```
00001: package hevs.fragil.patapon.drawables;
00002:
00003: import java.util.Random;
00004:
00005: import ch.hevs.gdx2d.components.geometry.Point;
00006: import ch.hevs.gdx2d.components.graphics.Turtle;
00007: import ch.hevs.gdx2d.lib.GdxGraphics;
00008: import hevs.fragil.patapon.mechanics.Param;
00009:
00010: /**
00011:  * Recursive random hexagonal tree using a Logo-like drawing utility called Turtle.
00012:  */
00013: public class Tree implements VisibleObject {
00014:
00015:     private Random r;
00016:     private long seed;
00017:     private int time = 0;
00018:     private int complexity;
00019:     private Point<Float> location;
00020:     private Turtle t;
00021:     private float size;
00022:     private int width;
00023:
00024:     public Tree(Point<Float> pos, int complexity, float size, int width) {
00025:         this.location = pos;
00026:         this.seed = (long) (Math.random() * 1000);
00027:         this.r = new Random(seed);
00028:         this.complexity = complexity;
00029:         this.size = size;
00030:         this.width = width;
00031:     }
00032:
00033:     /**
```

```
00034:      * Draws a line {@code length} long
00035:      *
00036:      * @param length
00037:      *            length of the branch
00038:      * @author loicg
00039:      */
00040:     private void drawDoubleLine(double length, double width) {
00041:         // draw extern line
00042:         t.turn(-90);
00043:         t.penUp();
00044:         t.forward(width);
00045:         t.turn(90);
00046:         t.penDown();
00047:         t.forward(length);
00048:         double oldAngle = t.getTurtleAngle();
00049:         t.penUp();
00050:         t.turn(90);
00051:         t.forward(width);
00052:         Point<Float> oldPos = t.getPosition();
00053:         t.forward(width);
00054:
00055:         // draw intern line
00056:         t.turn(90);
00057:         t.penDown();
00058:         t.forward(length);
00059:         // return to old pos
00060:         t.jump(oldPos.x, oldPos.y);
00061:         t.setAngle(oldAngle);
00062:         t.penUp();
00063:     }
00064:
00065:     private void drawLine(double length, double width) {
00066:         if (r.nextDouble() > 0.5)
```

```
00067:             drawDoubleLine(length, width);
00068:         else {
00069:             t.penDown();
00070:             t.forward(length);
00071:         }
00072:         t.penUp();
00073:     }
00074:
00075:     private void drawHexaBranch(double length, double width) {
00076:         // hexa node
00077:         if (r.nextDouble() > 0.2) {
00078:             drawLine(length / 3, width);
00079:             drawHexagon(length / 6);
00080:             drawLine(length / 3, width);
00081:         } else {
00082:             drawLine(length / 3, width);
00083:         }
00084:     }
00085:
00086:     /**
00087:      * Verifies {@code value} is between {@code min} and {@code max}, then
00088:      * change it if out of range
00089:      *
00090:      * @param value
00091:      *            the value to limit
00092:      * @param min
00093:      *            minimum value for {@code value}
00094:      * @param max
00095:      *            maximum value for {@code value}
00096:      * @author loicg
00097:      * @return
00098:      */
00099:     private double ensureRange(double value, double min, double max) {
```

```
00100:         return Math.min(Math.max(value, min), max);
00101:     }
00102:
00103:     private void drawHexaPart(double length) {
00104:         // draw extern line
00105:         t.forward(length);
00106:         Point<Float> oldPos = t.getPosition();
00107:         double oldAngle = t.getTurtleAngle();
00108:
00109:         t.penUp();
00110:         t.turn(120);
00111:         t.forward(length / 3);
00112:
00113:         // draw intern line
00114:         t.turn(60);
00115:         t.penDown();
00116:         t.forward(length * 2 / 3);
00117:         // return to old pos
00118:         t.jump(oldPos.x, oldPos.y);
00119:         t.setAngle(oldAngle);
00120:     }
00121:
00122:     private void drawSimpleHexagon(double length) {
00123:         t.penDown();
00124:         t.turn(-60);
00125:         for (int i = 0; i < 6; i++) {
00126:             t.forward(length);
00127:             ;
00128:             t.turn(60);
00129:         }
00130:         t.penUp();
00131:         t.turn(60);
00132:         t.forward(2 * length);
```

```
00133:     }
00134:
00135:     private void drawDoubleHexagon(double length) {
00136:         t.penDown();
00137:         t.turn(-60);
00138:         for (int i = 0; i < 6; i++) {
00139:             if (i % 2 == 0)
00140:                 drawHexaPart(length);
00141:             else
00142:                 t.forward(length);
00143:             ;
00144:             t.turn(60);
00145:         }
00146:         t.penUp();
00147:         t.turn(60);
00148:         t.forward(2 * length);
00149:     }
00150:
00151:     private void drawHexagon(double length) {
00152:         if (r.nextDouble() > 0.5)
00153:             drawDoubleHexagon(length);
00154:         else
00155:             drawSimpleHexagon(length);
00156:     }
00157:
00158:     /**
00159:      * Draws a random tree
00160:      *
00161:      * @param n
00162:      *            complexity of recursive function (n sub-calls)
00163:      * @param length
00164:      *            length of the first branch
00165:      * @param width
```

```
00166:      *            width of the first branch
00167:      * @param leavesColor
00168:      *            average color of the leaves
00169:      * @author loicg
00170:      */
00171:     private void drawHexaTree(int n, double length, int width) {
00172:         // basis width
00173:         t.setWidth(width);
00174:         int factor = (complexity - n) * 10;
00175:         if (n > 1) {
00176:             // draw basis
00177:             drawHexaBranch(length, width);
00178:
00179:             // save Y embranchement
00180:             Point<Float> oldPos = t.getPosition();
00181:             double oldAngle = t.getTurtleAngle();
00182:
00183:             // next left tree (with new random factors)
00184:             t.setAngle(oldAngle + 60 + (factor * Math.sin((time * 2 * Math.PI) / 360 + (r.nextDouble() * 10))));
00185:             double newLength = length * 2 / 3;
00186:             drawHexaTree(n - 1, newLength, (int) ensureRange(width * 2 / 3, 1, 10));
00187:
00188:             // return to old Y embranchement
00189:             t.jump(oldPos.x, oldPos.y);
00190:
00191:             // next right tree (with new random factors)
00192:             t.setAngle(oldAngle - 60 + (factor * Math.sin((time * 2 * Math.PI) / 360 + (r.nextDouble() * 10))));
00193:             drawHexaTree(n - 1, newLength, (int) ensureRange(width * 2 / 3, 1, 10));
00194:
00195:         } else
00196:             drawHexaBranch(length, width);
00197:     }
00198:
```

```
00199:     @Override
00200:     public void draw(GdxGraphics g) {
00201:         //Check if the trees are on screen
00202:         if(isVisible(g, location.x)){
00203:             if (t != null) {
00204:                 // for oscillation
00205:                 time += 4;
00206:                 t.jump(location.x, location.y);
00207:                 t.setAngle(90);
00208:                 // reset values (get the same tree as last one)
00209:                 r.setSeed(seed);
00210:                 drawHexaTree(complexity, size, width);
00211:             } else
00212:                 t = new Turtle(g, Param.CAM_WIDTH, Param.CAM_HEIGHT);
00213:
00214:         }
00215:     }
00216:
00217:     @Override
00218:     public boolean isVisible(GdxGraphics g, float objectPos) {
00219:         boolean visible;
00220:         float camPosX = g.getCamera().position.x;
00221:
00222:         if(objectPos < camPosX + Param.CAM_WIDTH && objectPos > camPosX - Param.CAM_WIDTH){
00223:             visible = true;
00224:         }
00225:         else
00226:             visible = false;
00227:
00228:         return visible;
00229:     }
00230: }
```