

```
1 import java.util.*;
2 import javax.imageio.ImageIO;
3 import java.util.Timer;
4 import java.awt.*;
5 import java.awt.event.*;
6 import java.awt.image.*;
7 import java.io.*;
8 import javax.swing.*;
9
10 class Game extends JPanel {
11     private Timer timer;
12     private Snake snake;
13     /**
14      *
15      */
16     private Point cherry;
17     public Point getCherry() {
18         return cherry;
19     }
20
21     public void setCherry(Point cherry) {
22         this.cherry = cherry;
23     }
24
25     private int points = 0;
26     private int best = 0;
27     private BufferedImage image;
28     private GameStatus status;
29     private boolean didLoadCherryImage = true;
30
31     private static Font FONT_M = new Font("MV Boli",
        Font.PLAIN, 24);
32     private static Font FONT_M_ITALIC = new Font("MV
        Boli", Font.ITALIC, 24);
33     private static Font FONT_L = new Font("MV Boli",
        Font.PLAIN, 84);
34     private static Font FONT_XL = new Font("MV Boli"
        , Font.PLAIN, 150);
35     private static int WIDTH = 760;
36     private static int HEIGHT = 520;
37     private static int DELAY = 50;
```

```
38
39     // Constructor
40     public Game() {
41         try {
42             image = ImageIO.read(new File("cherry.png
43         "));
44         } catch (IOException e) {
45             didLoadCherryImage = false;
46         }
47
48         addKeyListener(new KeyListener());
49         setFocusable(true);
50         setBackground(new Color(130, 205, 71));
51         setDoubleBuffered(true);
52
53         snake = new Snake(WIDTH / 2, HEIGHT / 2);
54         status = GameStatus.NOT_STARTED;
55         repaint();
56     }
57
58     @Override
59     public void paintComponent(Graphics g) {
60         super.paintComponent(g);
61
62         render(g);
63
64         Toolkit.getDefaultToolkit().sync();
65     }
66
67     // Render the game
68     private void update() {
69         snake.move();
70
71         if (cherry != null && snake.getHead().
72         intersects(cherry, 20)) {
73             snake.addTail();
74             cherry = null;
75             points++;
76         }
77
78         if (cherry == null) {
```

```

77         spawnCherry();
78     }
79
80     checkForGameOver();
81 }
82
83 private void reset() {
84     points = 0;
85     cherry = null;
86     snake = new Snake(WIDTH / 2, HEIGHT / 2);
87     setStatus(GameStatus.RUNNING);
88 }
89
90 private void setStatus(GameStatus newStatus) {
91     switch(newStatus) {
92         case RUNNING:
93             timer = new Timer();
94             timer.schedule(new GameLoop(), 0,
100         DELAY);
101             break;
95         case PAUSED:
96             timer.cancel();
97         case GAME_OVER:
98             timer.cancel();
99             best = points > best ? points : best
100         ;
101             break;
102     }
103
104     status = newStatus;
105 }
106
107 private void togglePause() {
108     setStatus(status == GameStatus.PAUSED ?
109     GameStatus.RUNNING : GameStatus.PAUSED);
110 }
111
112 // Check if the snake has hit the wall or itself
113 private void checkForGameOver() {
114     Point head = snake.getHead();
115     boolean hitBoundary = head.getX() <= 20

```

```

115         || head.getX() >= WIDTH + 10
116         || head.getY() <= 40
117         || head.getY() >= HEIGHT + 30;
118
119         boolean ateItself = false;
120
121         for(Point t : snake.getTail()) {
122             ateItself = ateItself || head.equals(t);
123         }
124
125         if (hitBoundary || ateItself) {
126             setStatus(GameStatus.GAME_OVER);
127         }
128     }
129
130     // Spawn a cherry at a random location
131     public void drawCenteredString(Graphics g,
String text, Font font, int y) {
132         FontMetrics metrics = g.getFontMetrics(font
);
133         int x = (WIDTH - metrics.stringWidth(text
)) / 2;
134
135         g.setFont(font);
136         g.drawString(text, x, y);
137     }
138
139     private void render(Graphics g) {
140         Graphics2D g2d = (Graphics2D) g;
141
142         g2d.setColor(Color.BLACK);
143         g2d.setFont(FONT_M);
144
145         if (status == GameStatus.NOT_STARTED) {
146             drawCenteredString(g2d, "SNAKE", FONT_XL
, 200);
147             drawCenteredString(g2d, "GAME", FONT_XL
, 300);
148             drawCenteredString(g2d, "Press any key
to begin", FONT_M_ITALIC, 330);
149

```

```

150         return;
151     }
152
153     Point p = snake.getHead();
154
155     g2d.drawString("SCORE: " + String.format ("%
02d", points), 20, 30);
156     g2d.drawString("BEST: " + String.format ("%
02d", best), 630, 30);
157
158     if (cherry != null) {
159         if (didLoadCherryImage) {
160             g2d.drawImage(image, cherry.getX(),
cherry.getY(), 60, 60, null);
161         } else {
162             g2d.setColor(Color.BLACK);
163             g2d.fillOval(cherry.getX(), cherry.
getY(), 10, 10);
164             g2d.setColor(Color.BLACK);
165         }
166     }
167
168     if (status == GameStatus.GAME_OVER) {
169         drawCenteredString(g2d, "Press enter
to start again", FONT_M_ITALIC, 330);
170         drawCenteredString(g2d, "GAME OVER",
FONT_L, 300);
171     }
172
173     if (status == GameStatus.PAUSED) {
174         g2d.drawString("Paused", 600, 14);
175     }
176
177     g2d.setColor(new Color(33, 70, 199));
178     g2d.fillRect(p.getX(), p.getY(), 10, 10);
179
180     for(int i = 0, size = snake.getTail().size
()); i < size; i++) {
181         Point t = snake.getTail().get(i);
182
183         g2d.fillRect(t.getX(), t.getY(), 10, 10

```

```

183 );
184     }
185
186     g2d.setColor(Color.RED);
187     g2d.setStroke(new BasicStroke(4));
188     g2d.drawRect(20, 40, WIDTH, HEIGHT);
189 }
190
191     // spawn cherry in random position
192     public void spawnCherry() {
193         cherry = new Point((new Random()).nextInt(
194             WIDTH - 60) + 20,
195             (new Random()).nextInt(HEIGHT - 60
196             ) + 40);
197     }
198
199     // game loop
200     private class KeyListener extends KeyAdapter {
201         @Override
202         public void keyPressed(KeyEvent e) {
203             int key = e.getKeyCode();
204
205             if (status == GameState.RUNNING) {
206                 switch(key) {
207                     case KeyEvent.VK_LEFT: snake.
208                     turn(Direction.LEFT); break;
209                     case KeyEvent.VK_RIGHT: snake.
210                     turn(Direction.RIGHT); break;
211                     case KeyEvent.VK_UP: snake.turn(
212                     Direction.UP); break;
213                     case KeyEvent.VK_DOWN: snake.
214                     turn(Direction.DOWN); break;
215                 }
216             }
217
218             if (status == GameState.NOT_STARTED) {
219                 setStatus(GameState.RUNNING);
220             }
221
222             if (status == GameState.GAME_OVER &&
223             key == KeyEvent.VK_ENTER) {

```

```
217         reset();
218     }
219
220     if (key == KeyEvent.VK_P) {
221         togglePause();
222     }
223 }
224 }
225
226 private class GameLoop extends java.util.
    TimerTask {
227     public void run() {
228         update();
229         repaint();
230     }
231 }
232 }
233
234
235 enum GameStatus
236 {
237     NOT_STARTED, RUNNING, PAUSED, GAME_OVER
238 }
239
240 // direction of snake
241 enum Direction {
242     UP, DOWN, LEFT, RIGHT;
243
244     public boolean isX() {
245         return this == LEFT || this == RIGHT;
246     }
247
248     public boolean isY() {
249         return this == UP || this == DOWN;
250     }
251 }
252
253
254 class Point {
255     private int x;
256     private int y;
```

```
257
258     public Point(int x, int y) {
259         this.x = x;
260         this.y = y;
261     }
262
263     public Point(Point p) {
264         this.x = p.getX();
265         this.y = p.getY();
266     }
267
268     public void move(Direction d, int value) {
269         switch(d) {
270             case UP: this.y -= value; break;
271             case DOWN: this.y += value; break;
272             case RIGHT: this.x += value; break;
273             case LEFT: this.x -= value; break;
274         }
275     }
276
277     public int getX() {
278         return x;
279     }
280
281     public int getY() {
282         return y;
283     }
284
285     public Point setX(int x) {
286         this.x = x;
287
288         return this;
289     }
290
291     public Point setY(int y) {
292         this.y = y;
293
294         return this;
295     }
296
297     public boolean equals(Point p) {
```



```
298         return this.x == p.getX() && this.y == p.  
           getY();  
299     }  
300  
301     public String toString() {  
302         return "(" + x + ", " + y + ")";  
303     }  
304  
305     public boolean intersects(Point p) {  
306         return intersects(p, 10);  
307     }  
308  
309     public boolean intersects(Point p, int tolerance  
310 ) {  
311         int diffX = Math.abs(x - p.getX());  
312         int diffY = Math.abs(y - p.getY());  
313         return this.equals(p) || (diffX <= tolerance  
314         && diffY <= tolerance);  
315     }  
316  
317     class Snake {  
318         private Direction direction;  
319         private Point head;  
320         private ArrayList<Point> tail;  
321  
322         public Snake(int x, int y) {  
323             this.head = new Point(x, y);  
324             this.direction = Direction.RIGHT;  
325             this.tail = new ArrayList<Point>();  
326  
327             this.tail.add(new Point(0, 0));  
328             this.tail.add(new Point(0, 0));  
329             this.tail.add(new Point(0, 0));  
330         }  
331  
332         public void move() {  
333             ArrayList<Point> newTail = new ArrayList<  
334             Point>();
```

```
335         for (int i = 0, size = tail.size(); i < size
; i++) {
336             Point previous = i == 0 ? head : tail.
get(i - 1);
337
338             newTail.add(new Point(previous.getX(),
previous.getY()));
339         }
340
341         this.tail = newTail;
342
343         this.head.move(this.direction, 10);
344     }
345
346     public void addTail() {
347         this.tail.add(new Point(-10, -10));
348     }
349
350     public void turn(Direction d) {
351         if (d.isX() && direction.isY() || d.isY
() && direction.isX()) {
352             direction = d;
353         }
354     }
355
356     public ArrayList<Point> getTail() {
357         return this.tail;
358     }
359
360     public Point getHead() {
361         return this.head;
362     }
363 }
364
365 public class Main extends JFrame {
366     public Main() {
367         initUI();
368     }
369
370     private void initUI() {
371         add(new Game());
```

```
372
373         setTitle("Snake");
374         setSize(800, 610);
375
376         setLocationRelativeTo(null);
377         setResizable(false);
378         setDefaultCloseOperation(JFrame.
    EXIT_ON_CLOSE);
379     }
380
381     public static void main(String[] args) {
382         EventQueue.invokeLater(() -> {
383             Main ex = new Main();
384             ex.setVisible(true);
385         });
386     }
387 }
```