```
00001: package hevs.fragil.patapon.units;
00002:
00003: import java.util.Vector;
00004:
00005: import com.badlogic.gdx.Gdx;
00006: import com.badlogic.gdx.graphics.Color;
00007: import com.badlogic.gdx.math.Vector2;
00008:
00009: import ch.hevs.gdx2d.lib.GdxGraphics;
00010: import ch.hevs.gdx2d.lib.interfaces.DrawableObject;
00011: import hevs.fragil.patapon.mechanics.CurrentLevel;
00012: import hevs.fragil.patapon.mechanics.Param;
00013: import hevs.fragil.patapon.mechanics.PlayerCompany;
00014: import hevs.fragil.patapon.physics.BodyPolygon;
00015: import hevs.fragil.patapon.physics.Tower;
00016:
00017: public abstract class Unit implements DrawableObject {
00018:     protected boolean isEnemy;
00019:     private Species species = Species.TAPI;
00020:
00021:     protected Skills skills ;
00022:     protected UnitRender render ;
00023:
00024:     private float counter;
00025:
00026:     private int attackStep;
00027:     private int nAttacks;
00028:
00029:     protected int collisionGroup;
00030:     private BodyPolygon hitBox;
00031:
00032:     Unit(){
00033:
```

```
00034:     }
00035:     Unit(int lvl, Species s, int attack, int defense, int life, int distance, int rangeMin, int rangeMax, float cooldown, boolean isEnnemi) {
00036:         skills = new Skills(lvl, life, attack, rangeMin, rangeMax, defense, cooldown);
00037:         this.isEnemy = isEnnemi;
00038:         this.species = s;
00039:
00040:         render = new UnitRender(4 * species.ordinal() + lvl);
00041:         if (isEnnemi)
00042:             this.collisionGroup = Param.ENNEMIES_GROUP;
00043:         else
00044:             this.collisionGroup = Param.HEROES_GROUP;
00045:     }
00046:
00047:     public void setPosition(int newPos, double totalTime) {
00048:         if (hitBox != null){
00049:             if(!isDying()){
00050:                 hitBox.moveToLinear(newPos, totalTime);
00051:             }
00052:         }
00053:         else {
00054:             hitBox = new BodyPolygon(new Vector2(newPos, Param.FLOOR_DEPTH), collisionGroup, skills.getLife());
00055:             hitBox.getBody().setFixedRotation(true);
00056:         }
00057:     }
00058:
00059:     protected Vector2 getPosition() {
00060:         if (hitBox != null)
00061:             return hitBox.getBodyWorldCenter();
00062:         else
00063:             return new Vector2(0, 0);
00064:     }
00065:
00066:     protected void setLife(int life) {
```

```
00067:         this.skills.setLife(life);
00068:     }
00069:
00070:     protected abstract float getAttackDelay();
00071:     protected abstract Color getColor();
00072:
00073:     public void setState(State s) {
00074:         render.setState(s);
00075:     }
00076:
00077:     public void receive(float damage) {
00078:         if (damage > getDefense()){
00079:             if(this.hitBox.applyDamage(damage)){
00080:                 render.setState(State.DYING);
00081:             }
00082:         }
00083:     }
00084:
00085:     private float getDefense() {
00086:         if (render.getState() == State.DEFEND)
00087:             return skills.getDefense();
00088:         else
00089:             return 0;
00090:     }
00091:
00092:     @Override
00093:     public void draw(GdxGraphics g) {
00094:         float x = Math.round(getPosition().x - g.getCamera().position.x + Param.CAM_WIDTH / 2);
00095:         float y = Math.round(getPosition().y - g.getCamera().position.y + Param.CAM_HEIGHT / 2 - 37);
00096:         float angle = hitBox.getBodyAngle();
00097:
00098:         if (unitsInRange() || !getTowersInRange().isEmpty())
00099:             render.setLook(Look.ANGRY);
```

```
00100:         else if(unitsInSight()){
00101:             if(isEnemy) render.setLook(Look.LEFT);
00102:             else render.setLook(Look.RIGHT);
00103:         }
00104:         else
00105:             render.setLook(Look.DEFAULT);
00106:
00107:         render.draw(g,x,y,angle);
00108:
00109:         // Some debug info (display unit range)
00110: //        if(!isEnemy){
00111: //            g.drawFilledRectangle(x + skills.getRangeMin(), y, 10, 10, 0, getColor());
00112: //            g.drawFilledRectangle(x + skills.getRangeMax(), y, 10, 10, 0, getColor());
00113: //        }
00114:     }
00115:
00116:     public void setDelay(int delay) {
00117:         skills.setCooldown(delay);
00118:     }
00119:
00120:     public float getDelay() {
00121:         return skills.getCooldown();
00122:     }
00123:
00124:     public void setCollisionGroup(int group) {
00125:         collisionGroup = group;
00126:         hitBox.setCollisionGroup(group);
00127:     }
00128:
00129:     public float getLife() {
00130:         return hitBox.getLife();
00131:     }
00132:
```

```
00133:     public Skills getSkills() {
00134:         return skills;
00135:     }
00136:
00137:     public boolean isDead() {
00138:         if (getLife() <= 0) {
00139:             render.setState(State.DYING);
00140:             render.setLook(Look.DYING);
00141:             //decrease opacity until total disappear
00142:             return render.die();
00143:         }
00144:         else return false;
00145:     }
00146:
00147:     public boolean isDying() {
00148:         if (getLife() <= 0) {
00149:             return true;
00150:         }
00151:         else return false;
00152:     }
00153:
00154:     public void destroyBox() {
00155:         hitBox.destroy();
00156:     }
00157:
00158:     protected abstract void attack();
00159:
00160:     public void attackRoutine(){
00161:         float dt = Gdx.graphics.getDeltaTime();
00162:         counter += dt;
00163:
00164:         if(!isDying()){
00165:             if(unitsInRange() || !getTowersInRange().isEmpty()){
```

```
00166:                 //Sort of state machine (PATATE MACHINE)
00167:                 switch(attackStep){
00168:                 case 0 :
00169:                     if(counter >= getCooldown()){
00170:                         //is remaining time sufficient for another shoot ?
00171:                         if(nAttacks < (int)(2f / (getCooldown()+0.8f))){
00172:                             //end of cooldown, launch animation
00173:                             render.launch(Gesture.ATTACK);
00174:                             attackStep++;
00175:                             counter = 0;
00176:                         }
00177:                         //stuck in cooldown state until the end, when time insufficient
00178:                     }
00179:                     break;
00180:
00181:                 case 1 :
00182:                     if(counter >= getAttackDelay()){
00183:                         //animation pre shoot ended, shoot
00184:                         counter = 0;
00185:                         attack();
00186:                         nAttacks++;
00187:                         attackStep++;
00188:                     }
00189:                     break;
00190:
00191:                 case 2 :
00192:                     if(counter >= 0.8f - getAttackDelay()){
00193:                         //animation ended, retun to cooldown state
00194:                         counter = 0;
00195:                         attackStep = 0;
00196:                     }
00197:                     break;
00198:                 }
```

```
00199:             }
00200:         }
00201:     }
00202:
00203:     public void applyImpulse(int intensity) {
00204:         Vector2 pos = hitBox.getBodyPosition();
00205:         Vector2 force = new Vector2(intensity, 0);
00206:         hitBox.applyBodyLinearImpulse(force, pos, true);
00207:     }
00208:
00209:     public void setExpression(Look exp) {
00210:         render.setLook(exp);
00211:     }
00212:
00213:     public int getEndurance() {
00214:         int defense = 0;
00215:         if (render.getState() == State.DEFEND)
00216:             defense = skills.getDefense();
00217:         return skills.getLife() + defense;
00218:     }
00219:
00220:     public boolean isFatal(int damage) {
00221:         boolean fatal = false;
00222:         if (damage >= getEndurance())
00223:             fatal = true;
00224:         return fatal;
00225:     }
00226:
00227:     public Vector<Unit> getUnitsInRange() {
00228:         Vector<Unit> unitsInRange = new Vector<Unit>();
00229:         Company enemies;
00230:         if(isEnemy)enemies = PlayerCompany.getCompany();
00231:         else enemies = CurrentLevel.getLevel().getEnemies();
```

```
00232:
00233:         for (Section s : enemies.sections) {
00234:             for (Unit u : s.units) {
00235:                 float distance = u.getPosition().x - this.getPosition().x;
00236:                 // Subtraction of two half-sprite to find center2center distance
00237:                 distance = Math.abs(distance) - Param.UNIT_BODY_WIDTH;
00238:                 if (distance < skills.getRangeMax() && distance > skills.getRangeMin()) {
00239:                     unitsInRange.add(u);
00240:                 }
00241:             }
00242:         }
00243:         return unitsInRange;
00244:     }
00245:
00246:     protected boolean unitsInRange() {
00247:         if (getUnitsInRange().isEmpty()){
00248:             return false;
00249:         }
00250:         return true;
00251:     }
00252:
00253:     protected boolean unitsInSight() {
00254:         if (unitToEnemiDistance() < Param.SIGHT && unitToEnemiDistance()!=0) {
00255:             return true;
00256:         }
00257:         return false;
00258:     }
00259:
00260:     protected boolean unitsTooClose() {
00261:         if (unitToEnemiDistance() < skills.getRangeMin()) {
00262:             return true;
00263:         }
00264:         return false;
```

```
00265:     }
00266:
00267:     protected boolean unitsTooFar() {
00268:         if (unitToEnemiDistance() > skills.getRangeMax()) {
00269:             return true;
00270:         }
00271:         return false;
00272:     }
00273:
00274:     protected float unitToEnemiDistance() {
00275:         Company enemies;
00276:         if(isEnemy)enemies = PlayerCompany.getCompany();
00277:         else enemies = CurrentLevel.getLevel().getEnemies();
00278:
00279:         float distance = -1;
00280:         if(!enemies.isEmpty()){
00281:             for (Section s : enemies.sections) {
00282:                 for (Unit u : s.units) {
00283:                     if(distance > Math.abs(u.getPosition().x - getPosition().x) || distance == -1)
00284:                         distance = Math.abs(u.getPosition().x - getPosition().x);
00285:                 }
00286:             }
00287:             // Subtract 64 (2 half-sprite) to match range (0 to x, ...)
00288:             distance -= 64;
00289:             return distance;
00290:         }
00291:
00292:         return 0;
00293:     }
00294:
00295:     protected float unitToUnitDistance(Unit u1){
00296:         if(u1 != null)
00297:             return Math.abs(u1.getPosition().x - getPosition().x);
```

```
00298:         else
00299:             return 0;
00300:     }
00301:
00302:     /**
00303:      * Check if new position is available in company range
00304:      * @return true when alright
00305:      */
00306:     protected boolean unitInCompanyRange(){
00307:         float dt = Gdx.graphics.getDeltaTime();
00308:         int newPos;
00309:         Range companyRange = new Range(getPosition().x - Param.COMPANY_MARGIN, getPosition().x + Param.COMPANY_MARGIN);
00310:
00311:         // First, process new position
00312:         // Else if enemies are too close, set move to left
00313:         if(unitsTooClose() && !isEnemy || unitsTooFar() && isEnemy){
00314:             newPos = (int)(getPosition().x - Param.UNIT_SPEED * dt);
00315:         }
00316:         // Else if enemies too far, set move to right
00317:         else {
00318:             newPos = (int)(getPosition().x + Param.UNIT_SPEED * dt);
00319:         }
00320:
00321:         // Then if destination is in company range, do not move anymore
00322:         if(newPos > companyRange.getMax() && newPos < companyRange.getMin()){
00323:             return false;
00324:         }
00325:         // If unit is a NPC, it can move, else must wait until player action
00326:         else{
00327:             // Problem, company center follow unit, so how block player company and not enemies
00328:             return isEnemy? true : false;
00329:         }
00330:     }
```

```
00331:
00332:     public float getCooldown() {
00333:         return skills.getCooldown();
00334:     }
00335:
00336:     public float getCounter() {
00337:         return counter;
00338:     }
00339:
00340:     public String toString() {
00341:         return ", Level : " + skills.getLevel() + ", Life : " + skills.getLife();
00342:     }
00343:
00344:     /** This is only to load files in the PortableApplication onInit method */
00345:     public void setLegsSprite(String url, int cols, int rows, boolean isEnnemi) {
00346:         render.setLegsSprite(url, cols, rows, isEnnemi);
00347:     }
00348:
00349:     /** This is only to load files in the PortableApplication onInit method */
00350:     public void setBodySprite(String url, int cols, int rows) {
00351:         render.setBodySprite(url, cols, rows);
00352:     }
00353:
00354:     /** This is only to load files in the PortableApplication onInit method */
00355:     public void setEyeSprite(String url, int cols, int rows) {
00356:         render.setEyeSprite(url, cols, rows);
00357:     }
00358:
00359:     /** This is only to load files in the PortableApplication onInit method */
00360:     public void setArmsSprite(int cols, int rows, boolean isEnnemi) {
00361:         render.setArmsSprite(getUrl(), cols, rows, isEnnemi);
00362:     }
00363:
```

```
00364:     protected abstract String getUrl();
00365:     public void resetGesture() {
00366:         counter = 0;
00367:         nAttacks = 0;
00368:     }
00369:
00370:     public Unit findNextReachableEnemy() {
00371:         Company enemies;
00372:         if(isEnemy)enemies = PlayerCompany.getCompany();
00373:         else enemies = CurrentLevel.getLevel().getEnemies();
00374:
00375:         Unit nearest = null;
00376:         float rangeDistance = -1;
00377:         if(!enemies.isEmpty()){
00378:             for (Section s : enemies.sections) {
00379:                 for (Unit u : s.units) {
00380:                     if(Math.abs(u.getPosition().x - getPosition().x + getSkills().getRangeMax()) < rangeDistance || rangeDistance == -1){
00381:                         rangeDistance = Math.abs(u.getPosition().x - getSkills().getRangeMax());
00382:                         nearest = u;
00383:                     }
00384:                     if(Math.abs(getPosition().x + getSkills().getRangeMin() - u.getPosition().x) < rangeDistance){
00385:                         rangeDistance = Math.abs(getSkills().getRangeMin() - u.getPosition().x);
00386:                         nearest = u;
00387:                     }
00388:                 }
00389:             }
00390:         }
00391:         return nearest;
00392:     }
00393:     public int desiredPos(boolean increaseDistance) {
00394:         float dt = Gdx.graphics.getDeltaTime();
00395:         float desiredPos = getPosition().x;
00396:         if(increaseDistance){
```

```
00397:             if(isEnemy)
00398:                 desiredPos += Param.UNIT_SPEED * dt;
00399:             else
00400:                 desiredPos -= Param.UNIT_SPEED * dt;
00401:         }
00402:         else{
00403:             if(isEnemy)
00404:                 desiredPos -= Param.UNIT_SPEED * dt;
00405:             else
00406:                 desiredPos += Param.UNIT_SPEED * dt;
00407:         }
00408:
00409:         return (int)desiredPos;
00410:     }
00411:     public boolean isInRange(float u2uDistance) {
00412:         if(skills.getRangeMin() < u2uDistance && u2uDistance < skills.getRangeMax())
00413:             return true;
00414:         return false;
00415:     }
00416:     public Vector<Tower> getTowersInRange() {
00417:         Vector<Tower> towers = new Vector<Tower>();
00418:
00419:         if(!CurrentLevel.getLevel().getDecor().toDraw.isEmpty()){
00420:             for (DrawableObject d : CurrentLevel.getLevel().getDecor().toDraw) {
00421:                 if(d instanceof Tower){
00422:                     if(((Tower)d).getLeftLimit() < getSkills().getRangeMax() + getPosition().x){
00423:                         towers.add((Tower)d);
00424:                     }
00425:                 }
00426:             }
00427:         }
00428:         return towers;
00429:     }
```

```
00430: }
```