Hénard Aymeric

Compte Rendu IEVA

Note : *l'équilibre des puissances des différents composants n'a pas été évident à mettre en place et n'est sans doute pas parfait, mais le résultat semble tout de même cohérent.*

Question 1 :

modifiez la classe Appli du fichier projet.js de façon à créer un terrain sur lequel des touffes d'herbes sont disposées de façon aléatoire.

```
//Placement d'herbe aléatoire
var maxPos=100; //Correspond à la taille de la map
for (let i = 0; i < 100; i++) {
    //Calcul d'une position aléatoire
    var x = getRandomInt(maxPos);
    var z = getRandomInt(maxPos);
    var name="randomHerbe"+i;
    //Création de la touffe d'herbe
    var randomHerbe = new Herbe(name,{couleur:0xaaff55},this);
    randomHerbe.setPosition(x-maxPos/2,0,z-maxPos/2);
    this.addActeur(randomHerbe) ;
}
```

Question 2 :

mettez en œuvre une nouvelle classe d'acteurs correspondant à des pingouins qui se déplacent en changeant de façon aléatoire de direction tout en restant dans leur « champ ».

Voici le code de la classe acteur correspondant :

Trois composants sont utilisés ici :

- Un composant de mouvement aléatoire
- Un composant de Frottement
- Un composant orientant le pingouin dans la direction dans laquelle il va

```
function Acteur2(nom,data,sim){
    Acteur.call(this,nom,data,sim,new Cylindre(this,{rayon:25,hauteur:1}),new Focus(this,{})) ;

    var repertoire = data.path + "/" ;
    var fObj       = data.obj + ".obj" ;
    var fMtl       = data.mtl + ".mtl" ;

    var obj = chargerObj("tux1",repertoire,fObj,fMtl) ;
    this.setObjet3d(obj) ;

    //Ajout des composants
    //Le pingouin se déplace aléatoirement
    this.ajouterComposant(new MouvementAleatoire(this,{}));
    //Le pingouin subit les frottements qui le ralentissent
    this.ajouterComposant(new Frottement(this,{}));
    //Le pingouin oriente son regard dans la direction dans laquelle il se déplace
    this.ajouterComposant(new RegardeLaOuTuVas(this,{}));
}
```

Et le code d'actualisation de l'acteur :

```
Acteur2.prototype.actualiser = function(dt){
    //console.log(this.objet3d.position) ;
    var t = this.sim.horloge ;

    //Actualisation de la position
    this.objet3d.position.addScaledVector(this.vitesse,dt) ;


    //Verification des bords de la map
    if(this.objet3d.position.x>50.0) this.objet3d.position.x=50.0;
    if(this.objet3d.position.z>50.0) this.objet3d.position.z=50.0;
    if(this.objet3d.position.x<-50.0) this.objet3d.position.x=-50.0;
    if(this.objet3d.position.z<-50.0) this.objet3d.position.z=-50.0;

    //Actualisation de la vitesse
    this.vitesse.addScaledVector(this.acceleration,dt) ;
    //Remise à 0 de l'accélération
    this.acceleration.set(0.0,0.0,0.0) ;

    //Limite la vitesse du pinguin à 15 ms
    var origin=new THREE.Vector3(0,0,0);
    var vit=this.vitesse.distanceTo(origin);
    if(vit > 15 ) {
        this.vitesse.normalize();
        this.vitesse.multiplyScalar(15);
    }
}
```

Voyons maintenant le code des composants utilisés dans cette question.

- Le composant des frottements :

```
//---Frottement---
function Frottement(entite,opts) {
    Composant.call(this,entite);
    this.force = new THREE.Vector3(0.0,0.0,0.0);
    this.k = opts.k || 0.1 ;
}

Frottement.prototype = Object.create(Composant.prototype);
Frottement.prototype.constructor = Frottement;
Frottement.prototype.actualiser = function(dt) {
    this.force.copy(this.entite.vitesse);
    this.force.multiplyScalar(-this.k);
    this.entite.appliquerForce(this.force);
}
```

- Le composant du mouvement aléatoire :

```
//---Mouvement aleatoire---
function MouvementAleatoire(entite, opts) {
    Composant.call(this,entite);
    this.force = new THREE.Vector3(opts.force  || 1.0, 0.0, 0.0 );
    this.alea = opts.alea || 0.1;
    this.puissance=opts.puiss || 10.0;
}

MouvementAleatoire.prototype = Object.create(Composant.prototype);
MouvementAleatoire.prototype.constructor = MouvementAleatoire;
MouvementAleatoire.prototype.actualiser = function(dt) {
    if(Math.random()<this.alea) {
        this.entite.appliquerForce(this.force);
        this.nouvelleForce();
    }
}

MouvementAleatoire.prototype.nouvelleForce = function() {
    //Création d'une force orienté aléatoirement
    var x=(Math.random()-0.5);
    var z=(Math.random()-0.5);
    this.force=new THREE.Vector3(x,0.0,z);
    this.force.normalize();
    //Modification de la puissance de cette force
    this.force.multiplyScalar(this.puissance);
}
```

- Le composant de l'orientation du regard du pingouin :

```
//---RegardeLaOuTuVas---
function RegardeLaOuTuVas(entite,opts) {
    Composant.call(this,entite);
}

RegardeLaOuTuVas.prototype = Object.create(Composant.prototype);
RegardeLaOuTuVas.prototype.constructor = RegardeLaOuTuVas;
RegardeLaOuTuVas.prototype.actualiser = function(dt) {
    var point=new THREE.Vector3();
    point.copy(this.entite.objet3d.position);
    point.addScaledVector(this.entite.vitesse,1.0);
    this.entite.objet3d.lookAt(point);
}
```

Question 3 et 4:

Les acteurs sont dotés d'un « nimbus ». Modifiez le comportement des acteurs pingouins de façon à ce qu'ils se déplacent de façon aléatoire mais que dès qu'ils sont dans le nimbus d'un autre acteur leur comportement se modifie :

• touffe d'herbe : le pingouin se dirige vers la touffe d'herbe et la mange quand il en est suffisamment proche (elle disparaît alors)

• avatar de l'utilisateur : quand la distance entre l'utilisateur et un pingouin est en dessous d'un seuil fixé a priori le pingouin cherche à fuir l'avatar

• pingouin : le pingouin s'approche de son congénère jusqu'à une distance définie a priori (afin d'éviter les collisions)

• La géométrie de la région associée au nimbus peut, au choix du concepteur, être une sphère, un cylindre ou un cône

• L'importance associée à un objet quand on est dans son nimbus peut varier (exemple : l'intensité d'un son en un point donné varie de façon inversement proportionnelle au carré de la distance de ce point à la source sonore considérée.

• On cherche à avoir une architecture aussi générique que possible

Plusieurs composants ont été créés pour répondre à cette question, ainsi que la notion de nimbus.

Pour commencer, je vais vous montrer le concept de nimbus réalisé.

```
function Nimbus(entite) {
    this.entite = entite;
}

Nimbus.prototype.positionIncluDansNimbus = function(pos) {}
Nimbus.prototype.getRayon = function() {}
```

Deux types de nimbus ont été réalisé dans ce projet pour montrer que le concept est généralisé :

- La sphère :

```
//---Sphere
function Sphere(entite,opts) {
    Nimbus.call(this,entite);
    this.rayon = opts.rayon || 1.0;
}

Sphere.prototype = Object.create(Nimbus.prototype);
Sphere.prototype.constructor = Sphere;
Sphere.prototype.positionIncluDansNimbus = function(pos) {
    var dist=this.entite.objet3d.position.distanceTo(pos);
    if(dist<this.rayon) {
        return true;
    }
    else {
        return false;
    }
}

Sphere.prototype.getRayon = function() {
    return this.rayon;
}
```

- Le cylindre :

```
//---Cylindre
function Cylindre(entite,opts) {
    Nimbus.call(this,entite);
    this.rayon = opts.rayon || 1.0;
    this.hauteur = opts.hauteur || 1.0;
}

Cylindre.prototype = Object.create(Nimbus.prototype);
Cylindre.prototype.constructor = Cylindre;
Cylindre.prototype.positionIncluDansNimbus = function(pos) {
    if(this.entite.objet3d.position.y<=(pos.y+this.hauteur) && this.entite.objet3d.position.y>=pos.y) {
        var point1=new THREE.Vector3();
        point1.copy(this.entite.objet3d.position);
        point1.y=0;
        var point2=new THREE.Vector3();
        point2.copy(pos);
        point2.y=0;

        var dist=point1.distanceTo(point2);

        if(dist<this.rayon) {
            return true;
        }
        else {
            return false;
        }
    }
    else {
        return false;
    }
}

Cylindre.prototype.getRayon = function() {
    return this.rayon;
}
```

Ensuite voici les différents composants qui ont été mis en place :

• *touffe d'herbe : le pingouin se dirige vers la touffe d'herbe et la mange quand il en est suffisamment proche (elle disparaît alors)*

Le pingouin se rapproche des touffes d'herbe :

```
//---HerbeAttirance---
function HerbeAttirance(entite,opts) {
    Composant.call(this,entite);
    this.puissance=15.0;
    this.max=15;  //Acceleration maximale que ce composant peut prodiguer
    this.min=0.5;   //Acceleration minimale que ce composant peut prodiguer
}

HerbeAttirance.prototype = Object.create(Composant.prototype);
HerbeAttirance.prototype.constructor = HerbeAttirance;
HerbeAttirance.prototype.actualiser = function(dt) {
    if(this.entite) {
        var tab = this.entite.sim.acteurs;
        var n = tab.length ;
        for(var i=0; i<n; i++){
            if(tab[i] instanceof Herbe) {
                if(tab[i].nimbus.positionIncluDansNimbus(this.entite.objet3d.position) && this.entite.focus.positionIncluDansFocus(tab[i].objet3d.position) ) {

                    var force=new THREE.Vector3();
                    force.subVectors(tab[i].objet3d.position,this.entite.objet3d.position);
                    force.normalize();


                    //Calcul de la distance entre l'herbe et le pingouin
                    var dist=tab[i].objet3d.position.distanceTo(this.entite.objet3d.position);
                    if(dist<0.01) dist=0.01;
                    if(dist>50) dist=50;

                    //Verification de la puissance de l'accélération par rapport à min et max
                    var rapport=this.puissance/(dist*dist);
                    if(rapport<this.min)rapport=this.min;
                    if(rapport>this.max)rapport=this.max;
                    force.multiplyScalar(rapport);

                    this.entite.appliquerForce(force);
                }
            }
        }
    }
```

Le pingouin mange les touffes d'herbe :

```
//---MangerHerbe---
function MangerHerbe(entite,opts) {
    Composant.call(this,entite);
}

MangerHerbe.prototype = Object.create(Composant.prototype);
MangerHerbe.prototype.constructor = MangerHerbe;
MangerHerbe.prototype.actualiser = function(dt) {
    if(this.entite) {
        var tab = this.entite.sim.acteurs;
        var n = tab.length ;
        for(var i=0; i<n; i++){
            if(tab[i] instanceof Herbe) {
                if(tab[i].objet3d.position.distanceTo(this.entite.objet3d.position)<=1.0) {
                    this.entite.sim.supprimerActeur(tab[i])
                    this.entite.vitesse=new THREE.Vector3(0,0,0);
                }
            }
        }
    }
}
```

(Pour supprimer les acteurs de la liste, des méthodes ont été ajoutées dans le code de sim.js)

• *avatar de l'utilisateur : quand la distance entre l'utilisateur et un pingouin est en dessous d'un seuil fixé a priori le pingouin cherche à fuir l'avatar*

Le composant permettant au pingouin de fuir l'utilisateur

```
//--EviterHumain---
function EviterHumain(entite,opts) {
    Composant.call(this,entite);
    this.puissance=10.0;
    this.max=10;
    this.min=0.5;
}

EviterHumain.prototype = Object.create(Composant.prototype);
EviterHumain.prototype.constructor = EviterHumain;
EviterHumain.prototype.actualiser = function(dt) {
    if(this.entite) {

        var dist=this.entite.objet3d.position.distanceTo(this.entite.sim.camera.position);
        if(dist<=10) {
            var force=new THREE.Vector3();
            force.subVectors(this.entite.objet3d.position,this.entite.sim.camera.position);
            force.normalize();
            force.y=0.0;
            if(dist<0.01) dist=0.01;
            if(dist>50) dist=50;

            var rapport=this.puissance/(dist*dist);
            if(rapport<this.min)rapport=this.min;
            if(rapport>this.max)rapport=this.max;
            force.multiplyScalar(rapport);

            force.multiplyScalar(this.puissance/(dist*dist));
            this.entite.appliquerForce(force);
        }

    }
}
```

• *pingouin : le pingouin s'approche de son congénère jusqu'à une distance définie a priori (afin d'éviter les collisions)*

Quand le pingouin est trop loin il s'en approche, si il est trop proche il s'éloigne :

```
//---HerbeAttirance---
function Acteur2Attirance(entite,opts) {
    Composant.call(this,entite);
    this.puissance=4.0;
    this.distance=2.0;
}

Acteur2Attirance.prototype = Object.create(Composant.prototype);
Acteur2Attirance.prototype.constructor = Acteur2Attirance;
Acteur2Attirance.prototype.actualiser = function(dt) {
    if(this.entite) {
        var tab = this.entite.sim.acteurs;
        var n = tab.length ;
        for(var i=0; i<n; i++){
            if(tab[i] instanceof Acteur2) {
                if(tab[i].nimbus.positionIncluDansNimbus(this.entite.objet3d.position)) {

                    var dist=this.entite.objet3d.position.distanceTo(tab[i].objet3d.position);
                    var force=new THREE.Vector3();
                    if(dist<this.distance) { //Trop proche, il s'éloigne

                        force.subVectors(this.entite.objet3d.position,tab[i].objet3d.position);
                    }
                    else { //Trop loin il se rapproche

                        force.subVectors(tab[i].objet3d.position,this.entite.objet3d.position);
                    }

                    force.normalize();
                    if(dist<1) dist=1;
                    if(dist>50) dist=50;

                    force.multiplyScalar(this.puissance/(dist*dist));
                    this.entite.appliquerForce(force);
                }
            }
        }
```

Question 5:

ajoutez à la notion de nimbus celle de focus. On pourra alors définir une notion de perception périphérique de B par A (par exemple : A dans le nimbus de B mais B n'appartenant pas au focus de A) et des comportements différents de A selon cette perception.

```
function Focus(entite,opts) {
    this.entite = entite;
    this.angle=opts.angle || 165;
    this.distance=opts.distance || 10;
}
Focus.prototype = Object.create(Focus.prototype);
Focus.prototype.constructor = Focus;
Focus.prototype.positionIncluDansFocus = function(pos) {
    var dist=pos.distanceTo(this.entite.objet3d.position);
    if(dist>this.distance) return false;
    var vec1=new THREE.Vector3();
    var vec2=new THREE.Vector3(0,0,1);
    vec2.applyQuaternion(this.entite.objet3d.quaternion);

    vec1.subVectors(pos,this.entite.objet3d.position);

    //Calcul de l'angle entre le centre de la vision
    var ang=vec2.angleTo(vec1);
    ang=ang*180.0/Math.PI;
    if(ang<this.angle) return true;
    else return false;
}
```

Afin de prendre en compte le focus, les pingouins ne sont attirés par l'herbe que si la touffe d'herbe est dans leur focus.

Question 6 :

L*es pingouins laissent derrière eux des traces de phéromones. Il s'agit de substances volatiles qui disparaissent avec le temps. Modifiez le comportement des pingouins de façon à ce qu'ils puissent être attirés par les phéromones des autres pingouins (sans pour autant qu'ils en oublient de se nourrir). Visualiser les phéromones par des sphères bleues plus ou moins transparentes selon leur « âge ». De façon à avoir des traitements réguliers, on considérera que les phéromones ont un nimbus.*

Voici la composition d'un acteur "Phéromone". Il a une durée de vie et connaît le pingouin qui l'a relâché. Comme demandé, ils sont plus transparents quand ils sont proche de disparaître.

```
function Pheromone(nom,data,sim,proprietaire){
    Acteur.call(this,nom,data,sim,new Sphere(this,{rayon:5}), null) ;

    var rayon    = data.rayon || 0.25 ;
    var couleur = data.couleur || 0x0000ff ;

    this.dureeDeVie =10.0;
    this.count=0;
    this.createur = proprietaire;

    var sph = creerSphere(nom,{rayon:rayon, couleur:couleur}) ;
    sph.material.transparent = true;
    sph.material.opacity = 1;
    this.setObjet3d(sph) ;
}


Pheromone.prototype = Object.create(Acteur.prototype) ;
Pheromone.prototype.constructor = Pheromone ;
Pheromone.prototype.actualiser = function(dt){
    this.count=this.count+dt;
    //Duree de vie écoulée, suppression du phéromone
    if(this.count>this.dureeDeVie) {
        this.sim.supprimerActeur(this);
    }
    //Mettre à jour la transparence en fonction de la duree de vie restante
    this.objet3d.material.opacity = 1*(this.dureeDeVie-this.count)/this.dureeDeVie;

}
```

Ensuite, voici le code permettant aux pingouins de libérer des phéromones.

```
function LibererPheromone(entite,opts) {
    Composant.call(this,entite);
    this.delai = 4.0;
    this.count = 0.0;
}

LibererPheromone.prototype = Object.create(Composant.prototype);
LibererPheromone.prototype.constructor = LibererPheromone;
LibererPheromone.prototype.actualiser = function(dt) {
    if(this.entite) {
        if(this.count>=this.delai) {
            var name="randomPheromone";
            var newPheromone = new Pheromone(name,{couleur:0x3333ee},this.entite.sim,this.entite) ;
            newPheromone.setPosition(this.entite.getPosition().x,0,this.entite.getPosition().z) ;
            this.entite.sim.addActeur(newPheromone) ;
            this.count=0.0;
        }
        this.count=this.count+dt;
    }
}
```

Et le code permettant aux pingouins de s'approcher des phéromones, similaire au code pour s'approcher de l'herbe :

```
//---PheromoneAttirance---
function PheromoneAttirance(entite,opts) {
    Composant.call(this,entite);
    this.puissance=7.0;
    this.max=7;
    this.min=0.5;
}

PheromoneAttirance.prototype = Object.create(Composant.prototype);
PheromoneAttirance.prototype.constructor = PheromoneAttirance;
PheromoneAttirance.prototype.actualiser = function(dt) {
    if(this.entite) {
        var tab = this.entite.sim.acteurs;
        var n = tab.length ;
        for(var i=0; i<n; i++){
            if(tab[i] instanceof Pheromone && tab[i].createur != this.entite) {
                if(tab[i].nimbus.positionIncluDansNimbus(this.entite.objet3d.position) && this.entite.focus.positionIncluDansFocus(tab[i].objet3d.position) )

                    var force=new THREE.Vector3();
                    force.subVectors(tab[i].objet3d.position,this.entite.objet3d.position);
                    force.normalize();

                    var dist=tab[i].objet3d.position.distanceTo(this.entite.objet3d.position);
                    if(dist<0.01) dist=0.01;
                    if(dist>50) dist=50;

                    var rapport=this.puissance/(dist*dist);
                    if(rapport<this.min)rapport=this.min;
                    if(rapport>this.max)rapport=this.max;
                    force.multiplyScalar(rapport);

                    this.entite.appliquerForce(force);
                }
            }
        }
    }
}
```

Le pingouin qui a relâché le phéromone ne peut pas être attiré par celui-ci.

Question 7 :

Introduisez un nouveau type de pingouins : ils évoluent en groupes en utilisant les concepts introduits par les boids (Reynolds), c'est-à-dire les règles de cohésion, séparation et alignement. Placez dans le champ quelques rochers que les membres du groupe doivent être en mesure d'éviter(ils reforment leur groupe après avoir franchi un obstacle).

Afin de répondre à cette question, nous avons créé un nouveau type de pingouin, avec des composants qui lui sont propre.

```
function Acteur3(nom,data,sim){
    Acteur.call(this,nom,data,sim,new Cylindre(this,{rayon:10,hauteur:1}),new Focus(this,{})) ;

    var repertoire = data.path + "/" ;
    var fObj       = data.obj + ".obj" ;
    var fMtl       = data.mtl + ".mtl" ;

    var obj = chargerObj("tux1",repertoire,fObj,fMtl) ;
    this.setObjet3d(obj) ;

    this.vitesseMax = 5.0; //Vitesse max en m/s que le pingouin peut atteindre

    //Ajout des composants
    this.ajouterComposant(new MouvementAleatoire(this,{puiss:20})); //Le pingouin se déplace aléatoirement
    this.ajouterComposant(new Frottement(this,{}));                 //Le pingouin subit les frottements qui le ralentissent
    this.ajouterComposant(new RegardeLaOuTuVas(this,{}));           //Le pingouin oriente son regard dans la direction dans laquelle il se déplace
    this.ajouterComposant(new Cohesion(this,{}));                   //Régle de cohésion des boids
    this.ajouterComposant(new Separation(this,{}));                 //Règle de séparation des boids
    this.ajouterComposant(new Alignement(this,{}));                 //Règle d'alignement des boids
    this.ajouterComposant(new EvitementDeRocher(this,{}));          //Le pingouin évite les rochers
}
```

A la différence de la classe acteur2, ce type de pingouin ricoche sur les bords, afin d'éviter d'y rester collé avec son boid.

```
Acteur3.prototype.actualiser = function(dt){
    var t = this.sim.horloge ;

    //Actualisation de la position
    this.objet3d.position.addScaledVector(this.vitesse,dt) ;


    //Verification des bords de la map
    var x=0;
    var z=0;
    if(this.objet3d.position.x>50.0) x=-1//this.objet3d.position.x=50.0;
    if(this.objet3d.position.z>50.0) z=-1; //this.objet3d.position.z=50.0;
    if(this.objet3d.position.x<-50.0) x=1; //this.objet3d.position.x=-50.0;
    if(this.objet3d.position.z<-50.0) z=1; //this.objet3d.position.z=-50.0;

    //Rebond si un bord de la map est atteint
    var rebond=new THREE.Vector3(x,0,z);
    rebond.multiplyScalar(20);
    this.appliquerForce(rebond);

    //Mise à jour de la vitesse
    this.vitesse.addScaledVector(this.acceleration,dt);

    //Reinitialisation de l'accélération
    this.acceleration.set(0.0,0.0,0.0) ;

    //Limite la vitesse du pinguin en fonction de this.vitesseMax
    var origin=new THREE.Vector3(0,0,0);
    var vit=this.vitesse.distanceTo(origin);
    if(vit > this.vitesseMax ) {
        this.vitesse.normalize();
        this.vitesse.multiplyScalar(this.vitesseMax);
    }
}
```

Ensuite, 3 composants représentant les 3 règles des boids de Reynold ont été créés.

- La cohésion :

```
//---Cohesion---
function Cohesion(entite,opts) {
    Composant.call(this,entite);
    this.k=1.0;  //Force de la cohésion (laisser à 1 de préférence)
}


Cohesion.prototype = Object.create(Composant.prototype);
Cohesion.prototype.constructor = Cohesion;
Cohesion.prototype.actualiser = function(dt) {
    if(this.entite) {

        var count = 0; //Nombre de pingouins proches
        var g =new THREE.Vector3(); //Correspond à la position moyenne des pingouins proches

        //Parcours de la liste des acteurs
        var tab = this.entite.sim.acteurs;
        var n = tab.length ;
        for(var i=0; i<n; i++){
            if(tab[i] instanceof Acteur3 && tab[i] != this.entite && this.entite.nimbus.positionIncluDansNimbus(tab[i].objet3d.position)) {
                count = count + 1;

                var p =tab[i].getPosition();

                g.addScaledVector(p,1.0); //Ajout de la position du pingouin au total
            }
        }

        if(count>0) {
            var force=new THREE.Vector3();
            g.multiplyScalar(1/count);
            force.subVectors(g,this.entite.objet3d.position); //Vecteur entre la position moyenne des pingouins proches et du pingouin
            force.normalize();
            force.multiplyScalar(this.k);
            this.entite.appliquerForce(force);
        }
    }
}
```

- La séparation :

```
//---Separation---
function Separation(entite,opts) {
    Composant.call(this,entite);
}


Separation.prototype = Object.create(Composant.prototype);
Separation.prototype.constructor = Separation;
Separation.prototype.actualiser = function(dt) {
    if(this.entite) {

        var count = 0; //Nombre de pingouin dans le nimbus
        var totalForce = new THREE.Vector3();

        //Parcours de la liste des acteurs
        var tab = this.entite.sim.acteurs;
        var n = tab.length ;
        for(var i=0; i<n; i++){
            if(tab[i] instanceof Acteur3 && tab[i] != this.entite && this.entite.nimbus.positionIncluDansNimbus(tab[i].objet3d.position)) {

                count=count+1;

                //Calcul de la direction de la force (s'éloigner du pingouin)
                var force=new THREE.Vector3();
                force.subVectors(this.entite.objet3d.position,tab[i].objet3d.position);
                force.normalize();

                //Ajout de la force au total
                totalForce.addScaledVector(force,1.0);
            }
        }
        if(count>0) {
            //Calcul de la moyenne
            totalForce.multiplyScalar(1/count);
            //Ajout de la force
            this.entite.appliquerForce(totalForce);
        }
    }
```

- L’alignement :

```
function Alignement(entite,opts) {
    Composant.call(this,entite);
    this.k=1.0;

}

Alignement.prototype = Object.create(Composant.prototype);
Alignement.prototype.constructor = Alignement;
Alignement.prototype.actualiser = function(dt) {
    if(this.entite) {

        var count = 0; //Nombre de pingouin dans le nimbus
        var vm = new THREE.Vector3();

        //Parcours de la liste des acteurs
        var tab = this.entite.sim.acteurs;
        var n = tab.length ;
        for(var i=0; i<n; i++){
            if(tab[i] instanceof Acteur3 && tab[i] != this.entite && this.entite.nimbus.positionIncluDansNimbus(tab[i].objet3d.position)) {

                count=count+1;

                //Ajout de la force au total
                vm.addScaledVector(tab[i].vitesse,1.0);         }
        }
        if(count>0) {

            vm.multiplyScalar(1/count);

            //Calcul de fa
            var fa=new THREE.Vector3();
            fa.subVectors(vm,this.entite.vitesse);
            vm.multiplyScalar(this.k);

            //Ajout de la force
            this.entite.appliquerForce(vm);
        }
    }
}
```

Enfin, un dernier composant à été créé afin de permettre aux pingouins d’éviter les rochers.

```
function EvitementDeRocher(entite,opts) {
    Composant.call(this,entite);
    this.k=2.0;  //Distance du palpeur par rapport a l'acteur
}


EvitementDeRocher.prototype = Object.create(Composant.prototype);
EvitementDeRocher.prototype.constructor = EvitementDeRocher;
EvitementDeRocher.prototype.actualiser = function(dt) {
    if(this.entite) {
        //Calculer la position du palpeur
        var palpeur=this.entite.vitesse.clone();
        palpeur.normalize();
        palpeur.multiplyScalar(this.k);
        palpeur.addScaledVector(this.entite.getPosition(),1.0);


        var tab = this.entite.sim.acteurs;
        var n = tab.length ;

        for(var i=0; i<n; i++){
            if(tab[i] instanceof Rocher && tab[i].nimbus.positionIncluDansNimbus(palpeur)) {

                //Calcul de "(r+d)"   {1}
                var r=this.entite.nimbus.getRayon();
                var d=tab[i].objet3d.position.distanceTo(this.entite.objet3d.position);
                var rd=r+d;

                //Calcul de " CP' / ||CP'|| "  {2}
                var cp=new THREE.Vector3();
                cp.subVectors(palpeur,tab[i].objet3d.position);
                cp.normalize();

                //Calcul final  "{1}*{2} - C"   {3}
                cp.multiplyScalar(rd);
                pFinale=new THREE.Vector3();
                pFinale.subVectors(cp,tab[i].objet3d.position);
                pFinale.y=0;

                //Seek({3})
                var force=new THREE.Vector3();
                force.subVectors(pFinale,this.entite.getPosition());
                force.normalize();
                force.multiplyScalar(10);

                //Ajout de la force
                this.entite.appliquerForce(force);
```