```
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 {
       private Timer timer;
11
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
       private static Font FONT_M = new Font("MV Boli",
31
   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);
       private static int WIDTH = 760;
35
       private static int HEIGHT = 520;
36
37
       private static int DELAY = 50;
```

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

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

```
|| head.getX() >= WIDTH + 10
115
                     || head.getY() <= 40
116
117
                     || head.getY() >= HEIGHT + 30;
118
119
            boolean ateItself = false;
120
121
            for(Point t : snake.getTail()) {
                ateItself = ateItself || head.equals(t);
122
123
            }
124
125
            if (hitBoundary || ateItself) {
                setStatus(GameStatus.GAME_OVER);
126
127
            }
128
        }
129
130
        // Spawn a cherry at a random location
131
        public void drawCenteredString(Graphics g,
    String text, Font font, int y) {
            FontMetrics metrics = g.getFontMetrics(font
132
    );
133
            int x = (WIDTH - metrics.stringWidth(text
    )) / 2;
134
135
            q.setFont(font);
            q.drawString(text, x, y);
136
        }
137
138
139
        private void render(Graphics q) {
140
            Graphics2D g2d = (Graphics2D) g;
141
142
            g2d.setColor(Color.BLACK);
143
            q2d.setFont(FONT_M);
144
145
            if (status == GameStatus.NOT_STARTED) {
146
                drawCenteredString(g2d, "SNAKE", FONT_XL
      200);
147
                drawCenteredString(q2d, "GAME", FONT_XL
    , 300);
148
                drawCenteredString(g2d, "Press
                                                  any
                                                       key
          begin", FONT_M_ITALIC, 330);
      to
149
```

```
150
                return;
151
            }
152
153
            Point p = snake.getHead();
154
155
            g2d.drawString("SCORE: " + String.format ("%
    02d", points), 20, 30);
            g2d.drawString("BEST: " + String.format ("%
156
    02d", best), 630, 30);
157
            if (cherry != null) {
158
159
                if (didLoadCherryImage) {
160
                    g2d.drawImage(image, cherry.getX(),
    cherry.getY(), 60, 60, null);
161
                } else {
162
                    q2d.setColor(Color.BLACK);
163
                    g2d.fillOval(cherry.getX(), cherry.
    qetY(), 10, 10);
164
                    q2d.setColor(Color.BLACK);
165
                }
166
            }
167
168
            if (status == GameStatus.GAME_OVER) {
                drawCenteredString(g2d, "Press
169
               again", FONT_M_ITALIC, 330);
    to
        start
                drawCenteredString(g2d, "GAME OVER",
170
    FONT_L, 300);
            }
171
172
173
            if (status == GameStatus.PAUSED) {
                g2d.drawString("Paused", 600, 14);
174
175
            }
176
            q2d.setColor(new Color(33, 70, 199));
177
            q2d.fillRect(p.getX(), p.getY(), 10, 10);
178
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
            q2d.setColor(Color.RED);
            g2d.setStroke(new BasicStroke(4));
187
            g2d.drawRect(20, 40, WIDTH, HEIGHT);
188
189
        }
190
191
        // spawn cherry in random position
192
        public void spawnCherry() {
            cherry = new Point((new Random()).nextInt(
193
    WIDTH -60) + 20,
194
                     (new Random()).nextInt(HEIGHT - 60
    ) + 40);
195
        }
196
197
        // game loop
        private class KeyListener extends KeyAdapter {
198
199
            @Override
            public void keyPressed(KeyEvent e) {
200
201
                int key = e.getKeyCode();
202
203
                if (status == GameStatus.RUNNING) {
204
                     switch(key) {
205
                         case KeyEvent.VK_LEFT: snake.
    turn(Direction.LEFT); break;
206
                         case KeyEvent.VK_RIGHT: snake.
    turn(Direction.RIGHT); break;
207
                         case KeyEvent.VK_UP: snake.turn(
    Direction.UP); break;
208
                         case KeyEvent.VK_DOWN: snake.
    turn(Direction.DOWN); break;
209
                     }
210
                }
211
212
                if (status == GameStatus.NOT_STARTED) {
213
                     setStatus(GameStatus.RUNNING);
                }
214
215
216
                if (status == GameStatus.GAME_OVER &&
    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() {
                update();
228
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 {
        UP, DOWN, LEFT, RIGHT;
242
243
        public boolean isX() {
244
245
            return this == LEFT || this == RIGHT;
246
        }
247
        public boolean isY() {
248
            return this == UP || this == DOWN;
249
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.qetY();
266
        }
267
268
        public void move(Direction d, int value) {
            switch(d) {
269
                case UP: this.y -= value; break;
270
271
                 case DOWN: this.y += value; break;
272
                 case RIGHT: this.x += value; break;
                case LEFT: this.x -= value; break;
273
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
        public boolean equals(Point p) {
297
```

```
298
            return this.x == p.qetX() && this.y == p.
    qetY();
299
        }
300
301
        public String toString() {
            return "(" + x + ", " + y + ")";
302
303
        }
304
305
        public boolean intersects(Point p) {
            return intersects(p, 10);
306
307
        }
308
309
        public boolean intersects(Point p, int tolerance
    ) {
310
            int diffX = Math.abs(x - p.getX());
            int diffY = Math.abs(y - p.getY());
311
312
313
            return this.equals(p) || (diffX <= tolerance</pre>
     && diffY <= tolerance);
        }
314
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));
            this.tail.add(new Point(0, 0));
329
330
        }
331
332
        public void move() {
333
            ArrayList<Point> newTail = new ArrayList<</pre>
    Point>();
334
```

```
for (int i = 0, size = tail.size(); i < size</pre>
335
    ; i++) {
336
                Point previous = i == 0 ? head : tail.
    qet(i - 1);
337
                newTail.add(new Point(previous.getX(),
338
    previous.getY());
339
            }
340
341
            this.tail = newTail;
342
            this.head.move(this.direction, 10);
343
344
        }
345
        public void addTail() {
346
347
            this.tail.add(new Point(-10, -10));
348
        }
349
        public void turn(Direction d) {
350
            if (d.isX() && direction.isY() || d.isY
351
    () && direction.isX()) {
                direction = d;
352
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");
            setSize(800, 610);
374
375
            setLocationRelativeTo(null);
376
            setResizable(false);
377
378
            setDefaultCloseOperation(JFrame.
    EXIT_ON_CLOSE);
        }
379
380
        public static void main(String[] args) {
381
            EventQueue.invokeLater(() -> {
382
                Main ex = new Main();
383
384
                ex.setVisible(true);
385
            });
386
        }
387 }
```