

# CONIC SEMESP

15º Congresso Nacional de Iniciação Científica

**TÍTULO:** GERAÇÃO PROCEDURAL COMPOSICIONAL PARA A GERAÇÃO DINÂMICA DE FASES

**CATEGORIA:** CONCLUÍDO

**ÁREA:** CIÊNCIAS EXATAS E DA TERRA

**SUBÁREA:** COMPUTAÇÃO E INFORMÁTICA

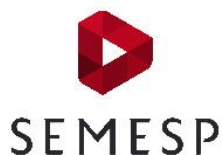
**INSTITUIÇÃO:** CENTRO UNIVERSITÁRIO FILADÉLFIA

**AUTOR(ES):** LUIZ ANTONIO LIMA RODRIGUES

**ORIENTADOR(ES):** MARIO HENRIQUE AKIHIKO DA COSTA ADANIYA, RICARDO INACIO ALVARES E SILVA

**COLABORADOR(ES):** PEDRO R. E. GESTAL, RODOLFO M. BARROS

Realização:



Apoio:



## 1 RESUMO

Geração Procedural de Conteúdos é uma técnica aplicada para automatização de conteúdos em diversas áreas. Este trabalho apresenta um novo algoritmo unindo Evolução Gramatical (EG) e Agentes de Software (AS) para a geração dinâmica. EG faz um mapeamento entre os símbolos gramaticais e os elementos do jogo, e cada elemento possui um AS que posiciona o elemento no ambiente. Uma versão de um jogo é utilizado para demonstração do algoritmo, gerando suas fases em tempo de execução e inéditas a cada nova geração de fase. O gênero do jogo é um *rogue-like* com gráficos 2D, onde o jogador deve sobreviver o máximo de dias possíveis, no qual cada dia representa uma fase gerada pelo framework. A técnica apresentada alcançou de forma satisfatória um grau de expressividade, e os resultados são discutidos a partir da análise utilizando métricas propostas na literatura: *exploração*; *controle de área* e; *controle estratégico de recursos*. A implementação buscou ser desenvolvida de forma genérica, permitindo a sua utilização em outros tipos de jogos.

## 2 INTRODUÇÃO

Com a evolução das tecnologias, o entretenimento digital também foi afetado necessitando uma criação de conteúdos novos constantemente. A geração de conteúdos é na maioria das vezes uma tarefa que demanda tempo, dinheiro e mão de obra, sendo responsável por até 40% orçamento de alguns jogos (HENDRIKX et al., 2013). Conteúdo em menor escala seriam os itens, objetos, personagens jogáveis e não-jogáveis, e em escala maior são as fases, níveis, mundos e o conjunto de ações.

A Geração Procedural de Conteúdo (GPC) é uma técnica capaz de gerar conteúdo através de procedimentos. Métodos de GPC podem ser aplicados em diversas áreas, como na geração de terrenos (DORAN; PARBERRY, 2010), cidades (KELLY; MCCABE, 2006), jogos (CARLI et al., 2011) para citar alguns exemplos.

Um Sistema de GPC (S-GPC) é um software que gera conteúdo através de um método de GPC. O uso de um S-GPC permite a redução do tempo gasto durante o desenvolvimento, além de custos financeiros. Também aumenta a variância do conteúdo gerado e permite sua adaptação através de parâmetros, devido às características da GPC.

## 3 OBJETIVOS

Este trabalho tem como objetivos: (1) criar uma framework que auxilie o desenvolvimento de jogos eletrônicos, através da (2) aplicação de uma técnica de GPC;

(3) aplicar o S-GPC desenvolvido na geração das fases<sup>1</sup>; (4) avaliar qualitativamente de acordo com métricas propostas para analisar padrões de jogos e sua expressividade.

## 4 METODOLOGIA

Para criar o algoritmo de GPC, a proposta apresentada em (TOGELIUS; JUSTINUSSEN; HARTZEN, 2012) foi utilizada como paradigma. Onde os autores propõe que métodos de GPC sejam unidos, criando um novo algoritmo. Com isto, foram utilizados dois algoritmos como base, Evolução Gramatical (EG) (O'NEILL; RYAN, 2001; SHAKER et al., 2012) e Agentes de Software (AS) (RUSSELL; NORVIG, 2003; DORAN; PARBERRY, 2010).

A ferramenta desenvolvida foi implementada na linguagem C Sharp (C#), com a IDE Visual Studio 2013. Devido ao suporte da Unity *engine* à linguagem C#, a interface utilizada na comunicação entre ela e o framework criado foi feita utilizando ferramentas desenvolvidas com as mesma características. Entretanto, o conteúdo gerado pela ferramenta pode ser utilizado por outras ferramentas, uma vez que foi projetada para ser independente de qualquer *engine*.

Os testes realizados no S-GPC desenvolvido foram inspirados em (SMITH; WHITEHEAD, 2010; LIAPIS; YANNAKAKIS; TOGELIUS, 2013). Em (SMITH; WHITEHEAD, 2010) os autores propõem que um gerador (S-GPC) não deve ser avaliado, principalmente, pelo número de saídas capaz de gerar ou tempo de geração, mas pela multiplicidade destas, além da variação obtida através da mudança de parâmetros. As métricas adotadas para esta avaliação foram escolhidas de acordo com (LIAPIS; YANNAKAKIS; TOGELIUS, 2013), onde os autores apresentam formas para a avaliação de fases em geral.

## 5 DESENVOLVIMENTO

Nesta seção é discutido os algoritmos para o desenvolvimento do jogo e implementação do framework. As regras e símbolos utilizadas pela gramática, mapeamento realizado de elementos e seus agentes.

### 5.1 Algoritmo e Framework

Durante a geração das fases, o algoritmo trabalha com a criação em três fases. Na primeira, a EG define quais elementos estarão presentes na fase em criação. Em seguida, cada um desses elementos serão posicionados na fase por seus respectivos

<sup>1</sup> O jogo utilizado é um tutorial desenvolvido pela Unity (<https://unity3d.com/pt/learn/tutorials/projects/2d-roguelike-tutorial>)

agentes. Finalmente, uma busca é feita na representação da fase gerada, a fim de detectar se esta é jogável ou não.

Esta abordagem busca maximizar a qualidade da saída gerada através da decomposição do algoritmo, onde cada um dos métodos usados é responsável por uma parte específica do problema. Entretanto, em compensação tem-se um aumento na complexidade para que ambas trabalhem juntas em prol de um resultado final.

A EG foi utilizada na definição dos elementos devido à estrutura proporcionada pela gramática, permitindo uma variedade de conjuntos de elementos (palavras) a serem escolhidos. Além disso, a EG permite a definição de uma função de avaliação (FA), responsável pela avaliação de saídas candidatas. Desta maneira, pode-se obter diversas palavras, que se aproximem ao máximo de um valor delimitado pela FA.

Todos os elementos que podem estar em uma fase, possuem um agente vinculado a si, o qual é responsável pela sua criação na representação da fase. Estes agentes são funções, que agem (posicionam) em um ambiente (representação da fase) através de sensores (elementos). Ainda assim, é possível que um mesmo agente aja de maneira diferente, uma vez que o framework desenvolvido permite a configuração de cada agente através de parâmetros.

A implementação do algoritmo desenvolvido na ferramenta criada foi feita de maneira genérica, desta maneira, é possível aplicá-la na geração de outros tipos de jogos. Implementou-se um módulo para jogos que possuem *tilemaps* como representação de fases.

## 5.2 Desenvolvimento do jogo

O jogo adotado é um tutorial da Unity, onde eles desenvolveram um *roguelike* em 2D utilizando *tilemaps*. Neste jogo o usuário é um sobrevivente em um apocalipse zumbi, e seu objetivo é sobreviver o maior número de dias possíveis. Cada dia é uma fase diferente, onde o usuário parte de um extremo dela (inferior esquerdo) e precisa chegar até o outro (superior direito) sem que seus mantimentos acabem. Durante o caminho deve-se coletar o maior número de mantimentos (refrigerante e maçãs, originalmente). Além disso, o jogador deve evitar que fique bloqueado por pedras ou que necessite cortar galhos, devido à constante perseguição de zumbis.

Para aplicação do algoritmo em um jogo devem ser realizadas as definições de configuração: da gramática; função de avaliação; agentes de cada um dos elementos e; função de avaliação da jogabilidade da fase. Note que os símbolos da gramática serão os elementos do jogo, que por sua vez, devem ser vinculados a um agente. A Figura 1 apresenta as produções da gramática desenvolvida para este jogo. Note que não foi optado por uma gramática pouco restrita, devido ao estilo do jogo.

A Tabela 1 demonstra o mapeamento realizado de elementos, que são os símbolos terminais da gramática, para seus respectivos agentes.

$$\begin{aligned}
P = \{ & \\
& 1. \langle S \rangle ::= \langle \text{init} \rangle \langle SD \rangle \langle W2 \rangle \langle EE \rangle \langle FD \rangle \langle BW \rangle \langle O1 \rangle \langle W1 \rangle \langle RE \rangle \langle O2 \rangle \\
& 2. \langle SD \rangle ::= \langle \text{sd} \rangle \mid \langle \text{sd} \rangle \langle EE \rangle \mid \langle \text{sd} \rangle \langle W1 \rangle \mid \langle \text{sd} \rangle \langle O2 \rangle \mid \langle \text{sd} \rangle \langle W2 \rangle \mid \langle \text{sd} \rangle \langle O1 \rangle \\
& 3. \langle W2 \rangle ::= \langle \text{w2} \rangle \mid \langle \text{w2} \rangle \langle FD \rangle \mid \langle \text{w2} \rangle \langle SD \rangle \mid \langle \text{w2} \rangle \langle O2 \rangle \mid \langle \text{w2} \rangle \langle BW \rangle \mid \\
& \quad \langle \text{w2} \rangle \langle RE \rangle \\
& 4. \langle EE \rangle ::= \langle \text{ee} \rangle \mid \langle SD \rangle \langle FD \rangle \mid \langle \text{ee} \rangle \langle O1 \rangle \langle W1 \rangle \\
& 5. \langle FD \rangle ::= \langle \text{fd} \rangle \mid \langle \text{fd} \rangle \langle W1 \rangle \mid \langle \text{fd} \rangle \langle EE \rangle \mid \langle \text{fd} \rangle \langle BW \rangle \mid \langle \text{fd} \rangle \langle O2 \rangle \mid \langle \text{fd} \rangle \langle W2 \rangle \\
& \quad \mid \langle \text{fd} \rangle \langle O1 \rangle \\
& 6. \langle BW \rangle ::= \langle \text{bw} \rangle \mid \langle \text{bw} \rangle \langle FD \rangle \langle SD \rangle \mid \langle \text{bw} \rangle \langle O2 \rangle \mid \langle \text{bw} \rangle \langle W2 \rangle \mid \langle \text{bw} \rangle \langle RE \rangle \\
& 7. \langle O1 \rangle ::= \langle \text{o1} \rangle \mid \langle \text{o1} \rangle \langle BW \rangle \mid \langle \text{o1} \rangle \langle W1 \rangle \mid \langle \text{o1} \rangle \langle FD \rangle \mid \langle \text{o1} \rangle \langle SD \rangle \mid \\
& \quad \langle \text{o1} \rangle \langle W2 \rangle \mid \langle \text{o1} \rangle \langle RE \rangle \\
& 8. \langle W1 \rangle ::= \langle \text{w1} \rangle \mid \langle \text{w1} \rangle \langle EE \rangle \mid \langle \text{w1} \rangle \langle FD \rangle \mid \langle \text{w1} \rangle \langle W2 \rangle \langle O1 \rangle \mid \langle \text{w1} \rangle \langle SD \rangle \mid \\
& \quad \langle \text{w1} \rangle \langle BW \rangle \\
& 9. \langle RE \rangle ::= \langle \text{re} \rangle \mid \langle \text{re} \rangle \langle W1 \rangle \langle W2 \rangle \mid \langle \text{re} \rangle \langle SD \rangle \langle FD \rangle \mid \langle \text{re} \rangle \langle O2 \rangle \langle O1 \rangle \\
& 10. \langle O2 \rangle ::= \langle \text{o2} \rangle \mid \langle \text{o2} \rangle \langle SD \rangle \mid \langle \text{o2} \rangle \langle EE \rangle \mid \langle \text{o2} \rangle \langle FD \rangle \mid \langle \text{o2} \rangle \langle BW \rangle \mid \\
& \quad \langle \text{o2} \rangle \langle W1 \rangle \\
& \}
\end{aligned}$$

Figura 1 – A definição da gramática utilizada na geração dos elementos que podem existir neste tipo de fase.

A fórmula utilizada como FA pode ser vista em (1) de maneira simplificada, onde  $x$  representa a fase em avaliação,  $w$  uma lista de pesos para cada elemento e pertencente ao conjunto de símbolos terminais  $T$ .

$$f(x) = \sum_{e \in T} w_e * Count(e) \quad (1)$$

A nova versão do jogo, contém dois modos de jogo: estático e procedural. No primeiro, o usuário joga por 15 fases, pré-definidas e geradas previamente. No modo procedural, o usuário joga fases inéditas, geradas enquanto o jogo esta sendo executado, após cada dia de sobrevivência ou reinício. Em ambos, a primeira fase (dia

Tabela 1 – Mapeamento de elementos para agentes

Elemento	Agente	Descrição
init	inicializador	cria o ponto de entrada e saída de acordo com uma distância definida por parâmetros.
bw	expansor de parede	cria blocos, horizontais ou verticais a partir alguma das paredes.
o1	gerador de blocos	cria blocos aproximadamente quadrados de tamanhos variados
ee	gerador de proximidade	cria o elemento à uma distância $x$ de um elemento $y$
sd, fd, w1, w2, o2, re	aleatório	posiciona o elemento de acordo com uma distribuição uniforme

0) é igual, para que o mesmo apenas reconheça o ambiente oferecido pelo jogo.

## 6 RESULTADOS

A fim de demonstrar os resultados obtidos, na geração das fases do jogo através do S-GPC aplicado, foram utilizadas três métricas extraídas de (LIAPIS; YAN-NAKAKIS; TOGELIUS, 2013): *exploração*; *controle de área* e; *controle estratégico de recursos* (CER). As funções de *exploração* e *controle de área* utilizam um conjunto de referências, enquanto o CER utiliza duas. Estas referências são conjuntos de *tiles* especiais à um tipo de fase(s) em específico. Para este jogo, foram definidos três grupos de referências: saídas, onde estarão entrada e saída de uma fase; mantimentos, conjunto contendo objetos consumíveis de uma fase, e; inimigos, onde estarão as posições iniciais de todos zumbis.

**Exploração** é uma função que para cada elemento  $l$  de seu conjunto de *tiles*  $N$ , calcula uma média do fator de exploração de  $l$  para todos elementos de  $N$  diferentes de  $l$ . Isto implica que quanto maior o espaço existente entre os elementos de  $N$ , em determinada fase, maior será sua exploração.

**Controle de Área** tem seu valor definido pela soma da cobertura de todos elementos de  $N$ , ponderado pelo número de posições não bloqueadas da fase. A cobertura de um elemento  $l$  é definida pela soma de todos *tiles*  $T$  contendo *segurança* menor que um valor constante  $C$ , definido como 0.5. Esta *segurança* de  $T$  para  $l$  é a menor distância entre  $l$  e todos elementos de  $N$ , normalizada pela diferença entre eles, ou zero. Desta maneira, o valor de controle é maior conforme o número de *tiles* passáveis é menor. A distribuição dos elementos de  $N$  também aumenta este valor, conforme estes estão posicionados à diferentes distâncias uns dos outros.

**CER** soma a maior *segurança* de todos elementos de  $M$  com respeito aos elementos de  $N$ , ponderando de acordo com o número de elementos em  $M$ . Este valor será maior conforme o posicionamento dos elementos de  $N$  contenham distâncias

similares, em relação aos elementos de  $N$ . Note que, caso algum dos conjuntos  $N$  ou  $M$  tenha tamanho menor que dois, este valor será insignificante.

Segundo a proposta apresentada em (SMITH; WHITEHEAD, 2010), a análise da expressividade de um gerador procedural deve: determinar métricas apropriadas para o conteúdo em avaliação; coletar uma amostra significativa do conteúdo gerado, pontuando cada entidade de acordo com as métricas definidas anteriormente; criar uma visualização da pontuação obtida pela amostra e; analisar o impacto da mudança de parâmetros na pontuação obtida.

As métricas escolhidas foram: *exploração* entre os *tiles* de saída; *CER*, dados inimigos e mantimentos como estes recursos e; *controle de área* entre a união dos conjuntos de saída e inimigos. O primeiro, foi escolhido com intuito de avaliar a exploração das fases, proporcionada pelo algoritmo, entre a entrada e o objetivo desta fase. A segunda métrica tem como objetivo avaliar o posicionamento de inimigos próximos aos mantimentos. Finalmente, a partir da união de zumbis e saídas, pode-se avaliar a distribuição obtida pelo posicionamento dos mesmos.

Foram geradas 10000 fases em duas etapas. Por simplicidade, foi-se definido um unico parâmetro para a FA utilizada geração destas fases, sendo este um limiar que pode variar entre 0 e 1. Então, foram geradas 10000 fases por etapa, onde foram usado limiares de 0,4 e 0,8 respectivamente. Além disso, todas as fases tiveram seus valores salvos e normalizados entre 0 e 1, através da diferença entre o valor mínimo e máximo de cada uma das métricas.

A Figura 2 demonstra, a partir das métricas de *exploração* e *CER*, a variação obtida através da alteração do parâmetro. Note que, com o limiar baixo, a maioria das fases contém um alto nível de exploração. Além disso, o controle entre inimigos e mantimentos teve sua maior concentração em valores baixos. Isto é, devido à maioria das fases conterem menos que dois elementos de cada, além de, na maioria das vezes, estes estarem localizados à uma distância semelhante. Devido à característica do jogo, na maioria das vezes isso facilita a jogabilidade, uma vez que todos estejam próximos, a fuga do grupo de zumbis é simplificada, com apoio no alto nível de exploração. Com o limiar aumentado, nota-se um aumento das fases com fator de exploração baixo, além de um maior balanço no controle dos recursos. Além de dificultar o caminho para seu objetivo, o CER também indica uma melhor distribuição no posicionamento dos recursos, acarretando em grupos de inimigos e mantimentos melhor separados, dificultando sua obtenção.

Na Figura 3 pode-se observar um certo equilíbrio na área de controle, utilizando um baixo limiar, com maior concentração de fases no meio e com valores baixos. Com o aumento deste limiar, além da diminuição da exploração comentada anteriormente, nota-se o aumento da área de controle. Isto é devido à diminuição de espaços para percorrer nas fases, assim como o aumento da segurança de alguns *tiles*, devido ao posicionamento de saídas e inimigos conterem maior diferença em

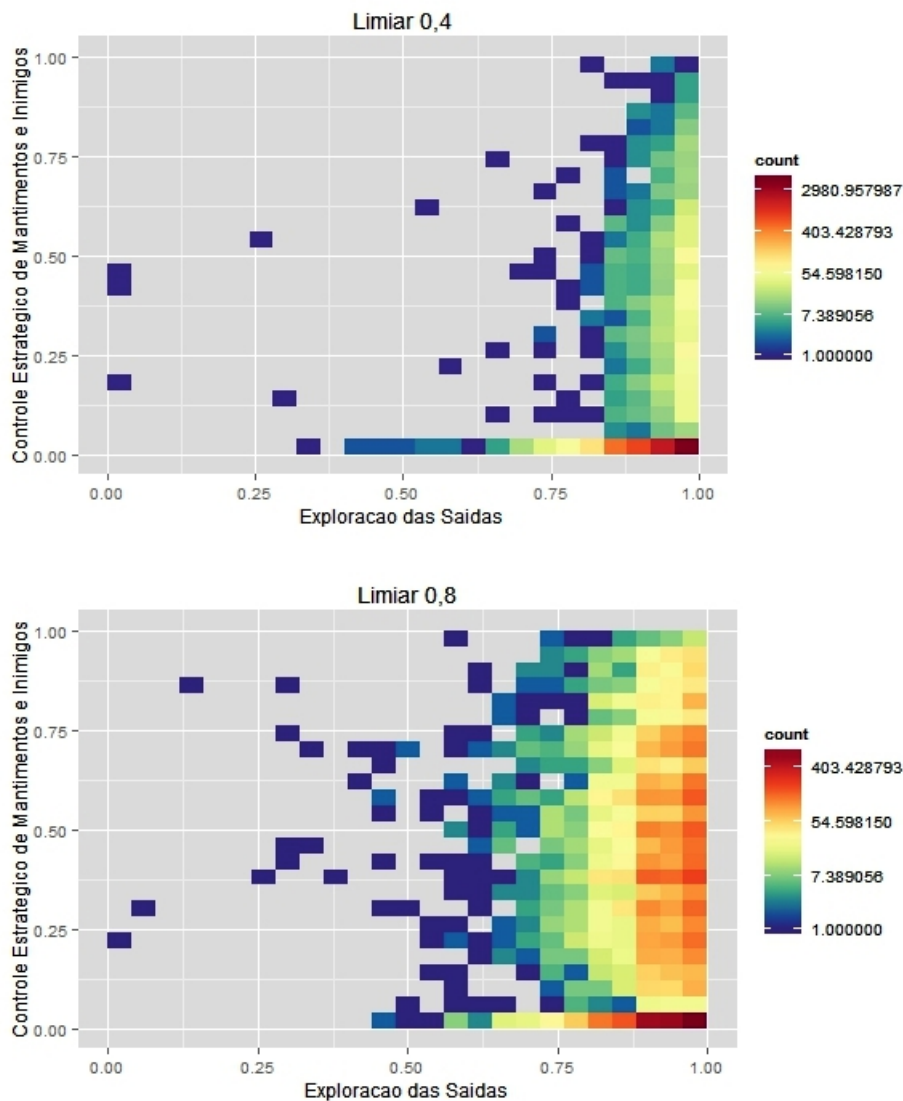


Figura 2 – Variação da expressividade das fases geradas, utilizando como métricas a exploração entre *tiles* de saída e o controle estratégico entre inimigos e mantimentos, com a mudança do parâmetro de geração.

seus posicionamentos, comparados uns aos outros.

Exemplos de fases para o jogo em questão podem ser vistos na Figura 4. As imagens superiores e inferiores demonstram fases geradas utilizando 0,4 e 0,8 como limiar, respectivamente. Note que, apesar das fases com o baixo limiar conterem vários zumbis, todos estão próximos, devido a isso, o jogador pode, seguir um caminho direto de sua posição inicial até a saída, evitando que os zumbis o alcancem. Com o limiar maior, a distribuição entre recursos é mais significativa, tornando o caminho da entrada até seu objetivo mais complexo.

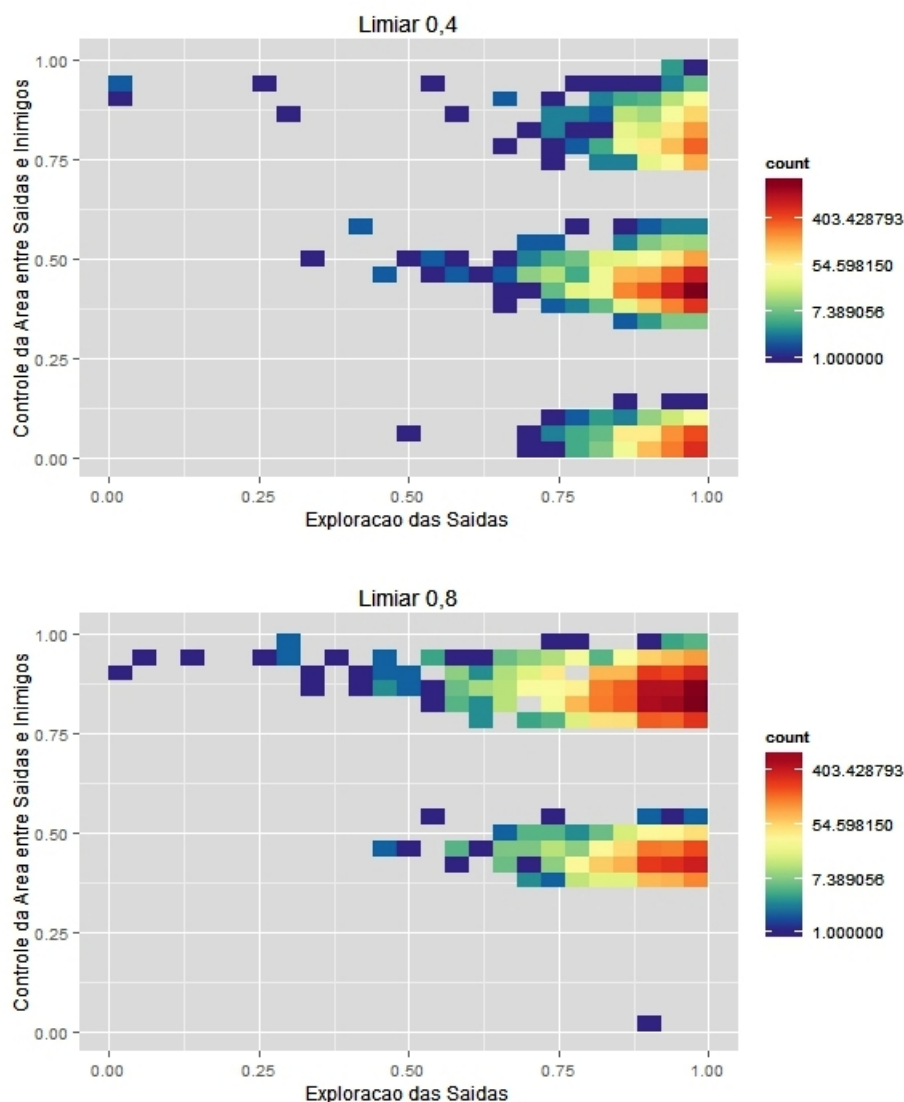


Figura 3 – Variação da expressividade das fases geradas, utilizando como métricas a exploração entre *tiles* de saída e o controle da área da união dos conjuntos saída e inimigos, com a mudança do parâmetro de geração.

## 7 CONSIDERAÇÕES FINAIS

Este trabalho apresentou um novo algoritmo para geração procedural de fases. Foi desenvolvido com uma abordagem composicional, onde são utilizados outros algoritmos já existentes como base, para que se possa criar um novo método. Além disso, foi desenvolvido uma ferramenta de código aberto implementando este algoritmo<sup>2</sup>, assim como todas formulas de avaliação utilizadas neste trabalho. Esta ferramenta tem como objetivo auxiliar no desenvolvimento de jogos, especificamente na área de criação de suas fases.

Como prova de aplicabilidade do algoritmo, assim como funcionalidade da fer-

<sup>2</sup> <https://bitbucket.org/gpgpc/compositional-level-generator>

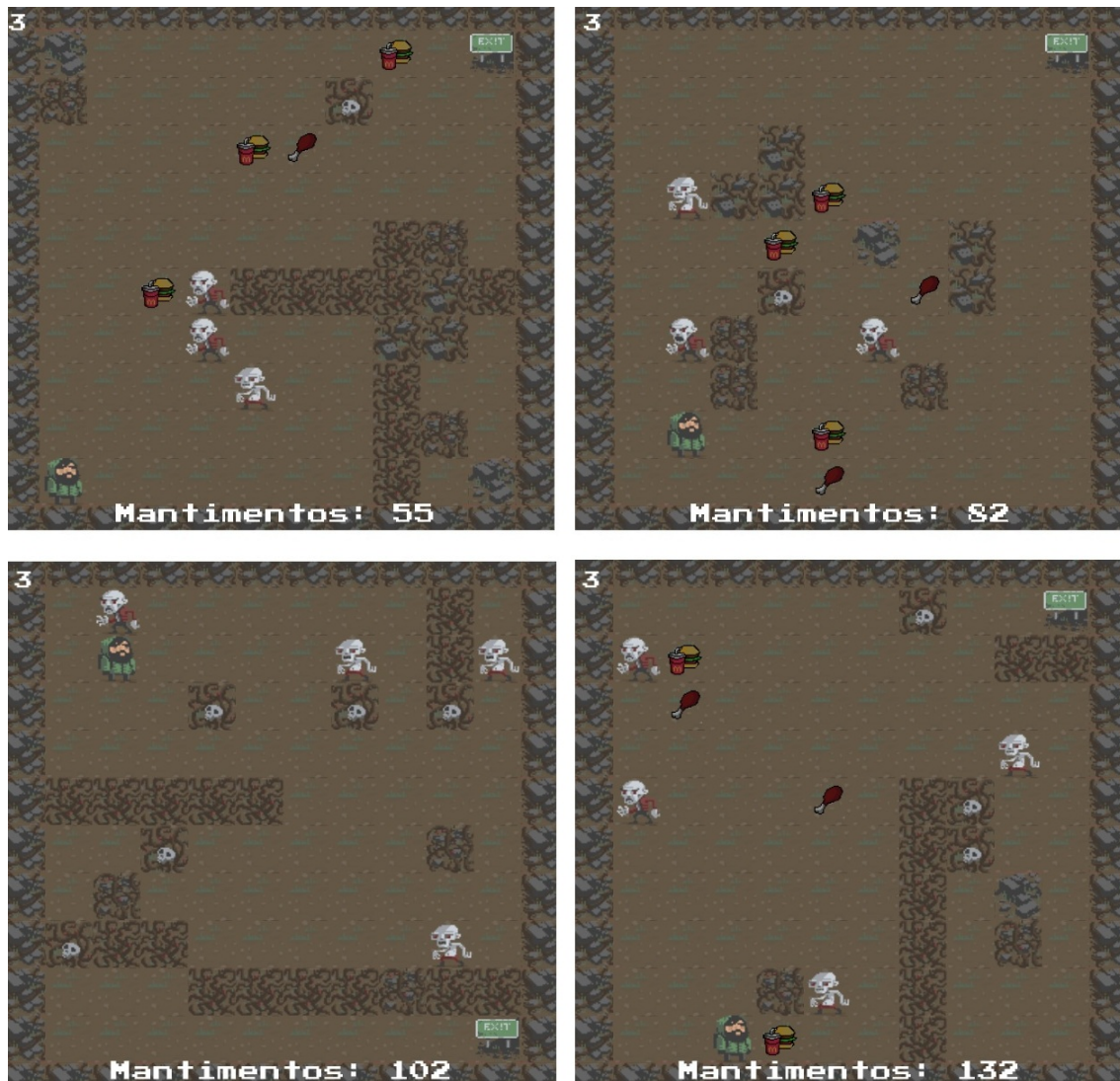


Figura 4 – Fases do jogo geradas proceduralmente. Imagens na parte superior utilizam limiar de 0,4 e na parte inferior limiar de 0,8.

ramenta, uma nova versão de um jogo criado pela Unity foi apresentado<sup>3</sup>. Neste jogo, o framework desenvolvido é utilizado na geração de todas suas fases, realizando-as em tempo de execução. Além disso, como avaliação do conteúdo gerado, foi realizada uma análise da expressividade do gerador desenvolvido.

Podemos concluir que o método criado alcança seu objetivo, uma vez que os testes demonstraram a variação de sua expressividade, estimulado pela troca de parâmetros, para diferentes métricas. Além do mais, a performance obtida no jogo, permite concluirmos que este é também aplicável em tempo real. Devido à isso, esta técnica proporciona um modo de jogo contendo um número de fases infinitas e inéditas.

Apesar dos testes se limitarem à um único tipo de jogo, a implementação do método na ferramenta desenvolvida permite sua utilização em outros tipos de jogos. Entretanto, para esta aplicação é necessário que um módulo, voltado ao tipo de jogo

<sup>3</sup> disponível para download em: [dl.dropboxusercontent.com/u/5064650/GameTCC/GameTCC.zip](https://dl.dropboxusercontent.com/u/5064650/GameTCC/GameTCC.zip)

desejado, seja implementado pelo desenvolvedor.

Os testes foram realizados com um único limiar, e a adição de outros parâmetros implicam em maior controlabilidade e capacidade de aproximação de uma saída desejada e serão analisados em trabalhos futuros. Porém, a alteração de regras da gramática, assim como dos agentes escolhidos, também podem realizar esta função e também serão analisados futuramente.

## 8 REFERÊNCIAS

- CARLI, D. et al. A survey of procedural content generation techniques suitable to game development. In: *Games and Digital Entertainment (SBGAMES), 2011 Brazilian Symposium on*. [S.l.: s.n.], 2011. p. 26–35. ISSN 2159-6654.
- DORAN, J.; PARBERRY, I. Controlled procedural terrain generation using software agents. *Computational Intelligence and AI in Games, IEEE Transactions on*, v. 2, n. 2, p. 111–119, June 2010. ISSN 1943-068X.
- HENDRIKX, M. et al. Procedural content generation for games: A survey. *ACM Trans. Multimedia Comput. Commun. Appl.*, ACM, New York, NY, USA, v. 9, n. 1, p. 1:1–1:22, fev. 2013. ISSN 1551-6857. Disponível em: <<http://doi.acm.org/10.1145/2422956.2422957>>.
- KELLY, G.; MCCABE, H. *ITB Journal A Survey of Procedural Techniques for City Generation*. 2006.
- LIAPIS, A.; YANNAKAKIS, G. N.; TOGELIUS, J. Towards a generic method of evaluating game levels. In: *AIIDE'13*. [S.l.: s.n.], 2013. p. –1–1.
- O'NEILL, M.; RYAN, C. Grammatical evolution. *Evolutionary Computation, IEEE Transactions on*, v. 5, n. 4, p. 349–358, Aug 2001. ISSN 1089-778X.
- RUSSELL, S. J.; NORVIG, P. *Artificial Intelligence: A Modern Approach*. 2. ed. [S.l.]: Pearson Education, 2003. ISBN 0137903952.
- SHAKER, N. et al. *Evolving Personalized Content for Super Mario Bros Using Grammatical Evolution*. 2012.
- SMITH, G.; WHITEHEAD, J. Analyzing the expressive range of a level generator. In: *Proceedings of the 2010 Workshop on Procedural Content Generation in Games*. New York, NY, USA: ACM, 2010. (PCGames '10), p. 4:1–4:7. ISBN 978-1-4503-0023-0. Disponível em: <<http://doi.acm.org/10.1145/1814256.1814260>>.
- TOGELIUS, J.; JUSTINUSSEN, T.; HARTZEN, A. Compositional procedural content generation. In: *Proceedings of the The Third Workshop on Procedural Content Generation in Games*. New York, NY, USA: ACM, 2012. (PCG'12), p. 16:1–16:4. ISBN 978-1-4503-1447-3. Disponível em: <<http://doi.acm.org/10.1145/2538528.2538541>>.