```
00001: package hevs.fragil.patapon.units;
00002: import java.util.Iterator;
00003: import java.util.Vector;
00004:
00005: import com.badlogic.gdx.Gdx;
00006: import com.badlogic.gdx.graphics.Color;
00007:
00008: import ch.hevs.gdx2d.lib.GdxGraphics;
00009: import ch.hevs.gdx2d.lib.interfaces.DrawableObject;
00010: import hevs.fragil.patapon.mechanics.CurrentLevel;
00011: import hevs.fragil.patapon.mechanics.Param;
00012: import hevs.fragil.patapon.physics.Arrow;
00013: import hevs.fragil.patapon.physics.Spear;
00014: import hevs.fragil.patapon.physics.Tower;
00015:
00016: public class Company implements DrawableObject {
00017:     public String name = "";
00018:     double feverFactor = 0.1;
00019:     public Vector<Section> sections = new Vector<Section>();
00020:     private State action;
00021:     private boolean ready;
00022:     private boolean freeToMove = false;
00023:     private int fixedPos;
00024:
00025:     public Company(){
00026:         this(0,"noname");
00027:     }
00028:     public Company(String name){
00029:         this(0,name);
00030:     }
00031:     public Company(int pos){
00032:         this(pos, "noname");
00033:     }
```

```
00034:     public Company(int pos, String name){
00035:         this.name = name;
00036:         this.ready = true;
00037:     }
00038:     public void setCollisionGroup(int collisionGroup){
00039:         for (Section s : sections) {
00040:             for (Unit u : s.units) {
00041:                 u.setCollisionGroup(collisionGroup);
00042:             }
00043:         }
00044:     }
00045:     public void setPosition(int newPos){
00046:         //move to the right, must check next position with right company limit
00047:         if(newPos > fixedPos){
00048:             if(areaClear(newPos + (Param.COMPANY_MARGIN +getMinWidth())/2)){
00049:                 fixedPos = newPos;
00050:             }
00051:         }
00052:         //move to the left, must check next position with left company limit
00053:         else if (areaClear(newPos - (Param.COMPANY_MARGIN +getMinWidth())/2))
00054:                 fixedPos = newPos;
00055:     }
00056:     private boolean areaClear(int posToTry) {
00057:         for (DrawableObject d : CurrentLevel.getLevel().getDecor().toDraw) {
00058:             if(d instanceof Tower){
00059:                 return !((Tower) d).isOccuped(posToTry);
00060:             }
00061:         }
00062:         return true;
00063:     }
00064:     public void setAction(State a){
00065:         if((ready && a != null) || a == State.IDLE){
00066:             action = a;
```

```
00067:             ready = false;
00068:             System.out.println("action : " + a + " set !");
00069:         }
00070:     }
00071:     public String toString(){
00072:         String t = "Start of company \n";
00073:         t += " This company is at position : "+ getPosition() +"\n";
00074:         t += " This company's fever factor : "+ feverFactor +"\n";
00075:         t += " This company contains : \n";
00076:         for (Section section : sections) {
00077:             t += section.toString()+"\n";
00078:         }
00079:         t += "End of Company";
00080:         return t;
00081:     }
00082:     public float getPosition(){
00083:         return fixedPos;
00084:     }
00085:     public int getNbUnits(){
00086:         int s = 0;
00087:         for (Iterator<Section> i = sections.iterator(); i.hasNext();) {
00088:             Section section = (Section) i.next();
00089:             s += section.units.size();
00090:         }
00091:         return s;
00092:     }
00093:     public int getNbSections(){
00094:         return sections.size();
00095:     }
00096:     public int getMinWidth(){
00097:         int width = 0;
00098:         for (Section section : sections) {
00099:             width += section.getWidth();
```

```
00100:         }
00101:         int nSections = sections.size();
00102:         return (int)(width + (nSections-1)*Param.SECTION_KEEPOUT);
00103:     }
00104:     public State getAction(){
00105:         return action;
00106:     }
00107:     public void add(Section s){
00108:         sections.addElement(s);
00109:     }
00110:     public void actionFinished(){
00111:         action = null;
00112:         ready = true;
00113:     }
00114:     public void remove(Section s){
00115:         sections.remove(s);
00116:     }
00117:     /**
00118:      * @param nb1 : number of archers
00119:      * @param nb2 : number of swordmans
00120:      * @param nb3 : number of shields
00121:      * @return a sample company that contains {@code nb1} archers,
00122:      * {@code nb2} swordmans and {@code nb3}shields.
00123:      */
00124:     public void initRandomHeroes(int nb1, int nb2, int nb3){
00125:         for(int i = 0 ; i < 3; i++){
00126:             add(new Section(Integer.toString(i)));
00127:         }
00128:         for(int i = 0 ; i < nb1; i++){
00129:             sections.elementAt(0).add(new Archer());
00130:         }
00131:         for(int i = 0 ; i < nb2; i++){
00132:             sections.elementAt(1).add(new Spearman());
```

```
00133:         }
00134:         for(int i = 0 ; i < nb3; i++){
00135:             sections.elementAt(2).add(new Shield());
00136:         }
00137:
00138:         int initialPos = getMinWidth()/2 + 50;
00139:
00140:         int width = getMinWidth();
00141:         float screenMargin = initialPos - width/2f;
00142:
00143:         if(screenMargin > 0){
00144:             float tempPos = screenMargin;
00145:             fixedPos = initialPos;
00146:             for (Section section : sections) {
00147:                 tempPos += section.getWidth()/2f;
00148:                 section.setPosition((int)tempPos, 0.1f);
00149:                 tempPos += section.getWidth()/2f + Param.SECTION_KEEPOUT;
00150:             }
00151:         }
00152:
00153:         //Load the image files
00154:         for (Section s : sections) {
00155:             for (Unit u : s.units) {
00156:                 u.setBodySprite("data/images/bodies.png", 5,5);
00157:                 u.setEyeSprite("data/images/eyes.png", 5, 2);
00158:                 u.setLegsSprite("data/images/legs.png", 4, 1, false);
00159:                 u.setArmsSprite(4, 6, false);
00160:             }
00161:         }
00162:
00163:         Arrow.setImgPath("data/images/fleche.png");
00164:         Spear.setImgPath("data/images/fleche.png");
00165:     }
```

```
00166:     public void initEnnemies(int nb1, int nb2, int nb3) {
00167:         for(int i = 0 ; i < 3; i++){
00168:             add(new Section(Integer.toString(i)));
00169:         }
00170:         for(int i = 0 ; i < nb3; i++){
00171:             sections.elementAt(0).add(new Shield(0,Species.TAPI,true));
00172:         }
00173:         for(int i = 0 ; i < nb2; i++){
00174:             sections.elementAt(1).add(new Spearman(0,Species.TAPI,true));
00175:         }
00176:         for(int i = 0 ; i < nb1; i++){
00177:             sections.elementAt(2).add(new Archer(0,Species.TAPI,true));
00178:         }
00179:
00180:         // Set the initial position
00181:         int initialPos = getMinWidth()/2 + 4000;
00182:
00183:         int width = getMinWidth();
00184:         float screenMargin = initialPos - width/2f;
00185:
00186:         if(screenMargin > 0){
00187:             float tempPos = screenMargin;
00188:             fixedPos = initialPos;
00189:             for (Section section : sections) {
00190:                 tempPos += section.getWidth()/2f;
00191:                 section.setPosition((int)tempPos, 0.1f);
00192:                 tempPos += section.getWidth()/2f + Param.SECTION_KEEPOUT;
00193:             }
00194:         }
00195:
00196:         //Load the image files
00197:         for (Section s : sections) {
00198:             for (Unit u : s.units) {
```

```
00199:                 u.setBodySprite("data/images/badbody.png", 1,1);
00200:                 u.setEyeSprite("data/images/badeyes.png", 5, 2);
00201:                 u.setLegsSprite("data/images/legs.png", 4, 1, true);
00202:                 u.setArmsSprite(4, 6, true);
00203:                 u.setExpression(Look.ANGRY);
00204:             }
00205:         }
00206:         Arrow.setImgPath("data/images/fleche.png");
00207:     }
00208:     @Override
00209:     public void draw(GdxGraphics g) {
00210:         for (Section section : sections) {
00211:             section.draw(g);
00212:         }
00213:     }
00214:     public void aiMove() {
00215:         if(freeToMove){
00216:             freeMove();
00217:         }
00218:         else{
00219:             regroup();
00220:         }
00221:     }
00222:
00223:     private void freeMove(){
00224:         for (Section s : sections) {
00225:             for (Unit u : s.units) {
00226:                 if(!u.isDying()){
00227:                     //when no enemy is in the units's range, unit must try to find a better place
00228:                     if(u.getUnitsInRange().isEmpty() && u.getTowersInRange().isEmpty()){
00229:                         float u2uDistance = u.unitToUnitDistance(u.findNextReachableEnemy());
00230:
00231:                         //if we are too near, we must increase the distance
```

```
00232:                         boolean increaseDistance = (u2uDistance < u.getSkills().getRangeMin());
00233:
00234:                         //get desired position depending of increase or decrease the distance with enemies
00235:                         int desiredPos = u.desiredPos(increaseDistance);
00236:
00237:                         //test if this new position is contained between the company maximum limits
00238:                         if(isInCompanyRange(desiredPos)){
00239:                             //if desiredPos is in company range and free from unit
00240:                             float distance = getNextUnitDistance();
00241:                             distance = Math.abs(distance);
00242:
00243:                             //avoid hidden units
00244:                             if(distance > Param.UNIT_BODY_WIDTH/2){
00245:                                 u.setPosition(u.desiredPos(false), Gdx.graphics.getDeltaTime());
00246:                             }
00247:                         }
00248:                     }
00249:                 }
00250:             }
00251:         }
00252:     }
00253:     private void regroup(){
00254:         float dt = Gdx.graphics.getDeltaTime();
00255:         for (Section s : sections) {
00256:             for (Unit u : s.units) {
00257:                 if(!u.isDying()){
00258:                     //get position in the perfect rank
00259:                     float desiredPos = u.getPosition().x;
00260:                     //move to the right
00261:                     if(u.getPosition().x < getOrderedPosition(u) - Param.UNIT_POSITION_TOLERANCE)
00262:                         desiredPos += Param.UNIT_SPEED * dt;
00263:                     //move to the left
00264:                     else if(u.getPosition().x > getOrderedPosition(u) + Param.UNIT_POSITION_TOLERANCE)
```

```
00265:                         desiredPos -= Param.UNIT_SPEED * dt;
00266:                     u.setPosition((int)desiredPos, dt);
00267:                 }
00268:             }
00269:         }
00270:     }
00271:     public void regroupUnits() {
00272:         freeToMove = false;
00273:     }
00274:     public void freeUnits(){
00275:         freeToMove = true;
00276:     }
00277:     private boolean isInCompanyRange(int desiredPos) {
00278:         if(fixedPos - (Param.COMPANY_MARGIN + getMinWidth())/2 < desiredPos && desiredPos < fixedPos + (Param.COMPANY_MARGIN + getMinWidth())/2)
00279:             return true;
00280:         return false;
00281:     }
00282:
00283:     private float getNextUnitDistance(){
00284:         float distance = 0;
00285:
00286:         for (Section s : sections) {
00287:             for (Unit u : s.units) {
00288:                 if(u.isEnemy){
00289:                     // Check next unit in section, if last of section, check next section
00290:                     if(!(s.units.firstElement() == u)){
00291:                         distance = s.units.elementAt(s.units.indexOf(u)-1).getPosition().x - u.getPosition().x;
00292:                     }
00293:                     else if(!(sections.firstElement() == s)){
00294:                         distance = sections.elementAt(sections.indexOf(s)-1).units.firstElement().getPosition().x;
00295:                         distance -= u.getPosition().x;
00296:                     }
00297:
```

```
00298:                 }
00299:                 else{
00300:                     // Check next unit in section, if last of section, check next section
00301:                     if(!(s.units.lastElement() == u)){
00302:                         distance = s.units.elementAt(s.units.indexOf(u)+1).getPosition().x - u.getPosition().x;
00303:                     }
00304:                     else if(!(sections.lastElement() == s)){
00305:                         distance = sections.elementAt(sections.indexOf(s)+1).units.firstElement().getPosition().x;
00306:                         distance -= u.getPosition().x;
00307:                     }
00308:                 }
00309:             }
00310:         }
00311:         distance = Math.abs(distance);
00312:
00313:         return distance;
00314:     }
00315:
00316:     /**
00317:      * Return ordered position of {@code unit}
00318:      * @param  unit : unit that must move
00319:      * @return desired position
00320:      */
00321:     private int getOrderedPosition(Unit unit) {
00322:         int index = 0;
00323:         int sectionNumber = 0;
00324:         for (Section s : sections) {
00325:             if(s.units.contains(unit)){
00326:                 index = s.units.indexOf(unit);
00327:                 break;
00328:             }
00329:             sectionNumber++;
00330:         }
```

```
00331:         int startPosition = fixedPos - getMinWidth() / 2;
00332:         int previousSectionsWidth = 0;
00333:         for (int i = 0 ; i < sectionNumber ; i++) {
00334:             previousSectionsWidth += sections.elementAt(i).getWidth() + Param.SECTION_KEEPOUT;
00335:         }
00336:         int orderedPos = startPosition + previousSectionsWidth + index * Param.UNIT_BODY_WIDTH;
00337:         return orderedPos;
00338:     }
00339:     public boolean isEmpty() {
00340:         return sections.isEmpty();
00341:     }
00342: }
```