IA nos Videojogos

Geração conteúdo

José Veríssimo Lamas nº2017259895

1 Introdução

Neste trabalho vai ser explorada a implementação do método *Procedual Generation* [1] na geração de conteúdos dos videojogos, mantendo os jogos dinâmicos e diferentes para o jogador explorar. Este método de inteligência artificial não só é usado em videojogos como também em filmes e geração de música eletrónica.

Procedual Content Generation [7] é um método de criação de informação, própria para videojogos, automática combinando algum fator aleatório criado pelo computador com o informação e regras criadas por programadores de forma a poder gerar o conteúdo pretendido pelo programador.

A sua implementação teve como inspiração o jogo de tabuleiro *Dungeons & Dragons*, um jogo onde o *game master* cria cenários, *dungeons* e um mundo no qual os jogadores vão explorar. Com base neste conceito, o sistema tem como objetivo a criação automática de conteúdos como salas, corredores e objetos respeitando regras criadas pelo programador de forma a criar um jogo complexo de uma forma mais rápida e eficaz.

A utilização de geração procedimental já é usada em videojogos desde os anos 80 com o exemplo de *Beneath Apple Mano* (1978) [2] ou *Rouge* (1980) [3] para a criação de *dungeons* com salas, corredores, desafios, inimigo e até tesouros.

2 Pseudorandom number generator

Um algoritmo bastante usado e simples na geração de conteúdo é o Gerador de números pseudo-aleatórios [4]. A sua implementação em jogos consiste na ideia de ter uma grande quantidade de conteúdos com pouco custo de memória.

Este algoritmo produz uma sequência de números parcialmente aleatórios. Os resultados obtidos não são realmente aleatórios dado que são determinados por um valor inicial atribuído, a *seed* [5]. A predefinição deste valor determina os valores gerados, podendo garantir a reprodução dos mesmos conteúdos usando sempre a mesma *seed*, passando a guardar na memória apenas a *seed* e não a grande quantidade de informação que os diversos conteúdos iriam ocupar.

Usando esta estratégia temos o exemplo do jogo *The Sentinel* (1986) [6] que consegue ter aproximadamente 10 000 níveis diferentes guardados em 48 ou 64 kilobytes (KB).

3 Geração de *dungeons* e níveis [8]

. Uma *dungeon* num jogo é parecido a um labirinto onde o jogador tem a oportunidade de as explorar enfrentando diversos desafios como inimigos, armadilhas ou puzzles na busca de recompensas. Um método de geração de uma *dungeon* eficaz pode basear-se nas seguintes estratégias.

3.1 Algoritmo de partição de espaço

. Para a criação do espaço de uma *dungeon* é usado um algoritmo de partição de espaço (*space-partitioning algorithm*) que irá dividir um espaço em vários *subsets* de espaços. Um algoritmo bastante usado é a *binary space partitioning* (BSP). A *dungeon* vai ser representada pela raiz da árvore da BSP e será repartida até uma certa condição ser cumprida, como um número definido de salas que a *dungeon* deve ter. Após a divisão, em cada partição serão escolhidos dois pontos aleatórios de forma a criar uma sala unindo as várias salas numa abordagem *bottom-up*.

3.2 *Agent-Based approach*

. A abordagem baseada num agente geralmente usa um único agente para criar caminhos e salas em sequência. Ao contrário de uma abordagem de partição de espaço, esta abordagem irá ter resultados menos organizados e até mesmo podendo existir colisões entre caminhos ou salas, ou podendo criar a *dungeon* toda apenas num canto do espaço disponível, havendo a necessidade de existir alguma tentativa-erro para obter uma boa *dungeon*. A aparência da *dungeon* depende do comportamento do agente, se este usar um método estocástico vai resultar uma *dungeon* muito caótica enquanto um agente com *look-head* poderá evitar as colisões.

4 Geração de terreno [8]

Muitos jogos modernos, principalmente os de três dimensões, contêm terrenos mais complexos e completos que um simples plano. Existem diversas abordagens geralmente usadas como métodos estocásticos, método de *Perlin-noise* e métodos baseados no uso de agentes para uma construção mais complexa.

A criação do terreno depende bastante do jogo e da forma como o jogador interage com este. Num dos extremos temos o terreno em jogos de simulação de voo, onde o terreno está maioritariamente como fundo e não é suposto interagir com o chão, a única possível interação seria a colisão mas pode ser generalizado a quando se atinge a altitude zero conclui-se que é uma colisão com o chão, dado que não haverá interação com o terreno este tem apenas o objetivo de ser visualmente atraente. No outro extremo temos jogos onde o terreno serve para restringir as ações dos jogadores como nos jogos *first-person shooter* onde a geração do terreno deve ser mais cuidadosa e semelhante aos níveis de plataforma.

Um conceito bastante usado na geração de terrenos é o ruído. O ruído na geração de terrenos não serve para deformar, mas serve para criar pequenas variações na superfície de forma a esta ter um aspeto mais real e completo, tendo como exemplo o céu, onde há ruídos de nuvens para apresentar um efeito mais completo, aleatório e realista.

Grande parte dos aspetos dos terrenos podem ser representados em matrizes bidimensionais de números reais onde a altura e a largura do mapa são transformados nas dimensões x e y de uma superfície retangular. Essa matriz é geralmente chamada *heightmap*, onde o valor de cada célula corresponde à altura do terreno nessas coordenadas. A altura do terreno que está contido entre dois pontos será a interpolação entre a altura desses dois pontos.

A geração de terreno de forma aleatória não é tão simples como usar um valor de altura aleatório na *heightmap*, um terreno assim construído apresenta um resultado inutilizável devido aos seus picos aleatórios.

No terreno a altura de cada ponto depende da altura dos pontos próximos. Existem diversos métodos que tentam resolver este problema, mas o seu objetivo inicial era para a geração de texturas dado que o mesmo problema se apresentava, mas com a cor gerada em cada ponto da textura.

Uma maneira de resolver o problema é criando terreno suave. A criação deste terreno mais suave pode ser feita usando *interpolated noise* onde apenas são gerados valores de altura aleatórios em apenas alguns pontos, interpolando os restantes pontos consoante os pontos com valor gerado mais próximos. Há várias maneiras de calcular a interpolação sendo duas delas *bilinear interpolation* e *bicubic interpolation*.

- O método *bilinear* obtém a altura de um ponto através da média dos pontos em dois eixos: direção horizontal (eixo x) e vertical (eixo y). Algumas das desvantagens do uso deste método são os declives das montanhas serem perfeitamente lineares e o pico das montanhas ser sempre pontiagudo.
- O método *bicubic* tenta resolver o problema dos picos da montanha dando-lhe uma forma em S, inicialmente o declive aumenta lentamente ficando mais íngreme á medida que se "sobe" na montanha e acabando com um topo arredondado. Para obter este efeito calcula-se a altura dos pontos geralmente usando a função não-linear s(x) = -2x^3 + 3x^2 (onde x é a distância, de 0 a 1, a que está entre os pontos mais próximos e s(x) a altura a que o ponto se deve encontrar).

4.1 *Gradient-based random terrain*

Um método alternativo é o *gradient-based random terrain* que gera os declives e obtém a altura dos pontos através do valor do declive. Os números que vão ser gerados pseudo aleatoriamente vão ser interpretados como gradientes aleatórios, por exemplo através da direção do declive, neste caso o *array* inicializado passa a chamar-se *gradient noise* ou *Perlin noise*.

Gerar gradientes em vez do valor da altura tem vantagens:

- dado que estamos a fazer a interpolação entre gradientes vai haver suavização não da altura, mas do rácio da altura (a forma como a altura muda) obtendo um efeito ainda mais suave;
- visto que os picos das montanhas não são gerados diretamente num dos pontos da matriz, mas são gerados dependendo da variação dos declives, obtemos resultados mais "naturais".

4.2 Criação de terreno usando uma abordagem baseada em agentes.

O uso de algoritmos *agent-based* oferecem um maior controlo sobre a forma e como o terreno está a ser gerado. São usados diversos agentes para a construção do terreno, nos casos em que é possível dividir o terreno em diversas áreas, são usados geralmente um agente para cada uma das áreas, por exemplo, na construção de uma cidade é possível usar agentes para cada um dos setores da cidade (como áreas residenciais, industriais, comerciais, etc) e usar outros agentes para a criação das estradas e da ligação entre os diversos setores.

5 Gramatica L-Systems [8]

Após a geração de terreno algo que habita em grande parte dos terrenos nos videojogos são plantas, como árvores e arbustos. Se só houver um modelo de árvores a ser utilizado os jogadores notam logo dando ao ambiente um ar muito repetitivo e monótono. Uma das melhores maneiras de conseguir diversificar estes modelos é gera-los através da gramática formal *L-System*.

Uma gramática formal é uma produção de regras de modo a transformar um conjunto de símbolos num outro símbolo. Cada regra é apresentada no formato A -> BC. O uso da gramática consiste em quando se encontra o(s) símbolo(s) da direita, podemos transformá-lo no símbolo da esquerda ou vice-versa.

L-Systems é uma classe de gramática onde a reescrita dos símbolos ocorre simultaneamente. Introduzida pelo biólogo Aristid Lindenmayer em 1968 com o modelo de crescimento de um sistema orgânico, como as plantas:

- A -> AB
- B -> A

Pode ser gerada uma sequência como a seguinte:

- A
- AB
- ABA
- ABAAB
- ABAABABA

- ABAABABAABAAB
- ABAABABAABAABABAABABA

A gramática *L-System* apesar de ser simples, demonstra a possibilidade de resultados regulares e complexos podendo assim ser usado eficazmente na produção de conteúdo. Uma das formas de se usar a *L-System* pode ser em instruções *turtle*, para que esta desenhe por onde anda. Considerando os seguintes símbolos:

- F: andar para a frente (por exemplo: 10 pixels)
- +: rodar 90 para a esquerda
- -: rodar 90 para a direita

e considerando apenas uma regra simples como:

- F -> F + F − F − F + F

podemos construir figuras gráficas complexas rapidamente.

Apesar de podermos desenhar figuras complexas, a *turtle* desenha sempre por onde anda não sendo possível andar sem desenhar. Para este problema são introduzidos mais dois símbolos '[' e ']', onde o símbolo '[' acrescenta a posição atual da *turtle* numa *stack* (pilha) e o símbolo ']' tira a posição da *stack* e coloca lá a *turtle* possibilitando a construção de figuras mais complexas.

5.1 Geração de missões e espaços com a gramática *L-System*

Geralmente as gramáticas usam *strings* para operar, mas não estão limitadas a isso, podendo usar estruturas diferentes como grafos ou *tile maps*. Para a geração de missões ou espaços os grafos são as estruturas mais usadas para que seja possível construir um certo nível de complexidade. Para a transformação da gramática dos grafos é necessário ter em atenção cinco aspetos como apresentados na figura 1:

- identificar o grafo que pertence ao lado esquerdo da regra e marcá-lo
- remover todas as arestas entre os nós marcados
- transformar o grafo transformando os nós marcados nos nós correspondentes do lado direito
- copiar as arestas dos nós do lado direito
- remover as marcações

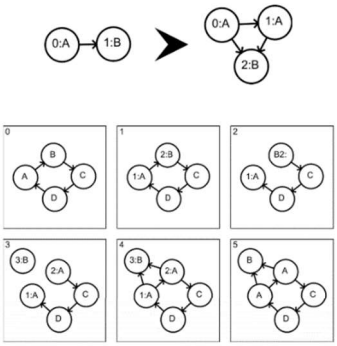

Fig. 1. [8]: Gramática de um grafo e um exemplo

6 Conclusão

Neste trabalho foi refletido alguns métodos e utilizações mais comuns do *Procedual Content Generation* (PCG), responsável por evitar a construção de conteúdo manualmente construindo uma experiência agradável para o jogador.

A implementação de PCG está presente em todos os jogos atuais e até em grande parte de jogos clássicos dado há sua variedade de técnicas, métodos e algoritmos capazes de gerar quase qualquer tipo de conteúdo desejado num jogo, desde uma simples *dungeon* para ser explorada a sistemas de geração de missões completas e complexas do qual o jogador pode desfrutar.

Grande parte das técnicas refletidas não são muito trabalhosas de implementar, mas são rápidas e eficazes a alcançar os seus objetivos.

7 Referências

[1] "Procedural generation", *En.wikipedia.org*, 2020. [Online]. Available: https://en.wikipedia.org/wiki/Procedural_generation#External_links. [Accessed: 19- Jun- 2020].
[2] "Beneath Apple Manor", *En.wikipedia.org*, 2020. [Online]. Available: https://en.wikipedia.org/wiki/Beneath_Apple_Manor. [Accessed: 19- Jun- 2020].
[3] "Rogue (video game)", *En.wikipedia.org*, 2020. [Online]. Available: https://en.wikipedia.org/wiki/Rogue_(video_game). [Accessed: 19- Jun- 2020].
[4] "Pseudorandom number generator", *En.wikipedia.org*, 2020. [Online]. Available: https://en.wikipedia.org/wiki/Pseudorandom_number_generator. [Accessed: 20- Jun- 2020].
[5] "Random seed", *En.wikipedia.org*, 2020. [Online]. Available: https://en.wikipedia.org/wiki/Random_seed. [Accessed: 20- Jun- 2020].
[6] "The Sentinel (video game)", *En.wikipedia.org*, 2020. [Online]. Available: https://en.wikipedia.org/wiki/The_Sentinel_(video_game). [Accessed: 20- Jun- 2020].
[7] "What Pcg Is - Procedural Content Generation Wiki", *Pcg.wikidot.com*, 2020. [Online]. Available: http://pcg.wikidot.com/what-pcg-is. [Accessed: 19- Jun- 2020].
[8] N. Shaker, J. Togelius and M. Nelson, *Procedural Content Generation in Games*. Cham: Springer, 2016.