ISEL
INSTITUTO SUPERIOR DE ENGENHARIA DE LISBOA

Licenciatura em Engenharia Informática e Multimédia (LEIM)

Modelação e Simulação de Sistemas Naturais

2020/2021 – Semestre Inverno

Trabalho nº2

LEIM32D – 13 dezembro 2020

Luís Fonseca nº 45125

Paulo Jorge nº 45121

Índice

Índice de Figuras

Introdução

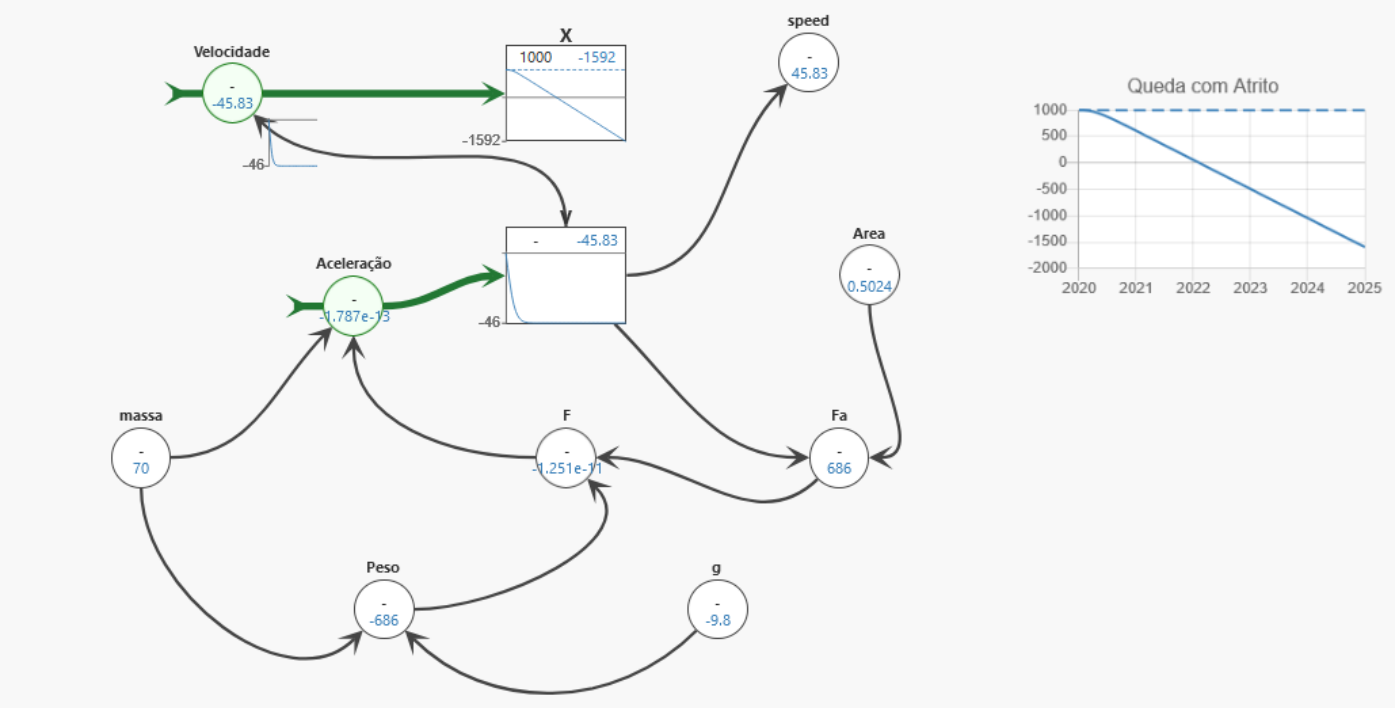
Neste trabalho de casa foi feito o estudo forças e construir modelos, usando o sheetless, acerca do tema em estudo. Com estes conceitos, foi proposto um trabalho de casa onde era posto em prática, a execução destes dois conceitos, pegando em vários exercícios e gerar diferentes resultados para eventuais estudos e conclusões.

Parte A - Modelação baseada em Agregados (Stocks & Flows)

Exercício escolhido: Para este exercício, foi criado um diagrama de modelação, simulando a queda de um corpo com atrito. Na figura debaixo é possível de visualizar, é possível de ver os diferentes stocks, flows e variáveis usadas para este exercício.

Para as variáveis, tivemos em conta o “Peso” e a “Massa” do corpo. Adicionamos o valor da aceleração gravítica, assim como uma força de atrito, e a área do corpo. Todos estes fatores contribuíram para que fosse estudado o tempo que um corpo chegasse ao chão.

No final deste exercício efetuamos um gráfico, para visualizar o acontecimento do corpo com todos estes fatores presentes.



ParteB – Sistema Solar e Sistema de Partículas

Neste exercício era pedido que fosse feita a representação de um sistema solar usando o processing, recorrendo às leis da física. Para isso foram construídas as classes: *CelestialBody*, *ParticleSystem*,*Particle*. Para testar o código criado, foi criada a classe *SolarSystemApp*.

Classe *CelestialBody:* é a classe que permite desenhar um corpo celestial, sendo este corpo um planeta, neste caso o Sol, Terra ou outro tipo de planeta. O código criado para fazer esta classe é o seguinte:

```
public class CelestialBody extends Body{
        public static final double G = 1e-4f;
        //public static final double G = 6.67e-11f;

        public CelestialBody(PVector pos, PVector vel, float mass,
                        float radius, int color){
                super(pos, vel, mass, radius, color);
        }

        public PVector attraction(Mover m){
            PVector r = PVector.sub(pos, m.pos);
            float dist = r.mag();
            float strength = (float)(G*mass*m.mass/Math.pow(dist, 2));
            return (r.normalize().mult(strength));
        }
}
```

O método *attraction(Mover m)* representa a lei de gravitação universal de Newton, na qual diz que, dois corpos atraem-se um ao outro com uma força diretamente proporcional ao produto da massa dos mesmos e inversamente proporcional ao quadrado da distância existente entre o centro destes corpos.

Classe *Particle:* representa uma partícula, sendo essa partícula de diferentes formas, nomeadamente pontos, triângulos, quadrados e retângulos. O código efetuado para o desenho de uma partícula é o seguinte:

```
public class Particle extends Mover {
        private float lifespan;
        private int color;
        private float timer;
        private int particleFormat;

        public Particle(PVector pos, PVector vel, int color,
                        float radius, float lifespan, int particleFormat) {
                super(pos, vel, 0f, radius);
                this.color = color;
                this.lifespan = lifespan;
                this.timer = 0;
                this.particleFormat = particleFormat;
        }

        @Override
        public void move(float dt)
        {
                super.move(dt);
                timer += dt;
        }

        public boolean isDead() {
                return (timer > lifespan);
        }

        public void display(PApplet parent, SubPlot plt) {
                parent.pushStyle();
                float alpha = PApplet.map(timer, 0, lifespan, 255, 0);
                parent.fill(color, alpha);
                parent.noStroke();
                switch(particleFormat) {
```

```
                case 0:
                        parent.stroke(parent.random(255),parent.random(255),parent.random(255));
                        parent.point(pos.x,pos.y);
                        break;
                case 1:
                        parent.ellipse(pos.x,pos.y,radius,radius);
                        break;
                case 2:
                        parent.rect(pos.x,pos.y,radius,radius);
                        break;
                case 3:
                        parent.triangle(pos.x, pos.y, pos.x +radius, pos.y+radius, pos.x - radius,
pos.y - radius);
                        break;
                }
                parent.popStyle();
        }
}
```

Classe *ParticleSystem*: representa um conjunto de partículas, é esta classe que permite fazer o desenho de cometas. O código criado encontra-se em baixo:

```
public class ParticleSystem extends Mover {
        private ArrayList<Particle> particles;
        private PVector particleSpeed;
        private int particleColor;
        private float lifetime;
        public PApplet parent;
        public ParticleSystem(PApplet parent,PVector pos, PVector vel, float mass,
                        PVector particleSpeed, int particleColor, float particleRadius,
                        float lifetime)
        {
                super(pos, vel, mass, particleRadius);
                this.particleSpeed = particleSpeed;
                this.particleColor = particleColor;
                this.lifetime = lifetime;
                this.particles = new ArrayList<>();
                this.parent = parent;
        }
        private void addParticle()
        {
                float vx = (float) (particleSpeed.x * 2*(Math.random()-0.5));
                float vy = (float) (particleSpeed.y * 2*(Math.random()-0.5));
                particles.add(new Particle(pos, new PVector(vx,vy), particleColor,
                                radius, lifetime, particleColor));
        }
        @Override
        public void move(float dt) {
                super.move(dt);
                addParticle();
                for (int i = particles.size()-1;i>=0;i--) {
                        Particle p = particles.get(i);
                        p.move(dt);
                        if (p.isDead()) particles.remove(i);
                }
        }
        public void display(PApplet p, SubPlot plt) {
                for (Particle particle : particles) {
                        particle.display(p, plt);
                }
        }
}
```

Classe *SolarSystemApp*: classe onde é feira a representação do sistema solar, com as classes CelestialBody e ParticleSystem. O código feito para correr a execucação desta classe é o seguinte:

```
public class SolarSystemApp implements IProcessingApp  {
        private ArrayList<CelestialBody> planets;
        private ArrayList<ParticleSystem> ps;
        private CelestialBody sun;
        private double[] window = {-10,10,-10,10};
        private float[] viewport = {0f,0f,1f,1f};
        private SubPlot plt;
        private float speedUp = 5f;
        private PImage bg;
        private float timer;
        private float timeInterval = 5.0f;
        @Override
        public void setup(PApplet parent) {
                plt = new SubPlot(window,viewport,parent.width,parent.height);
                sun = new CelestialBody(new PVector(), new PVector(),
300000,1,parent.color(255,128,0));
                planets = new ArrayList<CelestialBody>();
                bg = parent.loadImage("src/data/galaxy.jpg");
                ps = new ArrayList<ParticleSystem>();
                timer = 0;
        }
        @Override
        public void draw(PApplet parent, float dt) {
                parent.image(bg, 0, 0);
                timer += dt;
                float[]bb = plt.getBoundingBox();
                parent.fill(205,32);
                parent.rect (bb[0],bb[1],bb[2],bb[3]);
                sun.display(parent, plt);
                for(CelestialBody planet : planets) {
                        PVector f = sun.attraction(planet);
                        planet.applyForce(f);
                        planet.move(dt*speedUp);
                        planet.display(parent, plt);
                }
                if(timer > timeInterval) {
                        timer = 0;
                        ps.add(new ParticleSystem(parent,
                                new PVector(0,parent.random(0,300)),
                                new PVector(parent.random(100,200),200),
                                parent.random(0,5),
                                new PVector(80,30),
                                parent.color(parent.color(255),0,parent.random(128)),
                                        12,
                                        2));
             }
        for(ParticleSystem pS : ps) {
                PVector f2 = sun.attraction(pS);
                pS.applyForce(f2);
                pS.move(dt);
                pS.display(parent,plt);
                }
        }
        @Override
        public void keyPressed(PApplet parent) {}
        @Override
        public void mousePressed(PApplet parent) {
                if(plt.isInside(parent.mouseX, parent.mouseY)) {
                        double[] pp = plt.getWorldCoord(parent.mouseX, parent.mouseY);
                        float velx = parent.random(2,4);
                        float radius = parent.random(0.1f, 0.5f);
                        float mass = parent.random(0.5f, 0.5f);
                        float green = parent.random(100,255);
                        float blue = parent.random(100,255);
                        CelestialBody planet = new CelestialBody(new PVec
tor((float)pp[0],(float)pp[1]),
                        new PVector(velx,0), mass, radius,parent.color(0,green,blue));
                        planets.add(planet);}
        }
}
```

O resultado obtido pode ser encontrado em baixo:

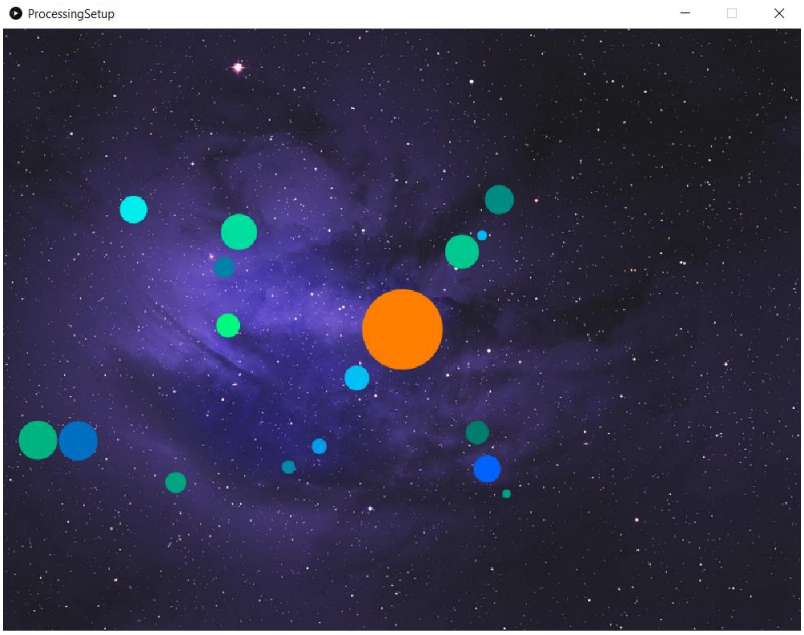


Figura 1 - output gerado, quando corrida a classe SolarSystemApp

Parte C - Modelação baseada em Agentes

Exercício 1

Neste exercício era pedido que fosse feito um boid com travão e acelerador. O código criado foi o seguinte:

```
public void keyPressed(PApplet p) {
	if(p.key == ' ') {
		brake = !brake;
	}
	if(p.keyCode == PConstants.UP) {
		dna.maxSpeed += 10;
		System.out.println("Max Speed UP " + dna.maxSpeed);
	}
	else if (p.keyCode == PConstants.DOWN) {
		dna.maxSpeed -= 10;
		System.out.println("Max Speed DOWN " + dna.maxSpeed);
	}
	else if(p.keyCode == PConstants.RIGHT) {
		dna.maxForce += 10;
		System.out.println("Max Force RIGHT " + dna.maxForce);
	}
	else if(p.keyCode == PConstants.LEFT) {
		dna.maxForce -= 10;
		System.out.println("Max Force LEFT " + dna.maxForce);
	}
	if (p.key == 'd') b.setDebug(true);
	else b.setDebug(false);
}
```

Para fazer o travão do boid, foi chamado o método *brake()*, selecionando a teclado do espaço. Para fazer com que o boid muda-se a velocidade e alterar a direção, do boid, foi feito o seguinte:

- Carregando na tecla da seta de cima, é aumentado o valor da velocidade;
- Carregando na tecla da seta de baixo, é diminuído o valor da velocidade;
- Carregando na tecla da seta da esquerda, é alterado a força do boid, virando para a esquerda;
- Carregando na tecla da seta da direita, é alterado a força do boid, virando para a direita;

Em baixo é visto um exemplo de output, para se ver o aumento da velocidade e da força do boid:

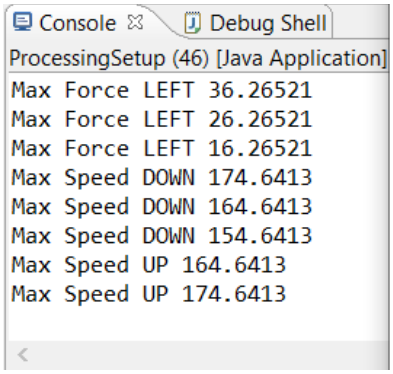


Figura 2 - Output gerado do primeiro exercício.

Exercício 3

Para este exercício, era pedido que fosse feito o comportamento *Wander.* Este comportamento consiste em o boid vaguear no mundo sem obstáculos. O código usado é o seguinte:

```
public PVector wander() {
	PVector center = pos.copy();
	center.add(PVector.mult(vel, dna.deltaTWander));
	 PVector target = new PVector(dna.radiusWander * PApplet.cos(phiWander),
1dna.radiusWander* PApplet.sin(phiWander));
	target.add(center);
	if(phiWander != 0)
		phiWander += p.random(-dna.deltaPhiWander, dna.deltaPhiWander);
	return seek(target);
}
```

Na app onde foi feito correr o programa, foi criado o seguinte bloco de código, permitindo que o boid faça o comportamento Wander:

```
public void draw(PApplet p, float dt) {
		p.background(255);
		PVector f;
		if(timer % 50 == 0) {
			f = b.wander();
			b.applyForce(f);
			timer = 0;
			v = f;
			flag = true;
		}else {
			if(!flag) {
				float value = p.random(-10,10);
				v = new PVector(value,value);
				f = b.seek(v);
				b.applyForce(f);
			}else {
				f = b.seek(v);
				b.applyForce(f);
			}
		}
		b.wander();
		b.move(dt*speedUp);
		b.display(p, plt);
		timer++;
	}
```

Parte D - Modelação baseada em Agentes(Flock)

Exercício 6

Neste exercício era pedido que fosse feito um flock, no qual no meio do flock, surgia um predador a seguir a sua presa. O código realizado pode ser encontrado em baixo:

```
public void draw(PApplet parent, float dt){
		parent.background(255);
		PVector pursuitF,fleeF,wanderF;
		pursuitF = chaser.pursuit(specialOne);
		fleeF = specialOne.flee(pursuitF);
		if(chaser.inSight(specialOne.getPos(), 100)) {
			chaser.applyForce(pursuitF);
		}else {
			if(changePos) {
				wanderF = chaser.seek(target);
				chaser.applyForce(wanderF);
			}else {
				wanderF = chaser.wander();
				chaser.applyForce(wanderF);
			}
		}
		if(specialOne.inSight(chaser.getPos(),200))specialOne.applyForce(fleeF);
		presas.applyBehaviour();
		presas.move(dt);
		presas.display();
		chaser.move(dt);
		chaser.display();
}
```

Quando corrido o programa, o resultado obtido é o seguinte:

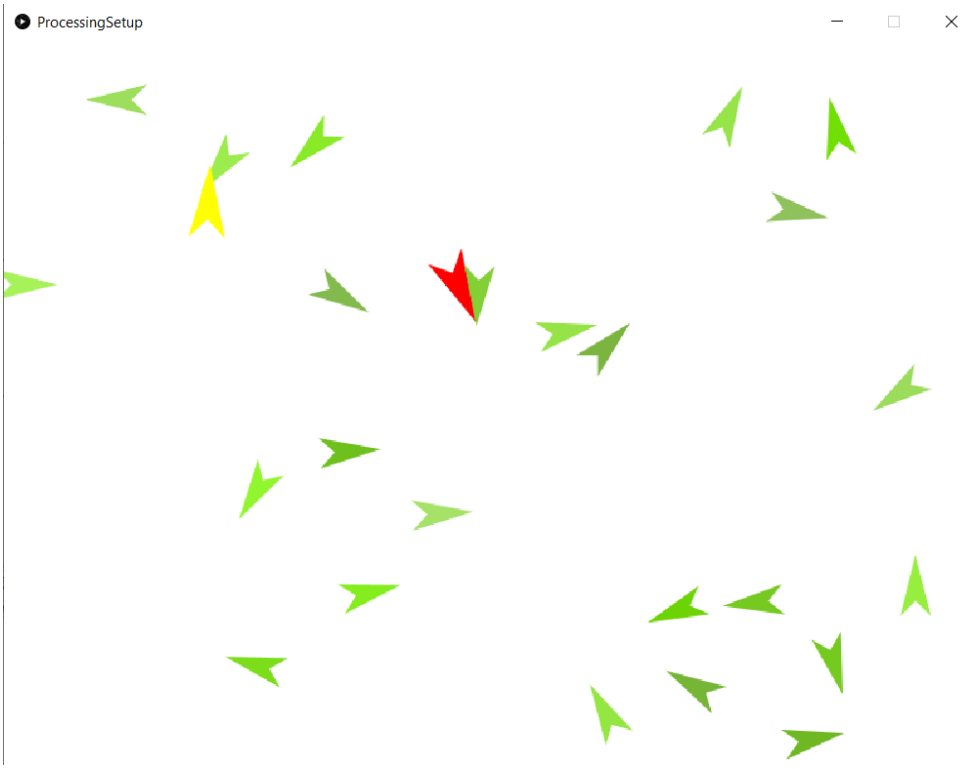


Figura 3 – predador (boid amarelo) a seguir a presa (boid vermelho)

Exercício 8

Neste exercício era pedido que fosse feito um flock, sendo que este comportamento tinha um líder associado. O código criado foi o seguinte:

```
public void draw(PApplet parent, float dt) {
            parent.background(0);
            y = flock.getBoid(nBoids - 1);
            y.setShape(parent.color(0,255,0), 10);
            if(boid.inSight(y.getPos(), 500)) {
                  boid.dna.velMax = parent.random(250,500);
                  PVector f = boid.seek(y.getPos());
                  boid.applyForce(f);
            }else {
                  PVector wander = boid.wander();
                  boid.applyForce(wander);
            }

            flock.applyBehaviour();
            flock.move(dt);
            //boid.move(dt);
            flock.display();
            //boid.display();

      }
```

Quando corrido o programa, o resultado obtido é o seguinte:

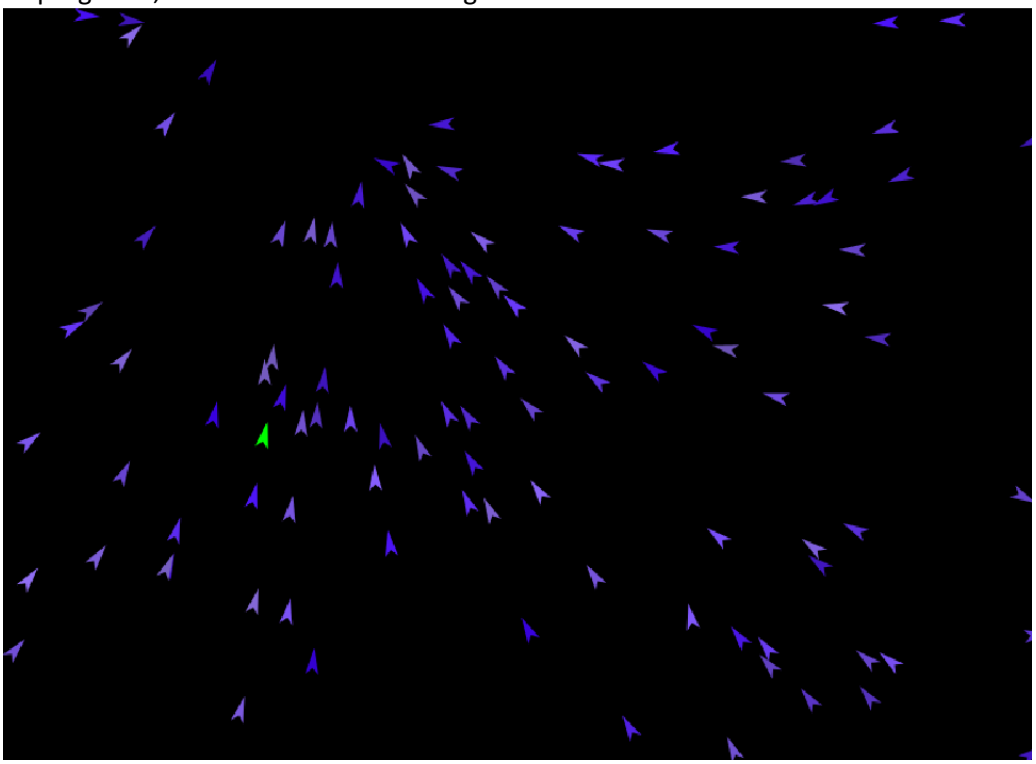


Figura 4 - flock a seguir o líder

Conclusões

Neste trabalho de casa, o grupo conseguiu aprender como associar diferentes comportamentos no boid, aprendendo assim como associar a cada boid, e o resultado que cada boid mostra. Os diferentes métodos permitiram-nos fazer uma simulação de como é que um planeta e uma estrela se comportam.

Assim sendo, o grupo conseguiu executar a maioria dos exercícios, não sendo possível a execução de todos, surgindo algumas dúvidas e complicações no meio. Apesar das dificuldades sentidas, o grupo conseguiu ficar a aprender estes conceitos e consolidar esta matéria, indo com uma "bagagem" que pode vir a ser útil para futuros trabalhos de casa e para o projeto final da disciplina

Bibliografia

1) Folhas fornecidos pelo docente Arnaldo Abrantes
2) https://en.wikipedia.org/wiki/Particle_system
3) https://en.wikipedia.org/wiki/Newton%27s_laws_of_motion