APLICAÇÃO DE *PROCEDURAL CONTENT GENERATION* A JOGOS: DEFINIÇÃO, CONCEITOS E PARADIGMAS

Rodrigo Friedrich (IC)

Prof. Dr. Luciano Silva e Prof. Ms. Pedro Henrique Cacique Braga (Orientadores)

Apoio PIVIT CNPq e Mackenzie

Resumo:

O crescente interesse da indústria de jogos digitais e consumidores por jogos gerados de forma procedural (*Procedural Content Generation*, ou PCG) leva os desenvolvedores a buscar as possíveis formas de implementar essas técnicas em seus jogos. O artigo, além de introduzir o leitor às motivações e origens das técnicas, expõe as principais formas de geração procedural de conteúdo, seus respectivos paradigmas e diversos conceitos, os problemas que a abordagem procedural pode resolver e as aplicações gráficas e mecânicas de diversas técnicas procedurais. Esses métodos aqui citados podem ser aplicados tanto na mecânica e regras de jogo em si, durante seu desenvolvimento e execução do jogo, ou em aspectos visuais como terrenos, folhagem e afins. Também é traçado um comparativo entre as técnicas e diversos exemplos de jogos que já as implementam são citados.

Palavras chave: Jogos, procedural, PCG

Abstract:

The growing interest from the games industry and consumers for procedurally generated content (PCG) drives developers to seek ways to implement PCG techniques into their games. The article describes the main ways of procedural content generation, their respective concepts and many paradigms, the problems that the procedural approach may solve, as well as graphical and mechanical applications of PCG. The methods shown may be applied to the game mechanics and rules (be it during production or just before a game starts); or to the visual elements of the game like terrain, landscape, foliage and many others. The article also compares many of the PCG techniques, and exemplifies with games that already implement them.

Keywords: games, procedural, PCG

1. Introdução

Com o crescimento da indústria de jogos digitais, surgiu a necessidade de controlar gastos, uma vez que o conteúdo dos jogos é cada vez mais detalhado e robusto. O custo de desenvolvimento de jogos grandes, os chamados “AAA” (*triple A,* em inglês) tem chegado a níveis exorbitantemente altos, como o jogo Destiny (2014, Activision) que custou cerca de 500 milhões de dólares (IGN.com, 2014 [1]) e empregou mais de 500 profissionais no projeto. A maioria dos desenvolvedores de jogos não possui tal orçamento e o desenvolvimento dos jogos deve ser menos custoso.

Uma solução que permite que jogos criem uma quantidade alta de conteúdo por um preço baixo e mantendo a qualidade pode ser a geração de conteúdo de forma procedural (*Procedural Content Generation*, ou PCG). A redescoberta da técnica pela indústria de games deve-se, principalmente, à palestra do respeitado desenvolvedor Will Wright, em sua palestra “The Future os Content” na *Game Developers Conference* de 2005 [2]. Na palestra, além de apresentar seu novo jogo, Spore (que utiliza técnicas de PCG), Will argumenta que o conteúdo de jogos pode ser criado de forma mais barata por meio de métodos procedurais, quando comparado ao mesmo volume de conteúdo produzido por um grupo de profissionais especializados da área.

1.1. Referencial teórico

O professor Julian Togelius da New York University (NYU), em suas publicações, explora as definições e conceitos das técnicas PCG , em contraste com as outras formas tradicionais de criação de conteúdo.

A professora Rilla Khaled do Department of Design and Computation Arts da Concordia University, é responsável pela definição das metáforas do design associado ao PCG, o que auxilia na compreensão do escopo que um sistema PCG pode tomar.

1.2.Origens no design de jogos

Hoje é comum encontrar jogos digitais de altíssima performance gráfica, com física realista e um volume grande de dados armazenados nos discos de DVD ou *Blu-Ray*. No fim da década de 1970, porém, grandes espaços de armazenamento como os de hoje não existiam e os jogos tinham que ocupar um espaço pequeno na memória do computador. Esse fato impedia que *game designers* criassem um número grande de fases em seus jogos, limitando a capacidade de jogar sem que o jogo se tornasse repetitivo.

Foi então que surgiram os jogos *roguelike*, que geram fases semialeatórias baseadas em um código relativamente curto. Todos os aspectos do jogo (exceto as regras) eram gerados pelo código, por meio de técnicas de PCG: salas, corredores, monstros, tesouros que o jogador encontra (Figura 1.2). No jogo Rogue, todos esses elementos eram baseados na tabela de códigos de caracteres American Standard Code for Information Interchange (ASCII). O estilo *roguelike* ainda é comum hoje, pois não necessita de horas gastas na criação de fases e cenários complexos e detalhados e é bastante utilizado por estúdios de desenvolvimento pequenos.

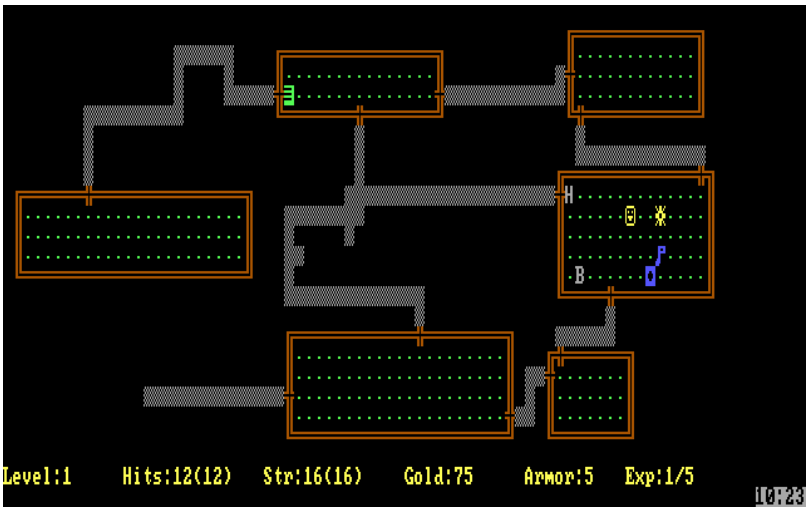

Figura 1.1: Imagem do jogo Rogue, que batizou o gênero *roguelike*.

Fonte: http://www.dosgamesarchive.com/download/rogue/

1.3.Definição de PCG

> "It would be futile to hope to come up with a definition of procedural content generation in games that everybody agrees on. PCG has been attempted by too many people with too many different perspectives for this to happen. A graphics researcher, a game designer in the industry and an academic working on artificial intelligence techniques would be unlikely to agree even on what 'content' is, and much less which generation techniques to consider interesting"
>
> *Togelius et al., 2011[3]*

Como é explicado por Togelius *et al.* [3], definir o que é PCG não é uma tarefa fácil. As diferentes perspectivas de diferentes autores impossibilitam uma definição exata. Muitas delas têm alguns elementos em comum e o mais recorrente é: a geração do conteúdo deve ser feita por meio de código com pouca ou nenhuma interferência do usuário, como definido em [3]. Em outras palavras, PCG pode ser descrito como um software que consegue criar conteúdo de jogos sozinho ou com a participação de jogadores e designers.

Esse conteúdo, no entanto, não é simplesmente todo e qualquer aspecto do jogo. Para Shaker *et al.* [4], numa abordagem PCG, conteúdo de um jogo são: fases, mapas, regras do jogo, texturas, histórias, itens, aventuras (*quests*), música, armas, veículos, personagens etc. O motor gráfico do jogo não conta como um tipo de conteúdo, assim como o comportamento de personagens que não são controladas pelo jogador, os *non playable characters (NPC),* uma vez que dentro do campo de inteligência artificial já existe muita pesquisa que não envolve as técnicas de PCG .

Para Khaled *et al.[5]* é possível descrever o comportamento de sistemas PCG quando em conjunto com o trabalho de um designer em quatro metáforas:

- Ferramenta: um sistema ou algoritmo PCG pode ser compreendido como uma ferramenta ou um dispositivo manipulado com o intuito de alcançar um objetivo no game design, que aprimora as habilidades de um designer
- Materiais: são “substâncias” dinâmicas, reconfiguráveis e geradas “proceduralmente” que podem ser manipuladas e moldadas pelo game designer para alcançar um objetivo
- Designers: são algoritmos ou sistemas PCG que são responsáveis por resolver problemas de game design e executar tarefas de design com pouca ou nenhuma intervenção de um designer humano. Designers PCG contrastam com ferramentas ou materiais, que dependem diretamente da ação de um designer (humano ou PCG). Inclusive, um designer PCG pode manipular outras ferramentas ou materiais PCG.
- Expert: Um algoritmo ou sistema PCG Expert pode ser dividido em Player Expert e Domain Expert. Ambos analisam e interpretam os estados do jogo e, normalmente, são empregados em *serious games*. Enquanto o Player Expert é responsável pela informação referente às ações do jogador, o Domain Expert manipula informações dos outros aspectos do jogo, como mapas e terrenos. Geralmente, Experts não requerem interação com designers.

Essas metáforas auxiliam na compreensão da capacidade de um algoritmo PCG e ilustram a flexibilidade e escopo que as técnicas podem alcançar. A versatilidade das técnicas PCG foi o que motivou Will Wright a descrevê-las como o “futuro do conteúdo”.

1.4. Por que usar PCG?

Como ilustrado anteriormente, a quantidade de funcionários e custo dos jogos modernos é cada vez maior e o PCG serve como alternativa, já que funcionários humanos são caros e comparativamente lentos[17]. Esse alto custo torna os projetos menos lucrativos e mina a capacidade da indústria de criar jogos inovadores e diversos, por medo de prejuízo. Se um

estúdio desenvolvedor consegue cortar gastos de produção, de forma mais rápida e barata e ainda mantendo a qualidade, ele tem uma clara vantagem competitiva.

Isso não significa que o designer humano deve – ou virá a ser – substituído por algoritmos PCG. As técnicas de geração de conteúdo podem aumentar a capacidade criativa de designers humanos, sendo ferramentas na construção de conteúdo único e/ou inovador. Um sistema PCG pode, inclusive, alcançar volumes de conteúdo inimagináveis aos criadores humanos. O principal exemplo é o jogo *No Man's Sky*, que possui cerca de 18 quintilhões de planetas[6]. A produção "manual" desse número de planetas é literalmente impossível.

2. Exemplos de aplicações de PCG

Uma vez compreendidas as características de um algoritmo ou sistema PCG, é possível analisar implementações de PCG de forma mais profunda. Nesse tópico são listados alguns breves estudos de caso de jogos que já implementam técnicas de geração procedural e, posteriormente, são listados os paradigmas que regem as aplicações dessas técnicas. Os jogos listados implementam de forma inovadora as técnicas de PCG e são exemplos de qualidade em game design aliado ao PCG. Os jogos são: *No Man's Sky, Minecraft e Spelunky.*

2.1. No Man's Sky

No Man's Sky é um jogo de exploração espacial, coleta de recursos e troca. Todo o universo do jogo é criado de forma procedural. Todos os aspectos gráficos como animais (Figura 2.1), terrenos, vegetação e alguns aspectos mecânicos como planetas e galáxias são criados por algoritmos procedurais. Ao todo, o jogo consegue gerar 18.446.744.073.709.551.616 planetas distintos [7].

O jogo utiliza sistemas de Lindenmeyer, sistemas de gramática formal e geradores de números pseudoaleatórios para criar um valor *seed* (definido no tópico 3.3) que, após ser processado pelo motor do jogo, sempre gera o mesmo resultado, permitindo que os diversos jogadores visitem o mesmo planeta diversas vezes, sem que haja alterações físicas nele. Esse resultado pode ser qualquer conteúdo dentro do espaço virtual.

Figura 2.1: Exemplo de animais gerados proceduralmente pelo jogo.

Fonte: http://spaceandtim.es/etc/just_add_water

2.1. Minecraft

Minecraft é um jogo de sobrevivência que utiliza um gerador de números pseudoaleatórios, assim como *No Man's Sky*, para gerar o mundo e permitir que, a partir desse valor *seed*, seja possível voltar a ele com pouca ou nenhuma alteração. O jogo utiliza o ruído de Perlin (abordado no tópico 4) para gerar os diferentes níveis de terreno e distribuir objetos pela superfície do mundo.

Além da altura do terreno, o jogo também cria diversos biomas [8] baseados em aspectos como distância horizontal do mar, diferença do nível do mar, pluviosidade, temperatura e altitude. Os elementos gráficos do jogo respondem a esses fenômenos. No topo de uma montanha muito alta, por exemplo, é comum encontrar neve e perto de rios e lagos normalmente há uma grande quantidade de vegetação. O sistema também é capaz de alterar sutilmente a tonalidade da folhagem com base na temperatura e pluviosidade do local. (Figura 2.1)

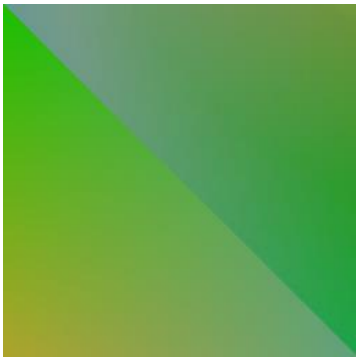

Figura 2.1: O jogo busca na imagem acima a tonalidade desejada para determinado bioma baseado na temperatura e níveis de chuva do ambiente. Essa imagem serve como um *template* para todas as cores verdes de plantas [8].

Fonte: http://minecraft.gamepedia.com/Biome

2.1. Spelunky

Spelunky é um jogo *roguelike* 2D de aventura e exploração, onde o protagonista deve chegar ao fim de fases geradas de forma procedural e coletar o máximo de tesouros e joias. Durante a geração da fase, ela é dividida em um *grid* de 4x4 quadrados (Figura 2.1.3), onde o algoritmo posiciona o chão, paredes, obstáculos, inimigos e prêmios. Antes do posicionamento, porém, o algoritmo gera um caminho entre a “sala” de entrada e a de saída.

Spelunky é um ótimo exemplo de combinação do conteúdo procedural e autoral [9]. As salas geradas seguem um modelo criado pelo designer e o algoritmo se encarrega de gerar variações desse modelo e posicionar os itens coletáveis.

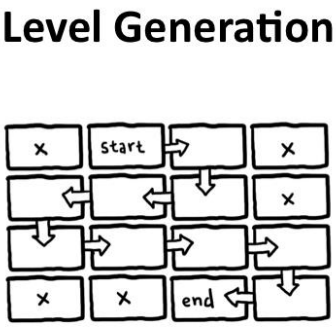

Figura 2.1: O processo de geração de uma fase de *Spelunky* [9].

Fonte: http://makegames.tumblr.com/post/4061040007/the-full-spelunky-on-spelunky

3. Paradigmas de PCG

A variedade intrínseca aos conceitos de PCG resulta em diversas formas de implementação e pode fazer com que desenvolvedores tenham dúvidas sobre qual a melhor abordagem para a resolução de problemas. É necessário, então, definir os paradigmas que regem tais implementações. Togelius *et al.* descrevem diversos paradigmas em [10] e o conteúdo é revisitado posteriormente por Shaker *et al.* em [4].

Os paradigmas são expostos em formato comparativo, uma vez que possuem implementações que podem ser compreendidas como contrárias. Temos, então, os paradigmas PCG de Shaker *et al.*:

3.1. *Online versus offline*

A geração do conteúdo por um algoritmo PCG pode ser feita em dois momentos. Se o conteúdo é gerado enquanto o jogador utiliza o ambiente virtual, gerando infinitas variações ou permitindo que o jogo se adapte às ações do jogador de forma dinâmica, o algoritmo PCG é chamado de *Online.*

Do contrário, se o algoritmo faz sua função antes do jogo começar ou até mesmo durante o processo de desenvolvimento do software, ele é chamado de *Offline*.

Minecraft e *No Man's Sky* implementam técnicas PCG *offline*, mesmo intuitivamente parecendo o contrário. O valor *seed* do cenário já está pré-determinado, o jogo só renderiza os objetos enquanto o jogador avança. Um bom exemplo de geração PCG online é o aclamado jogo *mobile Downwell [12]*, que altera o cenário de forma procedural e infinita enquanto o jogador avança pelo cenário.

3.2. Necessário *versus* Opcional

Um algoritmo PCG pode ser utilizado para gerar conteúdo que é necessário para que uma fase ou o próprio jogo seja completo. A delegação da geração de conteúdo essencial para um jogo para um algoritmo PCG o caracteriza como necessário. Todo e qualquer resultado vindo desse algoritmo deve ser sempre correto.

O PCG pode, porém, gerar aspectos secundários de um jogo, como algumas texturas, uma casa ou vilarejo que pode ser ignorado pelo jogador etc. Se esse conteúdo pode ser substituído posteriormente ou excluído sem perda da lógica do jogo, o PCG é chamado de opcional.

Spelunky é exemplo de ambos os paradigmas, uma vez que a rota que o jogador deve seguir para chegar ao fim da fase é gerada de forma procedural e é necessária para que esta seja completa. Há aspectos opcionais também, como alguns tesouros espalhados pelo mapa, que podem ser ignorados pelo jogador sem prejudicar o avanço da fase – mesmo o jogador almejando conseguir o máximo de pontos que provém desses tesouros, ter uma pontuação baixa não impede que o jogo seja jogado.

3.3. *Seeds* aleatórios *versus* vetor de parâmetros

É importante saber até que ponto o algoritmo gerador pode ser parametrizado. Quase todos os algoritmos PCG geram algum tipo de conteúdo "expandido", baseado em um valor de *input*.

A distinção entre os paradigmas ocorre na forma que o algoritmo recebe tal entrada, seja ele um valor *seed* semialeatório ou um vetor multidimensional altamente parametrizado.

Um *seed* é uma sequência de números semialeatórios usados como valor de entrada ou saída de um algoritmo qualquer (podendo ou não ser um algoritmo PCG).

Como já mencionado, *Minecraft* utiliza seeds semialeatórios para gerar os cenários.

3.4. Genérico *versus* adaptável

Se o conteúdo do jogo, ao ser gerado, não leva em consideração o comportamento do jogador, pode-se dizer que foi proveniente de um PCG genérico. Do contrário, ele é adaptável. É importante ressaltar que a adaptabilidade de um sistema PCG pode ter implicações grandes no campo de interação humano-computador (IHC) [11].

Os três jogos citados no tópico 2 são genéricos, pois não alteram nenhum aspecto em decorrência das ações do jogador.

O jogo *Flow* [16], utiliza dificuldade adaptável e algumas técnicas PCG

3.5. Estocástico *versus* determinista

A utilização do PCG determinista permite que, a partir de uma entrada no algoritmo (como um *seed* ou vetor multidimensional), seja possível alcançar o mesmo conteúdo criado. O oposto é o sistema PCG estocástico, que não permite que o resultado seja o mesmo, a partir do mesmo ponto de partida. *No Man's Sky* e *MInecraft* são deterministas, já que seus mundos podem ser gerados para outros jogadores que tiverem o mesmo *seed*.

3.6. Construtivo *versus* Gerar e testar

Se o conteúdo é gerado uma única vez e não são permitidas modificações, o sistema PCG utilizado é construtivo. Em contrapartida, se o conteúdo a ser gerado fica em loop até que o resultado seja satisfatório (definido por uma função de teste), o algoritmo PCG utiliza a tática gerar e testar.

Spelunky usa o paradigma gerar e testar ao gerar as fases, no processo de dividi-la em 16 quadrados e gerar o caminho até o fim.

3.7. Geração automática *versus* autoria mista

O conceito de geração automática é proveniente da delegação da criação do conteúdo somente para o algoritmo PCG, tendo nenhuma ou pouquíssima interação humana. Já a autoria mista é a junção do processo de criação PCG com a intervenção significativa de designers humanos.

Spelunky é um jogo de autoria mista, uma vez que existem vários *blueprints* de salas feitos pelo designer, que são então utilizados pelo algoritmo PCG para gerar a fase após algumas modificações.

4. Técnicas de implementação PCG

O artigo já tratou de muitos aspectos do PCG, desde sua origem até seus paradigmas. Nessa seção são finalmente apresentadas algumas das principais técnicas de implementação, ou seja, os algoritmos mais importantes. Serão tratados: abordagem *Search-based* e técnicas *noise-based*.

É importante ressaltar que existem muitas outras formas de implementar conceitos PCG e a literatura definitivamente se beneficiará de novos estudos em novas técnicas inovadoras.

4.1. A abordagem *Search-based*

A abordagem *Search-based* é recente e vem tomando espaço nas publicações sobre PCG e entre desenvolvedores de jogos. Esse algoritmo é uma variante dos algoritmos “gerar e testar”. A metáfora que rege o *search-based* é que a geração do conteúdo deve ser uma busca e não simplesmente resultados aleatórios.

A principal divergência entre o *search-based* e os demais algoritmos “gerar e testar” está no fato de o *search-based* dar uma nota para o conteúdo criado (pode ser um valor numérico ou um vetor numérico), ao invés de simplesmente aceitar ou descartar o produto do algoritmo, como o “gerar e testar”.

Esse processo requer, então, uma função avaliadora consistente e confiável. Essa função é chamada de *fitness, evaluation* ou *utility function* [10]. Nesse artigo, será utilizada a nomenclatura “função avaliadora” somente.

Assim que os candidatos são avaliados pela função, são selecionados os candidatos com a maior nota e eles viram o *input* do algoritmo, com o intuito de gerar candidatos com notas

cada vez mais altas. Esse processo tem origem em algoritmos evolutivos e é extremamente eficaz em gerar conteúdo de alta qualidade.

Algoritmos evolutivos têm conceitos em comum com o darwinismo, como população, gerações dessa população e indivíduos melhor adaptados, seleção e reprodução.

A figura 4.1, extraída de [10] feita por Togelius *et al.*, serve como um ótimo comparativo entre os algoritmos *search-based*, gerar e testar (G&T, na imagem), e construtivos.

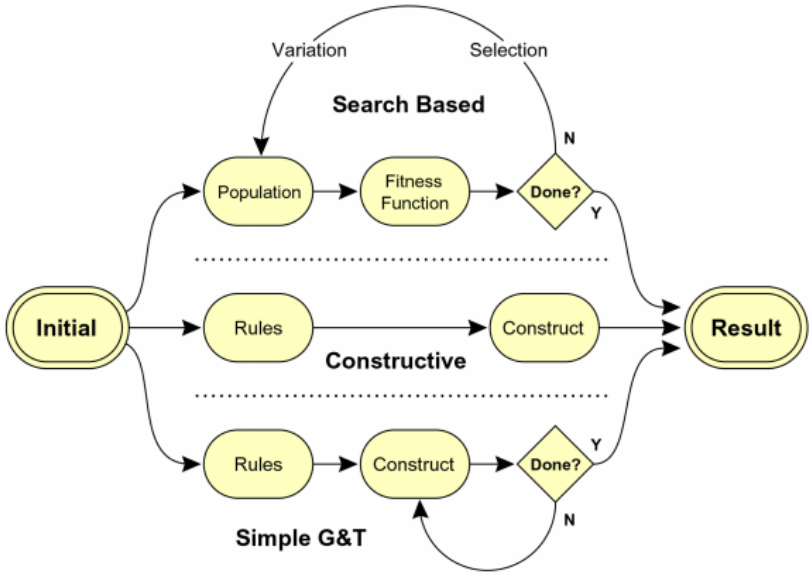

Figura 4.1: Fluxogramas comparativos de diferentes técnicas PCG.

Fonte: http://julian.togelius.com/Togelius2011Searchbased.pdf

4.2 Algoritmos baseados em ruídos

A utilização de ruídos em espaços virtuais é comum e muito vantajosa. Ruídos têm diversas aplicações e a principal e mais comum é a disposição semialeatória de conteúdo, como poeira sobre uma textura, disposição de árvores ou rochas sobre um terreno.

O ruído é fruto de diversos tipos de equações matemáticas multidimensionais que geram padrões que parecem aleatórios (Figura 4.2). O caráter semialeatório permite que sejam criadas texturas de forma dinâmica pelo computador.

O mais famoso algoritmo é o ruído de Perlin[13], que foi descoberto por Ken Perlin ao tentar criar texturas naturais para o filme *Tron* em 1982. Posteriormente, formalizou a descoberta e ganhou, em 1997, o *Academy Award for Technical Achievement.*

O algoritmo de Perlin, mesmo inovador, mostrou-se custoso computacionalmente[14], pois tinha complexidade $O(2^n)$, para n dimensões geradas. Foi então que surgiu o algoritmo

Simplex, de complexidade $O(n^2)$, que exige menos processamento. O Simplex, no entanto, é patenteado, mas pode ser executado de forma similiar pelo algoritmo OpenSimplex[15], de código aberto.

Figura 4.2: Resultado gráfico da função de Perlin

Fonte: https://docs.unity3d.com/ScriptReference/Mathf.PerlinNoise.html

5. Conclusão

A recente retomada de interesse pelo PCG acarreta a necessidade de formas inovadoras de criação e implementação das diversas técnicas. É possível que no futuro tenhamos jogos criados inteiramente por máquinas e algoritmos. As possibilidades e o potencial do PCG são quase imensuráveis e vão além de 18 quintilhões de planetas.

Fica claro neste artigo que a abordagem PCG é uma ferramenta poderosa no desenvolvimento de jogos. Além do *search-based* e *noise-based*, existem diversas outras técnicas de geração de conteúdo.

O PCG pode ser aliado a diversas áreas do conhecimento, como a matemática e até a biologia(*técnica usada em No Man's Sky*[19]). Fractais, linguagens formais, sistemas de Lindenmeyer etc.

A gama de aplicações das técnicas aqui expostas é vasta. O foco do artigo é a utilização do PCG em jogos, mas as técnicas PCG podem ser utilizadas em outros ambientes, como o cinema. Além do ruído de Perlin, existem softwares que utilizam PCG além dos jogos, como o Speedtree, que pode gerar vegetações para filmes. [18].

5.1. Considerações finais

Esperamos que este artigo sirva de base para futuros estudos e aplicações do PCG, seja por designers, acadêmicos ou até mesmo entusiastas, e que inspire designers a implementar as diversas técnicas de PCG.

6. Referências Bibliográficas

[1] Destiny is a $500 million gamble for Activision, says Kotick. IGN.com. Disponível em:

<http://www.ign.com/articles/2014/05/06/destiny-is-a-500-million-gamble-for-activision-says-kotick>

[2]Will Wright. The Future of Content. GDC 2005.Disponível em: <http://www.gdcvault.com/play/1019981/The-Future-of-Content-(English>

[3] Togelius, J., Kastbjerg, E., Schedl, D., Yannakakis, G.N.: What is procedural content generation? : Mario on the borderline.

[4] Shaker, Noor and Togelius, Julian and Nelson, Mark J. Procedural Content Generation in Games: A Textbook and an Overview of Current Research, Springer 2016

[5] Rilla Khaled, Mark J. Nelson, Pippin Barr. Design Metaphors for Procedural Content Generation in Games

[6] No Man's Sky versus the actual Universe. Kotaku. Disponível em <http://www.kotaku.co.uk/2016/04/20/no-mans-sky-versus-the-actual-universe>

[7] Exploring the 18,446,744,073,709,551,616 planets of No Man's Sky. Playstation Blog EU. Disponível em <https://blog.eu.playstation.com/2014/08/26/exploring-18446744073709551616-planets-mans-sky/>

[8]Biomes. Minecraft gamepedia. Disponível em : <http://minecraft.gamepedia.com/Biomes>

[9]Derek Yu. The Full Spelunky on Spelunky. Disponível em: <http://makegames.tumblr.com/post/4061040007/the-full-spelunky-on-spelunky>

[10] Julian Togelius, Georgios N. Yannakakis, Kenneth O. Stanley, Cameron Browne. Search-based Procedural Content Generation: A Taxonomy and Survey

[11] Georgios N. Yannakakis, Julian Togelius. Experience-Driven Procedural Content Generation

[12] Downwell, Devolver Digital. Disponível em: <http://downwellgame.com/>

[13]Ken Perlin. An image synthesizer. Disponível em: <http://dl.acm.org/citation.cfm?doid=325165.325247>

[14]Matt Zucker. The Perlin noise math FAQ. Disponível em: http://webstaff.itn.liu.se/~stegu/TNM022-2005/perlinnoiselinks/perlin-noise-math-faq.html#computation

[15]Uniblock. Noise! Disponível em: <http://uniblock.tumblr.com/post/97868843242/noise>

[16] Jenova Chen. Flow in Games: An Interactive Thesis on Dynamic Difficulty. Disponível em: <http://jenovachen.com/flowingames/introduction.htm.>

[17] Gamasutra.Procedural Content Generation: Thinking With Modules. Disponível em: <http://gamasutra.com/view/feature/174311/procedural_content_generation_.php?page=1>

[18] Speedtree Cinema, Speedtree. Disponível em: < http://www.speedtree.com/special-effects-software.php>

[19] Gielis, Johan (2003), "A generic geometric transformation that unifies a wide range of natural and abstract shapes", American Journal of Botany. Disponível em: <http://www.amjbot.org/content/90/3/333.abstract>