Universidade Federal de Juiz de Fora
Instituto de Ciências Exatas
Bacharelado em Sistemas de Informação

Geração procedural de quebra-cabeças em jogos *roguelike*

Marcus Vinícius Vasconcelos de Almeida Cunha

JUIZ DE FORA
DEZEMBRO, 2023

Geração procedural de quebra-cabeças em jogos *roguelike*

Marcus Vinícius Vasconcelos de Almeida Cunha

Universidade Federal de Juiz de Fora
Instituto de Ciências Exatas
Departamento de Ciência da Computação
Bacharelado em Sistemas de Informação

Orientador: Igor de Oliveira Knop
Coorientador: Marcelo Caniato Renhe

JUIZ DE FORA
DEZEMBRO, 2023

Geração procedural de quebra-cabeças em jogos *roguelike*

Marcus Vinícius Vasconcelos de Almeida Cunha

MONOGRAFIA SUBMETIDA AO CORPO DOCENTE DO INSTITUTO DE CIÊNCIAS EXATAS DA UNIVERSIDADE FEDERAL DE JUIZ DE FORA, COMO PARTE INTEGRANTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE BACHAREL EM SISTEMAS DE INFORMAÇÃO.

Aprovada por:

Igor de Oliveira Knop
D.Sc. Modelagem Computacional

Marcelo Caniato Renhe
D.Sc. Engenharia de Sistemas e Computação

Luciana Conceição Dias Campos
D.Sc. Engenharia Elétrica

Luiz Maurílio da Silva Maciel
D.Sc. Engenharia de Sistemas e Computação

JUIZ DE FORA

12 DE DEZEMBRO, 2023

Resumo

Os *roguelikes* são jogos nos quais o jogador precisa atravessar uma masmorra com inimigos, tesouros e cenários gerados aleatoriamente. O caráter aleatório da geração de elementos deste gênero de jogos se dá por técnicas de Geração Procedural de Conteúdo (GPC). Assim como demais elementos de um *roguelike*, *puzzles* (ou quebra-cabeças) também podem ser gerados por GPC. Os trabalhos na literatura sobre geração de quebra-cabeças normalmente se concentram em um único tipo, como resolução de labirintos, navegabilidade, *sokoban* e outros, mas pouco se fala da integração desses diversos modelos de desafios. Neste trabalho, será abordado como vários modelos de *puzzles* podem ser incorporados a jogos *roguelike* e técnicas diversas para GPC aplicadas em cada um desses *puzzles*. O resultado é a criação de um jogo expansível para uso acadêmico contendo *puzzles* diversos totalmente criados por técnicas de GPC. A avaliação é feita com jogadores reais por meio de formulários. Os dados deram indícios de uma boa aceitação dos *puzzles* gerados quanto à sua dificuldade, entendimento e variedade.

Palavras-chave: roguelike; desenvolvimento de jogos; geração procedural de conteúdo; quebra-cabeças.

Abstract

Roguelikes are games in which the player must traverse a dungeon with enemies, treasures, and randomly generated scenarios. The random character of the generation of elements in this genre of games is due to techniques of GPC. Like other elements of a *roguelike*, puzzles can also be generated by GPC. Works in the literature on puzzle generation typically focus on a single type, such as maze resolution, navigability, *sokoban*, and others, but little is said about the integration of these diverse models of challenges. In this work, we will address how various models of puzzles can be incorporated into *roguelike* games and various techniques for GPC applied to each of these puzzles. The result is the creation of an expandable game for academic use containing various puzzles entirely created by GPC techniques. The evaluation is done with real players through forms. The data gave indications of good acceptance of the generated puzzles regarding their difficulty, understanding, and variety.

Keywords: game development; procedural content generation; puzzles.

Agradecimentos

A minha filha Helena, por ter sido sempre minha maior fonte de encorajamento durante todos esses anos.

Aos professores Igor Knop e Marcelo Caniato pela orientação, amizade, paciência, sem a qual este trabalho não se realizaria, e pela liberdade criativa no desenvolvimento do projeto, o que rendeu pelo seu resultado final grande satisfação.

Aos professores do Departamento de Ciência da Computação pelos seus ensinamentos e aos funcionários do curso, que durante esses anos, contribuíram de algum modo para o nosso enriquecimento pessoal e profissional.

Conteúdo

Lista de Figuras

Lista de Abreviações e Siglas

BFS Busca em Largura. 44

GPC Geração Procedural de Conteúdo. 1, 2, 10–13, 16–18, 22, 25–29, 31, 32, 34, 35, 62–64

MCTS *Monte Carlo Tree Search*. 22, 45, 46, 63

NPC *Non-Player Character*. 20, 25, 33

PCGLab *Procedural Content Generation Lab*. 12, 25, 64

RPG *Role-Playing Game*. 10, 20, 63

1 Introdução

1.1 Apresentação

No ano de 2022, o mercado de jogos para PC e console enfrentou sua primeira retração em mais de uma década. Em comparação com 2021, o tempo de jogo geral caiu quase 15% e os gastos diminuíram na comparação entre os anos. No entanto, essas tendências não significam um problema sério para a indústria de jogos. Em vez disso, observou-se um retorno em relação aos níveis de engajamento pré-COVID e aponta-se que o número de jogadores aumentará, assim como o gasto médio com a compra de novos jogos (SIMON, 2023).

No Brasil, há um mercado em pleno crescimento e cada vez mais diverso e plural, contando com o crescimento dos *eSports* (esportes digitais) como motor para acelerar ainda mais o desenvolvimento dessa importante indústria. De acordo com um estudo publicado pela Abragames (Indústria Brasileira de *Games*), o mercado dos *games* no Brasil apresentou um aumento de 152% somente no ano de 2022. Em uma visão mais ampla, a expectativa é de que o mercado global dos jogos eletrônicos chegue ao faturamento de 200 bilhões de dólares em 2023 (PEREIRA, 2023).

Dada a sua expansão e importância no mercado, jogos digitais vêm empregando diversos tipos de profissionais. Na área de produção e talento, que é onde ocorre o desenvolvimento do jogo em si, podemos destacar *game designers*, programadores, produtores de jogos, roteiristas, engenheiros de som e testadores de *games* (Centro Universitário Tiradentes, 2023; Escola de Comunicação e Design Digital - Instituto Infnet, 2023).

Dentro da experiência do jogador, é importante que quem está jogando esteja em um estado de *flow*, um estado mental em que há tanto prazer e conexão que não se vê o tempo passar (COSTA, 2021). Para atrair e manter o jogador neste estado, é necessário que o jogo possua um *game design* bem desenvolvido e estruturado.

Game design é o processo que consiste em imaginar o jogo, definir seu funcionamento, descrever os elementos conceituais, funcionais, artísticos e outros que o compõem

e transmitir à equipe que irá construí-lo todas essas informações. Nem todo jogo é pensado da mesma maneira e, por isso, cada gênero de jogo tem uma maneira diferente de desenvolver seu *design*.

Os *puzzles* (ou quebra-cabeças) são por si só um gênero de jogo, exigindo lógica, reconhecimento de padrões e habilidades de resolver problemas complexos. Também podem auxiliar outros gêneros como uma maneira de tornar o jogo mais instigante ao jogador. No gênero *Role-Playing Game* (RPG), em que há foco em desenvolvimento de personagens e o objetivo é guiá-los por diversos desafios, é muito comum os *puzzles* conterem elementos narrativos e enriquecerem a história do jogo.

1.2 Contextualização

Este trabalho explora o gênero *roguelike*, que se trata de um subgênero de jogos RPG, em que o jogador precisa atravessar uma masmorra com inimigos, tesouros e cenários aleatorizados. Mais especificamente, o trabalho foca em como *puzzles* podem ser aplicados a este gênero de jogo digital. Os *puzzles* são uma forma de enriquecer a jogabilidade ao fazer o jogador parar para pensar em como resolver um problema. Podem estar tanto visíveis e explícitos, quando o jogador os reconhece como um obstáculo (por exemplo, uma sessão de *sokoban*[1] que precisa ser resolvida para prosseguir no jogo), como também enraizados na jogabilidade e narrativa, como acontece nos jogos *Tomb Raider*[2] e *Uncharted*[3], tornando-se quase que invisíveis (VENCO; LANZI, 2021).

O aspecto aleatório do gênero *roguelike* é trabalhado por meio de técnicas de Geração Procedural de Conteúdo (GPC). A GPC consiste em, algoritmicamente, gerar conteúdo de forma automática e aleatória, com o mínimo possível de interferência do projetista. Assim como os diferentes gêneros de jogos exigem técnicas distintas de geração procedural (TOGELIUS et al., 2011), diferentes tipos de *puzzle* trabalham com diferentes técnicas de GPC, tal como explica Kegel e Haahr (2019a). As técnicas variam em suas características, como, por exemplo, na exigência de alguma entrada de dados, na

[1] *Sokoban* é um jogo de quebra-cabeça baseado em uma grade 2D. O jogador pode se mover em quatro direções e empurrar uma única caixa para um espaço vazio ou um bloco de destino. O jogador vence se todas as caixas estiverem em blocos de destino (ZAKARIA; FAYEK; HADHOUD, 2023).

[2] https://pt.wikipedia.org/wiki/Tomb Raider

[3] https://pt.wikipedia.org/wiki/Uncharted

possibilidade de se adaptar à maneira de jogar do jogador, no método de avaliação, entre outras.

1.3 Descrição do Problema

É comum para um jogo ser desenvolvido por centenas de pessoas por um período superior a um ano ou mais. Humanos são lentos e custosos e, além disso, com o crescimento do mercado de *games*, a demanda por conteúdo novo, sofisticado, diferenciado e, por vezes, personalizado também aumenta - tendência essa que infelizmente não é acompanhada pelo custo alto e escalabilidade baixa da criação manual de conteúdo. Isso leva a uma situação em que poucos jogos de fato se tornam lucrativos e menos desenvolvedores podem se dar ao luxo de desenvolver um jogo, levando a uma menor tomada de riscos e menos diversidade no mercado de jogos (SHAKER; TOGELIUS; NELSON, 2016; MELOTTI; MORAES, 2019).

Ainda há discussão no que tange ao uso de *puzzles* enquanto recurso no desenvolvimento de outros gêneros de jogos (KEGEL; HAAHR, 2019a). Para que se possa dizer que o *puzzle* de fato é parte do jogo (e não somente um elemento avulso) é necessário que o *puzzle* incorpore as características do gênero. Em jogos *roguelike*, a GPC é uma característica considerada indispensável – o que significa que, se o jogo é gerado proceduralmente, os *puzzles* no jogo também precisam ser.

É um tanto complexa a equação *puzzle*-GPC-*roguelike*. Ao mesmo tempo que se enfrenta todos os desafios da própria GPC, como a geração correta de conteúdo e com alta variabilidade, não é trivial o *puzzle* gerado fazer sentido dentro do jogo que o cerca, ainda que não seja um elemento incorporado à narrativa do jogo, se tornando também um desafio de *game design*. Outro desafio é do próprio gênero *puzzle* com a geração procedural: certificar-se de que o conceito do *puzzle* em alto nível não seja tão simplificado a ponto de poder ser decifrado rapidamente, pois isto tornaria infrutífera qualquer variabilidade gerada.

1.4 Justificativa

Para contornar esse problema, tem-se um esforço para usar técnicas que visem tornar o desenvolvimento dos jogos mais simples e reduzindo a necessidade do emprego direto da mão-de-obra humana. A GPC, citada na seção anterior, permite que jogos possam ser desenvolvidos em menos tempo e com menos recurso humano, ao mesmo tempo que preserva a qualidade. Isso dá às empresas de desenvolvimento de jogos uma vantagem competitiva, além de possibilitar jogos mais criativos e centrados no jogador.

Portanto, é positivo para a indústria *gamer* que se desenvolva uma ferramenta capaz de suprir essa demanda ao mesmo tempo que assegura por meio de métricas a validade da influência dos elementos gerados. Tal ferramenta permitiria aos jogadores jogar *games* mais criativos e variados (como é o caso do jogo *No Man's Sky*[4], em que universos inteiros são gerados por GPC (VENCO; LANZI, 2021)), enquanto desenvolvedores teriam tempo e recursos poupados no desenvolvimento e poderiam focar em assuntos que exigem mais da criatividade humana para a criação dos jogos.

A ferramenta *Procedural Content Generation Lab* (PCGLab) é um ambiente de estudo dirigido para aplicação e avaliação de técnicas de GPC e vem recebendo contribuição e melhorias do trabalho de vários autores para este fim. Atualmente, não está implementada a GPC de *puzzles* no ambiente. O protótipo desenvolvido neste trabalho servirá para acrescentar estes elementos futuramente à ferramenta.

1.5 Motivação

A contribuição dos trabalhos de COSTA (2020), VERDE (2021), SANTANA (2021) e DIMA (2023) ao ambiente de estudo PCGLab apresenta um esforço para os estudos da GPC no ambiente acadêmico. Os trabalhos anteriores estudam a implementação de técnicas de GPC, avaliam o conteúdo gerado e modelam a progressão de dificuldade. Há, assim, uma direção no sentido de desenvolver uma ferramenta para criar jogos melhores e surpreendentes à experiência do jogador por meio do PCGLab, porém ainda há muito a ser feito. Muito embora neste trabalho o PCGLab não tenha sido utilizado diretamente,

[4]https://www.nomanssky.com

este trabalho pode ser considerado uma continuação direta dos trabalhos anteriores, a fim de que se chegue a uma ferramenta mais completa, com diversas técnicas de geração procedural que podem ser implementadas e avaliadas.

Espera-se que, em sua versão final, o ambiente de estudo possa ser usado tanto como ferramenta corporativa para auxiliar na produção de jogos comerciais quanto como ferramenta acadêmica para ensinar sobre desenvolvimento de jogos. Porém, a geração de um elemento específico de jogos, os *puzzles*, ainda carece no ambiente de estudo supracitado e é um desafio a se superar.

De uma perspectiva pessoal, o autor já teve contato com desenvolvimento de *puzzles* manualmente em jogos e tem conhecimento de que, mais do que apenas implementar em código um *puzzle*, pensar em um *design* de *puzzle* interessante ao jogador e que faça sentido no contexto do jogo é um grande desafio, e isso desperta o desejo de conhecer ferramentas que possam facilitar este trabalho e também tornar melhor a rejogabilidade e a narrativa do jogo.

Como última motivação, ainda pessoal, o autor também anseia se aprofundar em desenvolvimento de jogos em geral. Sendo GPC uma técnica conhecida e largamente utilizada, este trabalho também serve como um meio de adquirir maior conhecimento em desenvolvimento e *design* de jogos.

1.6 Objetivos

O objetivo geral deste trabalho é entender como os diferentes algoritmos de geração procedural afetam diretamente a experiência do jogador em *puzzles* e encontros de jogos *roguelike*. Para nos aproximar deste objetivo, os seguintes objetivos específicos devem ser alcançados:

1. listar e classificar modelos de encontros e *puzzles* para *roguelikes*;
2. adaptar os modelos de *puzzle* na implementação do jogo;
3. analisar o comportamento de cada método de geração procedural escolhido em função da topologia gerada;

4. assegurar que os elementos de jogo gerados são corretos (sem gerar elementos incorretos, como inacessíveis ou desnecessários) e únicos do ponto de vista da variabilidade;

5. avaliar o jogo sob a perspectiva da experiência do jogador, via formulário de opinião.

Após concluir esses objetivos, pode-se selecionar bons modelos para gerar proceduralmente os *puzzles* dentro dos jogos. Por meio da avaliação via formulário, obtemos a confirmação que, sob a ótica do jogador, o jogo continua interessante após ser jogado mais de uma vez e que os elementos gerados encantam e estimulam quem joga.

1.7 Material e métodos

Este trabalho busca realizar uma pesquisa exploratória amparada por desenvolvimento de software. Foram seguidos os seguintes passos a fim de atingir os objetivos propostos anteriormente:

1. implementar abordagens disponíveis no estado da arte;

2. definir e observar a solução através de algoritmos de busca;

3. realizar avaliação com jogadores reais e coleta de dados via formulários.

Primeiramente, foram desenvolvidos os principais métodos de geração de *puzzles* levantados na literatura. Depois, uma etapa de geração de *puzzles* foi inserida após a etapa de geração de topologia. Por fim, foi realizada uma avaliação com humanos. O método de coleta foi por meio de formulários, que foram respondidos após os testadores jogarem.

Em todo o desenvolvimento, foi usada a linguagem de programação JavaScript[5] na implementação, amparada pelo uso do editor de código Visual Studio Code[6] para apoiar o desenvolvimento de código e Github[7] para versionamento de código. O código-fonte do jogo está disponível no repositório[8].

[5]https://developer.mozilla.org/pt-BR/docs/Web/JavaScript
[6]https://code.visualstudio.com/
[7]https://github.com/
[8]https://github.com/MarcusVsc1/BlueMoonPCGLabTests

1.8 Organização do trabalho

Este trabalho está organizado em cinco capítulos. Além desta Introdução, o Capítulo 2 apresenta uma revisão dos principais conceitos a serem utilizados aqui. O Capítulo 3 apresenta alguns trabalhos da área e os compara a fim de analisar suas abordagens. O Capítulo 4 contempla o processo de desenvolvimento do problema abordado e analisa os resultados e, por fim, o Capítulo 5 contém as considerações finais, limitações e futuras possíveis frentes de trabalho, seguido pelas referências bibliográficas utilizadas.

2 Fundamentação Teórica

Este capítulo de fundamentação teórica tem como objetivo estabelecer as bases teóricas necessárias para a compreensão do trabalho e apresentar os principais conceitos relacionados à geração procedural de *puzzles* e encontros em jogos *roguelike*, sendo discutidos os conceitos fundamentais da área. A Seção 2.1 conceitua desenvolvimento de jogos digitais. A Seção 2.2 discute o que é o gênero de jogos *roguelike*, o que é a GPC e qual a sua relação com o gênero. *Puzzles* (ou quebra-cabeças) em jogos eletrônicos e como *puzzles* e encontros se aplicam a jogos *roguelike* são descritos na Seção 2.3. Os principais algoritmos conhecidos para gerar esses *puzzles*, incluindo os algoritmos que foram abordados em trabalhos relacionados, são definidos na Seção 2.4.

2.1 Desenvolvimento de Jogos digitais

Um jogo com um bom *design* é centrado no jogador. Isso significa que, acima de tudo, o jogador e seus desejos são verdadeiramente considerados (BRATHWAITE; SCHREIBER, 2008). Entretanto, não é trivial captar esses desejos. Um *designer* de jogos precisa ter habilidades multidisciplinares, passando desde as mais conhecidas, como animação, artes visuais, música e escrita criativa, como também outras como economia, antropologia e engenharia, que são menos óbvias (SCHELL, 2008). Junte isso a um público interessado em jogar *videogames* cada vez maior. Uma audiência mais ampla também significa uma demanda maior por jogabilidade de qualidade. À medida que jogadores se tornam mais experientes, aumenta também a ânsia por jogos mais sofisticados. Porém, a indústria de jogos é relativamente recente em comparação a outras formas de mídia, havendo ainda muito lugar para inovação (DORMANS, 2013).

Uma das maiores dores em desenvolvimento de jogos é que conteúdo de alta qualidade é caro de se produzir, e jogos produzidos manualmente são bastante dispendiosos. Assim, emergem como alternativa ferramentas que auxiliem na automatização do desenvolvimento, deixando ao processo manual apenas o que requer a criatividade e

inventividade humana (DORMANS, 2013).

2.2 Geração Procedural e *Roguelikes*

Roguelike é um subgênero de jogo eletrônico que se origina do jogo Rogue (KUITTINEN, 2001). Vários jogos mostravam-se similares ao Rogue, gerando os primeiros esforços em organizá-los em um termo genérico. Este termo permaneceu estável até meados de 2010, com o crescimento do cenário dos jogos *indie* e a difusão de jogos *roguelike* mais focados em ação (ZAPATA, 2017).

Muito se discute sobre o que define exatamente um jogo como *roguelike*, sendo que, em 2008, durante a *International Roguelike Development Conference*, estabeleceu-se por um grupo de entusiastas a chamada *The Berlin Interpretation*, que objetivou definir os princípios de um jogo que pertence ao gênero. Mesmo assim, ainda há controvérsias sobre o assunto, inclusive sendo cunhado o termo *roguelite*. Este termo foi cunhado para jogos que possuem algumas características do Rogue, porém também outras características que jamais existiriam no jogo original ou abandonando completamente algumas características consideradas essenciais. Em contrapartida, ao termo *roguelike* são somente definidos jogos cujas características se aproximam ao Rogue de forma mais inflexível (WINKIE, 2021). Para contexto, um *roguelite* pode manter a progressão do jogo após o jogador fracassar na aventura, afrouxando a regra da morte permanente (PRATA, 2022).

Segundo o que foi acordado na *Berlin Interpretation*, há alguns elementos que são considerados fundamentais para que um jogo seja um *roguelike*, como: morte permanente (o que significa que, quando o personagem morre, o jogo acaba, sem a possibilidade de reiniciá-lo); movimentação baseada em turnos; interações complexas de personagem-objeto-mundo; necessidade de gerenciar uma quantidade finita de recursos para sobreviver e; a cláusula que é o foco deste trabalho, níveis gerados proceduralmente ou aleatoriamente (MOSS, 2020).

A GPC é a criação algorítmica, com entrada de usuário limitada ou indireta, de conteúdo de jogo (SHAKER; TOGELIUS; NELSON, 2016). É um recurso largamente utilizado a fim de gerar economia de custo e tempo de desenvolvimento por meio da criação de conteúdo como mapas, inimigos, encontros e itens (TOGELIUS et al., 2011).

O uso de GPC traz, além do barateamento dos custos e redução do tempo de produção dos jogos, outras vantagens. Para jogos *roguelike*, do uso de GPC melhora a rejogabilidade do jogo, pois não somente a masmorra é gerada proceduralmente, como vários outros elementos, alterando totalmente a experiência do jogador a cada vez que o jogo é jogado (SHAKER; TOGELIUS; NELSON, 2016); além disso, a jogabilidade deixa de ser focada na memorização pelo jogador e passa a ser em sua habilidade (MOSS, 2020).

Um objetivo do uso da GPC é que se gere conteúdo único a cada vez que o jogo é jogado. Short e Adams (2019) chamam a atenção da necessidade de assegurar que não se caia na armadilha de, matematicamente falando, gerar uma infinidade de conteúdos proceduralmente, porém todos são similares de tal forma que o jogador não tem a percepção de estar jogando algo diferente.

2.3 *Puzzles* e Encontros

Quebra-cabeças (ou *puzzles*) são definidos como problemas em que o jogador precisa encontrar uma solução baseada em conhecimento prévio (de dentro ou fora do jogo) e/ou explorando o espaço solução (COLTON, 2002). Em jogos digitais, *puzzles* são um recurso de diversos gêneros de jogos e vêm em uma miríade de formas. A Figura 2.1 mostra a divisão dos tipos de *puzzle* em categorias e classifica-os de acordo com a sua variabilidade.

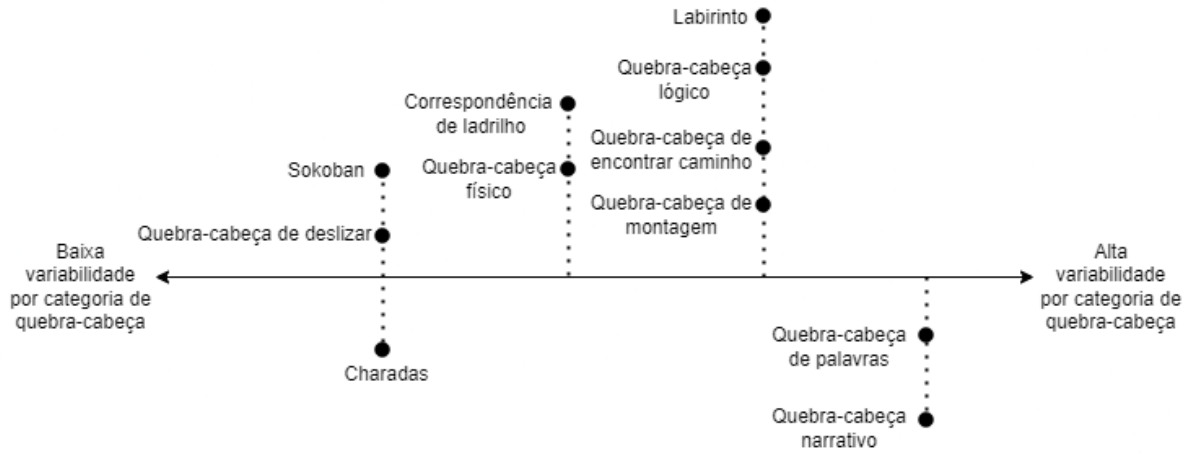


Figura 2.1: Mapa de categorias de *puzzles*. As categorias mais à esquerda possuem *puzzles* com alta similaridade entre si, enquanto à direita possuem *puzzles* com maior variabilidade. A posição vertical não é significante. Fonte: do autor, baseado em Kegel e Haahr (2019a)

Dentro do contexto de um jogo *roguelike*, uma categoria de *puzzle* que é bastante trabalhada é o *puzzle* narrativo, que consiste em *puzzles* que formam parte da progressão da narrativa, cujas soluções envolvem exploração e lógica, assim como pensamento criativo (KEGEL; HAAHR, 2019a). Alguns exemplos conhecidos do que são considerados padrões de *puzzles* narrativos:

- descobrir qual item um personagem deseja, levando geralmente a uma recompensa em troca;
- combinar logicamente dois objetos a fim de mudar suas propriedades ou gerar um objeto totalmente novo;
- desmontar um objeto em componentes úteis;
- dizer a coisa certa para conseguir ajuda de um personagem;
- achar uma chave que abre uma porta para dar acesso a outra área.

Puzzles de chave e porta são elementos que incrementam a exploração no jogo, ao tornar partes da aventura inacessíveis, forçando o jogador a passar por vários obstáculos para encontrar e usar as chaves ou itens similares a chave nas portas e afins. Exigem muitas vezes algum nível de *backtracking*, quando o jogador se depara com uma porta trancada e é forçado a retornar para áreas previamente exploradas para buscar a chave que a abre. Estes *puzzles* formam uma importante parte das restrições rígidas de um jogo, pois uma implementação ruim ou incorreta podem tornar o jogador incapaz de prosseguir no jogo caso não encontre ou não consiga usar a chave correta para uma porta (DETERDING et al., 2018). As chaves podem ser consumíveis, quando são destruídas no processo de abertura da porta, e persistentes, quando não são. As chaves consumíveis em geral são particulares para portas específicas, podendo ser usadas somente nestes locais, enquanto chaves persistentes costumam ser não-particulares e podem ser usadas em diversas portas, tornando-se multi-propósito (DORMANS, 2017). O jogo *Goof Troop*, assim como é mostrado na Figura 2.2, usa o *puzzle* de chave e porta para aumentar a exploração no jogo.

Um dos desafios dos *puzzles* narrativos é gerar encontros excitantes, de tal forma a produzir variações e combinações interessantes que se traduzam em qualidade de *design*, estrutura geral e objetivos de jogabilidade (ROZEN; DORMANS; SAMARITAKI, 2022).

O termo encontro vem do RPG e representa toda interação do personagem do jogador com outro personagem, controlado por outro jogador ou não (*Non-Player Character* (NPC)). Um encontro também pode ser visto como uma cena dentro da narrativa, mas, neste trabalho, o termo encontro será utilizado apenas para combates com NPCs inimigos. Ainda não existem mecanismos de diálogos ou NPCs amistosos nesta versão.

Figura 2.2: Cena do jogo *Goof Troop*. Este jogo possui *puzzles* de chaves que abrem portas para outros locais. Na ilha à esquerda inferior da tela, está posicionada uma chave consumível e não-particular. Fonte: Green (2013)

2.4 Geração procedural de *puzzles*

Assim como outros elementos de jogos, os *puzzles* também podem ser gerados proceduralmente. Apesar de serem uma maneira eficaz de enriquecer a jogabilidade, fazendo com que o jogador tenha que parar e pensar sobre como resolver um problema, uma vez que o jogador decifra como resolvê-lo, o *puzzle* muito provavelmente perde uma parte do que o torna divertido e interessante. Dessa forma, a geração procedural entra como uma solução, auxiliando no aumento da variabilidade do *puzzle*. Tal solução, no entanto, ainda mantém-se limitada, sendo usada principalmente em criar jogos com mecânicas baseadas em *puzzles* e não em criar *puzzles* como um recurso acessório a outros gêneros de jogos (VENCO; LANZI, 2021). O trabalho de Kegel e Haahr (2019a) traz uma excelente visão geral das técnicas de geração procedural mais utilizadas para cada estilo de *puzzle*.

Alguns trabalhos existentes na literatura são voltados a gerar elementos de *puzzle* em masmorras. A topologia, por exemplo, pode ser representada nos mapas do jogo como um caminho acíclico (DETERDING et al., 2018). Uma maneira de implementar um caminho acíclico é modelando em grafos e calculando uma árvore geradora mínima. Dois algoritmos possíveis para este fim são o algoritmo de Prim (FEOFILOFF, 2019b) e o algoritmo de Kruskal (FEOFILOFF, 2019a). Na questão de geração e posicionamento dos *puzzles* na topologia, os trabalhos são diversificados na questão de como implementar regras a fim de evitar criar impasses entre os elementos do jogo, o que impediria o progresso do jogador. O trabalho de Venco e Lanzi (2021) usa uma abordagem de modelo baseado em agentes para criar labirintos e elementos de *puzzle* no cenário, como *puzzles* de chave e porta e *puzzles* de alavanca, para mapas baseados em grafos. Já Green et al. (2019) têm uma abordagem dupla para geração procedural em um jogo *roguelike*, implementando e avaliando três geradores: um baseado em restrições, outro baseado em autômato celular e um terceiro em modelo baseado em agentes. Estes geradores são usados primeiramente para gerar a masmorra (os chamados "criadores de *layout*" ou espaço topológico) e, logo em seguida, para gerar inimigos, entradas, saídas, armadilhas e outros elementos (os chamados "mobiliadores"). Outro trabalho na área é o de Gellel e Sweetser (2020), cuja abordagem híbrida apresenta outras técnicas para construção de cenário e elementos de fases de *roguelikes*, como gramáticas, cadeias de Markov, algoritmos evolucionários, entre

outras.

Além de algoritmos de GPC voltados a posicionar os *puzzles* na topologia, os *puzzles* por si só podem lançar mão de técnicas de GPC. Dois exemplos de *puzzle* aplicáveis em jogos *roguelike* que podem ser gerados são *sokoban* e labirinto.

Sokobans são jogos de quebra-cabeça jogado em uma grade retangular. O objetivo é que o jogador empurre caixas para os quadrados marcados como objetivos. O desafio decorre da disposição das paredes, objetivos e caixas, e da restrição de que o avatar só é forte o suficiente para empurrar um bloco de cada vez e não pode puxar blocos de forma alguma (TAYLOR; PARBERRY, 2011). Para GPC de *sokoban*, existem abordagens *online* e *offline*, considerando em quanto tempo são gerados os *puzzles*. Kartal, Sohre e Guy (2016) usam *Monte Carlo Tree Search* (MCTS) para gerar um tabuleiro de *sokoban*, apresentando bons resultados mesmo com curto tempo para geração. Seguindo outra abordagem, Zakaria, Fayek e Hadhoud (2023) seguem a linha de usar *machine learning* para criar tabuleiros com qualidade mais garantida, porém necessitando de treinamento e reforço do modelo para tal, tratando-se, portanto, de uma estratégia *offline*.

Para GPC de labirintos, diversas técnicas podem ser aplicadas. O método apresentado por Kim e Crawfis (2018), por exemplo, consiste em escolher, em tempo de execução, a depender de métricas coletadas durante a geração, o algoritmo mais adequado. Alguns algoritmos possíveis de escolha são *backtracking* recursivo, os já citados anteriormente, algoritmos de Prim e Kruskal, entre outros. Outro método, menos sofisticado e de compreensão mais simples, é o de Ajith et al. (2022). Neste, são usados alguns modelos de peças para criar o caminho. As peças são escolhidas aleatoriamente de acordo com algumas restrições, gerando por fim um labirinto com diversas saídas possíveis.

2.5 Considerações Parciais

Este capítulo de fundamentação teórica foi essencial para apresentar conceitos fundamentais para a compreensão do trabalho e estabelecer as bases teóricas necessárias para a aplicação das técnicas de GPC de conteúdo em *puzzles* e encontros de jogos *roguelike*. A partir dos conceitos abordados, foi possível entender como a GPC de conteúdo se relaciona com o gênero de jogos *roguelike* e como os *puzzles* e encontros podem ser gerados

de forma automática e aleatória. O conhecimento adquirido será crucial para adaptar e avaliar os algoritmos e técnicas de geração procedural para o jogo. A partir daqui, será possível compreender o que será apresentado nas próximas seções e acompanhar o desenvolvimento do trabalho. O capítulo seguinte visa analisar e comparar alguns estudos na área, suas abordagens e possíveis contribuições para este trabalho.

3 Trabalhos relacionados

Este capítulo tem como objetivo apresentar uma revisão dos trabalhos relacionados ao tema da geração procedural de *puzzles* e encontros em jogos *roguelike*. A seleção dos trabalhos foi realizada de forma criteriosa, incluindo estudos acadêmicos, contribuições científicas e materiais provenientes da indústria de jogos. Essa abordagem de seleção proporciona uma visão abrangente e diversificada do campo, ao considerar trabalhos de diferentes origens, enriquecendo assim a compreensão do tema.

3.1 Definição dos trabalhos

Foram adotados métodos diversos de seleção, a fim de abranger uma gama maior de trabalhos relevantes. Primeiramente, foi considerado um vídeo de uma palestra proferida durante a *Roguelike Celebration*[9], uma conferência anual dedicada ao gênero de jogos *roguelike*. Essa fonte proporcionou *insights* e uma visão geral de uma solução de um produto bem aceito na área.

Além disso, foi selecionada uma dissertação de mestrado da área específica, contribuindo para o corpo de conhecimento existente, abordando aspectos teóricos e experimentais relevantes.

Outro conjunto de trabalhos selecionados incluiu quatro artigos científicos, provenientes das bases de dados IEEE Xplore, Scopus e ACM Digital Library. Para a seleção dos trabalhos, foi usada a seguinte *string* de pesquisa: `puzzle AND procedural AND generation`.

O corte dos trabalhos se deu ao filtrar a partir de 2018, a fim de selecionar somente resultados recentes. Por fim, foram eliminados trabalhos cujo título não condizia com a proposta da pesquisa e selecionou-se os relevantes pela leitura do *abstract*. Essas fontes proporcionaram uma visão ampla dos estudos mais relevantes, contemplando diferentes abordagens, algoritmos e técnicas utilizados.

[9] ⟨https://www.roguelike.club/⟩

Os resumos de cada um desses trabalhos estão nas Seções 3.2 até 3.7. Ao final deste capítulo, na Seção 3.8, os trabalhos serão comparados em relação à sua abrangência, considerando o tipo de jogo em que a GPC é aplicada, a técnica de GPC utilizada, se a geração depende de interferência humana e quais elementos específicos são gerados. Essa análise comparativa permitirá identificar as principais abordagens adotadas pelos estudos relacionados e quais podem contribuir ao conteúdo deste trabalho.

3.2 Walker (2018)

O trabalho de Walker (2018), apresentado durante a *Roguelike Celebration* de 2018, trata de técnicas de GPC focadas para jogos *roguelike* com perspectiva *top-down* – ou seja, com visão aérea, com foco no personagem, e para jogos de plataforma, simulando um jogo estilo Super Metroid[10], só que gerado proceduralmente. No entanto, serão comentadas apenas as técnicas para jogos com perspectiva *top-down*, pois é a categoria que se encaixa na proposta deste trabalho e do ambiente PCGLab.

Aqui, o autor lança mão de técnicas sofisticadas de GPC para gerar a topologia da fase, inimigos e *puzzles* dos mais variados tipos. Tanto para a implementação dos elementos quanto da topologia do mapa, usa-se autômatos celulares. Primeiramente, é gerado o mapa, cujas salas são criadas por meio de sobreposição de retângulos e círculos. Depois, selecionam-se células candidatas a portas. Estas salas podem se conectar ou não por meio de corredores - dependendo do quanto as salas estão próximas, podem se conectar via portas por meio de células de parede. As salas são geradas até que o mapa seja preenchido.

Os *puzzles* formam um conjunto concreto de objetivos no jogo e são usados diversos tipos de *puzzles* pela masmorra gerada. Além dos *puzzles* mais comuns, que envolvem portas, alavancas e chaves, podem ser gerados *puzzles* que envolvem um raciocínio mais ágil do jogador ou que envolvem várias mecânicas, como salas inundadas, salas que se incendeiam rapidamente (obrigando o jogador, por exemplo, a desviar rapidamente do fogo ou localizar um item que o torne imune a fogo), seções de lava e até uso de NPCs, como aranhas que constroem um caminho ao longo de uma sala que a princípio não pode

[10]https://www.nintendo.pt/Jogos/Super-Nintendo/Super-Metroid-279613.html

ser passada. Na Figura 3.1, o autor mostra um *puzzle* de campo de lava.

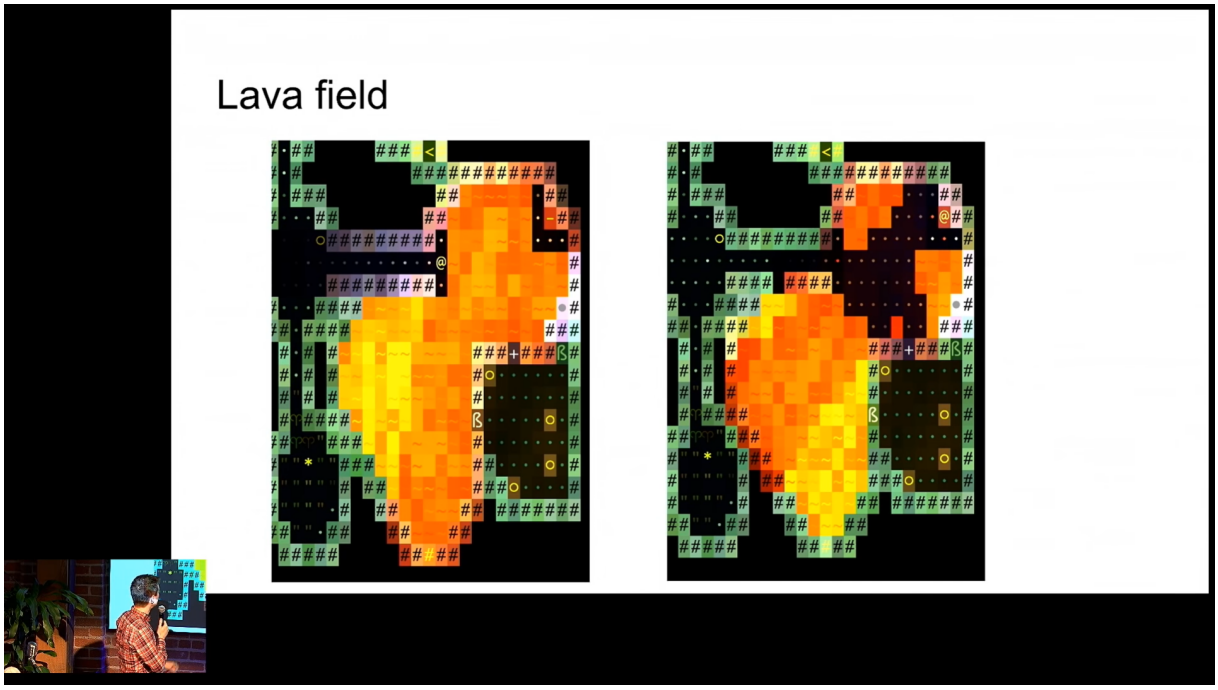


Figura 3.1: Momento da apresentação em que o autor mostra a aplicação de *puzzles* de campo de lava como um exemplo de possível *puzzle* a ser gerado em uma partida. Fonte: Walker (2018)

Todos os elementos que preenchem o mapa são gerados por agentes, que preenchem qualitativamente de acordo com o terreno e itens já incrementados, reescrevendo de forma iterativa o cenário.

3.3 Garcia (2022)

O trabalho de Garcia (2022) se trata de uma dissertação de mestrado com enfoque em jogos 3D de plataforma, mesclando jogabilidade de jogos *roguelike*.

A GPC é usada, principalmente, para a geração da topologia. O mapa consiste em um grande hexágono, composto de ladrilhos hexagonais variando em elevação que são agrupados em conjuntos. A justificativa para essa escolha de *design* é que o mapa de mundo consiste em diferentes temas exteriores e ladrilhos hexagonais geram um mapa mais orgânico comparado com mapas de células retangulares.

Sobre os *puzzles*, é usada a diferença de altura entre os ladrilhos como forma de criar um *puzzle* físico de plataforma. É citada a possibilidade de incluir outros tipos de *puzzle*, mas não são incorporados à ferramenta.

Toda a ferramenta é feita na *Unreal Engine* 4[11]. Com relação a técnicas, são usados como recursos para navegação dos inimigos pela malha gerada para a topologia os algoritmos A* (usado para gerar um campo de navegação que permite saltos imprevisíveis) e traçado-de-raio (permite que a inteligência artificial trace raios de luz a partir da sua posição atual até o ponto de destino, verificando se há obstáculos ou não, e é menos custoso que o A*) da própria *engine*. Não é citada qual técnica específica de GPC foi usada neste trabalho, diferindo dos demais trabalhos relacionados.

É usado um sistema de progressão por meio de *checkpoints* em cada área. Dado que novas áreas só são geradas e acessíveis após completar uma área anterior, a GPC usa métricas de progressão para gerar as novas áreas em tempo real. Assim, há uma influência da progressão de dificuldade na geração de conteúdo.

3.4 Pereira, Viana e Toledo (2022)

O foco deste trabalho é em jogos 2D de ação e aventura, mas também usável para jogos *roguelike*, conforme apontado pelos autores. Mas a maior contribuição aqui se dá pelo sistema de perfis, em que a GPC é voltada para diferentes perfis de jogador, tornando a experiência muito mais personalizada.

O sistema tem alguns módulos principais, mas os que se destacam são o analisador de perfil, o gerador de conteúdo procedural, o orquestrador de conteúdo e o jogo propriamente dito, que é desenvolvido de forma incremental conforme os resultados dos módulos anteriores na plataforma *Unity*[12].

O analisador de perfil funciona por meio de questionários que são feitos no início e no final de cada partida. Esses questionários geram dados que são processados e detectam o arquétipo de jogador mais adequado entre os arquétipos pré-determinados, sendo estes conquista, criatividade, imersão e maestria.

Os dados do analisador de perfil são então enviados ao gerador procedural. O gerador usa, primeiramente, estes dados em uma gramática formal simples para gerar possíveis missões (*quests*). Após isso, gera conteúdo de salas e *puzzles* de chave e porta

[11]https://www.unrealengine.com/en-US/blog/welcome-to-unreal-engine-4
[12]https://unity.com/

que se encaixam nestas missões e também inimigos com características condizentes. Estas características variam desde o tipo de inimigo, como também pontos de vida, dano, velocidade, padrões de movimento e arma utilizada.

O método usado anteriormente na GPC dos elementos supracitados era de algoritmo evolucionário, mas passou a ser usado o algoritmo MAP-Elites. Este algoritmo funciona criando nós representando salas, com as arestas sendo as conexões entre as salas, e travas podem bloquear arestas até que uma chave seja encontrada. O algoritmo também distribui os inimigos pelos nós e converte a representação em árvore em um mapa 2D de grade.

Por fim, o orquestrador de conteúdo, que é uma unidade de pós-processamento dos conteúdos criados, descarta conteúdo das masmorras cujos valores de adaptação estejam muito fora do desejado, baseando-se em valores retornados pelo algoritmo MAP-Elites. Após isso, descarta também algumas combinações de inimigos indesejadas e posiciona os inimigos nas salas escolhidas para cada. O orquestrador também posiciona os tesouros, mas segue uma rotina determinística para esta tarefa.

Com o fim da partida, outro questionário é passado ao jogador, a fim de adaptar o conteúdo de um próximo jogo com base na experiência do último jogo gerado, alimentando assim o analisador e tornando a experiência do jogador mais agradável.

Por mais que este trabalho tenha pouco enfoque nos *puzzles* (o único citado é de chave e porta), a maneira como adapta a GPC à experiência do jogador mostra uma abordagem que deve receber atenção.

3.5 Venco e Lanzi (2021)

Este trabalho não possui um foco em um gênero específico de jogo, mas é aplicável a qualquer jogo que tenha um mapa configurado em grade.

Os autores usam técnicas variadas de GPC para gerar os elementos do jogo. Para os mapas, é usado um algoritmo de particionamento de espaço binário, que irá gerar salas retangulares dentro de um mapa vazio. Grafos são usados como estruturas de dados para ilustrar as conexões entre as salas e também usados para as conexões entre os *puzzles*, controlando assim o acesso às áreas e aos itens do jogo. A Figura 3.2 mostra exemplos de

mapas gerados, como estes mapas são representados nos grafos de mapa, e os grafos dos *puzzles* inseridos na topologia.

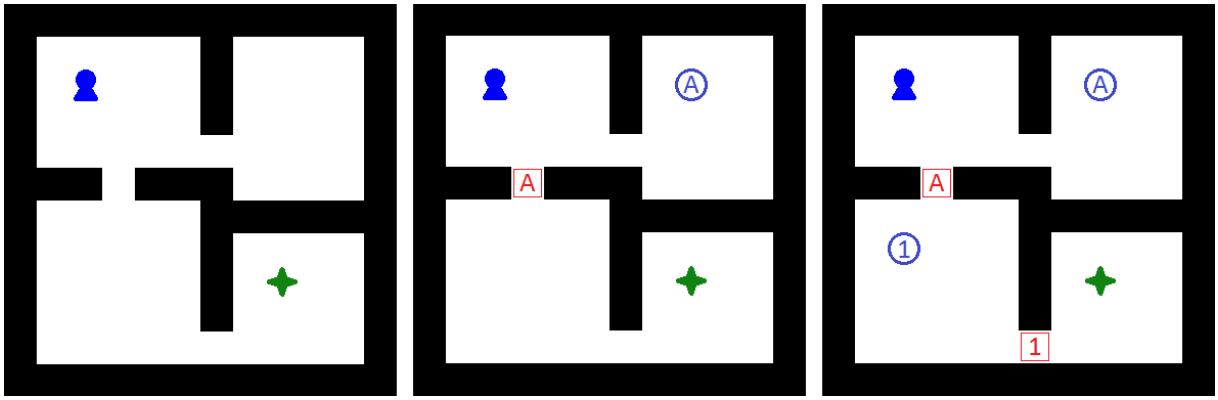
Os *puzzles* são gerados por agentes, cada um para um tipo diferente de *puzzle* – por exemplo chave e porta, interruptores, alavancas ou tochas. Estes agentes injetam os *puzzles* pelo mapa gerado, gerando um grafo de *puzzles* a partir do grafo do mapa, adicionando ações que são permitidas ao jogador. Os agentes são selecionados aleatoriamente, porém a estratégia de posicionamento pode ser tanto determinística quanto estocástica. Por exemplo, a critério do projetista, podem ser checados primeiro locais mais próximos do começo ou do fim da masmorra ou até mesmo em posições totalmente aleatórias no mapa. A cada iteração um novo nó é adicionado ao grafo de *puzzles* até que um limiar de qualidade seja alcançado ou que um determinado número de falhas para incluir um novo nó seja atingido.

O interessante da abordagem deste trabalho é que o uso dos grafos para os mapas e para os *puzzles* garante tanto que não haja salas desconexas quanto que os *puzzles* não gerem caminhos impossíveis. Portanto, o desafio sempre será solucionável e é robusto de tal forma que nenhuma sequência de ações irá impedir o jogador de chegar na saída. Adicionalmente, todos os elementos foram gerados em tempo computacional razoável - para mapas com grade de tamanho 22×22, a média de tempo foi 1,272 segundos. Deve-se levar em consideração que, para mapas nesta configuração, chegou-se a um máximo de 32 salas geradas, o que significa 32 nós no grafo do mapa.

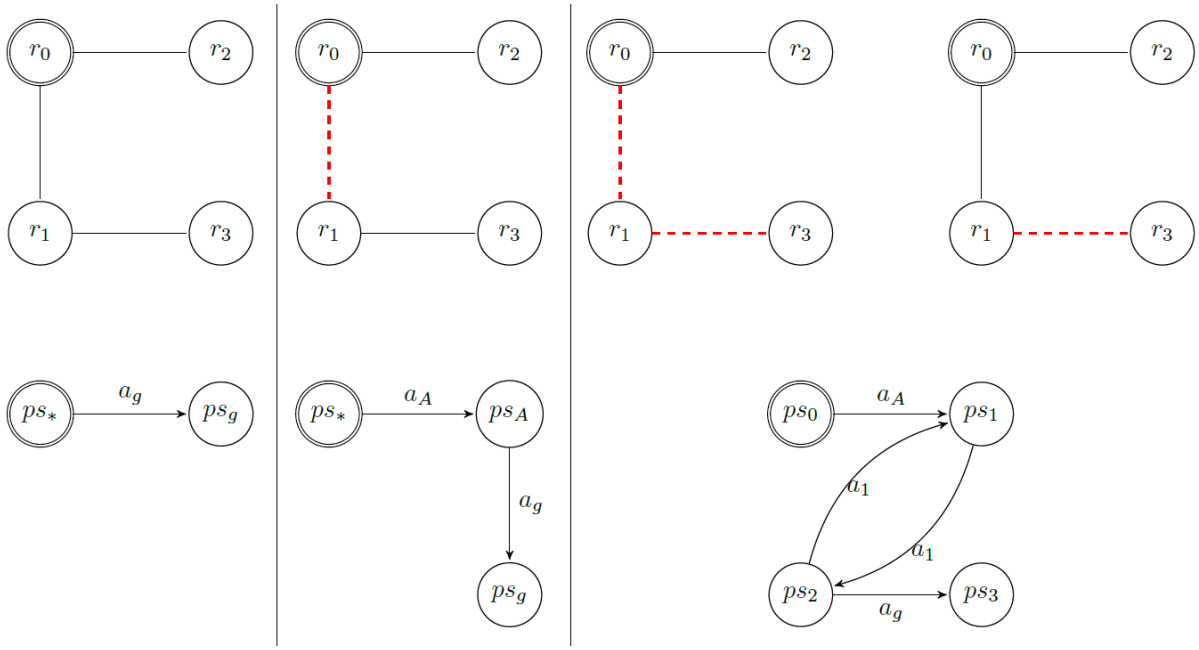
3.6 Kegel e Haahr (2019b)

Este trabalho também não é direcionado a um gênero específico de jogos, mas sim jogos que sejam mais voltados à história, pois os *puzzles* gerados são *puzzles* narrativos.

A GPC deste trabalho é fortemente dependente de dados externos. Isso significa que, para que haja a geração de conteúdo, o projetista precisa que uma base de dados alimente os dados do jogo. Neste caso, o gerador é alimentado por um banco de todos os itens usáveis, um conjunto de regras que descrevem o espaço de todos os *puzzles* e uma lista de áreas do jogo. É usada como abordagem gramática livre de contexto, tanto como base para o algoritmo de geração de conteúdo quanto para solução dos *puzzles*.



(a) Representação dos mapas gerados e seus puzzles.



(b) Representação em grafo. A primeira linha representa os grafos de mapa. Neste, os nós são as salas e as arestas em vermelho destacam a restrição de algum *puzzle*. A segunda linha representa os grafos de *puzzle*, em que os nós representam combinações dos estados possíveis dos *puzzles* e as arestas a forma como o jogador pode alcançar os estados.

Figura 3.2: Representação dos mapas gerados com *puzzles* e suas representações em grafo. As colunas representam o mapa correspondente. Fonte: Venco e Lanzi (2021)

As áreas do jogo (denominadas áreas de *puzzle*) são modulares e podem conter vários itens de *puzzle*. Para cada área é definido um objetivo e, a partir dele, são gerados os itens. Estes itens podem ser necessários para o objetivo desta sala ou não, dependendo de outras salas disponíveis. Além disso, múltiplos objetivos possíveis podem ser associados a uma área, em que o gerador checa se a lista de restrições dos itens foi completada, a fim de evitar que o *puzzle* seja finalizado prematuramente.

A geração ocorre em tempo de execução do jogo: após concluído o objetivo de uma área, irá liberar as demais que são conectadas a ela, que só então terão seus itens gerados. Entretanto, não necessariamente estas salas começarão vazias, pois elementos ambientais das salas (por exemplo, árvores ou riachos) podem servir como ponto para inclusão de itens que irão servir no objetivo desta sala – por exemplo, ao se ter uma árvore no ambiente, o jogo pode incluir um machado que irá cortar essa árvore.

O trabalho não faz menção à geração do espaço das salas ou especificidades da conexão entre estas. Também não é feita menção ao posicionamento dos itens gerados dentro das áreas. Por mais que torne abrangente para diversos tipos de jogo, outras abordagens de GPC podem ser exigidas como complemento ao se implementar em um jogo *roguelike*. Por fim, o uso de *puzzles* narrativos da maneira como é proposto neste trabalho aumenta o engajamento, imersão e interesse do jogador em jogos do gênero.

3.7 Weeks e Davis (2022)

Neste trabalho, não há um foco específico em jogos *roguelike*, por mais que sua abordagem seja perfeitamente utilizável para jogos do gênero. O foco neste trabalho é em jogos com perspectiva *top-down*, independentemente de seu gênero, e *puzzles* de chave-porta.

Após ser gerada a masmorra, que possui salas de diferentes formas geométricas e pode possuir corredores com caminhos sem saída, são inseridos aleatoriamente os elementos de chave e de porta. Após isso, é usada uma adaptação do algoritmo A* para calcular as distâncias entre os elementos gerados, usando como distâncias o caminho mais curto passando pelos obstáculos necessários para completar a masmorra e o caminho mais curto ignorando os obstáculos. Após isso, popula-se uma matriz de distância e, por fim, cria-se uma árvore, com o nó inicial como a raiz, arestas como a distância e demais nós

representando a localização dos demais itens.

Para avaliar a masmorra gerada, foram usadas três métricas. A primeira é se a masmorra é solucionável. A segunda, o seu nível de dificuldade, que depende da distância do caminho mais curto e do número de passos adicionais que são necessários para completar a masmorra. A última é o nível de engajamento, que é calculado pelo número de caminhos sem saída, número de salas e pelas distâncias usadas no algoritmo A*.

As técnicas usadas neste trabalho, conforme identificado pelos próprios autores, levam a alguns problemas. O primeiro é que não garante que a masmorra é solucionável – apenas verifica se é ou não, mas não garante que é durante a geração. Além disso, quanto mais elementos são adicionados, mais se entra em um problema de caixeiro viajante, tornando o método de geração da árvore e a determinação dos caminhos possíveis rapidamente insustentável computacionalmente. Há, também, dois problemas nos elementos gerados. O primeiro é que adicionar além de oito portas pode criar *bugs*, em que uma chave passou a abrir mais de uma porta ou portas que não eram abertas por nenhuma chave. O segundo problema é que em diversas masmorras haviam caminhos que simplesmente circulavam as portas, não necessitando passar por nenhuma para chegar ao objetivo final.

3.8 Conclusões Parciais

A Tabela 3.1 mostra um comparativo dos trabalhos revisados. Os trabalhos selecionados apresentaram diversas abordagens e contextos diferentes. Muitos deles não têm foco em jogos *roguelike*, mas as técnicas de GPC usadas são compatíveis com o gênero e podem ser exploradas neste trabalho.

Outro detalhe importante a ser notado é que muitos deles têm foco em *puzzle* de chave e porta, o que talvez denote uma falta de trabalhos na área que tenham um enfoque em uma diversidade maior de *puzzles*. Dado que *puzzles* de alavanca e interruptores, de certa forma, são muito similares aos de chave e porta, podemos dizer que somente o trabalho de Walker (2018) possui uma diversidade maior de tipos de *puzzle*. O trabalho de Kegel e Haahr (2019b) pode ser bastante diversificado em seus *puzzles* a depender da

Tabela 3.1: Tabela comparativa entre os projetos estudados.

Trabalho	Abrangência	Elementos gerados	Interferência humana na geração	Abordagem utilizada
Walker (2018)	*Roguelike*, jogos de plataforma	Chave e porta, *puzzles* espaciais, *puzzles* com NPC	Não possui	Autômatos celulares
Garcia (2022)	Jogos 3D (plataforma + *roguelike*)	Topologia 3D com desnível entre ladrilhos	Não possui	Não citada pelo autor
Pereira, Viana e Toledo (2022)	Jogos 2D de ação e aventura	Chave e porta	Formulários antes e depois das sessões	Gramática formal, algoritmo evolucionário, algoritmo MAP-Elites
Venco e Lanzi (2021)	Jogos com mapa de grade	Chave e porta, interruptores, alavancas, tochas	Não possui	Particionamento de espaço binário, agentes, grafos
Kegel e Haahr (2019b)	Jogos com ênfase em história / narrativa	Depende da base de dados usada	Base de dados externa	Gramática livre de contexto
Weeks e Davis (2022)	Jogos 2D de perspectiva *top-down*	Chave e porta	Não possui	Algoritmo A*, matriz de distâncias, árvore

base de dados que alimenta o gerador, mas usa apenas um tipo específico de *puzzle*, que é o narrativo.

Quantos às técnicas, todos os trabalhos apresentaram abordagens muito diferentes, mesmo os que usaram *puzzles* similares, e a maior parte teve bons resultados. Somente o trabalho de Weeks e Davis (2022) demonstrou que houve problemas nos resultados da abordagem utilizada.

De todos os trabalhos, somente o de Garcia (2022) tem menos aplicabilidade à proposta deste trabalho e todos os demais têm pelo menos um ponto em sua abordagem

que pode ser aplicado. Entretanto, três trabalhos têm ideias interessantes e que merecem um olhar mais cuidadoso.

O primeiro é o de Walker (2018) pela diversidade de *puzzles* que possui e pela maneira que os implementa no cenário do jogo. O segundo é o de Pereira, Viana e Toledo (2022) pela adaptação da GPC ao perfil do jogador. Por último, o trabalho de Kegel e Haahr (2019b), com seus *puzzles* narrativos, cria uma atmosfera no jogo que aproxima o jogador e aumenta consideravelmente sua variabilidade.

Os trabalhos revisados demonstraram as várias estratégias para gerar elementos de *puzzles*, abrangendo quebra-cabeças lógicos, enigmas ambientais e desafios de tomada de decisão. Essa variedade evidencia a capacidade da geração procedural em proporcionar experiências desafiadoras e distintas de jogo.

No próximo capítulo, será apresentado o desenvolvimento detalhado do trabalho, abordando a implementação completa do jogo. Serão discutidas as soluções adotadas para criar uma experiência desafiadora e envolvente, incluindo os algoritmos e técnicas de GPC utilizados e avaliação para o conteúdo gerado.

4 Desenvolvimento

Este capítulo descreve toda a implementação da topologia de um nível do jogo e dos modelos de *puzzles* selecionados, como os *puzzles* são integrados no jogo e como as técnicas de GPC se comparam a outras existentes na literatura. No final, será discutida a avaliação de todos os elementos gerados no jogo e também a avaliação por jogadores reais.

4.1 Visão geral do Sistema BlueMoon

O sistema BlueMoon foi criado como prova de conceito deste trabalho e se trata de um jogo *roguelike*, focado na resolução de *puzzles*. O jogador precisa atravessar uma masmorra gerada proceduralmente até chegar na sala que leva à batalha com um chefe. A masmorra gerada consiste em várias salas, cujas medidas de altura e comprimento variam entre 5 a 10 células, conectadas por corredores e populadas por elementos diversos. O jogador pode percorrer o cenário, interagir com os elementos do jogo usando uma espada ou atirar magia, limitada a cinco usos, para derrotar inimigos. O jogo termina se o jogador vencer o chefe da masmorra ou se o jogador perder os sete pontos de vida disponíveis. O jogador pode escolher se a masmorra terá 6, 12, 16 ou 20 salas. O método para gerar a topologia das masmorras consiste em posicionar aleatoriamente salas e conectá-las com corredores, que são decididos usando o algoritmo de Kruskal, fazendo com que a estrutura da masmorra seja de uma árvore geradora mínima e criando o chamado "grafo de mapa", cujos nós são as salas e as arestas os corredores. Definiu-se o custo das arestas como a quantidade de células existentes em um corredor, fazendo com que as salas sejam conectadas pelos menores corredores possíveis.

Os *puzzles* são inseridos dentro de salas por meio de agentes de *puzzles*, que são classificados em *puzzles* de obstáculo, que inserem desafios a serem resolvidos em mais de um local do jogo, e *puzzles* auxiliares, que criam desafios locais e aumentam em níveis de dificuldade pela proximidade que o jogador está do fim do jogo. Além dos agentes de *puzzles*, há também os chamados agentes comuns, que também populam as salas, mas que

não compõem os *puzzles*. O funcionamento dos elementos do jogo e como se relacionam é mostrado na Figura 4.1.

4.2 Implementação dos *Puzzles*

Esta seção traz detalhes da implementação dos *puzzles* de obstáculo, auxiliares e agentes comuns.

4.2.1 *Puzzles* de Obstáculo

Os *puzzles* de obstáculo contêm algum item de jogo (como chaves ou alavancas), que é necessário para liberar a progressão no conteúdo. Todos os *puzzles* de obstáculo possuem restrições para serem posicionados na masmorra. Além do item de jogo que é posicionado na sala, um elemento adicional, que requer o item para que se prossiga no jogo, é inserido em outro local.

Chave e Porta

Neste *puzzle*, uma chave é posicionada em uma sala, e uma porta, que será aberta por essa chave, é inserida em algum corredor da masmorra de forma aleatória, impedindo que

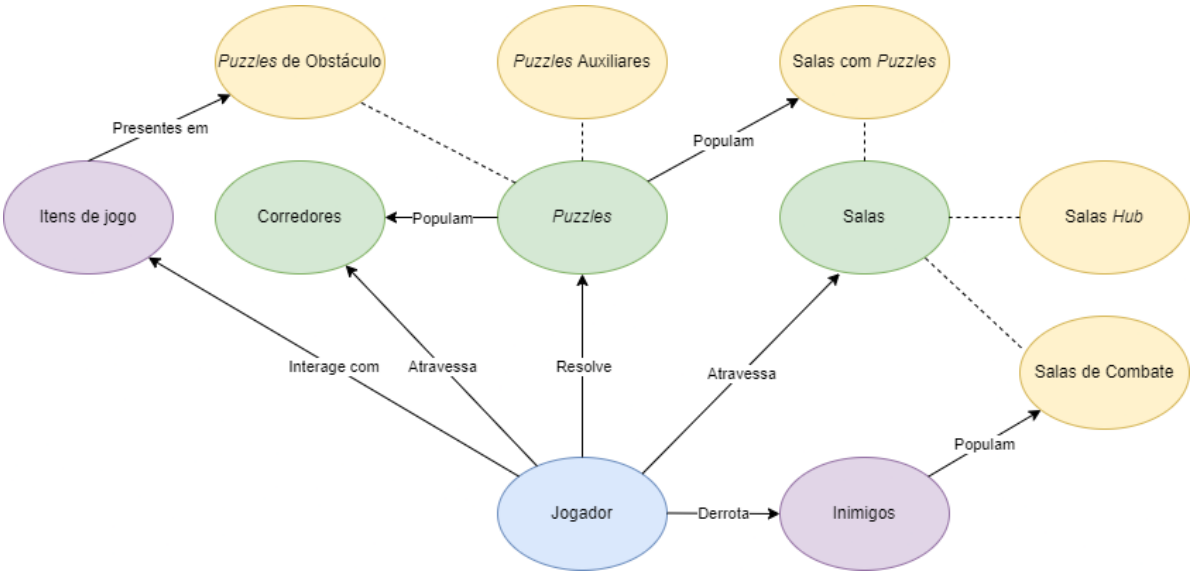


Figura 4.1: Visão geral dos elementos presentes no jogo BlueMoon. O jogador é identificado na cor azul; na cor verde, os elementos principais do jogo; em amarelo, subdivisões desses elementos; por fim, elementos menores do jogo na cor roxa. Fonte: do autor.

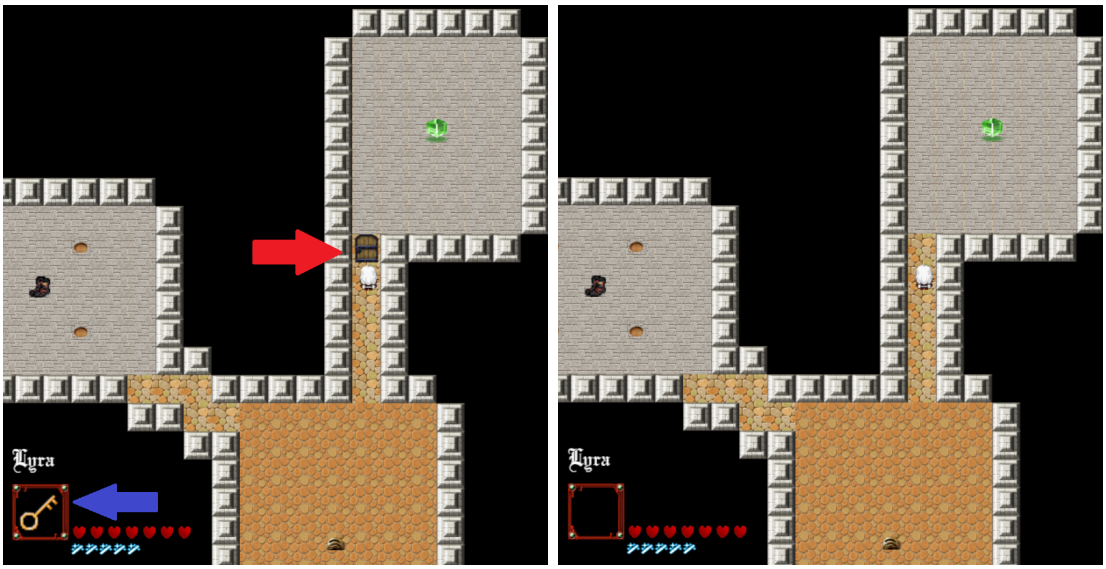
o jogador atravesse o corredor até ter adquirido a chave.

Como uma chave abre exatamente uma porta e uma porta só pode ser aberta por esta chave, sendo que, após usada, a chave desaparece do inventário do jogador, o elemento "chave" neste *puzzle* é considerado uma chave consumível e particular. A Figura 4.2 mostra como isso acontece dentro do jogo.

As restrições de posicionamento da porta são:

- deve ser possível encontrar todos os itens de jogo posicionados no jogo até então, mesmo com o corredor bloqueado pela porta;
- não deve haver outra porta no corredor;
- o corredor não deve dar acesso imediato à sala que se encontra a chave que a abre.

A primeira regra visa evitar que a porta crie um impasse com outros *puzzles* de obstáculo. Já a segunda e a terceira são decisões de *design* do jogo, a fim de tornar a resolução do *puzzle* mais interessante e menos óbvia.



(a) (b)

Figura 4.2: Demonstração da lógica de chave e porta. Na Figura 4.2a, o jogador possui a chave amarela no inventário (indicada pela seta azul) que abre a porta correspondente de mesma cor (indicada pela seta vermelha); na Figura 4.2b, ao abrir a porta, a chave desaparece do inventário e o caminho é liberado. Fonte: captura de tela do jogo, do autor.

Sala de Lava

O objetivo deste *puzzle* é preencher uma sala com lava, escolhida aleatoriamente entre as salas disponíveis, e posicionar uma bota anti-lava em outra sala que é explorada anteriormente à sala de lava, que permite o jogador adentrar a sala. É um *puzzle* que trabalha com a economia do jogo, ao lidar com recursos limitados do jogador (no caso, os pontos de vida). O jogador tem a opção de ignorar a lava e passar por ela, ao custo de perder uma ou mais vidas para ultrapassá-la sem a bota. Há, também, um dispositivo no meio da sala que, ao ser ativado, elimina toda a lava da sala e a torna acessível sem penalização, de forma que o jogador precisará passar pela lava apenas uma vez. A Figura 4.3 mostra como é adentrar a sala com a bota e o efeito de apertar o botão de apagar a lava.

Um detalhe importante é que a bota não é consumida em nenhum momento do jogo – apenas uma bota é posicionada em algum lugar do mapa, porém várias salas de lava podem existir, tornando a bota anti-lava uma espécie de chave persistente e não-particular. Nesta situação, caso o jogador opte por não perder nenhuma vida para atravessar a sala, ou o dano para atravessar a sala seja um dano fatal, encerrando assim o jogo, sala de lava se torna um obstáculo comparável ao *puzzle* de chave e porta, com a bota anti-lava sendo uma chave e a sala, enquanto elemento que impõe uma restrição para prosseguir no jogo, como uma porta.

Assim como o *puzzle* de chave e porta, este também possui restrições no posicionamento da sala de lava:

- deve ser possível encontrar todos os itens de jogo posicionados no jogo até então, mesmo sem acessar a sala de lava;
- nenhum agente pode ter populado a sala de alguma maneira;
- não pode ser uma sala terminal;
- não pode ser uma sala vizinha a outra já preenchida com lava;
- a sala de lava não deve ser vizinha à sala que contém a bota anti-lava.

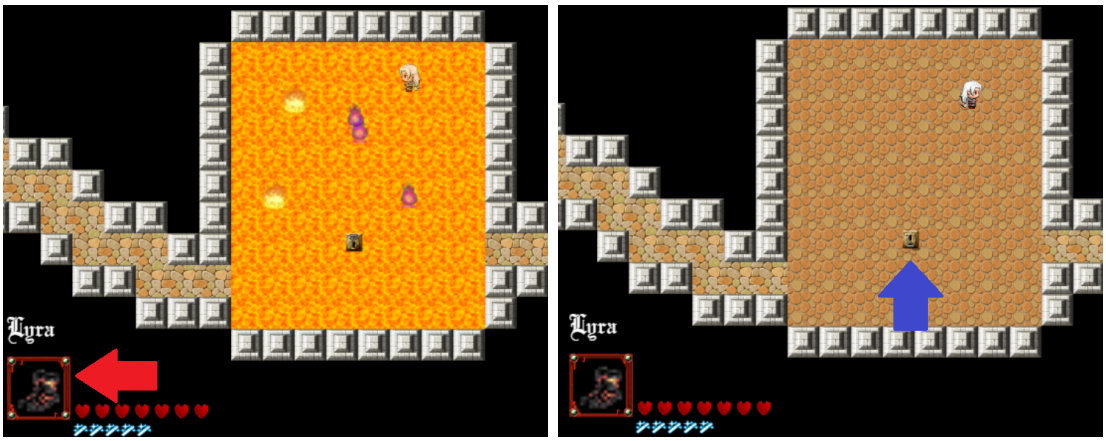
Assim como no *puzzle* anterior, a primeira regra visa evitar impasses com os outros *puzzles* de obstáculo. A segunda visa evitar que a sala de lava se sobreponha a

outra sala de lava ou algum outro *puzzle* que já tenha populado a sala. Apesar da sala de lava ser o único *puzzle* que seleciona salas aleatoriamente para popular, é tomado que, por este ser um jogo expansível, pode haver futuramente outro *puzzle* que também popule salas aleatoriamente. Portanto, esta regra evita a sobreposição de maneira geral. As demais regras são decisões de *design*, evitando que as salas de lava fiquem muito próximas ou sejam facilmente solucionáveis pela proximidade com a bota.

O agente responsável age tanto como um agente de *puzzle* de obstáculo (por posicionar a bota) como um agente de *puzzle* auxiliar (a lava que preenche a sala é considerada um agente de *puzzle* auxiliar). Portanto, também evolui em dificuldade com o andamento do jogo da seguinte forma:

- **nível 1**: nenhum elemento é adicionado;
- **níveis 2 a 4**: é adicionado um inimigo "fogo-fátuo" por nível.

Os inimigos "fogo-fátuo" são inimigos que se aproximam do jogador quando este se aproxima da sala de lava. Possuem a capacidade de transpassar paredes e espaços vazios, porém são apagados quando o dispositivo que apaga a lava da sala é ativado.



(a) (b)

Figura 4.3: Demonstração da sala de lava. Na Figura 4.3a, o jogador pode andar na sala por possuir o item de jogo "bota anti-lava" no inventário (indicado pela seta vermelha); na Figura 4.3b, ao apertar o dispositivo na parte inferior da sala (indicado pela seta azul), a lava é apagada e os inimigos "fogo-fátuo" desaparecem do cenário. A bota permanece no inventário. Fonte: captura de tela do jogo, do autor.

Alavanca

Para este *puzzle*, é posicionada uma alavanca na sala que alterna o posicionamento de um fosso entre dois corredores, escolhidos aleatoriamente. Ao ativar a alavanca, o fosso muda do corredor 1 para o corredor 2 e vice-versa ao ativar novamente a alavanca. A Figura 4.4 ilustra como os fossos se alternam ao acionar a alavanca.

Este *puzzle* foi o que exigiu mais regras de posicionamento pois, além de posicionar dois elementos bloqueantes (os fossos), corre-se também o risco de as alavancas causarem impasses, em que duas ou mais alavancas causam travamento no caminho a ponto de impedir o andamento do jogo, por mais que uma alavanca por si só não impeça o progresso do jogador. As regras para posicionamento são, tomando que o fosso 1 é que é ativado por padrão pela alavanca:

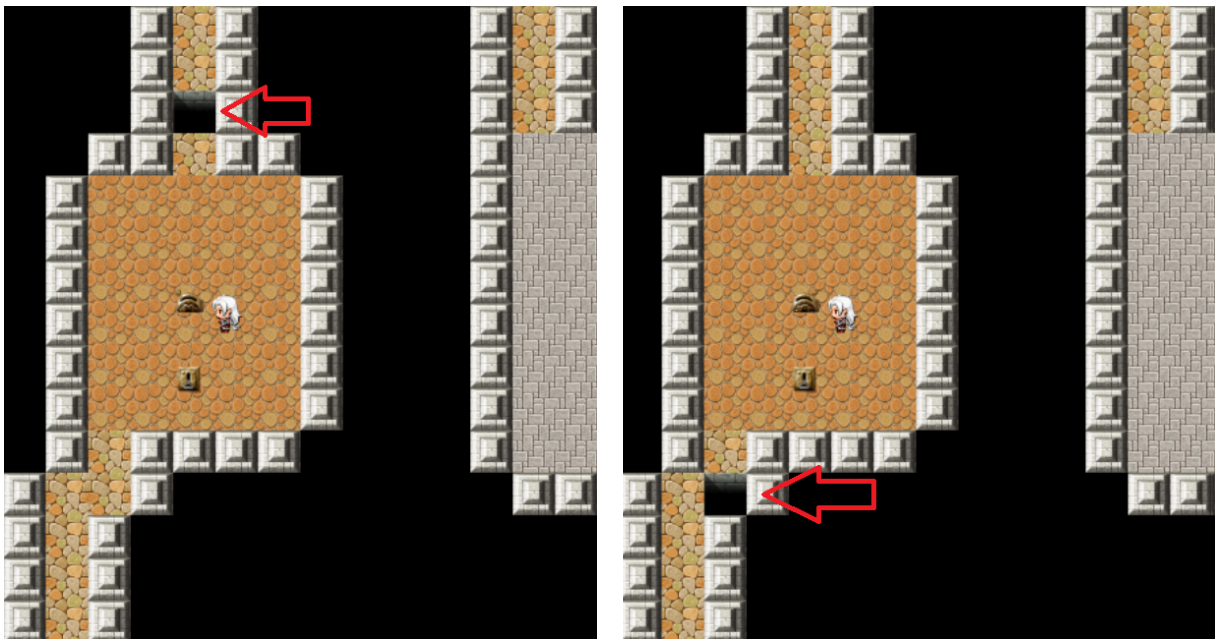
- deve ser possível encontrar todos os itens de jogo posicionados no jogo até então, mesmo com o corredor bloqueado pelo fosso 1;
- o caminho da alavanca até o fosso 1 não deve passar pelo corredor do fosso 2;
- o caminho da alavanca até o fosso 2 não deve passar pelo corredor do fosso 1;
- ao posicionar o fosso 2, não pode ser criado um impasse com as alavancas já existentes;
- não deve haver outro fosso posicionado no corredor ao posicionar os fossos 1 e 2, sendo este fosso 1 ou fosso 2 de qualquer alavanca.

Aqui, foram necessárias as regras de 1 a 4 para evitar impasses e a regra 5 como decisão de *design*. Neste caso, como os fossos possuem todos a mesma imagem, caso houvesse mais de um fosso por corredor poderia confundir e frustrar o jogador.

Para a implementação da regra 4, foi usada uma estratégia similar a que é usada no trabalho de Venco e Lanzi (2021), que usa, para o posicionamento global dos *puzzles*, um grafo de *puzzle* em que cada nó do grafo possui o estado atual de cada *puzzle*. Para este trabalho, não foi necessário guardar de todos os *puzzles*, apenas das alavancas, pois os demais *puzzles* não conseguem gerar impasses com outros. O funcionamento é detalhado no Algoritmo 1.

Algoritmo 1: Verificação de *impasse* entre alavancas

```
Resultado: Retornar verdadeiro se houver impasse, falso caso contrário
Inicializar salaInicial ← Sala de início do jogo;
estadosVisitados ← Conjunto de estados visitados;
salasVisitadas ← Conjunto de salas visitadas pelo algoritmo;
estadoDasAlavancas ← Array de tamanho igual a quantidadeDeAlavancas inicializado com 1s;
estado ← Inicializar estado com a sala igual a salaInicial e estadoDasAlavancas;
fila ← Fila de estados com estado;
while fila não está vazia do
    estadoAtual ← remover o primeiro elemento de fila;
    Inicializar salasVisitadasDoEstado, salasComAlavancaDoEstado, e filaDoEstado com estadoAtual.sala;
    while filaDoEstado não está vazia do
        salaAtual ← remover o primeiro elemento de filaDoEstado;
        Adicionar salaAtual a salasVisitadas e salasVisitadasDoEstado;
        if salaAtual contém uma alavanca then
            Adicionar salaAtual a salasComAlavancaDoEstado;
        end if
        for Todas as arestas corredor saindo de salaAtual do
            if corredor contém um fosso de alavanca e estadoAtual.estadoDasAlavancas[idDaAlavancaDoFosso] não é igual a numeroDoFosso then
                outraSala ← encontrar a outra sala na aresta saindo de salaAtual;
                if salasVisitadasDoEstado não contém outraSala then
                    Adicionar outraSala a filaDoEstado;
                end if
            end if
        end for
    end while
    for Cada salaComAlavanca em salasComAlavancaDoEstado do
        novoEstadoDasAlavancas ← copiar estadoAtual.estadoDasAlavancas;
        novoEstadoDasAlavancas[salaComAlavanca.idDaAlavanca] ← (novoEstadoDasAlavancas[salaComAlavanca.idDaAlavanca] == 1) ? 2 : 1;
        if Nenhum estado em fila tem estadoDasAlavancas igual a novoEstadoDasAlavancas e nenhum estado em estadosVisitados tem estadoDasAlavancas igual a novoEstadoDasAlavancas then
            Adicionar {sala: salaDaAlavanca, estadoDasAlavancas: novoEstadoDasAlavancas} a fila;
        end if
    end for
    Adicionar estadoAtual a estadosVisitados;
end while
return Tamanho de salasVisitadas não é igual a quantidade de salas;
```



(a) (b)

Figura 4.4: Demonstração da lógica das alavancas, alterando a localização do fosso (indicado por setas) dependendo do estado da alavanca. O estado da alavanca é visível ao jogador. Fonte: captura de tela do jogo, do autor.

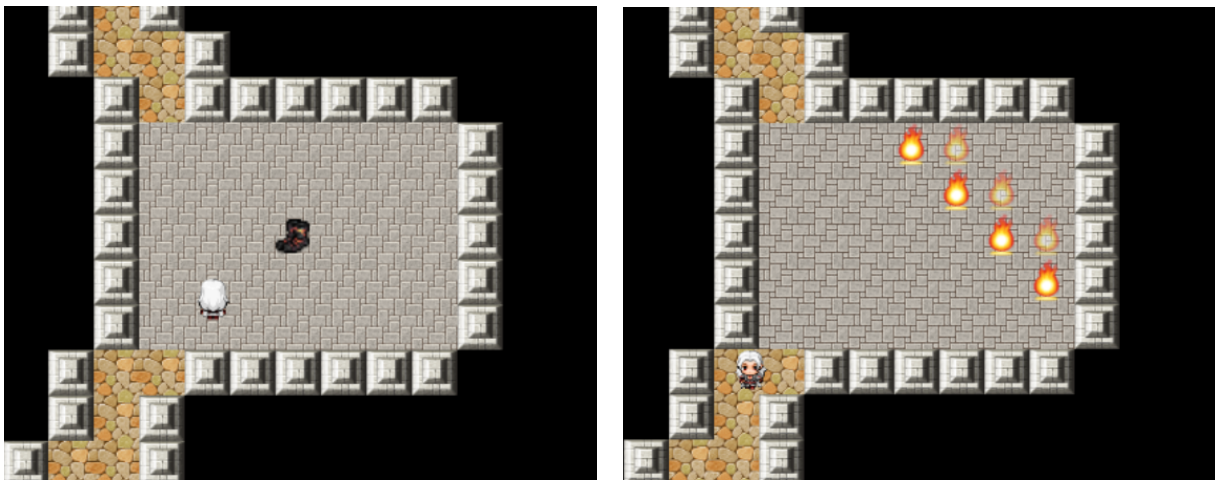
4.2.2 *Puzzles* Auxiliares

Os *puzzles* auxiliares vêm para complementar a diversidade dos *puzzles* de obstáculo, adicionando desafios a serem solucionados pelo jogador naquela sala para que o item de jogo possa ser adquirido. Eles se tornam mais difíceis com a progressão do jogador no jogo, incluindo mais inimigos e elementos que podem surpreender e tornar mais complexa a conclusão do *puzzle* pelo jogador.

Armadilha de Bola de Fogo

Neste *puzzle*, o item de jogo é centralizado na sala e, ao ser coletado ou acionado pela primeira vez, a sala é inundada pouco a pouco por bolas de fogo que podem ferir o jogador. As bolas de fogo somem rapidamente, tornando a sala acessível novamente. A primeira bola de fogo aparece a partir de uma célula de canto da sala, que é escolhida pela maior distância euclidiana para todas as saídas disponíveis na sala. Novas bolas de fogo surgem das células vizinhas a esta, e este processo se repete até que todas as células da sala tenham tido uma bola de fogo. A Figura 4.5 ilustra como é a mecânica deste *puzzle*.

Por ser um *puzzle* auxiliar, possui níveis de dificuldade, que aumentam à me-



(a) (b)

Figura 4.5: Demonstração da armadilha de bola de fogo. Na Figura 4.5a, o jogador coleta a bota anti-lava na sala e, na Figura 4.5b, as bolas de fogo inundam a sala como uma onda, obrigando o jogador a fugir para a saída mais próxima. Fonte: captura de tela do jogo, do autor.

dida que a sala com este *puzzle* é definida mais próxima do fim do jogo. No caso, a dificuldade se dá pelo tempo que as bolas de fogo inundam a sala - no nível 1 demoram mais tempo para inundar e a cada nível inundam mais rápido, fazendo com que no nível 4 a sala seja inundada rapidamente, exigindo mais do reflexo do jogador para correr até uma das saídas da sala.

Sala de Gelo

Para este *puzzle*, o terreno da sala é coberto de gelo, fazendo com que o jogador deslize ao se movimentar.

A progressão de dificuldade para este *puzzle* segue as seguintes regras:

- **nível 1**: nenhum elemento é adicionado. Item de jogo estará centralizado na sala;
- **níveis 2 e 3**: dois inimigos, de nível que pode ir de 1 até o nível deste *puzzle*-1, aparecem na sala. Se o item de jogo não puder ser um *drop*, será centralizado na sala. Senão, se tornará *drop* de um dos inimigos;
- **nível 4**: além do que acontece nos níveis 2 e 3, adiciona espinhos em locais próximos aos cantos diagonais da sala que podem machucar o jogador.

O termo *drop* é frequentemente utilizado para se referir aos itens ou recom-

pensas que um jogador recebe ao derrotar um inimigo. Portanto, para conseguir o item de jogo a partir do nível 2 se este puder ser um *drop*, o jogador deve derrotar o inimigo e, assim, o item aparecerá no local onde estava o inimigo derrotado. Atualmente, somente a alavanca não é considerado um *drop*. Na Figura 4.6, é mostrada uma sala de gelo com uma alavanca centralizada.

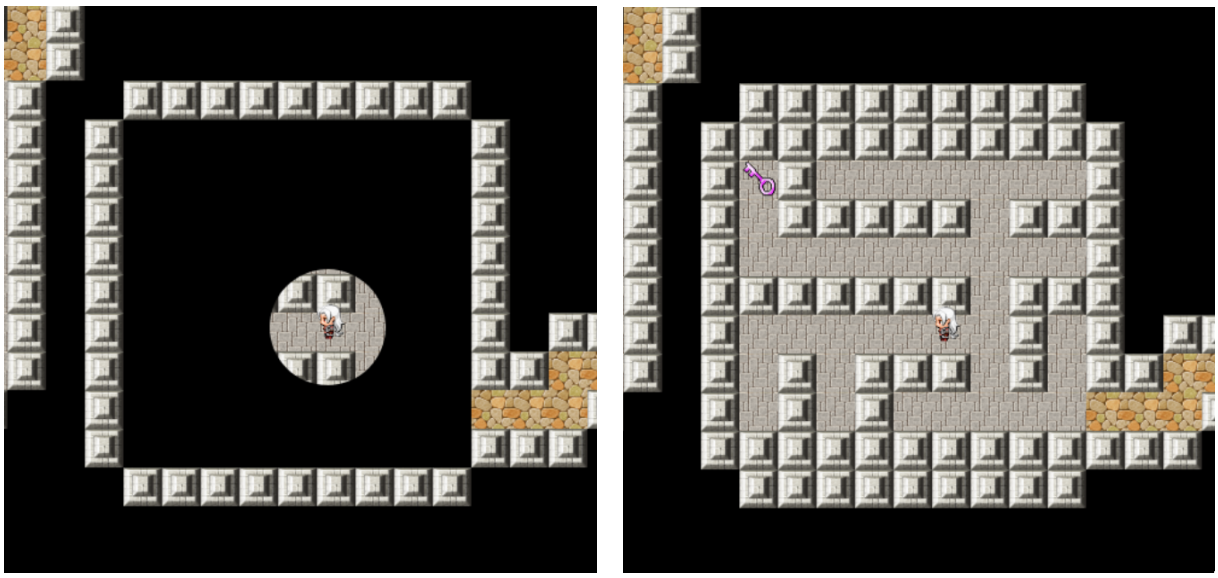
Figura 4.6: Demonstração da sala de gelo. Como a alavanca não pode ser um *drop*, fica centralizada na sala. Fonte: captura de tela do jogo, do autor.

Labirinto

Neste *puzzle*, o jogador precisa encontrar o item de jogo no fundo de uma sala escura que possui um labirinto dentro. O labirinto é gerado usando algoritmo de Prim. Ao entrar na sala, que inicialmente é totalmente escura, o jogador terá uma visibilidade limitada, enxergando apenas em um pequeno círculo ao seu redor. Este *puzzle* é inserido apenas em salas terminais. O item é inserido na célula mais distante da entrada, que é escolhida por meio de algoritmo de Busca em Largura (BFS). A Figura 4.7 mostra como é a mecânica da limitação de visão e um exemplo de labirinto gerado.

Para a progressão de dificuldade, segue-se as seguintes regras para nivelamento:

- **nível 1**: nenhum elemento é adicionado;
- **níveis 2 a 4**: um inimigo, de nível que pode ser de 1 até o nível deste *puzzle*-1, aparece na célula de entrada da sala.



(a) (b)

Figura 4.7: Exemplo de labirinto gerado. A Figura 4.7a mostra como é a visão para o jogador ao adentrar a sala; a Figura 4.7b, o mesmo labirinto, sem a restrição de visão ao jogador. Fonte: captura de tela do jogo, do autor.

Sokoban

Neste *puzzle*, o jogador precisa resolver um problema de *sokoban* que está dentro da sala. Após resolvido, o item de jogo aparece no centro da sala. Este *puzzle* somente pode aparecer em salas terminais, assim como o labirinto.

Para gerar o tabuleiro *sokoban*, foi usada uma adaptação do trabalho de Kartal, Sohre e Guy (2016). A adaptação se fez necessária pois, no trabalho de referência, os autores iniciam o tabuleiro com o jogador no centro, enquanto neste trabalho o jogador começa numa célula de canto da sala. Além disso, enquanto no trabalho original a sala se inicia preenchida com células de parede, inserindo células vazias e caixas como movimentos possíveis da MCTS, observou-se melhores resultados para este trabalho começando com um tabuleiro de células vazias, encaixando as caixas e preenchendo com parede as caixas não empurradas no tabuleiro e espaços vazios não visitados pelo jogador.

Em questão de progressão, este é o único *puzzle* que não apresenta aumento da dificuldade com níveis. Essa decisão foi necessária, pois: i) o algoritmo MCTS não consegue garantir um limiar preciso de qualidade no tempo estimado de geração, e ii) os autores do trabalho de referência admitem que, apesar da capacidade de gerar bons resultados, tabuleiros que tiveram notas mais altas muitas vezes não eram mais difíceis ao

jogador do que outros tabuleiros com notas mais baixas. Portanto, usar a nota do tabuleiro como parâmetro para a progressão de dificuldade não traria o resultado esperado.

A MCTS foi configurada para criar 1000 nós na árvore, fazendo com que a geração dos tabuleiros levasse, em média, um segundo de duração. Por este motivo, a geração do *sokoban* é o processo mais demorado na inicialização do jogo. Na Figura 4.8 é mostrado um exemplo de tabuleiro gerado com quatro caixotes.

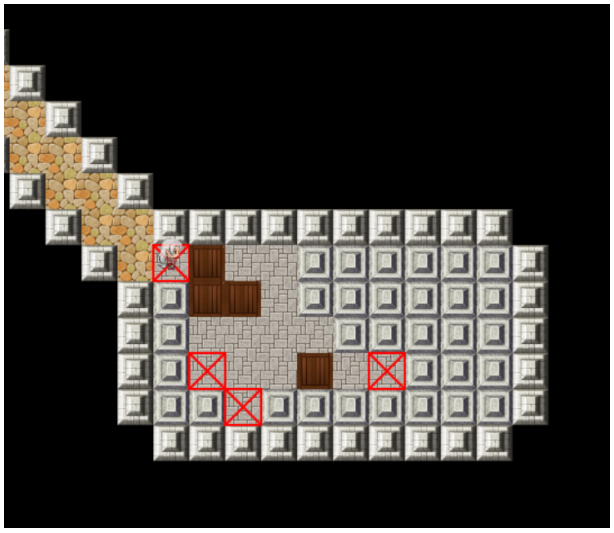


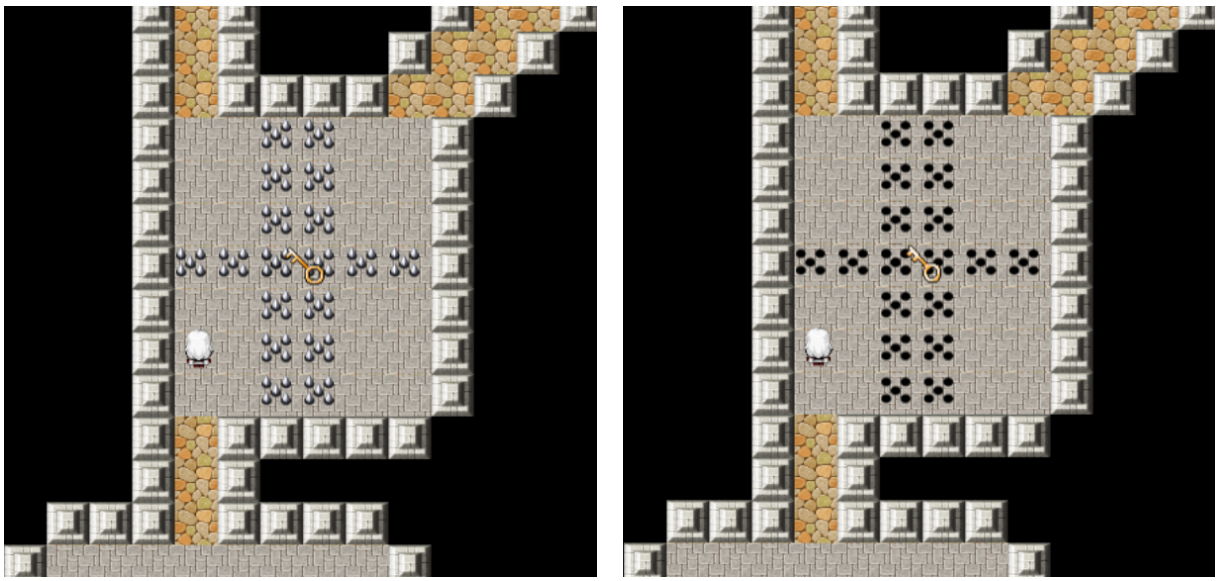
Figura 4.8: Exemplo de tabuleiro *sokoban* gerado. O jogador tem que empurrar os caixotes para as regiões marcadas. Fonte: captura de tela do jogo, do autor.

Espinhos

Este *puzzle* consiste em espinhos espalhados pela sala, que sobem e descem em um intervalo de tempo definido. Caso o jogador toque em algum espinho levantado, será ferido. O item de jogo é centralizado na sala, devendo o jogador atravessar os espinhos para coletá-lo ou atravessar a sala para se dirigir a outra sala. A Figura 4.9 mostra como é o comportamento dos espinhos.

A progressão de dificuldade deste *puzzle* se dá seguindo as seguintes regras:

- **nível 1**: espinhos preenchem a sala formando uma cruz;
- **nível 2**: espinhos preenchem a sala formando uma cruz, porém sobem e descem mais rápido do que no nível 1. Um inimigo, de nível que vai de 1 até o nível deste *puzzle*-1, aparece na sala;



(a) (b)

Figura 4.9: Demonstração do funcionamento dos espinhos. Na Figura 4.9a, os espinhos estão levantados e na Figura 4.9b os espinhos estão abaixados, permitindo que o jogador passe sem se machucar. Fonte: captura de tela do jogo, do autor.

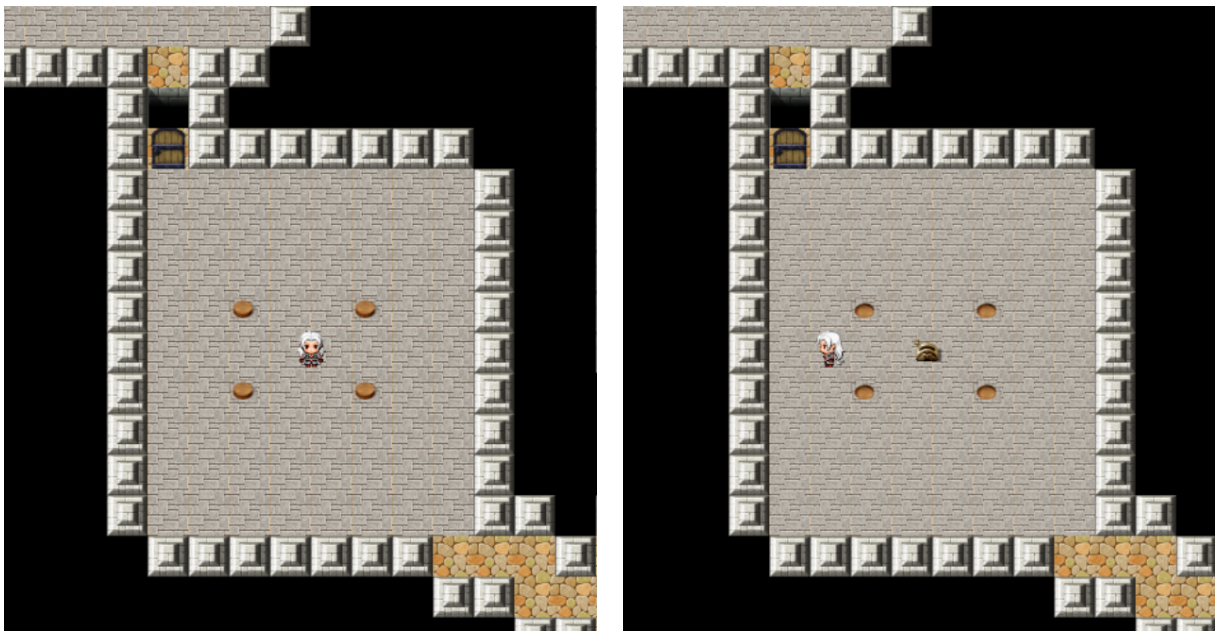
- **nível 3**: espinhos preenchem totalmente a sala. Um inimigo, de nível que vai de 1 até o nível deste *puzzle*-1, aparece na sala;
- **nível 4**: espinhos preenchem totalmente a sala, porém sobem e descem mais rápido do que no nível 3. Um inimigo, de nível que vai de 1 até o nível deste *puzzle*-1, aparece na sala.

Interruptores

Neste *puzzle*, o jogador precisa apertar quatro interruptores em sequência. Caso os aperte na ordem correta, o item de jogo aparece no centro da sala; se algum interruptor for apertado na ordem incorreta, todos os interruptores voltarão ao estado inicial. Cada conjunto de interruptores tem sua ordem em que devem ser apertados decidida quando o jogo é gerado por meio do algoritmo de Fisher-Yates, que consiste em realizar permutações em um conjunto para que seus elementos assumam uma ordem aleatória (WIKIPEDIA, 2007). A Figura 4.10 mostra como é o funcionamento deste *puzzle*, antes e depois do jogador apertar os interruptores na ordem correta.

Para progressão de dificuldade, usa-se as seguintes regras neste *puzzle*:

- **nível 1**: nenhum elemento é adicionado;



(a) (b)

Figura 4.10: Demonstração do funcionamento dos interruptores. Na Figura 4.10a, os interruptores não estão apertados; na Figura 4.10b, ao apertar os quatro interruptores na ordem definida pelo algoritmo de Fisher-Yates, o item de jogo aparece no centro da sala. Fonte: captura de tela do jogo, do autor.

- **nível 2**: um inimigo, de nível que pode ir de 1 até o nível deste *puzzle*-1, aparece na sala.
- **níveis 3 e 4**: dois inimigos, de níveis que podem ir de 1 até o nível deste *puzzle*-1, aparecem na sala.

4.2.3 Agentes comuns

Esta seção descreve os agentes que inserem elementos que são comuns a jogos do gênero *roguelike*, mas não possuem relação com *puzzles*. Se dividem em salas de combate e salas *hub*, que servem como ponto de descanso para o jogador.

Sala de Combate

Este agente tem como responsabilidade posicionar dois inimigos dentro da sala. São definidas duas restrições para que este agente possa atuar:

- nenhum agente pode ter populado a sala de alguma maneira;

- não pode ser uma sala terminal.

A primeira regra visa evitar sobreposição com algum *puzzle* já posicionado na sala. A segunda é por questão de *design*, pois a sala de combate é, em teoria, uma sala intermediária no caminho até o final. Portanto, não faria sentido que fosse uma sala com apenas um corredor de acesso, pois seria uma sala evitável para o jogador, sem que houvesse qualquer penalidade.

Os tipos de inimigos possíveis variam do nível 1 até o nível do agente. Estes mesmos inimigos aparecem nos *puzzles* auxiliares, quando aplicável, seguindo as mesmas regras de nivelamento. Estes inimigos são:

- **nível 1**: caveira ou diabinho, ambos são inimigos lentos com 1 ponto de vida;
- **nível 2**: morcego. É um inimigo veloz e possui 1 ponto de vida;
- **nível 3**: orc, um inimigo lento, porém possui 2 pontos de vida;
- **nível 4**: necromante, inimigo imóvel, porém invoca inimigos "caveira" em um intervalo de tempo determinado. Possui 1 ponto de vida.

A Figura 4.11 mostra uma sala com dois inimigos "necromante" e três inimigos "caveira" que foram invocados pelos necromantes.

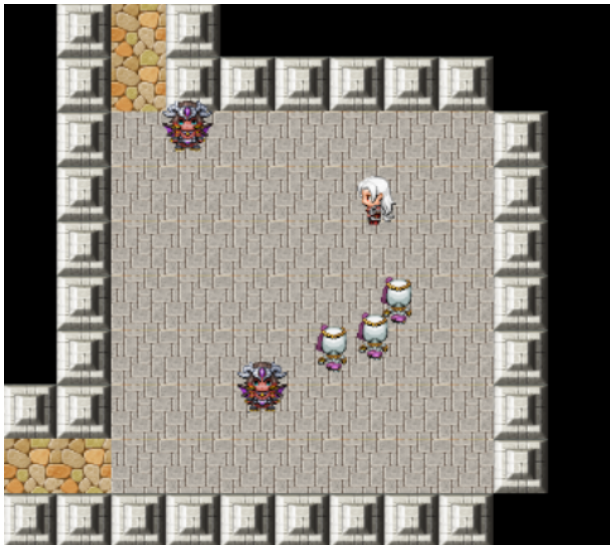


Figura 4.11: Demonstração da sala de combate, em que o jogador enfrenta dois inimigos "necromante", que invocam inimigos "caveira". Fonte: captura de tela do jogo, do autor.

Sala *Hub*

É considerada uma "sala de descanso", onde o jogador não tem que resolver *puzzles* ou enfrentar inimigos. São definidas três restrições para que este agente possa atuar:

- nenhum agente pode ter populado a sala de alguma maneira;
- não pode ser uma sala terminal.
- não pode ser uma sala vizinha à sala inicial do jogo.

Assim como o agente de sala de combate, a primeira regra visa evitar sobreposição com algum *puzzle* já posicionado na sala. A segunda também é por questão de *design*, pois também é uma sala considerada intermediária no caminho até o final. Assim, caso seja uma sala terminal, a sala seria isolada e desperdiçada para o jogo. Já a terceira regra existe pela própria definição do agente: sendo uma "sala de descanso", se ficar imediatamente após o começo do jogo, não houve *stress* ao jogador que justifique estar logo após o jogo começar.

Este agente também possui regras de nivelamento. Portanto, maiores níveis denotam quão mais próxima a sala está do fim do jogo. A vantagem ao jogador é poder recuperar pontos de vida ou de magia em alguns níveis do agente. As regras de nivelamento para isso são:

- **níveis 1 e 2**: a sala estará vazia, sem adição de elementos;
- **nível 3**: um coração vermelho, que recupera 1 ponto de vida, está disponível para o jogador;
- **nível 4**: um coração vermelho e um azul, que recuperam 1 ponto de vida e 1 ponto de magia, respectivamente, estão disponíveis para o jogador.

4.3 Planejamento Global

O planejamento global para posicionamento dos *puzzles* se divide em três fases: definição dos *puzzles* em cada uma das salas; nivelamento dos *puzzles* auxiliares tomando em consideração o andamento do jogo; e construção dos *puzzles* de acordo com o nível.

4.3.1 Definição dos *puzzles* nas salas

Nesta primeira fase, são definidos quais agentes irão construir *puzzles* em quais salas.

Na etapa de preparação, é decidida uma sala inicial para o jogo, com a restrição que tenha um ou dois corredores conectados. Após isso, as demais salas são ordenadas pela distância da sala inicial no grafo de mapa. Depois, são removidas da ordenação as salas que tenham uma quantidade de corredores igual ou superior ao número de agentes principais, pois ao posicionar um agente nesta sala, ao menos uma sala vizinha não poderia posicionar um agente, já que todas as salas possuem agentes, tanto principais quanto auxiliares, diferentes das salas vizinhas. Isso faz com que essas salas se tornem salas de descanso vazias.

Após a etapa de preparação, o algoritmo define os agentes que atuarão em quais salas, seguindo a ordenação das salas decidida na etapa anterior, e, portanto, usando como estratégia popular as salas a partir das mais próximas da sala inicial. São definidos dois grupos de agentes para popular as salas: os agentes principais (compostos pelos *puzzles* de obstáculo e agentes comuns) e os agentes auxiliares (compostos exclusivamente pelos *puzzles* auxiliares).

Para cada uma das salas, é sorteado um agente por vez. Antes de cada sorteio, são removidos do sorteio todos os agentes já sorteados em salas vizinhas da sala analisada.

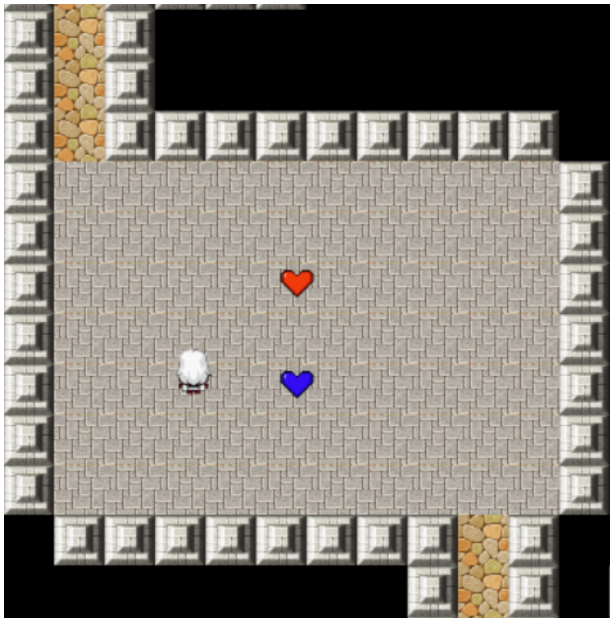


Figura 4.12: Demonstração de um *hub* com um coração vermelho e um azul disponíveis ao jogador. Fonte: captura de tela do jogo, do autor.

O sorteio é feito por meio de uma loteria ponderada, em que a chance é reduzida a um fator próprio de cada agente a cada vez que um agente é sorteado. A chance de sorteio e o fator de redução estão dispostos na Tabela 4.1.

Agente	**Chance inicial de sortear**	**Fator de Redução**
Sala de Lava	1	0.5
Alavanca	0.5	0.1
Sala de Combate	1	0.5
Hub	0.5	0.5
Chave e Porta	1	0.5

Tabela 4.1: Chances e fator de redução de chance para cada agente principal

A diferença existente para o agente de Alavanca e o agente de *Hub* é pelo primeiro gerar alta carga cognitiva ao jogador, pois os fossos e as alavancas não possuem distinção gráfica entre si e, portanto, o jogador precisa memorizar a posição e o estado de cada um, justificando que hajam poucas alavancas, se possível. Para o *Hub*, deseja-se que sejam posicionados após o jogador ter passado por algumas salas para que se tenha uma justificativa para uma sala de descanso. Lembrando que a intenção não é impedir completamente que isso aconteça nas salas mais próximas ao começo e sim reduzir a chance de escolha do agente. Portanto, ainda pode acontecer destes agentes serem posicionados mais próximos à sala inicial, principalmente quando outros agentes não puderem ser selecionados.

Após o sorteio, verifica-se se o agente pode ser posicionado na sala. Se o agente pode ou não ser posicionado depende das restrições de cada um dos agentes. Caso não possa ser sorteado, o agente é removido da lista de agentes disponíveis para sorteio daquela sala; caso possível, o agente é posicionado e sua chance de sorteio é multiplicada pelo fator de redução do agente. Este processo é repetido até que pelo menos um agente principal seja sorteado. Se nenhum agente puder ser escolhido, a sala fica vazia e o sorteio passa para a próxima sala.

Por último, é sorteado um agente auxiliar. Este sorteio acontece caso não haja um agente auxiliar já posicionado na sala (como é o caso da sala de lava, que já pode ter posicionado a lava na sala em questão). O sorteio de agentes auxiliares segue a mesma estratégia de sorteio dos agentes principais. A diferença é que todos os agentes auxiliares possuem chance inicial de sorteio de 1 e fator de redução de 0.5, além de que

não há restrições quanto ao posicionamento de nenhum deles quando as salas não possuem nenhum agente auxiliar.

Quando todas as salas tiverem tido seus sorteios de agentes principais e auxiliares, esta fase é encerrada. Mais detalhes estão disponíveis no Algoritmo 2.

Algoritmo 2: Definição de agentes nas salas

```
Dados: salasOrdenadas, agentesPrincipais, agentesAuxiliares, grafoDeMapa
Resultado: Salas com agentes principais e auxiliares definidos
Entrada: Lista de salas ordenadas salasOrdenadas
Saída  : Geração de tags de agentes para as salas
1  for indiceDaSala ← 0 to comprimento de salasOrdenadas do
2      sala ← salasOrdenadas[indiceDaSala];
3      if sala não possui nenhum agente then
4          agentesDisponiveis ←
             verificarAgentesDisponiveis(sala, agentesPrincipais);
5          while comprimento de agentesDisponiveis > 0 do
6              agente ← sortearAgente(agentesDisponiveis);
7              if !agente.posicionarAgentePrincipal(grafoDeMapa, sala) then
8                  Remover agente de agentesDisponiveis;
9              end if
10             else
11                 agente.chanceDeSorteio ←
                     agente.chanceDeSorteio × agente.fatorDeReducao;
12                 if sala não possui agente auxiliar then
13                     posicionarAgenteAuxiliar(sala);
14                 end if
15                 break;
16             end if
17         end while
18     end if
19 end for
```

4.3.2 Nivelamento dos *puzzles* auxiliares

Após definir quais *puzzles* preencherão quais salas, inicia-se a fase de nivelamento. O nivelamento dá diversidade e reduz a previsibilidade, para que, mesmo que o jogador já tenha visto um determinado *puzzle* na partida ou em outra que tenha jogado, o *puzzle* possa ser diferente, surpreendendo o jogador e, assim, melhorando a experiência da partida. Além disso, cria-se uma progressão de dificuldade.

O nivelamento considera o andamento de todos os *puzzles* disponíveis no jogo. Por exemplo, se uma sala é vizinha à sala inicial, mas existe uma porta impedindo seu

acesso, todas as salas pelas quais o jogador precisa passar até adquirir a chave que abre esta porta serão de nível menor ou igual ao da sala que está restrita pela porta, por mais que ela seja mais próxima da sala inicial. Podemos definir o nivelamento como uma ordem possível de visitação das salas pelo jogador dentro do jogo – o jogo pode permitir ordens de visitação diferentes quando não há restrições para visitar duas salas desimpedidas.

Começando com o primeiro elemento de uma fila de verificação, a sala definida para iniciar o jogo, verifica-se se é possível adentrar cada uma das salas vizinhas. Definimos essa possibilidade se o jogador já tiver acesso a todos os itens de jogo que permitam adentrá-la quando chegar até a sala ou se em algum momento após chegar à sala os itens necessários foram adquiridos. Se houver algum impedimento para acessar a sala vizinha, é criada uma "restrição" para aquela sala, que inclui todas as restrições para acessá-la. Cada uma das restrições da sala pode ser removida se o jogador já tiver o item para satisfazer a restrição. Verifica-se em seguida se não há mais restrição alguma para acesso e, neste cenário, a sala vizinha é adicionada para a fila de verificação. No que tange os itens de jogo, se a sala vizinha possuir um, este é adicionado à lista de itens e são removidas as restrições em cada sala que dependem deste item. Caso alguma sala não possua mais nenhuma restrição, ela é adicionada à fila de verificação. Após analisar todas as salas vizinhas, a sala atual é adicionada a uma lista de salas visitadas e é analisada a próxima sala da fila de verificação. Repete-se esse processo até que a fila esteja vazia. Por fim, o algoritmo retorna as salas por ordem de visitação - ou seja, a ordem em que foram adicionadas à lista de salas visitadas. O pseudocódigo do funcionamento deste processo está disponível no Algoritmo 3.

4.3.3 Construção dos *puzzles* por nível

Após a etapa de nivelamento, os *puzzles* auxiliares e agentes comuns são construídos nas salas. De acordo com a posição da sala na ordenação da fase anterior recebem um nível, que vai de 1 até 4. A Equação 4.1 mostra como é calculado o nível, com *N(x)* sendo o nível do agente da sala, *P(x)* a posição da sala na ordenação e Q a quantidade de salas do jogo.

Algoritmo 3: Nivelamento das salas

```
Entrada: grafoDeMapa
Saída  : salasVisitadas
Dados  : Inicializar visitados, fila com salaInicial, itens, restricoes
1  while a fila não está vazia do
2      salaAtual ←remover o primeiro elemento da fila;
3      if a salaAtual possui restrições then
4          restricao ← { sala: salaAtual, restricoes: [... cópia das restrições da
             salaAtual]};
5          Adicionar restricao a restricoes;
6          foreach item em itens do
7              verificarRestricoesDaSala (item, restricao);
8          end foreach
9          if restricao.restricoes estão vazias then
10             remover restricao da lista de restrições;
11         end if
12     end if
13     restricao ← primeiro elemento em restricoes com sala == salaAtual
14     if restricao é nulo then
15         if salaAtual possui um item de jogo then
16             Adicionar item a itens;
17             verificarDependenciasDoItem(salaAtual, restricoes, fila);
18         end if
19         corredoresNaoVisitados ← cada corredor a partir de salaAtual
             onde a sala do outro lado do corredor não está em visitados;
20         foreach corredor em corredoresNaoVisitados do
21             vizinho ← grafoDeMapa.trazerOutraSalaDoCorredor(corredor,
                 salaAtual);
22             if corredor possui restrições then
23                 restricao ← { sala: vizinho, restricoes: [... cópia das restrições
                     do corredor]};
24                 foreach item em itens do
25                     verificarRestricoesDaSala (item, restricao);
26                 end foreach
27                 if restricao.restricoes estão vazias then
28                     Adicionar vizinho a fila;
29                 end if
30                 else
31                     Adicionar restricao a restricoes;
32                 end if
33             end if
34             else
35                 Adicionar vizinho a fila;
36             end if
37         end foreach
38         Adicionar salaAtual a salasVisitadas;
39     end if
40 end while
41 return salasVisitadas;
```

$$N(x) = \lceil \frac{P(x)}{Q-2} \times 4 \rceil \tag{4.1}$$

Naturalmente, ao menos uma sala não terá nenhum elemento posicionado, dadas as restrições dos agentes. Então, a sala que estiver mais ao fim da ordenação e não tiver nenhum agente posicionado é decidida como a sala de chefe, em que é posicionado um teleporte que leva à batalha contra o chefe do jogo. Caso alguma outra sala não tenha elementos posicionados, será transformada em um *hub* melhorado, em que são posicionados os seguintes elementos, dependendo do nível da sala:

- **nível 1**: um coração vermelho, que recupera 1 ponto de vida, estará disponível para o jogador;
- **nível 2**: um coração vermelho e um azul, que recuperam 1 ponto de vida e 1 ponto de magia, respectivamente, estarão disponíveis para o jogador.
- **nível 3**: uma aura laranja, que recupera todos os pontos de vida, estará disponível para o jogador;
- **nível 4**: uma aura laranja e uma azul, que recuperam todos os pontos de vida e magia, respectivamente, estarão disponíveis para o jogador;

Como estas salas são salas isoladas, com apenas um corredor conectando a outras, sendo salas opcionais e muito provavelmente mais próximas do fim do jogo, justifica-se que possuam elementos mais robustos para facilitar o progresso do jogador.

4.4 Avaliação

A fim de avaliar a experiência do jogo a partir da ótica do jogador, foi desenvolvido um formulário dividido em duas seções para ser preenchido por jogadores reais após testarem o jogo. A primeira seção continha perguntas de múltipla escolha sobre quais configurações de jogo os testadores jogaram (6, 12, 16 ou 20 salas), quantas partidas jogaram (1, 2, 3 ou acima de 3 partidas) e uma pergunta dicotômica com sim ou não como opções, para identificar se o jogador conseguiu ou não chegar ao chefe do jogo alguma vez. A segunda

seção continha vinte e uma afirmações em escala *Likert*[13], com cinco opções de escolha cada, sendo que as mais próximas à primeira opção indicavam maior discordância da afirmação, e as mais próximas à última indicavam uma maior concordância. Como as partidas formam um conjunto de *puzzles* possíveis de aparecer, as afirmações relacionadas especificamente a cada *puzzle* não tinham teor obrigatório, mas os jogadores deveriam responder pelos *puzzles* que encontraram ao jogar. As afirmações são:

AF1 Eu me senti imerso no jogo enquanto jogava.

AF2 Senti vontade de jogar o jogo mais de uma vez.

AF3 Eu gostei de jogar o jogo.

AF4 Os desafios e *puzzles* do jogo são intuitivos e consegui compreender como funcionam.

AF5 Senti que há uma boa distribuição dos *puzzles* pelas salas.

AF6 Senti aumento da dificuldade dos *puzzles* com a progressão do jogo.

AF7 Os elementos do jogo me surpreenderam mais de uma vez.

AF8 Senti que o jogo depende mais de habilidade do que de memorização.

AF9 Foi desafiador o *puzzle sokoban*.

AF10 Foi desafiador o *puzzle* de chave e porta.

AF11 Foi desafiador o *puzzle* de alavancas.

AF12 Foi desafiador o *puzzle* de salas de lava.

AF13 Foi desafiador o *puzzle* labirinto.

AF14 Foi desafiador o *puzzle* de armadilha de bolas de fogo inundando a sala.

AF15 Foi desafiador o *puzzle* de salas cobertas de gelo.

AF16 Foi desafiador o *puzzle* de apertar botões em sequência para liberar um item.

AF17 Foi desafiador o *puzzle* de espinhos que sobem e descem.

[13]https://www.institutoqualibest.com/blog/escala-likert-o-que-e-e-quando-utiliza-la/

AF18 Os inimigos do jogo proporcionaram um bom desafio para serem derrotados.

AF19 As salas do jogo eram repetitivas.

AF20 Foi fácil memorizar o que acontecia no jogo.

AF21 Durante o jogo, me senti perdido para onde ir.

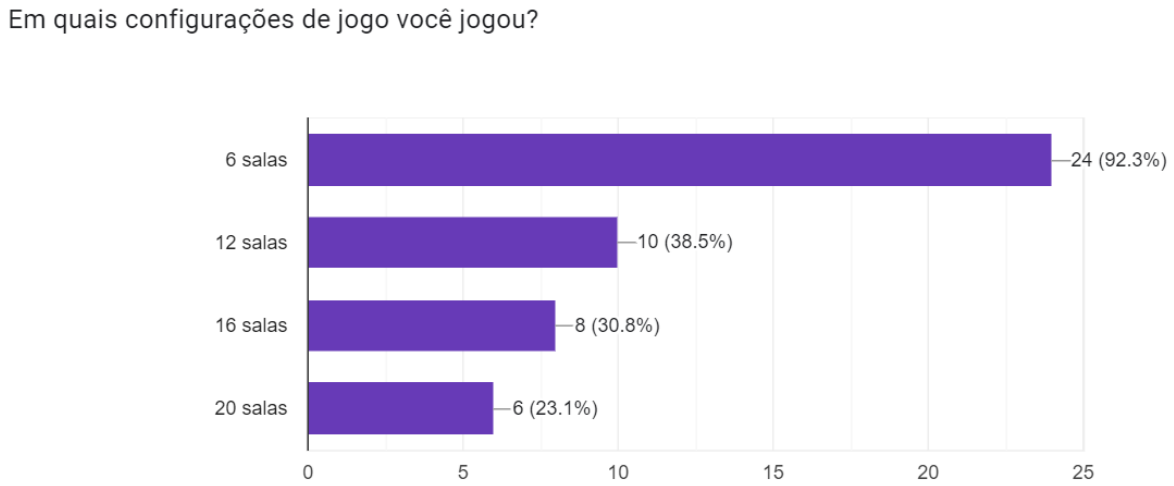
RL1 Deseja fazer alguma observação? Algo que chamou sua atenção ou acha que pode melhorar no jogo?

Essas afirmações almejam quantificar como cada jogador se sentiu em relação ao jogo, seus elementos e aos *puzzles*. A última questão (Item RL1) foi apresentada como uma caixa de texto, permitindo que os jogadores respondessem livremente sobre o jogo, comentando algo que lhes chamou a atenção ou sugerindo melhorias.

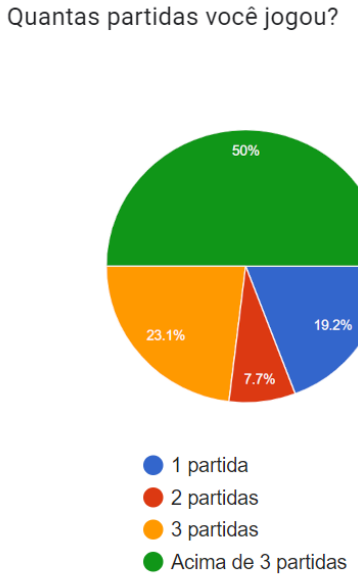
4.4.1 Análise dos Dados Coletados

O formulário de avaliação esteve disponível para responder no período de 09/11/2023 até 24/11/2023 e teve um total de 26 respostas de alunos matriculados nos cursos de Sistemas de Informação e Ciência da Computação da Universidade Federal de Juiz de Fora em períodos diversos.

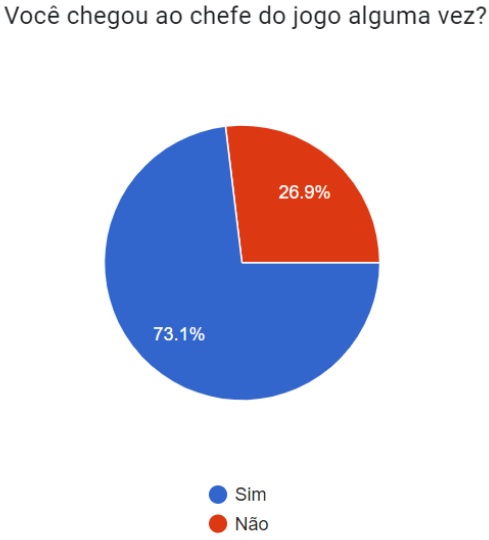
As respostas da primeira seção mostraram um retorno positivo por parte dos jogadores. Como observado na Figura 4.13a, a maioria das partidas aconteceu na configuração de 6 salas (92,3%). Entretanto, 38.5% dos jogadores se interessaram em jogar a configuração de 12 salas e 30,8% se interessaram em jogar a configuração de 16 salas. Menos participantes se interessaram pela configuração de 20 salas (23.1%), o que já era esperado, dado que esta configuração é mais complexa e demorada comparada com as demais. Para o número de partidas, a Figura 4.13b apresenta que 50% dos testadores jogaram mais de três, e 19,2% jogaram uma única partida, indicando a capacidade do jogo de prender a atenção de quem joga e instigar a jogá-lo múltiplas vezes. Além disso, 73,1% dos jogadores chegaram ao chefe do jogo pelo menos uma vez, mostrando que o jogo tem uma jogabilidade fácil de compreender e estimulante para o jogador seguir até o fim, como pode ser visto na Figura 4.13c.



(a) Gráfico com a quantidade de salas escolhida nas partidas jogadas.



(b) Gráfico referente a 26 respostas para o número de partidas jogadas.



(c) Gráfico com a resposta se chegaram no inimigo final (chefe) alguma vez.

Figura 4.13: Gráficos para as respostas da primeira seção do formulário. Fonte: do autor.

Já na segunda seção, as afirmações mais gerais sobre o jogo mostraram boa recepção. Para aquelas que indicavam aspectos positivos do jogo, nas afirmações AF1 até AF5 ninguém discordou totalmente, sendo que ambas AF1 e AF2 tiveram apenas 7,7% de discordância parcial e a AF3 3,3%, e as afirmações AF6 até AF8 tiveram baixo grau de discordância, com 15,3%, 11,5% e 15,3% dos participantes discordando parcialmente ou totalmente das afirmações, respectivamente. Nas afirmações que indicavam aspectos negativos do jogo, a AF19 teve 7,7% concordando totalmente e 11,5% concordando parcialmente, indicando que poucos jogadores sentiram que o jogo era repetitivo. Entretanto,

na AF20, 34,6% concordaram parcialmente e 26,9% concordaram totalmente, mostrando que há certa previsibilidade nos elementos do jogo pela facilidade de memorização. Cerca de 26,9% do jogadores, em menor ou maior grau, responderam na AF21 que ficaram perdidos pelo caminho, além de 30,8% que não concordam e nem discordam sobre ficarem perdidos no jogo.

Para os *puzzles*, a recepção foi mista. Os espinhos foram o *puzzle* com menor grau de discordância sobre ser desafiador, com apenas 4,3% discordando totalmente e nenhum participante discordando parcialmente. Os *puzzles* de interruptores, chave e porta, armadilha de bola de fogo, labirinto e sala de lava tiveram mais testadores concordando que são desafiadores do que os que não concordaram. No entanto, sala de gelo e alavanca tiveram percentuais de jogadores muito próximos que discordavam total ou parcialmente com os percentuais de jogadores que concordavam total ou parcialmente quanto ao desafio (38,9% discordando e 44,5% concordando para sala de gelo; para alavanca, os percentuais são 39,1% discordando e 43,4% concordando), o que pode indicar que os elementos adicionais por nível para as salas de gelo talvez não sejam suficientes para surpreender o jogador, além de que as alavancas possam estar causando frustração e não desafio aos jogadores. O que recebeu mais críticas foi o *sokoban*, com a maioria dos jogadores discordando totalmente sobre ser desafiador (28.6%). Por último, além da pouca discordância da AF18 (3,8% discordando totalmente e 7,7% parcialmente), também concordaram, em sua maioria, que os inimigos proporcionaram um bom desafio (34,6% concordaram totalmente e 26,9% parcialmente), mostrando que há diversidade nos inimigos e que os testadores precisaram pensar na melhor maneira de derrotá-los. A Figura 4.14 mostra o grau de concordância dos participantes a cada uma das afirmações.

Na pergunta de resposta livre, foi muito destacada a alta dificuldade do jogo. Os *puzzles* não foram muito comentados, mas comentou-se sobre o alto tempo para inicializar o jogo (causado pela geração do *sokoban*), e duas avaliações citaram que há um forte *backtracking* na configuração do jogo de 20 salas. Entretanto, um dos comentários elogiava essa característica, enquanto o outro a citou como um defeito do jogo.

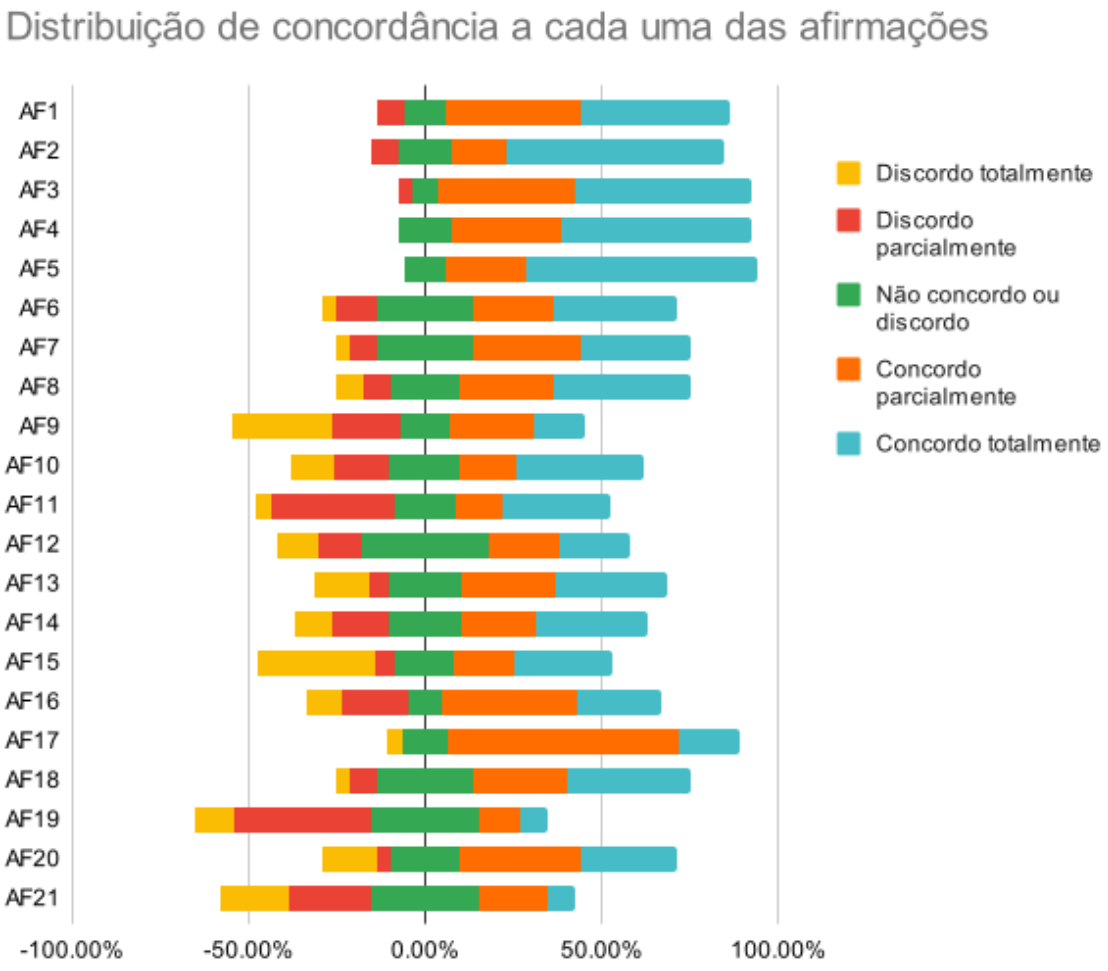


Figura 4.14: Distribuição da concordância para cada uma das afirmações do formulário. Fonte: do autor.

5 Considerações Finais

Este trabalho apresentou uma pesquisa exploratória amparada por desenvolvimento de software sobre como a Geração Procedural de Conteúdo (GPC) pode ser aplicada em *puzzles* e encontros em jogos *roguelike*. A GPC foi usada para o posicionamento global dos *puzzles* na topologia e também para geração de alguns *puzzles* que demandam técnicas mais específicas, como labirinto e *sokoban*. Além disso, através de um formulário de avaliação foi possível captar como é a influência dos elementos gerados na experiência do jogador.

Os objetivos específicos deste trabalho foram atingidos. Foram selecionados modelos de *puzzles*, que se encaixaram bem na topologia e tiveram o comportamento esperado para cada um. Por meio das restrições dos agentes geradores, não há a possibilidade de gerar elementos impossíveis ou inúteis dentro do jogo, e a abordagem de agentes dá ao jogo flexibilidade e manutenibilidade para incluir novos agentes de *puzzle* de forma prática, desde que as suas restrições sejam corretamente implementadas. A avaliação com jogadores confirmou que a experiência do jogador é positiva e que não há impasses inesperados causados pela GPC ou qualquer elemento do jogo.

Quando se busca na literatura sobre GPC de *puzzles*, o mais comum é encontrar técnicas para gerar proceduralmente tipos específicos de *puzzles*, mas pouco sobre usar *puzzles* criados por GPC dentro de outros jogos. Ao agrupar vários modelos de *puzzles* diferentes no mesmo jogo (que não se trata de um jogo do gênero *puzzle*, mas de um jogo *roguelike*), confirmamos que também foi alcançado o objetivo geral deste trabalho.

5.1 Limitações e Trabalhos Futuros

Algumas limitações tiveram que ser realizadas em cada um dos objetivos para que o estudo fosse possível. A topologia é gerada sobre um grafo de árvore geradora mínima, portanto todas as restrições dos agentes são voltadas a atender mapas com essa exata configuração - mapas com caminhos cíclicos, por exemplo, exigiriam restrições completamente diferentes.

Um caso em que isso causa um problema que é capaz de criar inviabilidade ao jogo é o que acontece no trabalho de Weeks e Davis (2022), pois como é permitido gerar caminhos cíclicos, várias vezes termina por gerar elementos inúteis ao jogo, tal qual portas que são ignoradas pelo jogador devido à existência de mais de um caminho que leva à mesma sala.

Sobre a forma das salas, estas possuem exclusivamente um formato retangular que facilita a inserção dos *puzzles* e suas restrições. Porém, também dá às salas inflexibilidade e um aspecto menos orgânico. Para salas de formato irregular, os *puzzles* e suas restrições teriam que ser retrabalhados.

Para os *puzzles*, os dois que exigiram algoritmos específicos também apresentaram limitações. O labirinto não é exatamente próprio para salas pequenas - por isso, a escolha de *design* de limitar a visão do jogador para a sala onde está o labirinto. Por outro lado, se houver um aumento da extensão da sala além do que é permitido no jogo, o labirinto pode escalar a ponto de se tornar confuso caso se mantenha a sala escura. No *sokoban*, o próprio trabalho de referência possui limitações. Os valores de algumas variáveis usadas no cálculo da nota dos tabuleiros não foram informados pelos autores, forçando a obtenção destes valores de maneira empírica ao adaptar o método de geração. Além disso, é dito que a MCTS do trabalho de referência não repete tabuleiros durante a fase de simulação, mas também não é citada qual a abordagem usada para evitar essa repetição, e isso poderia contribuir significativamente para gerar tabuleiros com notas mais altas. Outro problema é que, apesar da capacidade de gerar tabuleiros com certa complexidade, muitas vezes os tabuleiros gerados consistiam em duas caixas que eram empurradas duas ou três vezes para locais muito óbvios para o jogador, o que tornava a geração dos tabuleiros não somente infrutífera do ponto de vista da experiência do jogador, devido à trivialidade para resolvê-lo, como custosa para o jogo, pois a GPC do *sokoban* é o processo mais demorado na geração do jogo inteiro.

Outra limitação neste trabalho é que os *puzzles* não possuem relação ou contribuem com a narrativa do jogo. Não há uma preocupação em contar uma história através da ordem das salas. O jogo é um arquétipo de *roguelike* com elementos de RPG digital no qual o jogador passa por portas, alavancas e adquire itens. Com um maior foco em narrativa, aumenta-se o engajamento do jogador e a possibilidade do jogador entrar em

estado de *flow*. O uso de uma base de dados alimentada externamente, como é usada para *puzzles* narrativos no trabalho de Kegel e Haahr (2019b), aumentaria bastante a variabilidade ao jogo e auxiliaria a criar a atmosfera para uma história. Outra abordagem, mais ousada, consiste no uso de ferramentas GPT[14] a fim de auxiliar na geração deste tipo de *puzzle*, seja para criar missões com os elementos disponíveis na base externa, quanto para gerar diálogos para os personagens do jogo.

Outra limitação foi por não ter sido possível integrá-lo ao PCGLab. O ambiente possui diversas técnicas já implementadas que auxiliariam a compor melhor os *puzzles*, seja no posicionamento dos elementos ou dos inimigos. Um exemplo é o uso de pontos de interesse, posicionando itens em um local na sala e os inimigos de forma a protegê-los, em vez do posicionamento dos elementos no centro das salas e aleatório para os inimigos, que acontece atualmente. Além disso, o ambiente também possui métricas de avaliação implementadas, o que tornaria a avaliação dos *puzzles* mais consistente. Como trabalho futuro, essa integração ao PCGLab pode ser desenvolvida, pois o projeto motivou a elaboração deste e, ao dar continuidade com o que já foi desenvolvido, acredita-se que será possível criar um ambiente de estudo para GPC de grande utilidade para a academia e mercado. Portanto, o próximo passo, ao fim de todo o desenvolvimento deste projeto, seria sua adaptação completa ao ambiente.

Para a topologia, a geração em cima de outros modelos de mapas, com salas de tamanho e formato irregular, com ou sem corredores, também pode ser avaliada como uma nova proposta de implementação. Mapas com formatos diferentes geram experiências mais desafiadoras e complexas para o jogador, pois o formato com salas retangulares e corredores estreitos possui memorização e adaptação da habilidade do jogador notoriamente mais fáceis.

[14]https://aws.amazon.com/pt/what-is/gpt/

Bibliografia

AJITH, A.; BABU, S. R.; K, S.; K, S. A.; THOMAS, M. V. A novel method in procedural maze generation. In: *2022 IEEE Bombay Section Signature Conference (IBSSC)*. [S.l.: s.n.], 2022. p. 1–5.

BRATHWAITE, B.; SCHREIBER, I. *Challenges for Game Designers*. 1. ed. USA: Charles River Media, Inc., 2008. ISBN 158450580X.

Centro Universitário Tiradentes. *Mercado de trabalho para jogos digitais: Oportunidades e desafios*. 2023. Disponível em: ⟨https://www.unit.br/blog/mercado-de-trabalho-para-jogos-digitais-oportunidades-e-desafios⟩. Acesso em: 04 de maio de 2023.

COLTON, S. Automated puzzle generation. In: . [S.l.: s.n.], 2002.

COSTA, C. H. F. da. *Você Conhece o Estado de Flow?* 2021. Disponível em: ⟨https://spsicologos.com/2021/05/17/voce-conhece-o-estado-de-flow/⟩. Acesso em: 28 de abril de 2023.

COSTA, L. D. D. Geração procedural de conteúdo através de uma abordagem híbrida entre autômatos celulares e heurísticas. In: . [s.n.], 2020. Disponível em: ⟨https://www.sbgames.org/proceedings2020/WorkshopG2/209761.pdf⟩. Acesso em: 28 de abril de 2023.

DETERDING, S.; KHANDAKER, M.; RISI, S.; FONT, J. M.; DAHLSKOG, S.; SALGE, C. A. de L.; OLSSON, C. Graph-based generation of action-adventure dungeon levels using answer set programming. In: . [S.l.: s.n.], 2018.

DIMA, C. V. R. *Modelagem da Progressão de Dificuldade em Jogos para Mapas Gerados Proceduralmente*. Monografia (Graduação em Sistemas de Informação) — Universidade Federal de Juiz de Fora, 2023. Disponível em: ⟨http://monografias.ice.ufjf.br/tcc-web/tcc?id=713⟩.

DORMANS, J. *Engineering emergence: applied theory for game design*. [S.l.]: University of Amsterdam, 2013.

DORMANS, J. Cyclic generation. In: *Procedural Generation in Game Design*. [S.l.]: AK Peters/CRC Press, 2017. p. 83–96.

Escola de Comunicação e Design Digital - Instituto Infnet. *O Mercado de Design de Games: COMO É O TRABALHO COM JOGOS NO BRASIL*. 2023. Disponível em: ⟨https://blog.ecdd.com.br/guia-mercado-de-jogos-brasil/⟩. Acesso em: 04 de maio de 2023.

FEOFILOFF, P. *Algoritmo de Kruskal*. 2019. Disponível em: ⟨https://www.ime.usp.br/~pf/algoritmos_para_grafos/aulas/kruskal.html⟩. Acesso em: 27 de outubro de 2023.

FEOFILOFF, P. *Algoritmo de Prim*. 2019. Disponível em: ⟨https://www.ime.usp.br/~pf/algoritmos_para_grafos/aulas/prim.html⟩. Acesso em: 27 de outubro de 2023.

GARCIA, C. *Hexgen Roguevania*. Dissertação (Mestrado) — Lindenwood University, 2022.

GELLEL, A.; SWEETSER, P. A hybrid approach to procedural generation of roguelike video game levels. In: . [S.l.: s.n.], 2020. p. 1–10.

GREEN, A. *Goof Troop Review (Super Nintendo)*. Nintendo Life, 2013. Disponível em: ⟨https://www.nintendolife.com/reviews/snes/goof_troop⟩. Acesso em: 28 de abril de 2023.

GREEN, M.; KHALIFA, A.; ALSOUGHAYER, A.; SURANA, D.; LIAPIS, A.; TOGELIUS, J. Two-step constructive approaches for dungeon generation. In: . [S.l.: s.n.], 2019. p. 1–7. ISBN 978-1-4503-7217-6.

KARTAL, B.; SOHRE, N.; GUY, S. Data driven sokoban puzzle generation with monte carlo tree search. *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, v. 12, n. 1, p. 58–64, 2016.

KEGEL, B.; HAAHR, M. Procedural puzzle generation: A survey. *IEEE Transactions on Games*, PP, p. 1–1, 05 2019.

KEGEL, B.; HAAHR, M. Towards procedural generation of narrative puzzles for adventure games. In: _____. [S.l.: s.n.], 2019. p. 241–249. ISBN 978-3-030-33893-0.

KIM, P. H.; CRAWFIS, R. Intelligent maze generation based on topological constraints. In: *2018 7th International Congress on Advanced Applied Informatics (IIAI-AAI)*. [S.l.: s.n.], 2018. p. 867–872.

KUITTINEN, P. *Rogue - Exploring the Dungeons of Doom (1980)*. 2001. Disponível em: ⟨https://web.archive.org/web/20071217100401/http://users.tkk.fi/~eye/roguelike/rogue.html⟩. Acesso em: 30 de março de 2023.

MELOTTI, A. S.; MORAES, C. H. V. de. Evolving roguelike dungeons with deluged novelty search local competition. *IEEE Transactions on Games*, v. 11, n. 2, p. 173–182, 2019.

MOSS, R. C. *ASCII art + permadeath: The history of roguelike games*. 2020. Disponível em: ⟨https://arstechnica.com/gaming/2020/03/ascii-art-permadeath-the-history-of-roguelike-games/⟩. Acesso em: 28 de abril de 2023.

PEREIRA, F. *Games: Mercado segue crescendo no Brasil*. 2023. Disponível em: ⟨https://whow.com.br/games-pesquisa/⟩. Acesso em: 28 de abril de 2023.

PEREIRA, L. T.; VIANA, B. M. F.; TOLEDO, C. F. M. A system for orchestrating multiple procedurally generated content for different player profiles. *IEEE Transactions on Games*, p. 1–11, 2022.

PRATA, D. *Precisamos falar sobre os jogos roguelike/roguelite*. 2022. Disponível em: ⟨https://meiobit.com/458303/precisamos-falar-sobre-os-jogos-roguelike-roguelite/⟩. Acesso em: 27 de outubro de 2023.

ROZEN, R.; DORMANS, J.; SAMARITAKI, G. Debugging procedural level designs with mental maps. In: . [S.l.: s.n.], 2022. p. 1–3.

SANTANA, G. M. *Avaliação do Fluxo da Experiência do Jogador na Geração Procedural por Autômatos Celulares*. Monografia (Graduação em Sistemas de Informação) — Universidade Federal de Juiz de Fora, 2021. Disponível em: ⟨http://monografias.ice.ufjf.br/tcc-web/tcc?id=559⟩.

SCHELL, J. *The Art of Game Design: A book of lenses*. 1. ed. [S.l.]: CRC Press, 2008. ISBN 9780429100444.

SHAKER, N.; TOGELIUS, J.; NELSON, M. J. *Procedural Content Generation in Games: A Textbook and an Overview of Current Research*. [S.l.]: Springer, 2016.

SHORT, T. X.; ADAMS, T. *Procedural storytelling in game design*. [S.l.]: Crc Press, 2019.

SIMON, L. I. *What can we expect from PC and console gaming in 2023?* 2023. Disponível em: ⟨https://newzoo.com/resources/blog/pc-and-console-gaming-industry-in-2023⟩. Acesso em: 28 de abril de 2023.

TAYLOR, J.; PARBERRY, I. Procedural generation of sokoban levels. In: . [s.n.], 2011. Disponível em: ⟨https://api.semanticscholar.org/CorpusID:17543776⟩.

TOGELIUS, J.; KASTBJERG, E.; SCHEDL, D.; YANNAKAKIS, G. N. What is procedural content generation?: Mario on the borderline. In: ACM. *Proceedings of the 2nd international workshop on procedural content generation in games*. [S.l.], 2011. p. 5.

VENCO, F.; LANZI, P. L. An agent-based approach for procedural puzzle generation in graph-based maps. In: *2021 IEEE Conference on Games (CoG)*. [S.l.: s.n.], 2021. p. 01–08.

VERDE, R. C. V. *Avaliação da geração de conteúdo por Wave Function Collapse na experiência do jogador*. Monografia (Graduação em Ciência da Computação) — Universidade Federal de Juiz de Fora, 2021. Disponível em: ⟨http://monografias.ice.ufjf.br/tcc-web/tcc?id=572⟩.

WALKER, B. *Procedural level design in Brogue and beyond*. 2018. Disponível em: ⟨https://www.youtube.com/watch?v=Uo9-IcHhq_w⟩. Acesso em: 06 de julho de 2023.

WEEKS, M.; DAVIS, J. Procedural dungeon generation for a 2d top-down game. In: . [S.l.]: Association for Computing Machinery, 2022. p. 60–66. ISBN 9781450386975.

WIKIPEDIA. *Fisher–Yates shuffle*. 2007. Disponível em: ⟨https://en.wikipedia.org/wiki/Fisher%E2%80%93Yates_shuffle⟩. Acesso em: 13 de novembro de 2023.

WINKIE, L. *People who argue about the definition of roguelikes are annoying, but what if they're right?* 2021. Disponível em: ⟨https://www.pcgamer.com/people-who-argue-about-the-definition-of-roguelikes-are-annoying-but-what-if-theyre-right/⟩. Acesso em: 28 de abril de 2023.

ZAKARIA, Y.; FAYEK, M.; HADHOUD, M. Procedural level generation for sokoban via deep learning: An experimental study. *IEEE Transactions on Games*, v. 15, n. 1, p. 108–120, 2023.

ZAPATA, S. *On the Historical Origin of the "Roguelike" Term*. 2017. Disponível em: ⟨https://blog.slashie.net/on-the-historical-origin-of-the-roguelike-term/⟩. Acesso em: 28 de abril de 2023.