UNIVERSIDADE FEDERAL DE PERNAMBUCO
CENTRO DE INFORMÁTICA
SISTEMAS DE INFORMAÇÃO

CAROLINA MARIA DE PAIVA MELO

FERRAMENTA DE VISUALIZAÇÃO PARA EXPLORAÇÃO DE UMA TAXONOMIA DE GERAÇÃO PROCEDURAL DE CONTEÚDO PARA JOGOS

Recife
20 de maio de 2022

CAROLINA MARIA DE PAIVA MELO

FERRAMENTA DE VISUALIZAÇÃO PARA EXPLORAÇÃO DE UMA TAXONOMIA DE GERAÇÃO PROCEDURAL DE CONTEÚDO PARA JOGOS

Trabalho apresentado ao Programa de Graduação em Sistemas de Informação do Centro de Informática da Universidade Federal de Pernambuco como requisito parcial para obtenção do grau de Bacharel em Sistemas de Informação.

Orientador: Prof. Dr. Filipe Carlos de Albuquerque Calegario

Recife
20 de maio de 2022

CAROLINA MARIA DE PAIVA MELO

FERRAMENTA DE VISUALIZAÇÃO PARA EXPLORAÇÃO DE UMA TAXONOMIA DE GERAÇÃO PROCEDURAL DE CONTEÚDO PARA JOGOS

Trabalho apresentado ao Programa de Graduação em Sistemas de Informação do Centro de Informática da Universidade Federal de Pernambuco como requisito parcial para obtenção do grau de Bacharel em Sistemas de Informação.

Recife, 20 de maio de 2022

BANCA EXAMINADORA

_____

Prof. Dr. Filipe Carlos de Albuquerque Calegario (Orientador)
UNIVERSIDADE FEDERAL DE PERNAMBUCO

_____

Prof. Dr. Geber Lisboa Ramalho (2º membro da banca)
UNIVERSIDADE FEDERAL DE PERNAMBUCO

Dedico esse trabalho aos meus pais, Luiz e Niége, a minha irmã Camila e a minha companheira Giselle, que muito me apoiaram e incentivaram no encerramento desse importante ciclo da minha vida. Sem vocês eu não teria chegado até aqui, sou muito grata por tudo.

AGRADECIMENTOS

Agradeço ao Prof. Filipe Calegario, por toda sua paciência, orientação e empolgação em me incentivar e guiar na realização desse projeto. Não poderia ter escolhido orientador melhor.

Agradeço aos meus amigos pela empolgação e pela felicidade que sentem em cada conquista minha, quero sempre poder compartilhar o que me faz bem com vocês.

RESUMO

Este trabalho propõe uma ferramenta de visualização para exploração de uma taxonomia de geração procedural de conteúdo para jogos. Ele investiga as aplicações atuais existentes, assim como jogos que obtiveram proeminência pelo uso da tecnologia, em comparação com as definições formais trazidas pela academia. Adicionalmente, sumariza os principais métodos de geração procedural de jogos aplicados à geração de diversos conteúdos, e ressalta a importância dos métodos que foram mais inovadores dentro da sua área de aplicação, com os avanços que trouxeram para os estudos do tema. São expostas as dificuldades durante a construção da visualização e durante a catalogação de ferramentas, a fim de apresentar um panorama de como essa tecnologia vem sendo gerenciada pela sua comunidade de usuários. Por fim, este trabalho interliga a geração procedural de conteúdo com outras áreas, apresentando seu poder de crescimento nos anos seguintes.

Palavras-chave: Geração Procedural de Conteúdo, Jogos, Taxonomia da PCG

ABSTRACT

This article proposes a visualization tool for exploring a procedural content generation taxonomy for games. It investigates existing applications and games that have gained prominence through the use of this technology, compared to formal definitions brought by the academia. Additionally, it summarizes the main methods of procedural generation in games applied to generation of different contents, and emphasizes the importance of methods that stuck out within their area of application, with the advances they brought to the subject studies. We expose some difficulties during the construction of the visualization and during the listing process of the tools, to present an overview of how this technology has been managed by its user community. Finally, this work interconnects procedural content generation with other areas to present its growth power in the following years.

Keywords: Procedural Content Generation, Games, PCG Taxonomy

LISTA DE FIGURAS

3.1 Primeira etapa gráfica de categorização dos dados, partindo do tipo de conteúdo que se quer gerar. 25
3.2 Segunda etapa gráfica de categorização dos dados, partindo do método de geração do conteúdo. 26
3.3 Terceira etapa gráfica de categorização dos dados, partindo do algoritmo de geração do conteúdo. 27
3.4 Representação da lista de ferramentas resultantes da escolha do tipo de PCG que se quer trabalhar. 28
3.5 Estrutura hierárquica dos dados, no formato de árvore. 28
3.6 Fragmento da tabela utilizada como API de dados. 29

SUMÁRIO

1 INTRODUÇÃO **1**
1.1 MOTIVAÇÃO 1
1.2 OBJETIVO 2
1.3 O QUE É A GERAÇÃO PROCEDURAL DE CONTEÚDO PARA JOGOS? 2
1.4 POR QUE UTILIZAR A GERAÇÃO PROCEDURAL DE CONTEÚDO EM JOGOS? 2

2 ESTADO DA ARTE **4**
2.1 JOGOS QUE UTILIZAM PCG 4
2.1.1 SPORE (2008) 4
2.1.2 LEFT 4 DEAD 2 (2009) 4
2.1.3 CRYPT OF THE NECRODANCER (2015) 5
2.2 PRINCIPAIS MÉTODOS APLICADOS NA PCG 5
2.3 A ABORDAGEM BASEADA EM BUSCA 6
2.3.1 O QUE É A ABORDAGEM BASEADA EM BUSCA? 6
2.3.2 ALGORITMOS DE BUSCA EVOLUTIVA 6
2.3.3 REPRESENTAÇÃO DE CONTEÚDO 7
2.3.4 FUNÇÕES DE AVALIAÇÃO 8
2.4 MÉTODOS DE PCG PARA DUNGEONS E NÍVEIS (GERAÇÃO CONSTRUTIVA) 8
2.4.1 PARTICIONAMENTO DE ESPAÇO PARA GERAÇÃO DE DUNGEON 9
2.4.2 GERAÇÃO DE DUNGEON BASEADO EM AGENTES 10
2.4.3 AUTÔMATOS CELULARES 10
2.4.4 GERAÇÃO DE DUNGEON BASEADA EM GRAMÁTICA 11
2.4.5 MÉTODOS AVANÇADOS DE GERAÇÃO DE PLATAFORMAS APLICADOS À DUNGEONS 11
2.4.5.1 GERAÇÃO DE PLATAFORMA BASEADA EM RITMO 11
2.4.5.2 EXTENSÃO DE OCUPAÇÃO REGULADA (ORE) 11
2.5 MÉTODOS DE PCG PARA TERRENOS (FRACTAIS, RUÍDOS E AGENTES) 12
2.5.1 MAPAS DE ALTURA E DE INTENSIDADE 12
2.5.2 TERRENO RANDÔMICO 13
2.5.3 TERRENO DE FRACTAIS 13
2.5.4 CRIAÇÃO DE TERRENOS E PAISAGENS BASEADA EM AGENTES 14
2.5.4.1 A GERAÇÃO DE TERRENO DE DORAN E PARBERRY 14
2.5.5 CRIAÇÃO DE PAISAGENS BASEADA EM BUSCA 15
2.5.5.1 PROGRAMAÇÃO GENÉTICA DE TERRENO (GTP) 15
2.5.5.2 GERAÇÃO SIMPLES DE MAPA RTS 15
2.6 MÉTODOS DE PCG PARA VEGETAÇÃO E NÍVEIS 15
2.6.1 GRAMÁTICAS 16
2.6.2 L-SYSTEMS 16
2.6.3 GERAÇÃO DE MISSÕES E ESPAÇOS COM GRAMÁTICAS 16

2.7 MÉTODOS DE PCG PARA REGRAS E MECÂNICAS 17
2.7.1 CODIFICANDO REGRAS DE JOGO 17
2.7.2 JOGOS DE TABULEIRO 18
2.7.2.1 JOGOS SIMÉTRICOS / ESTILO XADREZ 18
2.7.2.2 JOGOS DE CARTAS 19
2.7.3 JOGOS 2D DE LÓGICA VISUAL 19
2.8 MÉTODOS DE PCG PARA HISTÓRIAS E MISSÕES 19
2.8.1 GERAÇÃO PROCEDURAL DE HISTÓRIA POR PLANEJAMENTO 19
2.8.2 PLANEJAMENTO COMO BUSCA ATRAVÉS DO ESPAÇO DE UM PLANO 20
2.8.3 MODELO DE DOMÍNIO 20
2.8.4 PLANEJANDO UM HISTÓRIA: ALGORITMO DE RIEDL E YOUNG 20
2.9 MÉTODOS DE PCG PARA LABIRINTOS - COM APLICAÇÃO DE ASP 21
2.10 AVALIANDO GERADORES DE CONTEÚDO 21
2.11 UMA TAXONOMIA DA PCG 22
2.11.1 ONLINE VS. OFFLINE 22
2.11.2 NECESSÁRIO VS. OPCIONAL 22
2.11.3 GENÉRICO vs. ADAPTATIVO 22
2.11.4 ESTOCÁSTICO VS. DETERMINÍSTICO 23
2.11.5 SEMENTES RANDÔMICAS VS. VETORES DE PARÂMETRO 23
2.11.6 CONSTRUTIVO VS. GERAR-E-TESTAR 23
2.11.7 GERAÇÃO AUTOMÁTICA VS. AUTORIA MISTA 23

3 SOLUÇÃO PROPOSTA 24
3.1 A REVISÃO BIBLIOGRÁFICA 24
3.1.1 A FERRAMENTA DE VISUALIZAÇÃO DE SOLUÇÕES DE PCG 24
3.1.2 OS PARÂMETROS HIERÁRQUICOS DA VISUALIZAÇÃO 25
3.1.3 ESTRUTURA DE CLASSIFICAÇÃO DOS DADOS 27
3.1.4 DIFICULDADES ENCONTRADAS NA CATALOGAÇÃO 29
3.1.5 CARÁTER EXPANSIVO DA FERRAMENTA PARA CRESCIMENTO DA MASSA DE DADOS 30

4 CONCLUSÃO 31
4.1 TRABALHOS FUTUROS 31
4.1.1 PCG E DEEP LEARNING 32
4.1.2 PCG E IA 32
4.1.2.1 I.A. SIMBÓLICA 32
4.1.2.2 I.A. CONEXIONISTA 32

1 INTRODUÇÃO

Os jogos são um tipo de conteúdo que tem estado sempre presentes na vida humana, seja em sua forma manual ou digital, permitindo a criação de conexões sociais em um contextos diferenciados, e o desenvolvimento de outras áreas da tecnologia permitiu o crescimento conjunto dos jogos, principalmente em sua forma digital.

Jogos fazem parte do nosso conteúdo cultural e permitem a integração de atividades de lazer necessárias a nós humanos [17], servindo não só como mídias de entretenimento e divertimento, como também propiciando uma forma diferente de consumo de arte. A indústria de jogos eletrônicos têm emergido como um resultado de uma inovação contínua [2], e empresas da área tem estado entre as que mais faturam do mundo inteiro, tendo essa indústria gerado um lucro de quase US$138 bilhões só no ano de 2018 [2].

Ao jogar um jogo, o jogador busca desafio e recompensa, envolvidos em atividades motivadoras [13], e isso aponta o quão cruciais e importantes são, a jogabilidade e o conteúdo apresentados, para se manter um jogador interessado na obra.

1.1 MOTIVAÇÃO

No que diz respeito à geração de conteúdo para jogos, existe uma extensa quantidade de materiais - de plugins à frameworks - que podem ser aplicados para a geração de artefatos. Também encontramos diversidade quando procuramos por ferramentas que façam a geração *procedural*, ou seja, a geração que segue uma sequência de procedimentos.

Existem algumas dificuldades em encontrar uma revisão centralizada dessas ferramentas; Muito se é visto que para as grandes empresas de jogos, na produção de jogos *AAA* - aqueles que movem grandes orçamentos -, há uma criação de softwares proprietários, ou seja, as próprias empresas criam as ferramentas necessárias para a demanda de seus projetos, e muitas vezes as tornam de uso exclusivo da empresa em questão [31]. Segundo Marklund et. al, as motivações para tal vão além da questão de "exclusividade", já que é uma característica universal nos jogos a impossibilidade de um planejamento preciso dos projetos. Há muitos requisitos de softwares que só surgem no meio dos processos de produção, e esse desenvolvimento de ferramentas proprietárias ajuda a acomodar essas novas necessidades [31].

No que diz respeito a empresas e desenvolvedores *indie*, aqueles que produzem um jogo de forma independente com aspectos técnico e financeiro limitados, é mais comum utilizar componentes de terceiros, como *engines* de jogo por exemplo, pois isso permite que o time de desenvolvimento foque nas funcionalidades principais do jogo, e é um fato determinante para a escolha da ferramente o quanto ela permite alterar as funcionalidades do produto desenvolvido [22]. Na pesquisa elaborada por Jussi Kasurinen, Jukka-Pekka Strandén e Kari Smolander, notou-se que o preço das ferramentas não foi considerado importante para as empresas, mesmo que startups [22] - mas é importante ressaltar que essa pesquisa foi realizada em 2010 e de lá para cá isso pode ter mudado um pouco por conta do crescimento de ferramentas open source na área.

De todo modo, em ambos os lados observados - o indie e o de grandes empresas -, é notável que para a escolha de uma ferramenta que não seja de construção própria, é preciso ou de uma pesquisa sobre ferramentas aplicadas as áreas que se quer gerar o conteúdo procedural, ou de se limitar à escolher as ferramentas mais utilizadas no mercado - mas que não necessariamente são as melhores para o resultado que se quer obter.

1.2 OBJETIVO

Este trabalho visa desenvolver uma ferramenta que apresente visualmente uma forma de revisão bibliográfica do material existente referente às aplicações de geração procedural de conteúdo voltadas a jogos eletrônicos. Essa revisão inclui não apenas ferramentas para se aplicar nas *engines* de jogos, como também softwares de modelagem 3D e qualquer outro software que gere conteúdo que possa ser inserido em jogos eletrônicos.

1.3 O QUE É A GERAÇÃO PROCEDURAL DE CONTEÚDO PARA JOGOS?

De acordo com Togelius et al., a geração procedural de conteúdo - abreviada PCG, do inglês *Procedural Content Generation* - em jogos se refere à criação automática de conteúdo utilizando algoritmos. As principais áreas de aplicação são na criação de dungeons, mapas, armas, vegetações e terrenos. Essa é uma definição concisa, porém não totalmente precisa, visto que a PCG pode ser visualizada de diferentes perspectivas por diferentes pessoas. Pode-se adicionar a definição que é necessária alguma entrada do usuário, seja ele um jogador ou designer, para que o conteúdo seja gerado. Dessa forma, PCG é a criação automática de conteúdo de jogo com entrada de usuário limitada ou indireta, sendo essa criação aleatória ou que se adapta a cada resultado gerado [50].

1.4 POR QUE UTILIZAR A GERAÇÃO PROCEDURAL DE CONTEÚDO EM JOGOS?

Existem várias motivações para aplicar PCG em um jogo, podendo trazer benefícios de natureza financeira, produtiva e/ou criativa. Podemos dizer que, para exemplificar, a PCG pode reduzir os custos de tempo e quantidade de mão de obra, beneficiando o financeiro. Pode também produzir mais resultado em um determinado tempo, se comparado a um trabalho manual humano, sendo mais produtivo. E pode gerar novas ideias criativas, aproveitando o material gerado pela máquina em conjunto com as ideias de humanos que acompanhem o processo, dando mais oportunidade para a geração de novos conceitos [50].

A criação e desenvolvimento de um jogo é um processo que exige custos tanto comerciais quantos humanos, e é cada vez mais necessária a adição de mais pessoas em uma ou mais equipes para gerar um produto em um limite de tempo. Fazer todo, ou grande parte, desse trabalho de forma manual e/ou repetitivo é contraprodutivo e possível gerador de custos extras, já que ocorre o desgaste humano e aumenta a probabilidade de falhas humanas. Isso acaba afetando a qualidade e quantidade de conteúdos produzidos para um jogo, já que pode diminuir

a diversidade de gêneros, narrativas e/ou materiais que possam ser encontrados em um produto desse tipo. Isso pode estagnar parte da indústria, em consequência, tanto pelo lado das grandes empresas fazerem "mais do mesmo"na suas propriedades intelectuais, quanto por dificultar que pequenos desenvolvedores consigam começar e terminar seus produtos de forma viável e rentável.

A geração procedural de conteúdo ajuda a diminuir a necessidade de interferência humana no sistema que a executa, de forma que é possível obter resultados mais rápidos e menos custosos dentro do processo de desenvolvimento de um jogo. Isso possibilita a criação de um produto novo e/ou diferente em relação aos já existentes no mercado - já que a geração pode trazer novas ideias criativas -, e pode também até trazer um aumento de qualidade do produto, a depender do gênero e de qual conteúdo ela se aplica.

Podemos dizer que a PCG possibilita a extensão da criatividade humana, ao tornar o conteúdo de um jogo mais diverso e rico e possível até de se moldar aos gostos ou forma de jogar de seus jogadores. Dependendo do ponto de vista que buscamos olhar para a PCG, ela pode não só acrescentar mais experiências de entretenimento, como também pode vir a auxiliar no aprendizado dos chamados jogos sérios, que são jogos com propósito educacional e intelectual explícito [24].

O uso de PCG aumenta o *replay value* [40] - o quanto "re-jogar"esse jogo acrescenta na experiência de um usuário - ao gerar novos conteúdos para o jogador dentro do mundo de um sistema de jogo. Isso impacta positivamente no fator novidade para o usuário que esteja jogando, e acreditamos que pode gerar um sentimento maior de curiosidade pelas possíveis novas variações que o sistema pode criar em momentos futuros.

2 ESTADO DA ARTE

2.1 JOGOS QUE UTILIZAM PCG

Hoje em dia, muitos jogos utilizam geração procedural de conteúdo para agregar mais a experiência do jogador e para promover uma variedade dentro do conteúdos proporcionados pelo jogo. Segundo o apresentado por Julian Togelius, Noor Shaker e Mark J. Nelson, a PCG tem recebido uma atenção crescente em jogos comerciais e, por isso, têm-se investido nos seus estudos, tanto comerciais quanto acadêmicos.

Listamos abaixo alguns jogos que, além de terem sido bastante populares na época de seu lançamento - e alguns manterem esse posto até hoje -, trouxeram uma grande visibilidade à PCG, justamente por inovarem na forma que conteúdo era gerado em jogos em suas respectivas épocas.

2.1.1 SPORE (2008)

Spore é um jogo simulador de vida e de estratégia em tempo real, desenvolvido pela Maxis e publicado pela Electronic Arts em 2008. Seu designer principal foi Will Wright, e ele descreve que Spore permite que o jogador controle o desenvolvimento de uma espécie, desde o seu nível celular até seu desenvolvimento como sociedade inteligente.

Spore aplicou a geração procedural em diferentes etapas do seu desenvolvimento, mas uma das quais chamou bastante atenção foi a forma como ele tentou desenvolver um gerador de criaturas com animação procedural [49]. Apesar de ser algo que ficou limitado à parte de edição das criaturas, o qual recebia entrada do usuário direta ao invés de proporcionar uma criação automática, o jogo foi visionário com o tipo de técnica utilizada para a época que foi lançado. Nele, é possível ganhar pontos de evolução que você pode gastar evoluindo características da sua espécie, como dando olhos extras, pernas adicionais, mudando seus hábitos alimentares, etc.

De acordo com um artigo de autoria de DeBry et. al [10], publicado na SIGGRAPH07 - Conferência do Grupo de Interesse Especial em Computação Gráfica e Técnicas Interativas, que ocorreu entre 5 e 9 de agosto de 2007, a equipe de desenvolvimento de Spore fez um grande uso de textura procedural guiada por usuário. O objetivo foi de amplificar a criatividade do usuário, ao permitir seu controle sobre o sistema de textura - tentando balancear uma interface detalhada para customização ao mesmo tempo que limitava o usuário a composições simples e texturas elaboradas previamente.

Ao montar sua criatura, o jogo gera de forma procedural um conjunto de animações, desde movimentos de dança até os de luta, e gera uma textura para incrementar na aparência original da criatura [10].

2.1.2 LEFT 4 DEAD 2 (2009)

Left 4 Dead 2 é um sucessor de um jogo de nome homônimo, Left 4 Dead, lançado em 2009 para Windows e Xbox 360. É um jogo multijogador, cooperativo, de *survival horror* e em

primeira pessoa, publicado pela empresa de jogos *Valve*. Nele, o jogador lutra contra uma horda de zumbis e o objetivo é sobreviver e cooperar com seus outros companheiros duranta as partidas.

A PCG é utilizada na inteligência artificial do jogo para promover um *replay value*, tanto na variação do comportamento de inimigos, como também nas variações de sua aparência, de forma que os inimigos não se tornem muito repetitivos [39].

Michael Booth, um programador e game designer que trabalhou na Valve durante o desenvolvimento do jogo, fez uma apresentação na conferência AIIDE-09 de Stanford de 2009, uma conferência anual sobre IA em jogos e em entretenimento interativo, na qual discorreu sobre os sistemas de inteligência artificial de Left 4 Dead. O nome da IA utilizada no jogo foi a "The Director"e nela os personagens "infectados"foram projetados para serem visíveis, mas não memoráveis, já que eles apareciam em grande quantidade pra desafiar os jogadores. Um problema encontrada pelos desenvolvedores foi a falta de diversidade em estágios iniciais, por conta do reuso de modelos, mas foram desenvolvidos outros sistemas de variação a partir disso [5].

O jogo tornou-se bastante popular em sua época de lançamento por conta da "inteligência"apresentada com a progressão do jogo, o que permitiu que o jogo aprendesse a forma de jogar do jogador e inteirasse sobre ela de forma a tornar o mesmo mais divertido e desafiador [19].

2.1.3 CRYPT OF THE NECRODANCER (2015)

Crypt of the NecroDancer é um jogo rítmico no estilo *roguelike* - um subgênero de RPG de fantasia caracterizado pela geração aleatória ou procedural de níveis -, lançado em 2015 pela desenvolvedora *indie Brace Yourself Games*, localizada em Vancouver. Neste jogo, o jogador controla uma seleção de personagens para explorar vários níveis de um dungeon underground. Segundo sua descrição, seu objetivo é andar no ritmo e acompanhar a batida da música para acertar os inimigos [6].

Em CotN - abreviação para o título do jogo-, são criadas zonas com salas de diferentes tamanhos, e é possível que o jogador altere a área ao "cavar"as paredes, podendo então adicionar complexidade e alterar a estrutura de algumas salas [8]. Outra estratégica de importância na geração de nível do jogo, segundo o apresentado no artigo de Castro, Mota e Fantini, é a definição de um caminho *crítico*, ou seja, o caminho mais curto que um jogador pode fazer para finalizar todas as atividades necessárias e encontrar a saída.

CotN dá ao jogador certas dicas de qual caminho seguir, seja por estímulos sonoros ou visuais, e itens podem melhorar a visibilidade do mapa pelo jogador [8]. O jogo também apresenta inimigos não determinísticos, ou seja, esses inimigos nem sempre respondem da mesma forma à situações semelhantes, promovendo uma variedade na experiência de jogabilidade [8].

2.2 PRINCIPAIS MÉTODOS APLICADOS NA PCG

Os tópicos a seguir serão apresentados de acordo com as principais áreas de aplicação da PCG, com exceção da primeira seção, que introduz a abordagem baseada em busca. O objetivo é demonstrar os tipos de algoritmos e métodos que melhor se aplicam em cada conjunto de

elementos que se quer gerar conteúdo de forma procedural. Os métodos e algoritmos aplicados à geração de terreno, por exemplo, podem ser diferentes dos métodos aplicados à geração de vegetação, tanto pela natureza do material que se reproduzir - em relação à quantidade de detalhes e variação entre eles - como também em relação ao que encontramos mais comumente implementado nos softwares existentes hoje no mercado.

Alguns termos utilizados para as descrições de métodos e algoritmos são específicos da taxonomia da PCG, e discorremos mais sobre eles na última seção deste capítulo. Então, em caso de dúvida quanto a definição destes termos, é possível consultá-los na seção 2.11 deste capítulo.

2.3 A ABORDAGEM BASEADA EM BUSCA

2.3.1 O QUE É A ABORDAGEM BASEADA EM BUSCA?

A geração procedural de conteúdo com abordagem baseada em busca é o uso de computação evolutiva - e métodos similares - para gerar conteúdo de jogo, de forma que essa geração capte conteúdo que atenda características específicas pré-definidas [53]. Computação evolutiva é uma técnica de inteligência computacional inspirada na evolução natural, e que pode ser aplicada à jogos na geração de algum conteúdo, como por exemplo, inimigos interessantes e divertidos de se jogar contra dentro de um jogo [29]. Segundo Togelius, em 2010 foi observado que essa abordagem ganhou proeminência nas pesquisas acadêmicas próximas desse ano.

São apresentados abaixo os principais conceitos desse tipo de abordagem para resolver um problema de geração de conteúdo, retirados do artigo de 2010 de Julian Togelius e Noor Shaker:

- *Um algoritmo de busca* é o mecanismo principal de um método de busca. Seu grau de complexidade pode levar em conta certas condições e ser especializado para uma *representação de conteúdo* em específico.
- *Uma representação de conteúdo* é a forma de representar artefatos a serem gerados, definindo e limitando o conteúdo que se quer gerar e determinando se a busca pode ser feita de forma efetiva, ou seja, com resultados relevantes e funcionais, como por exemplo as representações de níveis e missões.
- *(Uma ou mais) funções de avaliação* são funções de uma parte individual de conteúdo que indicam um número que representa a qualidade desse conteúdo. Um resultado de uma função de avaliação pode indicar a jogabilidade ou apelo visual de um nível, por exemplo. A atividade de se criar uma função de avaliação é considerada uma das mais difíceis dentro do desenvolvimento de um método de PCG baseado em busca.

Os tópicos seguintes detalham um pouco mais sobre cada um desses pontos para que seja mais perceptível a diferença entre eles e a importância de suas funções individuais.

2.3.2 ALGORITMOS DE BUSCA EVOLUTIVA

Um algoritmo de busca evolutiva é um algoritmo de busca estocástico vagamente inspirando na teoria da seleção natural proposta por Charles Darwin [53]. A ideia é manter uma população

de indivíduos em que cada geração é avaliada e somente os que mais se encaixem no padrão possam se reproduzir, removendo da população os que menos se encaixaram [53]. Dessa forma, gerando mutações que permeiam somente as qualidades mais desejadas ou que tragam mais benefícios para a população em questão.

Segundo um artigo publicado em 2020 por Rafael Guerra de Pontes e Herman Martins Gomes, um problema observado nesse tipo de abordagem de busca é que não há garantia que uma solução realmente válida seja encontrada. Portanto, talvez seja necessário adicionar mecanismos que garantam que restrições mínimas sejam mantidas após todas as interações, ou seja, que não se percam certas características com o "evoluir das gerações"[9]. Por um outro lado, algoritmos desse tipo são atrativos por serem possíveis de se trabalhar com problemas que sejam multiobjetivos, não contínuos e até problemas NP-completos - aqueles de tempo polinomial não-determinístico [9].

Segundo o artigo de K.F Man, K. S. Tang e S.Kwong, que aborda os conceitos e aplicações de algoritmos genéticos - que também se aplica a algoritmos evolutivos visto que algoritmo genético é uma classe de algoritmo evolutivo -, há mais algumas vantagens encontradas em sua utilização, como:

- Podem lidar com restrições de problemas simplesmente incorporando essas restrições na codificação dos "genes dos cromossomos", que no caso são os parâmetros que representam a solução de um problema [30].
- Utilizam de uma técnica fácil de se entender com pouca (ou mesmo nenhuma) matemática [30].
- Podem ser conectados à simulações e modelos existentes [30].

2.3.3 REPRESENTAÇÃO DE CONTEÚDO

Representação de conteúdo é um importante ponto no que diz respeito à conteúdo de jogo, já que uma representação impacta na eficiência de um algoritmo de geração e no quanto de espaço do conteúdo o método terá de cobrir. Em algoritmos evolutivos, as soluções no espaço de geração são codificadas como *genótipos*, utilizados para busca e validação eficientes, e depois são convertidos para *fenótipos*, que são as próprias entidades que estão sendo evoluídas [53]. Dentro do contexto de PCG, Togelius et. al exemplificam que o genótipo é análogo as instruções para se criar um nível de um jogo, enquanto que o fenótipo é equiparável ao próprio nível em questão desse mesmo jogo.

Togelius et. al utilizam um exemplo concreto de diferentes representações de conteúdo encontrada em um nível de *Super Mario Bros.* que acreditamos ser bem pertinente e sucintas para exemplificar representação de conteúdo em vários aspectos, com todos os pontos mencionados abaixo:

- Diretamente, como um mapa do nível, no qual cada variável no genótipo corresponde a um "bloco"no fenótipo, como no caso de tijolos e blocos com marcas de interrogações, por exemplo.

- Indiretamente, como a lista de posições e propriedades de diferentes entidades de jogos como inimigos, plataformas, buracos e montanhas, por exemplo.
- Indiretamente, como um repositório de padrões diferentes e reusáveis (como coleções de moedas, por exemplo) e uma lista de como eles estão distribuídos no mapa (com posições, rotações e tamanhos).
- Indiretamente, com uma lista de propriedades desejáveis como números de buracos, inimigos e moedas, por exemplo.
- Indiretamente, como um semente com número randômico.

Um problema encontrado nessa relação direta e indireta, exposto por Togelius et. al, é o alcance expressivo que pode ser ter com a representação escolhida: podemos, por exemplo, medir um gerador de níveis de um jogo de plataforma de acordo com o número diferente de configurações de blocos que ele pode produzir, mas seria mais relevante medir a qualidade dessas configurações, já que seria isso que de fato impactaria na experiência de um humano jogar o jogo.

2.3.4 FUNÇÕES DE AVALIAÇÃO

Funções de avaliação são, assim como o nome indica, funções que avaliam as soluções que são candidatas, codificadas como representação, que atribuem uma pontuação para cada uma das soluções apresentadas [53]. Essas funções precisam ser efetivas em sua avaliação, já que podem impactar em todo o processo de forma negativa caso não o sejam - provendo muitos resultados que não sejam o desejado por conta de avaliações que levam em conta características não relevantes, por exemplo.

Em suma, na PCG baseada em busca, são utilizados para criação conteúdo de jogo a computação evolutiva ou outros algoritmos estocásticos que seja de busca ou otimização. A criação de conteúdo é um processo de busca por um conteúdo que tenha o melhor resultado nas funções de avaliação, e os maiores problemas desse tipo de solução são justamente a função de validação a ser utilizada e o tipo de representação. Segundo Togelius et al., PCG baseada em busca é muito popular no meio acadêmico e há vários estudos publicados da área - ao menos na época de lançamento do artigo de 2010.

2.4 MÉTODOS DE PCG PARA DUNGEONS E NÍVEIS (GERAÇÃO CONSTRUTIVA)

Dungeon, do francês arcaico *donjon* - segundo definição do dicionário de Cambridge -, significa "grande torre de um castelo"e no inglês é muitas vezes utilizado com o sentido de masmorra/calabouço. Em jogos de computadores, uma dungeon muitas vezes representa um labirinto onde o jogador entra em um ponto com o objetivo de alcançar a saída e no seu percurso se depara com tesouros, monstros e desafios [44]. Dungeons são muito utilizadas em RPGs (jogos de interpretação de papéis, do inglês *Role-playing Games*), e, de acordo com Shaker et. al, têm sido proeminente o uso de PCG para a geração de conteúdo em tempo de execução dentro desse gênero.

Partimos da definição que níveis de dungeons de aventura e de RPG são ambientes labirínticos, que consistem, em sua maioria, de desafios que tem relações uns com os outros, recompensas e quebra-cabeças: esses elementos são bem colocados no tempo e espaço para oferecer progressões de jogo altamente estruturadas [26].

Dentro de dungeons os níveis são de livre exploração e apresentam uma progressão nos desafios, recompensas e quebra-cabeças que são encontrados durante o percurso. Por conta disto, o design de uma dungeon é de natureza complexa, e exige a construção de um complexo ambiente de jogo que apresenta uma jogabilidade predeterminada [44].

Segundo Shaker et. al, a aplicação de PCG em uma dungeon trabalha com a geração de topologia, geometria e de objetos ligados à jogabilidade de níveis específicos, e por isso um método comum de geração de dungeon segue esses três elementos:

1. Um modelo de representação, que é uma representação simplificada de uma dungeon com uma visão geral simples de sua estrutura final.

2. Um método para construir esse modelo de representação.

3. Um método para criar a geometria real de uma dungeon a partir do seu modelo de representação.

Os métodos apresentados por Shaker et. al têm os seus contrastes mas são similares em um ponto: todos são construtivos, ou seja, produzem uma instância de resultado por vez, em contraste aos métodos baseados em busca. Além disso, são rápidos e permitem um controle limitado de seus resultados e propriedades - o grau de controle é uma característica muito importante de qualquer método procedural.

2.4.1 PARTICIONAMENTO DE ESPAÇO PARA GERAÇÃO DE DUNGEON

Um algoritmo de particionamento de espaço, como indica o nome, produz a partição de um espaço, que em outras palavras é uma subdivisão de um espaço 2D ou 3D em subconjuntos distintos, de modo que qualquer ponto no espaço esteja exatamente em um desses subconjuntos, também são chamados de células [44]. Segundo Shaker et. al, esses algoritmos comumente operam de forma hierárquica: cada célula numa partição de espaço é posteriormente subdividida aplicando-se o mesmo algoritmo recursivamente, permitindo que sejam organizadas em uma *árvore de particionamento de espaço*. Esse tipo de árvore permite rápidas consultas geométricas e é importante para a área de computação gráfica, permitindo, ainda segundo Shaker et. al, *raycasting* - algoritmo utilizado em tratamento de imagem - de forma eficiente, *frustum culling* - visibilidade de um objeto em um mundo modelado - e detecção de colisão, por exemplo.

O método mais popular para o particionamento de espaço é o BSP - particionamento de espaço binário, traduzido do inglês *binary space partitioning* -, que divide de forma recursiva o espaço em dois subconjuntos, e permite representar o espaço como uma árvore binária, também chamada de *árvore BSP* [44].

Quando algoritmos de particionamento de espaço são usados em gráficos 2D ou 3D, seu objetivo geralmente é o de representar elementos existentes como polígonos ou pixeis, ao invés

de criar novos. Porém, o princípio de que as áreas resultantes desse particionamento não se sobrepõem é mais aplicado quando se quer criar salas em uma dungeon ou áreas distintas em um nível de jogo, no geral [44].

Gerar uma dungeon por BSP segue uma abordagem macro, no qual o algoritmo tem total visibilidade da dungeon, e esse algoritmo garante que não hajam duas salas se sobrepondo, permitindo uma aparência bastante estruturada de uma dungeon [44]. Em sua execução, o algoritmo particiona o espaço em objetos "folha", cada um sendo armazenado em uma estrutura de dados de árvore, e cada folha é divido de forma que haja uma sem filhos, sendo esta definida então como a *sala* [3].

2.4.2 GERAÇÃO DE DUNGEON BASEADO EM AGENTES

Geração de dungeon baseada em agentes utiliza um ou mais agente para realizar atividades que vão resultar em algum tipo de nível. Esse tipo de abordagem segue uma abordagem micro - ao contrário do macro descrito no método de particionamento - e tem mais chances de criar um dungeon orgânico e caótico - em oposição à um bem mais organizado por particionamento[44]. A "cara"da dungeon vai depender em grande parte do comportamento do agente: quanto mais estocástico, mais caótico. Porém, segundo Shaker et. al, não há garantia que esse método não vá sobrepor as salas de uma dungeon, ou uma dungeon que só carregue parte de uma área de si mesmo ao invés de sua integralidade.

2.4.3 AUTÔMATOS CELULARES

Autômatos celulares consistem de um conjunto de células, com um número finito de estados, que representam sistemas dinâmicos discretos no espaço e no tempo, sendo bastante versáteis e vários tipos deles se mostraram Turing-completo, ou seja, demonstraram que conseguem manipular qualquer máquina de Turing. São modelos computacionais discretos que são utilizados para estudar teoria da computação, matemática e sistemas biológicos, e também são utilizados na bioinformática [43].

Um autômato celular consiste de um *grid* de n-dimensões com conjuntos de estados e conjuntos e regras de transição. Segundo Shaker et. al, a maioria de autômatos celulares são ou uni-dimensionais (como vetores) ou bi-dimensionais (como matrizes). A distribuição das células de estados no começo de um experimento é o estado inicial do autômato, e a partir daí ele se desenvolve em passos discretos baseados nas regras desse próprio autômato em particular. Em cada tempo *t* cada célula decide seu novo estado baseada em seu estado atual e todas as suas células vizinhas em um tempo de *t-1*.

Em um artigo de Lawrence Johnson, Georgios Yannakakis e Julian Togelius, datado de 2010, é apresentando um algoritmo baseado em autômatos celulares que é capaz de gerar mapas jogáveis e bem desenhados baseados em túneis. O algoritmo apresentado tem um custo computacional baixo, permitindo uma geração em tempo real, e a representação de mapa proposta oferece flexibilidade suficiente respeitando o design do nível em questão [20].

A avaliação do algoritmo em questão mostrou a eficiência do mesmo e o acesso que ele permite à um game designer de ajustar os parâmetros do conteúdo e consequentemente as variações de seus resultados [20]. Utilizaram *grids* 2D em contraponto aos 3D por motivo de

ser mais fácil de se manter o nível do terreno nesse formato, e consequentemente permitir um maior acompanhamento dos testes de conectividade no mapa.

2.4.4 GERAÇÃO DE DUNGEON BASEADA EM GRAMÁTICA

Originalmente as gramáticas generativas foram desenvolvidas para descrever formalmente estruturas em linguagem natural, e esse processo é descrito com mais detalhes no tópico de métodos aplicados à geração procedural de vegetação e níveis. Porém, pode ser aplicada a gramática em diferentes áreas, e a geração de dungeon é uma delas.

No trabalho de conclusão de bacharelado de David Adam, ele e Michael Mendler utilizam gramáticas de grafos para gerar níveis em um jogo de FPS (tiro em primeiro pessoa) [34]. Eles descrevem que grafos de gramáticas são o tipo de gramática que manipula grafos ao invés de *strings* de texto, e demonstram duas abordagens principais, que devido a sua complexidade não vão ser apresentadas em detalhes nesse trabalho, mas devem ser mencionadas. Em uma dessas abordagens, o usuário tem menos controle em como os nós dos grafos são criados ao aplicar regras [34].

Uma limitação encontrada no trabalho de Adam é de que as regras da sua gramáticas são *hard-coded*, ou seja, escritas diretamente no código, não provendo uma solução escalável para o caso de alterar essas regras [34]. Porém foi um trabalho que demonstrou a importância da aplicação de gramática na geração de dungeon e abriu portas para outros trabalhos importantes na área, que não serão especificado aqui por sua complexidade - mas é importante saber da sua existência. Como é o caso do tralho de Joris Dormans [12] e de Roland van der Linden, Ricardo Lopes e Rafael Bidarra [25] - que inclusive foi inspirado pelo de Dormans, segundo menção de Shaker et. al.

2.4.5 MÉTODOS AVANÇADOS DE GERAÇÃO DE PLATAFORMAS APLICADOS À DUNGEONS

De acordo com Shaker et. al, existem dois métodos que foram originalmente propostos para a geração procedural de níveis de plataforma, mas que foi percebido que seus conceitos centrais também poderiam ser utilizados para melhorar a geração de dungeons [44].

2.4.5.1 GERAÇÃO DE PLATAFORMA BASEADA EM RITMO

No trabalho de Smith et. al [48], é proposta a geração de nível baseada na noção de ritmo relacionada ao *timing* e repetição das ações do usuário. A divisão de um nível em grupos de ritmo pode se conectar com a divisão de uma dungeon em grupos de dungeons que tenham característica de gameplay distintas, como a sua velocidade de evolução pelos níveis e mapas, por exemplo.

2.4.5.2 EXTENSÃO DE OCUPAÇÃO REGULADA (ORE)

No trabalho de Peter Mawhorter e Michael Mateas [32] é apresentada o algoritmo ORE - Extensão de Ocupação Regulamentada, do inglês *Occupancy-Regulated Extension*. ORE é um algoritmo de "montagem"de geometria generalista que dá suporte em escalas arbitrárias à au-

toria de níveis baseados em designs feitos por humanos. Ele atua diretamente na geração de níveis de plataforma 2D e se baseia em pedaços de nível "pré-feitos", de modo que ele possa adquirir esses pedaços de uma biblioteca e os utilizar para montar um nível. Como esse método é generalista para jogos de plataforma 2D, precisam ser preenchidos alguns elementos e mecânicas de jogo específicos, e esses "pedaços"precisam ser projetados e adicionados a um biblioteca [32].

2.5 MÉTODOS DE PCG PARA TERRENOS (FRACTAIS, RUÍDOS E AGENTES)

Terrenos estão presentes em todos os aspectos de jogos eletrônicos, já que tanto em jogos 2D ou 3D vão existir um terreno ou paisagem na qual o jogador possa ter diferentes níveis de interação, seja para percorrer o mapa ou seja para interagir com alguma mecânica que afete o jogo. Nesse contexto, reutilizar material pode afetar a credibilidade e originalidade de um terreno digital - padrões repetitivos podem tirar a noção de continuidade e deixar a atmosfera do jogo mais artificial. Em um jogo como *Grand Theft Auto* (a partir do jogo III, lançado em 2001), por exemplo, uma paisagem precisa ter apelo visual mas também ser crível, além de acompanhar estradas que cortem seu caminho - o que demonstra a importância do controle dos algoritmos utilizados nessa geração, para no final ter algo que funcione seguindo certas restrições.

Outro tipo comum de conteúdo de jogo são os ruídos, que são bem úteis para gerar variações mais detalhadas em uma superfície. Ruídos podem ser encontrados nas nuvens e composições de clima de um skybox - que é um tipo de plano de fundo tridimensional para dar sensação de profundidade maior em jogos, em partículas de poeira ou água de um solo e em diversos elementos que possam trabalhar com elementos naturais próximos do mundo real, inclusive na variação de topologia de um solo [46].

2.5.1 MAPAS DE ALTURA E DE INTENSIDADE

Ruídos e propriedades de um terreno podem ser representados como uma matriz bidimensional de números reais, sendo a largura e altura da matriz as dimensões x e y, respectivamente, de uma superfície retangular. Tratando-se de ruídos, chamamos essa representação de *mapa de intensidade*, com os valores das células representando o brilho de cada pixel associado. Já no caso de terrenos, o valor de cada célula representa a altura do terreno naquele ponto em específico, e chamamos essa representação de *mapa de altura* [46]. De forma geral, essas representação nos permite usar os mesmos métodos para geração de ruídos e de terrenos.

É importante observar que no caso de terrenos podemos precisar - de forma opcional ou necessária - de representações um pouco diferentes, como no caso de *Minecraft*, que utiliza grandes redes de voxels - que são cubos interligados que permitem estruturas que não são possíveis de serem representadas em mapas de altura, mas que em contrapartida ocupa um espaço de armazenamento muito maior.

2.5.2 TERRENO RANDÔMICO

Se codificarmos um terreno como um mapa de altura, representado por um array bidimensional, conseguimos gerar um terreno randomicamente ao chamar um gerador de número aleatório para preencher todas as células desse array? Shaker, Togelius e Nelson respondem a essa pergunta resumidamente com: não [46]. Segundo eles, enquanto essa técnica pode "funcionar", o resultado pode ser nada funcional. Um mapa de altura gerado dessa forma pode ser totalmente o contrário do que se espera de um terreno, e apresentar mudanças bem heterogêneas em sua continuidade.

O problema em preencher o mapa de altura com valores aleatórios está no fato de cada número ser gerado de forma independente, não levando em conta os outros valores. Diferentemente do que é apresentado em um terreno, já que o valor seguinte pode sofrer consequência do anterior. Isso pode ficar bem visível como exemplo quando imaginamos uma montanha que tenha uma descida "sutil". A curvatura da descida dessa montanha acompanha o valor anterior dela, o de "subida", de forma que a descida obedeça o padrão de um terreno normal e os fatores geográficos da área.

Existem várias formas de gerar mapas de altura aleatórios que não caiam nesse problema, e os métodos apresentados abaixo foram inicialmente projetados para texturas em computação gráfica, que sofriam do mesmo problema [46]. Abaixo descrevemos cada um, com breves descrições sobre seu funcionamento.

- Terreno randômico interpolado Nesse método, procuramos gerar terrenos "macios", ou seja, que as mudanças de altura seja interpoladas, interligadas, de forma "natural"[46]. Utilizamos aqui *ruído interpolado*, que gera alguns poucos valores randômicos e os interpola. Existem alguns métodos para interpolar esses pontos, mas os citados por Shaker são a interpolação bilinear e a bi-cúbica. A diferença entre elas é basicamente matemática, e não as exploraremos em detalhe, mas é importante saber de sua utilização.

- Terreno randômico baseado em gradiente (Perlin noise) Esse tipo de método para terreno gera as elevações/desníveis diretamente, inferindo os valores de altura, ao invés de gerar essas valores e interpolá-los. Esse tipo de inicialização randômica de array é chamada de *ruído gradiente*, e foi, curiosamente, aplicado pela primeira vez no filme *Tron* de 1982 [46]. Por ter sido aplicado no filme pelo cientista da computação Ken Perlin, esse algoritmo é também conhecido como *ruído de Perlin*, ou em inglês *Perlin noise*.

2.5.3 TERRENO DE FRACTAIS

Segundo o apresentando por Shaker, apesar do ruído gradiente parecer ser mais orgânico para um terreno, ele têm ondulações de frequência constante - diferente de um terreno real, no qual há variações em escalas múltiplas. Porém, em escalas menores, se observado mais de perto, os terrenos podem apresentar pequenos padrões de variações que são similares ao quando observado por uma escala maior [46].

Shaker defende que essa similaridade pode se comparar a base de fractais, e por esse motivo terrenos gerados com essa propriedade são chamados de terrenos de fractais. Eles podem ser

reproduzidos por vários métodos matemáticos, sendo o mais fácil pegar qualquer um dos métodos de terreno randômico, apresentados anteriormente, aplicados a uma única escala e rodá-los várias vezes em escalas múltiplas.

Um tipo de método para gerar terreno fractal que é muito utilizado em jogos por ser simples de implementar e eficiente computacionalmente é o algoritmo *diamond-square* [46]. Nele, nos definimos 4 vértices de um mapa de altura com valores de sementes, e ele funciona de uma forma quase que recursiva, pegando o valor médio desses vértices e os adicionando de outro valor aleatório. É como se ele fosse ao encontro de um valor médio para iterar, até encontrar um valor máximo de iterações, gerando um terreno com a aproximação do valor resultante [46].

2.5.4 CRIAÇÃO DE TERRENOS E PAISAGENS BASEADA EM AGENTES

A maior vantagem em se criar um terreno baseado em agente, em relação à criação com fractais, é de que esse tipo de geração oferece um grande grau de controle ao usuário, enquanto mantém outras propriedades desejáveis de um algoritmo de PCG [46]. Assim como a geração baseada em agentes utilizada em dungeons, esse método, utilizado em terrenos e paisagens, permite a evolução do conteúdo em questão através da ação de um ou mais agentes de software. Ou seja, agentes podem se responsabilizar por elementos menores de um grande conjunto que se queira gerar conteúdo, como é o caso de gerar e interligar estradas em um terreno montanhoso gerado proceduralmente.

2.5.4.1 A GERAÇÃO DE TERRENO DE DORAN E PARBERRY

Em seu artigo de autoria própria [11], Doran e Parberry exploram um sistema mais controlável que usa agentes inteligentes para gerar elevações em mapas de altura, de acordo com restrições definidas pelo usuário, permitindo que seja criado um terreno com propriedades bem específicas.

Os agentes de Doran e Parberry agem em 3 fases distintas, a *litorânea*, a de *relevo* e a de *erosão*. Na fase litorânea, um grande número de agentes trabalha para gerar o contorno da massa de terra; Na de relevo, uma quantidade maior de agentes trabalha para definir as características do mapa, como praias e crescimento de montanhas por exemplo; Na fase de erosão, rios são adicionados ao se "erodir"partes dos terreno. Esses agentes funcionam de forma concorrente e assíncrona em cada fase [11].

Ainda em seu trabalho, Doran e Parberry empregam como prova de conceito 5 tipos diferentes de agentes, sendo eles:

1. Os *agentes litorâneos*, que criam uma massa de terra, possivelmente cercada de água.
2. Os *agentes de suavização*, que realizam rondas aleatórias de forma a tirar uma média de altura dos pontos próximos.
3. Os *agentes de praia*, que produzem áreas planas em partes da área costeira.
4. Os *agentes montanhosos* levantam cadeias de montanhas.

5. E por fim, os *agentes fluviais*, que erodem o terreno produzindo rios que correm das montanhas ao oceano.

Todos os pontos acima foram retirados diretamente do artigo de Doran e Parberry [11] para compreensão e exemplificação dos agentes, mas os autores informam que são possíveis de se aplicar ainda mais agentes.

O sucesso desse método se deve não à complexidade dos agentes em si, mas sim na interação possível entre eles, que pode ser imprevisível mas controlada [11]. Acreditamos que esse resultado "imprevisível e controlado"é de grande qualidade para terrenos, visto que queremos um conteúdo de fácil controle mas que apresente resultado o mais próximo da referência de terreno natural que temos.

2.5.5 CRIAÇÃO DE PAISAGENS BASEADA EM BUSCA

Segundo apresentado por Shaker, houveram várias tentativas de se aplicar métodos baseados em busca em terrenos, já que esses métodos apresentam uma forma de controlabilidade desejável. Ele cita dois tipos de métodos que podem nos ajudar com esse tipo de geração.

2.5.5.1 PROGRAMAÇÃO GENÉTICA DE TERRENO (GTP)

Em sua tese de mestrado [15], Frade apresenta uma nova técnica de geração de terreno, a programação genética de terreno - em inglês, GTP -, que é baseada no design evolutivo com programação genética, e consiste em uma evolução guiada de acordo com características específicas ou um apelo estético desejáveis. Essa técnica pode apresentar tanto terrenos esteticamente bonitos como próximo a terrenos reais, e escala de forma invariável, significando que as características de um terreno vão ser preservadas entre diferentes níveis de detalhes.

Seu método permite controlar algumas características de localização de terreno, eliminado a principal desvantagem de técnicas procedurais tradicionais, segundo ele [15].

2.5.5.2 GERAÇÃO SIMPLES DE MAPA RTS

Um outro método descrito por Togelius et. al [51] é capaz de produzir em tempo real um mapa com uma suave variação de altura para um jogo de estratégia. O fenótipo neste problema consiste em um mapa de altura e as localizações de recursos e ponto de início. As representações são diretas e a localização da base e dos recursos é apresentada como coordenadas [51]. Este tipo de mapa tem aplicação específica mas é de especial utilidade para esse tipo de gênero de jogo.

2.6 MÉTODOS DE PCG PARA VEGETAÇÃO E NÍVEIS

Uma das histórias de sucesso da PCG é da sua aplicação na geração de vegetação. Muitos jogos utilizam vegetação gerada de forma procedural e há muitos softwares disponíveis para isso no mercado. Assim como terrenos, a vegetação está presente em quase todas partes de jogos, e é utilizada para popular paisagens com grama, árvores, arbustos e outros tipos de elementos.

A vegetação também precisa ser criada em grande quantidade, e cada item de seu conjunto precisa ser ao mesmo tempo similar e reconhecível, mas não exatamente iguais, justamente para serem fidedignos ao que conhecemos fora do mundo digital. Dentre os elementos gerados por PCG, podemos dizer que a geração de vegetação é "fácil"se levarmos em conta que em muitos jogos ela é pouco ou nada funcional, no sentido de que uma planta "mal feita"não vai quebrar o jogo - talvez apenas deixar o visual estranho ou artificial, diferente de uma geração de terreno mal-sucedida, por exemplo.

Segundo Togelius, Shaker e Dormans, uma dentre as melhores e mais fáceis formas de se gerar uma árvore ou folhagem é utilizando *L-system* - uma forma particular de *gramática formal* - e interpretar os resultados como instruções para se fazer um desenho. Essas instruções se assemelham à estrutura que costumamos encontrar em plantas, na qual é possível observar um certo padrão entre folhagens e estruturas da flora. Dessa fora conseguimos encontrar no L-systems uma boa forma de reproduzir similaridades de forma "orgânica"e não copiada em toda sua forma.

2.6.1 GRAMÁTICAS

Um linguista chamado Noam Chomsky estudou sobre gramáticas formais na década de 50 e propôs uma classificação de gramáticas, estando dentre elas as linguagens livres de contexto, como uma forma de modelar a linguagem natural. Ele propôs um processo que normalizava as gramáticas livres de contexto, e seu trabalho influenciou John Backus, um cientista da computação estadunidense criador da linguagem Fortran, que percebeu que as linguagens de programação são as que são mais livres de contexto [1]. Há uma rica taxonomia aplicada a gramáticas, mas duas de suas características tem forte relevância para a aplicação de gramáticas na geração procedural de conteúdo: se elas são determinísticas e a ordem na qual elas se expandem importa [52].

Gramáticas formais são bastante usadas em reconhecimento de fala, em sistemas de tradução e em sistemas de entendimento de linguagens. Elas podem ser equipadas com distribuições de probabilidade, e os dados resultantes podem ajudar a interpretar a linguagem falada e escrita de forma concisa [18].

2.6.2 L-SYSTEMS

L-systems são uma classe de gramática na qual a principal característica é a reescrita paralela, introduzida pelo biólogo Aristid Lindenmayer em 1968, com o objetivo de modelar o crescimento orgânico de plantas e algas [27]. Basicamente um L-system têm a capacidade de criar resultados regulares mas de grande complexidade, sendo de interessante aplicação na PCG.

2.6.3 GERAÇÃO DE MISSÕES E ESPAÇOS COM GRAMÁTICAS

Para gerar níveis que pareçam consistentes e coerentes, é importante usar técnicas que possam gerar estruturas de forma que se destaque as qualidades individuais delas. Um nível é composto basicamente de uma missão e um espaço, e é importante que essas duas estruturas trabalhem bem juntas [52]. Por isso é aplicado à gramática em forma de grafo para se gerar uma missão,

conforme explicamos mais abaixo.

O grafo é mais útil que um texto para representar missões e espaços de jogos, especialmente no caso de jogos com certa complexidade[52]. Para gerar uma missão usando gráfico de uma gramática, é preciso definir o alfabeto dessa gramática e, segundo Togelius, esse alfabeto é gerado por nós e arestas. O ponto de começo do jogador pode ser representado por um nó, enquanto que ações conectadas e seguidas podem ser representadas por arestas. Esse tipo de representação é mais facilmente compreendida de forma visual, já que sua representação escrita pode parecer trabalhosa.

2.7 MÉTODOS DE PCG PARA REGRAS E MECÂNICAS

Existem vários métodos para gerar conteúdo de elementos de um jogo, como já discutimos sobre alguns para vegetação e terreno nos tópicos anteriores, por exemplo. Mas tudo isso se baseia no fato de você já ter um "Jogo"pré-definido, ou seja, uma base que represente as regras de funcionamento do universo desse jogo. Em caso contrário, precisamos gerar essas regras ou da forma tradicional, ou podemos utilizar aplicar a PCG.

Segundo Nelson et. al, codificação de regras e avaliação de funções podem codificar a especialidade e estilo de game design, de forma que nos ajude a entender o próprio processo [de game design]. Formalizar regras é definir mais precisamente um espaço de possíveis outras regras, do que escrevê-las, em termos qualitativos - e ao escolher o que queremos formalizar, escolhemos os aspectos que queremos explorar e variar[37]. Segundo Nelson et. al, cada gerador de regras pode ser visto como "uma micro-teoria executável"de game design simplificada e até por vezes caricata.

2.7.1 CODIFICANDO REGRAS DE JOGO

O trabalho de um gerador de regras de jogo pode ser bastante complexo: para gerar regras precisamos codificá-las ou representá-las de uma forma que seja reconhecível por uma máquina e que possa trabalhar em conjunto com algum software, mas é difícil determinar qual o ponto de partida de um gerador generalista que seja voltado para jogos [37]. É difícil devido as características heterogêneas dos jogos: um jogo pode ser baseado em turno ou não, pode ser basear em elementos gráficos ou de texto, pode obedecer regras matemáticas ou não, pode se utilizar de elementos sonoros ou não, entre vários outros aspectos que podem ser encontrados em diferentes títulos e gêneros de jogos.

Com os pontos levantados anteriormente, Nelson et. al sugerem que, para termos um resultado mais relevante de regras geradas por PCG, precisamos escolher uma codificação que sua amostra inclua uma distribuição densa de elementos que sejam considerados jogos, e que seja determinado um critério básico de jogabilidade aplicado a todos eles. Além disso, também é importante que seja apontado um gênero de jogo específico, devido às diferenças entre gêneros que foram apontadas no parágrafo anterior.

Também não podemos deixar a codificação muito "limitada"à apenas um tipo de jogo, já que as variações encontradas nos resultados poderiam não ser muito interessantes para um sistema gerador de jogos ao apresentar um resultado específico a um tipo de entrada. Nelson et. al

sugerem que devemos encontrar um "meio termo"entre uma geração completamente genérica e uma completamente específica.

Um artigo de Barney Pell, publicado no livro *Heuristic programming in artificial intelligence 3: The third Computer Olympiad* em 1992, propõe um *Metagame* - uma ideia de um programa, disponível publicamente, que produziria outros programas e esses outros programas receberiam de entrada um conjunto de regras aplicadas à um conjunto de *novos* jogos dentro de uma classe previamente especificada.

Segundo Barney Pell, o Metagame encoraja os pesquisadores a considerarem todos os possíveis problemas de jogabilidade de um jogo, e, pelo fato da entrada ser composta de novos jogos, o usuário do Metagame teria que se atentar ao inserir as regras que definem os jogos e objetivos, evitando recursos já prontos que poderiam influenciar de forma não desejada o resultado final. Os programas têm recursos limitados e não se pode assumir que uma estratégia de busca fixa irá funcionar, já que os jogos variam muito seus espaços de busca - então os programas devem modificar as estratégias de busca de acordo com cada jogo recebido [41]. Pequenos espaços oferecem uma jogabilidade e controle mais densos, mas espaços maiores podem permitir uma variação maior que apresente um resultado mais interessante ao jogo aplicado [41].

Nelson et. al destacam em seu artigo duas seções de experimentos com geradores de jogos, nos quais, segundo eles, mais pesquisadores têm proposto estudos: os jogos de tabuleiro (ou de mesa) e os jogos 2D de lógica visual.

2.7.2 JOGOS DE TABULEIRO

De acordo com Nelson et. al [37], o primeiro domínio no qual sistemas foram criados para gerar regras de jogo proceduralmente foi o de jogos de tabuleiro. Eles apresentam várias características que indicam um bom lugar para se começar, como a estrutura finita e discreta dos jogos que simplifica a codificação, diferente de jogos de computador, que são definidos por um corpo de código que pode ser, por vezes, complexo [37].

2.7.2.1 JOGOS SIMÉTRICOS / ESTILO XADREZ

O sistema gerador de regras mais antigo, segundo Nelson et. al, foi o Metagame [41], que gerava jogos "simétricos"e "estilo jogo de xadrez". "Estilo xadrez"foi uma definição usada para um jogo que roda em uma grid e é estruturado em volta de dois jogadores que jogam em turnos, movendo as peças de acordo com determinadas regras Enquanto que a parte de simetria se refere ao fato dos jogadores começarem em lados opostos com configurações iniciais de jogo simétricas e todas as regras são aplicáveis a ambos [41]. Por isso, o Metagame não representa jogos assimétricos nos quais os jogadores iniciam com diferentes peças e obedecem a diferentes regras.

A motivação de Pell para construir o Metagame foi para testar sistemas de IA, e ele propôs que mais avanços na IA em testas a jogabilidade com diferentes tipos de jogos, nos quais as regras não seriam todas apresentadas para a máquina de primeira [41]. Dessa forma, segundo o próprio Pell, o Metagame foi criado para uma testagem mais generalista do espaço, se tornando o primeiro sistema de PCG para regras de jogos [37].

2.7.2.2 JOGOS DE CARTAS

Jogos de cartas têm algumas similaridades com jogos de tabuleiro, como o fato de serem baseados em turno e de terem unidades discretas - as cartas -, que se têm uma quantidade limitada [37]. Font et al. desenvolveram uma linguagem de descrição para jogos de cartas [14], e tentaram, no espaço definido por essa linguagem, utilizar a busca evolutiva.

2.7.3 JOGOS 2D DE LÓGICA VISUAL

Nelson et. al utilizam o termo "jogos 2D de lógica visual"para representar jogos em que a jogabilidade é baseada em elementos 2D movendo pelo mapa, colidindo e aparecendo e desaparecendo [37]. Apesar desse conceito parecer básico, vários jogos podem ser gerados a partir dessa ideia, como os citados por Nelson et. al no estilo *Pac-Man* e *Tetris*.

Geralmente eles apresentam agentes de jogo mais complexos do que jogos de tabuleiro, ou ainda interações entre agentes que possam ser facilmente manipuladas por cálculo humano em um jogo de tabuleiro [37]. Muitos desses jogos apresentam um avatar, no qual o jogador assume o controle e, ao invés de selecionar peças em um tabuleiro, ele *é* a própria peça.

2.8 MÉTODOS DE PCG PARA HISTÓRIAS E MISSÕES

Histórias são um elemento bastante frequente no universo de jogos, e as histórias tanto podem ser curtas como de grande importância para a progressão dentro um jogo. Assim como a geração de regras, a geração procedural de histórias é um pouco diferente dos elementos mais comuns presentes no contexto de conteúdo procedural, já que seu impacto varia diretamente com o gênero e mecânica do jogo que se quer integrar.

Uma forma comum de se integrar histórias a jogabilidade de um jogo é através de missões [7], na qual o jogador recebe um objetivo e, após atingi-lo, avança de alguma forma dentro do jogo.

Existem várias razões de se aplicar PCG em histórias, uma delas, apontada por Cheong et. al, é de que mundo de jogo gerados proceduralmente podem não oferecer motivação suficiente a um jogador, de modo que a história adiciona um fator de jogabilidade, permitindo que missões sejam flexíveis e até mutáveis a diferentes fatores [7].

2.8.1 GERAÇÃO PROCEDURAL DE HISTÓRIA POR PLANEJAMENTO

Segundo Cheong et. al, uma forma de pensar sobre geração procedural de histórias é considerá-las como um problema de planejamento. Em IA, algoritmos de planejamento procuram por sequências de ações que satisfaçam um objetivo, e a geração procedural funciona de forma similar.

Um dos primeiros e mais influente sistema de geração de história foi o *Tale-Spin*, que segundo seu autor é um programa que escreve histórias utilizando de conhecimento sobre resolução de problemas, espaço físico, relação interpessoal, dados de caráter, necessidades fisiológicas, estrutura de história e inglês [33]. Ele segue a abordagem descrita por Cheong et. al, de simular possíveis caminhos de uma história e de recontar os eventos a medida que os caminhos

se repetem.

Gerar histórias simulando um mundo de histórias tem suas desvantagens, já que essa geração não leva em conta o que de fato faz com que uma história interessante seja diferente de um mero *log* de eventos [7]. Para se resolver esse problema devemos olhar do ponto de vista do autor da história, no sentido de juntar narrativas em sequências que façam sentido para o objetivo que se quer alcançar [7].

No contexto de jogos, o planejamento do ponto de vista do autor pode ser um problema, segundo apontado por Cheong et. al, já que quem de fato atua no mundo do jogo é o jogador, e não o "diretor". Essa e outras questões ainda continuam abertas, segundo apontado por Chegong et. al, então a geração procedural de histórias em jogos ainda é uma área de pesquisa bastante ativa.

2.8.2 PLANEJAMENTO COMO BUSCA ATRAVÉS DO ESPAÇO DE UM PLANO

O planejamento pode ser visto como um processo de busca dentro de um espaço por soluções em potencial para um determinado problema, quando temos conhecimento sobre o domínio desse problema [7]. O problema em questão é chamado de *problema de planejamento* e consiste de dois estados: o *estado de objetivo*, sendo o objetivo o fim que se quer chegar, e o *estado inicial*. Uma solução para um problema de planejamento é um *plano*, que contém uma sequência de ações. Um plano é *sólido* se ele consegue chegar no estado de objetivo durante sua execução partindo do estado inicial [7].

Dessa forma, o conhecimento de domínio é representado como uma biblioteca de *operadores de planos*, em que cada operador consiste de um conjunto de pré-condições e um conjunto de efeitos. Pré-condições são aquelas que precisam se compridas para que um operador execute, e os efeitos são essas mesmas condições que são atualizada após a execução de um operador de plano. Esse tipo de execução é associada ao *algoritmo de espaço de estado* [7].

2.8.3 MODELO DE DOMÍNIO

Um *modelo de domínio* é a biblioteca de templates de operadores de planos que codifica o conhecimento em um domínio em particular [7]. Segundo Cheong et. al várias linguagens formais foram propostas para descrever problemas de planejamento em termo de estados, ações e objetivos. Duas desses linguagens foram mais usadas para planejamentos clássicos, sendo elas *STRIPS* e *ADL.*

De forma resumida, planos no estilo de STRIPS representam um estado como um literal proposital, onde literais são livres de variáveis e funções, e realizam suposições em mundo fechado [7]. Enquanto que ADL - sigla para Linguagem de descrição da ação - ajuda em problemas mais complexos adicionando algumas características, como a quantificação de variáveis e a permissão de efeitos condicionais.

2.8.4 PLANEJANDO UM HISTÓRIA: ALGORITMO DE RIEDL E YOUNG

No exemplo oferecido por Cheong et. al no que diz respeito a esse algoritmo, uma história pode ser "verdade"no que diz respeito as suas condições na gramática formal, mas pode não

fazer sentido em seu conteúdo lógico. Os autores [7] afirmam que esse tipo de problema pode acontecer com histórias centradas no autor, que tendem a ignorar as intenções de personagens individuais.

Uma abordagem alternativa seria a geração de histórias centradas em personagens, que permitiria a independência das ações dos personagens e poderia produzir um conjunto de ações lógicas consistente [7]. Porém o planejamento não pode ser feito de forma totalmente centrada em personagens, e, para resolver esse problema, Riedl e Young propuseram um algoritmo de planejamento orientado ao objetivo, que balanceia a abordagem centrada no autor com a centrada em personagem durante a geração da história [42].

2.9 MÉTODOS DE PCG PARA LABIRINTOS - COM APLICAÇÃO DE ASP

Answer set programming - ASP, com tradução livre para português como "Programação de conjuntos de respostas", é uma abordagem de programação lógica em que restrições e relações lógicas são declaradas em uma linguagem de programação do paradigma de Programação em Lógica Matemática [36].

Podemos especificar como queremos que nosso conteúdo gerado fique através de ASP, e, aplicando a entrada em um solucionador ASP, temos como saída o retorno do conteúdo que se enquadra nas especificações do programa em questão [36]. Essa saída é o que chamamos de *conjuntos de respostas* - o *answer set*. A linguagem utilizada na ASP é a *AnsProlog* e, apesar de ser sintaticamente similar ao *Prolog*, outra linguagem para lógica matemática, sua interpretação é bem diferentes das outras linguagens de programação [36];

Ao se fazer a combinação de restrições e regras mais simples, é possível gerar restrições mais complexas. Essa complexidade pode levar a propriedades de design de nível interessantes para a PCG [36];

2.10 AVALIANDO GERADORES DE CONTEÚDO

Avaliar um gerador de conteúdo é um atividade de grande importância, mas também é uma que apresenta bastante dificuldade em sua realização. Criar um gerador de conteúdo é uma tarefa tangível, mas, quando falamos em criar um gerador bom e eficiente, surgem várias dúvidas: o que é considerado um gerador bom? O que é bom para a geração de um conteúdo é bom para todos os outros?

Shaker, Smith e Yannakakis [45] discorrem que a qualidade de um conteúdo resultado de um processo de geração procedural pode ser avaliada tanto de uma perspectiva *top-down*, seguindo métricas estatísticas, como de uma *bottom-up*, observando a experiência direta do jogador com o conteúdo. Um ponto a se considerar é o de que a avaliação feita a partir da observação do jogador pode ser impactada por dados comportamentais e por dados mais "objetivos- como é o caso de aspectos fisiológicos, citado pelos autores.

O resultado final da pesquisa apresentada pelos autores [45] foca em uma avaliação subjetiva através de um formulário, mas eles também consideram que um entendimento mais integral pode ser obtido através de uma abordagem híbrida, que combine uma série de métodos

top-down e *bottom-up* relatados em sua pesquisa.

2.11 UMA TAXONOMIA DA PCG

Com a variedade de problemas na geração de conteúdo, é importante ter uma estrutura que permita observar as diferenças e similaridades entre as abordagens de métodos e algoritmos. Um trabalho importante com a proposta de uma taxonomia foi o apresentado por Togelius et. al [53] em 2010. Porém, do ano em que foi lançado até agora, houve a revisão de alguns aspectos, que foram apresentados também no livro de PCG em jogos [47]. Aproveitamos para mesclar essas definições a alguns conceitos mais atualizados e respectivamente citados e referenciados em cada tópico em que se apliquem.

2.11.1 ONLINE VS. OFFLINE

Técnicas de PCG podem ser usadas para gerar conteúdo à medida que o jogador progride, de forma online, ou offline. Na primeira, o conteúdo é "gerado"de acordo com a forma como o jogador joga, enquanto que na última a geração pode sugerir conteúdo para o game designer, que pode, por exemplo, editar e aperfeiçoar aquela versão específica do jogo. Também podem ocorrer casos intermediários, onde o algoritmo online sugere alguns comportamentos para o offline e vice-versa [53]. Kelly e McCabe [23] apresentam uma descrição familiar, porem sob o nome de *real-time*, ou seja, em tempo real, de execução.

2.11.2 NECESSÁRIO VS. OPCIONAL

Outra discussão acerca de conteúdo gerado proceduralmente é se esse conteúdo resultante é necessário ou opcional. O conteúdo que o jogador precisa para progredir no jogo é definido como necessário, enquanto que o opcional é aquele que pode ser "evitado"ou ignorado, como por exemplo itens e histórias que não afetem na narrativa principal de um jogo com progressão linear [53].

Oliveira e Seabra [38] também afirmam que é importante que o conteúdo esteja correto, independentemente de ser opcional ou não, e essa análise é complexa e precisa ser executada matematicamente, já que geralmente não conseguimos testar todas as entradas e saídas do gerador.

2.11.3 GENÉRICO vs. ADAPTATIVO

Esse conceito é apresentado no livro de PCG em jogos [47] e basicamente é definido como: conteúdo genérico se refere ao conteúdo que é gerado sem levar em conta o comportamento do jogador, enquanto que o adaptativo é personalizado e centrado no jogador. Neste último cada interação do jogador é analisado e o conteúdo é gerado com base no último comportamento captado. A maioria dos jogos, segundo Shaker et. al, utiliza PCG de forma genérica, mas a forma adaptativa têm recebido cada vez mais atenção pela academia.

2.11.4 ESTOCÁSTICO VS. DETERMINÍSTICO

Todos os algoritmos de PCG criam um conteúdo "expandido"até certo ponto, mas podemos distinguir a geração entre eles a partir do que pode ser parametrizável ou não. A PCG determinística permite a "regeneração"de um mesmo conteúdo, dado que receba o mesmo ponto de partida e os mesmos parâmetros para seu método. Dessa forma, a PCG estocástica age da forma oposta, onde geralmente não é possível recriar um mesmo conteúdo dada uma mesma entrada.

2.11.5 SEMENTES RANDÔMICAS VS. VETORES DE PARÂMETRO

A distinção dessas duas propriedades se encontra nos dados de entrada dos algoritmos. Alguns só recebem um valor de "semente"de um gerador de número randômico, enquanto que outros podem receber valores que controlem os resultados [38]. Segundo Oliveira e Seabra [38], há uma relação entre esses dois aspectos e o critério de controle de Kelly e McCabe [23], no sentido de que quanto mais parâmetros disponíveis, mais controle "fino"é possível para o usuário.

Entendemos que, de forma mais simplificada, as sementes criam aleatoriamente um conjunto de parâmetros que alimentam a geração, enquanto que os vetores permite a modificação de cada parâmetro pelo usuário de forma individualizada.

2.11.6 CONSTRUTIVO VS. GERAR-E-TESTAR

Algoritmos construtivos geram o conteúdo uma única vez, porém é preciso verificar a "qualidade"do conteúdo gerado [47], como no caso do uso de fractais para gerar terrenos - de forma simplificada, um fractal segue uma regra de reprodução em cadeia que pode gerar tanto um resultado artificial como um que esteja mais próximo de simular um elemento real.

Um algoritmo gerar-e-testar, traduzido livremente do inglês *generate-and-test*), gera seu conteúdo de acordo com seu mecanismo e, seguindo um critério pré-definido, testa esse resultado; Em caso de falha, todo ou parte do conteúdo em questão é descartada e regerada, de forma que esse processo continue até atingir um padrão desejado [47].

2.11.7 GERAÇÃO AUTOMÁTICA VS. AUTORIA MISTA

Esse ponto da taxonomia basicamente difere a geração de PCG feita totalmente por uma máquina, de uma que permita maior influência e manipulação de entradas por parte do usuário, no caso possivelmente um designer do jogos.

Dessa forma, a geração automática é a geração com entrada limitada do usuário, enquanto que a autoria mista é um paradigma que foca em incorporar ainda mais a entrada do usuário - seja ele o designer ou jogador - de forma que isso impacte no processo de design do jogo [47].

3 SOLUÇÃO PROPOSTA

3.1 A REVISÃO BIBLIOGRÁFICA

No início da pesquisa desse trabalho, buscamos diversas ferramentas de geração procedural de PCG, em sua grande maioria sendo soluções aplicadas a jogos, mas também a softwares relacionados - como de modelagem 3D, por exemplo.

O intuito foi de juntar um conjunto de alguns projetos open source e populares - ou de certa forma relevantes no que se propõem. Também consideramos importante que os projetos não fossem muito antigos, para evitar códigos abandonados por seus autores. Porém, isso não impede a catalogação de projetos antigos: se o projeto ainda for adequado à nossa realidade, acreditamos que é válido catalogá-lo para uso. Também tentamos abranger ao menos uma ferramenta pra cada área de aplicação de PCG apresentada nessa pesquisa, de forma que seja possível exemplificar a prática de geração em cada uma delas.

3.1.1 A FERRAMENTA DE VISUALIZAÇÃO DE SOLUÇÕES DE PCG

A ferramenta proposta para visualização [1] teve como inspiração o trabalho realizado por Valérian Fraisse, Catherine Guastavino e Marcelo M. Wanderley, publicado em 29 de abril de 2021 para a NIME - Conferência Internacional de Novas Interfaces para Expressão Musical. O nome do trabalho pode ser traduzido diretamente do inglês como "*Uma Ferramenta de Visualização para Explorar Instalações Interativas de Som*", e ele é basicamente composto por uma revisão e elaboração de uma taxonomia combinando diferentes perspectivas, proporcionando uma visualização dinâmica via uma aplicação web [16].

Para o desenvolvimento [2], priorizamos ferramentas que proporcionassem um rápido desenvolvimento de código, com uma framework que fosse fácil de se desenvolver e com bastante material em caso de dúvidas. Por esse motivo, optamos por utilizar a biblioteca JavaScript de código aberto *React*, realizando a formatação dos dados a serem exibidos e do contexto visual utilizando funções da biblioteca de visualização.

Para a organização e junção dos dados, utilizamos o *Planilhas Google*, do pacote gratuito de Editores do Google Docs, juntamente com a API gratuita *opensheet* - um projeto open source construído para buscar em uma Planilha Google dados e retorná-los para leitura em formato *JSON* [4] - Notação de Objeto JavaScript. Dessa forma, criamos a nossa própria API [3] a partir de uma planilha, e os dados foram organizados da melhor forma que correspondesse ao padrão aceito pela biblioteca de visualização (o D3.js, descrito logo abaixo) e que fosse possível fazer o menor número de alterações via código para exibi-los corretamente.

A ferramenta utilizada para o gráfico de visualização de dados foi a biblioteca JavaScript *D3.js* [35], que produz visualizações de dados dinâmicas e interativas em navegadores web. Ela apresenta diferentes padrões de gráficos e visualizações, e acabamos por optar pela variação "Zoomable Sunburst", tanto por ser similar ao utilizado no trabalho de inspiração de

[1] `https://pcg-tools-visualization.netlify.app/`

[2] `https://github.com/caroolpmelo/pcg-tools-visualization`

[3] `https://cutt.ly/LHmWN6f`

Visualização de Instalações Interativas de Som, como por acreditarmos que é o que oferece a melhor forma de visualização para os dados que queremos apresentar.

Para a hospedagem da aplicação desenvolvida, foi utilizada a plataforma de nuvem *Netlify*, tanto por sua facilidade de configuração e lançamento de servidor e domínio, como também por ser uma opção gratuita e popular.

3.1.2 OS PARÂMETROS HIERÁRQUICOS DA VISUALIZAÇÃO

Para categorização dos dados, e pela estrutura de visualização escolhida, tivemos que organizar os dados priorizando os seguintes critérios em ordem crescente de quantidade de soluções disponíveis para eles:

1. Área foco que se quer aplicar: se o objetivo é aplicar a uma geração procedural de terreno, de plantas, de missão ou de dungeon, por exemplo [Figura 3.1].

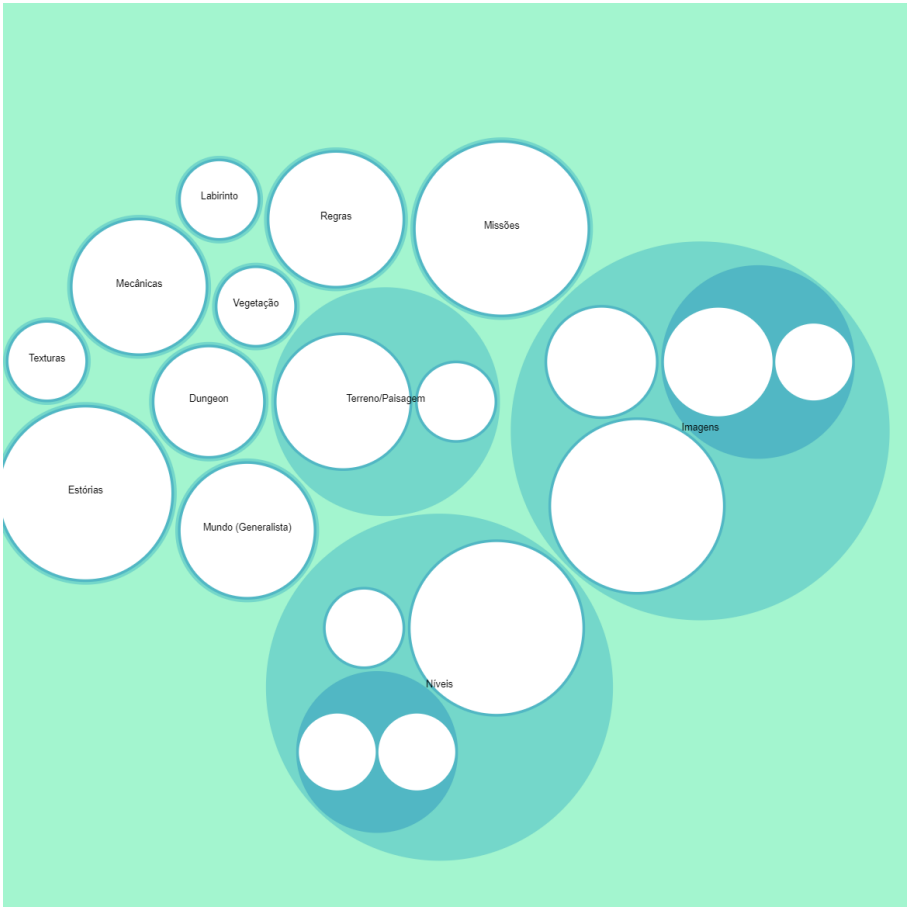

Figura 3.1 Primeira etapa gráfica de categorização dos dados, partindo do tipo de conteúdo que se quer gerar.

2. Tipo de método que se quer utilizar: se é desejável uma geração estocástica, baseada em agentes ou baseada em busca, por exemplo [Figura 3.2].

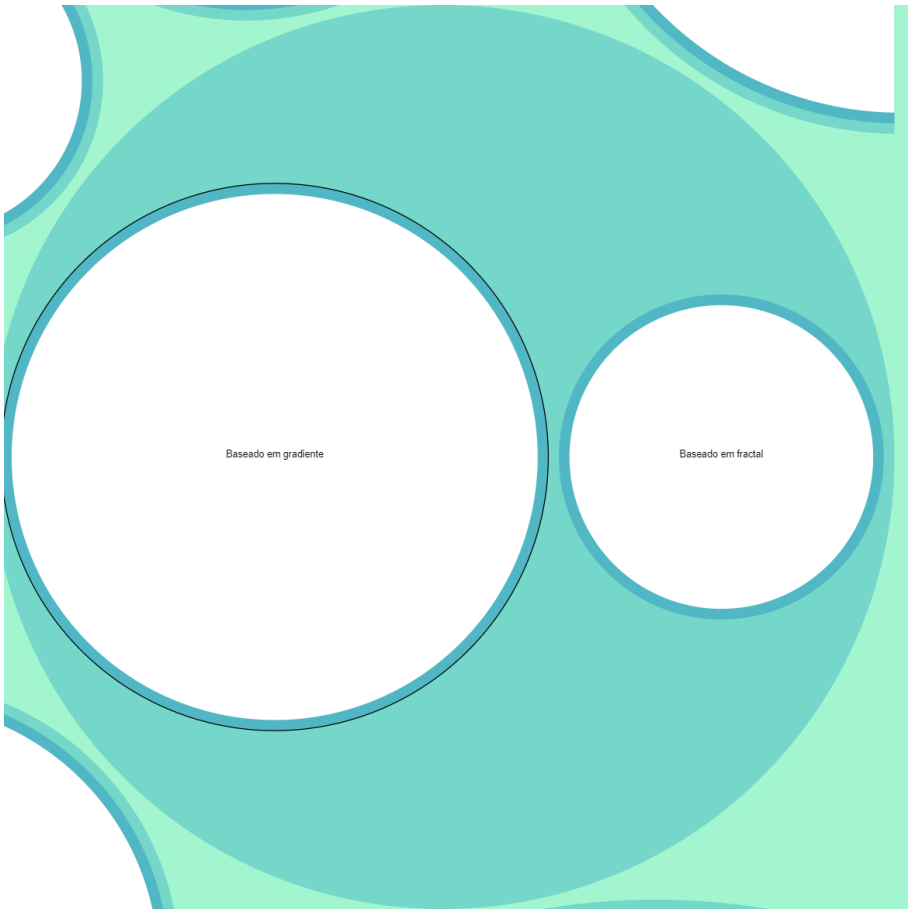

Figura 3.2 Segunda etapa gráfica de categorização dos dados, partindo do método de geração do conteúdo.

3. Algoritmo a ser priorizado na utilização: de acordo com o método utilizado na seleção anterior, são exibidos os possíveis algoritmos dentro de escopo - se as ferramentas não forem claras quanto aos algoritmos utilizados, é possível agrupá-las em uma categoria de algoritmos "desconhecidos"[Figura 3.3].

O número ao lado do nome de seleção representa a quantidade de ferramentas encontradas para cada "folha"dessa divisão. Ao realizar a seleção de uma dessas seções, são exibidas ferramentas que casam com a condição, juntamente com dados resumidos de cada uma, descritos mais a frente [Figura 3.4].

Optamos por priorizar ferramentas open source gratuitas, pois é uma forma de permitir a análise do código para fins de performance e estudo. Caso o(s) autor(es) da ferramenta em questão não tiverem informado o tipo de algoritmo e método que utilizaram para a geração, a ferramenta é agrupada em um grupo de métodos "desconhecido", dentro da sua área. Escolhemos trabalhar dessa forma pois parar para analisar cada repositório que encontrasse essa barreira iria demandar um tempo e atenção que dificultaria que esse trabalho fosse entregue a tempo - e infelizmente foi bem comum encontrar repositórios assim durante o processo de pesquisa.

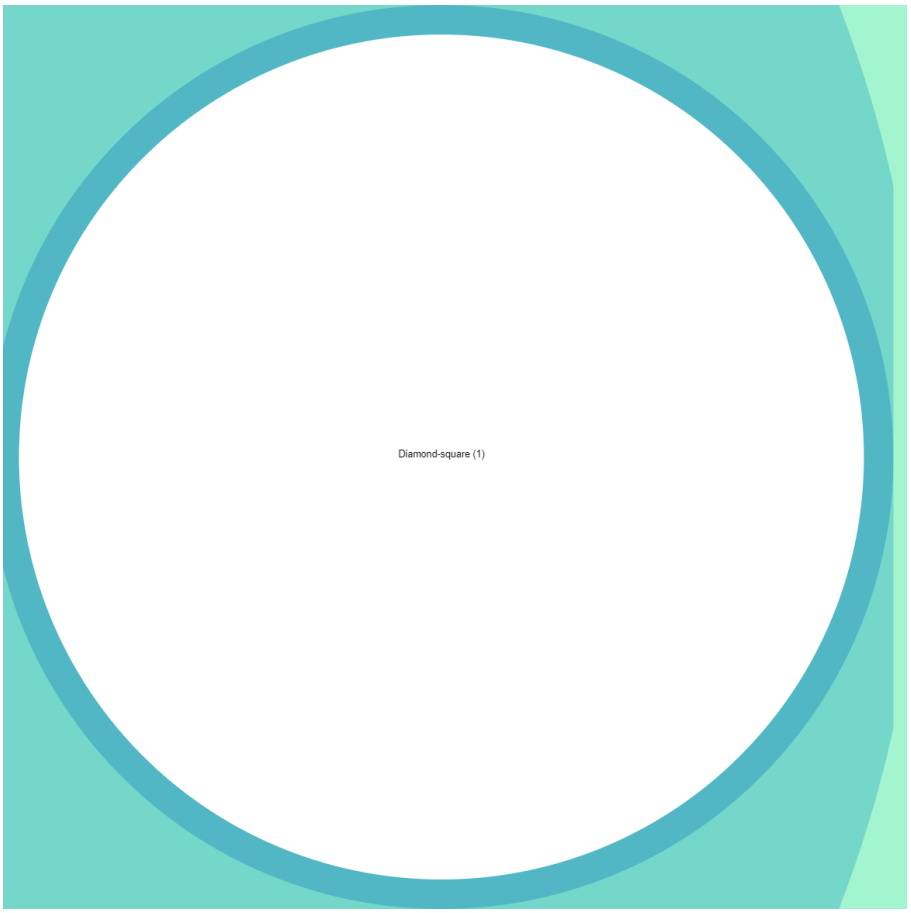

Figura 3.3 Terceira etapa gráfica de categorização dos dados, partindo do algoritmo de geração do conteúdo.

A aplicação desenvolvida como resultado desse trabalho busca ajudar na escolha de ferramentas de PCG para um jogo e tornar acessível à diferentes pessoas essa revisão ferramental centralizada. Optamos por evitar ferramentas pagas, já que isso poderia prejudicar o quesito financeiro, se essa fosse uma das limitações de quem procura por métodos de PCG mais acessíveis para seu jogo.

3.1.3 ESTRUTURA DE CLASSIFICAÇÃO DOS DADOS

Com o objetivo de uniformar os dados e facilitar sua catalogação, classificamos as ferramentas de PCG de forma de fácil entendimento para nós e de fácil organização da API na planilha. Adicionamos colunas que permitem apresentar os principais dados que acreditamos ser de importância ao se pesquisar uma ferramenta [Figura 3.6].

As colunas de dados se encontram dispostas da seguinte forma, na mesma ordem que apresentadas na planilha, da esquerda para a direita:

- *Area*: é a área de aplicação do método. Ela pode ser definida como "Vegetação", "Terreno"e "História", por exemplo. No caso de existir métodos aplicáveis em mais de uma,

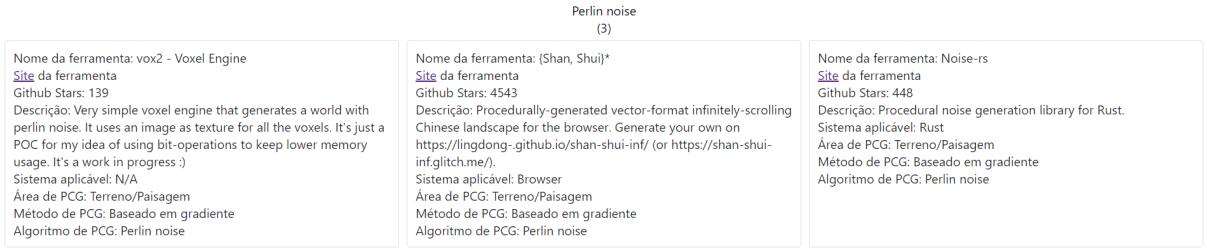

Figura 3.4 Representação da lista de ferramentas resultantes da escolha do tipo de PCG que se quer trabalhar.

Figura 3.5 Estrutura hierárquica dos dados, no formato de árvore.

seria ideal criar uma nova entrada com a junção separada por barra, como no caso de "Vegetação/Terreno".

- *Method*: é o método de aplicação da geração. Ele pode ser "Estocástico", "Baseado em Gramática"ou "Baseado em Busca", por exemplo.
- *Algorithm*: é o algoritmo de aplicação da geração. Como exemplo, "Perlin noise"e "L-system". Traduzir ou não esses termos não faz muita diferença, desde que se mantenha a mesma string para representação em outras linhas.
- *Tools*: é nome individual da ferramenta. Acabamos deixando este dado no plural por conta do formato de dado que se estava trabalhando anteriormente, mas ele representa uma string de texto de uma única ferramenta.
- *Website*: é o site da ferramenta em questão, podendo ser desde o repositório a qualquer outro que representa uma forma de consulta material.
- *Github Stars*: são as estrelas no repositório do GitHub dessa ferramenta. Este nome veio pela ideia de se observar a popularidade da ferramenta com base no quantitativo de "fãs", mas no caso de projetos que estejam em outros sites, podemos utilizar um outro parâmetro numérico de equivalência.

- *Description*: é a descrição exposta no repositório da ferramenta. Pode-se adicionar mais texto, desde que não fique muito extenso.
- *Engines/Systems*: são os sistemas ou engines nos quais essa ferramenta pode ser aplicada. Utilizamos o termo "N/A", quando esse dado não é exposto pelo autor ou quando a ferramente é generalista, e as vezes colocamos linguagens de programação por indicação de descrições dos autores.

A	B	C	D	E	F	G	H
Area	Method	Algorithm	Tools	Website	Github Stars	Description	Engines/System
Terreno/Paisagem	Baseado em gradiente	Perlin noise	vox2 - Voxel Engine	https://github.com/L	139	Very simple voxe It's a work in pro	N/A
Terreno/Paisagem	Baseado em gradiente	Perlin noise	{Shan, Shui}*	https://github.com/L	4543	Procedurally-ger	Browser
Terreno/Paisagem	Baseado em gradiente	Perlin noise	Noise-rs	https://github.com/R	448	Procedural noise	Rust
Terreno/Paisagem	Fractal	Diamond-square	A1essandro's Diamond A	https://github.com/A	15	Algorithm for ger	PHP
Vegetação	Baseado em gramática	L-system	tree-gen	https://github.com/fr	634	Procedural gene	Blender
Níveis	Baseado em busca	Algoritmo de busca A*	Moolt's Level Generator	https://github.com/M	100	Unity plug-in for	Unity

Figura 3.6 Fragmento da tabela utilizada como API de dados.

3.1.4 DIFICULDADES ENCONTRADAS NA CATALOGAÇÃO

Algumas das dificuldades já foram mencionadas nos tópicos anteriores, mas de forma centralizada podemos dizer que:

- Alguns repositórios não apresentaram detalhamento quanto aos métodos e algoritmos utilizados, o que dificultou o processo de pesquisa.
- Algumas ferramentes tem aplicações bem interessantes, mas infelizmente os autores a deixaram de atualizar em algum momento e o projeto parecia estar abandonado.
- Quanto mais específica a geração, mais difícil encontrar uma ferramenta que não esbarrasse nos pontos anteriores.
- O tempo disponível para realização da pesquisa, catalogação e escrita deste trabalho de graduação foi menor se comparado a períodos tradicionais. Como consequência alguns cortes tiverem que ser feitos, mas tentamos trabalhar da melhor maneira possível com o que tínhamos.
- Ainda em consequência desse ponto, tínhamos inicialmente planejado fazer uma pesquisa sobre a utilização da ferramenta proposta, porém não foi possível realizá-la em tempo hábil até o prazo final de entrega deste trabalho.

Também foram encontradas algumas dificuldades no desenvolvimento da ferramenta e na formatação dos seus dados. O primeiro ocorreu por conta da pouca experiência com o framework React, enquanto o segundo ocorreu pelo tipo de formato que o D3.js pede. Essa formatação teve boa parte da preocupação por ser essencial para a exibição correta do gráfico, e felizmente conseguimos resolver a tempo.

3.1.5 CARÁTER EXPANSIVO DA FERRAMENTA PARA CRESCIMENTO DA MASSA DE DADOS

Por fim, essa ferramente foi pensada em ser uma fonte de material em constante expansão de dados, pois sua API é obtida a partir dos dados da planilha disponibilizada no início desse capítulo. Qualquer pessoa que tenha acesso ao link pode adicionar mais dados de ferramentas em qualquer área. Apenas deve-se observar a importância de se manter a estrutura de dados das entradas anteriores para a correta leitura e distribuição dos mesmos na visualização.

4 CONCLUSÃO

A realização de uma pesquisa sobre os conceitos básicos da PCG, em conjunto com referências bibliográficas que tragam reforço ou expansão para conceitos consolidados no tema, mostra como a PCG é uma área com ampla aplicação e ferramental. Mesmo entrando no nicho de PCG para jogos, a quantidade de material fonte e de ferramentas disponíveis é gigantesca, necessitando até de uma certa cautela em se manter um foco do tipo de informação que se quer obter - pois é fácil se perder dentre tantas opções.

Como citado e referenciado em parágrafos de outros capítulos, a área de jogos cresce comercialmente cada vez mais, e isso impacta no maior crescimento e investimento na mesma. A forma de gerar conteúdo em jogo é um assunto que bastante nos interessa, por ser uma forma de explorar a criatividade computacional de diferentes maneiras nesse tipo de mídia. A PCG aplicada a jogos não gera apenas o divertimento, mas também a curiosidade e admiração pelo trabalho na área.

Acreditamos que a PCG ainda vai convergir bastante com outras áreas, como no caso da sua relação direta e indireta com métodos de inteligência artificial, e que vão ser descobertas novas formas de interação com essa mídia de jogo a medida que as formas de interação com conteúdo forem expandindo. No fim da década de 90 e começo dos anos 2000, tínhamos até uma visão futurista sobre o uso de realidade aumentada e virtual em jogos, por exemplo, se pensarmos em materiais como o filme *Tron* de 1982, que mostra como se é fácil criar expectativas sobre o funcionamento de um suposto futuro. Hoje, porém, já se passados 20 e poucos anos dessa época citada no começo, é comum encontrarmos à venda, em lojas, equipamentos e diversos títulos de jogos com esse tipo de jogabilidade - VR e AR. Essa foi apenas uma analogia, para mostrar que as vezes o futuro está logo na nossa frente e, quando menos esperamos, ele se torna nosso presente.

4.1 TRABALHOS FUTUROS

Durante a realização deste trabalho, percebemos que há um problema de estruturação no que diz respeito às aplicabilidades das ferramentas de PCG, e a visualização proposta por este trabalho é uma consequência desse problema. Acreditamos que uma forma ainda mais incisiva, e que permita uma maior participação e inclusão da comunidade de desenvolvedores de softwares relacionados à PCG, seja a de criar e promover uma forma de "auto-cadastro"dos autores de cada ferramenta. Ou ainda, um modelo de documento que estaria inserido no repositório das ferramentas, e que seus autores poderiam apontar essa informação em um cadastro. Desta forma faríamos o reconhecimento do arquivo específico e subsequente cadastro do projeto/ferramenta na nossa catalogação de aplicações direcionadas à PCG.

Assim, pouparíamos o trabalho manual de pesquisar e inserir dado um a um, e focaríamos em fazer as *perguntas certas* para categorizar a ferramenta de forma ideal. Essa cadastro poderia consistir de perguntas objetivas e subjetivas, no intuito de saber mais detalhes sobre o objetivo da ferramenta. Desta forma poderíamos, por exemplo, saber se uma ferramenta que gere vegetação, com seu método baseado em gramática, utilizando o algoritmo *L-system*, é in-

dicada para a geração de uma vegetação simples e rasteira, ou para uma vegetação de floresta densa e complexa.

4.1.1 PCG E DEEP LEARNING

A geração procedural de conteúdo em jogos tem uma longa história [28], e métodos de geração têm sido aplicados aos mais variados tipos de conteúdo, tais como níveis, mapas e texturas. Recentemente a aprendizagem profunda - também conhecida como *deep learning* - impulsionou uma variedade notável de aplicações dentro da produção de conteúdo [28] - que também engloba a produção para jogos. Enquanto alguns métodos de deep learning mais robustos são aplicados de forma direta, outros são aplicados combinados com métodos mais tradicionais.

Liu et. al sugerem que o uso de deep learning para atividades de geração fora do contexto de jogo - como a geração de imagens, voz e música no campo de criatividade computacional -, e o aumento no uso de aprendizagem de máquina em PCG, são tendências que construíram uma revolução do próprio deep learning, tornando a aprendizagem de máquina efetiva em classes de problemas completamente novos. Isso resultou em um *boom* na área e os autores [28] acreditam que veremos um rápido progresso nesse campo de pesquisa no futuro.

4.1.2 PCG E IA

Técnicas de inteligência artificial têm tradicionalmente sido divididas em duas categorias: IA simbólica e IA conexionista. Curiosamente, a IA conexionista ganhou mais popularidade com as histórias de sucesso recentes e o *hype* criado pela mídia em cima dela, devido a sua aplicação em redes neurais [21].

4.1.2.1 I.A. SIMBÓLICA

Segundo Bajada [21], a IA simbólica segue uma abordagem mais clássica de codificar um modelo de um problema. Nessa categoria, os sistemas trabalham com raciocínio dedutivo, inferência lógica, e por vezes algoritmos de busca, que encontram uma solução dentro das restrições do modelo especificado [21]. Isso inclui sistemas que utilizem regras e árvores de decisão para deduzir conclusões com base nos dados de entrada. Esses tipos de algoritmos geralmente tem uma complexidade NP-difícil, lidando com espaços de busca massivos ao tentar resolver problemas do mundo real [21].

Acreditamos que esse tipo de abordagem se assemelhe a alguns métodos de PCG mostrados nesse trabalho que mantém uma forte estrutura lógica e matemática, como é o caso de L-systems, por exemplo. Por isso, é fácil de se imaginar que possa haver trabalhos que façam a junção desses dois métodos para a criação de algo novo.

4.1.2.2 I.A. CONEXIONISTA

A IA conexionista, ainda segundo Bajada [21], tem esse nome por conta da topologia mais comum de se encontrar nos algoritmos que fazem uso dela. A técnica mais popular dessa categoria de IA é a Rede Neural Artificial (ANN), sinônimo de Deep Learning, que consiste de múltiplas camadas de neurônios que processam sinais de entrada com coeficiência e pesos.

Máquinas de suporte de vetores (SVMs) também entram na categoria conexionista.

Observamos que esse tipo de abordagem já é menos comum de se encontrar em PCG - ou pelo menos nos métodos que estudamos durante essa pesquisa. Acreditamos que isso seja pelo fato de que essa abordagem demonstra ser mais complexa em sua execução, e exige que a máquina "pense"mais, sem se prender necessariamente à lógica. Muitos jogos tem demonstrado que uma IA inteligente é importante para eles - pois, isso significaria, por exemplo, que o jogo teria inimigos mais inteligentes, e por tanto, seria mais desafiador para o jogador.

REFERÊNCIAS BIBLIOGRÁFICAS

[1] ATTENBOROUGH, M. 20 - language theory. In *Mathematics for Electrical Engineering and Computing*, M. Attenborough, Ed. Newnes, Oxford, 2003, pp. 479–490.

[2] BALTEZAREVIC, R., BALTEZAREVIC, B., AND BALTEZAREVIC, V. The video gaming industry (from play to revenue). *International Review* (01 2018), 71–76.

[3] BARON, J. Procedural dungeon generation analysis and adaptation. In *Proceedings of the SouthEast Conference* (04 2017), pp. 168–171.

[4] BEN BORGERS. opensheet - a free, super simple, hosted api for getting google sheets as json. `https://github.com/benborgers/opensheet`, 2021. [Online; accessed 01-May-2022].

[5] BOOTH, MICHAEL. The ai systems of left 4 dead. `https://steamcdn-a.akamaihd.net/apps/valve/2009/ai_systems_of_l4d_mike_booth.pdf`, 2009. [Online; accessed 16-May-2022].

[6] BRACE YOURSELF GAMES DESCRIPTION. Crypt of the necrodancer on steam. `https://store.steampowered.com/app/247080/Crypt_of_the_NecroDancer/`, 2015. [Online; accessed 16-May-2022].

[7] CHEONG, Y.-G., RIEDL, M., BAE, B.-C., AND NELSON, M. *Planning with applications to quests and story*. Springer International Publishing, 10 2016, pp. 123–141.

[8] DE CASTRO, B. P., DA MOTA, R. R., AND FANTINI, E. P. C. Level design on rogue-like games: An analysis of crypt of the necrodancer and shattered planet. In *Proceedings of SBGames* (2017).

[9] DE PONTES, R. G., AND GOMES, H. M. Evolutionary procedural content generation for an endless platform game. In *2020 19th Brazilian Symposium on Computer Games and Digital Entertainment (SBGames)* (Nov 2020), pp. 80–89.

[10] DEBRY, D. G., GOFFIN, H., HECKER, C., QUIGLEY, O., SHODHAN, S., AND WILLMOTT, A. Player-driven procedural texturing. In *ACM SIGGRAPH 2007 Sketches* (New York, NY, USA, 2007), SIGGRAPH '07, Association for Computing Machinery, p. 81–es.

[11] DORAN, J., AND PARBERRY, I. Controlled procedural terrain generation using software agents. *Computational Intelligence and AI in Games, IEEE Transactions on 2* (07 2010), 111 – 119.

[12] DORMANS, J. Adventures in level design: Generating missions and spaces for action adventure games. In *Proceedings of the 2010 Workshop on Procedural Content Generation in Games* (New York, NY, USA, 2010), PCGames '10, Association for Computing Machinery.

[13] FABRICATORE, C. Gameplay and game mechanics design : A key to quality in videogames. In *OECD-CERI Expert Meeting on Videogames and Education* (2007).

[14] FONT, J. M., MAHLMANN, T., MANRIQUE, D., AND TOGELIUS, J. Towards the automatic generation of card games through grammar-guided genetic programming. In *International Conference on the Foundations of Digital Games* (Chania, Crete, Greece, May 14-17 2013), G. N. Yannakakis, E. Aarseth, K. Jørgensen, and J. C. Lester, Eds., Society for the Advancement of the Science of Digital Games, pp. 360–363.

[15] FRADE, M. M. D. S. Genetic terrain programming. Master thesis, Universidad de Extremadura, Estremadura, Spain, 06 2008. Dissertation presented at the University of Extremadura to obtain a Diploma of Advanced Studies, guided by Francisco Fernandéz de Vega and Carlos Cotta.

[16] FRAISSE, V., GUASTAVINO, C., AND WANDERLEY, M. M. A visualization tool to explore interactive sound installations. In *NIME 2021* (4 2021). https://nime.pubpub.org/pub/i1rx1t2e.

[17] GALLOWAY, A. R. *Gaming: Essays on Algorithmic Culture*, ned - new edition ed., vol. 18. University of Minnesota Press, 2006.

[18] GEMAN, S., AND JOHNSON, M. Probabilistic grammars and their applications. In *International Encyclopedia of the Social & Behavioral Sciences*, N. J. Smelser and P. B. Baltes, Eds. Pergamon, Oxford, 2001, pp. 12075–12082.

[19] GREENWOOD-ERICKSEN, ADAMS. Why left 4 dead works. `https://www.gamedeveloper.com/design/why-i-left-4-dead-i-works`, 2010. [Online; accessed 16-May-2022].

[20] JOHNSON, L., YANNAKAKIS, G., AND TOGELIUS, J. Cellular automata for real-time generation of infinite cave levels. *PCGames '10: Proceedings of the 2010 Workshop on Procedural Content Generation in Games* (09 2010).

[21] JOSEF BAJADA. Symbolic vs connectionist a.i. `https://towardsdatascience.com/symbolic-vs-connectionist-a-i-8cf6b656927`, 2019. [Online; accessed 16-May-2022].

[22] KASURINEN, J., STRANDÉN, J.-P., AND SMOLANDER, K. What do game developers expect from development and design tools? *ACM International Conference Proceeding Series* (04 2013).

[23] KELLY, G., AND MCCABE, H. A survey of procedural techniques for city generation. *The ITB Journal 7* (2006), 5.

[24] LAAMARTI, F., EID, M., AND EL SADDIK, A. An overview of serious games. *International Journal of Computer Games Technology 2014* (10 2014).

[25] LINDEN, R., LOPES, R., AND BIDARRA, R. Designing procedurally generated levels. In *Proceedings of IDPv2 2013 - Workshop on Artificial Intelligence in the Game Design Process, co-located with the Ninth AAAI Conference on Artificial Intelligence in Interactive Digital Entertainment* (10 2013), pp. 41–47.

[26] LINDEN, R., LOPES, R., AND BIDARRA, R. Procedural generation of dungeons. *Computational Intelligence and AI in Games, IEEE Transactions on 6* (03 2014), 78–89.

[27] LINDENMAYER, A. Mathematical models for cellular interactions in development i. filaments with one-sided inputs. *Journal of Theoretical Biology 18*, 3 (1968), 280–299.

[28] LIU, J., SNODGRASS, S., KHALIFA, A., RISI, S., YANNAKAKIS, G. N., AND TOGELIUS, J. Deep learning for procedural content generation. *CoRR abs/2010.04548* (2020).

[29] LUCAS, S., AND KENDALL, G. Evolutionary computation and games. *Computational Intelligence Magazine, IEEE 1* (03 2006), 10 – 18.

[30] MAN, K., TANG, K., AND KWONG, S. Genetic algorithms: concepts and applications [in engineering design]. *IEEE Transactions on Industrial Electronics 43*, 5 (1996), 519–534.

[31] MARKLUND, B., ENGSTRÖM, H., HELLKVIST, M., AND BACKLUND, P. What empirically based research tells us about game development. *The Computer Games Journal 8* (12 2019), 1–20.

[32] MAWHORTER, P., AND MATEAS, M. Procedural level generation using occupancy-regulated extension. In *Proceedings of the 2010 IEEE Conference on Computational Intelligence and Games, CIG 2010, Copenhagen, Denmark, 18-21 August, 2010* (08 2010), pp. 351–358.

[33] MEEHAN, J. R. The metanovel: Writing stories by computer. In *Outstanding Dissertations in the Computer Sciences* (1976).

[34] MENDLER, M., AND ADAMS, D. Automatic generation of dungeons for computer games. Bachelor thesis, University of Sheffield, Sheffield, South Yorkshire, England, 01 2002. Bachelor Thesis handed in as part of the final examination.

[35] MIKE BOSTOCK. D3 (data-driven documents or d3.js). `https://d3js.org/`, 2021. [Online; accessed 01-May-2022].

[36] NELSON, M. J., AND SMITH, A. M. *ASP with Applications to Mazes and Levels*. Springer International Publishing, Cham, 2016, pp. 143–157.

[37] NELSON, M. J., TOGELIUS, J., BROWNE, C., AND COOK, M. *Rules and Mechanics*. Springer International Publishing, Cham, 2016, pp. 99–121.

[38] OLIVEIRA, N., AND SEABRA, R. D. Towards a comprehensive classification for procedural content generation techniques. In *Proceedings of SBGames* (09 2016), pp. 166–169.

[39] OXSPRING, S., KIRMAN, B., AND SZYMANEZYK, O. Attack on the clones: Managing player perceptions of visual variety and believability in video game crowds. In *Advances in Computer Entertainment* (Cham, 2013), D. Reidsma, H. Katayose, and A. Nijholt, Eds., Springer International Publishing, pp. 356–367.

[40] PECH, A., HINGSTON, P., MASEK, M., AND LAM, C. P. Evolving cellular automata for maze generation. In *Artificial Life and Computational Intelligence* (Cham, 2015), S. K. Chalup, A. D. Blair, and M. Randall, Eds., Springer International Publishing, pp. 112–124.

[41] PELL, B. D. Metagame: A new challenge for games and learning. In *Programming in Artificial Intellegence: The Third Computer Olympiad. Ellis Horwood* (1992), Ellis Horwood Limited, pp. 237–251.

[42] RIEDL, M. O., AND YOUNG, R. M. Narrative planning: Balancing plot and character. *J. Artif. Int. Res. 39*, 1 (sep 2010), 217–268.

[43] RODRÍGUEZ PUENTE, R., PÉREZ BETANCOURT, Y., AND MUFETI, K. Cellular automata and its applications in modeling and simulating the evolution of diseases. In *National Research Symposium* (09 2015).

[44] SHAKER, N., LIAPIS, A., TOGELIUS, J., LOPES, R., AND BIDARRA, R. *Constructive generation methods for dungeons and levels*. Springer International Publishing, 10 2016, pp. 31–55.

[45] SHAKER, N., SMITH, G., AND YANNAKAKIS, G. N. *Evaluating content generators*. Springer International Publishing, Cham, 2016, pp. 215–224.

[46] SHAKER, N., TOGELIUS, J., AND NELSON, M. *Fractals, noise and agents with applications to landscapes*. Springer International Publishing, 10 2016, pp. 57–72.

[47] SHAKER, N., TOGELIUS, J., AND NELSON, M. J. *Procedural content generation in games*. Springer, 2016.

[48] SMITH, G., TREANOR, M., WHITEHEAD, J., AND MATEAS, M. Rhythm-based level generation for 2d platformers. In *Proceedings of the 4th International Conference on Foundations of Digital Games* (New York, NY, USA, 2009), FDG '09, Association for Computing Machinery, p. 175–182.

[49] TOGELIUS, J., CHAMPANDARD, A., LANZI, P. L., MATEAS, M., PAIVA, A., PREUSS, M., AND STANLEY, K. Procedural content generation: goals, challenges and actionable steps. *Dagstuhl Follow-Ups 6* (01 2013), 61–75.

[50] TOGELIUS, J., KASTBJERG, E., SCHEDL, D., AND YANNAKAKIS, G. N. What is procedural content generation? mario on the borderline. In *Proceedings of the 2nd International Workshop on Procedural Content Generation in Games* (New York, NY, USA, 2011), PCGames '11, Association for Computing Machinery.

[51] TOGELIUS, J., PREUSS, M., AND YANNAKAKIS, G. N. Towards multiobjective procedural map generation. In *Proceedings of the 2010 Workshop on Procedural Content Generation in Games* (New York, NY, USA, 2010), PCGames '10, Association for Computing Machinery.

[52] TOGELIUS, J., SHAKER, N., AND DORMANS, J. *Grammars and L-systems with applications to vegetation and levels*. Springer International Publishing, Cham, 2016, pp. 73–98.

[53] TOGELIUS, J., YANNAKAKIS, G. N., STANLEY, K. O., AND BROWNE, C. Search-based procedural content generation. In *Proceedings of the 2010 International Conference on Applications of Evolutionary Computation - Volume Part I* (Berlin, Heidelberg, 2010), EvoApplicatons'10, Springer-Verlag, p. 141–150.