UNIVERSIDADE DE ITAÚNA
PRÓ-REITORIA DE ENSINO
COORDENAÇÃO DE CIÊNCIA DA COMPUTACAO
BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

GUILHERME NOGUEIRA TORRES

ALGORITMOS DE GERAÇÃO PROCEDURAL DE CAVERNAS PARA JOGOS DIGITAIS

ITAÚNA
2019

GUILHERME NOGUEIRA TORRES

ALGORITMOS DE GERAÇÃO PROCEDURAL DE CAVERNAS PARA JOGOS DIGITAIS

Projeto submetido à Coordenadoria do Curso de Bacharelado em Ciência da Computação da Universidade de Itaúna - Campus Verde, como requisito parcial para obtenção do grau de Bacharel em Ciência da Computação.

Área de pesquisa: Jogos Digitais

Orientador: Thiago Silva Vilela

Itaúna
2019

UNIVERSIDADE DE ITAÚNA
COORDENAÇÃO DE CIÊNCIA DA COMPUTAÇÃO

GUILHERME NOGUEIRA TORRES

Este projeto foi julgada adequada para a obtenção do Grau de Bacharel em Ciência da Computação, sendo aprovada pela coordenação de ciência da computação do curso de Bacharelado em Ciência da Computação do Campus verde da Universidade de Itaúna Ceará e pela banca examinadora:

Orientador: Thiago Silva Vilela
Universidade de Itaúna- UIT

Prof. Helbert Miranda
Universidade de Itaúna- UIT

Itaúna, 15 de Junho de 2019

Dedico este trabalho a toda a minha família e amigos que me apoiaram desde sempre.

“Não é sobre mim, nem sobre você. É sobre o legado,
o legado que deixamos para as gerações futuras.”

Tony Stark

Resumo

A indústria de jogos tem crescido muito nos últimos anos tornando-se uma das maiores indústrias de entretenimento do mundo, e por isso cada vez mais os jogos tentam aumentar as horas de conteúdo fornecida por eles fazendo valer a pena o dinheiro pago.

A geração procedural de conteúdo em jogos é um elemento que aumenta essas horas de conteúdo com um custo baixo de desenvolvimento podendo ser aplicado em qualquer área do jogo, desde a programação dos sistemas até o desenvolvimento de mapas, personagens, etc.

Por ser uma técnica ampla ela possui inúmeras aplicações e formas de implementação. Esse trabalho vai explorar uma de suas aplicações, a geração procedural de cavernas, e estudar e comprar alguma de suas implementações, comparando os resultados para apontar os algoritmos que mais se sobressaem em determinada situação.

Palavras Chave: Algoritmos de Geração Procedural, Jogos Digitais, Cavernas.

Abstract

The games industry has grown a lot in the passing years, and is now one of the world's largest industry. A trend games have been following is to try and increase the amount of content they provide.

Procedural content generation in games allows an increase of content with low cost of development and can be applied to many parts of a game, like the creation of characters, generations of maps and more.

This paper will explore PCG for generating dungeons, a problem also known as Procedural Dungeon Generation. Some algorithms for procedural dungeon generation will be studied, implemented and compared. At the end of the research, it should be possible to point advantages and disadvantages of each algorithm, as well as situations best suited for each one.

Sumário

Lista de Figuras

Lista de Abreviacoes

PCG	Geração Procedural de Conteúdo (*Procedural Content Generation*)
CA	*Cellular Automata*
RV	*Replay Value*
GG	*Generative Grammar*

CAPÍTULO 1

INTRODUÇÃO

1.1 Motivação

Um dos maiores desafios no desenvolvimento de qualquer jogo digital é apresentar uma grande quantidade de conteúdo de forma clara e imersiva, que não seja nem tão difícil e nem tão fácil, para que os jogadores possam se divertir e ter um desafio ao mesmo tempo. Gerar conteúdo para um jogo digital envolve, em geral, grande esforço humano. Além disso, o valor de entretenimento de um jogo está diretamente associado à quantidade de conteúdo que ele oferece. Quanto mais conteúdo, em geral, maior o *replay value*, ou rejogabilidade, que indica o potencial para que um jogo seja jogado mais de uma vez.

Mas como apresentar um conteúdo sem que ele fique repetitivo demais? Vários títulos de várias empresas têm adotado os algoritmos de geração procedural, que consistem em gerar parte do jogo de forma aleatória.

A geração procedural pode ser usada de inúmeras formas para atingir os mais diversos objetivos, como: gerar mapas inteiros como nos jogos *Civilizations* (MICROPROSE, 1991) e *Minecraft* (SPECIFICATIONS, 2009), gerar criaturas e vegetações diferentes, como no *No Man's Sky* (GAMES, 2016), gerar cavernas para exploração, como no *Diablo* (ENTERTAINMENT, 1996). Além de vários outras funcionalidades.

No jogo *Diablo*, por exemplo, o jogador encontra inúmeras cavernas durante a sua aventura, que são exploráveis por ele. Essas cavernas são geradas aleatoriamente toda vez que o jogador entra no jogo tornando a experiência única, mesmo que ele entre em uma mesma caverna repetidas vezes. A geração procedural de cavernas não é algo exclusivo desse título, sendo um método comum em vários outros jogos.

Existe um gênero de jogo denominado *Dungeon Crawler* que é voltado para geração procedural de cavernas. Nesse gênero o jogador deve explorar uma caverna

e vencer os vários desafios apresentados por ela que, na maioria das vezes, também são gerados aleatoriamente. Cada caverna é composta por vários andares ou fases e, quanto mais fases o jogador conseguir explorar, maior será a sua pontuação no jogo.

Em 1980 foi lançado Rogue, o primeiro jogo desse gênero, sendo também o primeiro jogo tendo a geração procedural de cavernas (COMPTON; MATEAS, 2006) e, desde então, vários algoritmos e técnicas foram propostos para gerar não só cavernas, mas diversos conteúdos em jogos digitais. No entanto, a área ainda apresenta diversos desafios e poucos estudos aprofundados sobre esses algoritmos.

1.2 Problema de Pesquisa

Com a necessidade de se gerar cavernas de forma procedural em jogos digitais, quais são as diferentes formas de gerá-las? E quais as vantagens e desvantagens de cada abordagem?

1.3 Objetivos

Com o passar do tempo os jogos têm ficado cada vez maiores: usam mais processamento, têm mapas maiores e com mais agentes e permitem mais interações. Por isso, os algoritmos de geração procedural de cavernas estão cada vez mais complexos, gerando cavernas maiores, mais robustas e de forma mais rápida.

Com isso, esse trabalho irá estudar as diferentes técnicas de se implementar um algoritmo de geração procedural de cavernas e suas vantagens e desvantagens em relação a outras técnicas, além de fazer comparações entre os resultados obtidos após a implementação de alguns algoritmos selecionados.

1.3.1 Objetivos Gerais

O objetivo geral desse trabalho é estudar os vários algoritmos de geração procedural de cavernas e avaliá-los em relação a eficiência, tempo de execução e a parametrização dos algoritmos, utilizando diversos conjuntos de testes e métricas de comparação.

1.3.2 Objetivos Específicos

De forma específica pretendemos, com esse trabalho:

- Analisar e avaliar diferentes técnicas e algoritmos usados para a geração procedural de cavernas em jogos digitais.
- Mostrar, de forma detalhada, vantagens e desvantagens das diferentes técnicas estudadas. As vantagens e desvantagens podem ajudar na escolha de certas técnicas para certos tipos de jogos.
- Implementar e fornecer uma ferramenta *open source* para a plataforma Unity3D (TECHNOLOGIES, 2005) que permite a geração procedural de cavernas usando as diferentes técnicas estudadas.

CAPÍTULO 2

TRABALHOS RELACIONADOS

Aina não temos, na literatura, muitos estudos aprofundados sobre a comparação direta dos algoritmos de geração procedural de cavernas, nem em quais situações eles produzem os melhores resultados, abrindo assim a oportunidade para tal estudo.

Entretanto existem muitos trabalhos falando sobre PCG, a maioria deles fala sobre um algoritmo específico ou discute as vantagens e desvantagens de um algoritmo em relação a outro, como em (LINDEN; LOPES; BIDARRA, 2013), o qual os autores discutem sobre os algoritmos *Cellular Automata*, *Generative Grammars*, Algoritmos Genéticos, dentre outros.

Os autores desse artigo, após a comparação dos algoritmos, concluíram que não existe um único algoritmo que fará todos os tipos de caverna, pois cada caverna tem seu requerimento e cada jogo tem seus elementos particulares. Entretanto usar um algoritmo de geração procedural, mesmo que ele não seja o mais rápido entre os algoritmos para gerar uma caverna, ainda é mais rápido do que criar cada caverna a mão, poupando tempo de desenvolvimento e dinheiro. O desafio é saber qual algoritmo escolher dependendo do nível de controle requerido na caverna, o design dela, entre outros fatores.

Dado essa conclusão, comparar os algoritmos e saber em quais situações quais se sobressaem e em quais eles não apresentam bons resultados é fundamental para o desenvolvimento desse trabalho.

Nepozitek (NEPOZITEK, 2018) faz um estudo aprofundado do algoritmo *Generative Grammar* além de sugerir algumas mudanças para que ele fique mais rápido. Depois o autor faz uma comparação de velocidade entre o algoritmo antigo e o modificado baseado em iterações, o que pode ser usado nesse trabalho, uma vez que os processadores apresentam tempos diferentes, e outros processos podem estar rodando ao lado do processo do jogo, o que pode causar interferência nos resultados caso a comparação seja feita por tempo e não por iteração.

Alguns autores falam sobre PCG de uma forma geral, tendo a geração de caverna como seu principal exemplo como os autores dos artigos (TOGELIUS; JUSTINUSSEN; HARTZEN, 2012) e (DORMANS; LEIJNEN, 2013). Ambos os artigos mostram que a PCG está atrelada ao *Game Design*, um conjunto de decisões que torna o jogo agradável e imersivo de se jogar, além das regras de dificuldades que não deixam que o jogo se torne impossível, mas também não seja fácil demais a ponto de ser monótono.

Como Gillian Smith fala em seu artigo (SMITH, 2014), criando novos sistemas com um estudo mais aprofundado em Geração Procedural de Conteúdo, os jogos podem se preocupar mais em seus *designs* e menos nos sistemas, produzindo jogos com mais qualidade e mais re-jogabilidade e diminuindo seus custo de produção.

CAPÍTULO 3

FUNDAMENTAÇÃO TEÓRICA

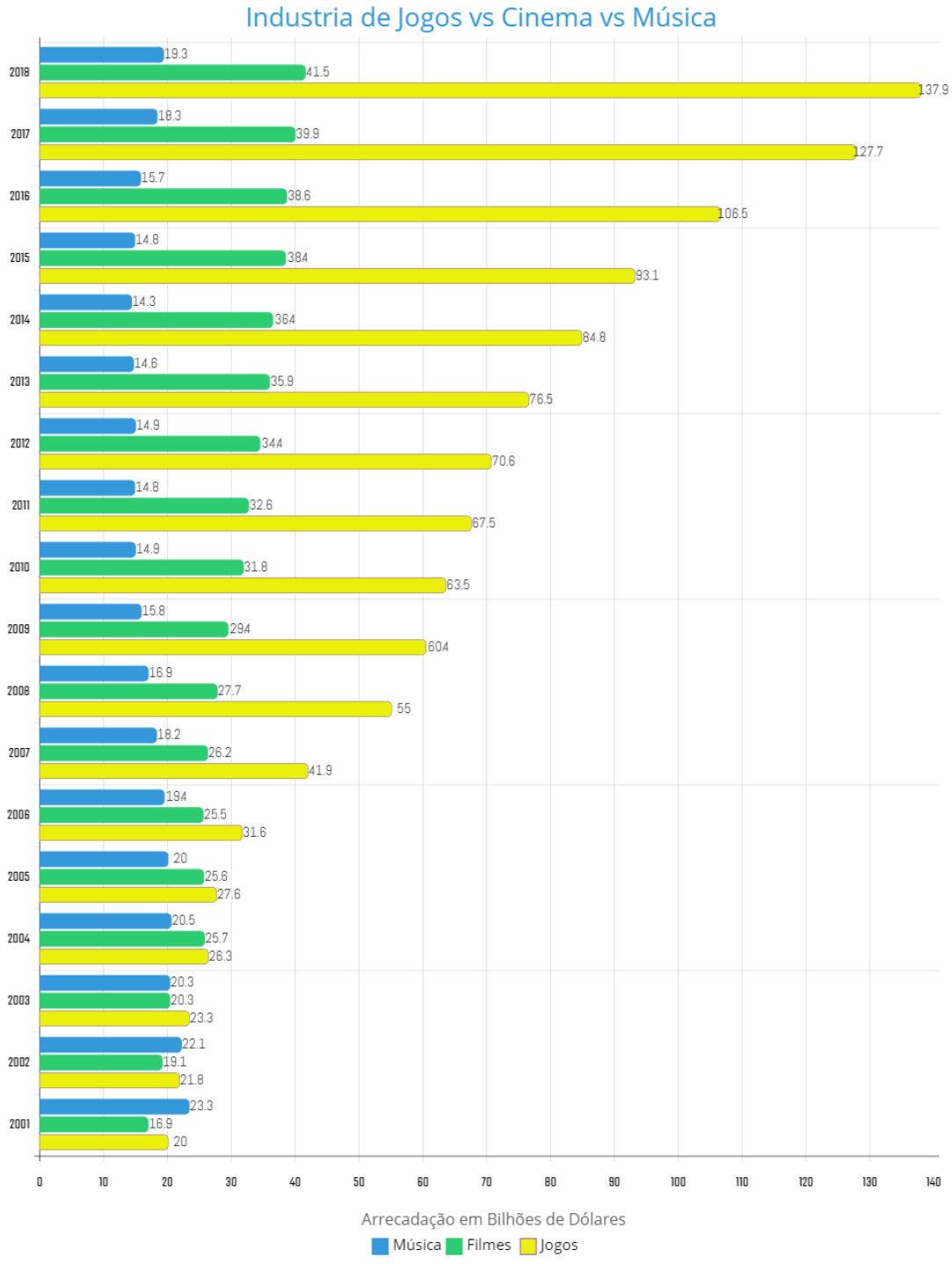
Nesse capítulo discutiremos, na seção 3.1, os conceitos básicos de jogos digitais, como surgiu e sua influência nos dias de hoje. Na seção 3.2, explicaremos qual a utilidade da PCG nos jogos digitais e seu impacto nos jogos. Na seção seguinte, 3.3, discutiremos o que é uma caverna em um jogo digital e seus atributos desejáveis, comentando sobre o *design*. Finalmente, na seção 3.4, falaremos sobre alguns dos algoritmos procedurais geradores de cavernas.

3.1 Jogos Digitais

Durante a década de 1940 a Teoria de Jogos foi proposta pelos matemáticos John von Neumann e Oskar Morgenstem, que passou a ser discutida e sedimentada até que, em meados da década de 1970, começaram a surgir os primeiros jogos digitais (LUCCHESE; RIBEIRO, 2009). Desde então os jogos digitais têm sido uma forma de entretenimento, cuja a industria tem crescido cada vez mais, tendo hoje uma receita maior que a industria de filmes e música juntos como mostrado na Figura 1 (ESPORTS, 2018).

Com o notável crescimento da industria de jogos, as desenvolvedoras se preocupam cada vez mais com o *replay value*, o quanto um jogo pode ser re-jogado sem perder sua essência, aumentando a vida útil do jogo. Alguns dos jogos mais jogados no presente usam desse fator como seu principal atributo, como *League of Legends* (GAMES, 2009) e *Fortnite* (GAMES, 2017), sendo que o segundo título usa da geração procedural para dar ao jogador equipamentos randômicos em partes específicas do mapa tornando cada jogo único mesmo que o mapa seja o mesmo.

Figura 1: Crescimento da industria de jogos em relação as industrias de música e cinema



3.2 Geração Procedural

O termo *Procedural Content Generation*, do inglês Geração Procedural de Conteúdo, pode ser definido mais facilmente se separarmos os termos. *Content*

(Conteúdo) nesse caso por se tratar de jogos pode ser definido como o que um jogo contem, por exemplo os personagens, armas, veículos, história, objetivos, mapas, etc. E *Procedural Generation* é um ou vários processos que o computador vai executar de forma procedural para gerar um resultado, sendo esse resultado algum dos conteúdos citados (SHAKER; TOGELIUS; NELSON, 2016).

A geração procedural, apesar de não ser o único fator que aumenta o RV de um jogo, tem sido bastante usada como já foi mostrado. E mesmo tendo um conceito simples, ela pode ser implementada e executada de várias formas diferentes, inclusive no mesmo jogo. Como é o caso da série de jogos *Diablo*, que já teve seu primeiro título citado anteriormente. No jogo, além das cavernas serem geradas proceduralmente, o equipamento que do personagem do jogador também usa essa técnica. O personagem pode equipar diversos itens diferentes como: capacete, armadura, armas e etc. e ao derrotar um monstro ele pode deixar cair um ou mais desses itens que terá seus atributos como: aparência, resistência ou o valor de ataque, gerado proceduralmente. Mesmo um sistema PCG podendo ser implementado em vários aspectos de um jogo, existem algumas propriedades do sistema que são esperadas como:

- **Velocidade**: o conteúdo tem que ser gerado da forma mais rápida possível, tanto se o conteúdo for gerado *in-game*, ou seja, em tempo de execução do jogo, para que o jogador não fique muito tempo em uma tela de carregamento (*loading*), quanto se for no gerado na hora do desenvolvimento do jogo, para que os desenvolvedores não tenham que perder muito tempo esperando o sistema entregar os resultados.
- **Credibilidade**: O conteúdo que for gerado pelo sistema tem que convencer o jogador que aquilo faz parte de jogo, apresentando diversidade para que seja algo realmente novo, e não só mais do mesmo, além de ter expressividade fazendo com que o sistema ajude na imersão do jogador ao jogo.
- **Controlabilidade**: Para os desenvolvedores é importante que o sistema possa ser reusado para evitar trabalho desnecessário e aproveitar ao máximo do sistema, por isso o sistema que implementa PCG precisa ser o mais parametrizável possível, assim o sistema pode gerar resultados mais específicos para casos diferentes, mantendo assim um mesmo sistema, mas usado em várias partes do jogo. Por exemplo em PCG de cavernas para um *Dungeon Crawler* ambientando em vários lugares diferentes, como florestas, desertos,

vulcões, etc., deve ter parâmetros para que todos esses cenários possam ser gerados sem a necessidade de alterar o sistema.

A maioria dos algorítimos de PCG propostos em geração de cavernas abrem mão de uma propriedade para ganhar em outra. Fazer um sistema que cobre as três propriedades pode envolver a implementação de mais de um sistema de forma com que um algorítimo decidira qual dos sistemas executará baseado nos parâmetros passados pelo usuário.

3.3 Cavernas e Level Desing

O conceito de uma caverna, ou calabouço, que pode ser explorado por um ou mais jogadores trazendo desafios perigosos e uma recompensa, foi introduzido, de acordo com um dos criadores do jogo de tabuleiro *Dungeons & Dragons* de 1974, em um *wargame*[1] no qual os jogadores se divertiam tanto explorando a caverna que logo virou um gênero, *Dungeon Crawler* (KUNTZ; GYGAX, 2004).

Cada jogo tem seu próprio estilo de caverna e alguns jogos apresentam até mais de um estilo. Algumas cavernas são mais naturais, outra parecem ser feitas por humanos, algumas tem mais armadilhas, outras mais monstros para o jogador enfrentar, algumas tem mais labirintos, outras mais corredores, a variedade é enorme.

As figuras 2, 3 e 4 mostram alguns exemplos de cavernas em seus respectivas jogos, demonstrado a grande variedade das mesmas.

Figura 2: Mapa de uma caverna no jogo The Binding of Isaac

[1] Wargame é um jogo jogando com figuras de ação representando unidades de guerra e dados para determinar resultados.

Figura 3: Caverna gerada no jogo Nuclear Throne, as partes separadas da caverna são salas que o jogador pode explorar. o jogo teletransporta o jogador para a porta dessa sala quando o mesmo interage com a porta correspondente na caverna e vice-versa.

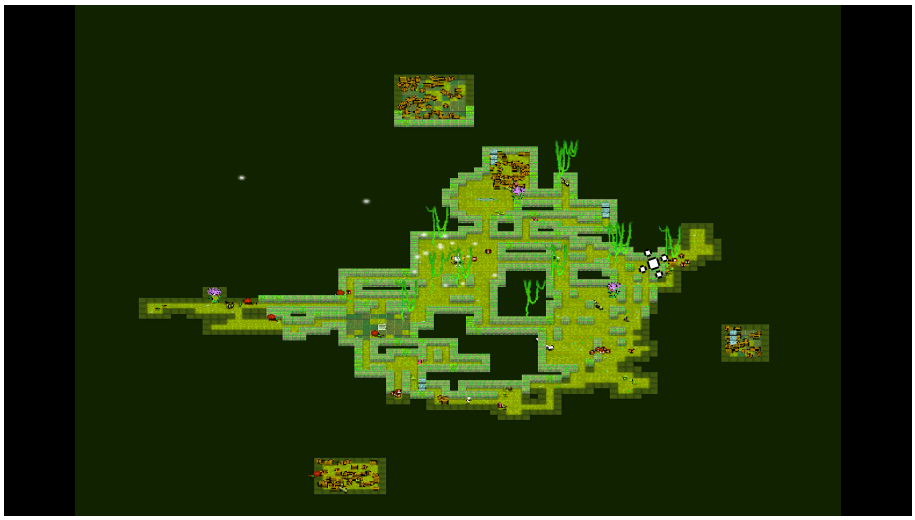


Figura 4: Mapa de uma caverna no jogo Diablo 3

Apesar da estrutura de uma caverna ser importante, o seu conteúdo também é de grande relevância. O *level design* da caverna compreende tanto sua estrutura

quanto seu conteúdo. De acordo com o site Techopedia, *level design* começa desde a concepção de um cenário até a criação de suas regras juntamente com a sua aparência, condições climáticas, ambientação, interação com o jogador, etc, sendo um processo tanto técnico quanto artístico (TECHOPEDIA, 2019).

Mike Stout no seu artigo "Learning From The Masters: Level Design In The Legend Of Zelda"(STOUT, 2012), divide o *level design* de uma caverna em quatro partes, *Level Flow* (Fluidez da Fase), *Intensity Ramping* (Rampa de Intensidade), *Variety* (Variedade) e *Training* (Treinamento).

- A **Fluidez da Fase** diz respeito ao caminho que o jogador deve pegar para sair do inicio dela e chegar ao fim, mas além disso a fase deve conter caminhos opcionais para que o jogador tome decisões ao longo do caminho além de poder receber recompensas bônus por escolher pegar o caminho mais longo até o objetivo.
- A **Rampa de Intensidade** define, em uma caverna, o quão difícil é desde a primeira sala até a última, se essa dificuldade vai aumentando ao longo da caverna.
- A caverna também deve apresentar **Variedade** em seu conteúdo. Ter inimigos diferentes uns dos outros, várias armadilhas com desafios diferentes, as próprias salas da caverna devem aparacer diferentes umas das outras.
- E por fim, mas não menos importante, o **Treinamento** que está ligado a Rampa de Intensidade. A medida que a caverna vai ficando mais difícil, isso é mostrado ao jogador de forma clara? Existem tutorias para apresentar uma nova mecânica? As novas armadilhas são claramente diferentes para que o jogador saiba como evitar cada uma delas?

O algoritmo de Geração Procedural de Caverna deve ter uma preocupação com o *design*, não necessariamente popular a caverna com seu conteúdo, mas saber que eles existem para que a caverna siga os padrões desejáveis descritos anteriormente.

3.4 Algoritmos de Geração Procedural de Cavernas

A seguir discutiremos algumas das inúmeras implementações dos algoritmos de geração procedural de cavernas, existem inúmeros outros. Alguns dos mais famosos

são o *Cellular Automata*, *Generative Grammars*, e Algorítimos Genéticos, como citado no *Procedural Generation of Dungeons* (LINDEN; LOPES; BIDARRA, 2013). Cada um deles possui sua vantagem e desvantagem dada uma situação. Escolhemos esses algoritmos por terem abordagens bem diferentes um dos outros e gerarem cavernas com diferentes propostas, dessa forma cobrindo uma grande gama de jogos.

Mas a estrutura da caverna não é tudo, o conteúdo dela também é de suma importância, como baús de tesouro, como recompensa para o jogador durante a sua jornada, monstros e/ou criaturas para aumentar o seu desafio e armadilhas para deixá-lo atento aos detalhes da caverna. E apesar da maioria dos algoritmos que serão apresentado formarem apenas a estrutura dela, deve-se ter uma preocupação com seu conteúdo, dado que todos esses agentes e objetos serão posicionados posteriormente na caverna.

Além disso existem diferentes formas de executar os algoritmos. A execução pode ser feita durante a produção do jogo, de forma que o jogador não terá que esperar a caverna a ser gerada, entretanto ficará limitado às que já foram previamente colocadas no jogo. Ou pode ser feita durante o tempo de execução, causando efeitos contrários a opção anterior: o jogador terá que esperar a geração, entretanto não será limitado quanto ao número de gerações embutida no jogo. Outra possibilidade é a geração remota e online, o que permite que o dispositivo com o qual o jogador está jogando não tenha que ser potente para executar os algoritmos, entretanto ele terá que ter uma conexão com a internet que satisfaça o tráfego da rede gerado pelo algorítimo.

3.4.1 Algoritmo de Nuclear Throne

No jogo Nuclear Throne (VLAMBEER, 2015), as cavernas são feitas usando uma espécie de "máquia de fazer caverna". Basicamente, ele considera que toda a grade de *Tiles* [2] é uma parede. Então o algoritmo cria uma maquina de fazer caverna inicial e ela é posicionada no centro da grade.

A máquina vai em linha reta e todo Tile que ela visita ela transforma em chão, a cada iteração, baseado nos parâmetros passados a ela, ela pode virar 90º para a esquerda, 90º para direita, continuar em linha reta ou virar 180º, e caso o Tile já seja chão ela ignora. Existe uma chance de a cada iteração essa máquina criar outra maquina com as mesmas propriedade, porém com uma menor chance de criar outra

[2] Tile é a porção mínima de um mapa

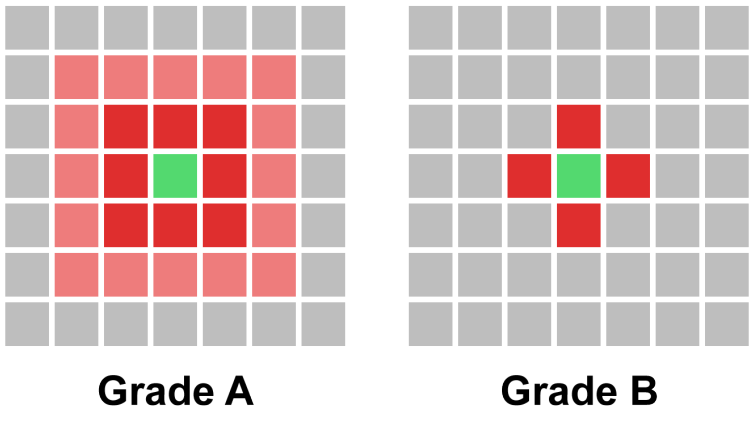
maquina, como mostrado por (NOVA, 2016).

3.4.2 Cellular Automata

O algoritmo CA foi desenvolvido por volta da década de 1950 por John von Neumann e Stanislaw Ulam (SCIENCE, 2002). Existem inúmeras adaptações e abordagens para o algoritmo. Será discutido apenas o essencial para o entendimento para a aplicação em um gerador de cavernas.

O algoritmo cria um grade de células, normalmente bidimensional, na qual as células possuem regras, ou comportamentos descritos. Esses comportamentos ou regras são baseados em seus vizinhos, sendo que os vizinhos podem ser considerados de forma diferentes vamos abordar as duas mais comuns, *Moore Neighbourhood* e *Von Neumann Neighbourhood*. Na figura a seguir vemos essas implementações, na grade A os vizinhos (em vermelho) são feitos em camadas (tons de vermelho diferentes representam camadas diferentes) sendo considerados todas as células em volta da célula selecionada (em verde). Na grade B vemos que os vizinhos (em vermelho) considerados são apenas as células ao norte, sul, leste e oeste da célula selecionada (em verde) (NIEMANN; PREUß, 2015).

Figura 5: Diferentes formas de se definir vizinhanças no algoritmo CA



Cada célula tem um número finito de estados que ela pode estar, normalmente são dois (vermelho ou verde, vivo ou morto, chão ou parede, etc), e baseado em seus vizinhos e as regras a eles atribuídos seus estados mudam ou não. Inicialmente a grade é preenchida com células em diferentes estados, esse preenchimento pode ser

um parâmetro ou feito randomicamente, depois baseado em um número n de iterações cada célula será consultada e verificado suas regras para que ela mude ou não seu estado.

As regras, normalmente, vão estar relacionadas aos vizinhos da célula selecionada. Por exemplo, supondo a implementação dos vizinhos de Von Neumann e cada célula tendo dois estados, vivo ou morto. Uma regra seria: "Se o vizinho norte estiver vivo e o leste também, essa célula morre."Assim toda célula que encaixar nessa regra terá seu estado mudado para morto, caso já não seja.

Para a aplicação em jogos digitais, mais especificamente gerando uma caverna, um conjunto de regras é definido sendo que o mesmo fará com que o CA gere estruturas que se pareçam com cavernas. Em alguns casos o formato gerado pode criar mais de uma estrutura fazendo com que nem toda a caverna seja explorada. Sebastian Lague (LAGUE, 2015) discute esse problema definindo uma sala principal e depois procurando por conexões entre as outras salas que não estão conectadas a ela.

3.4.3 Rooms and Mazes

Esse conjunto de algoritmos denominado "*Rooms and Mazes*" foi discutido por (NYSTROM, 2014) onde ele segue vários passos utilizando vários algoritmos e ao final deles uma caverna é formada.

Primeiro o algoritmo cria uma grade de Tiles e coloca todas as Tiles em um estado nulo (que ainda não faz parte da caverna). Depois ele gera várias salas com alturas e larguras diferentes que, basicamente, são um aglomerado de Tiles do tipo ’Chão’ cercados de Tiles do tipo ’Parede’. Então o algoritmo tenta posicionar uma sala gerada na grade de Tiles e, caso ela não interceda nenhuma outra sala, isto é, se nenhum dos Tiles da nova sala tocarem algum Tiles do tipo ’Chão’, a sala é posicionada, como exemplificado nas figuras 6 e 7.

Figura 6: Na Grade A foi posicionado uma sala (’Parede’ na cor vermelha e ’Chão’ na cor verde); e na Grade B vemos uma nova sala (’Parede’ na cor azul e ’Chão’ na cor amarela) que não pode ser posicionada pois um dos seus Tiles toca um Tile ’Chão’ da sala que foi posicionada antes dela

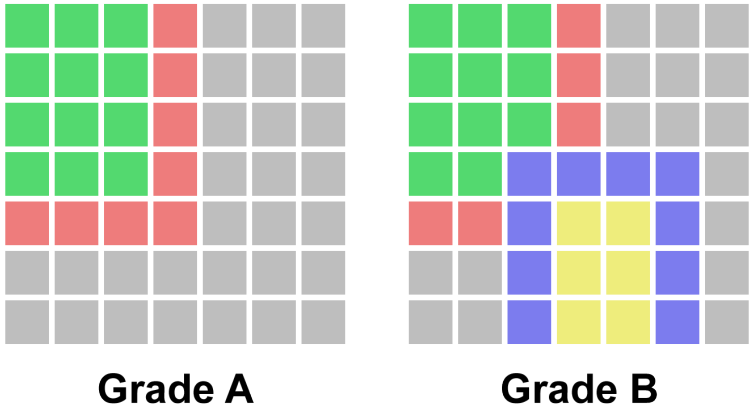
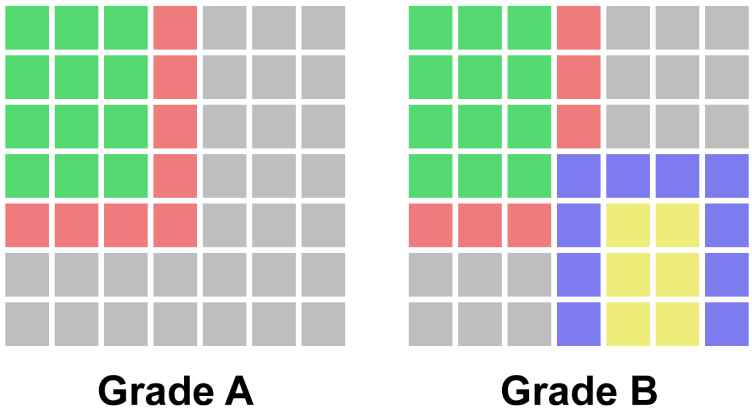


Figura 7: Na Grade A vemos a mesma primeira sala da figura 6; e na Grade B vemos uma nova sala (’Parede’ na cor azul e ’Chão’ na cor amarelo) que pode ser posicionada pois nenhum dos seus Tiles toca um Tile ’Chão’ de nenhuma sala que foi posicionada antes dela

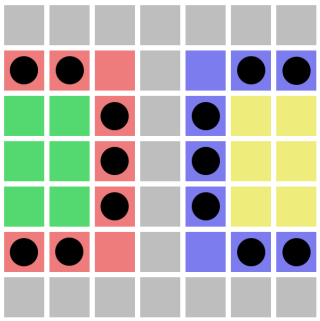


Depois de um número n de salas posicionadas (sendo n um parâmetro), é executado um algoritmo criador de labirintos em todos os Tiles que ainda são nulos. Esse labirinto também será formado por Tiles do tipo ’Chão’ e ’Parede’. Todas as diferentes salas e labirintos são separados por cor, pois o próximo passo do algoritmo vai precisar dessa distinção.

Nesse ponto já existe uma grande caverna cheia de salas e corredores, porem tudo está desconexo e o próximo passo é justamente conectar tudo. Para isso ele verifica

todos os Tile da grade do tipo 'Parede', se os vizinhos adjacentes ao Tile selecionado têm cores diferentes. Se sim, quer dizer que eles são uma possível conexão e são marcados dessa forma. Para facilitar vamos chamar esses Tiles de 'Conexão'. A figura 8 exemplifica Tiles de Conexão.

Figura 8: Duas salas de cores diferentes (a da esquerda: 'Chão' na cor verde e 'Parede' na cor vermelha e a da direita 'Chão' na cor amarelo e 'Parede' na cor azul) e todos os Tiles 'Conexão' marcados com um circulo preto.



Depois de verificar todas os Tiles do tipo 'Parede', o algoritmo seleciona uma sala aleatoriamente e a pinta de uma cor específica e que ainda não foi usada, pois ela já faz parte da caverna final. Então ele pega todos os Tiles do tipo 'Conexão' que cercam essa sala e escolhe aleatoriamente um deles, transforma-o em 'Chão', e pinta a outra sala ou labirinto ao qual esse Tile estava conectado, juntando assim essa parte à caverna principal. Por fim ele remove as conexões que faziam parte da primeira sala sobrando apenas as conexões da nova sala que foi incorporada a caverna principal. De novo, ele escolhe outro Tile 'Conexão' e repete o procedimento até que não sobre mais Tiles do tipo 'Conexão'.

Antes de continuar, o algoritmo verifica se existem salas que não foram conectadas à caverna principal e as descarta. Agora a caverna está praticamente pronta, todas as salas estão conectadas e o jogador pode visitar todas elas partindo de uma sala qualquer. O próximo passo é opcional e consiste em remover os becos sem saída para retirar os labirintos e deixar apenas os corredores que ligam duas salas. Para fazer isso um algoritmo verifica se um Tile do tipo 'Chão' possui três vizinhos do tipo 'Parede' e apenas um do tipo 'Chão'. Caso isso aconteça, esse Tile é um beco sem saída e por isso é transformado em Tile 'Parede'. Depois, o vizinho que era 'Chão' é verificado pois esse pode ter se tornado um beco sem saída.

3.4.4 Algoritmos Controlados

Algumas empresas de jogos acham os métodos anteriores muito aleatórios e difíceis de controlar e, por isso, preferem uma abordagem mais controlada como nos jogos da série e *Diablo* e no jogo *Path of Exile* (GAMES, 2013).

Nesses jogos os artistas e *designers* fazem uma coletânea de salas dos mais variados tipos, depois um algoritmo se encarrega de juntar as portas das salas de forma coerente e de forma a obedecer as regras de dificuldade do jogo em questão. Nesses casos os algoritmos realmente ficam menos randômicos, umas vez que as salas são planejadas previamente, entretanto o esforço para se fazer cada sala de forma com que o algoritmo consiga encaixá-las é bem maior do que nos outros algoritmos, pois neles os *assets* do jogo (como porta, parede, mesas cadeiras, baús, etc.) são feitos de forma modular.

3.4.5 Generative Grammar

O algoritmo GG foi desenvolvido inicialmente para escrever frases baseado em um conjunto de palavras seguidas por um conjunto de regras gramaticais (CHOMSKY, 1972). Baseado nessa premissa, o autor David Adams propõe o uso desse algoritmo para a geração de cavernas em jogos digitais (ADAMS; MENDLER, 2002). Desde então vários outros autores abordaram esse tema como Nepozitek (NEPOZITEK, 2018).

A adaptação desse algoritmo pede um conjunto de peças de mapa (por exemplo corredores e salas, sempre com saídas, pois elas serão importantes) e um conjunto de regras que servem para orientar a construção da caverna. Por exemplo se uma porta de uma sala precisa ser aberta por uma chave e esta chave está em uma sala separada, a sala com a chave deve vir antes da sala da porta.

O algoritmo de Nepozitek recebe um grafo ao invés das regras, no qual suas arestas representam conexões entre duas salas e as vértices representam as salas. Ele então monta e escolhe as peças recebidas de forma que as entradas das peças se encaixem umas nas outras formando o grafo passado.

Os outros algoritmos também montam suas cavernas encaixando as entradas das peças, mas eles seguem as regras para montar a caverna e muitas vezes adicionam mais salas, desde que as mesmas não atrapalhem as regras, para aumentar o corpo

da caverna.

3.4.6 Algoritmo de The Bind of Isaac

Alguns algoritmos tentam unir um pouco dos métodos muito aleatórios e os mais controlados, como é feito no jogo *The Binding of Isaac* (MCMILLEN, 2011), o qual as salas são preparadas previamente, mas apenas a parte artística dela. O conteúdo como itens, monstros e demais desafios são preenchidos pelo algoritmo de geração procedural.

CAPÍTULO 4

METODOLOGIA

4.1 Algoritmos Usados

Para esse trabalho escolhemos quatro algoritmos, dos discutidos anteriormente, para implementar usando o motor gráfico *Unity 3D* usando estruturas de dados semelhantes as que já foram discutidas. Além disso faremos uma avaliação das cavernas geradas de forma a comparar a velocidade com que a caverna foi gerada, a coesão da caverna e a parametrização do algoritmo. Os algoritmos escolhidos foram:

- Cellular Automata
- Algoritmo do Nuclear Throne
- Rooms and Mazes
- Generative Grammar

Escolhemos esse algoritmos pois as cavernas geradas por eles são bem distintas umas das outras. Além disso, as abordagens e técnicas usadas de cada um e os problemas resolvidos e/ou gerados por eles também são bem diferentes.

4.2 Modelo de Comparação

Faremos uma comparação entre os algoritmos de geração procedural de cavernas em três quesitos: velocidade, eficiência e parametrização.

Para a comparação de velocidade faremos um levantamento entre as iterações mais relevantes em cada algoritmo e mediremos quantas vezes elas são executadas, além da comparação por tempo, executando o algoritmo diversas vezes e medindo o tempo gasto até o fim de sua execução.

A comparação de eficiência será feita comparando algumas cavernas geradas por cada algoritmo de forma com que ela siga as três primeiras partes de *level design* descritas no capítulo 3.3 (Fluidez da Fase, Rampa de Intensidade, Variedade). Não usaremos a parte de Treinamento pois ela diz respeito mais as mecânicas propostas em um jogo do que as propriedades de uma caverna propriamente dita. Daremos notas de 1 a 10 em cada um dos três quesitos para fazer a média em cada algoritmo, sendo 1 não atendeu o quesito, e 10 atendeu completamente o quesito.

Finalmente, a comparação de parametrização será feita baseado na quantidade de parâmetros que cada algoritmo leva para gerar suas cavernas e o quanto cada um desses parâmetros impacta no resultado final da caverna.

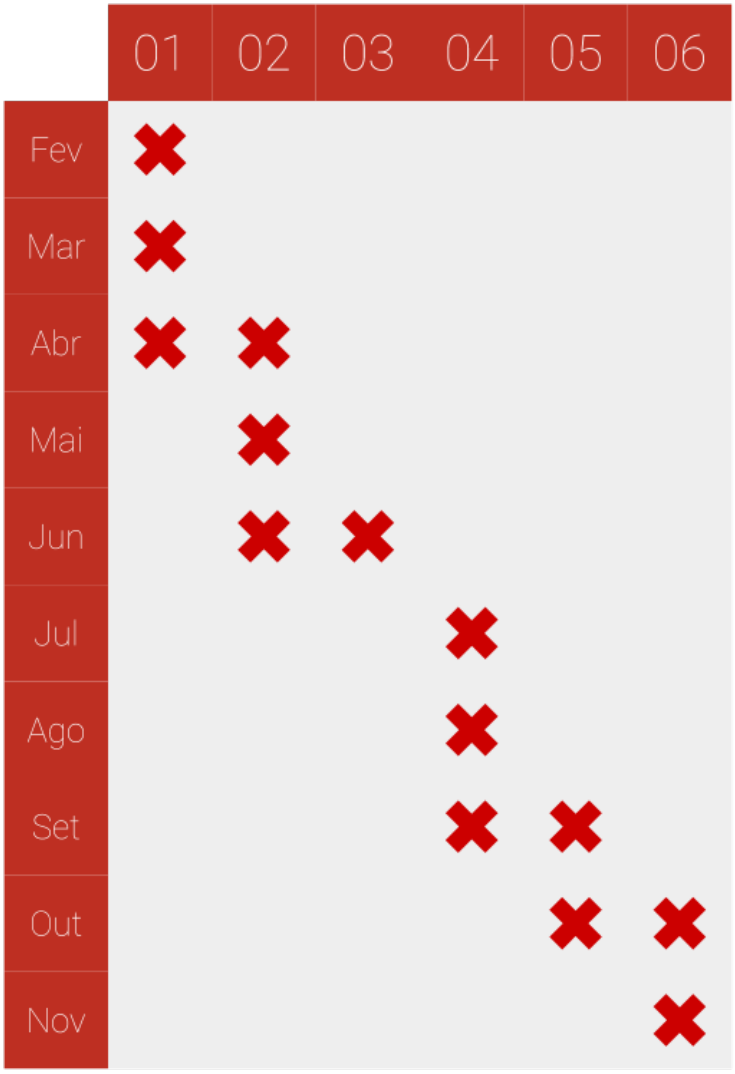
CAPÍTULO 5

PLANO DE TRABALHO

A seguir discutiremos as tarefas que temos pela frente juntamente com um cronograma esperado para a realização das mesmas:

1. **Estudo de PGC**: Nessa passo faremos o estudo de geração procedural como um todo: estudar o que é, como usar, as limitações que existem em seu uso e como tem sido usada no contexto de jogos digitais.
2. **Estudo de PGC de cavernas**: Depois do estudo de PGC como um todo focaremos o estudo em geração de cavernas.
3. **Analise e Escolha dos Algoritmos**: Baseado no que foi estudado escolheremos alguns algoritmos para que eles sejam implementados e analisados.
4. **Implementação**: Parte de implementação dos algoritmos escolhidos na ferramenta *Unity 3D*.
5. **Comparação** Com os algoritmos implementados faremos várias outras analises, mas dessa vez nos resultados produzidos pelos algoritmos.
6. **Criação da Ferramenta** Baseado nos resultados obtidos disponibilizaremos uma ferramenta capaz de escolher o melhor algoritmo, entre os que foram implementados, para a situação dada.

Figura 9: Esse cronograma mostra na horizontal o número da tarefa correspondente à lista do Plano de Trabalho e na vertical os meses os quais as tarefas serão feitas, marcas por um 'x'.



Referências Bibliográficas

ADAMS, David; MENDLER, Michael. Automatic Generation of Dungeons for Computer Games. University of Sheffield, 2002.

CHOMSKY, Noam. **Language and Mind**. [S.l.]: Cambridge, 1972.

COMPTON, Kate; MATEAS, Michael. Procedural Level Design for Platform Games. **Literature, Communication & Culture and College of Computing, Georgia Institute of Technology**, AIIDE, p. 109–111, 2006.

DORMANS, Joris; LEIJNEN, Stefan. Combinatorial and exploratory creativity in procedural content generation. Radboud Universiteit, 2013.

ENTERTAINMENT, Blizzard. **Diablo**. 1996.

ESPORTS, League of Professional. **The Video Games' Industry is Bigger Than Hollywood**. 2018. Disponível em: <https://lpesports.com/e-sports-news/the-video-games-industry-is-bigger-than-hollywood>.

GAMES, Epic. **Fortnite**. 2017.

GAMES, Grinding Gear. **Path of Exile**. 2013.

GAMES, Hello. **No Man's Sky**. 2016.

GAMES, Riot. **League of Legends**. 2009.

KUNTZ, Robert J.; GYGAX, Gary. **Dungeon**. [S.l.]: Paizo Publishing, 2004.

LAGUE, Sebastian. **[Unity] Procedural Cave Generation (E07. Ensuring Connectivity)**. 2015. Disponível em: <https://www.youtube.com/watch?v=NhMriRLb1fs&list=PLFt_AvWsXl0eZgMK_DT5_biRkWXftAOf9&index=7>.

LINDEN, Roland van der; LOPES, Ricardo; BIDARRA, Rafael. Procedural Generation of Dungeons. **IEEE Transactions on Computational Intelligence and AI in Games**, IEEE, 2013.

LUCCHESE, Fabiano; RIBEIRO, Bruno. Conceituação de Jogos Digitais. FEEC / Universidade Estadual de Campinas, 2009.

MCMILLEN, Edmund. **The Binding of Isaac**. 2011.

MICROPROSE. **Civilization**. 1991.

NEPOZITEK, Ondrej. Procedural 2d map generation for computer games. Faculty of Mathematics and Physics Charles University, 2018.

NIEMANN, Marco; PREUß, Dr. Mike. Constructive generation methods for dungeons. Westfalische Wilhelms-Universitat Munster, 2015.

NOVA, Indie. **Random Level Generation in Nuclear Throne**. 2016. Disponível em: <https://indienova.com/u/root/blogread/1766>.

NYSTROM, Bob. **Rooms and Mazes: A Procedural Dungeon Generator**. 2014. Disponível em: <https://journal.stuffwithstuff.com/2014/12/21/rooms-and-mazes/>.

SCIENCE, Wolfram. **Notes for Chapter 2: The Crucial Experiment**. 2002. Disponível em: <https://www.wolframscience.com/reference/notes/876b>.

SHAKER, Noor; TOGELIUS, Julian; NELSON, Mark J. **Procedural Content Generation in Games: A Textbook and an Overview of Current Research**. [S.l.]: Springer, 2016.

SMITH, Gillian. The Future of Procedural Content Generation in Games. AIIDE, 2014.

SPECIFICATIONS, Mojang. **Minecraft**. 2009.

STOUT, Mike. **Learning From The Masters: Level Design In The Legend Of Zelda**. 2012. Disponível em: <https://www.gamasutra.com/view/feature/134949/learning_from_the_masters_level_.php?page=1>.

TECHNOLOGIES, Unity. **Unity 3D**. 2005.

TECHOPEDIA. **Level Design**. 2019. Disponível em: <https://www.techopedia.com/definition/88/level-design>.

TOGELIUS, Julian; JUSTINUSSEN, Trandur; HARTZEN, Anders. Compositional procedural content generation. **Proceedings of the FDG Workshop on Procedural Content Generation**, 2012.

VLAMBEER. **Nuclear Throne**. 2015.