Universidade Federal do Rio de Janeiro
Instituto de Matemática - IM
Departamento de Ciência da Computação – DCC

GERAÇÃO PROCEDURAL DE MAPAS PARA JOGOS DE PLATAFORMA

Autores

CARLOS FELIPE MEDEIROS FARUOLO
FELIPE PIMENTEL DE AGUIAR

Orientador: Geraldo Bonorino Xexéo – D.Sc. COPPE/UFRJ
Rio de Janeiro, 2014

CARLOS FELIPE MEDEIROS FARUOLO
FELIPE PIMENTEL DE AGUIAR

Geração procedural de mapas para jogos de plataforma

Projeto Final de Curso submetido
ao Departamento de Ciência da
Computação do Instituto de Matemática
da Universidade Federal do Rio de
Janeiro como parte dos requisitos
necessários para obtenção do grau de
Bacharel em Informática.

Professores orientadores:
Geraldo Bonorino Xexéo – D.Sc. COPPE/UFRJ
Rio de Janeiro, 2014

Geração procedural de mapas para jogos de plataforma

Carlos Felipe Medeiros Faruolo
Felipe Pimentel de Aguiar

Projeto Final de Curso submetido ao Departamento de Ciência da Computação do Instituto de Matemática da Universidade Federal do Rio de Janeiro como parte dos requisitos necessários para obtenção do grau de Bacharel em Ciência da Computação.

APROVADA em _____ de _____ de 2014

Apresentado por:

_____
Carlos Felipe Medeiros Faruolo

_____
Felipe Pimentel de Aguiar

Aprovado por:

_____
Prof. Geraldo Bonorino Xexéo - D.Sc.
(Presidente)

_____
Adriano Joaquim de Oliveira Cruz - D.Sc.

_____
Rodrigo Penteado Ribeiro de Toledo- D.Sc.

RIO DE JANEIRO, RJ – BRASIL
SETEMBRO DE 2014

Agradecimentos

Carlos Felipe Medeiros Faruolo

Agradeço a essa Instituição, ao corpo docente e funcionários pelo ambiente amigável, estimulante, respeitoso e encorajador que tive a oportunidade de frequentar, pois meus horizontes de conhecimento nunca serão os mesmos após esses anos.
Agradeço ao meu orientador, D.Sc Geraldo Bonorino Xexéo pela oportunidade e pelo apoio na elaboração deste trabalho, no pouco tempo que lhe coube.
Agradeço aos meus amigos pelo apoio, motivação e pelos momentos de alegria e de estudo, que foram essenciais.
Agradeço aos meus pais, Mário e Teresa, pelo amor, incentivo e suporte incondicional nesta caminhada, sem os quais nada disto seria sequer possível.
Por fim, agradeço a minha namorada, Carolina, que embora tenha a encontrado já no final desta caminhada, meu deu apoio e motivação para finalizar esta fase de minha formação.

Felipe Pimentel de Aguiar

Agradeço primeiramente aos meus pais, Marcos Aguiar e Sheila Aguiar, e toda a minha família, por sempre me apoiarem em todas as minhas escolhas e me darem tudo que precisava para concluir esta faculdade.
Agradeço também à minha esposa, Isabela Port, por estar ao meu lado em todos os momentos que precisei. Agradeço também aos amigos que tive na faculdade, sem os quais não teria chegado onde estou agora.
E agradeço finalmente a Deus, por ter me proporcionado uma vida com pessoas tão especiais como as citadas anteriormente.

RESUMO

Geração procedural de mapas para jogos de plataforma

Carlos Felipe Medeiros Faruolo
Felipe Pimentel de Aguiar

Orientador: Geraldo Bonorino Xexéo – D.Sc

A ideia de gerar conteúdo de jogos de maneira procedural é uma tendência que ganhou força recentemente devido aos efeitos obtidos dessa prática, por vezes interessantes. Diversos jogos utilizam também um grau de aleatoriedade para tornar os resultados mais imprevisíveis e com isso, proporcionar um maior entretenimento ao usuário.

Cada jogo que utiliza estas técnicas possui mecanismos próprios e que, frequentemente, não estão disponíveis para reuso. Além disso, nem todos geram conteúdo utilizando elementos de aleatoriedade. Algumas ferramentas existem, como o *DungeonMaker*, porém a maioria produz resultados em um nicho específico ou são ferramentas proprietárias.

Este trabalho tem como objetivo programar uma ferramenta capaz de gerar conteúdo proceduralmente, mais especificamente, mapas bidimensionais para jogos de plataforma, simulando ambientes naturais, utilizando alguns algoritmos estudados. A ferramenta, apesar de fornecer resultados também específicos, poderá ser útil para um grande número de desenvolvedores de jogos, pois seus resultados serão gerados em formatos conhecidos.

Palavras-chave: geração procedural de conteúdo, mapas, jogos.

ABSTRACT

Procedural generation of maps for platform games

Carlos Felipe Medeiros Faruolo
Felipe Pimentel de Aguiar

Supervisor: Geraldo Bonorino Xexéo – D.Sc

The idea of generate game content in a procedural way is a trend that recently gained momentum due to the often interesting effects obtained from its practice. Many games also use a degree of randomness to make the outcomes more unpredictable and with that, provide a better user entertainment.

Each game that makes use of these techniques possesses their own implementation that, frequently, is not available for reuse. Furthermore, not all of these implementations generate content using randomness. Some tools exist, e.g., *DungeonMaker*, however, most of them produce results in a specific niche or are proprietary.

This work is meant to develop a tool capable of procedurally produce content – a 2-dimensional map for games, specifically, simulating natural environment, using some studied algorithms. The tool, albeit also providing specific results, may be useful for a large number of game developers, because its results are generated in well-known formats.

Keywords: procedural content generation, maps, games.

SUMARIO

ÍNDICE DE FIGURAS

1 INTRODUÇÃO

1.1 Cenário Geral

No cenário atual, os jogos eletrônicos já são um tipo de entretenimento já consolidado na sociedade. Podemos encontrar jogos dos mais variados gêneros, como esportes, ação, estratégia, jogos de tabuleiro etc.

Dentre os diversos gêneros existentes, há um gênero de jogo conhecido como mundo aberto (do inglês *open world*) que permite o jogador explorar livremente o mundo virtual do jogo. Nas últimas décadas, tal gênero ganhou força, tornando o aspecto de linearidade cada vez menos atraente no mercado. Dentro desse cenário, surgiu a tendência de produzir jogos com extenso uso de elementos de aleatoriedade, com elementos principais como mapa, cenário, itens, etc, sendo definidos aleatoriamente. Tal técnica, denominada Geração Procedural de Conteúdo (PCG)[1], ganhou notável reconhecimento nos últimos anos, devido aos efeitos obtidos com seu uso.

Frequentemente, cada jogo que utiliza estas técnicas possui mecanismos próprios, não estando disponíveis para reuso. Além disso, nem todos geram conteúdo utilizando elementos de aleatoriedade. Algumas ferramentas existem, como o *DungeonMaker*, porém a maioria produz resultados em um nicho específico ou são ferramentas proprietárias.

Dentro deste cenário, o objetivo deste trabalho consiste no estudo e desenvolvimento de algoritmos para a criação de um cenário virtual para jogos de plataforma bidimensionais, com toda sua morfologia gerada proceduralmente.

1.2 Objetivo do Trabalho

Este trabalho tem como objetivo a implementação de um software capaz de construir um mundo virtual 2D proceduralmente, com elementos de aleatoriedade. Através da execução de diversos algoritmos combinados, contendo parâmetros pseudo-aleatórios, serão gerados terrenos virtuais de formato e disposições sempre distintas, tomando por inspiração um cenário que se assemelha à natureza.

---

[1] Do inglês Procedural Content Generation

O software deverá receber como entrada certos parâmetros referentes aos algoritmos de geração suportados, por linhas de comando, e produzirá o resultado em um arquivo especificado. Nenhum outro tipo de interação é necessário por parte do usuário, uma vez que os algoritmos irão alimentar-se de dados pseudo-aleatórios obtidos em tempo de execução.

1.3 Organização do Trabalho

No capítulo 2 serão apresentados definições e exemplos de jogos de mundo aberto. No capítulo 3 será explicado o conceito de Geração de Conteúdo Procedural em jogos eletrônicos, bem como alguns exemplos de abordagens utilizadas em jogos. O capítulo 4 contém a explicação de cada etapa da geração do mundo, além de uma descrição detalhada dos algoritmos utilizados na construção do mesmo. No capítulo 5 será explicado o funcionamento da ferramenta desenvolvida, bem como os resultados obtidos na utilização do mesmo, seus aspectos positivos e problemas detectados. Finalmente, no capítulo 6, serão feitas as considerações finais deste projeto.

2 JOGOS

Neste capítulo será mostrado o conceito do gênero *mundo aberto*, exemplos de jogos famosos deste gênero, incluindo os tipos de jogos que poderão se beneficiar da ferramenta desenvolvida neste trabalho.

2.1 Jogos de mundo aberto

O gênero *mundo aberto*, em inglês conhecido como *open world*, *free roam* ou *sandbox*, é um gênero de jogos que tem como característica prover ao jogador um mundo virtual e permitir ao jogador a liberdade para explorá-lo e interagir com ele. A ideia deste gênero consiste em evitar a imposição de barreiras artificiais comuns na maioria dos jogos, provendo uma experiência de jogo que simula livre arbítrio, ainda que, na prática, tais barreiras ainda existam, devido à limitações técnicas ou à linearidade do jogo. (“Open world”, 2014)

Podemos citar como aspectos mais distintos para este estilo de jogo:

- Menor imposição de barreiras artificiais no mundo;
- Procura estimular o jogador à exploração;
- Permite a escolha e resolução de seus objetivos a critério do jogador;
- Ao invés de áreas separadas ou “fases”, normalmente apresenta o mundo ao jogador com todo o seu conteúdo livre do início ao final do jogo. (HARRIS, [s.d.])

A construção de um jogo mundo aberto baseia-se predominantemente em oferecer a possibilidade de que objetivos sejam alcançados de diversas maneiras diferentes e em qualquer ordem desejada. Tal característica se opõe ao conceito de linearidade. A linearidade descreve o grau em que um jogo ou a parte dele, é restrito sequencialmente. (“Linearity”, [s.d.]) Dessa forma a progressão não-linear dos objetivos dos jogadores significa um relaxamento na ordem em que os objetivos do jogo deve ser atingidos. Um jogo de mundo aberto apresenta, essencialmente, um baixo grau de linearidade. É importante notar que apesar do maior grau de liberdade, o gênero não impossibilita que exista linearidade na progressão do jogo. (JANSSEN, [s.d.])

Uma das maiores motivações dos das empresas para o desenvolvimento de jogos mundo aberto é capturar o desejo dos próprios jogadores por mais liberdade. Todd Howard, *Game Designer* na empresa Bethesda Studios, cita:

"Eles sentem-se mais similares ao personagem que estão jogando. Eles estão fazendo o que querem e não o que você, o designer, quer que eles façam. Quanto mais aberto, quanto mais reativo for, melhor será a experiência." (SEFTON, [s.d.])

2.1.1 Elite

Elite é um jogo de comércio espacial, publicado pela empresa Acornsoft em 1984 para os computadores *BBC Micro* e *Acorn Electron*, sendo posteriormente portado para diversas outras plataformas. ("*Elite* (video game)", 2014)

Em Elite, não existe uma maneira correta de jogar. O jogo gira em torno de viajar entre vários planetas do universo com o objetivo de ganhar créditos através de:

- Comercialização de bens entre estações espaciais;
- Ataques às outras naves;
- Recompensas por realização de missões;
- Mineração de asteroides.

Figura 1 Jogo Elite

Devido ao seu aspecto de jogabilidade não-linear e ao seu revolucionário gráfico 3D, Elite foi considerado um jogo histórico, e muito importante para a consolidação do gênero

Mundo Aberto, tendo influenciado muitos jogos posteriores como Grand Theft Auto e Eve Online. Elite é considerado um dos primeiros jogos do gênero Mundo Aberto, além de apresentar diversos aspectos do jogo gerados proceduralmente. (BARTON, 2009)

2.1.2 Landstalker: The Treasures of King Nole

Landstalker é um jogo desenvolvido pela empresa Climax Entertainment originalmente para o console Mega Drive/Genesis, sendo lançado no Japão no ano de 1992. O jogo é do gênero Aventura no estilo plataforma com visão isométrica, apresentando características como quebra-cabeças, combate, exploração e alguns elementos de RPG (Role-Playing Game). (“Landstalker”, [s.d.])

Figura 2 Jogo Landstalker

O jogo conta a história de um caçador de tesouros chamado Nigel, que é requisitado por uma fada, de nome Friday, para embarcar em uma longa jornada para encontrar o tesouro do Rei Nole. (“Landstalker”, [s.d.])

Landstalker apresenta um mundo virtual construído manualmente, isto é, previamente modelado, e vasto para exploração, com cidades, cavernas, selvas, montanhas, entre outros. Todas as cidades foram modeladas de maneira distinta no mundo, tendo variações desde os tipos

de criaturas que a povoam até a própria organização e arquitetura. (“Out-of-Print Archive • Mega Drive/Genesis reviews • Landstalker”, [s.d.])

2.1.3 The Elder Scrolls

The Elder Scrolls (frequentemente abreviado por TES) é uma série de jogos eletrônicos de RPG desenvolvido pela empresa *Bethesda Softworks* (“*The Elder Scrolls*”, 2014). A série é comumente conhecida pela sua jogabilidade e o fato dos jogos apresentarem aspectos do gênero mundo aberto, tendo os jogadores liberdade total de fazer o que quiserem com seus personagens no jogo. TES também possui como marca famosa o final aberto, isto é, continuação do jogo mesmo após o término dos objetivos principais, sugerindo uma idéia de um jogo infinito. Os principais títulos da série são: *Arena*, lançado em 1994, para DOS; *Daggerfall*, lançado em 1996, para DOS; *Morrowind*, lançado em 2002, para PC e Xbox; *Oblivion*, lançado em 2006, para PC e Xbox 360; e *Skyrim*, lançado em 2011, para PC, Xbox 360 e Playstation 3 (BLANCATO, 2007).

Figura 3 Jogo The Elder Scrolls: Oblivion

Para muitos amantes de jogos RPG eletrônicos, a série The Elder Scrolls é um dos melhores RPG’s da história, sempre fornecendo aos jogadores um mundo enorme e detalhado para explorar, a possibilidade de editar o personagem e sua aparência de diversas maneiras, diferentes linhas de progressão da história paralelas à principal, entre outros aspectos.

2.1.4 Minecraft

Minecraft é um jogo indie de gênero *sandbox* desenvolvido por Markus Persson e posteriormente publicado pela desenvolvedora de jogos eletrônicos sueca Mojang. O jogo foi inicialmente lançado para PC ainda no estágio *alpha* em maio de 2009, tendo sua versão final lançada somente em 2011, com versões para Windows, Android, iOS e Xbox 360. (ASHDOWN, 2010)

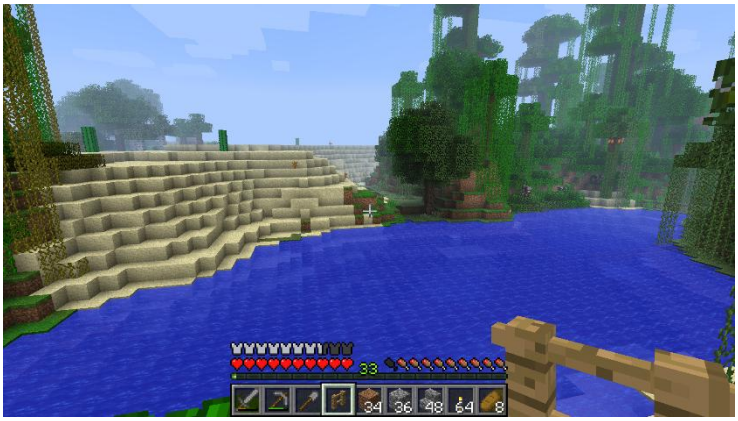

Figura 4 Jogo Minecraft

O jogo apresenta duas principais modalidades de jogo. Na primeira, chamada Sobrevivência (S*urvival*), o jogo é dividido em ciclos de dia e noite. Durante o dia, o jogador precisa utilizar seu tempo para adquirir recursos para sobreviver, seja coletando minérios, cavando túneis, pescando o colhendo alimentos. Durante a noite é necessário se proteger em abrigos devido à presença de perigosos monstros e bestas que aparecem para caçar. (“*Minecraft*”, 2014)

Na segunda modalidade, Criativo (*Creative)*, os jogadores disponibilizam-se de ainda mais liberdade do gênero sandbox. Aspectos como suprimentos de recursos ilimitados, habilidade de voar, e ausência de saúde e fome fazem com que este modo forneça ao jogador possibilidades potencialmente infinitas de construção. Embora seja necessário apenas um simples abrigo fechado, muitos jogadores exploram bastante esta área do jogo a ponto de modelarem imensas estruturas e recriarem famosas construções. [R]

2.1.5 Terraria

Desenvolvido e lançado em 2011 para Windows pela empresa Re-Logic, Terraria é um jogo Indie de ação no estilo sandbox, tendo como suas principais características a exploração, construção e combate. Tornou-se popular rapidamente graças à sua boa jogabilidade, tendo vendido milhares de cópias ao redor do mundo, e estando constantemente entre os jogos mais jogados pela plataforma Steam, da empresa Valve. [U]

Embora lançado recentemente, Terraria apresenta visão em duas dimensões e possui gráficos no estilo retrô, relembrando os jogos clássicos de consoles como Super Nintendo e Mega Drive, como Metroid e Super Mario Bros.

Figura 5 Jogo Terraria

A maior peculiaridade de jogos do gênero *sandbox* está nos objetivos apresentados no jogo. Diferente dos gêneros lineares, ou até mesmo Mundo Aberto, no *sandbox* não existe praticamente nenhum objetivo específico, o que faz com que as missões do jogo sejam definidas pelos próprios jogadores.

No caso de Terraria, apenas um objetivo é explicitado: sobreviver. Para isto é necessário que o jogador explore o mundo atrás de materiais que auxiliem sua sobrevivência, como matéria-prima para construção de abrigos, ferramentas, armas, armaduras etc.

O jogo apresenta um mundo vasto e bem diversificado, com biomas variando entre desertos e ambientes com neve, cada contendo uma grande quantidade de minérios e animais distintos. A maior parte do gameplay do jogo está, na verdade, na área subterrânea, onde o jogo

oferece todo um novo desafio de explorar e procurar por novos materias, através de centenas de sistemas de cavernas e túneis, apresentando biomas também abaixo da superfície. Além disso, nas áreas mais profundas do mundo, o jogo contém sua própria interpretação de um inferno! Vale ressaltar que Terraria é uma das motivações para este projeto, pois todo o conteúdo de um novo jogo é gerado proceduralmente.

3 GERAÇÃO PROCEDURAL DE CONTEÚDO

3.1 Geração Procedural

O termo geração procedural engloba várias técnicas de criação de elementos através de algoritmos. Os resultados obtidos com tais técnicas vão desde efeitos dinâmicos dentro de níveis pré-criados até geração de vozes, geografia e até mesmo personagens. [aP] Um exemplo de geração procedural são os fractais. [aO]

Amplamente utilizado na construção de mídias de diversos tipos, a geração procedural consiste em na criação dessas mídias de maneira algorítmica, em vez de manual. Frequentemente, devido às limitações técnicas, é inviável para certas aplicações o armazenamento de modelos, arquivos e conteúdo de maneira geral, ou a sua criação manual. [aO] Por esses motivos, a geração procedural é mais comumente relacionada à área de computação gráfica, tanto em filmes como em jogos eletrônicos. [W,X]

3.2 Geração Procedural de Conteúdo

Um aspecto comumente encontrado em alguns jogos de mundo aberto é a geração de conteúdo procedural, que corresponde à criação programática de conteúdo do jogo, através de processos estocásticos ou pseudo-aleatórios, gerando uma gama de resultados imprevisíveis no espaço do jogo, de maneira que sempre afeta sua jogabilidade em algum sentido. Nesses casos, é comum a aleatoriedade ser controlada a partir de alguns parâmetros e ser possível gerar uma grande diversidade de possíveis conteúdos considerados "válidos". [K, L] Exemplos de geração procedural de conteúdo em jogos incluem construção de labirintos, personagens, mapas inteiros ou até histórias em tempos de execução. [Y]

Entre as razões para se optar por geração procedural de conteúdo, podemos citar:

- Permitir ao criador a produção de uma grande quantidade de conteúdo em um menor espaço de tempo;
- Economia de mão de obra; a criação de um mundo virtual feito manualmente, embora pudesse possivelmente ser mais detalhista, requereria centenas ou até milhares de horas de trabalho realizadas por uma equipe de designer, ao passo que um algoritmo desenvolvido em algumas semanas poderia obter um resultado suficientemente satisfatório;

- Diminuição drástica do espaço de armazenamento, outrora necessário para conteúdo manualmente construído;
- Possibilidade de geração de produtos muito mais diversificados, muitas vezes não imaginados pelo criador [W].

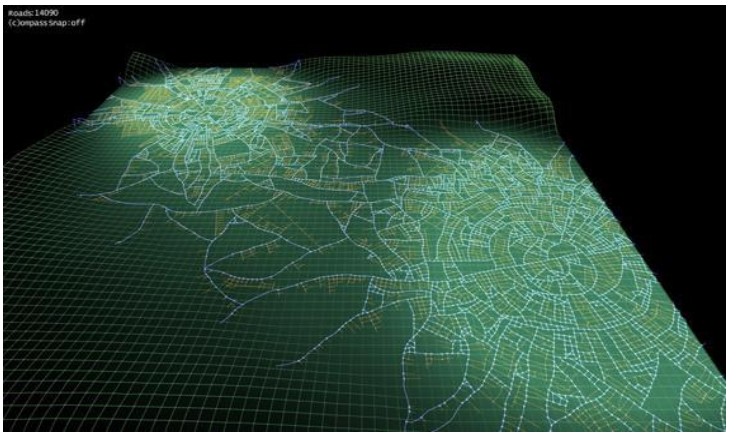

Figura 6 Imagem de um gerador procedural de cidades

Um aspecto fundamental para PCG é a aleatoriedade. Em certas aplicações, dado um conjunto de parâmetros de entrada, é desejável que seja possível criar um leque de possibilidades diferentes, tornando o resultado mais rico. [10]

Algumas denominações existem a fim de prover uma taxonomia dos diferentes tipos de aplicação de geração procedural de conteúdo. [aQ]

3.2.1 Momento da Geração: Tempo de Execução ou Desenvolvimento

Um aspecto importante a ser definido pelo desenvolvedor que emprega a geração de conteúdo procedural é quando usar de tais técnicas. O uso em tempo de execução gera conteúdo durante o uso da aplicação. Em um jogo em que cada área individual do jogo é gerado conforme o jogador progride é um exemplo deste uso. [aQ]

O uso das técnicas de geração procedural durante a fase de desenvolvimento da aplicação envolve gerar conteúdo para ser avaliado e retocado manualmente, geralmente por um *designer*. [aQ]

3.2.2 Crítico ou Opcional

Outra distinção possível é a análise se o conteúdo gerado é um elemento crucial para o jogo ou se é um elemento opcional, ou seja, a correção do resultado não é importante. [aQ]

O formato de um labirinto, por exemplo, é um elemento crítico, pois se este não for resolvível, o jogo resultante será defeituoso. Já um efeito de água, como usado no jogo *The Elder Scrolls: Morrowind*, é um elemento opcional, uma vez que este não interfere com a corretude do jogo. [aQ]

3.2.3 Semente ou Conjunto de Parâmetros

Uma distinção importante em relação aos algoritmos de geração procedural é o grau de parametrização dos mesmos. O uso de parâmetros diferentes resulta em conteúdos diferentes. Porém em um extremo, um conjunto de parâmetros são tomados como entrada para especificar as propriedades do conteúdo gerado. Em outro extremo, o resultado da execução de todos os algoritmos depende um valor apenas, chamado de semente (em inglês, *seed*) do algoritmo. [aQ]

3.2.4 Estocástico ou Determinístico

Um aspecto também importante é a se os algoritmos utilizados são determinísticos ou o resultado possui elementos de aleatoriedade. Para armazenamento de modelos ou texturas, é comum utilizar de algoritmos de geração procedural determinístico. Já para gerar mapas, níveis, itens, etc, é frequente o uso de algoritmos com graus de aleatoriedade, principalmente em jogos e computação gráfica. [aQ]

3.3 Exemplos de Métodos

Nesta sessão serão descritos alguns métodos matemáticos utilizados para a geração procedural de conteúdos em jogos, que podem aplicados em diversos aspectos.

3.3.1 Deslocamento de Ponto Médio (“*Midpoint Displacement*”)

O algoritmo de deslocamento de ponto médio (em inglês, *midpoint displacement*) é um algoritmo simples de se implementar e de execução rápida. A partir de um conjunto de pontos como entrada, é aplicada uma perturbação nos pontos médios, recursivamente. Uma das suas principais aplicações é gerar mapas de alturas, na maioria das vezes, de terrenos. Ele pode ser utilizado em mapas de diversas dimensões. [1]

Em sua versão de duas dimensões, o algoritmo pode ser usado para gerar um terreno para jogos de plataforma. A partir dos pontos dos extremos do mapa, gera-se um mapa de alturas. É possível usar o algoritmo com pontos discretos ou contínuos [2].

Na versão discreta o algoritmo para quando os intervalos chegam ao limite de detalhe e não é mais possível definir um ponto médio. A versão contínua do algoritmo tem como critério de parada um número de iterações especificado. É válido notar que, mesmo que o resultado desejado seja um mapa com posições discretas, é possível usar a versão contínua do algoritmo e depois de discretizar o resultado

3.3.2 Passeio aleatório ou Caminhada do Bêbado ("*Drunkard Walk*")

O algoritmo de passeio aleatório, ou caminhada do bêbado (em inglês, *drunkard walk*), é também um algoritmo de simples implementação, porém com aplicações diferentes do algoritmo anterior. Nesse algoritmo, a partir de um ponto, se desenha um caminho percorrido, escolhendo-se direções aleatórias. Usando-se de passos suficientemente pequenos, o resultado deste algoritmo é o movimento browniano. [3]

Este algoritmo é útil para geração de cavernas e trechos de minerais no processo de geração do mundo devido ao padrão gerado e a garantia de conectividade entre o ponto inicial e o ponto final do caminho traçado, já que, em cada passo do algoritmo, o próximo ponto é adjacente ao anterior. [4] Além disso, existem vários aspectos do algoritmo que podem ser explorados para se conseguir resultados diferentes. Algumas mudanças interessantes incluem:

3.3.2.1 Controle da direção

É possível definir as direções possíveis de se movimentar como sendo as comuns 4 direções cardeais (norte, sul, leste, oeste), as direções diagonais (noroeste, nordeste, sudeste, sudoeste) ou as 8 direções cardeais. A escolha das direções influencia na aparência do caminho quando se fazem caminhos longos.

Ao enviesar a escolha da direção para se movimentar, é possível reduzir a mudança de direção e traçar corredores longos. Para isso, define-se uma probabilidade com que o algoritmo muda de direção ou mantém a direção atual. Além disso, pode-se favorecer o direcionamento ao centro do mapa, evitando que o passeio encontre os extremos deste. [4]

3.3.2.2 Pincel como “marca”

A cada passo do algoritmo, marca-se o ponto atual da iteração. Porém a definição da marca utilizada é vaga e vai desde usar uma *flag* (ou limpar ela) até usar um carimbo (em inglês, o termo seria *brush*), que abrangeria uma área maior que o próprio ponto, produzindo artefatos visuais que lembram cavernas.

3.3.2.3 Múltiplos passeios

A aplicação do algoritmo apenas uma vez resulta em um caminho contíguo apenas. Em certas aplicações como, por exemplo, gerar cavernas em um mapa inteiro, é interessante repetir o algoritmo de maneira sistemática.

3.3.3 Autômato Celular

Em linhas gerais, um autômato celular é um modelo matemático discreto, introduzido por von Neumann e Ulam para auxiliar no estudo do processo de crescimento e auto reprodução [6].

Corresponde a uma grade de células, cada uma em um determinado estado, que variam conforme um conjunto de regras (que devem ser definidas) baseadas no estado da célula ou de seus vizinhos. Todas as células obedecem às mesmas regras, e a transição de estados deve ocorrer para todas simultaneamente [Z].

O autômato celular mais famoso é conhecido como Jogo da Vida, inventado pelo matemático John Horton Conway em 1970 para simulação de processos evolutivos. [aB, aC]

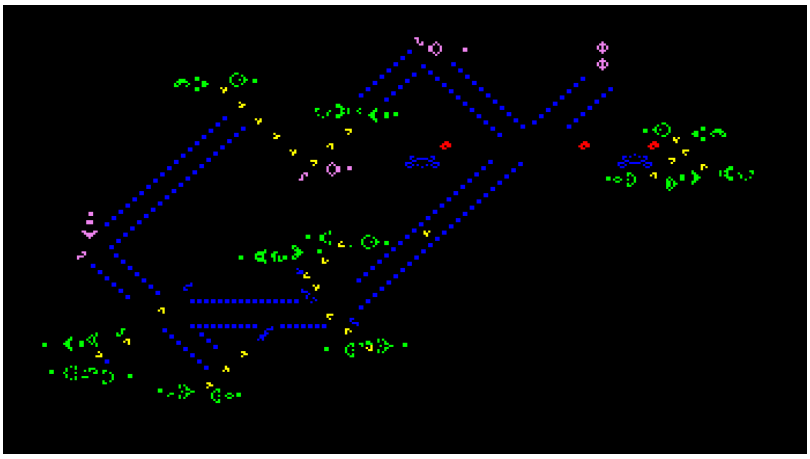

Figura 7 Jogo da Vida de John Conway

Existem diversas aplicações para autômatos celulares em geração procedural, devido ao aspecto orgânico dos resultados das iterações destes processos. As aplicações vão de simulações simples de hidrografias, bancos de areia, disposição de fluidos, até a simulação de crescimento de vegetação, ou até vida primitiva.

Dentro da área dos jogos, os autômatos celulares são mais comumente utilizados para gerações de cavernas ou calabouços, pois normalmente são capazes de gerar padrões com aspectos orgânicos [aA].

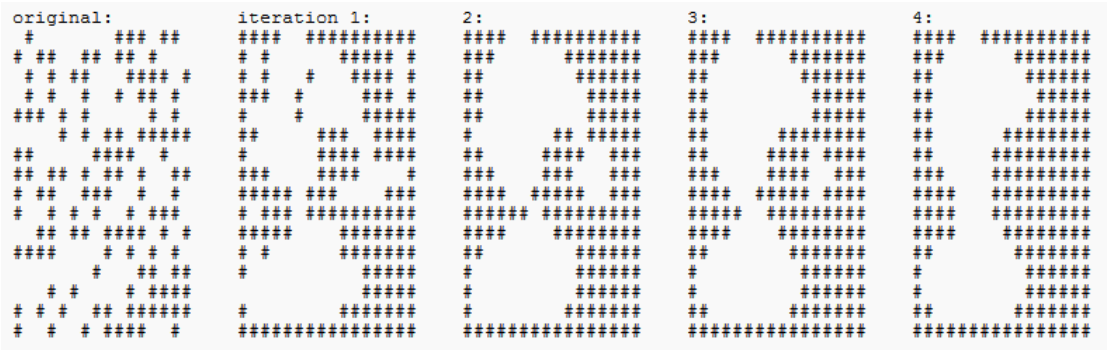

Figura 8 Exemplo de progressão das iterações de um autômato celular

3.3.4 Diagrama de Voronoi

O Diagrama de Voronoi corresponde a uma técnica de decomposição do espaço, tendo como base a proximidade de um determinado conjunto de elementos espalhados neste espaço. Nomeado após o matemático russo Georgy Fedosevich Voronoi em 1908, a técnica tem como resultado uma região retalhada em polígonos, cuja distância entre cada ponto dentro dos polígonos e o elemento contido nele é a menor possível [aD].

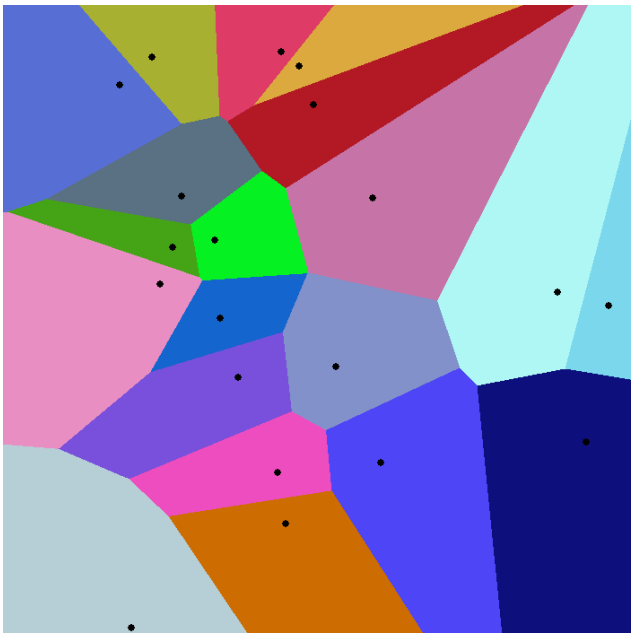

Figura 9 Exemplo de um Diagrama de Voronoi

As aplicações na área dos jogos eletrônicos são bem variadas. Por normalmente resultarem em uma região dividida com um padrão mais orgânico, os diagramas de Voronoi podem ser utilizados para demonstração de um mapa e seus países, com ilhas e mares, bem como geração de texturas [aD].

Existem algumas maneiras de se calcular o Diagrama de Voronoi (como algoritmo de Divisão-e-Conquista e Varredura), porém sua implementação demonstrou ser custosa (não em termos de complexidade de tempo, mas em termos de complexidade nos cálculos), não tendo sido, portanto, abordada na ferramenta proposta. [aG]

3.3.5 Triangulação de Delaunay

Inventado pelo matemático russo Boris Nikolaevich Delone em 1934, a triangulação de Delaunay é uma técnica de triangulação, aplicada um conjunto de vértices (devido a seu trabalho ter sido publicado em francês, ganhou o nome de Delaunay). Uma característica importante desta triangulação é que ela tende a evitar a geração de triângulos “magros”, pois maximiza o menor ângulo de todos os triângulos [aH].

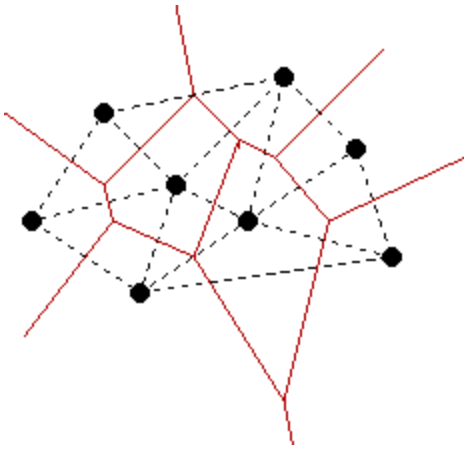

Figura 10 Exemplo de Triangulação de Delaunay

Na área de jogos, este conceito pode ser aplicado na geração de sistemas de cavernas, ou masmorras, tanto em duas como três dimensões. Uma abordagem seria obter a Árvore Geradora Mínima (que é na verdade um subconjunto do grafo de Delaunay), a fim de criar um caminho entre todos os pontos da região [aI].

3.4 Exemplos em Jogos

Como já foi descrito antes, algoritmos de PCG podem ser utilizados em diversos aspectos do jogo, desde a geração de texturas à composição de músicas em tempo de execução. A seguir serão descritas algumas aplicações desses algoritmos em games.

3.4.1 Dwarf Fortress

Desenvolvido pela empresa *Bay 12 Games* e lançado na data de 8 de Agosto de 2006 para Windows, Mac e Linux, Dwarf Fortress é um jogo de fantasia que oferece dois modos principais de jogo: Fortaleza, onde o jogador tem o objetivo de gerenciar e desenvolver uma fortaleza de anões, e Aventura, onde o jogador controla um aventureiro (podendo ser um anão, humano ou elfo), e tem o objetivo de explorar o mundo em busca de missões para cumprir, remetendo mais a um gênero de jogos conhecido como *roguelike.* [aQ2, aP2, aR]

Dwarf Fortress é também considerado um dos jogos mais complexos da história dos jogos para computador, não apenas pela quantidade de controles e inúmeros sistemas, mas também pelo visual, bastante antiquado para um jogo lançado em 2006. Toda a parte estética do jogo (personagem, mapas e todo ambiente) é renderizada na tela utilizando símbolos de código

ASCII (do inglês,*American Standard Code for Information Interchange*) como letras, símbolos matemáticos, entre outros. [aS]

Figura 11 Jogo Dwarf Fortress

Em ambos os modos, Dwarf Fortress oferece um mundo persistente gerado proceduralmente, apresentando relevos dos mais variados, cavernas, lagos e mares. Mas Dwarf Fortress dá um passo à frente na questão Geração Procedural de Conteúdo, criando aspectos como:

- Cidades, vilarejos e fortes, com respectivas civilizações, gerados proceduralmente;
- Vida Selvagem, bem como a quantidade de biomas, além de criação de “megabestas”, criaturas como dragões, titãs etc.

Idade do mundo, isto é, a quantidade de história gerada proceduralmente, o que consiste na quantidade de tempo que as civilizações tiveram para crescerem, conflitarem entre si e sofrerem os efeitos. Este aspecto também afeta o tempo que as megabestas passadas tiveram no mundo, o que pode ter levado a conseqüências como ataques a algumas civilizações, ou à sua própria morte, gerando um evento histórico. [aP2]

3.4.2 Diablo

Distribuída pela empresa *Blizzard*, *Diablo* é uma das séries mais famosas da história dos jogos, tendo vendido mais de 24,8 milhões de cópias ao redor do mundo. Contando com 3 jogos principais, *Diablo* é conhecido como sendo do gênero RPG com ação, e frequentemente rotulado também como estilo *hack n’ slash* [aJ, aK].

Figura 12 Jogo Diablo I

Uma das características mais famosas do jogo é a utilização de aleatoriedade para criação de conteúdo. Mapas, itens e monstros (a nível de posição, tipo e quantidade) são gerados em tempo de execução, fazendo com que o usuário tenha sempre uma experiência diferente a cada vez que for jogar. É considerado uma das maiores razões para o sucesso desta série, principalmente em seus primeiros jogos [aJ, aL].

A geração aleatória do mapa é o aspecto mais notório para os jogadores. A abordagem feita em *Diablo* consiste em combinações de pequenos "blocos" de mapa. Estes blocos são estão previamente fabricados, podendo então ser selecionados e arranjados de diversas maneiras diferentes a fim de construir o mapa [aL].

Figura 13 Exemplo de combinação de blocos de mapa no jogo Diablo III

3.4.3 Beat Hazard

Beat Hazard é um jogo desenvolvido por apenas um programador e lançado pela empresa *Cold Beam* em 2010 para PC com Windows, tendo atualmente suas versões para Mac, Linux, Xbox, Playstation 3, Android e iOS [aO2].

Com uma jogabilidade simples e clássica, o jogo coloca o jogador no controle de uma nave espacial, e tem como objetivo atacar e destruir todos os obstáculos e inimigos que surgirem em sua frente.

A grande peculiaridade de *Beat Hazard*, contudo, está na geração do conteúdo: pode-se dizer que os elementos do jogo são de certa maneira alimentados por dados que estão fora das fronteiras do jogo. Neste caso, os dados vêm na forma de música. Em *Beat Hazard*, desde os tiros que saem da nave do jogador até todo o leiaute apresentado na fase corrente reage de acordo com o ritmo da música que estiver sendo tocada [aM2].

Figura 14 Jogo Beat Hazard

O jogo vem com uma coleção de suas músicas próprias, porém ele permite que você toque suas próprias músicas dentro do jogo, o que expande as possibilidades de configuração de fases de maneira infinita [aM2, aN2].

Existem dois modos de jogar: o modo Única Música (ou *Single-Song*), onde o jogador deve conseguir sobreviver na fase até o término da música, e o modo Sobrevivência (ou *Survival*), onde é tocada uma lista de músicas em seqüência e a fase acaba somente quando

terminarem as vidas do jogador. No modo Única Música, são oferecidos também 4 níveis de dificuldade [aM2, aN2].

3.5 Exemplo de Ferramentas

3.5.1 DungeonMaker

Desenvolvida por Peter Henningsen, DungeonMaker é uma ferramenta concebida para criação de conteúdo para jogos do gênero RPG, como mapas, calabouços, etc., para diversos tipos de jogos, desde jogos de papel e caneta até jogos de computador. [aU]

O programa funciona recebendo um grande número de parâmetros de um arquivo e a partir desses parâmetros inicia o processo de geração do calabouço (em inglês, *dungeon*). [aX]

Figura 15 Ferramenta DungeonMaker

3.5.2 Terragen

Terragen é um programa de renderização e animação de paisagens realísticas desenvolvido pela *Planetside Software*. Esta ferramenta é capaz de gerar ambientes completos, especificando detalhes como clima, relevo, etc. [aV]

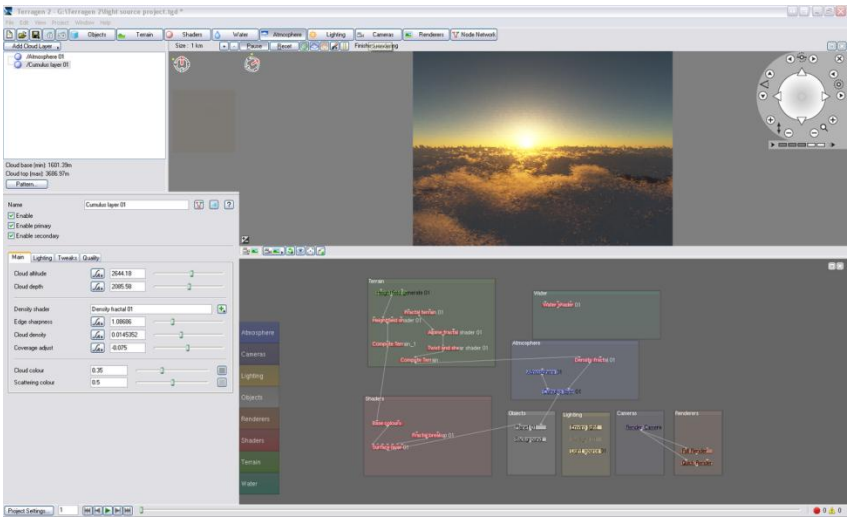

Figura 16 Ferramenta Terragen

A ferramenta, que atualmente já está na sua terceira versão, é bastante popular entre artistas amadores, principalmente a versão *classic*, que é de uso gratuito. Ela gera o terreno através de um mapa de altura, que pode ser exportado para utilização em outros programas. [aY]

4 GERAÇÃO DE MUNDO

Neste capítulo são apresentados os algoritmos utilizados pela ferramenta desenvolvida para o processo de geração do mundo, bem como um script recomendado para a ferramenta.

4.1 Conceitos básicos

No contexto do trabalho apresentado, o "mundo" gerado consiste em um cenário representado em duas dimensões: altura e largura. Tais características de cenário são típicas de um jogo de plataforma, em que um jogador pode andar pelo cenário, pulando obstáculos. É importante notar que esse formato difere do formato de cenário em que o ponto de vista é de cima para baixo (em inglês conhecido como *top-down*) – a gravidade (se presente) age sobre o personagem de maneira que o mesmo "caia" em direção a parte inferior do cenário.

O cenário desejado como resultado da ferramenta desenvolvida neste trabalho é baseado fortemente nos cenários do jogo *Terraria*. Neste jogo, o "mundo" é divido em uma malha (em inglês *grid*), de maneira que cada posição da malha possa ser ocupada por um tipo de objeto simultaneamente. Neste trabalho, tais posições da malha serão referidas também como blocos ou células – e o mapa gerado é nada mais que uma matriz com células populadas com tais objetos e blocos. Para sinalizar a presença de um objeto ou bloco em uma posição, é possível utilizar uma variável que indica a presença e o tipo deste – um tipo de variável comumente denominada *flag*. [aZ]

4.2 Implementação dos algoritmos

Neste trabalho, foi implementado uma pequena seleção de algoritmos de geração procedural. Segue abaixo um detalhamento dos algoritmos na forma em que foram implementados.

4.2.1 Implementação do algoritmo de deslocamento de ponto médio

A versão recursiva deste algoritmo, usando pontos discretos - utilizada nesse trabalho - funciona da seguinte maneira:

1. No início, se escolhem-se dois pontos, que serão os extremos do contorno a ser gerado. O valor da coordenada Y desses pontos fica a critério do programador (fixo ou escolhido aleatoriamente). A partir desses pontos, se repete:

2. Acha-se o ponto médio P_m entre os dois pontos P_i e P_f, extremos do intervalo. Se $P_i = P_f$, o programa termina;
3. Estipula-se o valor da coordenada Y de P_m como sendo a média das alturas de P_i e P_f.
4. Desloca-se a altura de P_m de um valor aleatório no intervalo [a, b].
5. Repete-se o passo 2 com extremos P_i e P_m.
6. Repete-se o passo 2 com extremos P_m e P_f.

Figura 17 Algoritmo Deslocamento de Ponto Médio em execução

Nesta versão, o algoritmo para quando não existem intervalos com tamanhos maiores que 2, ou seja, não é possível achar um novo ponto médio, chegando ao limite de detalhe.

4.2.2 Implementação do algoritmo de passeio aleatório

Neste trabalho, o algoritmo de passeio aleatório foi utilizado em diversas funções. Foram implementadas variações que utilizam direções enviesadas, pincéis (*brush*) e passeios múltiplos. Foram explorados ainda pincéis de largura variável, criando resultados ainda mais orgânicos.

O algoritmo genérico funciona da seguinte maneira:

1. Escolhe-se um ponto inicial para o algoritmo, a critério do programador (aleatório ou não). Execute o passo 2;
2. Marque o ponto atual. Execute o passo 3;
3. Escolhe-se uma direção aleatoriamente e “ande” para o ponto imediatamente naquela direção. Execute o passo 2;

O algoritmo para após um número de iterações desejado. [4]

Figura 18 Algoritmo Passeio Aleatório em execução

Em uma variação implementada neste trabalho, várias instâncias chamadas de mineradores (*miners*) executam passeios aleatórios simultaneamente sobre o mapa. Durante a sua execução, cada minerador cria, com certa probabilidade, outro minerador em uma posição aleatória no mapa. Opcionalmente, um minerador termina seu passeio ao chegar a uma área completamente marcada. [5]

4.2.3 Implementação do algoritmo usando autômato celular

Neste trabalho foi desenvolvido um algoritmo de autômato celular simples para simular a disposição de água no mapa, como resultado de precipitações de chuva. A regra utilizada segue abaixo:

1. Se houver posição vaga abaixo, marcar posição abaixo.
2. Se houver posição vaga à esquerda ou à direita (mas não ambas), marcar a posição vaga.
3. Se ambas as posições à esquerda e à direita estiverem vagas e a posição anterior era acima da atual, marcar uma das duas (escolhida aleatoriamente).
4. Se ambas as posições à esquerda e à direita estiverem vagas e a posição anterior era à esquerda da atual, marcar a posição à direita.
5. Se ambas as posições à esquerda e à direita estiverem vagas e a posição anterior era à direita da atual, marcar a posição à esquerda.

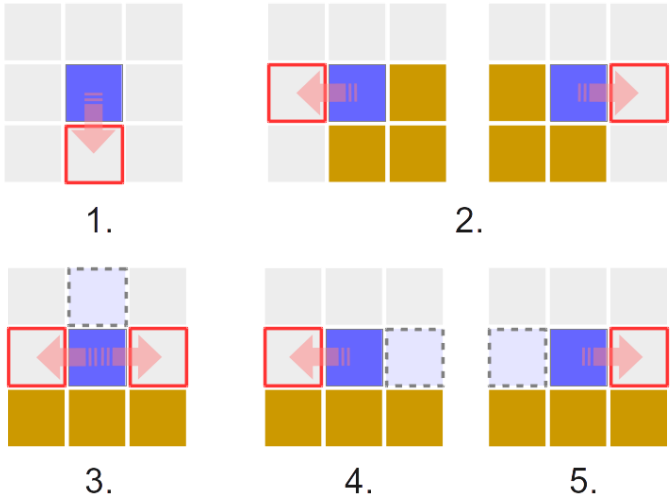

Figura 19 Ilustração das regras utilizadas no autômato celular de disposição de água

O mesmo algoritmo (ou uma variação) poderia ser utilizado para simular a precipitação de areia, lama, etc. É importante notar que, devido à natureza dos autômatos celulares, esse tipo de algoritmo é adequado apenas para mapas discretizados (malha de blocos).

Na implementação pela ferramenta desenvolvida neste trabalho, algumas melhorias foram introduzidas para melhorar o desempenho do algoritmo que fogem do conceito de autômato celular, como contadores de iterações sem movimento em gotas.

4.3 Roteiro da geração do mapa

Para que o resultado da ferramenta tenha características verossímeis, semelhantes à de uma estrutura "geológica", foi constatada a necessidade de usar um roteiro de algoritmos. Foi então desenvolvido um roteiro que exemplifique como os algoritmos podem ser usados para se obter resultados satisfatórios.

Baseado em diversos testes realizados, e tomando como referência os jogos previamente citados, os seguintes passos foram escolhidos:

4.3.1 Definição do Relevo

O primeiro passo no algoritmo consiste na modelagem do relevo do mapa. Para isto, diversos algoritmos podem ser utilizados. Neste trabalho, utilizamos o algoritmo descrito em 4.1.1. Este algoritmo preenche o mapa com terra, formando uma silhueta com formato que varia

aleatoriamente, criando artefatos como montanhas, planícies, depressões, etc., dependendo dos parâmetros utilizados.

Figura 20 Roteiro de geração do mapa: definição do relevo

4.3.2 Disposição das rochas

Nesta etapa, são inseridos trechos de rochas ao longo de toda a extensão do mapa. Para isso, o algoritmo em 4.1.2.4 proporciona bons resultados, com parâmetros adequados. Assim, alguns trechos escolhidos do solo gerado no passo anterior são alterados para rocha. O trecho é escolhido de maneira à melhor imitar o padrão natural. Embora não tenha sido abordado neste trabalho, essa etapa poderia ser repetida para dispor outros tipos de rochas ou minerais.

Figura 21 Roteiro de geração do mapa: disposição das rochas

4.3.3 Cavernas

Após dispor as rochas no solo, começa o processo de projeção das cavernas. Para esse passo, o algoritmo descrito em 4.1.2. Através dele, são criados corredores vazios que se estendem de maneira aleatória para o subsolo, variando direção e grossura. É importante ressaltar a necessidade de um tratamento especial para esta etapa, pois é necessário garantir certas condições, como, por exemplo, impedir cavernas nos extremos do mapa.

Figura 22 Roteiro de geração do mapa: cavernas

4.3.4 Disposição de água

Após gerar a morfologia por completo nos passos anteriores, inicia-se o processo de criação do litoral e de pequenos lagos. Para isso, utiliza-se o algoritmo descrito em 4.1.4. Através dele, são criadas duas fontes, uma em cada extremo do mapa, que despejam um certo volume de água, definido por parâmetro, a fim de formar os litorais. Além disso, é realizado um processo de gotejamento que simula o resultado de "chuvas" sobre a superfície. Assim, formam-se pequenas porções de água, de maneira semelhante a lagos.

Figura 23 Roteiro de geração do mapa: disposição de água

4.3.5 Disposição de vegetação

Finalmente, após definir o relevo e a hidrografia, cria-se uma camada de grama, utilizando um algoritmo simples, que transforma os blocos de terra da superfície por grama quando não existir água acima.

Figura 24 Roteiro de geração do mapa: disposição de vegetação

5 FERRAMENTA E RESULTADOS

A ferramenta desenvolvida neste trabalho consiste de um executável que funciona através de linha de comando, recebendo parâmetros especificados nos argumentos do programa. Após a execução, o resultado pode ser salvo em um arquivo, conforme for especificado na linha de comando.

5.1 Tecnologias empregadas

A aplicação foi desenvolvida em C++, utilizando o IDE Eclipse, que, apesar de ser um IDE utilizado principalmente para desenvolvimento com a linguagem Java, é capaz de trabalhar com a linguagem C ou C++ através de uma extensão chamada Eclipse CDT (*C/C++ Development Tools*). A razão dessa escolha é devido ao leque de funcionalidades oferecido por esse IDE – dentre elas, a função de auto-completar avançada do Eclipse sendo um dos maiores atrativos.

A escolha da linguagem C++ para utilização é devida ao fato de que a idéia da ferramenta aqui desenvolvida ter sido concebida durante o desenvolvimento de um jogo de plataforma também feito em C++, que demandava um gerador de mapas. Usando a mesma linguagem, uma possível integração da ferramenta com o jogo seria possível posteriormente.

Algumas bibliotecas foram utilizadas para prover certas funcionalidades da ferramenta. Para o tratamento dos argumentos da linha de comando da ferramenta, foi utilizada a biblioteca TCLAP (*Templatized C++ Command Line Parser Library*), capaz de prover uma maneira simples e elegante de checar os argumentos fornecidos. Para exportar o resultado da ferramenta no formato BMP, foi utilizada a biblioteca EasyBMP. Para exportar o resultado no formato TMX, foi utilizada a biblioteca RapidXML, que fornece uma maneira de ler e escrever arquivos no formato XML, do qual o formato TMX também o é. Por fim, foi utilizada a biblioteca SDL (*Simple DirectMedia Layer*) para criar a tela do modo interativo.

5.2 Argumentos

Para especificar os parâmetros de execução, a ferramenta aceita determinados argumentos de linha de comando. A seguir serão descritos os argumentos.

5.2.1 Parâmetros de tamanho

A ferramenta aceita os argumentos –c [VALOR] e –l [VALOR], sendo, respectivamente, o número de colunas (ou seja, a largura) do mapa e o número de linhas (ou seja, a altura) do mapa. Tais parâmetros, ambos inteiros sem sinal, especificam as dimensões do mapa a ser gerado.

5.2.2 Parâmetros de entrada

São um ou mais argumentos da forma –i [NOME DO ARQUIVO]. Especificam arquivos a serem interpretados, buscando parâmetros de transformação (veja 4.2.5).

5.2.3 Parâmetros de saída

São parâmetros da forma –x [NOME DO ARQUIVO], -b [NOME DO ARQUIVO] ou –r [NOME DO ARQUIVO]. Especificam em quais arquivos serão salvos o resultado da ferramenta. O parâmetro –x salva o resultado no formato *Tile Map XML* (*.tmx). O parâmetro –b salva o resultado em *Bitmap* (*.bmp), de fácil visualização. O parâmetro –r salva o resultado em um arquivo de texto simples, escrevendo uma matriz que representa a malha do mapa. Cada um desses parâmetros é aceito somente uma vez.

5.2.3.1 Formato Tile Map XML

O Formato *Tile Map XML* é um formato de mapas de jogos, derivado do formato XML. Ele foi concebido para ser um maneira flexível de descrever mapas baseados em *tiles*. É o formato padrão da ferramenta de edição de mapas *Tiled*. [7] Foi escolhido desenvolver a funcionalidade de saída neste formato devido à visibilidade do formato, do fato de ser de especificação aberta e livre e devido à possibilidade de visualizar e analisar os mapas produtos da ferramenta em uma ferramenta externa de fácil uso.

5.2.3.2 Formato *Bitmap*

O formato *bitmap* é um tipo de arquivo de imagem extremamente popular e suportado que armazena um malha de pixels, sem perdas. Este formato é útil para visualizar rapidamente o resultado da ferramenta, uma vez que qualquer visualizador de imagens atual é capaz de ler este formato. [8]

5.2.3.3 Saída em arquivo de texto

A saída da ferramenta em arquivo de texto consta apenas de uma matriz com as *flags* de cada posição da malha do mapa. É uma saída extremamente simples, feita no início do desenvolvimento da ferramenta com o intuito de analisar a corretude da mesma.

5.2.4 Parâmetros do modo interativo

Foi desenvolvida na ferramenta uma funcionalidade de demonstração dos passos de execução dos algoritmos. Batizada de modo interativo, esta funcionalidade provê uma tela que mostra o mapa sendo modificado pelos algoritmos. Desta maneira é possível analisar a dinâmica dos algoritmos em tempo real – algo tanto agradável quanto divertido. A tela depende da disponibilidade da biblioteca multimídia SDL para executar. No entanto, a biblioteca é amplamente suportada, o que facilita o seu uso. [9]

Os parâmetros para ativar o modo interativo são os seguintes:

`-S` ou`--show-window`: ativa a tela de previsão (modo interativo) ao longo do programa. O padrão é não ativar.

`-M` ou `--window-manual`: a tela de previsão irá esperar por confirmação manual do usuário, através da tecla*Enter* do teclado, a cada iteração. O padrão é rodar automaticamente.

`-Z [ZOOM]` ou`--window-zoom [ZOOM]`: *zoom* da tela de previsão. [ZOOM] deve ser um inteiro positivo. O padrão é 1 (sem ampliação).

`-D [DELAY]`ou`--window-delay [DELAY]`: se DELAY $\geq$ 0, DELAY é interpretado como tempo (em milissegundos) a esperar a cada iteração antes de mostrar a próxima. Se DELAY < 0, o valor é interpretado como número de iterações a se pular, não exibindo-a, a cada iteração exibida – mecanismo semelhante ao pulo de quadros (em inglês, *frame skip*), efetivamente aumentando a velocidade do processo de geração. DELAY deve ser um inteiro. O padrão é -50 (pular 50 iterações).

5.2.5 Parâmetros de transformação

No processo de geração do mundo, as diferentes combinações na ordem de aplicação dos algoritmos alteram o aspecto do resultado final. Por isso, foi criada uma funcionalidade capaz de executar os diversos algoritmos em qualquer ordem, quantas vezes forem necessárias.Tal recurso funciona recebendo parâmetros por linha de comando, executando os algoritmos especificados em ordem também especificada, de maneira semelhante a um script. Existe também a possibilidade de executar tais parâmetros de um arquivo, utilizando o parâmetro `-i` ou `--input`.

Os parâmetros de transformação especificam quais os algoritmos a serem empregados no processo de geração. Estes argumentos não são usados com o caractere hífen prefixados. Além disso, os parâmetros dependem da ordem, sendo isso uma limitação da implementação, e também intencional, já que a ordem dos argumentos também afeta o aspecto do resultado.

A seguir serão comentados os possíveis parâmetros de transformação aceitos atualmente pela ferramenta:

5.2.5.1 O parâmetro *fill*

Este parâmetro é o mais simples. Ele indica que a malha deve ser preenchida, de baixo para cima, com um valor (uma *flag*) especificado, até atingir uma fração do total, também especificada. A sintaxe é:

```
fill [FLAG] [FRAÇÃO]
```

Em que [FLAG] é um inteiro que representa uma *flag* e [FRAÇÃO] é um racional (do tipo *Double*) que indica a fração da malha que deve ser preenchida. Note que esta função sobrescreve qualquer valor que estiver nas posições a serem preechidas.

5.2.5.2 O parâmetro *mdisp*

Este parâmetro indica que deve ser utilizado o algoritmo de deslocamento de ponto médio (*midpoint displacement*). A sintaxe é:

```
mdisp [FLAG] [FATOR]
```

onde [FLAG] é um inteiro que representa uma flag e [FATOR] é um racional (do tipo Double) positivo, que representa o fator de suavidade do algoritmo. Note que esta função não irá apagar as células acima do contorno gerado.

Foi implementada também uma variação da função *mdisp*, a *rb_mdisp*, onde as alturas dos pontos da borda do mapa são escolhidas aleatoriamente, antes do inicio do algoritmo. Isso dispensa o uso de *fill* antes do uso de *mdisp*.

5.2.5.3 Os parâmetros r4walk, r8walk, cave e miners

Os parâmetros *r4walk* e *r8walk* indicam que deve ser utilizado o algoritmo de passeio aleatório com 4 e 8 direções possíveis, respectivamente. A sintaxe é:

```
r4walk [FLAG] [LINHA] [COLUNA] [COMP] [PROBTROCA] [BRUSHSIZE]
r8walk [FLAG] [LINHA] [COLUNA] [COMP] [PROBTROCA] [BRUSHSIZE]
```

onde [FLAG] é um inteiro que representa a *flag* que deve ser preenchida, [LINHA] e [COLUNA] são inteiros positivos que indicam a posição inicial do algoritmo na malha, [COMP] é um inteiro positivo que representa a duração, em iterações, do passeio, [PROBTROCA] é um racional no intervalo [0,1] que representa a probabilidade de troca de direção do passeio e, por fim, [BRUSHSIZE] um inteiro positivo que representa a grossura da "marca" deixada pelo passeio.

Valores baixos da probabilidade de troca de direção garantem passeios com mais "corredores" continuados.

O parâmetro *cave* funciona de maneira semelhante aos parâmetros *r4walk* e *r8walk*, porém com um argumento a mais, [BRUSHSIZERANGE], que especifica um intervalo de variação da grossura da marca, criando padrões mais orgânicos. A sintaxe é:

```
cave [FLAG] [LINHA] [COLUNA] [COMP] [PROBTROCA] [BRUSHSIZE]
[BRUSHSIZERANGE]
```

onde [BRUSHSIZERANGE] é um racional que representa o intervalo de variação da grossura da marca deixada pelo passeio.

O parâmetro miners indica que deve ser utilizado o variante do passeio aleatório, o algoritmo dos mineradores. A sintaxe é:

```
miners [FLAG] [FLAG_ALVO] [QTD] [PROB]
```

Onde [FLAG] é um inteiro que representa a *flag* com que deve ser preenchida, [FLAG_ALVO] é um inteiro que representa a *flag* da qual o minerador pode substituir, [QTD] é o número de mineradores a se criar, no total, e [PROB] é a probabilidade de se criar um novo minerador, a cada passo do algoritmo.

5.2.5.4 Os parâmetros *spring* e *springn*

Os parâmetros *spring* e *springn* indicam que deve ser utilizado o algoritmo de autômato celular para simular a precipitação de água. A sintaxe para *spring* é:

```
spring [FLAG] [VOLUME] [LINHA] [COLUNA]
```

Onde [FLAG] é um inteiro que representa a *flag* com que deve ser preenchida, [VOLUME] é um inteiro positivo indicando a quantidade de blocos a "ser precipitada", [LINHA] e [COLUNA] indicam o ponto onde se localiza a nascente, ou seja, de onde partirão os blocos gotas.

O parâmetro springn é semelhante ao anterior, porém especifica várias nascentes e a taxa de precipitação dessas nascentes. A sintaxe é:

```
springn [FLAG] [VOLUME] [TAXA] { [LINHA] [COLUNA] }n
```

Onde [FLAG] é um inteiro que representa a *flag* com que deve ser preenchida, [VOLUME] é um inteiro positivo indicando a quantidade de blocos a "ser precipitada", [TAXA] é um racional no intervalo [0, 1], que representa a taxa de criação de gotas (na verdade, tal taxa é uma probabilidade associada a, em cada iteração, criar uma gota), e 1 ou mais pares de argumentos [LINHA] [COLUNA], inteiros positivos, especificando uma posição de uma nascente.

É possível conseguir resultados interessantes ao empregar a função spring com parâmetro de taxa próximo de zero, com várias nascentes, obtendo um resultado semelhante à de uma chuva.

5.2.5.5 O parâmetro surfrep

Este parâmetro é utilizado para indicar o uso de um algoritmo simples que troca as *flags* do contorno por outras. Esta função é útil para obter um acabamento de grama na malha, simulando uma vegetação rasa natural. O algoritmo funciona partindo do topo da malha e, em cada coluna, desce até achar uma posição com a *flag* especificada e uma posição livre (com a *flag* 0) imediatamente acima. A sintaxe é:

```
surfrep [FLAG] [FLAG_ALVO]
```

Onde [FLAG] é um inteiro que representa a *flag* com que deve ser preenchida, [FLAG_ALVO] é um inteiro que representa a *flag* da qual pode-se substituir.

5.2.6 Outros parâmetros

Os parâmetros --version e –h mostram a versão do programa e um texto informativo sobre o emprego dos parâmetros no programa, respectivamente.

O parâmetro –e ou --empty especifica que o mapa gerado deve ser vazio. Tal parâmetro é desejável quando se deseja usar apenas os parâmetros de transformação para gerar o mundo.

5.3 Testes

Para assegurar a corretude da ferramenta, foram realizados testes ao longo do desenvolvimento para diferentes conjuntos de métodos e seus parâmetros, variando também a saída para mapas em diversas escalas.

Primeiramente foi executada uma bateria de testes para cada algoritmo individualmente, todos apresentando bom comportamento e resultados satisfatórios. Como a execução dos algoritmos apresenta saídas não determinísticas, foi necessário repetir os testes diversas vezes, a fim de assegurar sua corretude.

Em seguida, foram executados testes com todos os algoritmos em conjunto, variando agora o tamanho escolhido para o mapa. Foi observado que, para escalas menores, o resultado obtido dos testes foi em média satisfatório, produzindo mapas bem formados e com o efeito dos

algoritmos bem distribuído. Para mapas de escalas maiores, o que é mais comumente utilizado em jogos do gênero Mundo Aberto, os resultados foram igualmente satisfatórios, apesar ter sido notado um aumento significativo do tempo de execução do teste.

A seguir algumas imagens de resultados de algumas execuções da ferramenta:

Figura 25 Mapa gerado utilizando roteiro de execução

Figura 26 Mapa gerado utilizando roteiro de execução - 2

Figura 27 Mapa gerado utilizando parâmetros com duas camadas de relevo simples

Figura 28 Mapa gerado utilizando parâmetros com duas camadas de relevo, com cavernas e águas

Figura 29 Mapa gerado com um relevo simples e com muita águia, simulando uma ilha

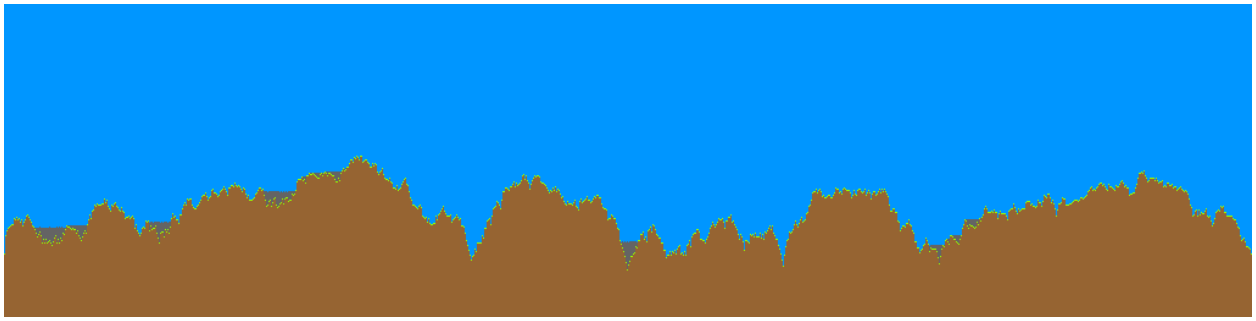

Figura 30 Mapa gerado com relevo espetado, preenchido completamente com água, simulando fundo de oceano.

Foram realizados também testes em diferentes sistemas operacionais (Windows 7, Ubuntu, Fedora) e em diferentes arquiteturas (x86 e x86_64) e em todos a execução ocorreu normalmente. Assim, pode-se considerar que ferramenta é multi-plataforma, embora, por motivos técnicos, não foram realizados testes em Mac.

5.4 Problemas

Durante o desenvolvimento e durante os testes, foram encontradas algumas dificuldades. A seguir, serão listadas:

5.4.1 Tempos de execução demasiadamente grandes

Quando se executa a ferramenta para gerar mapas grandes, o tempo de execução se torna proibitivamente lento. Isso se deve, principalmente, ao algoritmo de autômato celular, do qual o critério de parada está otimizado de acordo com o tamanho da largura do mapa. Infelizmente, a única solução definitiva é aperfeiçoar mais o algoritmo, ou não utilizar a ferramenta para mapas muito grandes.

5.4.2 Problemas com alguns algoritmos

Alguns algoritmos, como o de colocação de grama, quando executados em situações degeneradas (mapa vazio), causam erros de falha de segmentação. Devido a limitações de tempo, não foi investido tempo para tratar dessa situação.

6 CONSIDERAÇÕES FINAIS

6.1 Dificuldades encontradas

No desenvolvimento deste trabalho, foram encontradas diversas dificuldades, tanto de implementação, quanto de elaboração. Entre as dificuldades encontradas, destacam-se as citadas abaixo.

6.1.1 Escassez de referências

Foi notada uma escassez de publicações acadêmicas sobre o assunto. A maioria das referências encontradas eram páginas da internet, frequentemente sem as referências adequadas e com terminologias confusas e por vezes contraditórias.

6.1.2 Escolha dos algoritmos e parâmetros

A escolha dos algoritmos a ser usada na ferramenta proposta por este trabalho foi uma fonte de dificuldades. O motivo principal é a escassez de publicações acadêmicas sobre o assunto. Outro motivo é devido à complexidade de alguns algoritmos, que acabaram não sendo abordados pela ferramenta.

Além disso, definidos os algoritmos, dificuldades na escolha dos parâmetros foram encontradas. A maioria dos algoritmos permitia vasta flexibilidade, o que tornou confusa a definição de parâmetros significativos.

6.2 Trabalhos futuros

A ambição inicial do projeto era de fornecer uma ferramenta completa e extensível de geração de mapas. Porém, por causa de limitações de tempo, nem todas as funcionalidades planejadas foram implementadas e o escopo do trabalho foi reduzido.

Como futuros trabalhos, as seguintes funcionalidades poderiam ser programadas:

Mais algoritmos de PCG

Uma melhoria óbvia seria a implementação de mais algoritmos de PCG, como o diagrama de Voronoi, a triangulação de Delaunay, autômatos celulares de outras regras, etc.

Melhor parametrização

Uma das dificuldades encontradas foi a parametrização dos algoritmos. Assim, a escolha feita para a implementação desta ferramenta provavelmente pode ser melhorada, levando em considerações outros fatores como, proporção, continuidade, conjuntos pré-fabricados, etc.

Interface gráfica

Uma melhoria substancial seria a implementação de uma interface gráfica, agindo como *front-end* da ferramenta. Desta maneira, a criação de mapas seria mais prática e intuitiva.

6.3 Conclusão

Como produto deste trabalho, foi desenvolvida uma ferramenta para geração de mapas bidimensionais, usando diversos algoritmos estudados, a fim de prover uma ferramenta útil para diversos jogos de plataforma, principalmente do gênero mundo aberto, como o jogo *Terraria*.

Através do estudo de vários algoritmos e da pesquisa de vários jogos, foi possível obter conhecimento necessário para a realização dessa ferramenta. O estudo sobre conceitos de geração procedural foi particularmente útil na elaboração deste texto para explicar os termos usados aqui.

Concluímos que fora desenvolvida uma ferramenta capaz de gerar mapas em formatos conhecidos, oferecendo um leque de opções para o uso por outros programadores de jogos. A ferramenta pode ser expandida para vários outros algoritmos e métodos, uma vez que a estrutura do código foi concebida para ser expansível e versátil.

7 REFERÊNCIAS BIBLIOGRÁFICAS

[1] “Explanation of Midpoint Displacement”. [Online] Disponível em https://code.google.com/p/fractalterraingeneration/wiki/Midpoint_Displacement. [Acessado em 4-julho-2014]

[2] “Generating Random Fractal Terrain”. [Online] http://www.gameprogrammer.com/fractal.html [Acessado em 17-setembro-2013]

[3] “Random Walk”. [Online] http://en.wikipedia.org/wiki/Random_walk [Acessado em 22-outubro-2013]

[4] “Drunkard Walk”. [Online] http://pcg.wikidot.com/pcg-algorithm:drunkard-walk [Acessado em 22-outubro-2013]

[5] “Procedural Generation – The Caves”. [Online] http://noelberry.ca/2011/04/procedural-generation-the-caves/ [Acessado em 12-novembro-2013]

[6] “Cellular Automaton”. [Online] http://en.wikipedia.org/wiki/Cellular_automaton [Acessado em 2-dezembro-2014]

[7] “TMX Map Format”. [Online] https://github.com/bjorn/tiled/wiki/TMX-Map-Format [Acessado em 19-novembro-2014]

[8] “BMP File Format”. [Online] http://en.wikipedia.org/wiki/Bitmap_file_format [Acessado em 25-junho-2014]

[9] “Simple DirectMedia Layer”. [Online] https://www.libsdl.org/ [Acessado em 3-junho-2014]

[10] “What Pcg Is”. [Online] http://pcg.wikidot.com/what-pcg-is [Acessado em 11-julho-2013]

[K] “The Death of the Level Designer: Procedural Content Generation in Games - Part One”. [Online] http://roguelikedeveloper.blogspot.com.br/2008/01/death-of-level-designer-procedural.html [Acessado em 12-julho-2013]

[L] “The roots of open-world games”.[Online] http://www.gamesradar.com/the-roots-of-open-world-games/ [Acessado em 12-julho-2013]

[M] “Game Design Essentials: 20 Open World Games”. [Online] http://www.gamasutra.com/view/feature/1902/game_design_essentials_20_open_.php [Acessado em 20-agosto-2013]

[N] “Sandbox”. [Online] http://www.techopedia.com/definition/3952/sandbox-gaming [Acessado em 20-agosto-2013

[O] “Landstalker: Treasure of King Nole for Genesis [1992]”. [Online] http://www.mobygames.com/game/landstalker-treasure-of-king-nole [Acessado em 20-agosto-2013]

[P] “MegaDrive/Genesis Review - Landstalker”. [Online] http://www.outofprintarchive.com/articles/reviews/MegaDrive/Landstalker-GameFan1-3-1.html [Acessado em 21-agosto-2013]

[Q] “This is Minecraft”. [Online] http://www.ign.com/articles/2010/11/11/this-is-minecraft [Acessado em 21-agosto-2013]

[R] “Minecraft” . [Online] http://pt.wikipedia.org/wiki/Minecraft [Acessado em 21-agosto-2013]

[S] “Elite (video game)” . [Online] http://en.wikipedia.org/wiki/Elite_%28video_game%29 [Acessado em 13-julho-2014]

[T] “The History of Elite: Space, the Endless Frontier” .[Online] http://www.gamasutra.com/view/feature/3983/the_history_of_elite_space_the_.php [Acessado em 13-julho-2014]

[T2] “The Elder Scrolls”. [Online] http://en.wikipedia.org/wiki/The_Elder_Scrolls [Acessado em 10-julho-2013]

[T3] “Bethesda: The Right Direction”. [Online] http://www.escapistmagazine.com/articles/view/video-games/issues/issue_83/471-Bethesda-The-Right-Direction [Acessado em 10-julho-2013]

[U] “Terraria” . [Online] http://en.wikipedia.org/wiki/Terraria [Acessado em 10-julho-2013]

[W] “Procedural Content Generation: Thinking With Modules” [Online]http://gamasutra.com/view/feature/174311/procedural_content_generation_.php [Acessadoem 15-fevereiro-2014]

[X] “Procedural Content Generation” [Online] http://www.gamecareerguide.com/features/336/procedural_content_.php [Acessado em 15-fevereiro-2014]

[Y] “Procedural Generation” [Online] http://diablo.gamepedia.com/Procedural_Generation[Acessado em 17-fevereiro-2014]

[Z] “Cellular Automata” [Online] http://pcg.wikidot.com/pcg-algorithm:cellular-automata

[Acessado em 21-fevereiro-2014]

[aA] “Cellular Automata Method for Generating Random Cave-Like Levels” [Online] http://www.roguebasin.com/index.php?title=Cellular_Automata_Method_for_Generating_Random_Cave-Like_Levels[Acessado em 25-fevereiro-2014]

[aB] “John Conway’s Game of Life” [Online] http://www.bitstorm.org/gameoflife/ [Acessado em 25-fevereiro-2014]

[aC] “Jogo da vida” [Online] http://pt.wikipedia.org/wiki/Jogo_da_vida [Acessado em 02-janeiro-2014]

[aD] “Voronoi Diagram” [Online] http://pcg.wikidot.com/pcg-algorithm:voronoi-diagram[Acessado em 03-janeiro-2014]

[aE] “Diagrama de Voronoi e suas aplicações em SIG” [Online] http://andersonmedeiros.com/diagrama-de-voronoi-aplicacoes-sig/[Acessado em 04-janeiro-2014]

[aF] “Diagrama de Voronoi” [Online] http://www.ime.usp.br/~freitas/gc/voronoi.html [Acessado em 04-janeiro-2014]

[aG] “Fortune’s algorithm and implementation” [Online] http://blog.ivank.net/fortunes-algorithm-and-implementation.html [Acessado em 05-janeiro-2014]

[aH] “Delaunay Triangulation” [Online] http://www.cs.uu.nl/geobook/interpolation.pdf [Acessado em 12-janeiro-2014]

[aI] “Voronoi Dungeon Generator” [Online] http://www.avanderw.co.za/voronoi-dungeon-generator/ [Acessado em 14-janeiro-2014]

[aJ] “Diablo” [Online] http://pt.wikipedia.org/wiki/Diablo [Acessado em 18-janeiro-2014]

[aK] “Diablo (series)” [Online] http://en.wikipedia.org/wiki/Diablo_(series) [Acessado em 18-janeiro-2014]

[aL] “Randomization” [Online] http://www.diablowiki.net/Randomization [Acessado em 18-janeiro-2014]

[aM] “Open World” [Online] http://en.wikipedia.org/wiki/Open_world [Acessado em 19-janeiro-2014]

[aN] “Linearity” [Online] http://www.whatgamesare.com/linearity.html [Acessado em 22-fevereiro-2014]

[aO] “Procedural Generation” [Online] http://en.wikipedia.org/wiki/Procedural_generation [Acessado em 20-fevereiro-2014]

[aP] “Procedural Generation” [Online] http://www.whatgamesare.com/procedural-generation.html [Acessado em 20-fevereiro-2014]

[aQ] Togelius, J., Yannakakis, G.N, Stanley, K.O., Browne, C., *Search-based Procedural Content Generation*

[aM2] “Beat Hazard Wiki” [Online] http://beathazard.wikia.com/wiki/Beat_Hazard_Wiki [Acessado em 21-fevereiro-2014]

[aN2] “Beat Hazard (Game)” [Online] http://www.giantbomb.com/beat-hazard/3030-29407/ [Acessado em 22-fevereiro-2014]

[aO2] “Beat Hazard” [Online] http://store.steampowered.com/app/49600/ [Acessado em 22-fevereiro-2014]

[aP2] “The Dwarf Fortress Wiki” [Online] http://dwarffortresswiki.org/index.php/Main_Page [Acessado em 8-março-2014]

[aQ2] “Bay 12 Games: Dwarf Fortress” [Online] http://www.bay12games.com/dwarves/features.html [Acessado em 9-março-2014]

[aR] “Dwarf Fortress” [Online] http://en.wikipedia.org/wiki/Dwarf_Fortress[Acessado em 11-março-2014]

[aS] “Dwarf Fortress: Ten hours with the most inscrutable video game of all time” [Online] http://arstechnica.com/gaming/2013/02/dwarf-fortress-ten-hours-with-the-most-inscrutable-video-game-of-all-time/ [Acessado em 21-março-2014]

[aT] “Dwarf Fortress” [Online] http://www.rockpapershotgun.com/tag/dwarf-fortress/ [Acessado em 11-março-2014]

[aU] “DungeonMaker” [Online] http://pcg.wikidot.com/pcg-software:dungeonmaker [Acessado em 12-março-2014]

[aV] “Terragen 3” [Online] http://planetside.co.uk/products/terragen3 [Acessado em 12-março-2014]

[aX] “DungeonMaker Manual” [Online] http://dungeonmaker.sourceforge.net/DM2_Manual/manual1.html [Acessado em 14-março-2014]

[aY] “Terragen” [Online] http://en.wikipedia.org/wiki/Terragen [Acessado em 14-março-2014]

[aZ] “Flag word” [Online] http://en.wikipedia.org/wiki/Flag_word [Acessado em 15-março-2014]