```
00001: package hevs.fragil.patapon.mechanics;
00002:
00003: import java.util.Iterator;
00004: import java.util.Vector;
00005:
00006: import com.badlogic.gdx.Gdx;
00007: import com.badlogic.gdx.Input.Keys;
00008: import com.badlogic.gdx.graphics.Color;
00009: import com.badlogic.gdx.math.Vector3;
00010: import com.badlogic.gdx.physics.box2d.World;
00011: import com.badlogic.gdx.physics.box2d.joints.WeldJointDef;
00012:
00013: import ch.hevs.gdx2d.components.audio.SoundSample;
00014: import ch.hevs.gdx2d.components.physics.primitives.PhysicsPolygon;
00015: import ch.hevs.gdx2d.components.physics.utils.PhysicsConstants;
00016: import ch.hevs.gdx2d.components.screen_management.RenderingScreen;
00017: import ch.hevs.gdx2d.desktop.physics.DebugRenderer;
00018: import ch.hevs.gdx2d.lib.GdxGraphics;
00019: import ch.hevs.gdx2d.lib.interfaces.DrawableObject;
00020: import ch.hevs.gdx2d.lib.physics.PhysicsWorld;
00021: import hevs.fragil.patapon.drawables.Clouds;
00022: import hevs.fragil.patapon.drawables.Scenery;
00023: import hevs.fragil.patapon.drawables.Frame;
00024: import hevs.fragil.patapon.drawables.Mountains;
00025: import hevs.fragil.patapon.music.Drum;
00026: import hevs.fragil.patapon.music.Sequence;
00027: import hevs.fragil.patapon.physics.Floor;
00028: import hevs.fragil.patapon.physics.Fragment;
00029: import hevs.fragil.patapon.physics.Projectile;
00030: import hevs.fragil.patapon.physics.StickyInfo;
00031: import hevs.fragil.patapon.physics.Tower;
00032: import hevs.fragil.patapon.units.Company;
00033: import hevs.fragil.patapon.units.Section;
```

```
00034: import hevs.fragil.patapon.units.State;
00035: import hevs.fragil.patapon.units.Unit;
00036:
00037: /**
00038:  * Level class to instantiate a new Level. This contains its own
00039:  * Scenery and other elements particular to this level.
00040:  * For instance, the level is always the same.
00041:  * TODO This class should be able to read files and creates itself in function.
00042:  */
00043: public class Level extends RenderingScreen {
00044:     private Scenery scenery;
00045:     private Floor floor;
00046:     private Frame frame;
00047:     private Sequence sequence;
00048:     private SoundSample heNote, sNote, soNote, yesNote;
00049:     private SoundSample snap, track;
00050:     private Company enemies = new Company();
00051:
00052:     private boolean debugActive = false;
00053:     private MusicFlag snapState = MusicFlag.STOPPED;
00054:     private MusicFlag trackState = MusicFlag.TOPLAY;
00055:
00056:     DebugRenderer debugRenderer;
00057:
00058:     private Vector<Projectile> projectiles = new Vector<Projectile>();
00059:     private Vector<StickyInfo> toJoin = new Vector<StickyInfo>();
00060:     private Vector<PhysicsPolygon> toDisable = new Vector<PhysicsPolygon>();
00061:     private Vector<DrawableObject> toKill = new Vector<DrawableObject>();
00062:
00063:     private float stateTime;
00064:     public float sinceLastRythm;
00065:
00066:     private Vector3 camera;
```

```
00067:
00068:     // A world with gravity pointing down. Must be called!
00069:     World world = PhysicsWorld.getInstance();
00070:
00071:     public Level() {
00072:     }
00073:
00074:     public void add(Projectile o) {
00075:         projectiles.add(o);
00076:     }
00077:
00078:     @Override
00079:     public void dispose() {
00080:         super.dispose();
00081:         heNote.dispose();
00082:         sNote.dispose();
00083:         soNote.dispose();
00084:         yesNote.dispose();
00085:         track.dispose();
00086:         snap.dispose();
00087:     }
00088:
00089:     @Override
00090:     public void onInit() {
00091:         PhysicsWorld.getInstance();
00092:         CurrentLevel.setLevel(this);
00093:
00094:         scenery = new Scenery(Param.MAP_WIDTH, Param.CAM_HEIGHT, Param.BACKGROUND);
00095:         Mountains.loadFiles();
00096:         Clouds.loadFiles();
00097:
00098:         enemies.initEnnemies(2,0,0);
00099:
```

```
00100:         // Load the sound files
00101:         heNote = new SoundSample("data/music/HE.wav");
00102:         sNote = new SoundSample("data/music/S.wav");
00103:         soNote = new SoundSample("data/music/SO.wav");
00104:         yesNote = new SoundSample("data/music/YES.wav");
00105:         snap = new SoundSample("data/music/loop2.wav");
00106:         track = new SoundSample("data/music/loop1.wav");
00107:
00108:         // Create a default map and the floor that belong
00109:         frame = new Frame();
00110:         floor = new Floor(scenery.getWidth());
00111:         sequence = new Sequence();
00112:         Sequence.loadSprites("data/images/drums102x102.png");
00113:         SequenceTimer.loadFiles();
00114:
00115:         debugRenderer = new DebugRenderer();
00116:     }
00117:
00118:     public void onKeyDown(int keycode) {
00119:         if (keycode == Keys.NUM_1) {
00120:             heNote.play();
00121:             State toDo = sequence.add(Drum.HE, sinceLastRythm);
00122:             PlayerCompany.getCompany().setAction(toDo);
00123:         }
00124:         if (keycode == Keys.NUM_2) {
00125:             sNote.play();
00126:             PlayerCompany.getCompany().setAction(sequence.add(Drum.S, sinceLastRythm));
00127:         }
00128:         if (keycode == Keys.NUM_3) {
00129:             soNote.play();
00130:             PlayerCompany.getCompany().setAction(sequence.add(Drum.SO, sinceLastRythm));
00131:         }
00132:         if (keycode == Keys.NUM_4) {
```

```
            yesNote.play();
            PlayerCompany.getCompany().setAction(sequence.add(Drum.YES, sinceLastRythm));
        }
        if (keycode == Keys.A) {
            PlayerCompany.getCompany().setAction(State.ATTACK);
        }
        if (keycode == Keys.M) {
            PlayerCompany.getCompany().setAction(State.WALK);
        }
        if (keycode == Keys.R) {
            PlayerCompany.getCompany().setAction(State.RETREAT);
        }
        if (keycode == Keys.D) {
            debugActive = !debugActive;
        }
        if (keycode == Keys.S) {
            switch (snapState) {
            case STOPPED:
                snapState = MusicFlag.TOPLAY;
                break;
            case PLAYING:
                snapState = MusicFlag.TOSTOP;
                break;
            default:
                break;
            }
        }
        if (keycode == Keys.T) {
            switch (trackState) {
            case STOPPED:
                trackState = MusicFlag.TOPLAY;
                break;
            case PLAYING:
```

```
00166:                 trackState = MusicFlag.TOSTOP;
00167:                 break;
00168:             default:
00169:                 break;
00170:             }
00171:         }
00172:
00173:         // Some manual actions to camera
00174:         if (keycode == Keys.LEFT) {
00175:             scenery.addManualOffset(-500);
00176:         }
00177:         if (keycode == Keys.RIGHT) {
00178:             scenery.addManualOffset(500);
00179:         }
00180:         if (keycode == Keys.CONTROL_RIGHT){
00181:             scenery.centerCamera();
00182:         }
00183:
00184:         if (keycode == Keys.ESCAPE) {
00185:             dispose();
00186:             System.exit(0);
00187:         }
00188:     }
00189:
00190:     public void onGraphicRender(GdxGraphics g) {
00191:         PhysicsWorld.updatePhysics(Gdx.graphics.getDeltaTime());
00192:
00193:         // process camera position inside map limits
00194:         camera = scenery.cameraProcess(PlayerCompany.getCompany(), enemies);
00195:
00196:         // apply camera position
00197:         //TODO play with scale to play with zoom :D enjoy your pain
00198:         g.moveCamera(camera.x, 0, Param.MAP_WIDTH, Param.MAP_HEIGHT);
```

```
00199:
00200:         if (debugActive) {
00201:             g.clear();
00202:             debugRenderer.render(PhysicsWorld.getInstance(), g.getCamera().combined);
00203:         }
00204:         else {
00205:             // clear the screen with the decor background
00206:             g.clear(scenery.getBackground());
00207:         }
00208:
00209:         // stick flying objects
00210:         createJoints();
00211:
00212:         // update objects
00213:         stepProjectiles(g);
00214:         stepFragments();
00215:         rythm();
00216:         action();
00217:         sequence.step();
00218:         killUnits();
00219:         destroyObjects();
00220:
00221:         if(!debugActive){
00222:
00223:             // display objects
00224:             floor.draw(g);
00225:             scenery.draw(g);
00226:             frame.draw(g);
00227:             sequence.draw(g);
00228:             PlayerCompany.getCompany().draw(g);
00229:             enemies.draw(g);
00230:             drawProjectiles(g);
00231:
```

```
00232:             // display help
00233:             g.setColor(Color.BLACK);
00234:             g.drawString(g.getCamera().position.x-700, 870, "Sequence Walk : He He He S");
00235:             g.drawString(g.getCamera().position.x-700, 850, "Sequence Attack : S S He S");
00236:             g.drawString(g.getCamera().position.x-700, 830, "Sequence Retreat : S He S He");
00237:             g.drawString(g.getCamera().position.x-700, 780, "Fever : " + sequence.getFever());
00238:             g.drawStringCentered(760, "T to disable/enable track");
00239:             g.drawStringCentered(740, "S to disable/enable snap");
00240:             g.drawStringCentered(720, "D to disable/enable debug mode");
00241:             g.drawStringCentered(700, "Num(1, 2, 3, 4) = Note(He, S, So, Yes)");
00242:             g.drawStringCentered(680, "Use A, M, R for Attack, Walk, Retreat and arrows to move camera");
00243:             g.drawStringCentered(660, "The hexagonal tower symbolize the level end, destroy it to 'win' ");
00244:         }
00245:         stateTime += Gdx.graphics.getDeltaTime();
00246:     }
00247:
00248:     private void stepFragments() {
00249:         Vector<DrawableObject> toDestroy = new Vector<DrawableObject>();
00250:
00251:         for (DrawableObject d : scenery.toDraw) {
00252:             if(d instanceof Fragment){
00253:                 if(((Fragment)d).step()){
00254:                     toDestroy.add(d);
00255:                 }
00256:             }
00257:
00258:         }
00259:
00260:         for (DrawableObject d : toDestroy) {
00261:             ((Fragment)d).destroy();
00262:             scenery.toDraw.remove(d);
00263:         }
00264:
```

```
00265:         toDestroy.removeAllElements();
00266: 
00267:     }
00268: 
00269:     private void destroyObjects() {
00270:         Vector<DrawableObject> toDestroy = new Vector<DrawableObject>();
00271:         Vector<DrawableObject> newFragments = new Vector<DrawableObject>();
00272:         for (DrawableObject d : scenery.toDraw) {
00273:             int h = 0;
00274:             int x = 0;
00275:             if(d instanceof Tower){
00276:                 if(((Tower)d).isExploded()){
00277:                     x = ((Tower)d).getPos();
00278:                     h = ((Tower)d).getHeight();
00279:                     ((Tower)d).destroy();
00280:                     toDestroy.add(d);
00281:                 }
00282:                 //create h/20 lines of 5 bricks
00283:                 for(int i = 0 ; i < h ; i++){
00284:                     for(int j = 0 ; j < 5 ; j++){
00285:                         newFragments.add(new Fragment(x - 50 + j*20, Param.FLOOR_DEPTH + i*20, 20, 20));
00286:                     }
00287:                 }
00288:             }
00289:         }
00290:         for (DrawableObject d : toDestroy) {
00291:             scenery.toDraw.remove(d);
00292:         }
00293:         scenery.toDraw.addAll(newFragments);
00294:         toDestroy.removeAllElements();
00295:     }
00296: 
00297:     private void killUnits() {
```

```
00298:         //Remove heroes
00299:         Company c = PlayerCompany.getCompany();
00300:         for (Section s : c.sections) {
00301:             for (Unit u : s.units) {
00302:                 if (u.isDead()) {
00303:                     toKill.add(u);
00304:                 }
00305:             }
00306:             //remove the section if all of its units are killed
00307:             if(toKill.containsAll(s.units)){
00308:                 toKill.add(s);
00309:             }
00310:         }
00311:         //remove every object to kill
00312:         for (Object o : toKill) {
00313:             if(o instanceof Unit){
00314:                 ((Unit) o).destroyBox();
00315:                 for (Section s : c.sections) {
00316:                     if (s.units.contains(o)) {
00317:                         s.units.remove(o);
00318:                     }
00319:                 }
00320:             }
00321:             else if(o instanceof Section){
00322:                 c.remove((Section) o);
00323:             }
00324:         }
00325:         toKill.removeAllElements();
00326:
00327:         //Do the same to remove enemies
00328:         c = enemies;
00329:         for (Section s : c.sections) {
00330:             for (Unit u : s.units) {
```

```
00331:                 if (u.isDead()) {
00332:                     toKill.add(u);
00333:                 }
00334:             }
00335:             if(toKill.containsAll(s.units)){
00336:                 toKill.add(s);
00337:             }
00338:         }
00339:         for (Object o : toKill) {
00340:             if(o instanceof Unit){
00341:                 ((Unit) o).destroyBox();
00342:                 for (Section s : c.sections) {
00343:                     if (s.units.contains(o)) {
00344:                         s.units.remove(o);
00345:                     }
00346:                 }
00347:             }
00348:             else if(o instanceof Section){
00349:                 c.remove((Section) o);
00350:             }
00351:         }
00352:         toKill.removeAllElements();
00353:     }
00354:
00355:     public void createWeldJoint(StickyInfo si) {
00356:         toJoin.add(si);
00357:     }
00358:
00359:     public void disable(PhysicsPolygon p) {
00360:         toDisable.add(p);
00361:     }
00362:
00363:     private void rythm() {
```

```
00364:         sinceLastRythm += Gdx.graphics.getDeltaTime();
00365:
00366:         if (sinceLastRythm >= 0.5) {
00367:             // every 500ms
00368:             changeTrack();
00369:             sinceLastRythm -= 0.5f;
00370:             frame.toggle();
00371:         }
00372:     }
00373:
00374:     private void changeTrack() {
00375:         switch (trackState) {
00376:         case TOSTOP:
00377:             track.stop();
00378:             trackState = MusicFlag.STOPPED;
00379:             break;
00380:         case TOPLAY:
00381:             track.loop();
00382:             trackState = MusicFlag.PLAYING;
00383:             break;
00384:         default:
00385:             break;
00386:         }
00387:         switch (snapState) {
00388:         case TOSTOP:
00389:             snap.stop();
00390:             snapState = MusicFlag.STOPPED;
00391:             break;
00392:         case TOPLAY:
00393:             snap.loop();
00394:             snapState = MusicFlag.PLAYING;
00395:             break;
00396:         default:
```

```
00397:             break;
00398:         }
00399:     }
00400:
00401:     private void action() {
00402:         SequenceTimer.run(PlayerCompany.getCompany(), sequence.getFever());
00403:         EnemiesTimer.run(enemies);
00404:     }
00405:
00406:     public Company getEnemies() {
00407:         return enemies;
00408:     }
00409:
00410:     private void createJoints() {
00411:         while (toJoin.size() > 0) {
00412:             // get last element and delete it
00413:             StickyInfo si = toJoin.remove(0);
00414:             // create a new joint
00415:             WeldJointDef wjd = new WeldJointDef();
00416:             wjd.bodyA = si.bodyA;
00417:             wjd.bodyB = si.bodyB;
00418:             wjd.referenceAngle = si.bodyB.getAngle() - si.bodyA.getAngle();
00419:             wjd.initialize(si.bodyA, si.bodyB, PhysicsConstants.coordPixelsToMeters(si.anchor));
00420:             PhysicsWorld.getInstance().createJoint(wjd);
00421:         }
00422:     }
00423:
00424:     private void stepProjectiles(GdxGraphics g) {
00425:         // Should be used like that
00426:         for (Iterator<Projectile> iter = projectiles.iterator(); iter.hasNext();) {
00427:             Projectile projectile = iter.next();
00428:
00429:             projectile.step(Gdx.graphics.getDeltaTime());
```

```
00430:
00431:             // If a ball is not visible anymore, it should be destroyed
00432:             if (projectile.shouldBeDestroyed()) {
00433:                 // Mark the ball for deletion when possible
00434:                 projectile.destroy();
00435:                 // Remove the ball from the collection as well
00436:                 iter.remove();
00437:             }
00438:         }
00439:     }
00440:     private void drawProjectiles(GdxGraphics g){
00441:         for (Iterator<Projectile> iter = projectiles.iterator(); iter.hasNext();) {
00442:             Projectile projectile = iter.next();
00443:
00444:             projectile.draw(g);
00445:
00446:             // If a ball is not visible anymore, it should be destroyed
00447:             if (projectile.shouldBeDestroyed()) {
00448:                 // Mark the ball for deletion when possible
00449:                 projectile.destroy();
00450:                 // Remove the ball from the collection as well
00451:                 iter.remove();
00452:             }
00453:         }
00454:     }
00455:
00456:     public float getStateTime() {
00457:         return stateTime;
00458:     }
00459:
00460:     public Scenery getDecor() {
00461:         return scenery;
00462:     }
```

```
00463: }
```