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

Proceduroidvania

um jogo metroidvania com missões geradas de forma procedural

Rogerio Marcos Fernandes Neto

Monografia Final

mac 499 — Trabalho de Formatura Supervisionado

Supervisor: Dr. Wilson Kazuo
Cossupervisor: Prof. Dr. Fabio Kon

São Paulo
2022

Para você.

Agradecimentos

Agradecimentos especiais à minha noiva Letícia que está junto de mim a muito tempo e sempre me trouxe forças e alegria ao longo da minha vida.

Resumo

Rogerio Marcos Fernandes Neto. **Proceduroidvania: *um jogo metroidvania com missões geradas de forma procedural***. Monografia (Bacharelado). Instituto de Matemática e Estatística, Universidade de São Paulo, São Paulo, 2022.

É muito difícil de se criar jogos com game design interessante. Diversos jogos utilizam a geração procedural como um recurso de game design que permite rejogabilidade e mais velocidade no desenvolvimento. Joris Dormans, em sua tese de doutorado, propõe uma *framework* para geração de jogos por meio de uma formulação do conceito de missão em jogos através da linguagem de grafos. Dessa forma, pode-se gerar estruturas através de regras de reescritas para grafos. Esse método promete ser bem expressivo e atender uma grande variedade de gêneros, permitindo gerar estruturas para até mesmo jogos de ação e aventura. Buscou-se avaliar a facilidade de utilização do método através da criação de um jogo do gênero metroidvania. O jogo final desenvolvido permite a circulação de um personagem em diversas salas conectadas de forma procedural com chaves e trancas. Tais estruturas foram geradas através de quatro regras de reescrita que acrescentam trancas e chaves, vértices e arestas no grafo de forma aleatória. Com poucos elementos a serem gerados de forma procedural, o método se adapta bem, mas para inserir mais elementos procedurais o código pode adquirir complexidade indesejável tanto na implementação quanto no desempenho. Nesse caso, incentiva-se a utilização de bibliotecas especializadas em analise sintática e transformação de grafos.

Palavras-chave: Geraçao procedural. Jogos digitais. Jogos. Metroidvania. Gramática de grafos.

Abstract

Rogerio Marcos Fernandes Neto. **Proceduroidvania: *a metroidvania game with procedurally generated misisons***. Capstone Project Report (Bachelor). Institute of Mathematics and Statistics, University of São Paulo, São Paulo, 2022.

It's really hard to create games with an interesting design. Many games use procedural generation as a resource for improving design, which allows replayability and more speed for development. Joris Dormans, in his P.h.D. thesis, proposes a framework for generating games by formulating the concept of mission with graph language. This way, one can generate complex structures with graph via rewrite rules. This method promises to be very expressive and suit many game genres, allowing procedural generation even in adventure games. The final game developed in this work allows a character to walk through different scenarios that are procedurally connected through keys and locks. Such structures were generated by four rewrite rules which add keys and locks, vertices, and edges to the graph randomly. With just a few elements to be procedurally generated the method adapts very well, but to fit more procedural elements the code may acquire unwanted complexity both in the implementation and the performance. In this case, the use of specific libraries to parse and transform the graph is incentivized.

Keywords: Procedural generation. Games. Digital games. Metroidvania. Digital games. Graph grammar.

Lista de figuras

Sumário

Introdução

Contexto

Game design é o conjunto de todos os recursos artísticos e mecânicas que compoem um jogo (Schell, 2020). Esses elementos são selecionados e compostos de tal forma a passar uma experiência que os criadores visavam no momento de concepção do jogo. Colocar um ritmo mais acelerado ou mais lento a um jogo, por exemplo, pode mudar completamente o que jogador sente durante sua jogatina e cabe aos *game designers* saber quais tipos de recursos explorar para passar a mensagem que buscam. A palavra *design* nesse contexto, diferentemente do que pode sugerir, engloba não só aspectos estéticos, mas sim a interação e harmonia entre elementos artísticos e mecânicas de jogo a fim de invocar certos tipos de sentimento ao jogador e compor a experência. É natural que para determinados gêneros de jogos, certos aspectos sejam mais relevantes do que outros, e grande parte do foco dos desenvolvedores recaia sobre eles. Para um jogo de quebra-cabeças, como por exemplo Portal (Valve, 2007), grande parte do *game design* está em como elaborar novos quebra-cabeças. Por outro lado, em jogos de corrida como Mario Kart (Nintendo, 1992 - 2014), grande parte do *game design* está no desenho das pistas e seus obstáculos. Nos jogos de ação e aventura, como o The Legend of Zelda (Nintendo, 1986 - 2020), o *design* se concentra nas missões que o personagem deve cumprir para ser capaz de prosseguir e os ambientes que a personagem percorre.

Como criar um bom *design* para jogos é uma tarefa difícil, desenvolvedores utilizam varias técnicas para facilitar esse processo, como por exemplo a geração procedural. Quando o conteúdo para um jogo é gerado através de algoritmos, diz-se que que esse jogo, ou parte dele, é gerado de forma procedural. Uma vasta gama de conteúdo pode ser gerada de forma procedural e acontecem em duas principais formas: são usados na fase de desenvolvimento ou fazem parte das mecânicas do jogo. Certos jogos de mapa aberto possuem um cenário muito grande, como por exemplo The Witcher 3 (CD Projekt RED, 2014) (Figura 1a), que tornaria seu desenvolvimento muito complexo, mas os *designers* conseguem explorar recursos de geração procedural para gerar paisagens grandes e complexas para partir como base e adequar a estrutura com a temática do jogo. Detalhes como grama e nuvens também podem ser gerados de forma procural a fim de trazer mais realismo e naturalidade. Outros jogos, como Minecraft (Mojang, 2011) , também apresentam mapas bastante extensos, mas por meios diferentes. No Minecraft, em vez do mapa ser gerado ao longo desenvolvimento, o mapa é gerado de forma procedural toda vez que o jogador inicia um novo mundo, proporcionando uma experiência nova a cada mapa novo gerado (Figura 1b). Diversos outros recursos podem ser gerados com o mesmo método empregado por esses

dois jogos. Portanto, as ferramentas de desenvolvimento procedural apresentam grande valor prático, uma vez que funcinam tanto como ferramenta de desenvolvimento e como mecânica de jogo, o que permite maior velocidade na criação do jogo e também permitem alto nível de rejogabilidade (Short e Adams, 2017).

Motivação

Apesar de ter grande importância, a geração de conteúdo procedural encontrado em jogos atualmente é pautada em algoritmos que utilizam força bruta para gerar uma grande quantidade de estruturas que em seguida são selecionadas de acordo com algum critério (Johnson *et al.*, 2010) de exigência do jogo. Apesar de funcionarem bem no contexto de jogos atuais, esses métodos não se transportam bem para outros tipos de jogos, pois, em geral, são métodos utilizados especificamente para algum gênero de jogo. *Roguelikes*, por exemplo, tem seu conteúdo procedural focado na construção do personagem. Portanto, não possuem a expressividade necessária para criar jogos de ação e aventura onde exploração, resolução de quebras cabeças e desafios de destreza ocupam maior parte da jogatina.

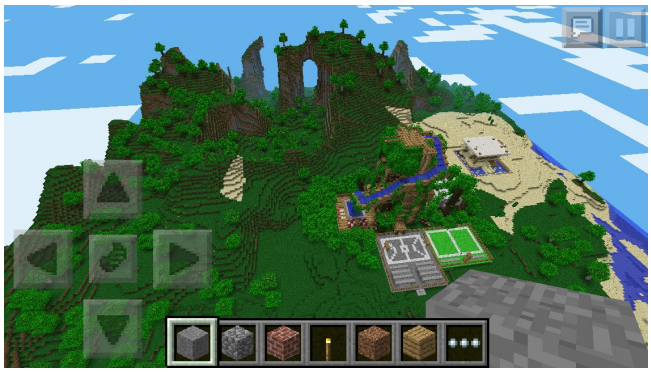
Joris Dormans, em sua tese de doutorado, propõe uma *framework* para a geração de conteúdo procedural com maior expressividade para jogos de ação e aventura. A base para a geração desse conteúdo são dois principais conceitos: missão e espaço. Missão diz respeito ao conjunto de desafios que o jogador terá de superar ao longo do jogo para finalizá-lo. Diversos tipos de objetivos podem ter o papel de missão, desde eliminar enimigos de uma região até encontrar algum item escondido em uma região. Por outro lado, espaço diz respeito ao ambiente que o personagem irá percorrer ao longo da realização dessas missões. Talvez o cenário seja mais aberto e permita a visualização dos inimigos com facilidade ou talvez o cenário seja mais fechado e os inimigos irão aparecer de forma repentina. Ambos os conceitos são próximos e podem ser facilmente confundidos devido a dependencia que um tem ao outro. Esses conceitos são facilmente transportados para uma formulação em grafos, onde diversos algoritmos podem ser aplicados para transformar esses grafos e obter estruras interessantes para as missões e o espaço do jogo.

Objetivo

Apesar de possuir uma formulação interessante, a *framework* de Joris Dormans é fortemente estruturada na operação de transformação de grafos através de regras de transformação. Esse é um problema NP-completo (Drewes *et al.*, 2020) e com pouco conteúdo disponível para guiar a implementação de algoritmos que o resolvam. A complexidade desse problema poderia gerar um trabalho próprio.

Dessa forma, uma análise sobre a viabilidade do uso dessa técnica se faz necessária. Saber os custos do uso dessa técnica é de grande valor durante a fase de planejamento e concepção dos recursos de um jogo. Por outro lado, saber a expressividade desse método auxilia na decisão de utilizar a técnica ou não para produzir certos tipos de conteúdo de forma procedural.

(a) *Mapa explorado no jogo* THE WICTHER 3 *(CD Projekt RED, 2015)*



(b) *Limite visível no mapa gerado em* MINECRAFT

Figura 1: *Diferentes mapas gerados de forma procedural.*

Metodologia

Para fazer uma análise crítica sobre o uso da *framework* de Dormans, foi desenvolvido um jogo de ação e aventura do gênero Metroidvania. Jogos no estilo Metroidvania apresentam uma estrutura que se encaixa muito bem com a técnica, pois, devido a sua estrutura de salas e portas, possuem uma transposição direta para grafos de missões, uma vez que consegue-se relacionar cada sala diferente com um nó no grafo de missão. O desenvolvimento do jogo buscou compreender os pontos de maior atrito na implementação da técnica bem como analisar o resultado final da construção do protótipo.

O desenvolvimento foi feito de forma incremental. A cada nova semana de desenvolvimento uma nova funcionalidade foi agregada ao jogo. Após cada nova função construída, foram tomadas notas sobre as dificuldades ao longo da implementação analisandos adequações feitas à estrutura do jogo bem como as limitações encontradas.

Estrutura do texto

No Capítulo 1 é feita uma análise sobre como a geração procedural funciona com jogo atualmente. No Capítulo 2 o gênero de jogo Metroidvania é explicado em mais detalhes, a fim de se justificar sua escolha para a criação do jogo. O Capítulo 3 explica em mais detalhes os conceitos de missão e espaço descritos por Dormans em sua tese de doutorado. No capitulo 4 é explicado como utilizar o método de Dormans para gerar missões de forma procedural. O Capítulo 5 mostra como o método foi aplicado para desenvolver o jogo, bem como o resultado final obtido. Por fim, no Capítulo 6 são expostos os desafios encontrados durante o desenvolvimento.

Capítulo 1

Geração procedural em jogos

A geração procedural em jogos se dá de várias formas. Elas podem ser utilizadas durante fase de desenvolvimento do jogo ou até mesmo como um recurso de design do jogo que fará parte da experência do jogador. No jogo Horizon Zero Dawn (Guerrilla Games, 2017), os desenvolvedores criaram ferramentas para os designer posicionarem vegetação no mapa de forma bastante simplificada, semelhante a pintar uma superfíci (Sanders, 2020)e. O detalhamento da vegetação então é feito de forma procedural, poupando aos designers a necessidade de se desenvolver todos as paricularidades do ambiente e se concentrar em partes mais cruciais. Nesse caso a geração procedural concede ganho de tempo aos desenvolvedores, que podem alocar recursos para o desenvolvimento de outras características do jogo em questão.

Um dos primeiros grandes jogos a utilizar geração procedural como característica fundamental de seu deisgn foi o Rogue (Michael Toy e Glenn Wichman, 1980). Esse jogo surgiu em 1980 e destacava-se por proporcionar experiências novas ao jogador a cada nova sessão, pois novas salas e novas conexões são geradas de forma procedural toda vez que o jogo é iniciado (Figura 1.1). Dessa forma, o jogo se torna interessante e desafiador até mesmo aos desenvolvedores, visto que a a cada nova jogatina, uma experência nova será gerada. Rogue se tornou tão popular que nomeou um novo gênero de jogos, os *roguelikes*. Aqui a geração procedural desempenha um papel fundamental na rejogabilidade do jogo, tornando-o divertido por mais tempo e desafiador até mesmo para os desenvolvedores do jogo.

Outros jogos que geram conteúdo de forma procedural incluem Diablo (Blizzard Entertainment, 1996 - 2012), Torchlight (Runic Games, 2009), Spore(Maxis, 2008) e Minecraft. Muito dos algoritmos utilizados nesses jogos são agoritmos de força bruta modelados para gerar estruturas compatíveis com o jogo. Em outros casos, uma grande quantidade de possíveis cenários são gerados e os mais adequados são escolhidos (Johnson *et al.*, 2010). Outros métodos apenas avaliam a cena para remover trechos inacessíveis (Johnson *et al.*, 2010). Em outros casos, a técnica é inicialmente construir um cenário com estruturas que representem pedras abaixo da superficie e cavar túneis nela a partir de uma entrada. Dessa forma, muitos caminhos podem ser gerados se conectarmos essas estruturas com mais túneis.

Existe uma infinidade de algoritmos e métodos para geração de conteúdo procedural mas muitos deles não conseguem ser reaproveitados na geração de outros tipos de jogos. Em sua tese de doutorado, Joris Dormans propõe uma *framework* para geração de conteúdo procedural que pode ser aplicado em uma diversidade de gêneros e utiliza primariamente os conceitos de missão e espaço.

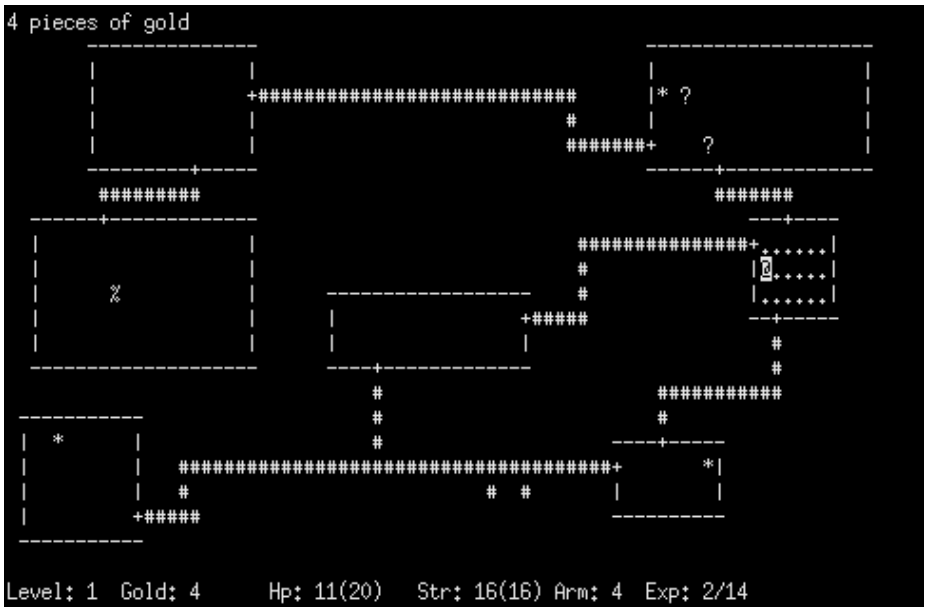


Figura 1.1: *Uma disposição de cenário gerado por Rogue*

Capítulo 2

Metroidvania

Metroidvania é um gênero de jogos eletrônicos. Geralmente são jogos de ação e aventura em 2D que possuem um grande mundo interconectado a ser descoberto pelo jogador. Sua principal característica é fato de o jogador inicialmente ser exposto a certos tipos portões e passagens inacessíveis, para só posteriormente ter acesso à habilidades e/ou chaves que permitam superar esses obstáculos. As distâncias entre chave e portões variam ao longo do jogo e não seguem um regra certa, mas em grande parte das vezes exige que o jogador retroceda parte do mapa em busca de passagens que agora consegue abrir. Isso leva a outra característica fundamental do *Metroidvania*, o *backtracking*, isso é, o jogador deve andar novamente em grande parte de território ja percorrido em busca de portas anteriormente inacessíveis, causando um sentimento de vaivém e surpreendendo o jogador as conexões possíveis entre regiões do mapa. Pode ser que o jogador enxergue um item valioso em uma região inacessível para, apenas depois de alguns momentos de exploração, se deparar com uma passagem que leve até o item visto. As possibilidades são grandes.

Os jogos que nomeiam o gênero são Metroid (Nintendo, 1986) e Castlevnia (Konami, 1986). Ambos os jogos e seus sucessores de mesmo nome fazem uso extensivo das características descritas. É natural que nesse genêro boa parte do *design* do jogo esteja contido no mapa e suas conexões. No jogo Super Metroid (Nintendo, 1994), por exemplo, seu mapa possui a configuração vista na Figura 2.1.

Outro exemplo mais recente é o Hollow Knight (Team Cherry, 2017). Hollow Knight é um jogo indepentente que segue o estilo metroidvania. Seus criadores conseguiram criar o mapa visto na Figura 2.2. Tanto Super Metroid quanto Hollow Knight possuem um mapa denso em conexões, com ciclos e bastante liberdade quanto aos caminhos a serem tomados. A construção do mapa permite ao jogador explorar o mundo de diferentes formas começando por diferentes regiões. Isso torna a jornada de cada jogador única além de proporcionar rejogabilidade.

Como esses jogos são fortemente baseados em passagens que requerem algum tipo de desbloqueio, é natural que encontremos inumeras portas ao longo do jogo. Transitar entre essas portas geralmente ocasiona na transição para outro cenário, que pode conter novos inimigos e novos itens a ser arquiridos. Assim, é bem direto criar uma relação entre os cenários em um metroidvania e um grafo. Podemos representar cada sala como um vértice e

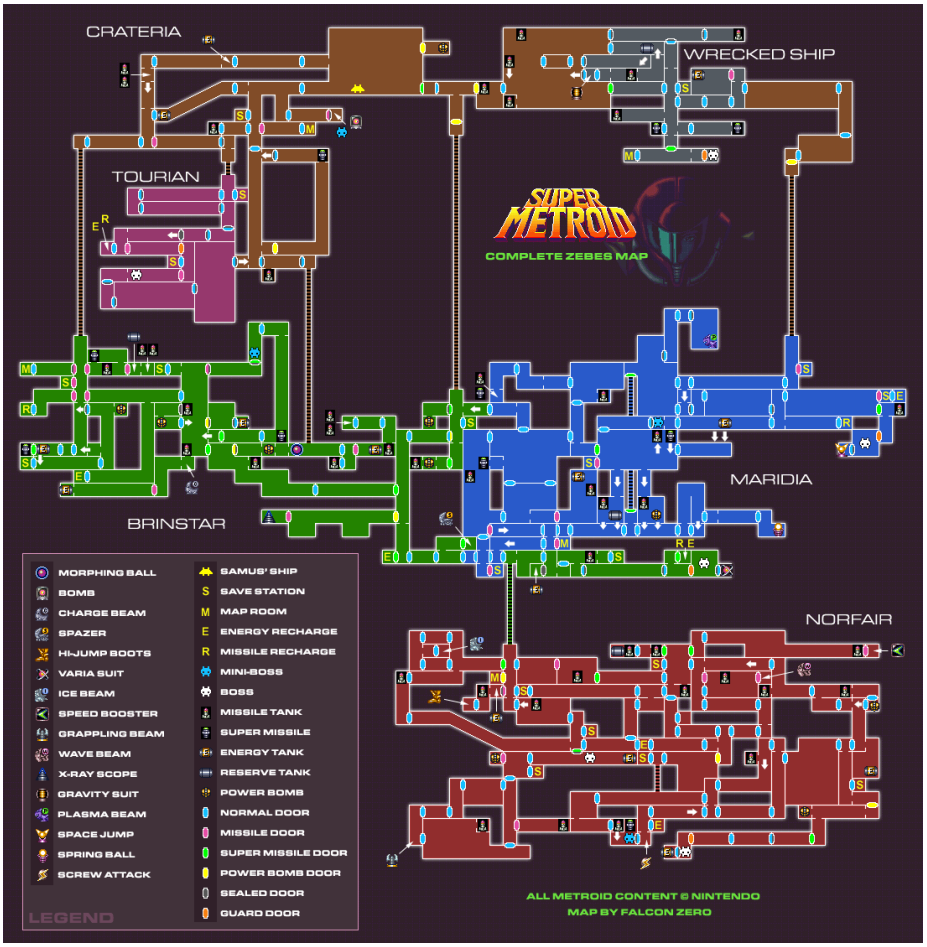


Figura 2.1: *Mapa do jogo Super Metroid. As elipses coloridas no mapa indicam portões que o personagem precisa abrir para transitar entre as salas. Cada cor exige um disparo diferente que é adquirido ao longo do jogo.*

Figura 2.2: *Mapa do jogo Hollow Knight*

cada porta como uma aresta. Tal representação é bem conveniente à *framework* proposta por Dormans, e portanto, apresenta grande potêncial de exploração para geração procedural. Principalmente por esse motivo, esse gênero foi escolhido para o desenvolvimento do jogo.

Capítulo 3

Missão e espaço

Como explicado em (Dormans, 2012), os conceitos de missão e espaço foram criados a fim de fornercer uma estrutura para organizar as mecânicas e os elementos de jogos de forma concisa. Ao fazer a divisão dessa forma os desenvolvedores podem ter uma visão mais clara sobre as responsabilidades de cada parte e como elas interagem entre sí. Dessa forma é possível modificar e experimentar os elementos do jogo de forma mais fácil, estimulando criatividade e inovação.

3.1 Missão

O conceito de missão faz referência aos objetivos que o jogador deve cumprir ao longo de sua jornada a fim de completar um jogo. Para ele, as missões tem um papel fundamental em modelar a experência do jogador, pois trazem propósito e ajudam a guiar as ações e decisões do jogador.

De acordo com Dormans, missões podem tomar diferentes formas, desde simples tarefas como coletar items e derrotar inimigos, até objetivos mais complexos como resolver quebra-cabeças e completar *quests*. Essas missões geralmente são organizadas em uma estrutura hierárquica, com pequenas tarefas servindo de guia para objetivos maiores. Isso ajuda a dividir a jornada em pedaços menores, tornando mais fácil para os jogadores se engajarem e progredirem no jogo.

Além de dar senso de propósito e estrutura, missões também fornecem um senso de progresso e relização aos jogadores. Conforme as missões são completadas, geralmente são presenteados com recompensas ou novos recursos, o que os mantém motivados a jogar e alcançar seu objetivo final. Essa sensação de progresso pode ser importante em jogos de mundo aberto, onde os jogadores possuem uma grande quantidade de liberdade para explorar e interagir com os arredores.

Um aspecto interessante no design de missões que Dormans menciona é a "ramificação de missões". Esse conceito faz alusão a ideia de tornar possível aos jogadores escolherem quais missões ou caminhos desejam seguir, onde cada caminho pode levar a diferentes consequencias. Isso pode adicionar bastante rejogabilidade ao jogo, uma vez que os jogadores são encorajados a tomar diferentes caminhos e ver o que acontece quando diferentes

escolhas são tomadas.

Ramificação de missões também acrescenta mais profundidade à narrativa do jogo, pois os jogadores são capazes de tomar decisãoes que influenciem a história e o futuro das personagens. Isso pode resultar em uma experiência mais engajante e imersiva aos jogadores, uma vez que sentem que suas ações possuem consequências no mundo explorado.

Outro conceito importante no design de missões é o "ritmo das missões". Esse conceito faz referência ao modo como as missões são espaçadas e estruturadas ao longo do jogo, e pode ter um impacto significante na experência do jogador. Se as missões são muito faceis ou muito frequentes, o jogador pode se sentir frustrado ou entendiado. Por outro lado, se as missões são muito difíceis ou infrequentes, o jogador pode se sentir desanimado e perder a motivação. Portanto, os *game designers* devem tomar cuidado com o ritmo das missões a fim de tornar a esperência do jogador mais agradável.

A estrutura das missões também pode ser usada para guiar o jogador para certas estratégias e estilos de jogo. Por exemplo, o jogo pode encorajar os jogadores a explorar os cenários através de missões que o recompensem por fazer isso. Dessa forma, as missões podem ajudar a guiar a forma de se jogar o jogo e explorar suas mecânicas.

O conceito de missão é elemento chave em vários jogos e tem papel fundamental em moldar a experência do jogador. Através de tarefas simples ou grandes objetivos, missões fornecem sentimento de propósito e realização ao jogador, conforme avançam na história. O *design* das missões, incluindo as ramificações, ritmo e estrutura geral, podem impactar de forma significativa a experiência do jogador e devem ser consideradas com atenção pelos game designers a fim de criar uma experência rica ao jogador.

3.2 Espaço

De acordo com Dormans, espaço são os ambientes ou o mundo que é explorado ao longo do jogo. Esse conceito inclui não somente os aspecto visual que os elementos trazem na experência do jogador mas também como os elementos são colocados a fim de trazer desafios e interagir de forma direta com as missões que o jogador deve cumprir.

Em muitos jogos, o espaço serve como elemento central na experiencia do jogador, influenciando diretamente na forma como jogo é jogado. O ambiente pode, por exemplo, influenciar quais estratégias o jogador deve desempenhar para completar um objetivo. Em jogos com temática furtiva como Metal Gear Solid (Konami, 1998 - 2015) diversos obstáculos são posicionados de forma que o jogador possa se esconder e se movimentar sem ser percebido. Em outros casos, o próprio ambiente pode ser o fator desafiante em uma missão, colocando obstáculos que o jogador deve superar. Jogos de plataforma, por exemplo, geralmente apresentam terrenos com diversos buracos a fim de exigir cuidado e precisão na movimentação do jogador.

Assim como as missões, o ambiente possui um peso narrativo indispensável ao jogo, trazendo a atmosfera necessária. Fazer um jogo de terror, por exemplo, pode ser muito difícil se o ambiente não proporcionar o clima necessário ao jogador. O *design* do ambiente ajuda a trazer imersão ao jogador, inserindo-o no mundo.

A partir do gênero do jogo pode-se fazer inferências sobre como o espaço se apresentará ao jogador. Para jogos de mapa aberto, por exemplo, supõe-se uma grande liberdade na exploração do ambiente, onde o jogador possui uma infinidade de caminhos possíveis de se seguir. Grande nomes desse gênero são Grand Theft Auto (1997-2022) e Assassin's Creed (2007-2022). Em outros gêneros, como *Hack and Slash*, o ambiente funciona como tuneis conectando cenários e um grau muito menor de exploração é encontrado.

De forma geral, o espaço é elemento crucial de vários jogos e tem papel fundamental em construir a experiência do jogador. Através de seu impacto nas mecânicas do jogo ou na construção da atmosfera do jogo, o espaço desempenha um papel fundamental no *game design* e está ligado de forma muito próxima com o conceito de missão.

Capítulo 4

Gerando missões de forma procedural

O método de Dormans é fortemente baseado em grafos. Através deles podemos descrever estruturas de missões por meio de objetivos menores e assim, ditar o ritmo do jogo. Em cima desses grafos é possível aplicar transformações e assim obter novas estruturas para as missões. Com um bom leque de transformações e um pequeno grafo inicial podem-se gerar diversas estruturas para as missões de um jogo.

4.1 Grafos de missões

Dormans, em sua tese, propõe uma repsentação por meio de grafos para as missões de um jogo. Os grafos de missões representam o progresso do jogador através das tarefas que ele deve completar para terminar um nível. Esse é um grafo direcionado que funciona com uma lista onde cada nó representa uma tarefa a ser executada.

Quanto ao estado, cada nó pode estar em um dos seguintes três:

1. **Disponível**. Tarefas que o jogador pode executar.
2. **Indisponível**. Tarefas que o jogador não pode executar.
3. **Completada**. Quano o jogador completou com sucesso uma tarefa.

As arestas do grafo, por outro lado, mostram como mudanças no estado de um nó altera a disponibilidade de missões subsequentes. Existem três tipos (veja figura 4.1):

1. **Requisito forte**. Indica que a tarefa destino se torna disponível apenas se a tarefa de origem ja foi executada. São representados através de uma seta dupla.
2. **Requisito fraco**. Indica que a tarefa destino se torna disponível apenas se a tarefa de origem também está disponível ou foi completada. São representados através de uma seta simples.
3. **Inibição**. Indica que a tarefa destino se torna indisponível caso pelo menos uma tarefa de origem tenha sido completada, a menos que a tarefa destino ja tenha sido

completada. São representadas por uma linha com um pequeno circulo na ponta.

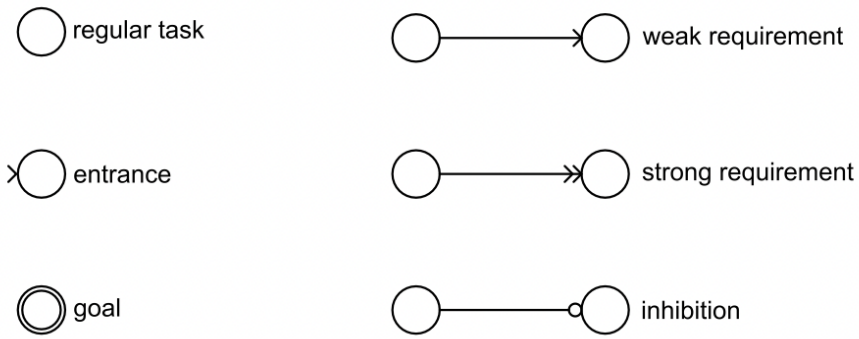


Figura 4.1: *Elementos básicos de grafos de missões . Figura retirada de Dormans (2012)*

Caso uma vértice possua muitos antecessores, seu estado é determinado da seguinte forma:

Se a tarefa está completada, seu estado não se altera mais.

Senão, se possui algum inibidor, então seu estado é colocado como indisponível se algum inibidor estiver completo.

Senão, se possui requisito forte, então seu estado é colocado como disponível quando todos os requisitos fortes estiverem completos.

Senão, se possui requisito fraco, então seu estado é colocado como disponível se pelo menos algum requisito estiver completo ou disponível.

Senão, seu estado é indisponível.

Existem 3 tipos de vértices no grafo de missão (veja figura 4.1):

1. **Normal**. Representados por circulos simples. Cores e letras podem ser usados para diferenciar qual tipo de tarefa se trata.

2. **Entrada**. Representada por circulos com uma ponta de seta apontando para ela. O grafo de missão precisa ter pelo menos um vértice desse tipo. Esses vértices não podem ter pre-requisitos.

3. **Objetivo**. Representada por circulos com contorno duplo. O grafo de missão precisa ter pelo menos um vértices desse tipo. Não podem ser requisitos e nem inibidores de outras taredas. Quando completada a missão é cumprida.

A Figura 4.2 mostra como se parece um grafo de missões com progresso no estado das tarefas. Os vértices em cinza representam as tarefas indisponíveis, vértices brancos vazios representam as tarefas disponíveis e os vértices marcados em verde representam as tarefas ja completadas.

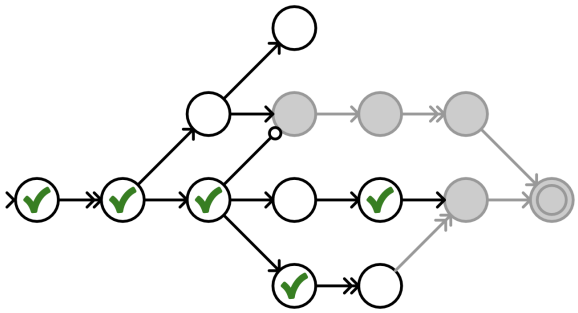


Figura 4.2: *Grafo de missões representando estado e progresso . Figura retirada de* Dormans *(2012)*

4.2 Gramática de grafos

Gramáticas formais são originadas da linguística e são usadas para descrever conjuntos de frases que constituem as linguagens naturais. Em linguagem formal, uma linguagem é um conjunto de palavras. Uma gramática é uma caracterização finita de uma linguagem. Uma gramática formal para uma linguagem formada por sequências de símbolos possui quatro elementos:

1. Um conjunto finito de **terminais** que a linguagem irá produzir. Esse conjunto também é chamado de alfabeto. Geralmente são representados por letras minúsculas. Por exemplo: $\{a, b\}$.

2. Um conjunto finito de **não-terminais** que não são elementos da linguagem que a gramática irá produzir. As regras da gramática irão substituir não-terminais por terminais. Geralmente são representados por letras maiúsculas. Por exemplo: A, B, S.

3. Um símbolo do conjunto de não-terminais que é o **símbolo-inicial**. É usual denotá-lo por S.

4. Um conjunto finito de **regras gramaticais** (ou regras de reescrita). Regras gramaticais possuem a forma $\alpha \implies \beta$ onde α e β são palavras compostas por terminais e não terminais. Cada regra especifica uma sequencia de simbolos (lado esquerdo) que pode ser dubstituida por outra sequencia de símbolos (lado direito).

Gramáticas de grafos que geram grafos consistindo de vértices e arestas foram descritas por Rekers e Schürr (1995). Sistemas de reescrita para grafos para gramáticas de grafos foram discutidas por Heckel (2006). Numa gramática de grafos, uma parte do grafo pode ser substituída por outra estrutura, como exemplificado pelas Figuras 4.3 e 4.4. Uma vez que a os vértices são selecionados pelos critérios da regra, eles são numerados de acordo com o lado esquerdo da regra de reescrita (passo 2, Figura 4.4). Em seguida, todas as arestas entre os vértices selecionados são apagadas (passo 3). Os vértices números são então substituídos pelos vértices equivalentes (de mesmo número) na regra (passo 4). Então os vértices da regra que não possuem equivalentes são adicionados ao grafo (passo 5). Por fim, os novos vértices são conectados pelas arestas indicadas na regra (passo 6) e os números dos vértices são removidos (passo 7).

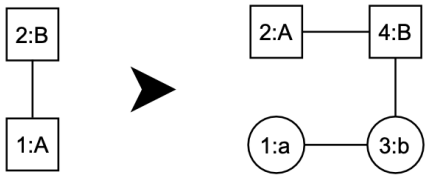


Figura 4.3: *Regra de reescrita para grafo. Quadrados determinam não-terminais e circulos determinam os terminais. Figura retirada de Dormans (2012)*

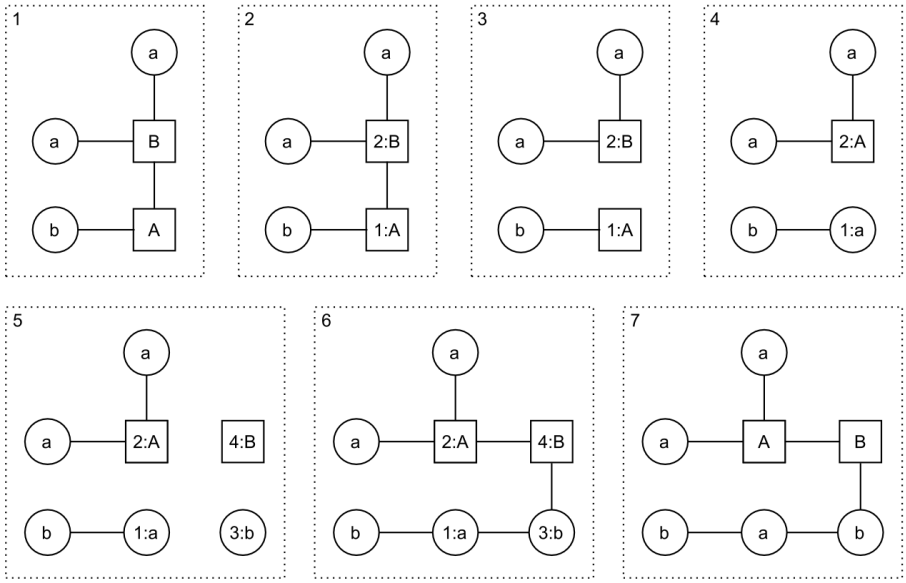


Figura 4.4: *O processo de aplicar uma regra a um grafo. Figura retirada de Dormans (2012)*

4.3 Gerando missões

Para gerar as missões de forma procedural começamos com um pequeno grafo de missões como base. Usamos então essa estrutura para aplicar diversas transformações por meio das regras de reescrita. Os critérios para decidir quando parar de aplicar as transformações podem ser diversos. Pode-se, por exemplo, gerar estruturas até que um número de vértices seja alcançado. Em outros casos, pode-se utilizar como critério de parada a existência de certos recursos ou tipos de missões. De forma mais simples, pode-se utilizar como critério, a aplicação de n transformações.

Um tipo de regra de reescrita que gera resultados interessantes para jogos são as regras envolvendo chaves e portas. Essencialmente, o que esse tipo de estrutura permite é fazer com que o designer modifique uma estrutura linear de missões, tornando-a mais ramificada (Figura 4.5), favorecendo a criação de missões não lineares.

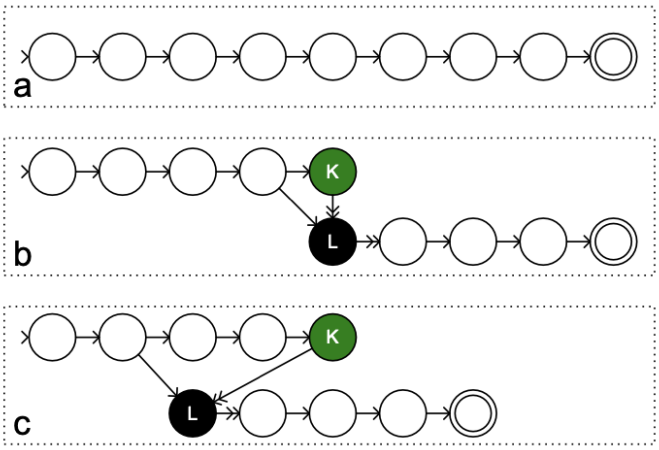


Figura 4.5: *Adição de chaves e portas em uma estrutura linear (a) pode torná-la mais ramificada (b), permitindo o deslocamento da chave (c). Figura retirada de* Dormans *(2012)*

Capítulo 5

Desenvolvimento do jogo

Para a aplicação do método de Dormans foi necessário pensar em como resolver cada problema de uma vez. Inicialmente criou-se as estruturas de dados para representação dos grafos. Em seguida, outra estrutura foi construida para mapear os dados do grafo para a cena. Por fim, os algoritmos para aplicar as regras de reescrita foram feitos. Tudo isso foi desenvolvido por meio de métodos ageis, trazendo a cada nova versão uma característica nova ao jogo.

5.1 Metodologia

O jogo foi feito utilizando-se metodologia ágil. As *sprints* iniciais foram dedicadas ao desenvolvimento de funcionalidades básicas do jogo como movimentação, animação e conexão entre diferentes salas. As *sprints* finais foram dedicadas ao desenvolvimento da lógica para a geração de missões procedurais. Ao final de cada *sprint* as dificuldades eram expostas e os próximos passos eram definidos em conjunto com o orientador do projeto.

Todo o desenvolvimento foi salvo e documentado através de um repositório no github disponível em https://github.com/rogeriomfneto/proceduroidvania. O trabalho final foi postado em uma página no itch.io disponível em https://rogeriomfneto.itch.io/proceduroidvania.

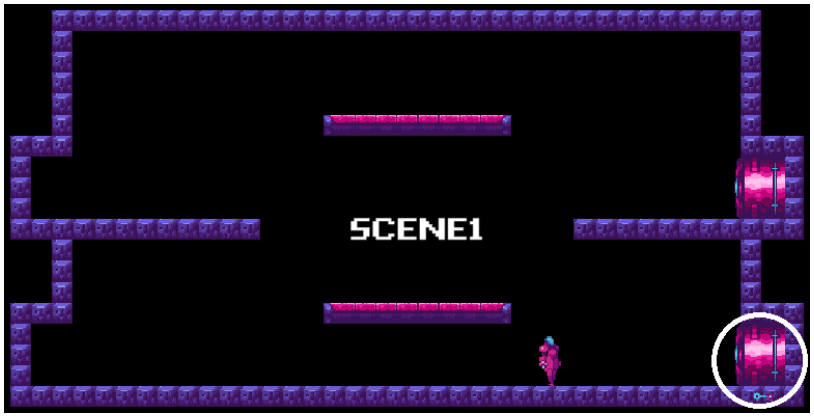
5.2 Descrição do jogo

Dentre as potencialidades do método de Dormans, gerar missões de forma procedural é o que de fato se adequa mais aos conceitos do metroidvania. Portanto, teve-se como objetivo criar um jogo que explora a geração procedural de missões, isso é, que gera de forma procedural estruturas interessantes que envolvam encontrar chaves e abrir portas, levando-se em consideração a característica fundamental dos metroidvanias: ter contato com um certo tipo de fechadura antes de ter os recursos adequados para abrí-la.

Trata-se de um jogo 2D, de ação e aventura no gênero metroidvania. O objetivo do jogo

é chegar na sala 3 (ou *scene 3*). Para alcançar a sala 3 o jogador deve percorrer inumeras outras salas que se diferenciam apenas pela presença de chaves, quantidade de portas e o código da sala estampado ao centro da cena. Para percorrer as salas o jogador pode andar e pular. Cada sala pode ter quatro portas no máximo, duas à esquerda e duas à direita. Além disso, uma sala pode ter uma chave que abre alguma porta no mapa. Cada chave abre uma e somente uma porta e é posicionada na plataforma superior da sala (Figura 5.1a). As chaves são diferenciadas por sua cor e formato e as portas são identificadas pela figura da chave logo abaixo dela (Figura 5.1b). Caso não haja nenhuma figura a porta não é bloqueada por nenhuma chave.

(a) *Chave azul encontrada na sala 8, circulada em branco.*



(b) *Porta azul encontrada na sala 1, circulada em branco*

Figura 5.1: *Recursos de chaves e portas*

5.2.1 Visual

O visual escolhido para o jogo vem do pacote de *assets* Warped[1]. O pacote conta com diversas animções e estruturas para criação de cenário, o que permite uma boa implementação de mecânicas ao jogo. Além disso, o visual é em pixelart e fortemente inspirado nos antigos jogos da série Metroid (Nintendo, 1986 - 2021), como Super Metroid(Nintendo, 1994) e Metroid Fusion (Nintendo, 2002).

[1] Disponível em: https://ansimuz.itch.io/warped-caves

5.2.2 Ferramentas usadas

Para o desenvolvimento do jogo foi utilizada a *engine* Unity[2]. *Game engines* são plataformas e/ou um conjunto de bibliotecas que permitem desenvolver um jogo de forma mais fácil (Gregory, 2019). Através dela, é possível, por exemplo, usufruir de um sistema de física ja pronto, o que dispensa o gasto de tempo com esse tipo de particularidade e permite ao desenvolvedor focar seus esforços em outros aspectos do jogo. Existem diversas engines disponíveis e amplamente usadas pela comunidade de desenvolvedores de jogos. Para nomear algumas, pode-se citar *Godot*[3], *Unreal*[4] e *Game Maker*[5], por exemplo. Uma engine pode ser especializada para facilitar a construção de certos tipos de gêneros de jogos (como, por exemplo, *RPG Maker*[6]) ou ser de uso geral, para construir qualquer tipo de jogo (como, por exemplo, *Unity*).

Essa engine foi escolhida principalmente por sua popularidade e pela já familiaridade do autor deste trabalho com a interface dessa ferramenta (Figura 5.2). O Unity construiu uma grande comunidade de usuários ao longo dos anos o que torna mais fácil encontrar respostas para possíveis problemas e ajuda online e, além disso, possui uma boa documentação.

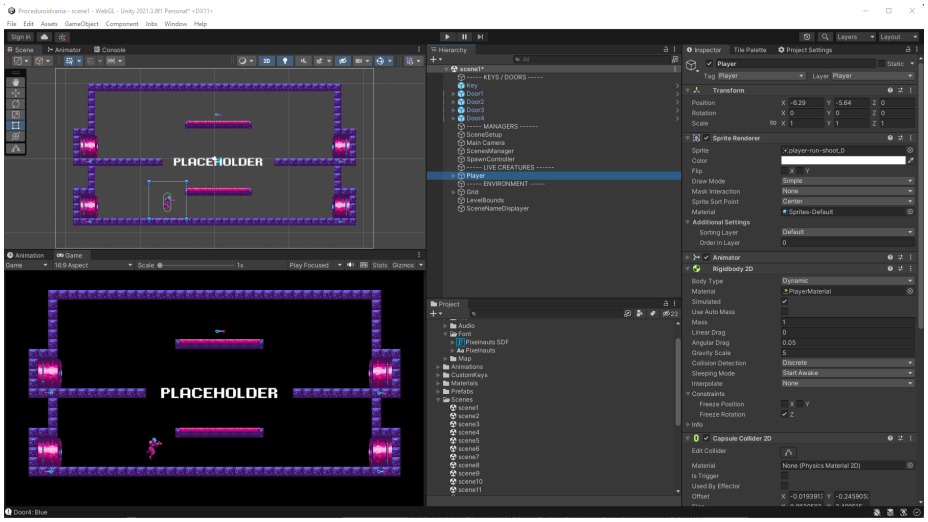


Figura 5.2: *Interface da game engine Unity*

O *Unity* implementa um sistema bastante popular no desenvolvimento de jogos, o chamado Entity-Component-System (ECS) (Nystrom, 2014). Nesse sistema, os objetos do jogo – as entidades – possuem pouco ou nenhum comportamento, e vão sendo incrementados por meio da inserção de componentes. Componentes, por sua vez, encapsulam diversos tipos de comportamentos e podem ser aplicados a diversos objetos. Nesse contexto, os objetos do jogo funcionam, em essência, como contâineres nos quais as componentes são aplicadas para desenvolver comportamento. Esse tipo de arquitetura favorece a composição

[2] Mais informaçoes em https://unity.com/

[3] Mais informações em https://godotengine.org/

[4] Mais informações em https://www.unrealengine.com/pt-BR

[5] Main informções em https://gamemaker.io/en

[6] Mais informações em https://www.rpgmakerweb.com/

em oposição à herança, o que faz com que os objetos sejam definidos não por hierarquia, mas sim pelos componentes associados à ele.

5.3 Criação do grafo de missões

Para a geração procedural utilizada nesse jogo foi utilizada uma simplificação em relação ao modelo proposto por Dormans. Aqui cada missão equivale a uma sala diferente pela qual o personagem precisa passar ou coletar um item. Além disso, em vez de representar uma porta por um vértice, as portas são representadas pelas próprias arestas do grafo. Caso uma porta seja bloqueada por uma chave, anotamos acima da aresta qual tipo de chave bloqueia aquela porta. Dessa forma, cada vértice identifica uma sala, e suas arestas representam as portas dessa sala. Nessa formulação as missões se confudem com o mapa em si, tendo uma relação direta em como as missões são conectadas e como elas são conectadas no mapa. Dormans originalmente não confunde espaços com missões dessa forma, mas os gera de forma independente.

Ao atravessar uma porta que está localizada à direita de uma sala, é natural esperar que o personagem apareça à frente de uma porta à esquerda na sala destino (Figura 5.3). Analogamente, ao atravessar uma porta à esquerda, é esperado que o personagem apareça em uma porta a direita na sala destino. Para resolver esse tipo de problema, foi determinado que cada vértice no grafo de missões possui no máximo dois arcos de entrada e dois arcos de saída. Os arcos de entrada são mapeados para portas localizadas à esqueda da sala e arcos de saída são mapeados em portas à direita da sala.

Para gerar a sequência de salas no mapa, inicialmente tem-se um grafo de apenas três vértices conectados diretamente (Figura 5.4). Em cima desse grafo são aplicados quatro tipos de transformações, que podem ser vistos na Figura 5.5:

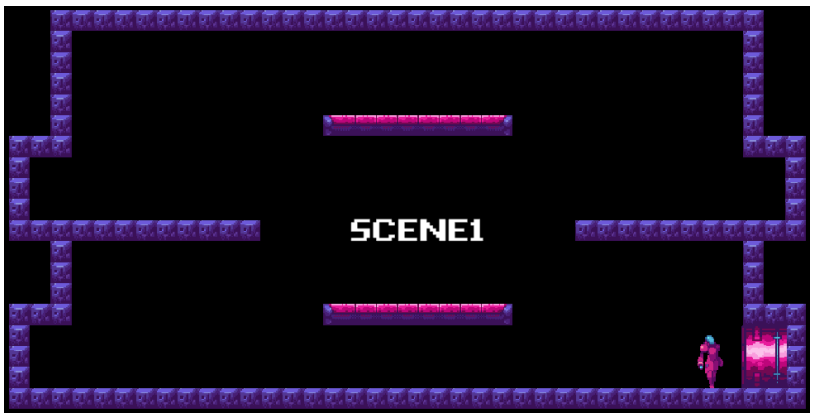
1. A primeira regra (Figura 5.5a) pega um par de vértices conectados através de uma porta simples onde o segundo vértice não contém uma chave e adiciona um vértice entre eles. Essa regra permite ter um maior volume de vértices (missões) no grafo.

2. A segunra regra Figura 5.5b) pega um par de vértices conectados através de uma porta simples, bloqueia a porta através de uma chave e disponibiliza a chave através de uma sala conectada ao primeiro vértice. Ela quebra a linearidade das missões uma vez que agora é necessário desviar do caminho para coletar uma chave. No jogo são permitidas no máximo quatro chaves com as cores vermelho, verde, azul e amarelo.

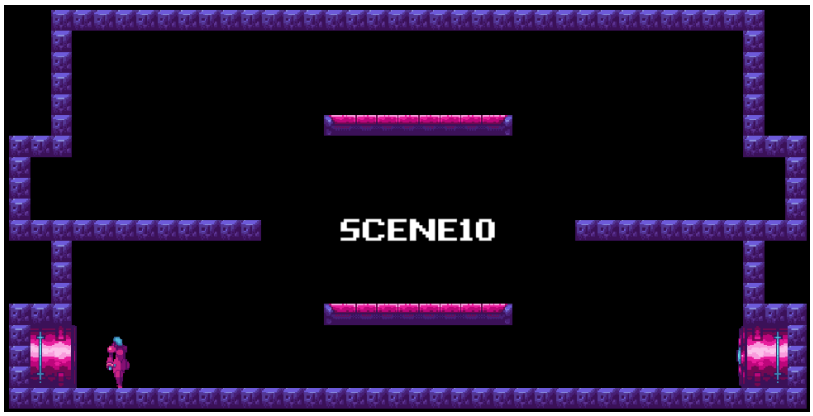
3. A terceira regra (Figura 5.5c) pega um par de vértices conectados por uma porta simples onde o segundo vértice representa uma sala contendo uma chave e adiciona um vértice entre eles. Essa regra prolonga as ramificações, uma vez que agora o caminho para alcançar uma chave é mais longo.

4. A quarta regra pega dois vértices não conectados onde o segundo vértice não contém uma chave e os conecta através de uma porta simples. Essa missão cria pontes através das missões, permitindo mais de um caminho para chegar em certos pontos do mapa, além de atribuir uma certa característica de labirinto.

As três primeiras regras acrescentam vértices, a última acrescenta arestas. Para gerar



(a) *Entrando em um porta à direita.*



(b) *Saindo em uma porta à esquerda*

Figura 5.3: *Fluxo de movimento indo de uma porta à direita em direção a uma porta à esquerda*

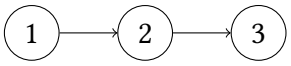
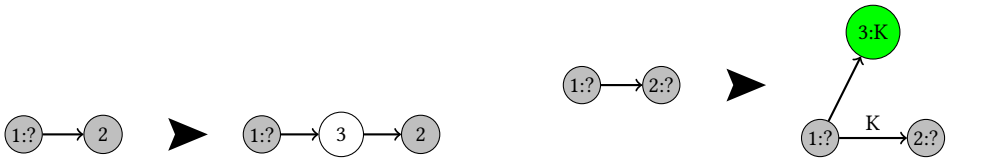


Figura 5.4: *Grafo inicial para geração das missões*



(a) *adiciona vértice entre dois vértices quaisquer*

(b) *adiciona uma chave entre dois vértices quaisquer*

(c) *adiciona um vértice entre uma chave e um vértice de origem*

(d) *adiciona uma aresta entre dois vértices quaisquer*

Figura 5.5: *Regras de reescrita utilizadas na geração das missões no jogo*

as missões do jogo, as três primeiras regras são aplicadas sucessivamente até obter-se dezesseis salas no jogo. Por fim a última regra é aplicada quatro vezes para obter-se mais complexidade nas interligações no grafo.

5.4 Implementação do método de Dormans

A *framework* proposta por Dormans promete uma grande espressividade, permitindo atender uma vasta gama de jogos, inclusive com complexidades narrativas, como são os jogos de aventura. Apesar de ter grande expressividade, ela traz dúvidas quanto à sua facilidade de implementação, uma vez que utiliza de modelagens em grafos que podem não ser obvias à primeira vista.

O método utilizado para encontrar os padrões de cada regra no jogo foi de força bruta. Todos os vértices do jogo são percorridos a fim de se buscar pelo padrão definido na regra. Cada regra é representada por uma classe que implementa dois métodos: `findMatch` e `applyTransformation` (Figura 5.6). O primeiro método encontra todas os conjuntos de vértices no grafo que satisfaçam o critério da regra. O segundo método, por outro lado, é reponsável por escolher uma de todas opções possíveis fornecidas pelo primeiro e aplicar a transformação da regra.

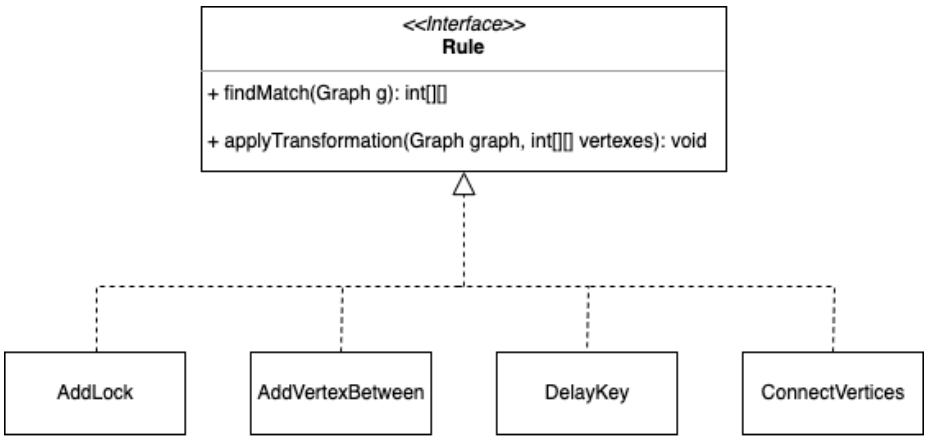


Figura 5.6: *Diagrama de classe das regras implementadas na criação do jogo em UML*

5.5 Mapeamento do grafo de missões para cenas

Após o grafo de missões ser gerado, as missões são então mapeadas em uma estrutura de dados que armazena os metadados para cada uma delas como, por exemplo, a presença de uma chave naquela sala, e para qual outra sala cada porta presente na sala deve levar.

Dessa forma, cada sala segue um modelo padrão (figura 5.7) que permite o posicionamento de quatro portas e a presença de uma chave no centro da sala. Ao adentrar uma nova sala os metadados são consultados e a sala é alterada para representar as características descritas pela estrutura de dados. Todo esse processo acontece em poucos milissegundos.

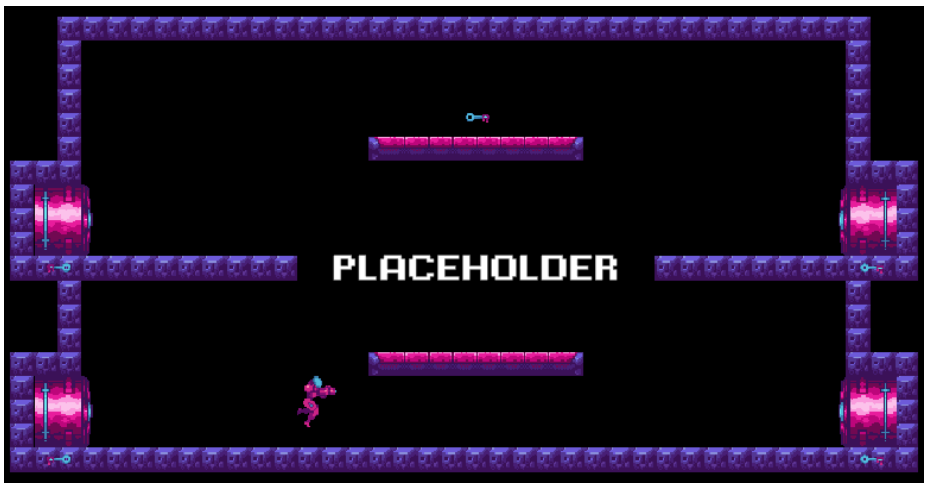


Figura 5.7: *Modelo padrão de sala utilizado por todas as salas do jogo*

5.6 Resultado

O programa final gera mapas interessantes com bom número de conexões. O jogo adquire um visual muito semelhante a um labirinto, com salas parecidas forçando o jogador a memorizar locais pelos quais já passou. A Figura 5.8 apresenta um mapa gerado durante uma iteração qualquer do jogo. Nele é possível ver os ciclos gerados pelo posicionamento de arestas aleatórias e também como as salas contendo as chaves se distanciam de suas respectivas portas em alguns casos.

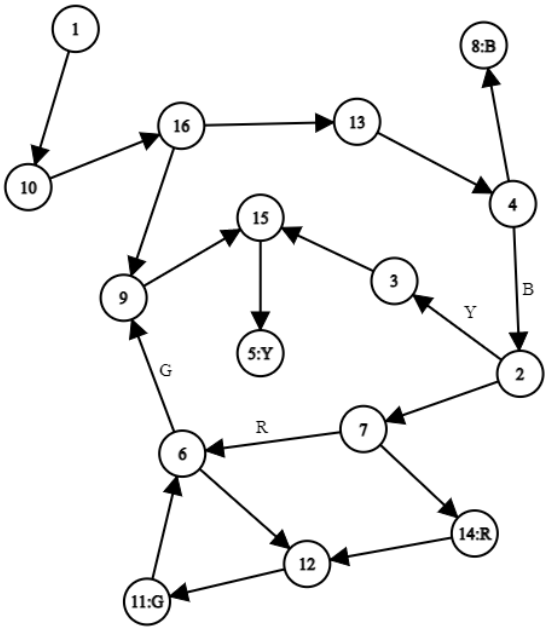


Figura 5.8: *Mapa gerado pelo jogo. As letras R, G, B, Y identificam as chaves de cor vermelha, verde, azul e amarela respecticamente. Quando posicionada acima de uma aresta significa que a aquela porta está bloqueada por uma chave da cor indicada. No rótulo de um vértice, ela indica que aquela sala contém a chave da cor indicada. O jogo se inicia no vértice 1 e termina e o objetivo é chega no vértice 3. Toda as portas são de via dupla, a direção dos arcos serve para indicar se uma porta está localizada à direita (aresta saindo) ou à esquerda (aresta entrando) de uma sala.*

Capítulo 6

Conclusão

O método apresentado por Dormans para a geração de jogos de forma procedural é poderoso e serviu muito bem a proposta de jogo. O gênero metroidvania se adequou com poucas adaptações às caracterizações de grafos de missões e regras de reescrita para geração das missões. Com quatro regras foi possível gerar estruturas de salas desafiantes em que o jogador deve tomar caminhos fora do ramo principal para encontrar chaves e desbloquear seu caminho.

Bibliotecas que fizessem o processamento e aplicação das regras foram procuradas mas não foi achada nenhuma que possuísse uma integração de fácil uso com C# (linguagem utilizada pela Unity). A implementação do método por meio de herança funciona muito bem com transformações simples, mas critérios mais sofisticados, com grande quantidade de vértices, envolvendo ciclos e/ou outras estruturas em grafos podem ficar mais complicados de serem feitos com pura força bruta, além de trazer danos ao desempenho do programa. Nesse caso uma biblioteca específica para aplicação de transformações em grafos pode ser encorajada.

Os tipos de estruturas utilizadas na criação do jogo são basicamente portas e chaves. Mais recursos podem ser utilizados como a presença de inimigos, salas com formatos diferentes, portas que permitam fluxo em apenas em um sentido, etc. Com a implementação atual, cada um desses tipos de característica tem que ser mapeado como um tipo para um vértice e posteriormente ser mapeado como um metadado de uma sala específica. A cada novo metadado necessário para a construção do mapa, mais complexidade é adicionada às regras de transformação do grafo e mais esforço é necessário na construção das salas do jogo. Portanto o custo da implementação de novos recursos procedurais deve ser considerado na concepção do projeto.

O curso de ciência da computação foi fundamental para a compreensão de desenvolvimento do projeto. O desenvolvimento de jogos atualmente é fortemente baseado em orientação a objetos. Personagens e recursos dos cenários são objetos que devem expor, através de interfaces, os comportamentos desejados para a interação mútua e composição do jogo. Toda a formulação da técnica empregada é fortemente baseada em teoria de grafos e estruturas de dados além de se fundamentar em conceitos sobre autômatos.Tais conceitos são trabalhados com maestria desde o começo do curso e são desenvolvidos ao longo dos

anos.

Trabalhos futuros

Apesar de já trazer resultados interessantes, dois conceitos imediatos podem ser aplicados ao jogo para maior avaliação da expressividade da *framework*: inimigos e portas com apenas um sentido.

Trazer inimigos ao jogo permite testar a geração procedural com alguns parâmetros básicos como por exemplo presença ou não de inimigos, quantidade e movimentação deles. Através da quantidade pode-se determinar a dificuldade de um trecho. A movimentação, por outro lado, pode determinar quais areas do mapa em que se deve ter mais cuidado ao se movimentar.Inimigos são elementos fundamentais em jogos de aventura e testar sua viabilidade pode trazer grande valor.

No desenvolvimento desse jogo, foram utilizadas apenas portas que permitem a movimentação para ambos os lados uma vez que a porta esteja desbloqueada. Um outro tipo de portas possível são aquelas que permitem movimentação apenas em um sentido. Isso pode ocorrer porque após entrar numa sala o personagem cai num buraco que não permite retorno ou porque algum dispositivo de bloqueio é acionado. Tal tipo de porta pode trazer outra dimensão à geração do grafo de missões e trazer estruturas ainda mais interessantes ao jogo.

Foi muito difícil encontrar uma biblioteca de facil integração que fizesse o processamento dos grafos de forma eficiente. Dedicar o tempo necessário para a construção de tal biblioteca pode trazer mais poder ao método e facilitar o processo de aplicação dessa técnica de desenvolvimento procedural.

Referências

[Dormans 2012] Joris Dormans. "Engineering emergence. Applied Theory for Game Design". Tese de dout. University of Amsterdan, 2012 (citado nas pgs. vii, 11, 16–19).

[Drewes *et al.* 2020] Frank Drewes, Berthould Hoffman e Mark Minas. "Graph parsing as graph transformation". Em: *Lecture Notes in Computer Science* (2020) (citado na pg. 2).

[Gregory 2019] Jason Gregory. *Game Engine Architecture*. CRC Press, 2019 (citado na pg. 23).

[Heckel 2006] Reiko Heckel. "Graph transformation in a nutshell". Em: *Electronic Notes in Theoretical Computer Science* (2006) (citado na pg. 17).

[Johnson *et al.* 2010] Lawrence Johnson, Georgios N. Yannakakis e Julian Togelius. "Cellular automata for real-time generation of infinite cave levels." Em: *Proceedings of the Foundations of Digital Games Conference* (2010) (citado nas pgs. 2, 5).

[Nystrom 2014] Robert Nystrom. *Game Programming patterns*. Genever Benning, 2014 (citado na pg. 23).

[Rekers e Schürr 1995] Jasper Rekers e Andy Schürr. "A graph grammar approach to graphical parsing". Em: *Proceedings of the 11th International IEEE Symposium on Visual Languages* (1995) (citado na pg. 17).

[Sanders 2020] Gilbert Sanders. *Between Tech and Art: The Vegetation of Horizon Zero Dawn*. Youtube. 2020. url: https://www.youtube.com/watch?v=wavnKZNSYqU (citado na pg. 5).

[Schell 2020] Jesse Schell. *The art of game design: a book of lenses*. CRC Press, 2020 (citado na pg. 1).

[Short e Adams 2017] Tanya X. Short e Tarn Adams. *Procedural Generation in Game Design*. CRC Press, 2017 (citado na pg. 2).