Mar Moreno
Marina Castellano

INFORME PRÀCTICA 1: IMPLEMENTING A CLASS

En el laboratori 1 hem implementat el disseny que vam dur a termer en en el Seminari 1, on vam haver d'escriure els atributs i mètodes per a les classes AGENT i WORLD.
L'objectiu d'aquesta pràctica és implementar les dues classes, utilitzant les funcions respectives, per a fer que 10 agents es moguin de forma arbitraria dintre l'entorn de la simulació de l'espai world.

A continuació explicarem amb detall quins han estat els atributs i mètodes implementats per a cada classe.

1. WORLD

Per a la classe World, els atributs que hem implementat són: l'altura i l'amplada de l'entorn de la simulació de l'espai world, el array agents, l'atribut margin, que serveix perquè els agents no surtin del world, i el número d'agents.

```
private int width;
private int height;
private Agent[] agents;
private int margin;
private int numberA;
```

Mètodes:

Els mètodes que han estat implementats són: World, addAgent, simlutationStep i randomPos i randomRadius (venïen ja implementades i dites en l'enunciat).
Veiem que tots els mètodes són públics menys els de randomPos i randomRdius que són privats ja que només s'utilitzen en la classe world i no en les demés com els altres mètodes que els utilitzem en classes com WorldGUI i Agent.

Per implementar **World** hem utilitzat els paràmetres: width, height i cap (número d'agents = 10) i inicialitzem els paràmetres. Hem de tindre en conte que quan cridem a la funció World per assignar les dimensions d'aquest mètode, no podem utilitzar qualsevol numero ja que hem definit les variables com a enters, llavors no es podria executar.

Hem fet un **for** que recorre els agents i els hi hem assignat una posició, un radi, un target i una velocitat, és a dir, les funcions que hem implementat en la classe agent.

Després hem fet una funció per afegir agents:

```
public void add (Agent a){
      agents[this.numerA] = a;
}
```

I per últim, hem implementat simulationStep. Hem fet un **for** que itera mentre **i** sigui més petit que el número d'agents. SI l'agent arriba al seu target, li assignem un altre target, sinó li actualitzem la posició.

```
public void simulationStep(){
        for(int i = 0; i < numerA; i++){
            if(agents[i].reachedTarget()== true){
                agents[i].setTarget(randomPos());
            }else{
                agents[i].updatePosition();
            }

        }
```

Hem intentat fer la funció manageCollisions, però el resultat no ha estat l'esperat. No hem pogut hem pogut acabar de resoldre la funció.
El codi que hem implementat és el següent:

```
public void manageCollisions(){

        for(int i = 0; i < 10; i++ ){
            for(int j = 0; j < 10; j++){
               if(agents[i].isColliding(agents[j]) == true ){
                 agents[i].setTarget(randomPos());
                 agents[j].setTarget(randomPos());
            }
        }
    }
```

Després de llegir l'enunciat, hem decidit comprobar després dels dos fors, si els agents col·lisionen entre ells cridant al mètode de la classe Agent isColliding i comprovant si era true, per assignar-l'hi un nou objectiu amb el mètode setTarget i el randomPos. Seguidament hem executat el codi varies vegades i hem vist que alguns dels agents aconseguien xocar entre ells pero d'altres no, per tant hem vist que el codi pensat no ha sigut ben implementat.

2. AGENT

Per a la classe Agent, els atributs que hem implementat són: la posició, la direcció, el radi, el target i la velocitat.

```
private Vec2D pos;
private Vec2D dir ;
private double radius;
private Vec2D target;
private double speed;
```

Mètodes:

Primer hem implementat la funció Agent, i li hem passant una velocitat i un radi:

```
public Agent(Vec2D v,double r){
      pos = v;
      radius = r;
  }
```

Després hem fet la funció setTarget, on hem passat el paràmetre target i l'hem inicialitzat. Inicialitzem el vector direcció creant un nou vector amb el target de l'agent. Després restem la posició actual amb el vector direcció amb el mètode de java .subtract() i ha continuació la normalitzem amb .normalize() per obtenir la direcció actual de l'agent.

```
this.target = t;
dir = new Vec2D(target);
dir.subtract(pos);
dir.normalize()
```

A continuació hem fet la funció setSpeed, implementant `this.speed = s,`i la funció updatePosition, on hem creat un nou vector i l'hem sumat a la posició actual.

```
        Vec2D vector = new Vec2D(speed*dir.getX(), speed*dir.getY());
        pos.add(vector);
```

Per últim hem implementat les funcions reachedTarget i isColliding.
La funció reachedTarget comprova si un agent ha arribat al target.
Hem creat un nou vector, què és la diferència entre la target i la posició actual.

```
Vec2D vector2 = new
Vec2D(pos.getX()-target.getX(),pos.getY()-target.getY());
```

Comprovem si ha arribat al target. Fem un if/else, imposant com a condició que perquè l'agent arribi al target s'ha de complir que la longitud del vector sigui més petita que la del radi de l'agent.

```
        if(vector2.length() < radius){
            return true;
        }else{
            return false;
        }
```

Per fer el mètode isColliding() hem creat un nou vector amb la posició per després restar-la amb la posició de l'agent que hem introduït a la funció i comprovar que la longitud del nou vector sigui més petita que la suma dels dos radis, i si és així és que están col·lisionant. Hem pensat en utilitzar el vector "pos" en contés del vector "dir" peró hem vist que la millor opció era utilitzar el vector "dir", i que no afecta a la funció de isColliding.

```
public boolean isColliding(Agent a){

        Vec2D vector = new Vec2D(dir);
        vector.subtract(a.dir);
        if(vector.length() < radius + a.radius ){
            return true;
        }else {
            return false;
        }
    }
```

CONCLUSIÓ

Ara que hem explicat com hem implementat totes les classes, atributs i mètodes, executem el codi i se'ns mostra una pantalla on es troben 10 objectes vermells cilíndrics que es mouen en direccions aleatòries, amb radis diferents, que xoquen amb les parets assignades de l'espai i entre ells interseccionan.

Veïent la simulació resultat, creiem poder afirmar que les funcions de les classes han estat ben implementades, ja que els objectes es mouen com previst. Al veure la simulació podem observar que la majoria d'objectes no s'esquiven entre ells, degut a que no hem pogut resoldre la implementació del manageCollisions, com explicat anteriorment.