Design de Jogos Eletrônicos de Quebra-Cabeças usando Geração Procedural

Danilo dos Santos Bezerra, Francisco Isidro Massetto e Rafaela Vilela da Rocha
Centro de Matemática, Computação e Cognição
Universidade Federal do ABC, Santo André, SP, Brasil
Email: {danilo.bezerra, francisco.massetto, rafaela.rocha}@ufabc.edu.br

***Resumo*—Jogos de quebra-cabeças costumam ser bastante divertidos, tal gênero é comum em dispositivos móveis que é a plataforma que detém mais da metade do mercado. A geração procedural é um artifício interessante para construção de jogos eletrônicos, pois possibilita a criação de grande quantidade de conteúdo com poucos recursos. Por isso, a utilização de geração procedural pode tornar o desenvolvimento de jogos eletrônicos do gênero quebra-cabeça mais rápido e fácil. Este trabalho contém a descrição de um protótipo que usa geração procedural e termina com uma discussão sobre possíveis melhorias para potencializar seus benefícios.**

***Keywords*-design de jogos; quebra-cabeça; geração procedural**

I. Introdução

Atualmente, de acordo com a pesquisa [1], os jogos eletrônicos crescem 25,5% ao ano nos dispositivos móveis, atingindo a marca de 70,3 bilhões de dólares. Isso é metade da receita total desse mercado, que inclui computadores e consoles. Novak [2] observa que plataformas portáteis como smartphones e *tablets* são ideais para jogos casuais do gênero quebra-cabeça.

O crescimento desse mercado e a exigência por qualidade têm exaurido cada vez mais a linha de produção. Além disso, nem sempre as equipes são compostas de membros suficientes para suprir todas as áreas necessárias na criação de um projeto de jogo, que muitas vezes são desenvolvidos por poucas pessoas.

Sendo assim, se faz cada vez mais necessário utilizar de técnicas e artifícios para otimizar o processo de produção dos jogos, prescindindo-se da criação de conteúdo manualmente e optando-se por um procedimento que o gere automaticamente, segundo a filosofia da modelagem procedural [3]. De acordo com Hendrikx et al. [4], técnicas procedurais são uma alternativa para criar mundos de jogos complexos em uma quantidade de tempo limitada e sem colocar um grande fardo nos *designers* de conteúdo de jogo.

Este trabalho visa relatar a criação de um protótipo de um jogo eletrônico do gênero quebra-cabeça, baseado em jogos como *Puzzle Ball*, conforme descrito na seção III. O sistema de geração automática no protótipo tem o objetivo de economizar recursos mantendo o engajamento dos jogadores. Para tal intuito, a solução proposta será construída a partir de uma taxonomia que é abordada na seção II. A seção apresenta IV o teste e validação do protótipo, seguida das considerações finais (na Seção V).

II. Revisão da bibliografia

A. Jogos eletrônicos do gênero quebra-cabeça

O gênero de jogos eletrônicos quebra-cabeça, também conhecido como *puzzle*, é consolidado e bastante aceito no mercado. Rabin [5] reflete que na indústria há um longo debate sobre os quebra-cabeças serem jogos ou não, para o autor eles certamente são o suficiente para justificar um gênero baseado neles.

> Os *puzzles* se constituem um tipo de enigma ou problema cuja finalidade está em desenvolver o raciocínio, lógico e matemático, em seus níveis ontológicos, cognitivos, como força motriz para as mais diversas formas de produção de conhecimentos, como citamos os *puzzles* podem estar presentes em games com diferentes temas e objetivos, seja na matemática ou qualquer outra ciência. [6].

Jogos desse gênero consistem em solucionar quebra-cabeças ou enigmas, Novak [2] afirma que nos quebra-cabeças a narrativa é mínima ou inexistente e que eles geralmente funcionam de maneira temporizada, baseada em turnos ou em tempo real. Entre os jogos mais conhecidos do gênero podemos citar o famoso *Tetris* (1984), na figura 1, que tem como finalidade empilhar tetraminós na tela.

Figura 1. Versão do jogo *Tetris* para IBM PC no ano 1986.

O gênero é conhecido por ter mecânicas geralmente bem simples e que prendem o jogador pelos desafios envolvidos na sua resolução, para Rabin [5] os jogos de quebra-cabeça são um desafio para seu desenvolvedor, porque é

importante oferecer obstáculos e controle a quem estiver jogando. Um *puzzle* que não exiba nenhum tipo de parecer sobre seu progresso pode ser desencorajador ao jogador e uma possível solução para o problema, por exemplo, seria revelar os passos aos poucos através das tentativas e erros dos jogadores.

B. Geração Procedural de Conteúdo

A Geração Procedural de Conteúdo (GPC) é a criação de conteúdo para jogos a partir de poucos parâmetros que permitem um grande e imprevisível número de possibilidades, usando um processo aleatório ou pseudoaleatório. Togelius et al. [7] tenta definir GPC como: "a criação algorítmica do conteúdo do jogo com entrada de usuário limitada ou indireta", ou seja, baseia-se em substituir o trabalho de gerar texturas, sons e fases de jogos pela construção de sistemas responsáveis por essa tarefa.

No início dos anos 1980 esse método era amplamente utilizado no desenvolvimento de jogos, principalmente para lidar com as limitações dos computadores da época. Usando pouca memória *Elite*, na figura 2, gerava oito galáxias cada uma contendo 256 planetas com características próprias. O clássico *Rogue*, na figura 3, jogo de aventura que deu origem ao subgênero de jogos *Role Playing Game* - RPG chamado *Roguelike* construía níveis aleatórios a partir de salas interligadas por túneis.

Figura 2. Versão do jogo *Elite* para BBC Micro no ano 1984.

Figura 3. Versão ASCII do jogo *Rogue* no ano 1980.

Com o passar dos anos a GPC tornou-se cada vez mais presente nos jogos comerciais. Ao invés de contornar limitações como nos exemplos anteriores o jogo *Spelunky*, na figura 4, usa a GPC para criar variações dos cenários oferecendo ao jogador um conjunto de 16 níveis divertidos e atrativos completamente diferentes a cada partida [8], combinando o gênero plataforma com o subgênero *roguelike*.

Figura 4. Versão original *freeware* do jogo *Spelunky*.

Conforme discutido anteriormente a GPC constitui-se de diversas formas e pode ser utilizada para resolver diversos problemas, por essa razão, Togelius et al. [9] estabelece uma taxonomia para classificar as diferentes abordagens da GPC nas suas distinções mais comuns, a seguir.

1) Online ou offline: Diz respeito ao conteúdo ser gerado durante a execução do jogo ou durante seu desenvolvimento através de uma ferramenta.

2) Conteúdo necessário ou opcional: O conteúdo gerado afeta diretamente a progressão do jogador ou serve apenas de suporte para as mecânicas presentes no jogo.

3) Seeds aleatórios ou vetores de parâmetros: Refere-se ao grau de controle, se o conteúdo será gerado aleatoriamente por um *seed* apenas ou a partir de parâmetros pré-definidos que irão definir suas possíveis propriedades.

4) Geração estocástica ou determinística: Dado um determinado valor de entrada, será gerado um conteúdo a cada vez que o algoritmo for executado ou o resultado sempre será o mesmo.

5) Construtivo ou gerar-e-testar: O algoritmo será encarregado de construir o conteúdo enquanto, ao mesmo tempo, realiza as validações necessárias ou se terá um mecanismo responsável por testar a validade do resultado após sua construção.

III. Desenvolvimento

O protótipo a ser descrito nesse trabalho utilizará as seguintes abordagens, de acordo com a taxonomia citada anteriormente: *online*, *conteúdo necessário*, *vetores de parâmetros*, *determinística* e *construtiva*. Ou seja, o conteúdo será gerado durante execução, afetará diretamente a progressão do jogador, será construído a partir de parâmetros pré-definidos, dados os mesmos valores o resultado sempre será o mesmo e será validado durante a sua própria construção.

Trata-se de um jogo eletrônico do gênero quebra-cabeça, cuja condição de vitória consiste em conectar os blocos de origem e destino, que serão identificados neste protótipo pelas cores verde e vermelho respectivamente. Esses blocos coloridos são fixos no tabuleiro, enquanto os brancos de conexão podem somente ser movimentados em direção aos espaços vazios identificados pela cor preta. A figura 5 ilustra como serão representadas as características dos blocos que estão distribuídos pelo tabuleiro do jogo.

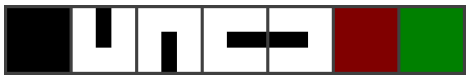

Figura 5. Representações das características dos blocos. Da esquerda para a direita: espaço vazio, conexões para cima, baixo, lados esquerdo e direito, destino e origem. É importante observar que os blocos de conexão serão sobrepostos para formar mais de uma saída.

Os blocos de conexão precisam conectar-se estritamente com outros dois que estejam localizados em suas adjacências e que tenham características complementares a suas próprias. Por exemplo, um bloco que tenha saídas para cima e esquerda pode conectar-se com outros que imediatamente nas posições esperadas tenham conexões para baixo e direita respectivamente, conforme ilustrado na figura 6.

Figura 6. Sob o canto inferior direito, o bloco tem saídas para cima e esquerda, conectando-se com os blocos adjacentes posicionados imediatamente acima e a sua esquerda.

Um *designer* de jogos poderia facilmente construir cada fase de um jogo dessa espécie em qualquer motor da atualidade, modelando e definindo as características de cada bloco manualmente, porém, isso rapidamente pode se tornar um trabalho oneroso em função da quantidade de fases com qualidade e unicidade. Uma possível solução para esse problema é delegar as tarefas de construção do tabuleiro e definição dos blocos para um algoritmo, que será responsável por construir as fases a partir de valores atribuídos previamente.

De modo a cumprir esse objetivo, o tabuleiro será gerado proceduralmente a partir de uma matriz quadrada $M_{4\times4}$, composta de números inteiros. Cada elemento da matriz corresponde aos valores que irão definir as características de cada bloco que será instanciado no tabuleiro. Valores de números inteiros por si só não são suficientes para armazenar as informações necessárias, então para contornar essa limitação, será utilizada uma enumeração com potências de dois para representar as características de um bloco de acordo com a tabela I. Serão realizadas operações simples de lógica binária com os bits armazenados nos seus valores.

Tabela I
ENUMERAÇÃO DAS CARACTERÍSTICAS COM SEUS VALORES EM POTÊNCIAS DE DOIS E SUAS RESPECTIVAS REPRESENTAÇÕES EM BITS

Características	Valor	Bits
None	0	000000
Top	1	000001
Bottom	2	000010
Right	4	000100
Left	8	001000
Target	16	010000
Source	32	100000

A. *Deslocamento bit a bit*

A razão para a utilização de potências de dois como valores da enumeração é por causa das suas representações binárias, os números inteiros 2, 4 e 8, por exemplo, equivalem respectivamente a 10, 100 e 1000 quando representados em bits, ou seja, consistem em sucessivos deslocamentos para a esquerda sobre o número inteiro 1.

B. *Disjunção binária*

Então para efetivamente atribuir mais de uma característica para um bloco é preciso realizar uma operação lógica de disjunção, ou OU, sobre os valores inteiros da enumeração mostrada na tabela I, por exemplo, a operação $2 \vee 4 = 6$ combina as características *Bottom* e *Right*, representadas em bits como 110. Os elementos da matriz $M_{4\times4}$ correspondem ao resultado de disjunções sobre os valores da enumeração, ou seja, das características do bloco.

C. *Conjunção binária*

O algoritmo gerador precisa de uma condição para identificar quais características devem ser modeladas no momento da criação do bloco, sendo necessário realizar a operação inversa da qual foi usada para atribuir os elementos da matriz $M_{4\times4}$. Será necessária uma operação lógica de conjunção, ou E, para extrair as características atribuídas, por exemplo, para identificar se o valor 6 contém a característica *Bottom* verificamos que a condição $6 \wedge 2 \neq 0$ é verdadeira.

IV. TESTE E VALIDAÇÃO

O protótipo utilizado como referência neste trabalho foi desenvolvido com o motor de jogos *Unity*, por ser uma ferramenta acessível e bastante indicada para rápida prototipagem, mas também pode ser criado em qualquer outro motor ou biblioteca que tenha suporte a jogos 2D e leitura de arquivos. A matriz de números inteiros será armazenada em um arquivo texto que será lido a partir de uma requisição *web*, dessa forma, os arquivos poderão ser hospedados no local em que for mais conveniente. Para maior praticidade, os arquivos textos utilizados no protótipo serão hospedados

localmente. A seguir, os elementos de uma possível matriz pertencente a um dos arquivos texto:

$$M_{4\times4} = \begin{bmatrix} 3 & 6 & 10 & 12 \\ 3 & 5 & 5 & 24 \\ 36 & 10 & 9 & 3 \\ 12 & 5 & 0 & 3 \end{bmatrix}$$

Os elementos da matriz $M_{4\times4}$ são os valores de números inteiros que serão convertidos em características para os blocos presentes no tabuleiro gerado, de acordo com a tabela II, o algoritmo usará as respectivas representações em bits desses valores.

Tabela II
CARACTERÍSTICAS E REPRESENTAÇÕES EM BITS CORRESPONDENTES AOS VALORES DA MATRIZ EXEMPLO

Características	Valor	Bits
None	0	000000
Top, Bottom	3	000011
Top, Right	5	000101
Bottom, Right	6	000110
Top, Left	9	001001
Bottom, Left	10	001010
Right, Left	12	001100
Left, Target	24	011000
Right, Source	36	100100

Após a leitura e identificação dos valores, resta apenas gerar proceduralmente o tabuleiro a partir das informações obtidas. Finalmente, a figura 7 mostra o resultado do processamento da matriz $M_{4\times4}$, após todos os blocos serem devidamente instanciados.

Apesar de atingir o resultado esperado com o protótipo, alguns pontos não puderam ser aprofundados neste trabalho, por exemplo, ainda é necessária intervenção humana na criação das fases, porque atualmente o designer precisa atribuir manualmente os elementos da matriz $M_{4\times4}$ que servirá como entrada para o algoritmo. Duas possíveis alternativas para avançar nesse protótipo são: (1) desenvolver uma ferramenta que torne próximo de trivial a construção das matrizes ou (2) desenvolver um mecanismo para gerar as matrizes automaticamente e de forma aleatória.

V. CONSIDERAÇÕES FINAIS

A construção do protótipo se manteve alinhada ao escopo definido inicialmente. Deste modo, foi possível alcançar o resultado desejado ao desenvolver um protótipo de jogo do gênero quebra-cabeça usando geração procedural com o intuito de economizar tempo e recursos.

Considerando o que foi feito até o momento pode-se dizer que, apesar da dificuldade, desenvolver sistemas de geração procedural tem um ótimo custo-benefício e pode ser um ótimo investimento que a médio e longo prazo renderá em bastante conteúdo para o jogo.

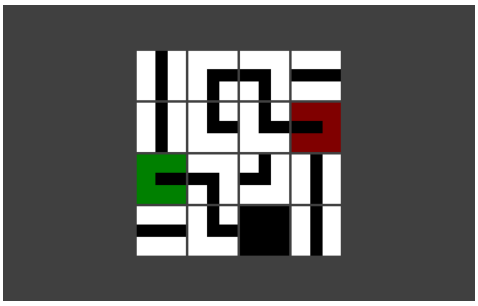

Figura 7. Tabuleiro gerado proceduralmente a partir da matriz de números inteiros que está armazenada em um arquivo texto.

REFERÊNCIAS

[1] T. Wijman. (2018) Mobile revenues account for more than 50% of the global games market as it reaches $137.9 billion in 2018. [Online]. Available: https://newzoo.com/insights/articles/global-games-market-reaches-137-9-billion-in-2018-mobile-games-take-half/

[2] J. Novak, *Desenvolvimento de games*. Cengage Learning, 2010.

[3] R. M. Smelik, K. J. De Kraker, T. Tutenel, R. Bidarra, and S. A. Groenewegen, "A survey of procedural methods for terrain modelling," in *Proceedings of the CASA Workshop on 3D Advanced Media In Gaming And Simulation (3AMIGAS)*, 2009, pp. 25–34.

[4] M. Hendrikx, S. Meijer, J. Van Der Velden, and A. Iosup, "Procedural content generation for games: A survey," *ACM Transactions on Multimedia Computing, Communications, and Applications (TOMM)*, vol. 9, no. 1, p. 1, 2013.

[5] S. Rabin, *Introdução ao desenvolvimento de games: vol. 1: entendendo o universo dos jogos*. Cengage Learning, 2011.

[6] C. N. Tonéis, "O design de puzzles nos jogos digitais," *Proceedings of SBGames 2016: Art & Design Track – Full Papers*, pp. 404–411, 2016.

[7] J. Togelius, E. Kastbjerg, D. Schedl, and G. N. Yannakakis, "What is procedural content generation?: Mario on the borderline," in *Proceedings of the 2nd International Workshop on Procedural Content Generation in Games*. ACM, 2011, p. 3.

[8] D. Yu, *Spelunky (Boss Fight Books Book 11)*. Boss Fight Books, 2016.

[9] J. Togelius, G. N. Yannakakis, K. O. Stanley, and C. Browne, "Search-based procedural content generation: A taxonomy and survey," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 3, no. 3, pp. 172–186, 2011.