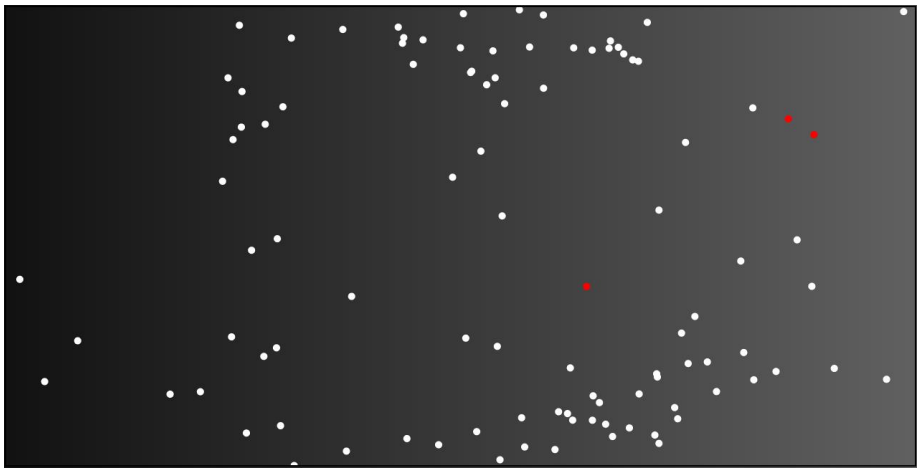
Flocking Simulation
Relatório de Projeto



Salvador de Araújo Coder Meira

Aluno nº 44886

Sistemas Multi-Agente

Mestrado em Engenharia Informática

Faculdade de Ciências da Universidade de Lisboa

30/01/2021

1. Introdução

Este projeto teve como finalidade desenvolver em JavaScript uma representação visual de Flocks, isto é, grupos de animais como rebanhos, bandos ou cardumes, segundo o modelo de Craig Reynolds [1]. O autor descreve três regras fundamentais pelas quais se segue cada elemento deste tipo de conjuntos:

- Separação: Orientar-se para evitar aglomeração de elementos locais;

- Alinhamento: Orientar-se para tomar o rumo médio dos elementos locais;

- Coesão: Orientar-se para se mover em direção à posição média dos elementos locais;

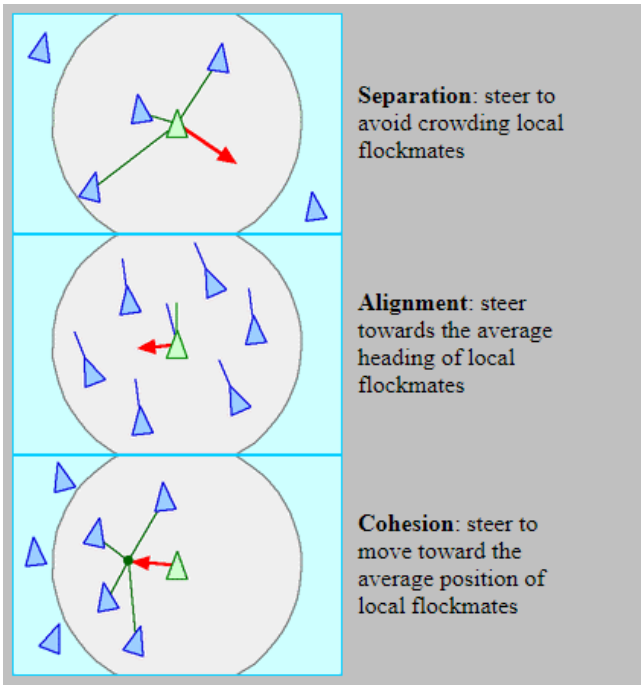


Figura 1. As três regras do modelo de Craig Reynolds [1].

Coube-me então implementar estas regras de forma a que cada elemento do conjunto adquirisse este comportamento, agindo consoante as posições e velocidades dos elementos em seu redor. Para além disto, implementei também um predador que persegue e come os elementos comuns, ignorando as regras do grupo, e um segundo predador maior que come apenas os primeiros predadores e segue-se pelas restantes regras de movimento dos elementos comuns.

2. Abordagem

Para desenvolver esta representação decidi usar JavaScript, pois é uma linguagem de programação com a qual já possuo algum hábito e destreza ao usar, devido a bases e conhecimentos em Java adquiridos ao longo da minha aprendizagem académica. O resultado observável deste projeto encontra-se numa página web local e foi conseguida através de desenvolvimento em HTML, CSS e JavaScript. A fim de conseguir representar as posições, velocidades e outras forças das partículas, adaptei uma classe Vetor bidimensional desenvolvida por Joe Grainger e disponível publicamente no github [2].

3. Desenvolvimento

3.1. Representação e animação

Comecei por utilizar o elemento Canvas de HTML, criando a janela onde se desenrola a animação. A partir daqui explorei formas de desenhar e animar as partículas. O método requestAnimationFrame() do JavaScript permite criar um ciclo contínuo onde a animação pode ser atualizada a cada frame.

3.2. Construtor da classe Boid

Cada partícula foi definida com a classe Boid, nome que o autor dá a estas partículas, e estes Boids estão colocados num array de Boids chamado Flock. A posição, velocidade e aceleração de cada elemento foram definidas através de vetores bidimensionais que possuem uma coordenada X e uma coordenada Y. A posição é inicializada num ponto aleatório da Canvas, a velocidade é inicializada com magnitude e direção aleatórias e a aceleração começa como vetor nulo. Para além destes três vetores, a classe Boid tem ainda os atributos perceção, isto é o raio de visibilidade de cada partícula, que só age consoante as partículas locais, ou seja, as que estão dentro deste raio; a força, isto é a máxima força que cada regra pode aplicar sobre si; a máxima velocidade, fazendo com que o elemento aja a esta velocidade sempre que segue uma das regras (o predador tem uma maior velocidade máxima do que os elementos comuns e do que o segundo predador); o tipo, ou seja, se é uma partícula comum, se é um predador, ou se é um segundo predador (tipos 0, 1 ou 2, respetivamente); e por fim, um valor booleano que indica se este Boid está vivo ou morto.

```
class Boid {
    constructor(type){
        this.position = new Vector(random(0, width), random(0, height));
        this.velocity = new Vector();
        this.velocity.setMagnitude(random(2, 4));
        this.velocity.setDirection(random(0, 2 * Math.PI));
        this.acceleration = new Vector();
        this.perception = 200;
        this.maxForce = 0.1;
        this.maxSpeed = 3;
        this.type = type;
        this.isAlive = true;
        if (this.type == 1) this.maxSpeed = 5;
    }
```

Figura 2. Construtor da classe Boid.

3.3. Regras de movimento conjunto

Estas regras são a separação, o alinhamento e a coesão. As três regras têm em comum as seguintes características:

- Retornam um vetor de “steering”, isto é a força de orientação aplicada por cada regra;

- Nenhuma das regras se aplica ao predador que persegue os elementos comuns;

- Calculam a distância entre si próprios e os elementos à sua volta. Caso a distância esteja dentro do raio de perceção, o total de elementos para o cálculo é incrementado e a regra aplica-se;

- Ao vetor steering é sempre definida a magnitude com a velocidade máxima, para que a partícula aja segundo a regra a esta velocidade; é subtraído o vetor da própria velocidade atual, para que o vetor de steering seja gradualmente adaptado tendo em conta o rumo que já levava anteriormente; e é limitado à máxima força, para que cada regra afete o elemento de forma moderada até certo ponto.

3.3.1. Separação

A separação serve para prevenir a aglomeração de elementos no mesmo local. Neste método é criado um vetor relativo às diferenças entre as posições de cada dois elementos que com que estamos a lidar. Assim, subtrai-se a posição doutro elemento a este elemento e é criado um vetor com sentido oposto ao outro elemento, mas com o mesmo comprimento. A fim de haver um movimento de separação proporcional à proximidade dos elementos, ou seja, mais separação para elementos mais próximos e menos separação para elementos mais afastados, este vetor de diferença é dividido pela distância entre os elementos. São então somadas à força steering todos estes vetores de diferença. Se o total de elementos em redor for superior a zero, ou seja, caso haja outros elementos dentro do raio de perceção, o steering é dividido pelo número total de elementos e tem-se então a média das diferenças. Este vetor é retornado depois de ser definida a magnitude da velocidade máxima, ser subtraída a velocidade atual e ser limitado à máxima força. Caso não aja elementos próximos, o vetor retornado é o vetor nulo.

```
separation(boids){
    var steering = new Vector();
    if (this.type == 1) return steering;
    var total = 0;
    for (var other of boids) {
        var d = this.position.distance(other.position);
        if (other != this && other.type != 1 && d < this.perception) {
            var difference = this.position.subNew(other.position);
            difference.divide(d);
            steering.add(difference);
            total++;
        }
    }
    if (total > 0) {
        steering.divide(total);
        steering.setMagnitude(this.maxSpeed);
        steering.subtract(this.velocity);
        steering.limit(this.maxForce);
    }
    return steering;
}
```

Figura 3. Método que define a força da separação.

3.3.2. Alinhamento

O alinhamento faz com que os elementos se movam na mesma direção que a direção média dos elementos vizinhos, fazendo com que no final tomem todos o mesmo rumo. Aqui, o importante a apontar são as velocidades dos elementos vizinhos, pois é o vetor de velocidade que define a direção de

um elemento. De forma análoga ao método anterior, são somadas ao vetor steering todas as velocidades dos elementos vizinhos e dividido pelo número total de vizinhos. Tendo a velocidade média dos vizinhos, este elemento deve mover-se gradualmente para a direção resultante do cálculo efetuado.

```
alignment(boids){
    var steering = new Vector();
    if (this.type == 1) return steering;
    var total = 0;
    for (var other of boids) {
        var d = this.position.distance(other.position);
        if (other != this && other.type != 1 && d < this.perception) {
            steering.add(other.velocity);
            total++;
        }
    }
    if (total > 0) {
        steering.divide(total);
        steering.setMagnitude(this.maxSpeed);
        steering.subtract(this.velocity);
        steering.limit(this.maxForce);
    }
    return steering;
}
```

Figura 4. Método que define a força do alinhamento.

3.3.3. Coesão

Com a coesão, os elementos vão se movendo em direção à posição média dos elementos vizinhos, criando visualmente a uma forma de um conjunto. O importante a apontar na coesão é a posição média dos elementos vizinhos. Assim, são somadas todas as posições dos elementos vizinhos e é subtraído o número total de elementos. O vetor steering resultante é então o vetor em direção a esta posição média.

```
cohesion(boids){
    var steering = new Vector();
    if (this.type == 1) return steering;
    var total = 0;
    for (var other of boids) {
        var d = this.position.distance(other.position);
        if (other != this && other.type != 1 && d < this.perception) {
            steering.add(other.position);
            total++;
        }
    }
    if (total > 0) {
        steering.divide(total);
        steering.subtract(this.position);
        steering.setMagnitude(this.maxSpeed);
        steering.subtract(this.velocity);
        steering.limit(this.maxForce);
    }
    return steering;
}
```

Figura 5. Método que define a força da coesão.

3.4. Caça

O método caça é referente ao movimento que as partículas fazem para caçar ou para fugir a um predador e é dividido em três partes, sendo uma para cada tipo de partícula (comum ou branca, primeiro predador ou predador vermelho, e segundo predador ou predador amarelo).

3.4.1. Fuga dos elementos comuns

Se o Boid que estiver a ser lidado no momento for um elemento comum, tudo o que tem a fazer é evitar os predadores vermelhos, ignorando os predadores amarelos. Para isto, e analogamente ao cálculo da separação, é criado um vetor contrário à posição do predador, caso este esteja dentro do raio de perceção, e esse é o vetor do rumo a tomar.

```
hunting(boids){
    var steering = new Vector();
    if (this.type == 0){
        for (var other of boids) {
            if (other.type == 1) {
                var d = this.position.distance(other.position);
                if (d < this.perception) {
                    var difference = this.position.subNew(other.position);
                    difference.divide(d);
                    steering = difference;
                    steering.setMagnitude(this.maxSpeed);
                    steering.subtract(this.velocity);
                    steering.limit(this.maxForce);
                }
            }
        }
    }
```

Figura 6.1. Caça – Parte do método que define a fuga dos elementos comuns.

3.4.2. Caça e fuga dos primeiros predadores

Aqui, os predadores vermelhos vão fugir dos predadores maiores da mesma forma que os elementos comuns fogem de si. A perseguição para caçar é feita de forma semelhante ao método da coesão, só que aqui a posição pretendida é a posição da vítima, caso esta esteja dentro do raio de perceção, e não a posição média dos vizinhos.

```
else if (this.type == 1) {
    for (var other of boids) {
        if (other.type == 0) {
            var d = this.position.distance(other.position);
            if (d < this.perception) {
                steering.add(other.position);
                steering.subtract(this.position);
                steering.setMagnitude(this.maxSpeed);
                steering.subtract(this.velocity);
                steering.limit(this.maxForce);
            }
        }
        if (other.type == 2) {
            var d = this.position.distance(other.position);
            if (d < this.perception) {
                var difference = this.position.subNew(other.position);
                difference.divide(d);
                steering = difference;
                steering.setMagnitude(this.maxSpeed);
                steering.subtract(this.velocity);
                steering.limit(this.maxForce);
            }
        }
    }
}
```

Figura 6.2. Caça – Parte do método que define a caça e fuga dos primeiros predadores.

3.4.3. Caça dos segundos predadores

Os segundos predadores são maiores e consomem apenas os primeiros predadores, estando assim no topo de uma cadeia alimentar figurada. Estes ignoram os elementos comuns e caçam os predadores

vermelhos da mesma forma que estes caçam os elementos comuns. No final do método é retornado o vetor steering referente ao vetor de movimento a tomar pela ação de caça ou de fuga de cada elemento.

```
else {
    for (var other of boids) {
        if (other.type == 1) {
            var d = this.position.distance(other.position);
            if (d < this.perception) {
                steering.add(other.position);
                steering.subtract(this.position);
                steering.setMagnitude(this.maxSpeed);
                steering.subtract(this.velocity);
                steering.limit(this.maxForce);
            }
        }
    }
}
return steering;
```

Figura 6.3. Caça – Parte do método que define a caça dos segundos predadores e retorna o steering final.

3.5. Eliminação

Este método faz com que cada elemento que se aproxime muito do seu predador seja morto e consequentemente retirado do array de Boids. Para os primeiros predadores, a distância para matar é de 10 pixéis, ou seja, uma presa é morta caso se aproxime a esta distância do predador. Para os segundos predadores a distância é de 20 pixéis, a fim de ficar proporcional ao seu maior tamanho. Caso um Boid seja morto, o seu estado *isAlive* fica com valor *false* e este desaparece da tela.

```
killing(boids){
    if (this.type == 0) return;
    if (this.type == 1) {
        var killingDistance = 10;
        for (var other of boids) {
            var d = this.position.distance(other.position);
            if (other != this && other.type == 0 && d < killingDistance) {
                other.isAlive = false;
            }
        }
    }
    else {
        var killingDistance = 20;
        for (var other of boids) {
            var d = this.position.distance(other.position);
            if (other != this && other.type == 1 && d < killingDistance) {
                other.isAlive = false;
            }
        }
    }
}
```

Figura 7. Método que dita a morte das partículas.

3.6. Flock

Este método junta as forças do steering da separação, alinhamento, coesão e caça, somando-as à aceleração de cada Boid. São aqui também atribuídos escalares de multiplicação, que podem ser definidos através de *sliders* na página web principal.

```
flock(boids){
    var separation = this.separation(boids);
    var alignment = this.alignment(boids);
    var cohesion = this.cohesion(boids);
    var hunting = this.hunting(boids);
    separation.multiply(separationMultiplier);
    alignment.multiply(alignmentMultiplier);
    cohesion.multiply(cohesionMultiplier);
    this.acceleration.add(separation);
    this.acceleration.add(alignment);
    this.acceleration.add(cohesion);
    this.acceleration.add(hunting);
}
```

Figura 8. Método que junta as forças das três regras e da caça.

3.7. Atualizar

Este método é chamado a cada frame e serve para atualizar a posição de cada Boid, somando sempre o seu vetor de velocidade. Ao vetor de velocidade é também somado o vetor de aceleração que conterá o vetor resultante das forças anteriormente descritas e este é limitado ao atributo da velocidade máxima. No final, a aceleração é metida a zero novamente para impedir a acumulação de valores de aceleração anteriores.

```
update() {
    this.position.add(this.velocity);
    this.velocity.add(this.acceleration);
    this.velocity.limit(this.maxSpeed);
    this.acceleration.multiply(0);
}
```

Figura 9. Método que atualiza a posição e velocidade e *reseta* o valor da aleceração a cada frame.

3.8. Desenho

Por fim, o método *show()* desenha as partículas como pequenos círculos ou pontos, sendo os elementos comuns pontos brancos, os primeiros predadores pontos vermelhos e os segundos predadores pontos um pouco maiores e de coloração amarela.

```
show() {
    if (this.type == 0) {
        ctx.fillStyle = "white";
    } else if (this.type == 1) {
        ctx.fillStyle = "red";
    } else {
        ctx.fillStyle = "#ffff66";
    }
    ctx.beginPath();
    if (this.type < 2) {
        ctx.arc(this.position.x,this.position.y,5,0,2*Math.PI);
    }
    else {
        ctx.arc(this.position.x,this.position.y,10,0,2*Math.PI);
    }
    ctx.fill();
}
```

Figura 10. Método que desenha as partículas.

4. Resultados

Para facilitar e melhorar a experiência do utilizador, criei um interface com várias opções de input que incluem o número de Boids brancos, o número de predadores vermelhos, o número de predadores amarelos, e valores dos multiplicadores da separação, alinhamento e coesão.

Figura 11. Interface da página web.

4.1. Apenas Boids comuns

Com apenas Boids brancos comuns, não há perigo em navegar livremente. As partículas movem-se então consoantes as regras de separação, alinhamento e coesão, representando um conjunto de animais semelhante a um rebanho, bando ou cardume.

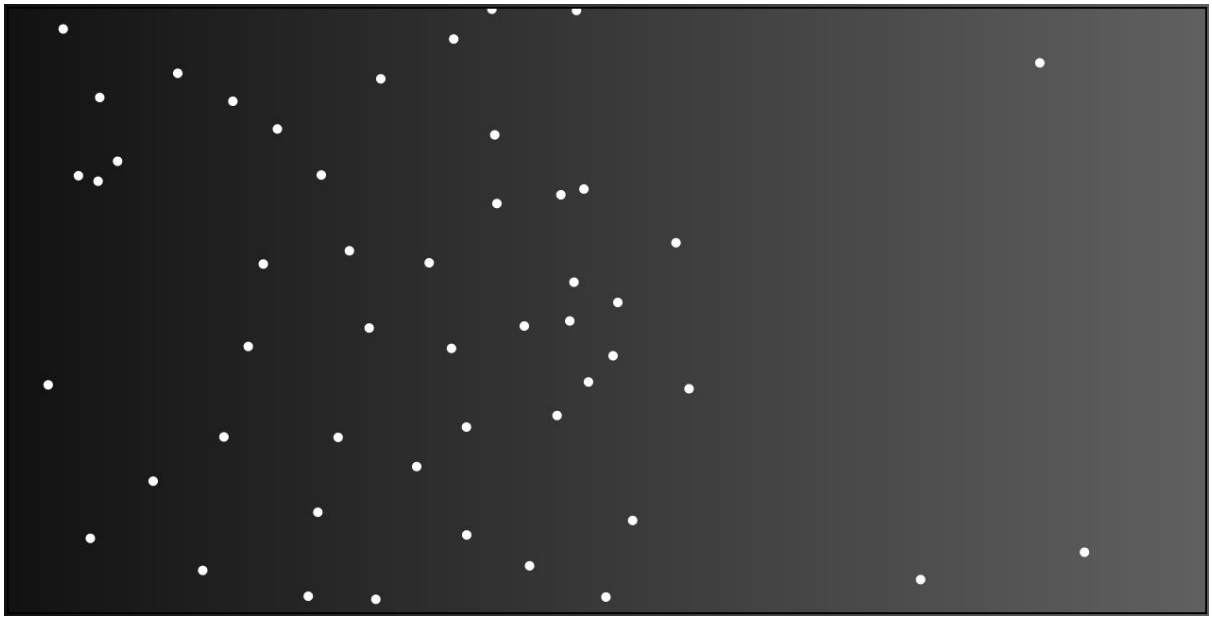


Figura 12. 50 Boids brancos em grupo.

4.2. Boids comuns e predadores vermelhos

Aqui, os Boids brancos seguem-se pelas regras de movimento, mas tendo um predador por perto, fazem o possível para se escaparem deste. Os predadores vermelhos são mais rápidos e vão consumindo os Boids brancos até ficarem sozinhos na tela.

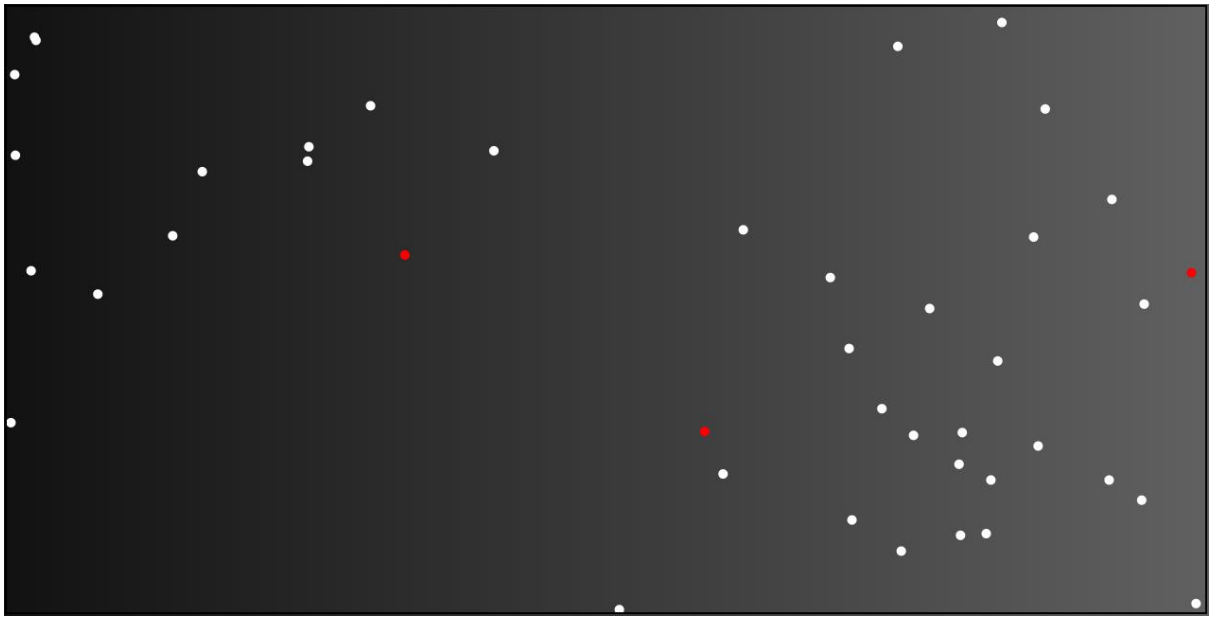


Figura 13. 50 Boids brancos e 3 predadores vermelhos.

4.3. Boids comuns, predadores vermelhos e predadores amarelos

Neste caso, os Boids brancos andam livremente com os Boids maiores amarelos, mas fogem dos predadores vermelhos. Os predadores vermelhos tentam caçar os brancos ao mesmo tempo que se

tentam escapar dos predadores amarelos. Os predadores amarelos andam em manada pacífica com os Boids brancos, mas caçam os vermelhos mal tenham oportunidade.

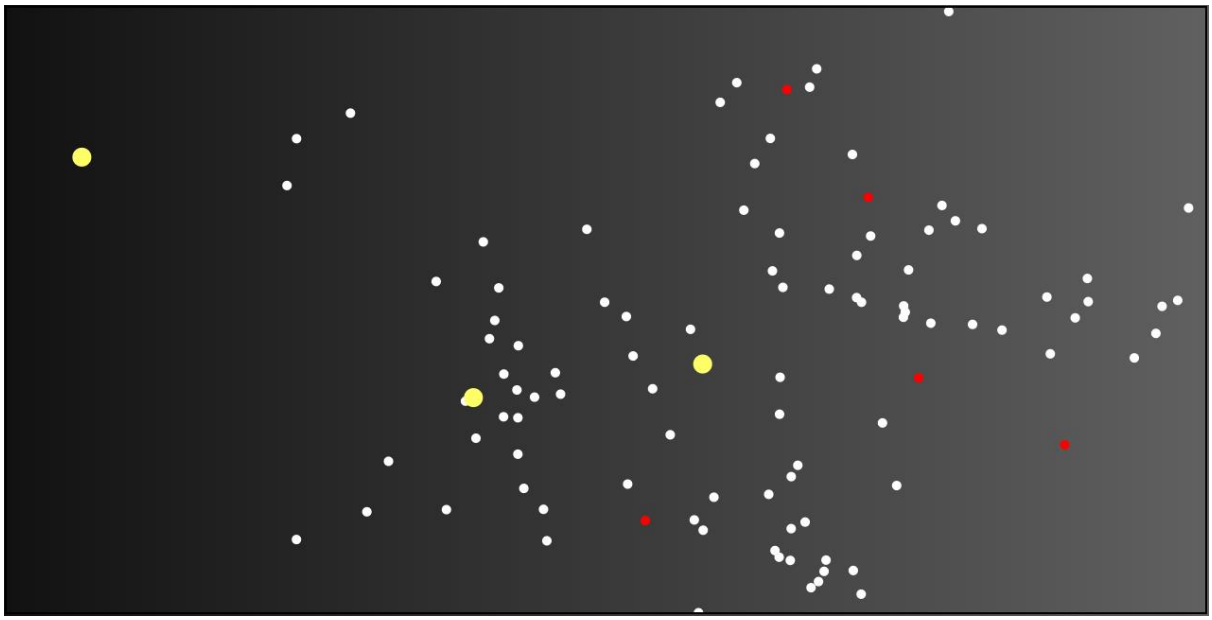


Figura 14. 100 Boids brancos, 5 predadores vermelhos e 3 predadores amarelos.

5. Conclusão

Pode-se dizer que os movimentos efetuados pelas partículas foram os pretendidos. Para criar um comportamento universal de manada implementei o algoritmo definido pelo modelo de Craig Reynolds. Com o intuito de dar uma ainda maior complexidade ao sistema, desenvolvi um método de caça que conta com a existência e interação de diferentes agentes. Um resultado interessante da experiência é que não havendo predadores grande amarelos, os Boids brancos acabam por ser consumidos pelos predadores brancos. Contudo, quantos mais predadores amarelos existirem, maior será a probabilidade de sobrevivência dos Boids comuns, replicando o que acontece num ambiente real de cadeia alimentar.

6. Referências

[1] Craig Reynolds, “Boids (Flocks, Herds, and Schools: a Distributed Behavioral Model)” https://www.red3d.com/cwr/boids/ (visitado pela última vez em 30/01/2021).

[2] Joe Grainger, “A simple Vector class in javascript” https://gist.github.com/jjgrainger/808640fcb5764cf92c3cad960682c677 (visitado pela última vez em 30/01/2021).