```
00001: package hevs.fragil.patapon.drawables;
00002:
00003: import java.util.Vector;
00004:
00005: import com.badlogic.gdx.graphics.Color;
00006: import com.badlogic.gdx.math.Vector3;
00007:
00008: import ch.hevs.gdx2d.components.geometry.Point;
00009: import ch.hevs.gdx2d.lib.GdxGraphics;
00010: import ch.hevs.gdx2d.lib.interfaces.DrawableObject;
00011: import hevs.fragil.patapon.mechanics.Param;
00012: import hevs.fragil.patapon.physics.BasicTower;
00013: import hevs.fragil.patapon.physics.HexaTower;
00014: import hevs.fragil.patapon.units.Company;
00015:
00016: /**
00017:  * Scenery class to manage every item drawn in the game window.
00018:  * Principally manages camera moves and dynamic visible elements drawing.
00019:  * For instance, this class creates its own elements given in the constructor.
00020:  * TODO This class should be able to read some map files and instantiate them.
00021:  */
00022: public class Scenery {
00023:     private int width;
00024:     private int height;
00025:     private int manualOffset;
00026:     private Color background;
00027:     private Vector3 camera = new Vector3();
00028:     public Vector<DrawableObject> toDraw = new Vector<DrawableObject>();
00029:
00030:     /**
00031:      * Scenery constructor
00032:      * @param w    define the map width
00033:      * @param h define the map height
```

```
00034:      * @param b define the map background color
00035:      */
00036:     public Scenery(int w, int h, Color b) {
00037:         this.width = w;
00038:         this.setBackground(b);
00039:         // Calculate a forest
00040:         Point<Float> origin = new Point<Float>(0f, (float) Param.FLOOR_DEPTH);
00041:         toDraw.add(new Clouds(100, 4));
00042:         toDraw.add(new Mountains(0, 2));
00043:         toDraw.add(new Mountains(3800, 2));
00044:         processForest(10, origin, 5, 200f, 5);
00045:         toDraw.add(new BasicTower(1500, 10));
00046:         toDraw.add(new BasicTower(2000, 10));
00047:         toDraw.add(new BasicTower(4000, 10));
00048:         toDraw.add(new HexaTower(5500, 10));
00049:     }
00050:
00051:     /**
00052:      * The camera follows the given company
00053:      * @param c1
00054:      * @return camera x position
00055:      */
00056:     public Vector3 cameraProcess(Company c1){
00057:         if(c1.isEmpty())
00058:             return new Vector3(0,0,0);
00059:
00060:         camera.x = c1.getPosition() + Param.CAM_OFFSET + manualOffset;
00061:         camera.y = 0;
00062:         camera.z = 0;
00063:
00064:         return camera;
00065:     }
00066:
```

```
00067:     /**
00068:      * The camera is placed depending of both companies positions with
00069:      * priority to the player company.
00070:      * @param c1
00071:      * @param c2
00072:      * @return camera's complete position (x, y, z)
00073:      */
00074:     public Vector3 cameraProcess(Company c1, Company c2) {
00075:         if(c1.isEmpty())
00076:             return cameraProcess(c2);
00077:         if(c2.isEmpty())
00078:             return cameraProcess(c1);
00079: 
00080:         // Process absolute distance
00081:         float x1 = c1.getPosition();
00082:         float x2 = c2.getPosition();
00083:         float absDistance = Math.abs(x2 - x1);
00084: 
00085:         // Camera always stick on the floor
00086:         camera.x = x1 + Param.CAM_OFFSET + manualOffset;
00087:         camera.y = 0;
00088: 
00089:         // When companies are not so far, camera will dezoom to show both
00090:         if(absDistance < Param.CAM_RANGE){
00091:             camera.z = 5;
00092:         }
00093: 
00094:         // When companies are close enough OR if enemies too far camera follow heroes
00095:         else if (absDistance < Param.CAM_WIDTH || absDistance > Param.CAM_RANGE) {
00096:             camera.z = 1;
00097:         }
00098: 
00099:         // Input invalid!
```

```
00100:         else {
00101:             camera.x = Param.CAM_OFFSET + manualOffset;
00102:             camera.z = 1;
00103:         }
00104:
00105:         // TODO work only with a return new Vector3?
00106:         return camera;
00107:     }
00108:
00109:     /**
00110:      * Process a simple tree at given position
00111:      * @param position
00112:      */
00113:     public void processTree(Point<Float> position) {
00114:         toDraw.add(new Tree(position, 3, 200f, 5));
00115:     }
00116:
00117:     /**
00118:      * Process a forest with following median parameters
00119:      * @param density        give the forest density
00120:      * @param origin        give the first tree position
00121:      * @param complexity    give the trees complexity
00122:      * @param size            give the trees size
00123:      * @param penWidth        give the pen width, so the tree basic width
00124:      */
00125:     public void processForest(int density, Point<Float> origin, int complexity, float size, int penWidth) {
00126:         float x = origin.x, y = origin.y;
00127:         float ratio = width/density;
00128:
00129:         for (int i = 0; i < density; i++) {
00130:             x += ratio;
00131:
00132:             // Add some "natural" rendering
```

```
00133:             float randomOffset = (float)(ratio*Math.random());
00134:             if(randomOffset < ratio/3 && randomOffset > 0){
00135:                 if(Math.random() > 0.5f)
00136:                     x += randomOffset;
00137:                 else
00138:                     x -= randomOffset;
00139:             }
00140:
00141:             toDraw.add(new Tree(new Point<Float>(x, y), complexity, size, penWidth));
00142:
00143:             randomOffset = 0;
00144:         }
00145:     }
00146:
00147:     /* Here are all the getters and setters */
00148:     public int getWidth() {
00149:         return width;
00150:     }
00151:
00152:     public void setWidth(int width) {
00153:         this.width = width;
00154:     }
00155:
00156:     public int getHeigth() {
00157:         return height;
00158:     }
00159:
00160:     public void setHeigth(int heigth) {
00161:         this.height = heigth;
00162:     }
00163:
00164:     public Color getBackground() {
00165:         return background;
```

```
00166:     }
00167:
00168:     public void setBackground(Color background) {
00169:         this.background = background;
00170:     }
00171:
00172:     public void draw(GdxGraphics g) {
00173:         for (DrawableObject d : toDraw) {
00174:             d.draw(g);
00175:         }
00176:     }
00177:
00178:     public int getManualOffset(){
00179:         return manualOffset;
00180:     }
00181:
00182:     public void setManualOffset(int newValue){
00183:         manualOffset = newValue;
00184:     }
00185:
00186:     /**
00187:      * Allow to move the camera manually.
00188:      * @param amountPixels : user given offset
00189:      */
00190:     public void addManualOffset(int amountPixels){
00191:         // Set a minimal value to camera (placed by center of window)
00192:         manualOffset += amountPixels;;
00193:     }
00194:
00195:     /**
00196:      * Automatic camera centering on player company by setting the camera offset to zero
00197:      */
00198:     public void centerCamera(){
```

```
00199:          manualOffset = 0;
00200:      }
00201: }
```