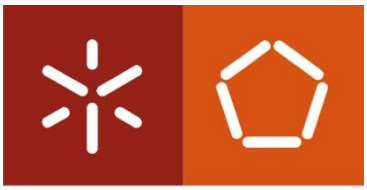

Universidade do Minho

Escola de Engenharia

Departamento de Informática

Mestrado em Engenharia Informática

1ºAno

Computação Gráfica – Modelação e Visualização

Ano Letivo de 2013/2014

Animação de Personagens Virtuais - Boids

Paulo Marques PG25325

Tiago Leite PG25305

Resumo

Neste projecto será apresentado um programa ao carregar um ficheiro XML cria uma série de personagens virtuais com determinados comportamentos de acordo com o ficheiro fornecido.

Área de Aplicação: Computação gráfica, Inteligência artificial

Palavras-Chave: Boid, Catmull-Rom, R-Tree, XML, Map

Índice

Índice de Figuras

1 Introdução

A animação de personagens virtuais é cada vez mais importante nos dias que correm. A simulação de comportamentos animais ou humanos é útil em diversas áreas da computação gráfica e da inteligência artificial, seja para serem utilizadas em jogos para dar vida ao ambiente, como por exemplo dar vida uma cidade pela qual o jogador anda, ou mesmo em simulações de casos reais, como por exemplo um incendio num edifício cheio de pessoas.

Desta forma neste trabalho é procurado dar vida a personagens virtuais dando-lhes uma serie de comportamentos básicos que podem ser usados para criar outros comportamentos mais complexos. É também procurado criar uma estrutura em que seja sempre possível adicionar novos comportamentos que ainda não foram implementados.

2 Desenvolvimento

2.1 Funcionalidades

O objetivo deste projeto, além da implementação das funcionalidades/comportamentos básicos de um sistema de Boids[1], passou por não se tornar limitado a estes mesmos, permitindo um controlo total pelo utilizador. Isto é, elaborámos uma linguagem simples e intuitiva para descrever todos os comportamentos implementados de raiz, incluindo a adição de funcionalidades, permitindo ao utilizador da biblioteca ter acesso a um conjunto de propriedades novas que possa estar interessado e desenvolver os seus próprios sistemas de interação entre os boids.

Nesta secção será feita uma breve descrição das funcionalidades.

Começando pelos comportamentos básicos descritos em [1], para um sistema de simulação de multidões são necessárias 3 funcionalidades básicas:

- **Separação** - Evitar colisões com os colegas.

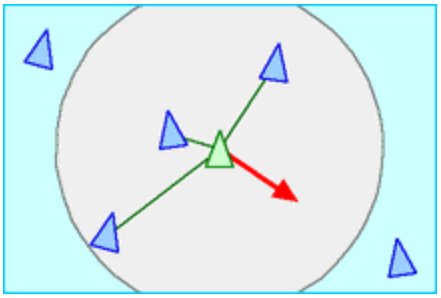

Figura 1 - Separação dos Boids

- **Alinhamento** - Ajustar a velocidade de cada elemento para igualar com a velocidade do resto do grupo.

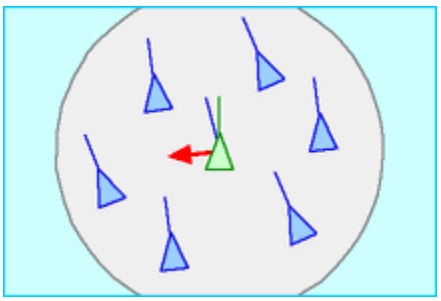

Figura 2 - Alinhamento dos Boids

- **Coesão** - Atrair elementos e mantê-los próximos.

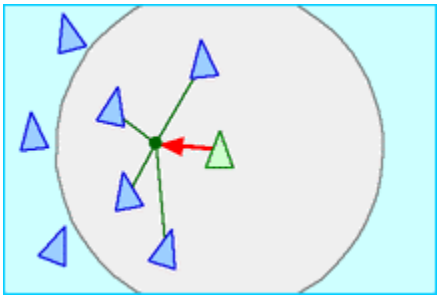

Figura 3 - Coesão dos Boids

Com o intuito de adicionar mais realismo e liberdade, são disponibilizadas mais um conjunto de funcionalidades:

- **Lideres** - Um grupo de boids pode ter um líder o qual deve seguir;
- **Ponto de Objetivo** – Os seus elementos podem ter um certo local como objectivo.
- **Predadores** - Um grupo de boids pode ter um ou mais predadores, os quais devem evitar a proximidade. Um grupo de boids pode ter como predadores todos os elementos de um outro grupo;
- **Caminhos** - Um boid pode ter um caminho específico o qual deve percorrer;
- **Obstáculos** - Como não tem muita utilidade um cenário sem obstáculos, é permitida a inserção destes, os quais são contornados/evitados.
- **Aleatoriedade** – Como na vida real os comportamentos não são assim tão previsíveis, é possível também adicionar um fator aleatório no movimento.

2.1.1 Funcionalidades Especificas

Além das funcionalidades referidas, existe um sistema de controlo de todos os fatores que influenciam o comportamento dos boids, sejam em grupo ou individuais. Sempre em mente a total liberdade é também permitido definir todos os pesos dos fatores referidos para o comportamento final, podendo dar prioridade a certos comportamentos e ignorando outros. Isto é uma forma simples de, por exemplo, ignorar obstáculos em determinados grupos, colocando o peso deste fator a zero nos grupos em questão. Um caso contrário seria aumentar o peso do fator para seguir um caminho, podendo ignorar a influência de outros boids.

A totalidade destas funcionalidades será listada posteriormente, na definição da linguagem XML.

2.2 Estrutura/Implementação

Para cenários complexos, é necessária uma estrutura que permita todos os comportamentos em simultâneo entre vários grupos e boids, sem conflitos no que diz respeito às referências destes mesmos. Como exemplo, se a estrutura que guarda os boids referentes a um grupo for completamente separada dos restantes grupos/boids, seria mais complexo representar algo como:

- Boid A é líder do grupo X;
- Boid A é predador do grupo Y;

Do que se a estrutura for de certa forma partilhada, tendo acesso ao Boid A independentemente do seu papel num determinado grupo.

Para resolver este e outros problemas semelhantes, foi pensada uma estrutura principal que armazenará todos os elementos, não fazendo distinção de grupo nem papel desempenhado, devolvendo apenas elementos consoante o seu id. O mesmo foi adotado no que diz respeito ao armazenamento de caminhos, obstáculos e grupos. Para representar esta estrutura optou-se pela utilização de maps, que associam facilmente uma chave (identificação) ao seu valor. Desta forma temos acesso a toda a informação a partir de ids.

De seguida, são então explicadas as decisões para cada participante do sistema.

2.2.1 Boid:

Começando pela classe que representa um individuo/boid, é guardado um conjunto de parâmetros que permite adequar o seu comportamento a diferentes cenários.

A lista de parâmetros inclui desde propriedades básicas como a distância de visão e o espaço mínimo entre dois boids até propriedades mais complexas que permitem saber se o boid já terminou o caminho que lhe estava atribuido, quantos amigos estão perto, definir a rotação ou aceleração máxima por segundo entre outras.

Como foi referido inicialmente, o objetivo seria não tornar as propriedades limitadas, possibilitando o utilizador mais exigente controlar totalmente comportamentos ou adicionar simples propriedades. Para isto utilizamos um sistema de propriedades extras, que pode ser acedido e executar o que for conveniente perante os seus valores. Um exemplo desta implementação seria o armazenamento em cada boid do modelo a fazer render, acedendo a esta propriedade e escolhendo o modelo perante a mesma. Outro exemplo mais complexo, seria guardar por exemplo, o clube de futebol, nacionalidade ou idade, e caso encontrasse colegas com o mesmo valor ou outro específico, poderia executar um comportamento específico.

A forma como o movimento é calculado é através de funções independentes que representam as diferentes regras, podendo o grupo a que o boid pertence invocar e dar-lhe o peso que desejar na soma de todos os componentes. Isto é o que permite dar prioridades a seguir o líder, a seguir um caminho ou a outros movimentos.

2.2.2 Swarm:

Sendo esta classe a representação de um grupo de boids, é necessário ter acesso a todos boids pertencentes de alguma forma ao grupo em questão. Como foi referido anteriormente, todos os boids são guardados numa estrutura comum, com o objectivo de termos um fácil acesso a todos os boids do "mundo" a partir de qualquer grupo. Após isto, fica claro que esta classe que representa um grupo de boids, precisará de receber apenas os ids destes mesmos. Através destes, por uma questão de performance, ao invés de os colectar sempre que necessário, constrói uma lista de apontadores para os seus elementos, que será utilizada em todos os updates para os cálculos dos movimentos de cada elemento do mesmo.

É nesta classe que o update principal será realizado, invocando para cada boid as funções que representam determinados comportamentos e atribuindo um peso definido pelo utilizador a cada um. No final são somados os resultados e caso exceda a velocidade máxima ou a aceleração permitida, limitasse a mesma. É também possível atribuir um peso a um factor aleatório, que faz com que os elementos hajam apenas pelas regras ou com um certo peso de aleatoriedade nos seus comportamentos. É também no grupo que definimos se os seus elementos se movimentam nas 3 dimensões ou apenas em duas dimensões, sendo isto definido pelo utilizador quando cria um grupo no ficheiro XML.

```
for each Boid boid
v1 = boid->flock(s_boidsptr) * flockWeight;
v2 = boid->disperse(bwolrd->getBoids()) * disperseWeight;
//dispersa de todos, não apenas do swarm a que pertence
v3 = boid->matchVelocity(s_boidsptr) * matchWeight;
.
.
.
Vec3 newVelocity = boid->getVelocity() + (v1 + v2 + v3 + . . . +
randomFactor);
boid->limit_Velocity(newVelocity, maxSpeed, maxAceleration);
boid->updatePosition(deltaTime,dim2D);
```

Figura 4 - Simples representação do ciclo principal

Uma característica importante de um grupo é o tempo de resposta/raciocínio dos seus elementos. Com isto em consideração, este ciclo representado é apenas

executado de x em x tempo, sendo que nas restantes invocações da função de update em que não tenha passado o tempo referido, é utilizada a velocidade calculada na última entrada do ciclo. Isto é, faz-se um update da posição do boid sem calcular uma nova velocidade. Este tempo é também definido pelo utilizador, podendo querer que um grupo tenha reações lentas e um outro reaja rapidamente.

```
if(deltaTimeSinceLastUpdate < updateCycleTime)
{
        for each Boid boid
        {
                boid->updatePosition(deltaTime, dim2D);
        }
}
else
{
        Ciclo representado na figura anterior
}
```

Figura 5 - Complemento do ciclo principal

2.2.3 Caminho/Path:

Para a construção de caminhos foram usadas Catmull-Rom Splines.

Catmull-Rom Splines são splines cúbicas que pegam num conjunto de valores (pontos) para descrever uma curva cúbica suave que passa pelos pontos fornecidos.

Para criar uma curva destas são necessários no mínimo quatro pontos v0, v1, v2 e v3.

$$f(x) = [1, x, x^2, x^3] \times M \times [v0, v1, v2, v3]$$

A equação acima define um novo ponto entre v1 e v2. Este valor depende de t que será o valor usado para interpolar um dos pontos da curva.

O valor de t deve variar entre 0.0 e 1.0. Desta forma se x = 0.0 o valor retornado pela função é v1 e se x = 1.0 o valor retornado é v2.

A matriz M é fixa e é definida por:

$$M = \begin{bmatrix} 0.0 & 1.0 & 0.0 & 0.0 \\ -0.5 & 0.0 & 0.5 & 0.0 \\ 1.0 & -2.5 & 2.0 & -0.5 \\ -0.5 & 1.5 & -1.5 & 0.5 \end{bmatrix}$$

O resultado final pode ser algo como o apresentado na figura abaixo.

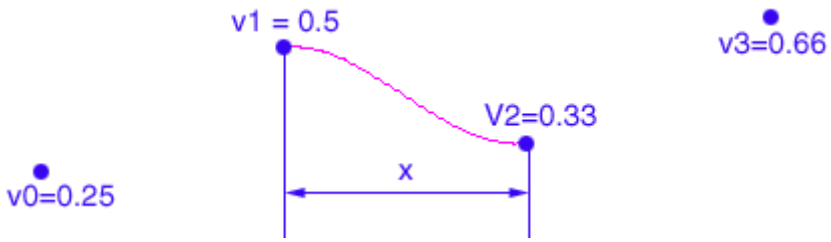

Figura 6 - Resultado do uso de Catmull-Rom para a criação de uma spline

A implementação feita neste projecto tem em conta que as cenas possuem três eixos, logo os cálculos têm que ser feitos para todos os eixos.

```
for(int k = 0;  k < 50; k++){
float t = k*0.02;

c1 =          M12*p1x;
c2 = M21*p0x          + M23*p2x;
c3 = M31*p0x + M32*p1x + M33*p2x + M34*p3x;
c4 = M41*p0x + M42*p1x + M43*p2x + M44*p3x;

pontox = (((c4*t + c3)*t +c2)*t + c1);

c1 =          M12*p1y;
c2 = M21*p0y          + M23*p2y;
c3 = M31*p0y + M32*p1y + M33*p2y + M34*p3y;
c4 = M41*p0y + M42*p1y + M43*p2y + M44*p3y;

pontoy = (((c4*t + c3)*t +c2)*t + c1);

c1 =          M12*p1z;
```

```
c2 = M21*p0z          + M23*p2z;
c3 = M31*p0z + M32*p1z + M33*p2z + M34*p3z;
c4 = M41*p0z + M42*p1z + M43*p2z + M44*p3z;

pontoz = (((c4*t + c3)*t +c2)*t + c1);

Vec3 ponto = Vec3(pontox,pontoy,pontoz);

caminho.push_back(ponto);
}
```

Figura 7 - Ciclo que cria os novos pontos da spline a partir dos fornecidos

O ciclo acima constrói um caminho de 50 pontos entre p1 e p2. O cálculo de c1, c2, c3 e c4 é feito 3 vezes, uma vez para cada eixo do referencial obtendo assim a coordenada x, y e z do novo ponto do caminho.

2.2.4 Paths:

Esta classe serve de container de todos os caminhos, armazenando-os associando um id a um caminho, ou seja, num map<string,Caminho>. As únicas funções que tem são as de adicionar e retornar caminhos.

2.2.5 Obstacle:

Um obstáculo pode ser definido como uma box ou como uma esfera, sendo que o utilizador define como lhe der mais jeito para cada obstáculo. É uma classe simples apenas com as informações referentes aos limites da BOX ou do centro e raio da esfera.

2.2.6 Scene:

Para o armazenamento dos obstáculos é utilizada uma classe Scene, que armazena os obstáculos associados ao seu id. Como os obstáculos são fixos, não alterando de posição e apenas precisámos ter em conta os obstáculos próximos do boid em que os cálculos estão a ser efectuados, foi implementado também um sistema de intersecção através de uma R-Tree que é opcional a sua utilização, ou seja, em paralelo com o armazenamento dos obstáculos, é criada uma R-Tree que associa um id a uma caixa no espaço 3D. Quando queremos saber os obstáculos próximos de um dado boid, é dada a posição deste e o raio a que queremos pesquisar obstáculos, utilizando esta árvore para pesquisar se há intersecções. Se houverem é criada uma lista de ids que será utilizada para retornar os obstáculos que intersectaram a zona pesquisada.

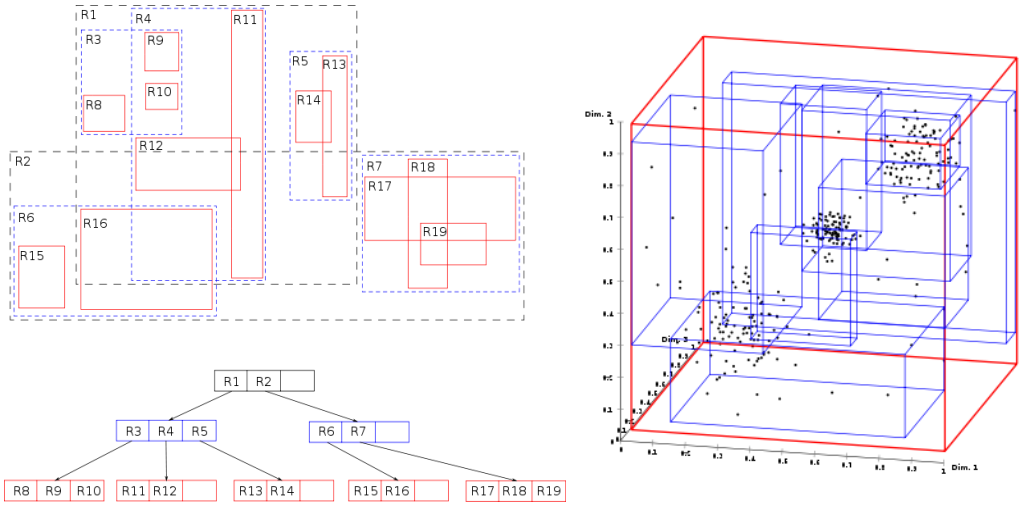

Figura 8 - Esquema de uma R-Tree e representação a 3 dimensões.

2.2.7 BWorld:

Finalmente, a classe BWorld é a que representa a totalidade do mundo, tendo basicamente como objetivo o armazenamento de todos os intervenientes da aplicação e permitir uma conexão entre todos estes. Como armazena tudo baseado em mapas de correspondência entre id (chave) e o elemento referido (valor), facilita então a referência de elementos em diferentes locais. De outra forma que não através de ids seria difícil representar por exemplo a lista de amigos de um dado boid, sendo estes amigos quaisquer boids, independentemente se pertencem ao mesmo grupo ou não.

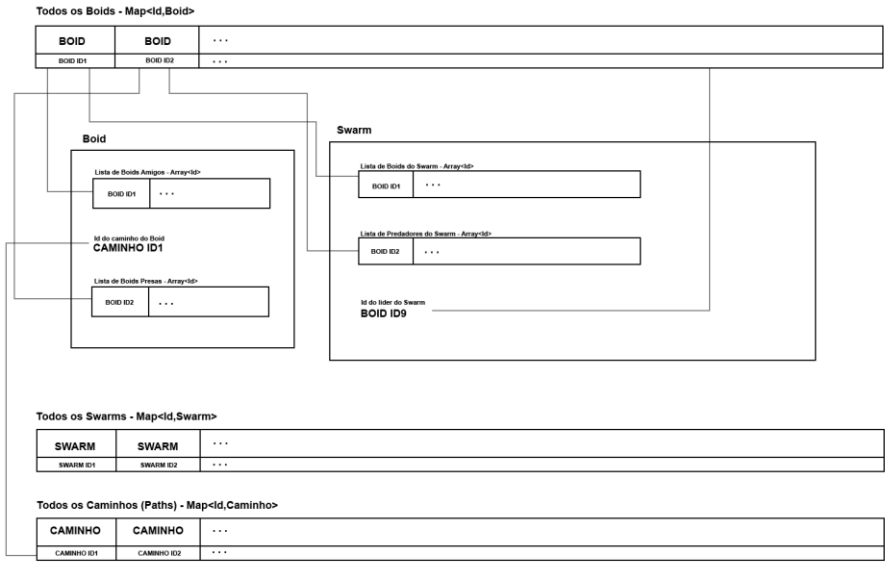

Figura 9 - Esquema das estruturas usadas para armazenar a informaçao do BWorld

Pode ser inicializada através de um construtor vazio, que permitirá o programador adicionar e controlar os diferentes elementos e os seus comportamentos, ou através de um construtor que lhe é dado o caminho de um ficheiro XML, e carregando todas as definições para a aplicação. Útil para definir todo o sistema sem necessitar de olhar para o código em si. Este ficheiro XML deve ser escrito segundo a linguagem definida no capítulo seguinte.

2.3 Linguagem / XML

De forma a criar as cenas o programa carrega um ficheiro XML que descreve os elementos da cena bem como os seus comportamentos.

Nesta secção é explicada a estrutura que um ficheiro deve ter para ser lido corretamente e serão mostradas todas as opções disponíveis e como devem ser utilizadas.

Para começar, de forma a criar uma cena deve ser usada a tag 'World'. Assim, todos os componentes da cena devem estar entre <World> e </World>.

Na criação de Boids é possível criar grupos de Boids ou então criar Boids individuais que podem ser depois adicionados a um grupo. Esta opção existe para casos em que num grupo é necessário que um determinado Boid tenha opções especiais.

Desta forma é necessário ter em atenção que Boids individuais devem ser criados antes do grupo a que pertencem para que não seja possível incluir num grupo Boids que não existem.

Assim sendo o processo de criação de um Boid pode ser algo como:

```
<Boid id="id01">
        <Attribute velocity="0.2" />
        <Attribute acceleration="0.2" />
        <Attribute space="10" />
        <Attribute sight="20" />
        <Attribute steeringDeadZone="20" />
        <Attribute maxSteeringBySecond="100" />
        <Attribute boundary="40" />
        <Attribute centerOfBoundary="(0.0, 0.0 , 0.0)" />
        <Attribute dim="0" />
        <Attribute coord2D="0.2" />
        <Attribute path_id="0" />
        <Attribute property="( model : carroA )" />
</Boid>
```

Figura 10 - Exemplo XML para a criação de um Boid

Como é possível ver um Boid tem que ter um id definido pelo utilizador. Os ids não devem ser apenas números, como por exemplo '10', uma vez que na criação de grupos a distribuição de ids é feita usando inteiros que são sendo incrementados. Logo na criação de um grupo é provável que um dos Boids fique com o id='10'.

Um Boid tem também um conjunto de atributos opcionais:

- **velocity** - Velocidade do Boid.
- **acceleration** - Aceleração do Boid.
- **space** - Zona pessoal do Boid. Nenhum outro boid entra neste espaço.
- **sight** - Visão do Boid. Distancia até onde o Boid consegue ver outros Boids ou obstaculos.
- **steeringDeadZone** - Angulo ignorado na rotação de um Boid de forma a suavizar as rotações. Em graus.
- **maxSteeringBySecond** - Angulo maximo de rotação por segundo.
- **boundary** - Espaço máximo por onde o Boid pode circular.
- **centerOfBoundary** - Ponto central do do boundary.
- **dim** - Dimensão em que o Boid se move. 0 para 3D, 1 para fixar o x, 2 para fixar o y e 3 para fixar o z
- **coord2D** - No caso de x,y ou z estarem fixos, coord2D diz qual o valor da coordenada. Por exemplo x fixo em 10.
- **path_id** - Id do caminho que o Boid irá seguir.
- **property** - Outras propriedades que se queiram atribuir ao Boid.

A criação de grupo de boids (swarm) pode ser algo do tipo:

```
<Group id="g01">
        <Attribute cycleTime="10" />
         <Attribute number="20" />
        <Attribute velocity="4.1" />
        <Attribute acceleration="0.2" />
        <Attribute space="10" />
        <Attribute sight="20" />
        <Attribute steeringDeadZone="20" />
        <Attribute maxSteeringBySecond="100" />
        <Attribute boundary="40" />
        <Attribute centerOfBoundary="(0.0, 0.0 , 0.0)" />
        <Attribute targetPos="(0.0, 0.0 , 0.0)" />
        <Attribute dim="0" />
        <Attribute coord2D="0.2" />
        <Attribute liderId="b05" />
        <Attribute predatorId="b34" />
        <Attribute predatorsGroup="g02" />
        <Attribute add="bobby" />
        <Attribute flockWeight="0.5" />
        <Attribute disperseWeight="0.5" />
        <Attribute matchWeight="0.5" />
        <Attribute boundaryWeight="1" />
        <Attribute followLiderWeight="1" />
        <Attribute followPathWeight="1" />
        <Attribute randomWeight="0" />
        <Attribute avoidObstaclesWeight="1" />
        <Attribute avoidPredatorsWeight="1" />
</Group>
```

Figura 11 - Exemplo XML para a criação de um grupo de Boids (Swarm)

Tal como nos Boids individuais, um grupo tem que ter um id, também definido pelo utilizador.

Quanto a atributos um grupo pode ter:

- **cycleTime** - Intervalo de tempo para a actualização das posições e velocidades dos boids do grupo.

- **number** - número de Boids que vão ser criados para o grupo. Os ids dos Boids criados são dados automaticamente pela aplicação e têm um valor crescente.
- **velocity** - Velocidade dos Boids do grupo;
- **acceleration** - Aceleração dos Boids do grupo;
- **space** - Zona pessoal de cada Boid. Nenhum outro boid entra neste espaço.
- **sight** - Visão de cada Boid. Distancia até onde o Boid consegue ver outros Boids ou obstaculos.
- **steeringDeadZone** - Angulo ignorado na rotação de um Boid de forma a suavizar as rotações. Em graus.
- **maxSteeringBySecond** - Angulo maximo de rotação por segundo.
- **boundary** - Espaço máximo por onde cada Boid pode circular.
- **centerOfBoundary** - Ponto central do do boundary.
- **targetPos** – Ponto do mundo para onde o grupo irá tender.
- **dim** - Dimensão em que os Boids do grupo se movem. 0 para 3D, 1 para fixar o x, 2 para fixar o y e 3 para fixar o z
- **coord2D** - No caso de x,y ou z estarem fixos, coord2D diz qual o valor da coordenada. Por exemplo x fixo em 10.
- **liderId** - Id do lider do grupo. Necessita de ser um Boid individual.
- **predatorId** - Id de um predador do grupo. Necessita de ser um Boid individual.
- **predatorsGroup** - Id de um grupo em que todos os Boids são predadores deste grupo.
- **add** - Id de um Boid individual que queira ser incluido neste grupo.
- **flockWeight, disperseWeight, matchWeight, followLiderWeight, followPathWeight, ramdomWeight, avoidObstaclesWeight, avoidPredatorsWeight, targetWeight** - Pesos de cada funcionalidade que contribui para a direção e velocidade dos boids do grupo.

Outra funcionalidade que é possível utilizar são os caminhos.

Os caminhos são criados e devem depois ser atribuídos a Boids individuais. Mais uma vez é necessário ter em atenção que o caminho deve ser criado antes do Boid individual de forma a evitar a atribuição de caminhos que não existem a um Boid.

Um caminho usa a tag 'Path' e tem a seguinte forma:

```
<Path id="0">
        <Attribute point="( 35.0 , 0.0 , -35.0 )"/>
        <Attribute point="( 35.0 , 0.0 , 35.0 )"/>
        <Attribute point="( -35.0 , 0.0 , 35.0 )"/>
        <Attribute point="( -35.0 , 0.0 , -35.0 )"/>
        <Attribute point="( 35.0 , 0.0 , -35.0 )"/>
        <Attribute point="( 35.0 , 0.0 , 35.0 )"/>
</Path>
```

Figura 12 - Exemplo XML para a criação de um caminho

Cada caminho deve ter um id atribuído pelo utilizador.

Como atributos deve ter uma lista de pontos. Atenção que para que um caminho seja criado são necessários no mínimo quatro pontos e como são usadas Catmull-Rom Splines para criar as curvas, o caminho apenas existe entre o segundo e o penúltimo ponto.

Atributos:

- **point** - ponto pertencente a caminho.

Outra das opções disponíveis é o facto de que Boids podem ter uma lista de outros Boids ao qual irão reagir se estes entrarem no seu espaço.

A esta relação é atribuída a tag 'Friends' e pode ser criada da seguinte forma:

```
<Friends A="id02" B="id03" />
```

Figura 13 - Exemplo XML para a criação de uma relação unidirecional entre dois Boids

Aqui A e B são ids de Boids, logo só podem ser criadas ligações destas entre Boids individuais. Mais uma vez os Boids da relação têm que ser criados antes da relação em si.

Esta relação é também unidirecional, ou seja, o Boid de com id 'id02' tem na lista de friends o Boid com id 'id03' mas o mesmo não acontece para o Boid de id 'id03'.

Para criar relações bidirecionais é necessário fazer:

```
<Friends A="id02" B="id03" />
<Friends A="id03" B="id02" />
```

Figura 14 - Exemplo XML para a criação de uma relação bidirecional entre dois Boids

Desta forma é deixado em aberto este tipo de relação podendo ser criados diversos comportamentos atreves dela.

Por fim é possível também colocar obstáculos na cena. Estes obstáculos são fixos, ou seja, nunca se movem, independentemente do que os Boids possam fazer.

Para criar um obstáculo é usada a tag 'Obstacle'. Podem ser criados obstáculos esféricos ou cúbicos (paralelepipedos).

Um obstáculo cúbico tem a forma:

```
<Obstacle id="2" min="(-10.0 , -10.0 , -10.0)" max="(10.0 , 10.0 , 10.0)" />
```

Figura 15 - Exemplo XML da criação de um obstáculo cúbico

Um obstáculo esférico tem a forma:

```
<Obstacle id="2" center="(-10.0 , -10.0 , -10.0)" radius="10" />
```

Figura 16 - Exemplo XML da criação de um obstáculo esférico

Como pode ser visto, cada obstáculo tem um id atribuído pelo utilizador.

No caso de obstáculos cúbicos os atributos são:

- **min** - ponto inferior esquerdo (atrás).
- **max** - ponto superior direito (frente).

No caso de obstáculos esféricos os atributos são:

- **center** - ponto centrar da esfera.
- **radius** - raio da espera.

3 Conclusão

Neste projecto foi apresentada uma solução para a atribuição de comportamentos a personagens virtuais através de ficheiros XML. Foram criados comportamentos básicos que podem ser combinados de forma a fazer outros mais complexos.

Este projecto tem o potencial de crescer ainda mais com a inclusão de novos comportamentos que não foram incluídos neste trabalho. Assim, este oferece uma base com que qualquer outra pessoa interessada possa trabalhar.

4 Referências

- http://www.lighthouse3d.com/tutorials/maths/catmull-rom-spline/
- http://www.red3d.com/cwr/boids/
- http://www.vergenet.net/~conrad/boids/index.html
- http://www.cs.toronto.edu/~dt/siggraph97-course/cwr87/
- http://en.wikipedia.org/wiki/R-tree
- https://github.com/nushoin/RTree