```
00001: package hevs.fragil.patapon.physics;
00002:
00003: import com.badlogic.gdx.math.Vector2;
00004:
00005: import ch.hevs.gdx2d.components.physics.primitives.PhysicsPolygon;
00006: import ch.hevs.gdx2d.components.physics.utils.PhysicsConstants;
00007: import ch.hevs.gdx2d.lib.GdxGraphics;
00008: import ch.hevs.gdx2d.lib.physics.AbstractPhysicsObject;
00009: import ch.hevs.gdx2d.lib.physics.PhysicsWorld;
00010: import hevs.fragil.patapon.mechanics.CurrentLevel;
00011:
00012: public abstract class Projectile extends PhysicsPolygon{
00013:     protected int startAngle;
00014:     protected boolean stuck = false;
00015:     protected float life = 1.0f;
00016:     protected float damage;
00017:
00018:     protected Projectile(Vector2 startPos, int startAngle, int collisionGroup, int distance, int damage, Vector2[] vertices, String name) {
00019:         super(name, startPos, vertices, 1f, 0f, 1f, true);
00020:         this.startAngle = startAngle;
00021:         this.damage = damage;
00022:
00023:         // air resistance
00024:         setBodyAngularDamping(15f);
00025:
00026:         // same negative index to disable collisions between projectiles
00027:         getBody().setBullet(true);
00028:         setCollisionGroup(collisionGroup);
00029:         enableCollisionListener();
00030:
00031:         applyBodyForceToCenter(processForce(startPos, distance), true);
00032:         if(distance<0)
00033:             startAngle = 180 - startAngle;
```

```
00034:
00035:         CurrentLevel.getLevel().add(this);
00036:     }
00037:     /**
00038:      * Compute the necessary force for the arrow to travel the {@code distance} parameter
00039:      * @param startPos : start position from the arrow (give essentially the y coordinate)
00040:      * @param Distance : the distance to travel
00041:      * @return the force to apply during 1 fps
00042:      */
00043:     private Vector2 processForce(Vector2 startPos, double distance) {
00044:         double startAngleRad = Math.toRadians(startAngle);
00045:         Vector2 g = PhysicsWorld.getInstance().getGravity();
00046:         double absDistance = Math.abs(distance);
00047:
00048:         double a = -g.y / 2 / Math.tan(startAngleRad);
00049:         double b = 0;
00050:         double c = -PhysicsConstants.PIXEL_TO_METERS * (startPos.y / Math.tan(startAngleRad) + absDistance);
00051:
00052:         double t1 = (-b - Math.sqrt(Math.pow(b, 2) - 4 * a * c)) / (2 * a);
00053:         double t2 = (-b + Math.sqrt(Math.pow(b, 2) - 4 * a * c)) / (2 * a);
00054:
00055:         double t = Math.max(t1, t2);
00056:
00057:         double vx = (PhysicsConstants.PIXEL_TO_METERS * absDistance / t);
00058:         double v = vx / Math.cos(startAngleRad);
00059:         double a0 = v * 60;
00060:         double f0 = a0 * getBodyMass();
00061:
00062:         Vector2 force = new Vector2();
00063:         force.x = (float) (f0 * Math.cos(startAngleRad));
00064:         force.y = (float) (f0 * Math.sin(startAngleRad));
00065:
00066:         if(distance < 0)
```

```
00067:             force.x = -force.x;
00068:
00069:         return force;
00070:     }
00071:
00072:     @Override
00073:     public void collision(AbstractPhysicsObject theOtherObject, float energy) {
00074:         //Change collision group to avoid undesired new connections
00075:         if(theOtherObject instanceof BodyPolygon && !stuck){
00076:             ((CollidedObject)theOtherObject).applyDamage(damage);
00077:             setCollisionGroup(((CollidedObject)theOtherObject).getCollisionGroup());
00078:         }
00079:         if(theOtherObject instanceof Tower && !stuck){
00080:             ((CollidedObject)theOtherObject).applyDamage(damage);
00081:             setCollisionGroup(((CollidedObject)theOtherObject).getCollisionGroup());
00082:         }
00083:         if(!(theOtherObject instanceof Projectile) && !stuck){
00084:             //Create a joint to stick to the other object if not already stuck nor hit an arrow
00085:             CurrentLevel.getLevel().createWeldJoint(new StickyInfo(this.getBody(), theOtherObject.getBody(), getSpike()));
00086:             stuck = true;
00087:         }
00088:     }
00089:
00090:     public Vector2 getSpike() {
00091:         Vector2 temp = getBodyWorldCenter();
00092:         double angle = getBodyAngle() + startAngle;
00093:         temp.add((float) (Math.cos(angle) * 28), (float) (Math.sin(angle) * 28));
00094:         return temp;
00095:     }
00096:
00097:     public void draw(GdxGraphics g){};
00098:
00099:     public abstract void step(float dt);
```

```
00100: 
00101:     public boolean shouldBeDestroyed() {
00102:         if (life <= 0)
00103:             return true;
00104:         return false;
00105:     }
00106:     /**
00107:      * Convert vertices to Vector2 array
00108:      * @param float[] array to convert
00109:      * @return Vector2[] converted array
00110:      */
00111:     protected static Vector2[] verticesToVector2(float[] vertices) {
00112:         if (vertices.length % 2 == 0) {
00113:             Vector2[] temp = new Vector2[vertices.length / 2];
00114:             int j = 0;
00115:             for (int i = 0; i < vertices.length / 2; i++) {
00116:                 temp[i] = new Vector2(vertices[j], vertices[++j]);
00117:                 j++;
00118:             }
00119:             return temp;
00120:         } else {
00121:             return null;
00122:         }
00123:     }
00124: }
```