Processo de *design* de Fases por Lógica Booleana para Geração Procedural de Conteúdos em Jogos Digitais

Danny Duarte Moreira Mendes, Débora Mara Campos Silva, Henrique Nazar Rabelo,
Iuri Braga Amaral, Vitor Santor Granja, Marcelo Souza Nery
Pontifícia Universidade Católica de Minas Gerais
Instituto de Ciências Exatas e Informática,
Belo Horizonte, Brasil
dannydmms@gmail.com, debora__mara@live.com, henriquenrabelo@outlook.com,
iuribraga10@gmail.com, vitorsgranja@hotmail.com.br, msnery@gmail.com

***Resumo*—A geração procedural de conteúdos em jogos tem se mostrado uma boa ferramenta para produção, uma vez que possibilita (i) otimizar o tempo de criação, (ii) evitar o retrabalho de construção manual de conteúdos (como fases, modelagem e outros), (iii) controlar o balanceamento e *game design*, (iv) evitar experiências repetitivas, (v) contornar a falta de criatividade dos *designers*, (vi) testar diferentes parâmetros para serem avaliados na criação desses conteúdos, entre outros. Neste artigo é apresentada uma metodologia para a geração procedural de cenários blocados com perspectiva *RPG oblique*, com aplicação no jogo "*Sunken Shoot*" no estilo *bullet hell roguelite* em *pixel art*.**

***Keywords*-Geração procedural; Tilemap; Procedural Content Generation; Pixel art; Lógica booleana.**

I. Introdução

Existe uma relação direta entre a quantidade de conteúdo que um jogo tem e o tempo gasto manualmente para produzi-lo, o que reflete em um aumento no custo de sua produção [1]. Por isso, houve um aumento nos investimentos das empresas de jogos em soluções que otimizem tempo e custo, criando relevantes ferramentas de geração de conteúdo automatizado (procedurais, *Procedural Content Generation* ou PCG) para o *game designer* [2].

A geração automatizada permite que o próprio computador crie variações de conteúdos com base em um conjunto de regras previamente definidas. A implementação da geração procedural é feita para cada situação específica em cada jogo, tendo em vista suprir as particularidades de cada um, o que pode tornar o processo difícil e demorado. Ou seja, para jogos com pouco conteúdo, a geração procedural pode não ser uma boa opção, mas, para os que possuem um conteúdo considerável, tem a capacidade de auxiliar as equipes de arte e programação, conforme discutido por Blatz [3].

O fato dessa técnica ser aleatória pode tornar o conteúdo gerado por ela inconsistente, dependendo da quantidade de regras e restrições criadas, podendo gerar conteúdos bons e ruins do ponto de vista do *level design*, de acordo com Heaven [4]. Desta forma, a técnica sendo bem configurada fará com que a geração procedural consiga criar conteúdos apropriados, como explica Fort [2].

Entretanto, a inconsistência citada anteriormente é bem vinda em alguns tipos de jogos, como *Minecraft* [5] ou *Crypt of the Necrodancer* [6], pois é uma maneira de se quebrar experiências repetitivas e monótonas, aumentando o valor de rejogabilidade. Encontrar o meio termo entre esse *trade-off* é uma tarefa criativa do *game designer*, uma vez que a geração procedural não incluiu o *feeling* humano em suas regras — ela é meramente um processo de auxílio na redução de tempo e custo de produção.

Figura 1. Imagem do jogo *Sunken Shoot* com o mapa de geração procedural de *dungeons* aplicado.

A. O Problema

Ao se criar um jogo com o objetivo de dar uma experiência diferente cada vez que é jogado, montar manualmente este tipo de *design* se torna um processo moroso e cansativo.

Utilizando a PCG, é possível obter uma diminuição considerável na quantidade de trabalho que a equipe de produção nesse processo enfrentará, com os integrantes podendo se concentrar em outras necessidades do projeto.

B. Motivação

Com base no problema exposto, explorar técnicas de geração procedural torna-se interessante para jogos onde o conteúdo a ser criado é relativamente grande; desse modo, a criação automatizada busca reduzir os esforços manuais e,

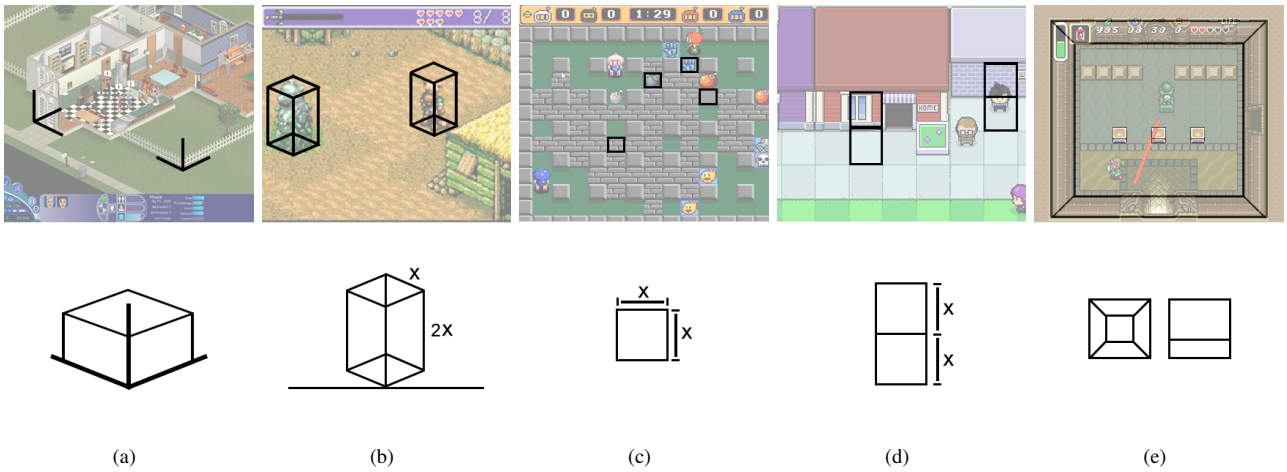


Figura 2. Jogos com seus tipos de projeções: (a) *The Sims* com a perspectiva *isométrica*, (b) *Landstalker: The Treasures of King Nole* com a perspectiva *dimétrica*, (c) *Bomberman 2* com a perspectiva *ortho top down*, (d) *Danger Crew* com a perspectiva *RPG oblique* e (e) *Legend of Zelda: A Link to the Past* com a perspectiva *Zelda top-down*.

consequentemente, tempo e custo. Além disso, a variabilidade de *design* proporcionada é relevante e infinita, além de ser uma técnica ágil e prática.

Contudo, gerar cada regra manualmente torna-se uma tarefa igualmente difícil e demorada dependendo do grau de complexidade desejado no conteúdo criado e o número de variáveis envolvido — chegando a ser explosivo o número de combinações possíveis; encontrar uma forma de organizar melhor essas regras e restrições facilitaria a PCG em qualquer tipo de jogo e problema desta natureza.

Assim, propõe-se nesse trabalho uma metodologia de criação de regras para uma PCG aplicada no jogo *Sunken Shoot*, um projeto acadêmico de conclusão de curso.

C. Organização do Texto

Este artigo apresenta uma metodologia para a geração procedural do *level design* no jogo *Sunken Shoot*. Na Seção 2 são discutidos trabalhos relacionados à PCG. A Seção 3 apresenta tanto a metodologia proposta quanto o jogo onde esta é aplicada. Na Seção 4 são demonstrados os testes e a avaliação dos conteúdos gerados. Por fim, a Seção 5 apresenta a conclusão e trabalhos futuros.

II. Estado da Arte

Umas das primeiras decisões a se tomar quando se fala em geração procedural são os aspectos ligados à arte: isso decorre do fato de que, quanto mais complexo forem os módulos básicos de construção, mais complexas serão as regras de criação de conteúdos. Desta forma, dada a restrição de complexidade, buscou-se compreender as características de diferentes estilos artísticos aplicados em uma PCG.

Dois aspectos importantes devem ser considerados: (i) a dimensionalidade do jogo (2D ou 3D) e (ii) o tipo de projeção escolhida (posição e forma de rebatimento dos raios na câmera que visualiza o cenário). Assim como nos passos descritos em [7] em seu livro *"Level Up"*, a definição do esboço de um jogo é crucial para um melhor aproveitamento do tempo. Neste caso, o estilo escolhido foi 2D em *pixel art* com uma visão *RPG oblique*, conforme ilustra a Figura 2.

Nas subseções a seguir serão apresentados os tipos de projeção, as características do estilo *pixel art* e alguns trabalhos relacionados à PCG.

A. Projeções

Existem duas formas básicas de projeção: *perspectiva* e *ortográfica*. Na projeção perspectiva, há um (ou mais) pontos de fuga onde os raios de projeção se concentram. Os efeitos decorrentes deste tipo de projeção são vários, como por exemplo a alteração do tamanho de objetos projetados na tela de acordo com sua distância em relação à câmera e também o efeito paralaxe. Na projeção ortográfica, os raios de projeção são paralelos, removendo o efeito de alteração de tamanho e paralaxe. Jogos 2D utilizam em geral uma projeção ortográfica (emulando o efeito paralaxe com deslocamento de planos sobrepostos em velocidades diferentes), enquanto jogos em primeira pessoa usam projeção perspectiva.

Existem três divisões principais na projeção ortográfica, sendo elas axonométrica, cônica e oblíqua [8]. Diversas formas de projeções podem ser usadas em jogos, e algumas são voltadas especificamente para essa mídia.

Como mostrado em [8], a projeção **isométrica** se dá quando os três ângulos projetados no plano são iguais entre si (de 120°), tendo como exemplo o jogo *The Sims* [9]. A **dimétrica** ocorre quando dois ângulos projetados no plano são iguais e um é diferente, como pode ser visto em *Landstalker: The Treasures of King Nole* [10]. A projeção **ortho top-down** é onde apenas um plano é visível, comum em

jogos mais antigos como, por exemplo, o *Super Bomberman 2* [11]. Ainda, como mostrado em [12], a **RPG oblique** é o tipo de projeção onde há dois planos visíveis e com três eixos com as mesmas proporções, que é possível encontrar em jogos como *Danger Crew* [13]. Por fim, na projeção **Zelda top-down**, cinco planos são visíveis para as salas, mas para objetos de cena utiliza-se a perspectiva *RPG top-down*, comum nos jogos da série *Legend of Zelda*, como em *Legend of Zelda: A Link to the Past* [14]. A Figura 2 ilustra todos os tipos de projeção citados.

B. Pixel Art

Pixel art é uma forma de arte digital na qual as imagens são criadas ou editadas tendo como elemento básico os *pixels* [15]. Seu termo foi apresentado por Goldberg e Flegal para denominar o que na época chamaram de imagens escaneadas combinadas a programas computacionais gráficos. Atualmente esse conceito se refere à manipulação de imagens a partir de sua menor informação possível (o *pixel* ou *picture element*), sendo assim imperativo que sejam consideradas imagens de baixa resolução, com essa especificidade existindo primordialmente devido às limitações tecnológicas da época dos primeiros sistemas gráficos [16].

É possível perceber as diferentes formas de *pixel art* analisando as características de (i) paleta de cores disponível, (ii) a resolução e (iii) a quantidade de elementos simultâneos na tela para cada uma das gerações de consoles (Figura 3). Nos *arcades*, em 1974, começaram a ser usados os *sprites* — uma sequência de imagens que são exibidas em uma sucessão de tempo pré-definida criando uma ilusão de animação. Na época, o *hardware* permitia apenas até 14 *sprites* simultâneos, com apenas uma cor e um *tile* ou camada em *bitmap* como plano de fundo. Na segunda geração a resolução passou a ser de 102×58 a 128×64 *pixels* visíveis na tela e um máximo de oito cores simultâneas. Durante a terceira geração houve uma grande evolução impulsionada pela melhoria da capacidade gráfica, surgindo a era 8 *bits*, com a resolução de 256×224 a 256×240 pixels, capacidade de até 64 *sprites* na tela, 25 cores simultâneas de uma paleta total de 64 cores. Na quarta geração começaram os jogos de 16 *bits*, com *sprites* maiores de 64×64 pixels até 16×512, com capacidade para rotação e 64 a 4096 cores na tela, paletas de 512 (9 *bits*) a 65.536 (16 *bits*) cores, permitindo várias novas técnicas serem aplicadas. Já a partir da quinta e sexta gerações, conhecidas como a era 32 e 64 *bits* respectivamente, iniciou-se a tridimensionalização dos gráficos [16].

Apesar de ter surgido de uma limitação tecnológica, o estilo agradou o público pelo mundo, a ponto de mesmo depois do surgimento de consoles mais poderosos, muitos estúdios e desenvolvedores independentes continuarem apostando neste formato, impulsionados pela nostalgia nos jogadores que cresceram jogando os clássicos dos anos 80 e 90, fazendo com que ao usar o *pixel art* abra mais possibilidades para

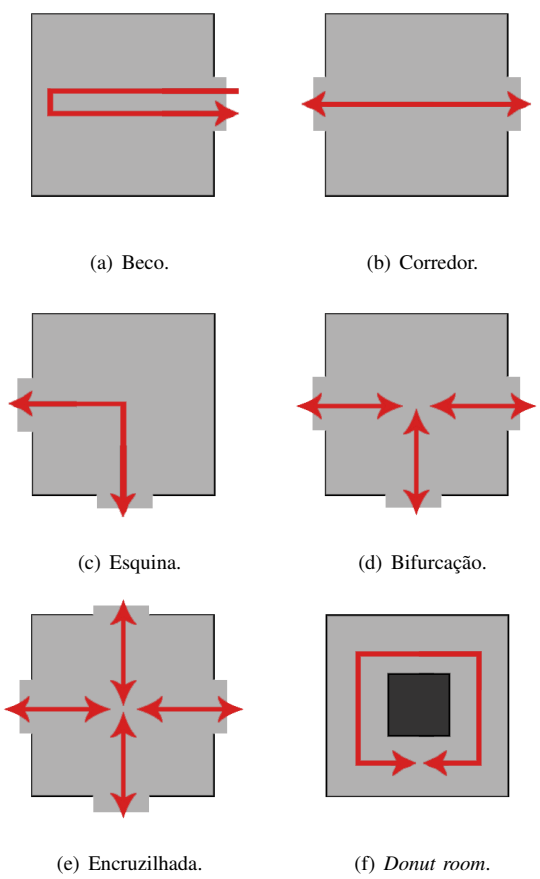
Figura 3. Jogos das gerações de *pixel art*, respectivamente: *Basketball* de *arcades*, *Pac Man* de *Fairchild Cannel F*, *Super Mario Bros* da era *8 bits*, *Sonic* da era *16 bits*, *Metal Slug X* da era *32 bits* e *Enter the Gungeon* da era *64 bits*

alcançar diferentes públicos, desde crianças até adultos [17]. Uma outra vantagem comum é para desenvolvedores de dispositivos móveis, onde as limitações de tamanho de tela, resolução e capacidade de processamento torna-se, para alguns equipamentos, um limitador a ser considerado.

C. Procedural Content Generation

A geração procedural pode ser dividida em dois tipos, de acordo com suas consequências: (i) quando afeta a jogabilidade e (ii) quando não afeta a jogabilidade. A *Geração Procedural* pura e simples, que não afeta a jogabilidade, cria apenas dados de forma aleatória ou pseudo-aleatória (é comum utilizar este termo para se referir mais a produtos visuais, sonoros ou objetos). Já a *Geração Procedural de Conteúdo* (PCG) afeta a jogabilidade de algum modo ao criar terrenos, mapas, itens, *quests* dentre outros elementos de jogo [18].

Uma das primeiras técnicas usadas de PCG foi a geração de números pseudo-randômicos, sendo usado pelo jogo *Elite* na década de 80 para gerar um largo universo [19]. A partir de então, diversas técnicas foram criadas, como em *Shatered Planet* onde há um controle na criação de salas com parâmetros como a altura, a largura, o número e o tipo em cada *level*. Dados estes parâmetros, os *levels* são compostos por números balanceados de salas pequenas e



(a) Beco. (b) Corredor.

(c) Esquina. (d) Bifurcação.

(e) Encruzilhada. (f) *Donut room*.

Figura 4. Tipos básicos de salas, definidos através dos portais de comunicação e o seus respectivos *gameflows*, representados pelas flechas vermelhas, conforme [22].

grandes de acordo com o desejo do *designer*. *Crypt of the Necrodancer* é um jogo similar, criando zonas com salas de tamanhos distintos, com jogadores inclusive capazes de mudar seus arredores [20].

Um elemento importante que Stout [21] apresenta é o *critical path* ou *main gameflow*, que é basicamente o caminho mais curto a se passar pelo *level* sem o uso de quaisquer trapaças ou atalhos. Em outras palavras, seria o menor caminho que o desenvolvedor deseja que o jogador experimente, não buscando que este complete cem por cento da fase, mas cumpra os objetivos principais.

Uma forma de trabalhar o projeto de uma fase é estabelecer blocos de construção básicos (salas ou *rooms*), que se interligam para criar o *gameflow*.

A Figura 4 ilustra os blocos básicos necessários para a criação de uma fase, variando-se apenas o tamanho, forma e quantidades de entradas/saídas (ou portais). Aqui, as unidades representadas são as mais simples possíveis no processo de construção de uma fase. Toda sala possui pelo menos dois elementos básicos: o portal (passagem que comunica uma sala à outra), que pode ocorrer entre 0 e n vezes nesta sala, e as paredes, que limitam o jogador dentro de uma região específica e controlável. É importante salientar que as imagens representadas na Figura 4 ilustram uma visão de topo de algumas salas, mas o *level design* pode ser pensado também tridimensionalmente, interligando as salas por meio de escadas, buracos no chão/teto, canos e outros elementos que fazem o papel de um portal. A seguir, serão discutidas as características principais de cada uma dessas salas.

- **Beco:** o beco (Figura 4 (a)) é uma sala sem saída, utilizada para indicar o fim de um ciclo de *gameflow* e onde normalmente itens são colocados escondidos para o jogador recolher. O fluxo no beco faz com que o jogador entre na sala, a explore e retorne para o ponto de entrada, representado pela seta vermelha. Becos quebram o fluxo linear de *gameflow* e podem ser um ponto de vulnerabilidade para o jogador ao ficar encurralado neste. Por fim, no caso do beco ser normalmente o ponto final de uma fase, ele representa o maior desafio onde está o *boss* e prêmios.
- **Corredor:** o corredor (Figura 4 (b)) é uma sala em que o fluxo do jogo é entrar por um portal e sair por outro, podendo estes portais estar dispostos horizontalmente ou verticalmente. A principal característica é que os portais devem estar em paredes opostas, mas isso não impede que o projetista os posicione um de frente ao outro. Portais faceados no contra-fluxo do jogo fazem com que certas regiões das salas funcionem como locais secretos para posicionamento de itens.
- **Esquina:** um corredor difere de uma esquina (Figura 4 (c)) pelo fato deste possuir seus portais em paredes vizinhas, criando assim um fluxo em "L". Esse recurso é extremamente útil em jogos, visto que corredores e esquinas em conjunto possibilitam ocultar parte de uma fase, visualmente falando, e também quebram o fluxo de caminho linear do jogo, forçando o jogador a mudar seu padrão de comportamento ao andar.
- **Bifurcação e Encruzilhada:** uma bifurcação (Figura 4 (d)) cria uma dinâmica diferente na fase: o jogador sempre deve optar por mais de um caminho a seguir. Isso introduz escolhas ao jogador e abre possibilidades de jogabilidade distintas para cada caminho. O mesmo ocorre em uma encruzilhada (Figura 4 (e)), aumentando as opções de caminho e escolhas. Uma encruzilhada geralmente ocorre no coração da fase, em seu centro, e é onde os diversos *puzzles* e fluxos de jogo se encontram.
- **Donut room:** por fim, o *donut* é uma sala que possui uma obstrução no meio do caminho. Na imagem da Figura 4 (f) a sala está representada sem portais, mas poderá haver quantos desejar (ou nenhum, caso o jogador caia na sala ou desça por uma escada a partir

de um nível acima).

Mesmo que as salas possuam modelagem mais orgânica, que os portais não necessariamente sejam literalmente portas, tais conceitos auxiliam ao *level designer* no processo de entedimento do fluxo do jogo. É por meio desse fluxo que se analisa onde posicionar determinados itens, passagens secretas, colecionáveis, inimigos e outros, elementos importantes para uma PCG. Tal esquema proporciona um meio mais simples de modelar bi/tridimensionalmente as fases, auxilia na otimização do ambiente e facilita o balanceamento de dificuldades do jogo.

Resta ainda observar que as salas não precisam ser pensadas apenas delimitadas por paredes. Imagine uma cena de um grande galpão onde há caminhões estacionados em posições diversas. Esses caminhões podem controlar o fluxo de caminho do jogador, tal qual as salas básicas o fazem – assim, pode-se mascarar o efeito de salas com elementos da própria cena do jogo, introduzindo maior imersão e contextualização do que simplesmente um conjunto de paredes delineando caminhos ao jogador.

Note que em nenhum momento é tratado nessa discussão aspectos do tamanho das salas, subdivisões internas, iluminação, texturas e elementos de cena. Esses aspectos mais artísticos são importantes, mas primeiro deve-se preocupar com o fluxo do jogo e em seguida com a estruturação visual.

III. Metodologia

Como visto, uma das principais dificuldades na criação das regras de uma PCG é a complexidade com que se deseja que os conteúdos sejam gerados. Para isso, as regras precisam ser pensadas não somente do ponto de vista lógico, mas sofre influência direta da arte do jogo, visto que os elementos devem encaixar visualmente para fazerem sentido para o jogador.

Desse modo, nas subseções seguintes serão apresentadas as características do jogo escolhido para aplicação da metodologia aqui proposta, que busca facilitar a PCG por meio de sua estruturação utilizando lógica booleana.

A. *Sunken Shoot: O Jogo*

Para aplicar a metodologia proposta, desenvolveu-se um jogo denominado *Sunken Shoot*, do gênero *bullet hell*, inspirado em jogos como *Enter the Gungeon* [23]. O jogo também possui características do gênero *roguelite*, dado ao seu estilo de progressão, tendo como inspiração o jogo *The Binding of Isaac* [24]. Nele, o jogador deve andar pelas salas de um navio naufragado e enfrentar hordas de inimigos sem esgotar sua vida, podendo conseguir itens em salas específicas para melhorar suas habilidades. Caso a vida chegue ao limite, a fase é reiniciada e o jogador recomeça o jogo. O jogador deverá enfrentar em cada nível um *boss* para continuar sua progressão. A criação de cada fase é feita proceduralmente utilizando a metodologia de PCG aqui descrita.

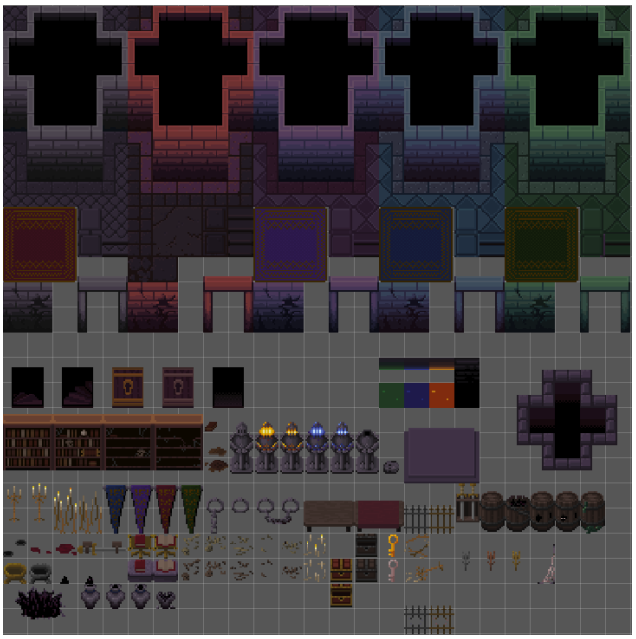


Figura 5. *Tile pallete* do jogo *Sunken Shoot*.

A escolha da visão *RPG top-down* deveu-se ao fato de que a construção dos módulos básicos de encaixe (paredes, chão, portas e outros) é facilitado nesse tipo de projeção por conter uma dimensão a menos do que blocos tridimensionais. A arte em *pixel art* também facilita a criação das regras de PCG, e a inspiração se deu nos jogos já citados *Enter the Gungeon* [23] e *The Binding of Isaac* [24], bem como em *Hyper Light Drifter* [25] e *Relic Hunters* [26].

O *gameflow* em *Sunken Shoot* é definido parcialmente pelo *script* de geração procedural e parcialmente pelas decisões do jogador, pois este tem o poder de escolha sobre o caminho percorrido até a sala do *boss*, podendo escolher um caminho direto ou explorar todo o mapa. Por esse motivo, um jogador experiente e um iniciante podem desfrutar igualmente de uma mesma fase do jogo, entrando ambos no estado mental do *flow* [27] — que indica que o ideal é manter um balanço entre desafios e habilidades de um jogador para que ele usufrua mais de um jogo e não se canse de jogar, não se estressando nem ficando entediado.

B. *Tilemap*

Para que o sistema procedural funcione corretamente e tenha um aspecto artístico não repetitivo, é necessário que os blocos básicos de construção das salas (e os elementos de cena) sejam pensados de modo a encaixar e tenham uma variabilidade tal que permita um número grande de combinações. Para isso, a técnica empregada foi a de *tilemap*. O *tilemap* é uma forma de organizar os *sprites* de um jogo bidimensional em um *grid*; o conjunto de todos esses blocos é denominado *tile pallete*, conforme ilustrado na Figura 5.

O uso do *tilemap* norteou na criação do sistema procedural, mas apesar da PCG ajudar em criar uma quantidade de fases muito alta, é necessário fornecer um conjunto elevado de salas pré-projetadas, ou *prefabs*. Sendo assim, foram projetadas 56 salas para a geração procedural obter um resultado não repetitivo e divertido. Esta forma de produção é uma ótima opção, visto que ela permite que o tempo de criação de cada sala seja mais otimizado por causa da facilidade de usar um espaço discreto de trabalho, e também pela simplicidade que o *tile pallete* proporciona ao construir salas modulares por ser possível inserir conjuntos de *sprites* de maneira organizada e, a partir desse conjunto organizado, pintar as salas.

C. Geração Procedural de Sunken Shoot

Em *Sunken Shoot* utiliza-se do método procedural baseado em *seed* [28], que é um valor inteiro que define o estado inicial de todos os métodos que geram resultados aleatórios e, uma vez definida a *seed*, o programa gerará sempre o mesmo resultado para aquela *seed* específica.

O organograma, ilustrado na Figura 6, mostra a sequência de passos que são tomados para que a geração do mapa de *Sunken Shoot* seja possível; cada um dos passos são descritos a seguir. A Figura 7 mostra um exemplo passo a passo do resultado obtido (apenas um minimapa para facilitar a compreensão).

Na primeira etapa, denominada **Criar sala base**, um *seed* é escolhido aleatoriamente e uma quantidade de salas mínimo por fase também, sendo pelo menos três — a sala inicial, a sala de recompensa e a sala do *boss*. Todo o espaço de jogo é dividido em células atômicas, cada uma correspondendo a uma sala completa. Na célula central deste espaço é criada, provisoriamente, uma sala do tipo *encruzilhada*, que é o ponto de partida de todo o mapa.

Em seguida, na etapa **Sorteia uma sala** é escolhida uma configuração de portais dentre as possíveis, já que não há uma sala isolada neste jogo (sem portal). As configurações de portais básicos estão indicadas na Figura 4 — atingindo quinze possíveis com as rotações destes. Uma sala só pode existir com um portal de saída encaixando com o portal de entrada de outra sala vizinha. Contudo, portais na direção de locais sem salas são permitidas, possibilitando a expansão radial do espaço. Caso ao verificar todas as possíveis posições na matriz e a sala não se encaixar em nenhuma delas, retorna-se para o passo anterior, como indicado em **O padrão se encaixa no jogo?**.

Como o processo de encaixe de portais é complexo (e aumenta exponencialmente com o número de portais por sala), observou-se que uma tabela-verdade poderia resolver facilmente o problema. Ao todo, doze variáveis foram necessárias para a construção desta tabela-verdade, o que atinge 2^{12} combinações ou 4096 possibilidades — inviável para um programador codificar além de correr o risco de errar algum valor de decisão. Para facilitar a construção das regras

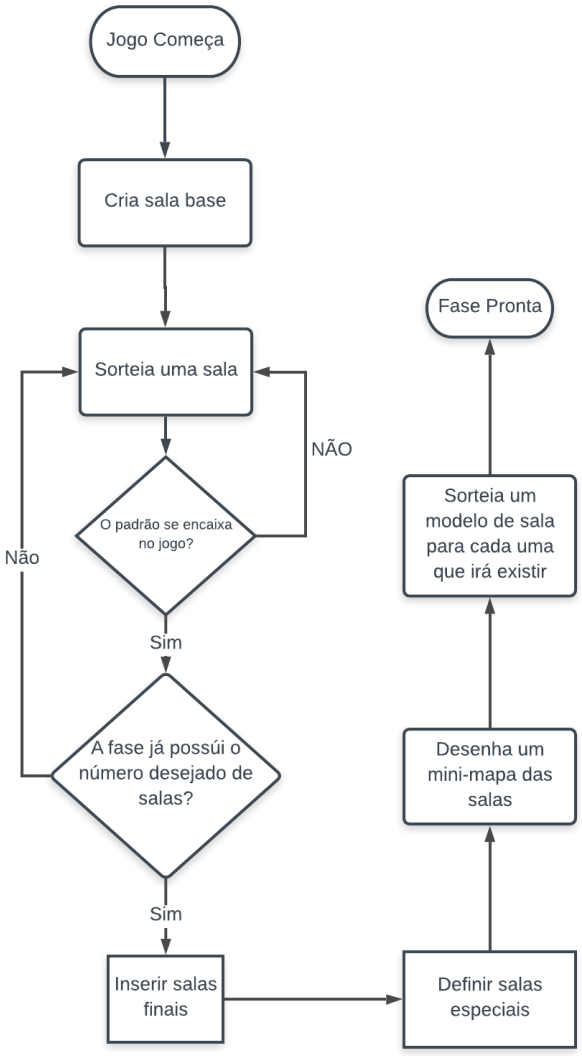


Figura 6. Processo completo de geração de níveis no jogo *Sunken Shoot* seguindo a metodologia proposta.

da PCG, utilizou-se o programa *Logisim* [29] e a Figura 8 mostra a estrutura lógica apenas do processo de validação da sala escolhida aleatoriamente. Nesta imagem, as entradas correspondem às condições da sala a ser inserida e do seu ambiente ao redor; as portas lógicas representam as etapas de decisão e a saída o resultado obtido.

Sempre que uma sala é inserida na matriz com sucesso, verifica-se se o número mínimo de salas inseridas já foi atingido na etapa **A fase já possui o número desejado de salas?**; caso ele seja menor que o número desejado, retorna-se para o primeiro passo; caso seja igual ao número desejado, o método segue para a próxima etapa.

Ainda podem existir salas nas extremidades com portais em direção a lugares sem salas, que são então inseridas

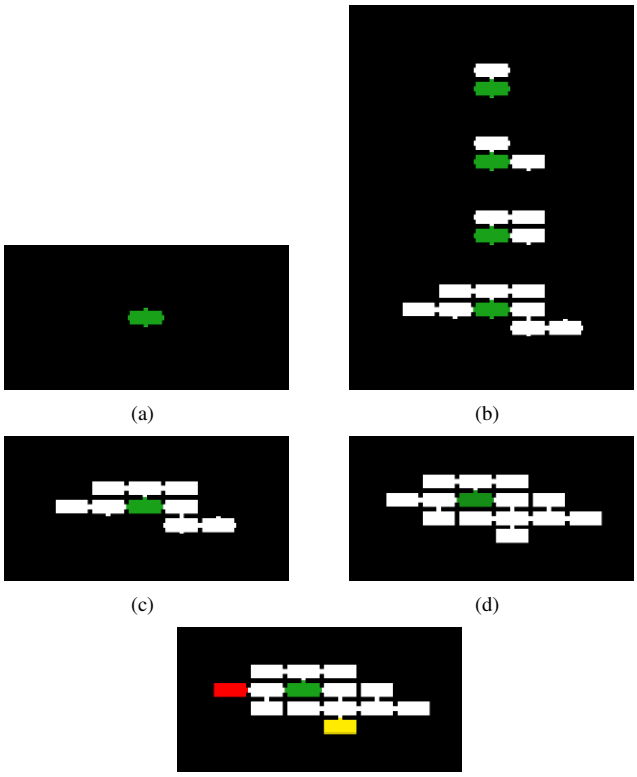


Figura 7. Exemplo com etapas da geração procedural de um mapa utilizando a metodologia proposta, respectivamente: (a) geração da sala inicial (verde), (b) *loop* de criação de salas vizinhas sorteadas (com adaptação de portais), (c) remoção de portais da sala base, (d) inserção de salas finais com base nos portais existentes que não possuem sala destino e (e) definição de salas especiais — sala de recompensa (amarela) e sala de *boss* (vermelha).

respeitando-se a regra de portais adjacentes na etapa **Inserir salas finais**.

Na etapa **Definir salas especiais**, diferencia-se as salas em grupos de funcionalidades, com exceção da sala base, que já é diferenciada como sala inicial desde sua primeira criação. Para que sejam definidas, todas as salas da matriz são analisadas em busca de salas *beco*, para garantir que essas salas estejam sempre em uma extremidade. As salas especias são a sala inicial (verde), a sala de recompensa (amarela), que possui sempre uma nova arma, e a sala de *boss* (vermelha), ilustradas na Figura 7 (e).

Com todas as informações sobre o *layout* do mapa já definidas, constrói-se o minimapa na etapa **Desenha mini-mapa**. Para evitar que todas as salas tivessem, artisticamente, o mesmo padrão interno de conteúdo visual, criou-se um conjunto base de modelos de salas manualmente, possibilitando assim que a equipe de arte tivesse um controle sobre o projeto dos *levels*. Ao todo, a equipe criou 56 salas desse tipo e algumas dessas podem ser vistas na Figura 9.

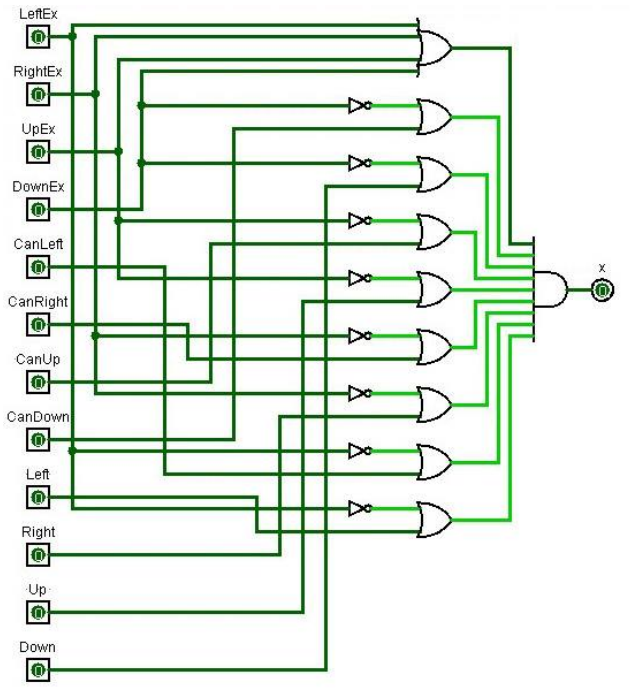


Figura 8. Estrutura lógica utilizada para o *loop* de inserir e validar uma sala com as regras de encaixe impostas, resultado representado na Figura 7 (b).

Assim, um desses modelos é aleatorizado na etapa **Sorteia um modelo de sala para cada uma que irá existir**, finalizando-se assim o processo de geração procedural.

Ao se derrotar o *boss*, nova fase do jogo é criada seguindo todo o processo descrito.

IV. Avaliação e Testes

Durante todo o processo de desenvolvimento do jogo, houve a realização de testes de usuários, que foram divididos em dois grupos: (i) os desenvolvedores, com conhecimento profundo do projeto, e (ii) os jogadores, sem informações sobre a geração procedural de conteúdos.

Os testes realizados pelos que conheciam o funcionamento do projeto, no quesito da geração procedural, foram simples; foi feito um teste repetitivo de criação de fase com alguém supervisionando as salas geradas a procura de erros e, sempre que eles aconteciam, anotavam-se suas características para facilitar aos desenvolvedores isolar o erro e corrigi-lo.

O teste realizado por jogadores foi feito em uma mostra de jogos. Ao todo, 38 participantes testaram o jogo por cinco minutos, sendo 32 homens e 6 mulheres, preenchendo um formulário com *feedbacks* ao final, cada um com subsecções para avaliar a jogabilidade, o *game design* e a arte. As análises dos dados nortearam melhorias no jogo como:

- acerca do tamanho das salas, que poderiam ter uma variedade maior de formato;

Figura 9. Alguns exemplos das 56 salas criadas manualmente pela equipe de arte.

- a diversificação no padrão artístico das salas, que poderia apresentar maior quantidade de elementos visuais;
- uma melhor distribuição de inimigos no cenário, para aprimorar o *flow*.

Assim, foi possível começar a traçar um plano de metas para uma futura atualização do jogo buscando cobrir os pontos menos qualificados observados pelo público.

V. Conclusão

Com o desenvolvimento do jogo *Sunken Shoot* foi notado que as técnicas de geração procedural de conteúdo auxiliam muito na velocidade de criação e variabilidade das fases. Contudo, deve-se tomar cuidado com a PCG, pois, diferentemente do método manual, pequenos erros de lógica podem ocasionar em conteúdos gerados de modo ruim (do ponto de vista do *design*), gerando retrabalho. Como a quantidade de variáveis a ser gerenciada para a PCG pode ser grande, a combinação destas pode ser explosiva; o método aqui proposto, utilizando lógica booleana, facilita o entendimento e codificação de tais regras.

Desta forma, foi constatado que a técnica de geração procedural é uma forma ainda pouco explorada comercialmente em jogos eletrônicos principalmente pelo grau de dificuldade da sua aplicação; no entanto, sua execução bem-sucedida traz consigo um avanço considerável no projeto.

A. Trabalhos Futuros

Uma extensão a esse projeto está sendo feita com o intuito de aumentar a diversidade de salas a serem criadas no jogo. Para isso, as salas poderão ocupar mais espaço na matriz base. Por exemplo, elas poderão ocorrer com um tamanho de 2×2 e poderão tomar qualquer combinação neste *grid*, formando um corredor longo, um "L" ou um quadrado com o dobro das dimensões de uma sala comum, conforme demonstrado na Figura 10 pelas salas roxas.

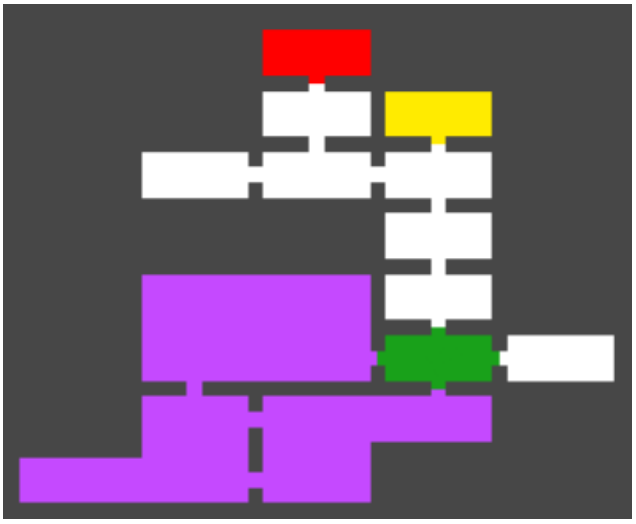


Figura 10. Exemplo de extensão da metodologia para Salas Grandes, visão do mini-mapa

As salas grandes serão inseridas logo após a etapa **Define salas especiais** (Figura 6) e ela só alterará as salas classificadas como normais (brancas). Assim sendo, as variantes a serem criadas pela equipe de *design* serão menores, não incluindo neste processo as salas especiais (inicial, recompensa e *boss*). Isso, por si só, já criaria uma sensação de maior variabilidade para os jogadores, visto que a quantidade de salas normais supera em número as salas especiais.

Isso está de acordo com as recomendações de Chris Wilson, um dos desenvolvedores de *Path of Exile* [30], que explica que "exagerar quantitativamente em uma característica específica tem efeitos reduzidos em sensação de diversidade e, por isso, deve-se inserir novos eixos de variação, para que se possa gerar mais possibilidades de fases com menos quantidade de conteúdo feito manualmente".

REFERÊNCIAS

[1] J. Ramos, M. Brasileiro, P. Moreira, Y. Oliveira Couto, M. Nery, "Análise da Aplicação de Geração Procedural no Contexto de Desenvolvimento do Jogo Caso Sombrio," *XVII Simposio Brasileiro de Jogos e Entretenimento Digital - SBGames*, pp. 28 – 37, 2018.

[2] F. Travis, "Controlling Randomness: Using Procedural Generation To Influence Player Uncertainty In Video Games," Bachelor Thesis in Digital Media, University of Central Florida, 2015.

[3] M. Blatz, O. Korn, A. Amato, W. Walk, D. Görlich, M. Barrett, H. Hacihabiboglu, G. Chan, A. Whitehead, A. Parush, V. Schwind, K. Wolf, N. Henze, Y. Oliveira, J. Schaal, and M. Kerssemakers, *Game Dynamics. Best Practices in Procedural and Dynamic Game Content Generation*. 03 2017.

[4] D. Heaven, "When infinity gets boring: What went wrong with No Man's Sky." URL: https://www.newscientist.com/article/2104873-when-infinity\-gets-boring-what-went-wrong-with-no-mans-sky/. [Online; acessado em 10 de Julho de 2019].

[5] M. A. Persson, "Minecraft." URL: minecraft.net, 2009. [Online; acessado em 30 de junho de 2019].

[6] R. Clark, "Crypt of the necrodancer." braceyourselfgames.com/crypt-of-the-necrodancer, 2015. [Online; acessado em 30 de junho de 2019].

[7] S. Rogers, *Level Up! The Guide to Great Video Game Design*. Wiley Publishing, 2nd ed., 2014.

[8] L. A. Ribeiro, "Apostila de perspectivas." Material de apoio didático, 2015.

[9] W. Wright, "The Sims." URL: www.ea.com/games/the-sims, 2000. [Online; acessado em 4 de julho de 2019].

[10] K. Orimo, "Landstalker: The Treasures of King Nole." URL:https://store.steampowered.com/app/71118/Landstalker_The_Treasures_of_King_Nole/, 1992. [Online; acessado em 02 de outubro de 2019].

[11] R. Shimada, "Super Bomberman 2," 1994.

[12] D. Santos, "Tutorial de pixel art." URL:davisan.wordpress.com/2015/11/02/tutorial-de-pixel-art-parte-00, 2015. [Online; acessado em 3 de Julho de 2019].

[13] D. Conley, G. LaBarre, "Danger Crew." URL: thedangercrew.com, 2019. [Online; acessado em 4 de julho de 2019].

[14] S. Miyamoto, "The Legend of Zelda: A Link to The Past." URL: https://www.nintendo.com/, 1991. [Online; acessado em 2 de Outubro de 2019].

[15] D. Silber, *Pixel Art for Game Developers*. 6000 Broken Sound Parkway NW, Suite 300: Taylor Francis Group, LLC., 2016.

[16] F. Alencar, "Pixel Art Low Poly Art: catalização criativa e a poética da nostalgia." Material de apoio didático, 2017.

[17] R. Dias, "Pixel Art: Tudo que Você Precisa Saber pra Começar." URL: https://producaodejogos.com/pixel-art. [Online; acessado em 15 de Julho de 2019].

[18] G. Cruz, "Estimativa da qualidade de mapas procedurais para jogos do gênero *roguelike*," *Journal of Sketchy Physics*, pp. 90–98, 2014.

[19] M. Hendrikx, S. Meijer, J. Van Der Velden, and A. Iosup, "Procedural content generation for games: A survey," *ACM Trans. Multimedia Comput. Commun. Appl.*, p. 24, 2011.

[20] B. Castro, R. Mota, E. Fantini, "Level Design on Rogue-Like Games: An Analysis of *Crypt of the Necrodancer* and *Shattered Planet*," *XV Simposio Brasileiro de Jogos e Entretenimento Digital-SBGames*, 2016.

[21] M. Stout, "Learning From The Masters: Level Design In The Legend Of Zelda." URL:www.gamasutra.com/view/feature/134949/learning_from_the_masters_level_.php, 2012. [Online; acessado em 5 de Julho de 2019].

[22] M. Nery, "Fundamentos de Jogos Digitais." Material de apoio didático, 2014.

[23] D. Crooks, "Enter the Gungeon." URL: dodgeroll.com/gungeon, 2016. [Online; acessado em 30 de junho de 2019].

[24] E. McMillen, "The Binding of Isaac." URL: bindingofisaac.com/, 2011. [Online; acessado em 30 de junho de 2019].

[25] A. Preston, "Hyper Light Drifter." URL: https://store.steampowered.com/app/257850/Hyper_Light_Drifter/, 2016. [Online; acessado em 02 de outubro de 2019].

[26] M. Venturelli, "Relic Hunters." URL: www.roguesnail.com/games/relic-hunters-zero, 2015. [Online; acessado em 12 de julho de 2019].

[27] M. Csikszentmihalyi, *Flow: The Psychology of Optimal Experience*. New York, NY: Harper Perennial, March 1991.

[28] P. Sampaio , A. Baffa , B. Feijó , M. Lana, "A Fast Approach for Automatic Generation of Populated Maps with Seed and Difficulty Control," in *16th Brazilian Symposium on Computer Games and Digital Entertainment, SBGames 2017, Curitiba, Brazil, November 2-4, 2017*, pp. 10–18, 2017.

[29] C. Burch, "Logisim." URL: cburch.com/logisim, 2008. [Online; acessado em 14 de julho de 2019].

[30] C. Wilson, "Path of exile." URL: pathofexile.com, 2013. [Online; acessado em 30 de junho de 2019].