake of color c  
18 \* @param c The color of the snake\*/  
19 public Snake(Color c) {  
20 segments = new ArrayList<Segment>();  
21 mySpeed = 4;  
22 myDir = Math.PI \* 3.0 / 2.0;  
23 myColor = c;//Color taken from slider bar  
24 myLength = 0; }  
25 /\*\*Moves the snake in its current direction\*/  
26 public void move() {  
27 segments.remove(myLength - 1);  
28 int dx = (int)(mySpeed \* Math.cos(myDir));  
29 int dy = (int)(mySpeed \* Math.sin(myDir));  
30 Segment newHead = new Segment(segments.get(0).getX() + dx, segments.get(0).getY() + dy, myColor);  
31 segments.add(0, newHead); }   
32 /\*\*Add a segment to the snake and increment length  
33 \*@param seg The segment to be added\*/  
34 public void addSegment(Segment seg) {  
35 segments.add(seg);  
36 myLength++; }  
37 /\*\*Remove a segment and decrement length  
38 \*@return The removed segment\*/  
39 public Segment removeSegment() {  
40 myLength--;  
41 return segments.remove(myLength); }  
42 /\*\*Returns the direciton of the snake  
43 \*@return The current direction\*/  
44 public double getDir() {  
45 return myDir; }  
46 /\*\*Returns the length of the snake  
47 \*@return The current length\*/  
48 public int getLength() {  
49 return myLength; }  
50 /\*\*Returns the color of the snake  
51 \*@return The current color\*/  
52 public Color getColor() {  
53 return myColor; }  
54 /\*\*Sets the snake's color  
55 \*@param c The new color of the snake\*/  
56 public void setColor(Color c) {  
57 myColor = c;  
58 for(Segment seg : segments)  
59 seg.setColor(c); }  
60 /\*\*Returns the Segments of the snake  
61 \*@return The segments field making up the snake\*/  
62 public ArrayList<Segment> getSegments() {  
63 return segments; }  
64 /\*\*Returns the Segment of number k in the segments field  
65 \*@param k number of the segment  
66 \*@return A segment of number k\*/  
67 public Segment getSegment(int k) {  
68 return segments.get(k); }  
69 /\*\*Returns first Segment in the snake; identical to calling getSegment(0)  
70 \*@return The first Segment in the segments field  
71 \*@see Segment getSegments(int k)\*/  
72 public Segment getHead() {  
73 return segments.get(0); }   
74 /\*\*Turns snake right\*/   
75 public void turnRight() {  
76 myDir+=(Math.PI / 8.25); }  
77 /\*\*Turns snake left\*/  
78 public void turnLeft() {  
79 myDir-=(Math.PI / 8.25); }  
80 /\*\*Changes the snake's direction  
81 \*@param dir The new direction of the snake\*/  
82 public void setDir(double dir) {  
83 myDir = dir; }  
84 /\*\*Returns snake's distance from a point  
85 \*@param x x-coordinate of point  
86 \*@param y y-coordinate of point  
87 \*@return distance of snake from point\*/  
88 public double getDistance(int x, int y) {  
89 int xDist = segments.get(0).getX() - x;  
90 int yDist = segments.get(0).getY() - y;  
91 return Math.sqrt(Math.pow(xDist, 2) + Math.pow(yDist, 2)); } }

1 import java.awt.\*;  
 2 /\*\*The segment class that makes up the Snake. \*/  
 3 public class Segment {  
 4 /\*\*Color of the segment\*/  
 5 public Color myColor;  
 6 /\*\*Coordinates of the segment\*/  
 7 public int myX, myY;  
 8 /\*\*Size of the segment\*/  
 9 public int mySize;  
10 /\*\*Square with upper left corner of the segment's coordinates\*/  
11 public Rectangle myRect;  
12 /\*\* Segment constructor  
13 \* @param x The x-coordinate, in pixels, of myLoc  
14 \* @param y The y-coordinate, in pixels, of myLoc  
15 \* @param color The color of this segment. \*/   
16 public Segment(int x, int y, Color color) {  
17 myX = x;  
18 myY = y;  
19 myColor = color;  
20 myRect = new Rectangle(x, y, 4, 4); }   
21 /\*\*Get the color of the segment  
22 \*@return the current color of the segment\*/  
23 public Color getColor() {  
24 return myColor; }  
25 /\*\*Get the x-coordinate of the segment  
26 \*@return the x-coordinate of the segment\*/  
27 public int getX() {  
28 return myX; }  
29 /\*\*Get the y-coordinate of the segment  
30 \*@return the y-coordinate of the segment\*/  
31 public int getY() {  
32 return myY; }  
33 /\*\*Get the size of the segment  
34 \*@return the current size of the segment\*/  
35 public int getSize() {  
36 return mySize; }  
37 /\*\*Returns the square representing the segment  
38 \*@return the square with upper left corner of the segment's coordinates\*/  
39 public Rectangle getRect() {  
40 return myRect; }  
41 /\*\*Set the color of the segment  
42 \*@param c the new color of the segment\*/  
43 public void setColor(Color c) {  
44 myColor = c; }  
45 /\*\*Set the x-coordinate of the segment  
46 \*@param x the new x-coordinate of the segment\*/  
47 public void setX(int x) {  
48 myX = x;  
49 myRect.setBounds(x, myY, 4, 4); }  
50 /\*\*Set the y-coordinate of the segment  
51 \*@param y the new y-coordinate of the segment\*/  
52 public void setY(int y) {  
53 myY = y;  
54 myRect.setBounds(myX, y, 4, 4); }  
55 /\*\*Set the size of the segment  
56 \*@param size the new size of the segment\*/  
57 public void setSize(int size) {  
58 mySize = size; } }

1 import java.awt.\*;  
 2 import java.awt.geom.\*;  
 3 /\*\*The Food class; eaten by the Snake\*/  
 4 public class Food {  
 5 /\*\*Coordinates of food\*/  
 6 public int myX, myY;  
 7 /\*\*Worth of food\*/  
 8 public int myWorth;  
 9 /\*\*Circle bounded by square with upper left corner myX, myY\*/  
10 public Ellipse2D.Double myCirc;   
11 /\*\*Color of food\*/  
12 public Color myColor;  
13 /\*\*Constructor for the food  
14 \*@param x x-coordinate of food  
15 \*@param y y-coordinate of food  
16 \*@param worth worth of food\*/  
17 public Food(int x, int y, int worth, Color c) {  
18 myX = x; myY = y; myWorth = worth;  
19 myCirc = new Ellipse2D.Double(x, y, worth, worth); myColor = c; }  
20 /\*\*Returns x-coordinate of food  
21 \*@return the x coordinate of the food\*/  
22 public int getX() {  
23 return myX; }  
24 /\*\*Returns y-coordinate of food  
25 \*@return the y coordinate of the food\*/  
26 public int getY() {  
27 return myY; }  
28 /\*\*Returns the circle defining the food  
29 \*@return the circle bounded by the square with the upper left corner of the food's coordinates\*/  
30 public Ellipse2D getCircle() {  
31 return myCirc; }  
32 /\*\*Returns the worth of the food  
33 \*@return the worth of the food\*/  
34 public int getWorth() {  
35 return myWorth; }  
36 /\*\*Sets color of food  
37 \*@param c Color of food\*/  
38 public void setColor(Color c) {  
39 myColor = c; }  
40 /\*\*Returns color of food  
41 \*@return Color of food\*/  
42 public Color getColor() {  
43 return myColor; } }

API

(Note: to save paper, all inherited fields, inherited methods, and nested classes were not included)

**Class SnakeDriver** java.lang.Object **SnakeDriver**

public class **SnakeDriver**extends java.lang.Object

The Driver of Snake, where it all begins. IMPORTANT--> See Snake class and OptionPanel class for data structure All of this was done by Raynor Kuang Version: 6/1/2011

|  |  |
| --- | --- |
| **Field Summary** | |
| private static javax.swing.JFrame | [**frame**](SnakeDriver.html#frame)          Frame of the game. |

|  |  |
| --- | --- |
| **Constructor Summary** | |
| [**SnakeDriver**](SnakeDriver.html#SnakeDriver())() |  |

|  |  |
| --- | --- |
| **Method Summary** | |
| private static void | [**initialize**](SnakeDriver.html#initialize())()          Actually initializes the program. |
| static void | [**main**](SnakeDriver.html#main(java.lang.String[]))(java.lang.String[] args)          Runs the game. |
| static void | [**reload**](SnakeDriver.html#reload(javax.swing.JPanel, java.lang.String))(javax.swing.JPanel j, java.lang.String s)          Reloads the frame with a new panel. |

|  |
| --- |
| **Field Detail** |

**frame**

private static javax.swing.JFrame **frame**

Frame of the game.

|  |
| --- |
| **Constructor Detail** |

**SnakeDriver**

public **SnakeDriver**()

|  |
| --- |
| **Method Detail** |

**main**

public static void **main**(java.lang.String[] args)

Runs the game.

**initialize**

private static void **initialize**()

Actually initializes the program.

**reload**

public static void **reload**(javax.swing.JPanel j,

java.lang.String s)

Reloads the frame with a new panel.

**Parameters:**

j - JPanel run in the game frame

s - Title of the panel

**Class MenuPanel**

java.lang.Object

java.awt.Component

java.awt.Container

javax.swing.JComponent

javax.swing.JPanel

**MenuPanel**

**All Implemented Interfaces:**

java.awt.image.ImageObserver, java.awt.MenuContainer, java.io.Serializable, javax.accessibility.Accessible

public class **MenuPanel**extends javax.swing.JPanel

MenuPanel that opens when the game runs.

**See Also:**

[Serialized Form](serialized-form.html#MenuPanel)

|  |  |
| --- | --- |
| **Field Summary** | |
| static java.awt.Color | [**bgc**](MenuPanel.html#bgc)          Background color |
| static java.awt.Color | [**fc**](MenuPanel.html#fc)          Food color |
| static java.util.Map<java.lang.Integer,java.lang.Integer> | [**myKeys**](MenuPanel.html#myKeys)          Map of the possible input keys and their actual input keys |
| private  OptionPanel | [**optPan**](MenuPanel.html#optPan)          OptionPanel that opens options screen |
| static java.awt.Color | [**sc**](MenuPanel.html#sc)          Snake color |
| private  SnakePanel | [**snakePan**](MenuPanel.html#snakePan)          SnakePanel that runs game screen |

|  |  |
| --- | --- |
| **Constructor Summary** | |
| [**MenuPanel**](MenuPanel.html#MenuPanel(boolean))(boolean firstOpen)           Creates new MainPanel with buttons |  |

|  |  |
| --- | --- |
| **Method Summary** | |
| private  void | [**credButtonActionPerformed**](MenuPanel.html#credButtonActionPerformed(java.awt.event.ActionEvent))(java.awt.event.ActionEvent evt) |
| private  void | [**optButtonActionPerformed**](MenuPanel.html#optButtonActionPerformed(java.awt.event.ActionEvent))(java.awt.event.ActionEvent evt)           Actions of option button |
| void | [**paint**](MenuPanel.html#paint(java.awt.Graphics))(java.awt.Graphics g) |
| static void | [**setColors**](MenuPanel.html#setColors(java.awt.Color, java.awt.Color, java.awt.Color))(java.awt.Color snakec, java.awt.Color foodc, java.awt.Color backc) |
| static void | [**setKeys**](MenuPanel.html#setKeys(java.util.Map))(java.util.Map<java.lang.Integer,java.lang.Integer> keys) |
| private  void | [**snakeButtonActionPerformed**](MenuPanel.html#snakeButtonActionPerformed(java.awt.event.ActionEvent))(java.awt.event.ActionEvent evt)           Actions of game button |
| private  void | [**tutButtonActionPerformed**](MenuPanel.html#tutButtonActionPerformed(java.awt.event.ActionEvent))(java.awt.event.ActionEvent evt)           Displays tutorial |

|  |
| --- |
| **Field Detail** |

**optPan**

private OptionPanel **optPan**

OptionPanel that opens options screen

**snakePan**

private SnakePanel **snakePan**

SnakePanel that runs game screen

**myKeys**

public static java.util.Map<java.lang.Integer,java.lang.Integer> **myKeys**

Map of the possible input keys and their actual input keys

**sc**

public static java.awt.Color **sc**

Snake color

**fc**

public static java.awt.Color **fc**

Food color

**bgc**

public static java.awt.Color **bgc**

Background color

|  |
| --- |
| **Constructor Detail** |

**MenuPanel**

public **MenuPanel**(boolean firstOpen)

Creates new MainPanel with buttons

|  |
| --- |
| **Method Detail** |

**optButtonActionPerformed**

private void **optButtonActionPerformed**(java.awt.event.ActionEvent evt)

Actions of option button

**snakeButtonActionPerformed**

private void **snakeButtonActionPerformed**(java.awt.event.ActionEvent evt)

Actions of game button

**tutButtonActionPerformed**

private void **tutButtonActionPerformed**(java.awt.event.ActionEvent evt)

Displays tutorial

**credButtonActionPerformed**

private void **credButtonActionPerformed**(java.awt.event.ActionEvent evt)

**setKeys**

public static void **setKeys**(java.util.Map<java.lang.Integer,java.lang.Integer> keys)

**setColors**

public static void **setColors**(java.awt.Color snakec,

java.awt.Color foodc,

java.awt.Color backc)

**paint**

public void **paint**(java.awt.Graphics g)

**Overrides:**

paint in class javax.swing.JComponent

**Class OptionPanel**

java.lang.Object

java.awt.Component

java.awt.Container

javax.swing.JComponent

javax.swing.JPanel

**OptionPanel**

**All Implemented Interfaces:**

java.awt.image.ImageObserver, java.awt.MenuContainer, java.io.Serializable, javax.accessibility.Accessible

public class **OptionPanel**extends javax.swing.JPanel

**See Also:**

[Serialized Form](serialized-form.html#OptionPanel)

|  |  |
| --- | --- |
| **Nested Class Summary** | |
| class | [**OptionPanel.Listener**](OptionPanel.Listener.html)           Events that occur every second |

|  |  |
| --- | --- |
| **Field Summary** | |
| private  javax.swing.JLabel | [**backgroundLabel**](OptionPanel.html#backgroundLabel)          Examples of slider colors |
| private  javax.swing.JSlider | [**bgb**](OptionPanel.html#bgb)          Sliders for background color |
| private  java.awt.Color | [**bgc**](OptionPanel.html#bgc)          Colors of snake, food, and background |
| private  javax.swing.JSlider | [**bgg**](OptionPanel.html#bgg)          Sliders for background color |
| private  javax.swing.JSlider | [**bgr**](OptionPanel.html#bgr)          Sliders for background color |
| private  java.lang.Integer | [**current**](OptionPanel.html#current)          Current key being changed |
| private  javax.swing.JButton | [**escape**](OptionPanel.html#escape)          Buttons that can be clicked to change keys |
| private  javax.swing.JSlider | [**fb**](OptionPanel.html#fb)          Sliders for Food color |
| private  java.awt.Color | [**fc**](OptionPanel.html#fc)          Colors of snake, food, and background |
| private  javax.swing.JSlider | [**fg**](OptionPanel.html#fg)          Sliders for Food color |
| private  javax.swing.JLabel | [**foodLabel**](OptionPanel.html#foodLabel)          Examples of slider colors |
| private  javax.swing.JSlider | [**fr**](OptionPanel.html#fr)          Sliders for Food color |
| private  javax.swing.JLabel | [**jLabel1**](OptionPanel.html#jLabel1)          Instructions on key changing |
| private  javax.swing.JPanel | [**jPanel1**](OptionPanel.html#jPanel1)          Panel used in GUI |
| private  javax.swing.JButton | [**left**](OptionPanel.html#left)          Buttons that can be clicked to change keys |
| private  javax.swing.JButton | [**menu**](OptionPanel.html#menu)          Buttons that can be clicked to change keys |
| private  javax.swing.JButton | [**mute**](OptionPanel.html#mute)          Buttons that can be clicked to change keys |
| private  java.util.Map<java.lang.Integer,java.lang.Integer> | [**myKeys**](OptionPanel.html#myKeys)          Map containing all the important actions and the respective keyboard locations |
| private  javax.swing.JButton | [**pause**](OptionPanel.html#pause)          Buttons that can be clicked to change keys |
| private  javax.swing.JButton | [**reset**](OptionPanel.html#reset)          Buttons that can be clicked to change keys |
| private  javax.swing.JButton | [**right**](OptionPanel.html#right)          Buttons that can be clicked to change keys |
| private  javax.swing.JSlider | [**sb**](OptionPanel.html#sb)          Sliders for Snake color |
| private  java.awt.Color | [**sc**](OptionPanel.html#sc)          Colors of snake, food, and background |
| private  javax.swing.JSlider | [**sg**](OptionPanel.html#sg)          Sliders for Snake color |
| private  javax.swing.JLabel | [**snakeLabel**](OptionPanel.html#snakeLabel)          Examples of slider colors |
| private  javax.swing.JSlider | [**sr**](OptionPanel.html#sr)          Sliders for Snake color |
| private  javax.swing.Timer | [**timer**](OptionPanel.html#timer)          Timer to update color of Snake, Food, and Background |

|  |  |
| --- | --- |
| **Constructor Summary** | |
| [**OptionPanel**](OptionPanel.html#OptionPanel(java.util.Map, java.awt.Color, java.awt.Color, java.awt.Color))(java.util.Map<java.lang.Integer,java.lang.Integer> keys, java.awt.Color snakec, java.awt.Color foodc, java.awt.Color backgroundc)           Creates new form OptionPanel |  |

|  |  |
| --- | --- |
| **Method Summary** | |
| private  void | [**escapeActionPerformed**](OptionPanel.html#escapeActionPerformed(java.awt.event.ActionEvent))(java.awt.event.ActionEvent evt)          Action when escape button is clicked |
| private  void | [**formKeyPressed**](OptionPanel.html#formKeyPressed(java.awt.event.KeyEvent))(java.awt.event.KeyEvent evt)          Replaces current key with input |
| private  void | [**initComponents**](OptionPanel.html#initComponents())()          Create all elements of GUI |
| private  void | [**leftActionPerformed**](OptionPanel.html#leftActionPerformed(java.awt.event.ActionEvent))(java.awt.event.ActionEvent evt)          Action when left button is clicked |
| private  void | [**menuActionPerformed**](OptionPanel.html#menuActionPerformed(java.awt.event.ActionEvent))(java.awt.event.ActionEvent evt)          Takes user back to the menu |
| private  void | [**muteActionPerformed**](OptionPanel.html#muteActionPerformed(java.awt.event.ActionEvent))(java.awt.event.ActionEvent evt)         Action when mute button is clicked |
| private  void | [**pauseActionPerformed**](OptionPanel.html#pauseActionPerformed(java.awt.event.ActionEvent))(java.awt.event.ActionEvent evt)       Action when pause button is clicked |
| private  void | [**resetActionPerformed**](OptionPanel.html#resetActionPerformed(java.awt.event.ActionEvent))(java.awt.event.ActionEvent evt)         Action when reset button is clicked |
| private  void | [**rightActionPerformed**](OptionPanel.html#rightActionPerformed(java.awt.event.ActionEvent))(java.awt.event.ActionEvent evt)         Action when right button is clicked |

|  |
| --- |
| **Field Detail** |

**myKeys**

private java.util.Map<java.lang.Integer,java.lang.Integer> **myKeys**

Map containing all the important actions and the respective keyboard locations

**current**

private java.lang.Integer **current**

Current key being changed

**timer**

private javax.swing.Timer **timer**

Timer to update color of Snake, Food, and Background

**sc**

private java.awt.Color **sc**

Colors of snake, food, and background

**fc**

private java.awt.Color **fc**

Colors of snake, food, and background

**bgc**

private java.awt.Color **bgc**

Colors of snake, food, and background

**backgroundLabel**

private javax.swing.JLabel **backgroundLabel**

Examples of slider colors

**snakeLabel**

private javax.swing.JLabel **snakeLabel**

Examples of slider colors

**foodLabel**

private javax.swing.JLabel **foodLabel**

Examples of slider colors

**bgb**

private javax.swing.JSlider **bgb**

Sliders for background color

**bgg**

private javax.swing.JSlider **bgg**

Sliders for background color

**bgr**

private javax.swing.JSlider **bgr**

Sliders for background color

**escape**

private javax.swing.JButton **escape**

Buttons that can be clicked to change keys

**left**

private javax.swing.JButton **left**

Buttons that can be clicked to change keys

**right**

private javax.swing.JButton **right**

Buttons that can be clicked to change keys

**menu**

private javax.swing.JButton **menu**

Buttons that can be clicked to change keys

**pause**

private javax.swing.JButton **pause**

Buttons that can be clicked to change keys

**reset**

private javax.swing.JButton **reset**

Buttons that can be clicked to change keys

**mute**

private javax.swing.JButton **mute**

Buttons that can be clicked to change keys

**fb**

private javax.swing.JSlider **fb**

Sliders for Food color

**fg**

private javax.swing.JSlider **fg**

Sliders for Food color

**fr**

private javax.swing.JSlider **fr**

Sliders for Food color

**jLabel1**

private javax.swing.JLabel **jLabel1**

Instructions on key changing

**jPanel1**

private javax.swing.JPanel **jPanel1**

Panel used in GUI

**sb**

private javax.swing.JSlider **sb**

Sliders for Snake color

**sg**

private javax.swing.JSlider **sg**

Sliders for Snake color

**sr**

private javax.swing.JSlider **sr**

Sliders for Snake color

|  |
| --- |
| **Constructor Detail** |

**OptionPanel**

public **OptionPanel**(java.util.Map<java.lang.Integer,java.lang.Integer> keys,

java.awt.Color snakec,

java.awt.Color foodc,

java.awt.Color backgroundc)

Creates new form OptionPanel

|  |
| --- |
| **Method Detail** |

**leftActionPerformed**

private void **leftActionPerformed**(java.awt.event.ActionEvent evt)

Action when left button is clicked

**rightActionPerformed**

private void **rightActionPerformed**(java.awt.event.ActionEvent evt)

Action when right button is clicked

**pauseActionPerformed**

private void **pauseActionPerformed**(java.awt.event.ActionEvent evt)

Action when pause button is clicked

**escapeActionPerformed**

private void **escapeActionPerformed**(java.awt.event.ActionEvent evt)

Action when escape button is clicked

**resetActionPerformed**

private void **resetActionPerformed**(java.awt.event.ActionEvent evt)

Action when reset button is clicked

**muteActionPerformed**

private void **muteActionPerformed**(java.awt.event.ActionEvent evt)

Action when mute button is clicked

**menuActionPerformed**

private void **menuActionPerformed**(java.awt.event.ActionEvent evt)

Takes user back to the menu

**formKeyPressed**

private void **formKeyPressed**(java.awt.event.KeyEvent evt)

Replaces current key with input

**initComponents**

private void **initComponents**()

Create all elements of GUI

**Class DiffPanel**

java.lang.Object

java.awt.Component

java.awt.Container

javax.swing.JComponent

javax.swing.JPanel

**DiffPanel**

**All Implemented Interfaces:**

java.awt.image.ImageObserver, java.awt.MenuContainer, java.io.Serializable, javax.accessibility.Accessible

public class **DiffPanel**extends javax.swing.JPanel

**See Also:**

[Serialized Form](serialized-form.html#DiffPanel)

|  |  |
| --- | --- |
| **Nested Class Summary** | |
| class | [**DiffPanel.EasyListener**](DiffPanel.EasyListener.html)          Gives description of easy level |
| class | [**DiffPanel.HardListener**](DiffPanel.HardListener.html)          Gives description of hard level |
| class | [**DiffPanel.InsaneListener**](DiffPanel.InsaneListener.html)          Gives description of insane level |
| class | [**DiffPanel.MediumListener**](DiffPanel.MediumListener.html)          Gives description of medium level |

|  |  |
| --- | --- |
| **Field Summary** | |
| private  java.awt.Color | [**bgc**](DiffPanel.html#bgc)          Snake, Food, and background colors |
| (package private)  javax.swing.JPanel | [**cPanel**](DiffPanel.html#cPanel)          subPanel with buttons |
| (package private)  javax.swing.JTextArea | [**description**](DiffPanel.html#description)          Description of each difficulty |
| (package private)  javax.swing.JButton | [**easy**](DiffPanel.html#easy)          Buttons for difficulty choice |
| private  java.awt.Color | [**fc**](DiffPanel.html#fc)          Snake, Food, and background colors |
| (package private)  javax.swing.JButton | [**hard**](DiffPanel.html#hard)          Buttons for difficulty choice |
| (package private)  javax.swing.JButton | [**insane**](DiffPanel.html#insane)          Buttons for difficulty choice |
| (package private)  javax.swing.JButton | [**medium**](DiffPanel.html#medium)          Buttons for difficulty choice |
| private  java.util.Map<java.lang.Integer,java.lang.Integer> | [**myKeys**](DiffPanel.html#myKeys)          Map of the possible input keys and their actual input keys |
| private  java.awt.Color | [**sc**](DiffPanel.html#sc)          Snake, Food, and background colors |
| (package private)  SnakePanel | [**snake**](DiffPanel.html#snake)          SnakePanel to be loaded for game |

|  |  |
| --- | --- |
| **Constructor Summary** | |
| [**DiffPanel**](DiffPanel.html#DiffPanel(java.util.Map, java.awt.Color, java.awt.Color, java.awt.Color))(java.util.Map<java.lang.Integer,java.lang.Integer> keys, java.awt.Color snakec, java.awt.Color foodc, java.awt.Color backc)           Creates a difficulty panel where you can choose difficulty of the Snake game |  |

|  |
| --- |
| **Field Detail** |

**easy**

javax.swing.JButton **easy**

Buttons for difficulty choice

**medium**

javax.swing.JButton **medium**

Buttons for difficulty choice

**hard**

javax.swing.JButton **hard**

Buttons for difficulty choice

**insane**

javax.swing.JButton **insane**

Buttons for difficulty choice

**description**

javax.swing.JTextArea **description**

Description of each difficulty

**cPanel**

javax.swing.JPanel **cPanel**

subPanel with buttons

**snake**

SnakePanel **snake**

SnakePanel to be loaded for game

**myKeys**

private java.util.Map<java.lang.Integer,java.lang.Integer> **myKeys**

Map of the possible input keys and their actual input keys

**sc**

private java.awt.Color **sc**

Snake, Food, and background colors

**fc**

private java.awt.Color **fc**

Snake, Food, and background colors

**bgc**

private java.awt.Color **bgc**

Snake, Food, and background colors

|  |
| --- |
| **Constructor Detail** |

**DiffPanel**

public **DiffPanel**(java.util.Map<java.lang.Integer,java.lang.Integer> keys,

java.awt.Color snakec,

java.awt.Color foodc,

java.awt.Color backc)

Creates a difficulty panel where you can choose difficulty of the Snake game

**Parameters:**

keys - The user chosen controls

snakec - Color of snake

foodc - Color of food

backc - Color of background

**Class SnakePanel**

java.lang.Object

java.awt.Component

java.awt.Container

javax.swing.JComponent

javax.swing.JPanel

**SnakePanel**

**All Implemented Interfaces:**

java.awt.image.ImageObserver, java.awt.MenuContainer, java.io.Serializable, javax.accessibility.Accessible

public class **SnakePanel**extends javax.swing.JPanel

The JPanel that runs the main game

**See Also:**

[Serialized Form](serialized-form.html#SnakePanel)

|  |  |
| --- | --- |
| **Nested Class Summary** | |
| class | [**SnakePanel.Key**](SnakePanel.Key.html)          Codes for keyboard input |
| class | [**SnakePanel.LeftListener**](SnakePanel.LeftListener.html)          Turns snake left constantly after start by keyboard until stop by keyboard |
| class | [**SnakePanel.Listener**](SnakePanel.Listener.html)          Events that occur every second |
| class | [**SnakePanel.RightListener**](SnakePanel.RightListener.html)          Turns snake right constantly after start by keyboard until stop by keyboard |
| class | [**SnakePanel.SpeedListener**](SnakePanel.SpeedListener.html)          Moves snake and increments score; change in speed of timer changes speed of snake |

|  |  |
| --- | --- |
| **Field Summary** | |
| private  javax.swing.Timer | [**action**](SnakePanel.html#action)          Timers for animation and other time-related processes |
| private  long | [**anyTime**](SnakePanel.html#anyTime)          Used as a stopwatch |
| private  java.awt.Color | [**BACKGROUND**](SnakePanel.html#BACKGROUND)          Color of background |
| private  boolean | [**betweenLevels**](SnakePanel.html#betweenLevels)          On a screen between levels? |
| private  Sound | [**chomp**](SnakePanel.html#chomp)          Sound that plays when food is eaten |
| private  java.lang.String | [**diff**](SnakePanel.html#diff)          Difficulty |
| private  int | [**FRAME**](SnakePanel.html#FRAME)          Size of frame |
| private  java.util.ArrayList<java.lang.String> | [**hsnames**](SnakePanel.html#hsnames)          List of highscore names |
| private  java.util.ArrayList<java.lang.Integer> | [**hsscores**](SnakePanel.html#hsscores)          List of highscore scores |
| private  boolean | [**isGameOver**](SnakePanel.html#isGameOver)          Is the game over? |
| private  boolean | [**isMuted**](SnakePanel.html#isMuted)          Is the game muted? |
| private  long | [**lastPressProcessed**](SnakePanel.html#lastPressProcessed)          Keeps the Snake from turning too fast |
| private  javax.swing.Timer | [**leftTimer**](SnakePanel.html#leftTimer)          Timers for animation and other time-related processes |
| private  int | [**level**](SnakePanel.html#level)          Current level |
| private  Food | [**mainFood**](SnakePanel.html#mainFood)          Main game food; can be eaten and regenerated |
| private  java.awt.Graphics2D | [**myBuffer**](SnakePanel.html#myBuffer)          Graphics class for graphics |
| private  java.awt.image.BufferedImage | [**myImage**](SnakePanel.html#myImage)          Image that graphics are painted on to |
| private  java.util.Map<java.lang.Integer,java.lang.Integer> | [**myKeys**](SnakePanel.html#myKeys)          Map of the possible input keys and their actual input keys |
| private  javax.swing.Timer | [**rightTimer**](SnakePanel.html#rightTimer)          Timers for animation and other time-related processes |
| private  double | [**score**](SnakePanel.html#score)          Current score |
| Sound | [**scottwinter**](SnakePanel.html#scottwinter)          Music that plays during game |
| private  Snake | [**snake**](SnakePanel.html#snake)          Main game snake |
| private  javax.swing.Timer | [**snakespeed**](SnakePanel.html#snakespeed)          Timers for animation and other time-related processes |
| private  Sound | [**stageclear**](SnakePanel.html#stageclear)          Sound that plays on level up |
| private  long | [**startTime**](SnakePanel.html#startTime)          Start time |
| private  int | [**wall**](SnakePanel.html#wall)          Size of in-game wall |

|  |  |
| --- | --- |
| **Constructor Summary** | |
| [**SnakePanel**](SnakePanel.html#SnakePanel(int, java.util.Map, java.awt.Color, java.awt.Color, java.awt.Color))(int speed, java.util.Map<java.lang.Integer,java.lang.Integer> keys, java.awt.Color sc, java.awt.Color fc, java.awt.Color bgc)           Constructor of the game panel |  |

|  |  |
| --- | --- |
| **Method Summary** | |
| void | [**drawFood**](SnakePanel.html#drawFood())()          Draw the food on the screen |
| void | [**drawSnake**](SnakePanel.html#drawSnake())()          Draw the snake on the screen |
| void | [**eatFood**](SnakePanel.html#eatFood(Food))(Food f)          Eat the food, update score, and generate new food |
| void | [**gameOver**](SnakePanel.html#gameOver())()          Carries out game over situation |
| void | [**increaseLevel**](SnakePanel.html#increaseLevel())()          Increases the level |
| boolean | [**isDead**](SnakePanel.html#isDead())()          Tests if the snake is dead and the game is over |
| void | [**paintComponent**](SnakePanel.html#paintComponent(java.awt.Graphics))(java.awt.Graphics g)          Paint the screen |
| void | [**saveScores**](SnakePanel.html#saveScores())()          Save high scores to file |

|  |
| --- |
| **Field Detail** |

**FRAME**

private final int **FRAME**

Size of frame

**See Also:**

[Constant Field Values](constant-values.html#SnakePanel.FRAME)

**BACKGROUND**

private java.awt.Color **BACKGROUND**

Color of background

**myImage**

private java.awt.image.BufferedImage **myImage**

Image that graphics are painted on to

**myBuffer**

private java.awt.Graphics2D **myBuffer**

Graphics class for graphics

**snake**

private Snake **snake**

Main game snake

**mainFood**

private Food **mainFood**

Main game food; can be eaten and regenerated

**lastPressProcessed**

private long **lastPressProcessed**

Keeps the Snake from turning too fast

**isGameOver**

private boolean **isGameOver**

Is the game over?

**wall**

private int **wall**

Size of in-game wall

**score**

private double **score**

Current score

**level**

private int **level**

Current level

**diff**

private java.lang.String **diff**

Difficulty

**hsnames**

private java.util.ArrayList<java.lang.String> **hsnames**

List of highscore names

**hsscores**

private java.util.ArrayList<java.lang.Integer> **hsscores**

List of highscore scores

**betweenLevels**

private boolean **betweenLevels**

On a screen between levels?

**anyTime**

private long **anyTime**

Used as a stopwatch

**startTime**

private long **startTime**

Start time

**action**

private javax.swing.Timer **action**

Timers for animation and other time-related processes

**snakespeed**

private javax.swing.Timer **snakespeed**

Timers for animation and other time-related processes

**leftTimer**

private javax.swing.Timer **leftTimer**

Timers for animation and other time-related processes

**rightTimer**

private javax.swing.Timer **rightTimer**

Timers for animation and other time-related processes

**isMuted**

private boolean **isMuted**

Is the game muted?

**scottwinter**

public Sound **scottwinter**

Music that plays during game

**stageclear**

private Sound **stageclear**

Sound that plays on level up

**chomp**

private Sound **chomp**

Sound that plays when food is eaten

**myKeys**

private java.util.Map<java.lang.Integer,java.lang.Integer> **myKeys**

Map of the possible input keys and their actual input keys

|  |
| --- |
| **Constructor Detail** |

**SnakePanel**

public **SnakePanel**(int speed,

java.util.Map<java.lang.Integer,java.lang.Integer> keys,

java.awt.Color sc,

java.awt.Color fc,

java.awt.Color bgc)

Constructor of the game panel

|  |
| --- |
| **Method Detail** |

**paintComponent**

public void **paintComponent**(java.awt.Graphics g)

Paint the screen

**Overrides:**

paintComponent in class javax.swing.JComponent

**drawSnake**

public void **drawSnake**()

Draw the snake on the screen

**drawFood**

public void **drawFood**()

Draw the food on the screen

**eatFood**

public void **eatFood**(Food f)

Eat the food, update score, and generate new food

**Parameters:**

f - The Food to be eaten

**gameOver**

public void **gameOver**()

Carries out game over situation

**isDead**

public boolean **isDead**()

Tests if the snake is dead and the game is over

**Returns:**

Is the snake dead?

**increaseLevel**

public void **increaseLevel**()

Increases the level

**saveScores**

public void **saveScores**()

Save high scores to file

**Class Sound**

java.lang.Object

**Sound**

public class **Sound**extends java.lang.Object

A Sound class that plays music and other sounds

|  |  |
| --- | --- |
| **Field Summary** | |
| (package private)  javax.sound.sampled.AudioInputStream | [**ais**](Sound.html#ais) |
| private static int | [**EXTERNAL\_BUFFER\_SIZE**](Sound.html#EXTERNAL_BUFFER_SIZE)          Buffer rate of the sound |
| (package private)  javax.sound.sampled.Clip | [**line**](Sound.html#line)          The Clip that plays the sound |
| (package private)  boolean | [**oncePlayed**](Sound.html#oncePlayed)          Has the sound been played at least once? |

|  |  |
| --- | --- |
| **Constructor Summary** | |
| [**Sound**](Sound.html#Sound(java.lang.String))(java.lang.String filename)           The constructor of SoundTest. |  |

|  |  |
| --- | --- |
| **Method Summary** | |
| void | [**exit**](Sound.html#exit())()          Close and exit the file |
| void | [**loop**](Sound.html#loop())()          Loops the music |
| void | [**play**](Sound.html#play())()          Plays the music. |
| void | [**stop**](Sound.html#stop())()          Pauses the music |

|  |
| --- |
| **Field Detail** |

**EXTERNAL\_BUFFER\_SIZE**

private static final int **EXTERNAL\_BUFFER\_SIZE**

Buffer rate of the sound

**See Also:**

[Constant Field Values](constant-values.html#Sound.EXTERNAL_BUFFER_SIZE)

**line**

javax.sound.sampled.Clip **line**

The Clip that plays the sound

**oncePlayed**

boolean **oncePlayed**

Has the sound been played at least once?

**ais**

javax.sound.sampled.AudioInputStream **ais**

|  |
| --- |
| **Constructor Detail** |

**Sound**

public **Sound**(java.lang.String filename)

The constructor of SoundTest. The filename is used to be read in and the audio is later gotten from it and used.

**Parameters:**

filename - The filename of the file containing the sound

|  |
| --- |
| **Method Detail** |

**play**

public void **play**()

Plays the music.

**loop**

public void **loop**()

Loops the music

**stop**

public void **stop**()

Pauses the music

**exit**

public void **exit**()

Close and exit the file

**Class Snake**

java.lang.Object

**Snake**

public class **Snake**extends java.lang.Object

A Snake class. SEE segments FIELD FOR DATA STRUCTURE

|  |  |
| --- | --- |
| **Field Summary** | |
| java.awt.Color | [**myColor**](Snake.html#myColor)          Color of snake |
| double | [**myDir**](Snake.html#myDir)          Direction of snake |
| int | [**myLength**](Snake.html#myLength)          Length of snake i.e. |
| double | [**mySpeed**](Snake.html#mySpeed)          Speed factor of snake |
| java.util.ArrayList<Segment> | [**segments**](Snake.html#segments)          The segments that make up the snake. |

|  |  |
| --- | --- |
| **Constructor Summary** | |
| [**Snake**](Snake.html#Snake(java.awt.Color))(java.awt.Color c)          Creates a snake of color c |  |

|  |  |
| --- | --- |
| **Method Summary** | |
| void | [**addSegment**](Snake.html#addSegment(Segment))(Segment seg)          Add a segment to the snake and increment length |
| java.awt.Color | [**getColor**](Snake.html#getColor())()          Returns the color of the snake |
| double | [**getDir**](Snake.html#getDir())()          Returns the direciton of the snake |
| double | [**getDistance**](Snake.html#getDistance(int, int))(int x, int y)          Returns snake's distance from a point |
| Segment | [**getHead**](Snake.html#getHead())()          Returns first Segment in the snake; identical to calling getSegment(0) |
| int | [**getLength**](Snake.html#getLength())()          Returns the length of the snake |
| Segment | [**getSegment**](Snake.html#getSegment(int))(int k)          Returns the Segment of number k in the segments field |
| java.util.ArrayList<Segment> | [**getSegments**](Snake.html#getSegments())()          Returns the Segments of the snake |
| void | [**move**](Snake.html#move())()          Moves the snake in its current direction |
| Segment | [**removeSegment**](Snake.html#removeSegment())()          Remove a segment and decrement length |
| void | [**setColor**](Snake.html#setColor(java.awt.Color))(java.awt.Color c)          Sets the snake's color |
| void | [**setDir**](Snake.html#setDir(double))(double dir)          Changes the snake's direction |
| void | [**turnLeft**](Snake.html#turnLeft())()          Turns snake left |
| void | [**turnRight**](Snake.html#turnRight())()          Turns snake right |

|  |
| --- |
| **Field Detail** |

**segments**

public java.util.ArrayList<Segment> **segments**

The segments that make up the snake. Segments can be added, removed, and gotten from index using this data structure

**mySpeed**

public double **mySpeed**

Speed factor of snake

**myDir**

public double **myDir**

Direction of snake

**myColor**

public java.awt.Color **myColor**

Color of snake

**myLength**

public int **myLength**

Length of snake i.e. number of segments

|  |
| --- |
| **Constructor Detail** |

**Snake**

public **Snake**(java.awt.Color c)

Creates a snake of color c

**Parameters:**

c - The color of the snake

|  |
| --- |
| **Method Detail** |

**move**

public void **move**()

Moves the snake in its current direction

**addSegment**

public void **addSegment**(Segment seg)

Add a segment to the snake and increment length

**Parameters:**

seg - The segment to be added

**removeSegment**

public Segment **removeSegment**()

Remove a segment and decrement length

**Returns:**

The removed segment

**getDir**

public double **getDir**()

Returns the direciton of the snake

**Returns:**

The current direction

**getLength**

public int **getLength**()

Returns the length of the snake

**Returns:**

The current length

**getColor**

public java.awt.Color **getColor**()

Returns the color of the snake

**Returns:**

The current color

**setColor**

public void **setColor**(java.awt.Color c)

Sets the snake's color

**Parameters:**

c - The new color of the snake

**getSegments**

public java.util.ArrayList<Segment> **getSegments**()

Returns the Segments of the snake

**Returns:**

The segments field making up the snake

**getSegment**

public Segment **getSegment**(int k)

Returns the Segment of number k in the segments field

**Parameters:**

k - number of the segment

**Returns:**

A segment of number k

**getHead**

public Segment **getHead**()

Returns first Segment in the snake; identical to calling getSegment(0)

**Returns:**

The first Segment in the segments field

**See Also:**

getSegments(int k)

**turnRight**

public void **turnRight**()Turns snake right

**turnLeft**

public void **turnLeft**()Turns snake left

**setDir**

public void **setDir**(double dir)

Changes the snake's direction

**Parameters:** dir - The new direction of the snake

**getDistance**

public double **getDistance**(int x,

int y)

Returns snake's distance from a point

**Parameters:**

x - x-coordinate of point

y - y-coordinate of point

**Returns:** distance of snake from point

**Class Segment**

java.lang.Object

**Segment**

public class **Segment**extends java.lang.Object

The segment class that makes up the Snake.

|  |  |
| --- | --- |
| **Field Summary** | |
| java.awt.Color | [**myColor**](Segment.html#myColor)          Color of the segment |
| java.awt.Rectangle | [**myRect**](Segment.html#myRect)          Square with upper left corner of the segment's coordinates |
| int | [**mySize**](Segment.html#mySize)          Size of the segment |
| int | [**myX**](Segment.html#myX)          Coordinates of the segment |
| int | [**myY**](Segment.html#myY)          Coordinates of the segment |

|  |  |
| --- | --- |
| **Constructor Summary** | |
| [**Segment**](Segment.html#Segment(int, int, java.awt.Color))(int x, int y, java.awt.Color color)          Segment constructor |  |

|  |  |
| --- | --- |
| **Method Summary** | |
| java.awt.Color | [**getColor**](Segment.html#getColor())()          Get the color of the segment |
| java.awt.Rectangle | [**getRect**](Segment.html#getRect())()          Returns the square representing the segment |
| int | [**getSize**](Segment.html#getSize())()          Get the size of the segment |
| int | [**getX**](Segment.html#getX())()          Get the x-coordinate of the segment |
| int | [**getY**](Segment.html#getY())()          Get the y-coordinate of the segment |
| void | [**setColor**](Segment.html#setColor(java.awt.Color))(java.awt.Color c)          Set the color of the segment |
| void | [**setSize**](Segment.html#setSize(int))(int size)          Set the size of the segment |
| void | [**setX**](Segment.html#setX(int))(int x)          Set the x-coordinate of the segment |
| void | [**setY**](Segment.html#setY(int))(int y)          Set the y-coordinate of the segment |

|  |
| --- |
| **Field Detail** |

**myColor**

public java.awt.Color **myColor**

Color of the segment

**myX**

public int **myX**

Coordinates of the segment

**myY**

public int **myY**

Coordinates of the segment

**mySize**

public int **mySize**

Size of the segment

**myRect**

public java.awt.Rectangle **myRect**

Square with upper left corner of the segment's coordinates

|  |
| --- |
| **Constructor Detail** |

**Segment**

public **Segment**(int x,

int y,

java.awt.Color color)

Segment constructor

**Parameters:**

x - The x-coordinate, in pixels, of myLoc

y - The y-coordinate, in pixels, of myLoc

color - The color of this segment.

|  |
| --- |
| **Method Detail** |

**getColor**

public java.awt.Color **getColor**()

Get the color of the segment

**Returns:**

the current color of the segment

**getX**

public int **getX**()

Get the x-coordinate of the segment

**Returns:**

the x-coordinate of the segment

**getY**

public int **getY**()

Get the y-coordinate of the segment

**Returns:**

the y-coordinate of the segment

**getSize**

public int **getSize**()

Get the size of the segment

**Returns:**

the current size of the segment

**getRect**

public java.awt.Rectangle **getRect**()

Returns the square representing the segment

**Returns:**

the square with upper left corner of the segment's coordinates

**setColor**

public void **setColor**(java.awt.Color c)

Set the color of the segment

**Parameters:**

c - the new color of the segment

**setX**

public void **setX**(int x)

Set the x-coordinate of the segment

**Parameters:**

x - the new x-coordinate of the segment

**setY**

public void **setY**(int y)

Set the y-coordinate of the segment

**Parameters:**

y - the new y-coordinate of the segment

**setSize**

public void **setSize**(int size)

Set the size of the segment

**Parameters:**

size - the new size of the segment

**Class Food**

java.lang.Object

**Food**

public class **Food**extends java.lang.Object

The Food class; eaten by the Snake

|  |  |
| --- | --- |
| **Field Summary** | |
| java.awt.geom.Ellipse2D.Double | [**myCirc**](Food.html#myCirc)          Circle bounded by square with upper left corner myX, myY |
| java.awt.Color | [**myColor**](Food.html#myColor)          Color of food |
| int | [**myWorth**](Food.html#myWorth)          Worth of food |
| int | [**myX**](Food.html#myX)          Coordinates of food |
| int | [**myY**](Food.html#myY)          Coordinates of food |

|  |  |
| --- | --- |
| **Constructor Summary** | |
| [**Food**](Food.html#Food(int, int, int, java.awt.Color))(int x, int y, int worth, java.awt.Color c)           Constructor for the food |  |

|  |  |
| --- | --- |
| **Method Summary** | |
| java.awt.geom.Ellipse2D | [**getCircle**](Food.html#getCircle())()          Returns the circle defining the food |
| java.awt.Color | [**getColor**](Food.html#getColor())()          Returns color of food |
| int | [**getWorth**](Food.html#getWorth())()          Returns the worth of the food |
| int | [**getX**](Food.html#getX())()          Returns x-coordinate of food |
| int | [**getY**](Food.html#getY())()          Returns y-coordinate of food |
| void | [**setColor**](Food.html#setColor(java.awt.Color))(java.awt.Color c)          Sets color of food |

|  |
| --- |
| **Field Detail** |

**myX**

public int **myX**

Coordinates of food

**myY**

public int **myY**

Coordinates of food

**myWorth**

public int **myWorth**

Worth of food

**myCirc**

public java.awt.geom.Ellipse2D.Double **myCirc**

Circle bounded by square with upper left corner myX, myY

**myColor**

public java.awt.Color **myColor**

Color of food

|  |
| --- |
| **Constructor Detail** |

**Food**

public **Food**(int x,

int y,

int worth,

java.awt.Color c)

Constructor for the food

**Parameters:**

x - x-coordinate of food

y - y-coordinate of food

worth - worth of food

|  |
| --- |
| **Method Detail** |

**getX**

public int **getX**()

Returns x-coordinate of food

**Returns:** the x coordinate of the food

**getY**

public int **getY**()

Returns y-coordinate of food

**Returns:** the y coordinate of the food

**getCircle**

public java.awt.geom.Ellipse2D **getCircle**()

Returns the circle defining the food

**Returns:** the circle bounded by the square with the upper left corner of the food's coordinates

**getWorth**

public int **getWorth**()

Returns the worth of the food

**Returns:** the worth of the food

**setColor**

public void **setColor**(java.awt.Color c)

Sets color of food

**Parameters:** c - Color of food

**getColor**

public java.awt.Color **getColor**()

Returns color of food

**Returns:**

Color of food