Universidade de São Paulo
Instituto de Matemática e Estatística
Bachalerado em Ciência da Computação

Mathias Van Sluys Menck

Jogo com geração procedural guiada pelo jogador

São Paulo
Dezembro de 2018

Sumário

Capítulo 1

Introdução

Geração procedural — a tecnologia de computadores gerarem, a partir de regras impostas via código, algum conteúdo como música, peças artísticas, ou até o próprio código — vem sendo usada cada vez mais em jogos virtuais como uma alternativa de se montar coisas como níveis ou equipamento à mão ou até como técnica de se conservar memória.

No entanto, nesses jogos, o jogador pode encontrar algum conteúdo gerado que não gosta, possivelmente repetidas vezes, dependendo do quão vasto é o espaço de possibilidades do gerador.

Este projeto propõe um jogo virtual que se utiliza de geração procedural e permite o jogador interagir com o processo de geração, guiando o gerador para tipos de conteúdo de que mais gosta.

O conteúdo gerado para este projeto são as *dungeons* de um jogo, *dungeon* aqui significando um mapa contendo corredores, salas e inimigos que o jogador deve percorrer para avançar ao próximo nível. Os inimigos também são, de certa forma, gerados: todos compartilham uma base comum a qual, dependendo da dificuldade do nível atual, se adicionam algumas propriedades extras.

Teve-se em mente um foco em manter os geradores usados abrangentes no conteúdo gerado, possibilitando desta forma que os tipos de níveis gerados possam ser variados o suficiente para que sejam perceptíveis as diferenças no conteúdo gerado e assim que o usuário as sinta sem necessidade de as buscar explicitamente.

Capítulo 2

O Jogo para Validação

O estilo de jogo escolhido é um *top down dungeon crawler*, em que a câmera é fixa em cima de um personagem controlado pelo jogador e este deve percorrer um nível indo do início à saída e desviando dos ataques dos inimigos.

No entanto, não se apenas desvia dos inimigos. O jogador também tem o poder de atirar neles, assim eliminando as ameaças mais imediatas. Isso também se torna necessário pois o jogador não pode atravessar os inimigos diretamente, e várias vezes eles bloqueiam uma passagem que o jogador precisa tomar.

Foi escolhido este tipo de jogo para desenvolvimento por este se encaixar bem com geração procedural: a jogabilidade e os níveis podem ser simples o suficiente para que não seja necessária uma lógica complexa de dependências na geração, possibilitando que o computador gere conteúdo mais diverso do que se fosse restrito por muitas regras.

Com isso em mente, foi determinado que os níveis podem ser compreendidos como matrizes de dados e subsequentemente manipulados e visualizados com facilidade, então não é necessária uma abstração maior ou uma estrutura de dados mais complexa para a montagem dos níveis. Com isso os inimigos se tornam um conceito fácil de se gerar, com suas localizações sendo determinadas por um valor na matriz e seus atributos específicos decididos em tempo de montagem do nível.

Os inimigos em si são simples: todos são estacionários e atiram em direção ao jogador numa frequência fixa quando este está dentro de um determinado raio de detecção.

A decisão de que tipo de jogo usar no projeto ditou quase todas as outras decisões. Sem saber que tipo de experiência básica se queria fornecer ao jogador não seria possível imaginar os modos de se pedir para o computador variar o jogo. Tendo então o jogo final em mente, se iniciou o trabalho nos geradores usados, considerando que os atributos que o jogador iria afetar seriam o mapa e a dificuldade.

Capítulo 3

Algoritmos de Geração

Os algoritmos foram primeiramente testados usando a ferramenta Processing[1], pois possibilita rápida visualização dos resultados de cada iteração. Serão mostradas como exemplo nessa seção imagens desta ferramenta mostrando os níveis gerados pelo algoritmo descrito. Os espaços brancos seriam espaços vazios no mapa e os pretos paredes, com os pontos verde e vermelho indicando o início e a saída do mapa, respectivamente.

3.1 Mapa

O mapa gerado é uma matriz de valores 0 e 1. O valor 0 consignado a espaço livre e 1 a uma parede do labirinto. A escolha inicial de algoritmos foi afetada em grande parte pela preferência de usar algoritmos não presos a blocos de nível já montados, para assim garantir um maior espaço de possibilidades. Ou seja, usar um algoritmo que seleciona moldes de salas e corredores de uma lista não proporcionaria ao algoritmo a liberdade desejada.

Foram testadas três técnicas diferentes para a criação do mapa, sendo que todas pediam que a matriz do mapa já estivesse com as primeiras e últimas linhas e colunas da matriz já com paredes e o resto contendo espaços livres. O mapa é gerado dentro dessa moldura.

O primeiro algoritmo envolve *Random Selection* de pontos da matriz, em que um ponto, ao ser selecionado aleatoriamente, se torna uma parede. O número n de vezes que essa seleção aleatória ocorre é um dos parâmetros que seria modificável. O algoritmo não verifica se um ponto sorteado já havia sido sorteado antes, pois tratar este caso leva a um consumo maior de tempo ou mesmo ao algoritmo perder sua garantia de que termina.

Figura 3.1: *Exemplo de mapa gerado usando o primeiro algoritmo (n = 1500)*

O segundo algoritmo é similar ao primeiro, mas, ao invés de se sortear pontos n vezes, cada ponto tem chance p de ser uma parede. Ele tem a vantagem, em relação ao *Random*

[1]Página web da ferramenta: `https://processing.org/`

Selection, de que cada ponto é visto apenas uma vez e logo não ocorrem empates num sorteio. Este p seria o parâmetro que é modificável neste algoritmo.

Figura 3.2: *Exemplo de mapa gerado usando o segundo algoritmo (p = 0.55)*

Pode-se notar que ambos algoritmos acabam demonstrando como resultado mapas bem similares, com quase nenhuma estrutura definida e caracterizados principalmente por uma distribuição irregular de paredes e espaços livres.

O terceiro algoritmo testado consiste de uma partição binária da matriz. Neste algoritmo o espaço livre da matriz é dividido em 2, se tornando duas salas. Cada uma dessas salas então é também cortada em 2, e isso segue recursivamente até isso se repetir um determinado número de vezes ou as salas estarem pequenas demais para serem cortadas.

O número máximo de iterações seria, nesse caso, o parâmetro modificado de acordo com o *input* do jogador.

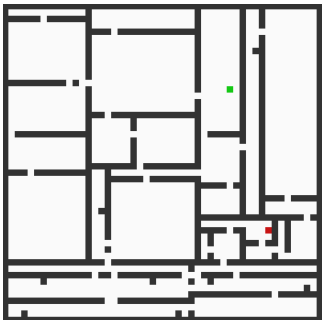

Figura 3.3: *Exemplo de mapa gerado usando o terceiro algoritmo (número máximo de iterações = 6)*

Também foi testado aplicar um autômato celular nos 3 algoritmos, sendo este um algoritmo que modifica a matriz ao verificar o quão populada por paredes está a vizinhança de um ponto após a geração inicial, e este ponto então ser mudado de espaço livre para parede ou vice-versa dependendo de quão populada está essa vizinhança. Isso foi implementado a priori para se obter um mapa com aparência menos caótica no caso dos dois primeiros algoritmos e fornecer mais um parâmetro alterável, sendo este o *threshold* de quando que um ponto deve ser parede ou não. No caso do terceiro algoritmo usar esta técnica retira quase toda a estrutura que o algoritmo provém.

Ao longo da implementação destes algoritmos nota-se que, para garantir que todo nível é terminável, ou seja, para garantir que o jogador teria acesso a todo espaço livre de um dado nível e logo pudesse ir de qualquer ponto entrada para qualquer ponto saída, seria necessário fazer um pós processamento da matriz após gerar o nível no caso dos dois primeiros algoritmos. No caso do terceiro, no entanto, percebe-se que para garantir isso só era preciso

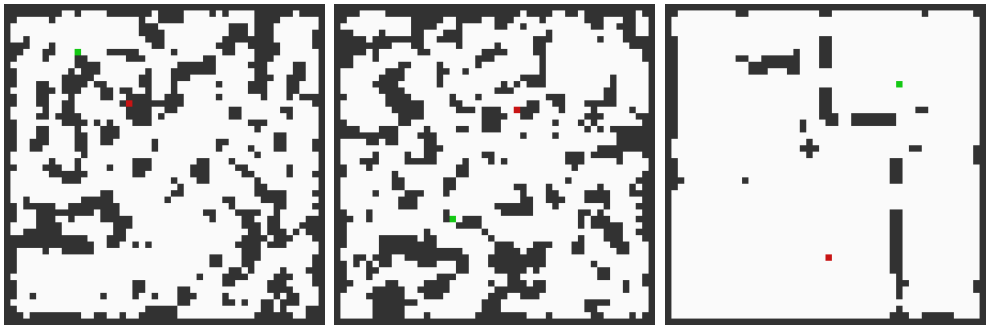

Figura 3.4: *Exemplos de mapas gerados após serem submetidos ao automato celular. Os algoritmos usados nelas são, respectivamente: Random Selection, cada ponto ter uma probabilidade de ser parede, e partição binária. (threshold = 4*

adicionar "portas"que conectassem as duas salas criadas a cada corte, e isso poderia se dar dentro do próprio algoritmo já montado, e logo custaria menos tempo durante a execução.

Por essa razão e pela qualidade organizada de seus resultados, optou-se por utilizar o terceiro algoritmo no jogo em si. As estruturas formadas com este algoritmo poderiam evidenciar com mais força para o jogador quando algo muda, pois a estrutura geral dos níveis gerados muda, e não apenas o número de espaços livres. Logo o jogador poderia ter mais consciência sobre se o conteúdo é ou não de seu gosto. Com esta escolha também decidiu-se não utilizar o autômato celular, já que os seus resultados quando usado em conjunto com a partição binária retira toda esta estrutura.

Depois dessa escolha se notou que mais dois parâmetros modificáveis poderiam ser usados na geração do mapa: Quando se corta o mapa na partição binária, é possível escolher entre o eixo maior ou menor. Escolhendo o eixo maior com mais frequência proporciona mais corredores no nível, enquanto o menor leva a um nível composto mais por quartos quadrados. Com isso decidiu-se criar chances diferentes para essa escolha de eixo, possibilitando então um controle maior do tipo de nível gerado.

Outro parâmetro utilizado foi que, ao criar uma parede na partição binária, é possível inserir uma ou mais passagens entre os dois lados da partição. Mudando então o número máximo dessas passagens se obtém níveis mais ou menos conexos, possibilitando mais controle do conteúdo ao usuário.

3.2 Dificuldade

A geração da dificuldade difere da de mapa no sentido de que não é adotado um algoritmo específico, mas que a dificuldade é determinada para cada quarto gerado pela partição binária usando uma distribuição normal, sendo a média e a variância da distribuição os parâmetros que variam com o *input* do jogador.

Este valor de dificuldade de um quarto afeta o número de inimigos que nele são colocados, sendo esse número determinado a partir de um cálculo que envolve tanto o valor de dificuldade quanto a área total do quarto. Isso é feito para que, mesmo em dificuldades mais altas, sempre haja algum espaço para o jogador se mexer, mesmo em quartos pequenos com vários inimigos. Tendo então o número de inimigos a ser posto num quarto, eles são dispostos aleatoriamente pelo quarto, e, similarmente ao *Random Selection* do segundo algoritmo testado na geração de mapas, não se checa se um inimigo é posto onde já tem um inimigo. Nesse caso isso ajuda também a não se obter constantemente o mesmo número de inimigos em quartos de áreas e dificuldades iguais.

A dificuldade de um dado quarto também pode dar "habilidades especiais"para os inimigos nele, podendo essas serem: mais pontos de vida, a habilidade de atirar através de paredes, ou tamanho menor, sendo assim mais difíceis de acertar. Quanto maior a dificuldade de um dado inimigo ele pode ter duas ou até mesmo as três habilidades concorrentemente.

Outras propriedades de um inimigo que são afetadas pelo valor de dificuldade são o seu raio de detecção do jogador, ou a que distância o jogador deve estar para o inimigo começar a atacá-lo, e também a velocidade com que o inimigo atira. Ambos atributos crescem quanto maior a dificuldade do inimigo.

Foi escolhido este método de geração de dificuldade pois ele é simples e permite que um mesmo nível tenha uma abrangência grande de dificuldades, se a variância for alta, ou que ela se mantenha constante, com variância baixa. Com isso há uma diversidade grande de conteúdo com o qual o jogador pode interagir sem ser necessário criar uma grande quantidade de conteúdo, e o que é gerado continua sendo perceptivelmente alterado quando os parâmetros de geração são alterados.

Capítulo 4

Desenvolvimento do Jogo

O desenvolvimento do jogo em si se iniciou apenas tendo todos os algoritmos de geração decididos, e os parâmetros que são alterados com a entrada do jogador são:

- Número de iterações da partição binária.
- Chances da partição binária cortar no eixo x, y, ou qualquer um deles.
- Quantas portas podem ser inseridas numa parede de partição binária.
- O valor médio da distribuição normal de dificuldade.
- A variança da distribuição normal da dificuldade.

A *engine* escolhida para desenvolver o jogo foi a Godot[1], uma engine *open source* em que já se tinha prévia experiência e logo não seria necessário um grande período de aprendizado para se conseguir montar o jogo. Outra qualidade que destacou a Godot como a engine a ser escolhida é sua modularidade. Nela um projeto é dividido em várias cenas e scripts, o que possibilita uma divisão dos códigos de geração e dos códigos do jogo em si.

Havia uma preocupação se implementar o gerador em *GDscript*, a linguagem própria da Godot, seria eficiente, e assim foi pesquisada a funcionalidade da Godot de usar códigos externos em suas bibliotecas, para se eventualmente implementar o gerador desta forma. No entanto isso não foi necessário pois o gerador roda sem problemas na Godot, e a espera entre níveis, enquanto o gerador cria a próxima *dungeon*, não passa de dois a três segundos, tempo de duração da transição entre as cenas do jogo.

Foi decidido que não haveriam inimigos em torno do ponto de início, em um raio maior que o possível os inimigos alcançarem para que o jogador não seja atacado sem ter tempo de reagir. Essa retirada de inimigos é o único pós processamento que ocorre no nível após este ser gerado.

Devido à natureza de Trabalho de Conclusão de Curso do projeto, não foi considerado necessário compor sons ou animações complexas para o jogo. Fora isso os elementos visuais foram feitos a mão sem grandes preocupações para se ter aparência de um jogo comercial.

Os controles escolhidos durante *gameplay* foram as teclas W, A, S, e D para mover o personagem e o mouse para apontar e atirar. Isso foi decidido quase que puramente pela simplicidade e o quão comum esse esquema de controle é, assim não foi gasto muito tempo na escolha dos controles. As únicas exceções a esse esquema são a tela inicial do jogo, a tela de *game over* e a tela de dar notas ao nível completo. Nestas, toda interação do jogador com o jogo se dá com o mouse.

[1]Site da engine: `https://godotengine.org/`

A cada nível o jogador dá uma nota para a exploração e para a dificuldade do nível percorrido de acordo com seu aproveitamento deste, e isso altera os parâmetros na geração do próximo nível. Essa mudança é simples: para todos os parâmetros é gerado um novo valor dentro de seu respectivo intervalo de possibilidade. Por exemplo, os parâmetros de probabilidade são valores entre 0 e 1, a média da dificuldade é entre 1 e 10, etc. Tendo esses novos parâmetros é então calculada uma média ponderada entre o valor atual de um determinado parâmetro e o novo valor gerado. Seja pA o valor atual do parâmetro, pN o valor gerado, n a nota dada e $nMax$ a nota máxima possível. O cálculo do novo parâmetro é:

$$\frac{pA * n + pN * (nMax - n)}{nMax}$$

Este método foi escolhido graças a sua simplicidade tanto lógica quanto de implementar. No entanto restou uma vontade de alterar este método para forçar uma distância do parâmetro atual dependendo mais da nota dada pelo jogador do que do valor gerado no momento, o que não foi possível devido a falta de tempo para realizá-lo. Um método que talvez possibilitasse isso seria usar as notas não como os pesos numa média ponderada mas como delimitadores do intervalo sobre o qual o próximo parâmetro é gerado.

Exemplos de mudança nos parâmetros do gerador alterando o póximo nível:

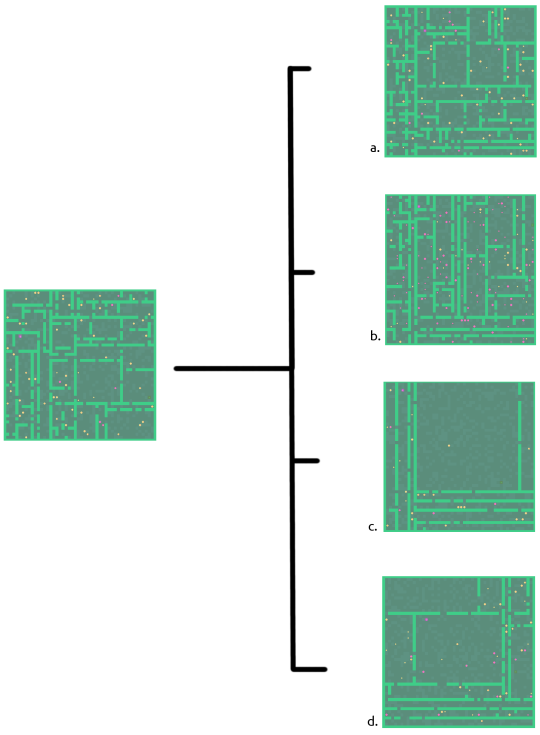

Figura 4.1: *Os exemplos da direita são derivados do da esquerda, sendo as notas dadas:* ***a****: Exploração-4 Dificuldade-4;* ***b****: Exploração-4 Dificuldade-0;* ***c****: Exploração-0 Dificuldade-4;* ***d****: Exploração-0 Dificuldade-0.*

Capítulo 5

O Jogo

Ao iniciar o jogo o jogador vê a tela de início do jogo, e para prosseguir é necessário apenas clicar em *Start!*:

Figura 5.1: *Tela inicial do jogo. Seu propósito é apenas que o jogador não seja enviado diretamente ao jogo.*

Após o fazer, o jogador é posto diretamente no jogo, ganhando controle do personagem. É apenas neste momento, entre a tela de início e do jogo em si, que o nível é gerado.

Um exemplo da típica tela de jogo durante *gameplay* é:

Figura 5.2: *A barra no canto superior direito indica a vida do jogador, e o número abaixo dela quantos níveis foram completos em sequência sem perder o jogo. A figura rosa é o personagem do jogador e as amarelas são os inimigos. Na imagem apresentada o inimigo central está atirando no jogador.*

O jogo acaba quando o jogador perde todos seus pontos de vida e é levado a essa próxima tela, onde, ao clicar *Again*, é levado a um novo nível diretamente.

Figura 5.3: *Tela indicativa da perda do jogador.*

Ao chegar no fim de um nível é pedido que o jogador dê notas ao nível que acabou de completar, de acordo com preferências pessoais. São duas notas a serem dadas:

- Exploração, que comprime os parâmetros de criação do mapa em si. Estas afetam o layout do próximo mapa.

- Dificuldade, que altera a distribuição normal usada para se atribuir dificuldades.

As notas são representadas pelas barras respectivas, com o quanto mais á direita estiver o indicador na barra, maior a nota e mais os parâmetros de tal atributo se manterão iguais.

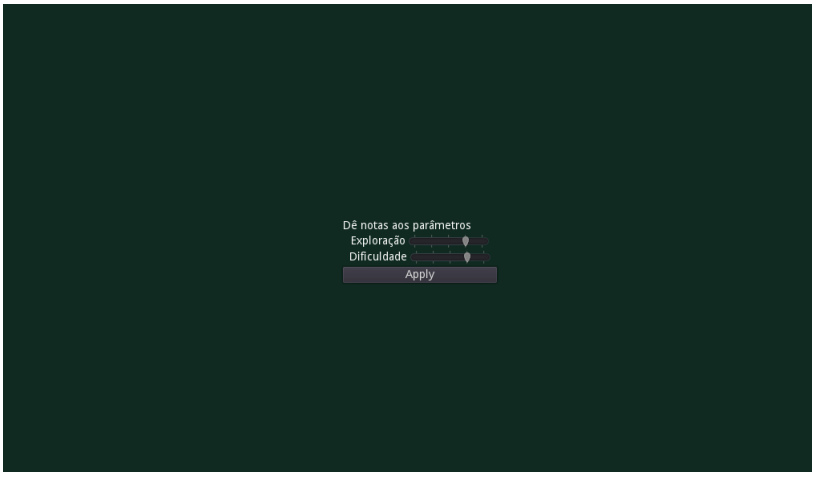

Figura 5.4: *A barra de notas tem 5 níveis, de 0 a 4.*

Ao se selecionar as notas desejadas o usuário deve clicar em *Apply* e irá para o próximo nível, já com os novos parâmetros sendo usados.

Capítulo 6

Conclusão

A geração se mostrou eficiente com o algoritmo de partição binária, possibilitando um grande número de alternativas para o nível gerado mas ainda mantendo uma estrutura de modo que o jogador possa reconhecer facilmente certas qualidades no mapa gerado, como o número de salas, uma presença maior de corredores, salas mais expansivas ou mais constritas, etc. Isso ajuda principalmente na hora do jogador determinar se o que foi gerado é ou não de seu gosto, e assim levar o gerador a um estado mais desejável. Isso é parte do que levou à escolha da partição binária sobre os outros dois algoritmos testados. Neles o algoritmo base levava a níveis que possibilitavam uma mobilidade muito baixa, necessitando assim um pós processamento para garantir que o jogador pudesse ir de qualquer ponto a a qualquer ponto b. Isso, é claro, levaria a uma eficiência mais baixa e logo a uma espera maior entre cada nível para o jogador, além de não dispor de estruturas mais reconhecíveis.

A escolha da partição binária também aumentou o número de parâmetros que são afetados pela nota dada pelo jogador. Isso ajudou a prover ao usuário um controle maior, mesmo que indireto, sobre o mapa gerado. Os parâmetros de dificuldade, apesar de serem menos que os de exploração, também são o suficiente pois possibilitam já um grande número de variação com o número de inimigos por sala e as variações entre os próprios inimigos. Tendo em mente a simplicidade do jogo, não faria sentido aumentar o número dos parâmetros de dificuldade sem se adicionar ainda mais tipos de obstáculos ao jogador.

No entanto num jogo mais complexo ainda seria possível usar as técnicas utilizadas neste projeto, sendo necessário principalmente se ter em mente que parâmetros de um dado gerador podem ser modificados e como, para se manter sempre a lógica do próprio gerador. Um exemplo seria, num jogo que se utiliza de um ambiente 3D, gerando mapas neste espaço, e que pede ao jogador pular de plataforma em plataforma em busca de um fim para um determinado nível: variar a distância entre as plataforma junto com a gravidade do nível, modificando assim a dificuldade de certos pulos; ou dando preferências a níveis mais verticais ou horizontais, o que possibilitaria ao jogador indicar o tipo de nível que prefere explorar. Neste caso poderia ser determinado que o valor da gravidade está fora de "Exploração"ou "Dificuldade", mas as técnicas descritas aqui possibilitariam mesmo assim uma implementação similar para uma terceira categoria de notas a serem dadas. Mesmo em um jogo mais comercial, em que as pausas entre cada nível para dar as notas não são desejadas, seria possível usar talvez uma função de tempo com o qual o jogador interagiu com um dado conteúdo para graduar ele, tendo em mente que o jogador passaria mais tempo com o conteúdo que gosta, mas nesse caso teria de se ter mais conhecimento sobre o comportamento de jogadores do que se propõe neste projeto para fazer algo de fato funcional.