UNIVERSIDADE DE LISBOA
FACULDADE DE CIÊNCIAS
DEPARTAMENTO DE INFORMÁTICA

Algoritmo de Grafos Cíclico para Geração de Dungeons

Gonçalo Pessoa de Amorim Megre do Amaral

Mestrado em Engenharia Informática

Dissertação orientada por:
Prof. Doutor Luis Manuel Ferreira Fernandes Moniz

2023

Agradecimentos

Gostaria de agradecer, em primeiro lugar, aos meus pais e à minha irmã por sempre me apoiarem no meu percurso académico, por toda a motivação que sempre me transmitiram, pelo seu apoio incondicional e por todos os conselhos que me foram dando ao longo do meu percurso académico.

Um agradecimento especial ao Prof. Doutor Luís Moniz por me orientar e me aconselhar neste trabalho final de mestrado que é a dissertação, pela sua disponibilidade, pelo saber e apoio que me transmitiu ao longo deste ano letivo.

E por último, agradeço aos meus amigos e colegas de curso com quem colaborei e convivi, pelo trabalho que efetuámos durante este percurso académico e pelos bons momentos passados.

Resumo

Esta dissertação debruça-se sobre a implementação de algoritmos de Geração Procedimental de Conteúdo (GPC) no desenvolvimento de videojogos.

A indústria dos videojogos continua em pleno crescimento sendo lançados todos os anos novos produtos cada vez mais elaborados e sofisticados, quer do ponto de vista gráfico quer das mecânicas do jogo e, sobretudo, produtos cada vez mais desafiantes quer do ponto de vista de quem os desenvolve quer do ponto de vista de quem os joga.

A GPC torna-se particularmente interessante neste contexto, dado que é um método pelo qual é possível criar novos conteúdos de várias naturezas (geração de música, de arte, de gráficos, de cenários, de itens, de missões, etc), de forma assistida ou totalmente autónoma. Esta metodologia é extremamente útil no desenvolvimento de videojogos, facilitando e melhorando significativamente o processo de produção dos mesmos, pela inovação que permite ao nível dos conteúdos, criando novas ou diferentes experiências para os jogadores; por facilitar o contorno de eventuais restrições de hardware, encontrando diferentes e inovadoras formas de ultrapassar esses desafios; por aumentar a produtividade dos trabalhadores das empresas e estúdios de jogos sem incrementar (ou muito pouco) os custos de produção ao permitir que se desenvolva o mesmo conteúdo mais rapidamente ou até mais conteúdo pelo mesmo custo.

Uma outra vantagem desta tecnologia é a de permitir aumentar a rejogabilidade dos jogos, como por exemplo, gerando novas e diferentes masmorras, fazendo com que o jogador possa retirar maior proveito do seu jogo, usufruindo do mesmo durante mais tempo.

Nesta dissertação, explora-se a implementação de um algoritmo de grafos cíclico para a geração de masmorras e efetua-se a sua comparação com a aplicação de outros algoritmos conhecidos e destinados ao mesmo fim, tais como o Walker, o Branching Trees e um baseado no motor de física do game engine Godot.

Para tal, desenvolvi um jogo roguelike para avaliar se o algoritmo de grafos cíclico é viável para a criação de masmorras (e consequentemente de níveis em jogos) e comparo a implementação deste algoritmo às dos outros mencionados anteriormente, destacando as suas vantagens, desvantagens e diferenças.

Palavras-chave: Ciclos, Geração, Roguelike, Masmorras

Abstract

This thesis is about the implementations of Procedural Content Generation (PCG) in games development.

The video game industry continues to grow strong, with new products being launched every year that are ever more elaborate and sophisticated, both in terms of graphics and game mechanics, and above all, increasingly challenging products to either play or develop.

PCG becomes particularly interesting in this context, as it is a method by which it is possible to create new content of various natures (generation of music, art, graphics, scenarios, items, missions, etc.) in an assisted or fully autonomous way. This methodology is extremely useful in the development of video games, facilitating and significantly improving their production process, by allowing the game developers to innovate the way content is generated, by creating new or different experiences for players; for facilitating the bypassing of possible hardware restrictions, finding different and innovative ways to overcome these challenges or for increasing the productivity of workers in companies and game studios without increasing (or very little) production costs.

Another advantage of this technology is the increase of replayability in games, for example, generating new dungeons so that the player can get more out of his game, staying entertained for a longer period of time.

In this dissertation, I explore the implementation of a cyclic graph algorithm for the generation of dungeons, and compare this algorithm to other known algorithms, such as the Walker, Branching Trees and one based on the game engine's physics engine. For this, I will develop a roguelike game to experiment and analyze if the cyclic graph algorithm is viable for the creation of dungeons (and consequently levels in games) and I will compare this algorithm to the others mentioned above, regarding their advantages, disadvantages and differences regarding their implementations.

Keywords: Cyclic, Generation, Roguelike, Dungeons

Conteúdo

Lista de Figuras

Lista de Tabelas

Glossário

Jogo Roguelike - um jogo que utiliza mecânicas de jogo, iguais ou semelhantes, às implementadas no jogo "Rogue", lançado em 1980[8]. Estas mecânicas incluem a morte permanente, o combate por turnos, o movimento na grelha, a geração procedimental de níveis. Não é necessário implementar todas as mecânicas para o jogo ser considerado roguelike.

Sprites - Sprite é uma imagem de duas dimensões que geralmente se encontra integrada numa cena ou cenário maior. Este termo era originalmente usado para referir objetos de tamanho fixo que se encontravam num plano de fundo.

Grafos - Os grafos mencionados nesta dissertação advêm da teoria dos grafos, uma matéria relativa à área da matemática. Os grafos são estruturas matemáticas que possuem vértices (também chamados de nós ou pontos) e que estão ligados entre si por arestas. Estas estruturas matemáticas são bastante usadas para modelar as relações entre os objetos, sendo por isso uma possível alternativa na geração procedimental de conteúdos.

Masmorra - Masmorras, ou em inglês "Dungeons", são um tipo de prisão da época medieval tipicamente localizada em níveis inferiores de castelos. Relativamente aos videojogos, estas masmorras costumam ser desafios onde os níveis são tipicamente retratados com divisões ou salas quadradas ou retangulares, podendo possuir, dentro dessas divisões o que os produtores do jogo quiserem colocar. Como exemplo, podem ser introduzidas nessa divisões, inimigos, puzzles, itens, NPCs.

Tile - Num contexto de desenvolvimento de jogos, um tile é um espaço do mapa, sendo tipicamente um quadrado ou um hexágono.

Script - Em programação, um script é uma sequência de instruções, ou um programa, que é corrido por outro programa e não pelo sistema operativo.

Heurística - Heurística é uma abordagem de resolução de problemas ou de auto descoberta, onde é empregado um método prático que permite chegar a uma solução num curto espaço de tempo. Não garante, no entanto, que a solução seja perfeita, ótima ou até mesmo racional.

NPC - Non Player Character. Os NPCs são as entidades ou personagens criadas no jogo que não são jogadas pelo jogador, tais como personagens secundárias ou inimigos.

Bot - É um programa autónomo com as suas funcionalidades definidas que pode interagir com outros sistemas e/ou utilizadores. Este programa pode ser online ou offline.

Backtracking - Este termo quando usado nos jogos, refere-se ao caminho que o jogador já percorreu e terá de voltar a percorrer inversamente para cumprir o objetivo. Existem jogos onde o jogador poderá percorrer um caminho sem utilidade, sendo obrigado a voltar para trás e ir por outro caminho.

Capítulo 1

Introdução

1.1 Motivação

A indústria de videojogos, a nível mundial, tem vindo a crescer significativamente, tendo registado um aumento nos anos da pandemia de covid-19, em consequência do confinamento que forçou grande parte da população mundial a manter-se em casa. A pandemia desencadeou um crescimento excecional deste setor, o qual se explica sobretudo pela adesão de milhões de novos jogadores de todas as idades e grupos demográficos, sendo estimado que o número total de jogadores em todo o mundo seja 2.7 mil milhões[9]. Neste período, os videojogos foram utilizados não só como forma de entretenimento, distração e/ou manutenção de contactos sociais, mas também como forma de aprendizagem. Para dar um exemplo, alguns professores utilizaram-nos para motivar os seus alunos a interessarem-se pelas matérias escolares de uma forma lúdica, cativando a sua atenção em aulas online, servindo-se de jogos, como por exemplo, o "Assassin's Creed Origins"que inclui um "Discovery Mode"onde os jogadores podem explorar o Egito no reinado de Cleópatra[10].

De acordo com dados da PwC, as receitas mundiais do setor dos videojogos poderão vir a atingir 321 mil milhões de dólares em 2026, isto é, mais de 50% que o valor apurado em 2021 de 214 mil milhões de dólares[11].

Ao longo do tempo, os jogos têm-se tornado cada vez mais elaborados e complexos. O primeiro videojogo interativo, foi registado em 1947 com o nome "Cathode-ray tube amusement device"criado por Thomas T. Goldsmith Jr. e Estle Ray Man[12]. Este jogo permitia ao utilizador controlar o arco parabólico de um ponto no ecrã do oscilador para simular um míssil sendo disparado contra alvos, que são desenhos no papel fixos no mesmo ecrã. Desde então, cada vez mais jogos têm sido desenvolvidos e lançados no mercado, jogos que crescem em dimensão e em complexidade de produção, em função das suas mecânicas de jogo e do realismo pretendido. Naturalmente, decorrente do crescente custo e complexidade de desenvolvimento, as empresas gastam milhões de dólares/euros a desenvolverem os jogos. Exemplo disso é o jogo "Cyberpunk 2077", que custou 174

milhões de dólares em custos de desenvolvimento, não só pelo tempo despendido mas também pelo número de trabalhadores que foram necessários nas várias áreas de desenvolvimento, tais como, em programação, em arte, em história, em sound design, em level design[13]. Em consequência, as empresas têm vindo a procurar formas de conseguir aumentar a sua produtividade sem fazer crescer significativamente os custos de produção, isto é, têm vindo a investir e a investigar processos que otimizem a produção de jogos e conteúdos.

Na década de 80, quando o mercado de videojogos estava a ganhar muita relevância, surgiu um novo tipo de software que se tornou muito utilizado e que respondia, em parte, a esta necessidade: os game engines[14]. Estes programas ajudam bastante a simplificar o desenvolvimento de jogos, tratando de todas as tarefas comuns e necessárias em todos jogos, evitando a construção de cada novo jogo a partir do zero como ocorria até então; os game engines facilitam o desenho dos gráficos no ecrã, a gestão dos inputs dos comandos, a gestão do áudio e a atualização da lógica do que ocorre no jogo. Estes motores de jogo são muito complexos e dispendiosos de se desenvolver, sendo o mesmo motor usado frequentemente em vários jogos para diluir o seu custo. Existem vários game engines tais como o Unity ou o Unreal Engine; alguns game engines aplicam-se tanto em jogos 2D como em 3D, enquanto que outros se focam apenas em jogos 2D ou em jogos 3D.

Outra ferramenta que contribui para o desenvolvimento de jogos, é a Geração Procedimental de Conteúdo (GPC) que é um método que permite produzir mais conteúdo em menos tempo, com equipas mais pequenas e com menos investimento, ultrapassando inclusivamente certas limitações de hardware, nomeadamente a capacidade de armazenamento.

A GPC é uma técnica de programação que utiliza algoritmos para criar conteúdos virtuais, de forma automática, aleatória e dinâmica, em áreas tão diferentes como: jogos, matemática, animação, gráficos, modelação 3D, edições de vídeo, música. No caso específico dos jogos, pode ser utilizada na geração de arte, itens, missões e, o mais relevante para este projeto, nos cenários. Como exemplos de jogos conhecidos que utilizam o GPC destaco o "Pokémon Mystery Dungeon"e o "Dwarf Fortress". O "Pokémon Mystery Dungeon"é um jogo do tipo roguelike lançado para a consola Game Boy Advance, com hardware limitado, onde cada masmorra é completamente gerada de raiz, desde as divisões da masmorra, aos itens e aos inimigos, tudo de forma autónoma[15]. O "Dwarf Fortress"é também um jogo roguelike, com características de gestão e simulação lançado para PCs. Este jogo utiliza a geração procedimental para gerar o mapa e tudo que está dentro dele, como a ascensão e queda de civilizações, as personalidades das personagens e entidades, as suas religiões, as culturas, os animais, cada parte do seu corpo, etc [16].

Só foi possível desenvolver estes dois jogos, graças à utilização da geração procedimental de conteúdo.

Jogos roguelike são jogos que utilizam as mesmas mecânicas do jogo "Rogue"que foi

lançado na década de 80[8]. As mecânicas utilizadas neste jogo são:

- A morte permanente, isto é, a personagem perde todo o seu progresso no jogo, tendo de recomeçar quando perde toda a sua vida.
- O combate por turnos. O jogador executa uma ação como mover ou atacar e, ao encerrar o turno, o jogo atualiza os movimentos.
- O movimento de personagens e entidades do jogo são feitos em grelha.
- Geração procedimental de masmorras. É utilizado um ou mais algoritmos para gerar as masmorras, onde cada cenário gerado será diferente.
- Todas as ações de movimento e habilidades da personagem têm de estar disponíveis ao jogador desde o início do jogo.
- O jogador pode atingir os seus objetivos de formas diferentes.
- Para sobreviver, o jogador terá de gerir os seus pertences que vai acumulando ao progredir no jogo.
- Mecânica de combate baseada no ”Hack-and-Slash”, isto é, combate ”corpo a corpo”.
- No jogo, existir um mundo para explorar e encontrar itens que ajude o jogador a progredir no objetivo.

De notar que não é necessário implementar todas estas mecânicas para o jogo ser considerado “roguelike”. Alguns exemplos de jogos deste tipo são: “Cult of the Lamb”, “Into the Breach”, a série “Pokémon Mystery Dungeon”, “Hades”, “Enter the Gungeons”, “Rogue Legacy”.

Existem também programas como o “Speed Tree”e “Gaia”que auxiliam o level designer na geração de vegetação e terrenos, aumentando a sua produtividade.

Com a automatização da produção de conteúdos, a GPC tem assim a vantagem de reduzir a intervenção humana no desenvolvimento do jogo, permitindo que os especialistas em desenvolvimento de software se debrucem sobre aspetos menos repetitivos do jogo e se foquem noutras tarefas, reduzindo também o tempo da produção e os respetivos custos, adicionando valor ao jogo.

Existem vários tipos de algoritmos diferentes para gerar conteúdo de modo procedimental. Nesta dissertação é explorada uma técnica recente, o algoritmo de grafos cíclico, que foi implementado no jogo “Unexplored”. Este algoritmo visa resolver o problema da existência de becos sem saída, que prejudicam a experiência do jogador, forçando-o a voltar para trás com frequência e/ou longos percursos. Este é um problema comum noutros algoritmos. Procura-se entender concretamente as capacidades e limitações que este algoritmo tem, e compará-lo a outros algoritmos mais utilizados para gerar cenários, tanto do ponto de vista da implementação, como dos resultados.

1.2 Objetivos

O desenvolvimento de videojogos requer ferramentas que auxiliam a produção de conteúdos, sendo a Geração Procedimental de Conteúdos um método eficaz. Num jogo 2D roguelike, onde um jogador pode explorar vários níveis de uma masmorra, a criação destes níveis pode ser gerada automaticamente através de vários tipos diferentes de algoritmos de GPC. Estes geram, a partir de um ponto de partida arbitrário, túneis, corredores, salas que vão sendo adicionados automaticamente ao nível do jogo. Contudo, cada algoritmo possui certas limitações ou desvantagens. Exemplo disso é, o Branching Trees onde a geração da estrutura da masmorra origina becos sem saída, obrigando o jogador a voltar atrás frequentemente no nível para concluir os objetivos do mesmo; estas ocorrências tornam-se desmotivadoras para o jogador, podendo fazê-lo desistir do jogo.

O objetivo desta dissertação é a implementação de um algoritmo de grafos cíclico, que permita gerar masmorras, comparar o processo de geração deste algoritmo com outros algoritmos tais como o Walker, o Branching Trees e um algoritmo baseado no motor de física do Godot e analisar a complexidade de implementação de cada algoritmo. Para isso, criei um jogo roguelike com mecânicas simples e algumas regras de geração, para se perceber as vantagens e desvantagens do algoritmo, bem como se a sua utilização é viável para o desenvolvimento de um projeto desta natureza.

1.3 Contribuições

Para esta dissertação foram implementados os seguintes algoritmos de GPC para geração de cenários: o algoritmo grafos cíclico, o Walker, o Branching Trees e um baseado no motor de física do Godot num jogo 2D roguelike.

Os algoritmos desenvolvidos para a geração de cenários, foram criados com o intuito de possuírem os seguintes atributos:

- Divisões ou salas distribuídas pelo mapa e que não se sobreponham;
- Corredores que liguem as divisões, e no caso de se intercetarem, não inviabilizem ou comprometam a evolução do jogo;
- Inexistência de becos sem saída, ou caso ocorram, não obriguem o jogador a recuar muito no nível;
- Colocar as características das divisões no cenário de jogo com o intuito de criarem desafios;
- Geração das várias entidades como o próprio jogador e os inimigos, etc.

Todos os algoritmos funcionaram, tendo gerado mapas que foram analisados em função da concretização dos vários atributos que garantem a jogabilidade de cada cenário.

Assim, foram considerados os seguintes aspetos: a estrutura e a dimensão do mapa; a localização das salas (se são suficientes e lógicas, se não se sobrepõem); a localização dos corredores (inexistência dos caminhos cruzados); a localização dos corredores (avaliar se são em número adequado e bem distribuídas); os becos sem saída (inexistentes ou, caso ocorram, que não penalizem a experiência do jogo obrigando o jogador a recuar muito no nível).

As várias implementações foram analisadas e comparadas entre si, com o objetivo de se apurar se o algoritmo de grafos cíclico seria ou não viável para o desenvolvimento de cenários nos videojogos; sendo que foram também identificadas as vantagens e desvantagens das várias técnicas e em que situações cada algoritmo é mais adequado dependendo das características do projeto a ser desenvolvido.

Dado que o objetivo final de um jogo é o de ser interessante para o jogador, o desempenho dos 4 algoritmos em estudo foi medido também pela avaliação da jogabilidade dos cenários gerados por cada um, motivo pelo qual foi efetuada uma apreciação, sob forma de um inquérito, a qual está refletida na secção 4.5.

Tanto o código como um vídeo a demonstrar os algoritmos a gerar as masmorras podem ser encontrados no github[17]

1.4 Estrutura do documento

Este documento está organizado da seguinte forma:

- Capítulo 2 – Neste capítulo é apresentado o estado da arte de vários algoritmos para geração de cenários, o trabalho relacionado com a geração de grafos cíclicos, por último, as ferramentas consideradas e usadas para a criação do jogo.

- Capítulo 3 – Descrição do jogo e das suas funcionalidades, descrição dos algoritmos e respetivas vantagens e desvantagens, identificadas durante o desenvolvimento do jogo e as dificuldades de desenvolvimentos do jogo.

- Capitulo 4 - Descrição e análise dos resultados da geração de masmorras pelos algoritmos desenvolvidos e resultados do inquérito realizado.

- Capitulo 5 - Conclusão do trabalho efetuado e dos resultados obtidos, incluindo referência ao possível trabalho futuro.

Capítulo 2

Metodologia

2.1 Estado da arte

Geração Procedimental de Conteúdo (GPC) tem sido um tema explorado ao longo das últimas décadas devido à grande potencialidade desta metodologia na automatização e desenvolvimento de conteúdos. Existe investigação em curso em diversas áreas tais como na geração de música ou de terrenos 2D e 3D, podendo estas técnicas ser usadas online e offline. Existem também jogos que utilizam geração procedimental de conteúdo de uma forma inovadora, nomeadamente em cenários. É o caso de "No Man's Sky", um jogo de exploração espacial onde existem 15 quintiliões de planetas a explorar. Esta é uma escala de criação de cenários de uma dimensão sem precedentes e só possível com GPC; quase tudo no jogo é gerado procedimentalmente, isto é, todas as galáxias, planetas, a fauna e a flora, as estações espaciais, as missões e as estrelas.

2.1.1 Geração de Cenários

A seleção do tipo de terreno a utilizar num jogo é um fator com grande impacto na sua conceção. O terreno ou o cenário pode ser em 2D ou em 3D e este fator condiciona a abordagem e a complexidade da técnica a utilizar. Caso seja pretendido gerar um cenário 3D que pareça realista, com alterações da altura do terreno, com árvores, diferentes tipos de vegetação (como por exemplo no "Minecraft"), este terá um maior nível de complexidade no desenvolvimento que um terreno com apenas duas dimensões.

As técnicas mencionadas em seguida podem todas elas ser utilizadas tanto em 2D como em 3D.

Autómato Celular

Autómato Celular é um modelo de computação discreto que tem aplicação nas mais diversas áreas como física, biologia, modelação de micro-estruturas e em jogos[18]. Um autómato celular consiste numa grelha regular, onde cada célula tem um número de estados finitos, como "on"ou "off", ou alternativamente 1 e 0. Após a definição das regras

relativas ao comportamento das células ocupadas e das adjacentes, é só adicionar ou colocar as células, manualmente ou automaticamente na grelha, e depois é só iterar cada célula com as respetivas regras.

Na figura 2.1 está exemplificado o funcionamento do autómato celular com as seguintes regras:

- Se a célula tiver o estado 0 e pelo menos 1 vizinho com estado 1, então muda-se o estado da célula para 1.
- Se a célula tiver o estado 1 e pelo menos dois dos seus vizinhos também tiverem estado 1, então muda-se o estado da célula para 0.

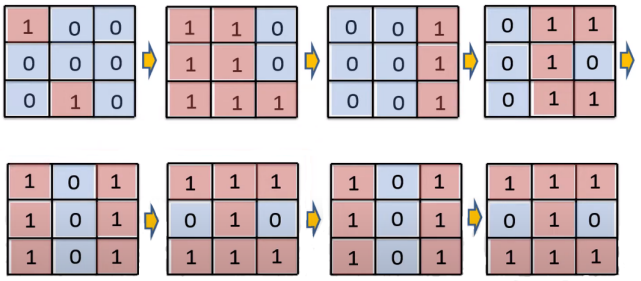

Figura 2.1: Exemplo de um Autómato Celular[1]

Este método permite gerar mapas ou estruturas em 2D, apresentando resultados rápidos e úteis, e como é determinístico, ao usar a mesma random seed, irá apresentar os mesmos resultados.

Agregação Limitada de Difusão

Agregação Limitada de Difusão é um processo onde as partículas são submetidas a um movimento browniano (movimentação aleatória de partículas no meio liquido e gasoso) [19]; quando estas encontram outras partículas acabam por se juntar e criar agregados de partículas.

Com este processo podemos gerar um mapa 2D, como o que está representado na figura 2.2, sendo que, primeiro, temos de colocar um pequeno agregado de pontos no mapa, e de seguida, disparar outros pontos em várias direções diferentes sequencialmente. Os pontos que forem contra o agregado de partículas irão colar-se a esse agregado de partículas, aumentando o mapa.

Diagramas de Voronoi

Um diagrama de Voronoi, como demonstrado na figura 2.3, é a partição de um plano em regiões que estão fechadas entre si. Cada região é uma célula (célula Voronoi) com os seus

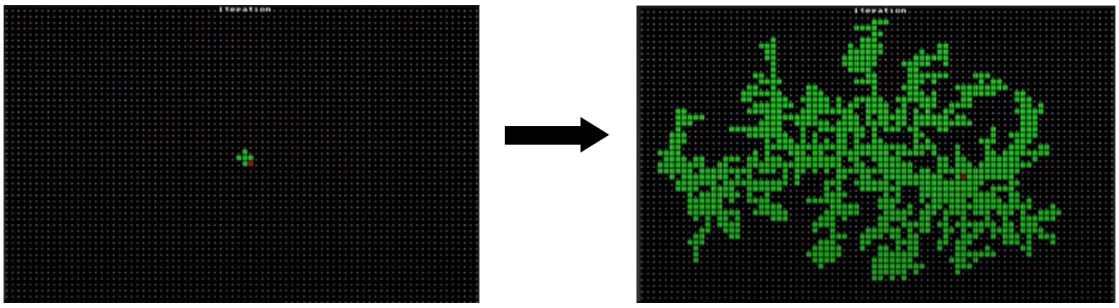

Figura 2.2: Exemplo do Algoritmo de Agregação Limitada de Difusão

pontos e vértices. Utilizando este diagrama, pode-se dividir um mapa em várias áreas para depois o popular com o que o level designer quiser. Este método de repartição de plano é aplicado em diversas áreas diferentes tais como, em arquitetura, aviação, inteligência artificial entre outros[20]. Em desenvolvimento de jogos, pode-se utilizar este algoritmo também de diversas formas, como por exemplo nos jogos de futebol, para controlar a distância entre jogadores ou para ajudar a controlar a lógica e o comportamento dos NPCs (exemplo: pode calcular se o NPC deve arriscar ir buscar um item que esteja fora da sua área/célula, podendo analisar se existem vários jogadores nessa área[21]). Em relação à geração de cenários, este diagrama é normalmente utilizado para definir regiões do mapa com determinadas características, como um ecossistema, em que se definem a fauna e flora de uma área, ou uma cidade com zonas residenciais, industriais ou comerciais[22]. Os diagramas de voronoi costumam ser utilizados juntamente com outros algoritmos para gerar cenários mais complexos, isto é, que exijam mais do que repartir espaços.

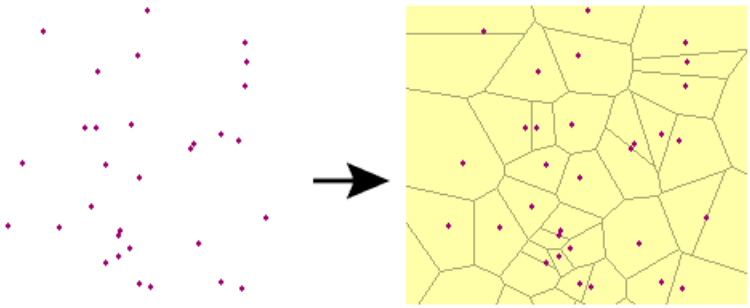

Figura 2.3: Exemplo de Diagrama de Voronoi[2]

Ruído de Perlin

O ruído de Perlin é um algoritmo de gradientes para a geração procedimental de conteúdo, sendo que é bastante utilizado na geração de cenários; é utilizado para gerar, por exemplo, texturas e terreno procedimentalmente[23]. Funciona em 2D e em 3D. Para cada ponto o algoritmo irá atribuir um valor entre 1 e -1 sendo este valor contínuo, e à medida que passa para os pontos seguintes o número a atribuir será sempre um número próximo do

anterior, para não haver uma diferença contrastante; no entanto, o valor poderá evoluir em sentido positivo ou negativo. Nas figuras 2.4 e 2.5 estão representados, como exemplo, um resultado em 2D e outro em 3D.

Nos jogos, é muito comum o seu uso, para gerar ou atribuir ao mapa as diferentes altitudes dos terrenos.

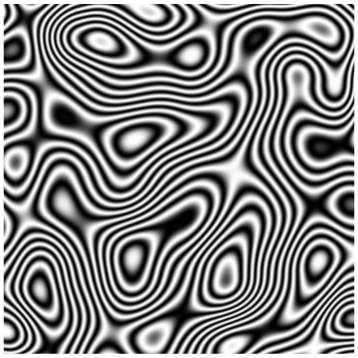

Figura 2.4: Exemplo de Ruído de Perlin em 2D [3]

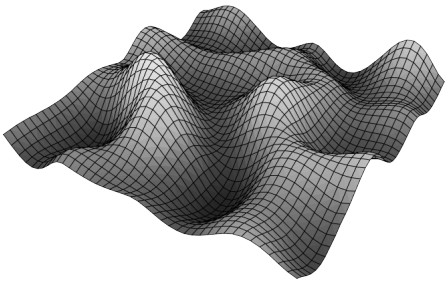

Figura 2.5: Exemplo de Ruído de Perlin em 3D [4]

Walker - Geração baseada em Agente

Um método que pode ser usado para gerar um mapa 2D, é o método baseado em agente. Este método cria um agente ou entidade, que está presente no mapa e se movimenta com regras definidas para implementar os tiles do mapa e desta forma criar o seu cenário. Os mapas gerados com este método costumam ser mais orgânicos e também mais caóticos, e não organizados. Como o comportamento do agente é inteiramente dependente do programador, o resultado da geração do cenário varia bastante pois o agente pode ser completamente estocástico, ou por outras palavras, ter um comportamento aleatório; alternativamente, o programador pode programar o agente para que consiga ver os passos seguintes e evitar fazer, por exemplo, interseções de corredores ou divisões[18].

Branching Trees

Este é um dos métodos mais populares utilizados para a geração procedimental de cenários. O Branching Trees é uma técnica que permite gerar cenários ramificados, isto é, onde o cenário é gerado a partir do ponto inicial, e todos os pontos seguintes gerados (à exceção dos pontos com objetivos) vão ser ramificações com becos sem saída, como demonstrado na figura 2.6. Embora este método funcione bem para cenários mais pequenos, demonstra as suas fragilidades à medida que se aumenta o tamanho do mapa a ser gerado. Esta fragilidade pode ser mitigada na fase de implementação do design das regras de geração do cenário.

Implementei esta técnica na geração dos cenários do jogo desenvolvi. Usei o ”Simple Room Replacement”para criar as divisões da masmorra, e o algoritmo de procura Prim para ligar essas divisões. A implementação está detalhada na secção 3.4.2.

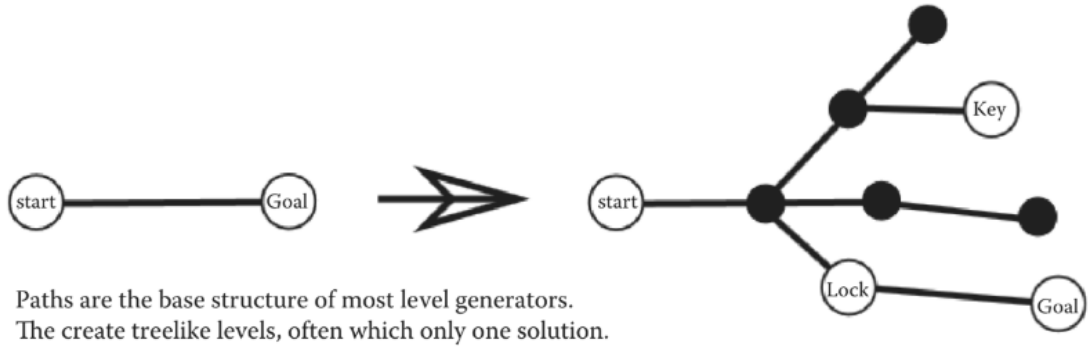

Figura 2.6: Topologia de um Grafo [5]

Geração baseada no motor de física

Esta técnica de geração, utiliza o motor de física incluído no game engine. No início do processo de geração, criam-se todas as divisões num único ponto, e atribuem-se as propriedades físicas a essas divisões. O motor de física irá separar aleatoriamente todas as divisões até deixarem de ficar sobrepostas entre si. É possível influenciar o motor para que espalhe as divisões de uma forma mais horizontal e vertical. Depois deste processo estar concluído removem-se algumas divisões e criam-se os corredores entre elas. Este método apresenta resultados diferentes consoante as definições do motor de física, e de algumas variáveis do algoritmo.

Aprendizagem Automática

Aprendizagem Automática ou Machine Learning, é um campo da Engenharia Informática que evoluiu do estudo de reconhecimento de padrões e da teoria da aprendizagem computacional em inteligência artificial. Permite criar programas com capacidades impressio-

nantes, capazes de fazer reconhecimento de imagens, processar linguagem natural, criar modelos de recomendação, entre outros. Esta tecnologia está a ser investigada também para ser utilizada em jogos; já existem várias técnicas interessantes que foram demonstradas em vários artigos publicados.

Para exemplificar, é possível gerar níveis do Super Mário, um jogo 2D, utilizando técnicas como o NeuroEvolution of Augmenting Topologies (NEAT) usando uma representação chamada Functional Scaffolding for Musical Composition (FSMC), pois permite que se cortem os vários níveis verticalmente numa sequência por cada tile horizontal e, usando esse dataset, pode treinar-se o modelo de inteligência artificial para prever qual seria a próxima sequência vertical e assim gerar o nível[24]. Este método também possibilita que os níveis gerados tenham em conta o comportamento do jogador, isto é, quanto mais itens o jogador apanhar e mais obstáculos ultrapassar, mais itens e obstáculos o algoritmo poderá gerar (nesse nível).

Algoritmo de Grafos Cíclico

A utilização deste tipo de algoritmo para a geração de cenários, partiu de Joris Dormans, professor assistente da universidade de Leiden e fundador do estúdio Ludomotion, como uma forma de poder gerar cenários que permitissem ao jogador concluir sempre o seu objetivo sem ter que voltar para trás, como acontece em muitos jogos roguelike[6].

Este algoritmo começa por gerar um grafo, onde existem dois nós interligados por duas ligações, e todos os subsequentes nós, que representam as várias divisões e as suas características, serão gerados sempre com dois caminhos que se ligam a outros nós. Os caminhos poderão ser bidirecionais ou unidirecionais. Na figura 2.7 podemos ver como este conceito se reflete no desenho da topologia do grafo. Esta técnica permite que hajam dois caminhos para se poder chegar ao fim, ou alternativamente, permite que o jogador depois de ter atingido o objetivo (adquirido o tesouro), volte a subir os níveis sem repetir o caminho feito anteriormente, oferecendo-lhe uma nova experiência no mesmo nível.

Para alterar o grafo inicial para um mais complexo, como evidenciado na figura 2.7, uma das formas de o fazer, é utilizando regras de transformação que implementam padrões de desenho para alterar a topologia do grafo. Joris Dormans criou várias regras de transformação que estão refletidas na figura 2.8, as quais permitem criar padrões de desenho diferentes resultando numa topologia mais complexa e interessante para gerar o cenário. Estas regras de transformação são aplicadas várias vezes sobre a topologia da masmorra, de modo a desenvolver a topologia para uma complexa e estruturalmente interessante. Para além da geração e transformação da topologia do grafo, é necessário criar uma camada de abstração para gerar o cenário no jogo pois não existe uma relação direta entre o grafo e o mapa do jogo.

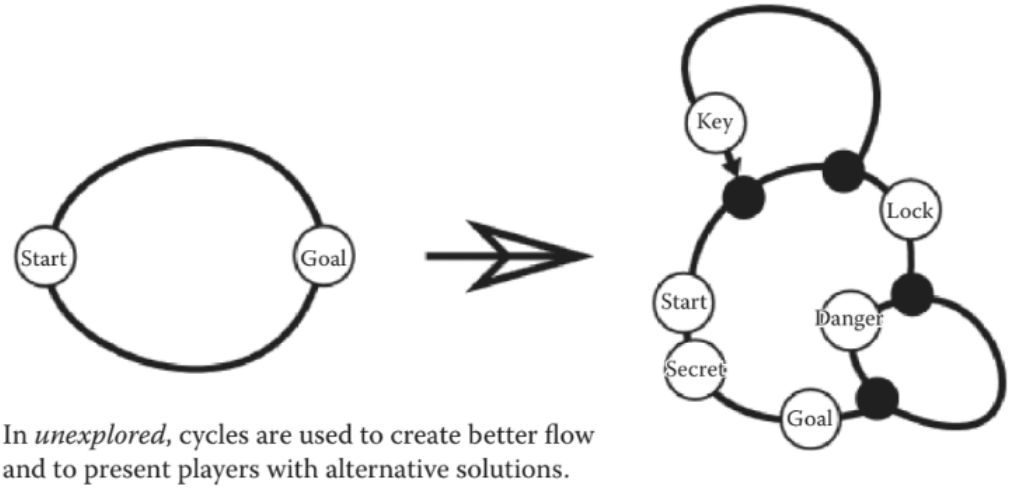

Figura 2.7: Topologia de um Grafo [5]

2.2 Trabalho relacionado

Infelizmente existe muito pouca investigação sobre a Geração Procedimental de Conteúdo utilizando um algoritmo de grafos cíclico, pelo que aproveito para apresentar neste capítulo, dois jogos roguelike que utilizam geração procedimental extensivamente e que tomei como inspiração neste trabalho.

2.2.1 Unexplored

Joris Dormans, realizou um trabalho muito importante relacionado com o algoritmo de grafos cíclico ao desenvolver um jogo roguelike onde as masmorras são implementadas por este algoritmo.

Trata-se do jogo "Unexplored"que foi lançado no mercado em 2017. Neste jogo 2D roguelike, o jogador tem por objetivo encontrar um amuleto que se encontra escondido numa masmorra e sair dela com vida. Para tal, tem de percorrer vários níveis da masmorra encontrando itens, resolvendo puzzles, ultrapassando vários obstáculos e derrotando inimigos até encontrar o amuleto de Yendor que está guardado por um poderoso dragão.

Tanto as masmorras como tudo o que está presente nelas, desde as divisões até aos inimigos e itens, é gerado na topologia do algoritmo de grafos cíclico, automaticamente. Como referido na secção 2.1.1, este algoritmo tem a característica de todos os nós estarem ligados por dois caminhos, permitindo a existência de percursos alternativos ao longo do jogo, possibilitando ao jogador, após obter o amuleto, efetuar o caminho de regresso com um percurso diferente daquele que percorreu para o encontrar, oferecendo-lhe uma nove experiência em cada nível de jogo.

Joris Dormans utiliza também regras de transformação para implementar padrões de desenho e assim alterar a topologia do grafo, criando topologias da masmorra mais com-

plexas e interessantes para o jogador.

Na figura 2.8 estão representados alguns padrões de desenho possíveis na geração de cenários. A geração do cenário inicial do jogo "Unexplored"[6], costuma corresponder aos exemplos representados à esquerda na figura 2.8. Nestes gráficos podem observar-se os dois nós iniciais e os dois caminhos de ligação entre eles; os dois caminhos podem ser ambos longos ou ambos curtos ou, alternativamente, uma combinação dos dois, isto é um longo e um curto. Os caminhos podem ter também direções diferentes.

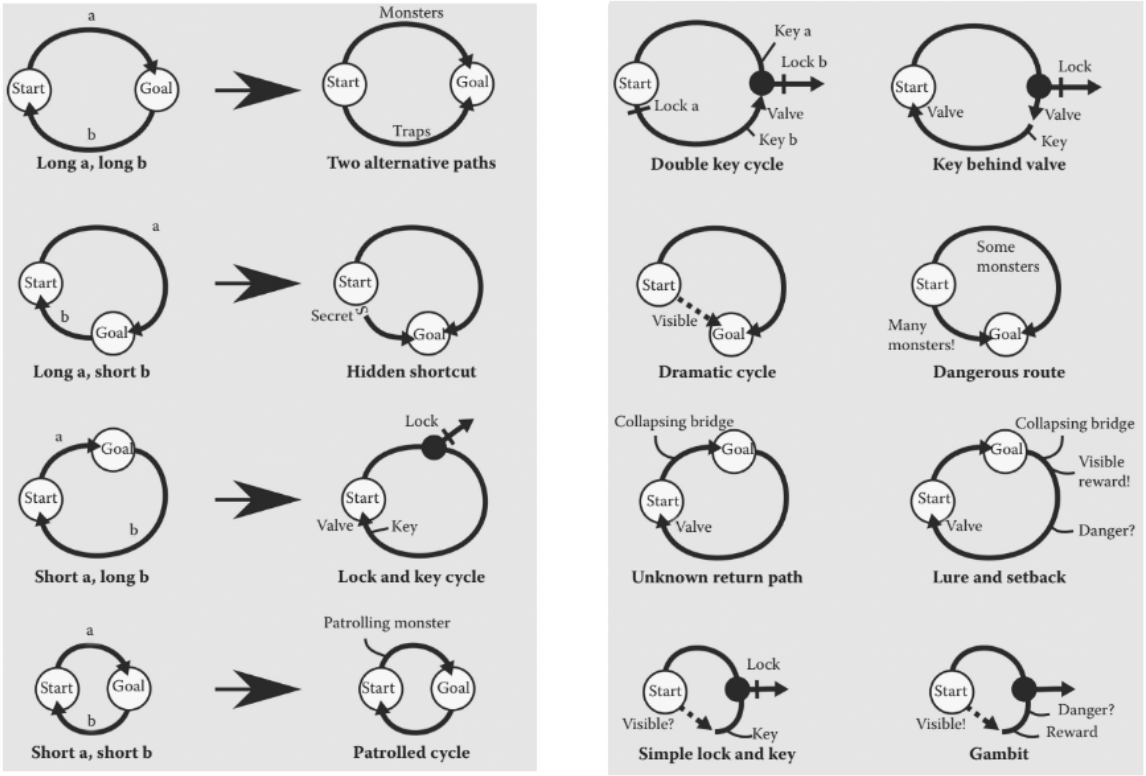

Figura 2.8: Padrões de Desenho no Unexplored [5]

Tendo a base da topologia definida, o algoritmo irá modificar o cenário inicial por um mais rico, inserindo itens no caminho (monstros, armadilhas, etc), e/ou utilizando o "Lock and Key", uma mecânica também representada na figura 2.8, a qual aproveita as potencialidades do algoritmo. Esta mecânica tem por princípio gerar uma porta e a correspondente chave para a abrir. Esta porta pode assumir várias formas tais como uma porta normal, um rio de lava ou uma cascata de água a correr.

O algoritmo de grafos cíclico é adequado para fazer a implementação desta técnica pois pode assegurar, dependendo das regras definidas, que existem caminhos que passem por todos os pontos de interesse do cenário sem que haja becos sem saída que obriguem o jogador a voltar para trás, garantindo que seja possível adquirir a chave em pontos diferentes da masmorra sendo todos eles acessíveis.

Quanto à geração do "Unexplored", o algoritmo começa a gerar os 20 níveis do jogo

logo que o jogador entra na página de personalização da personagem, permitindo assim a redução do tempo de carregamento desses níveis na ótica do jogador. A geração de todos os níveis é realizada por duas componentes: o gerador da tipologia do grafo e o gerador do cenário da masmorra.

- Para cada nível, o gerador da tipologia começa por gerar o grafo com os nós e os respetivos caminhos, que tanto podem ser bidirecionais como unidirecionais; seguidamente, vai gerar o primeiro ciclo, com o "Lock and Key"e o caminho alternativo. Depois, gera os restantes detalhes para o nível, isto é, os inimigos que patrulham, os obstáculos e os vários itens. Nesta fase é também decidido o tema do nível (ex: nível com características de fogo e lava) e qual o tipo de "Lock and Key"que deverá ser utilizado, atendendo às características temáticas definidas (figuras 2.9 e 2.10); neste passo o gerador também define os tamanhos das divisões.

- A última etapa cabe ao gerador do nível, o qual vai converter o grafo gerado e os seus atributos para o mapa do jogo colocando-os no cenário, vai introduzir as decorações no mapa e outras características como as formações rochosas do cenário e o formato de cada divisão, tal como se pode observar nas figuras 2.11 e 2.12.

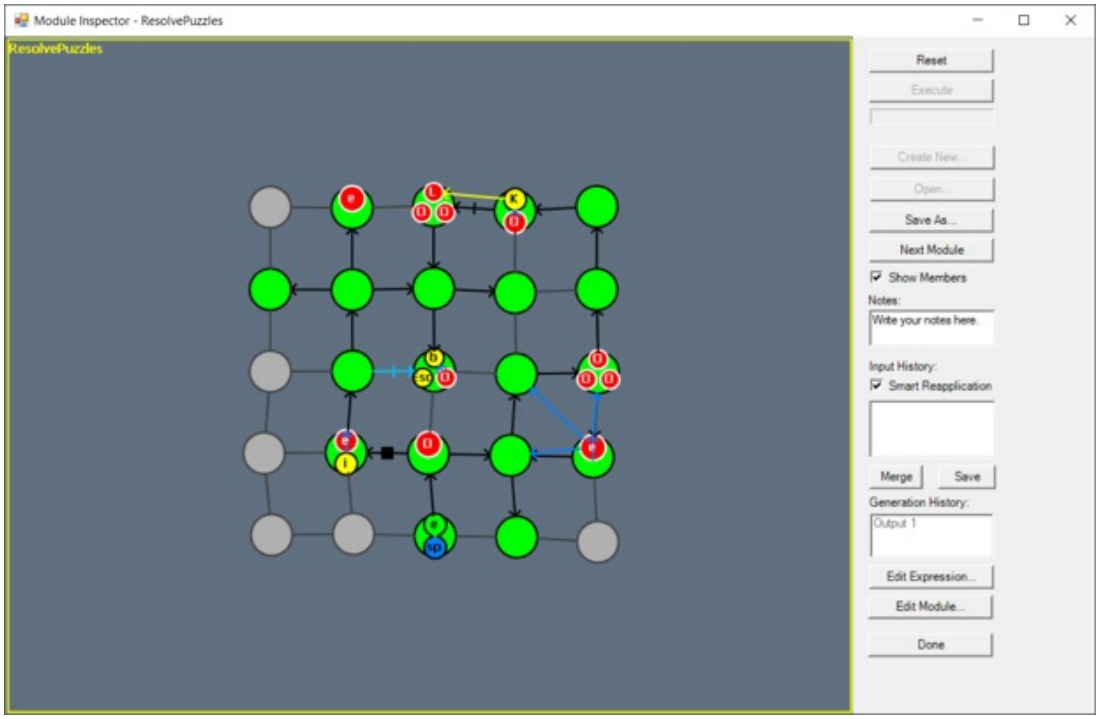

Figura 2.9: Geração Inicial da Topologia no Unexplored [6]

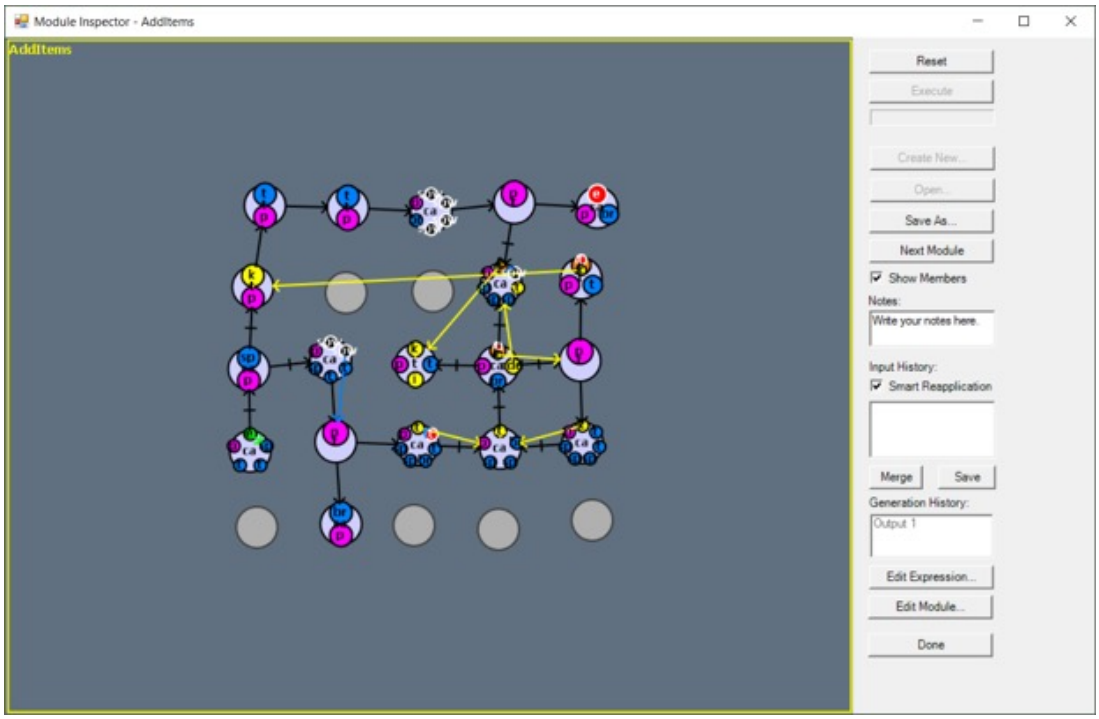

Figura 2.10: Geração Final da Topologia no Unexplored [6]

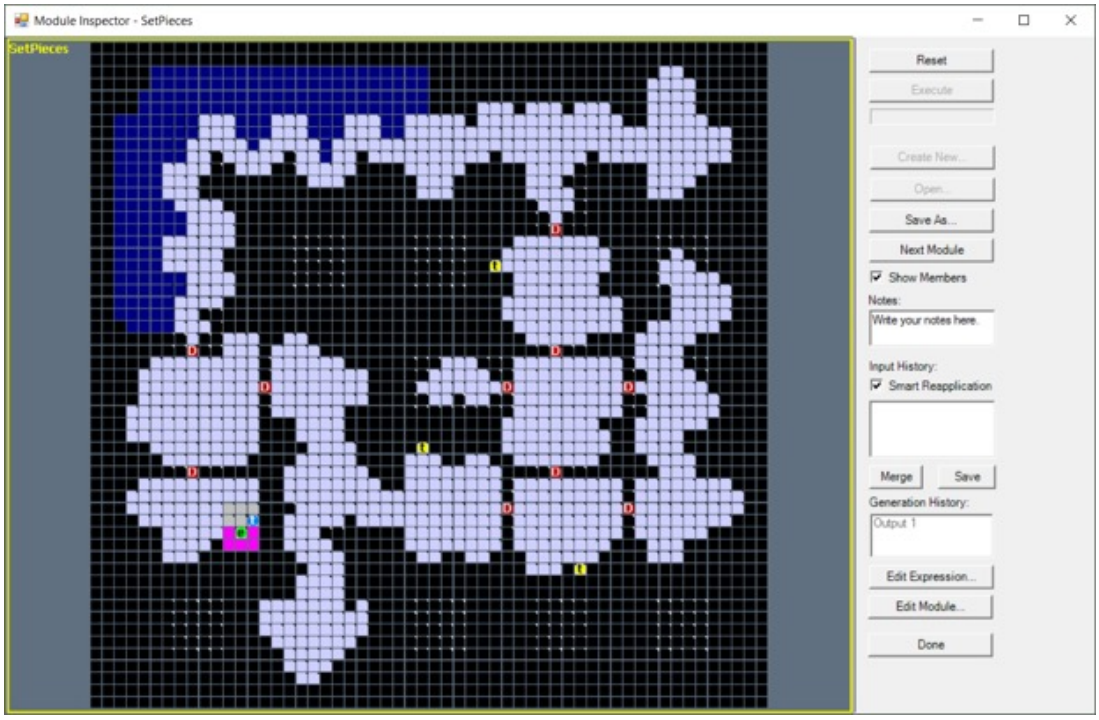

Figura 2.11: Primeira etapa da geração do mapa no Unexplored [6]

Figura 2.12: Segunda Etapa da geração do mapa no Unexplored [6]

2.2.2 Pokémon Mystery Dungeon: Red Rescue Team

Acho importante referir ainda um outro jogo que também me inspirou, apesar de utilizar uma técnica diferente dos grafos cíclico. Trata-se do jogo "Pokémon Mystery Dungeon: Red Rescue Team", lançado no final de 2006 para o GameBoy Advance. Este jogo possui diversos elementos roguelike: o facto de ser baseado em masmorra, de ter mecânicas de combates por turnos e a característica mais importante, a de utilizar a geração procedimental de conteúdo na construção dos níveis da masmorra, divisões e corredores, objetivos, itens e inimigos.

Contudo, o método de geração deste jogo é baseado em agente, pelo que o agente se desloca pelo mapa a colocar os sprites e os itens no mapa da masmorra. Este método funciona da seguinte forma [23]:

- O primeiro passo da geração do jogo começa pela colocação das paredes à volta do mapa para garantir que não existe possibilidade de o jogador sair desse mapa.

- De seguida procede-se à geração aleatória de divisões ao longo do mapa; de notar que, neste passo, o mapa está dividido em partes iguais por uma grelha, para evitar que o agente crie divisões muito concentradas no mapa. O número de partições da grelha é determinado pelo tipo de algoritmo usado para o nível.

- Após as divisões serem geradas, o agente inicia a sua ligação (das divisões) e, para isso, escolhe 2 pontos de paredes de divisões diferentes, que sejam paralelas entre si, procedendo à sua ligação. Neste passo é possível que o agente escolha gerar

intencionalmente um caminho que volte a ligar à mesma divisão ou um caminho sem saída que irá até ao limite do mapa, até estar muito próximo de uma divisão.

- O agente coloca nas entradas das divisões um chão especial para que não seja gerado nenhum objeto que possa prejudicar o jogador.
- Dependendo do algoritmo em utilização, o agente poderá adicionar características ambientais ao mapa, tais como rios de água ou lava.
- Por último, o agente não só insere os objetos no mapa (itens, inimigos, a saída do nível), como aplica um chão especial na entrada das divisões para assegurar que os objetos não são colocados nas respetivas entradas, impedindo o jogador de prosseguir.

O estúdio que desenvolveu este jogo optou por implementar mais de 10 algoritmos diferentes para a geração de masmorras, todos eles baseados no algoritmo descrito acima, mas introduzindo diferentes parâmetros em cada um. Exemplificando:

- 3 destes algoritmos apenas diferem na quantidade de espaço do mapa que irão ocupar (metade, dois terços, ou a totalidade do mapa).
- Um outro algoritmo cria muitas divisões perto dos limites do mapa sem gerar no centro, criando uma estrutura circular.

Embora este processo seja um pouco ineficiente (obriga a repetir o mesmo código várias vezes, apenas alterando alguns parâmetros, ocupando desnecessariamente espaço no armazenamento do jogo), permite obter diferentes resultados de cenários, evidenciando a flexibilidade do método baseado em agente.

2.3 Ferramentas

Irei apresentar neste capítulo as ferramentas que considerei e as que decidi utilizar para o desenvolvimento do jogo.

2.3.1 Game Engines

Um Game Engine é um programa de software para ajudar no desenvolvimento de jogos. Este programa contém funcionalidades muito importantes, que tratam de muitos aspetos do desenvolvimento, tais como, desenhar os gráficos (em 2D ou 3D), um motor de física, som, lógica, animação, inteligência artificial, networking, gestão de memória, entre outros.

Os game engines são dotados de um ambiente de trabalho gráfico que permite efetuar várias partes do desenvolvimento do jogo, na janela do programa. É possível programar

o jogo, colocar os objetos, iluminação e sons no espaço do jogo, fazer a animação dos objetos, etc... Este ambiente de trabalho juntamente com as bibliotecas e ferramentas disponíveis nos game engines, ajudam os estúdios de jogos a poupar tempo e dinheiro, permitindo o enfoque dos trabalhadores em aspetos mais importantes do jogo.

Os game engines também seguem uma ordem determinada de execução que, apesar de variar de game engine para game engine, seguem a mesma filosofia. Esta ordem de execução é tipicamente um ciclo, que possui a seguinte forma:

- Início do programa
- Verificação do input
- Execução da função de atualização
- Apresentação dos gráficos
- Fechar programa se o utilizador desejar

Este ciclo está sempre a ser executado enquanto o jogo está ativo. Na figura 2.13 pode ver-se a ordem de execução de um script no unity, onde se pode verificar que a complexidade é muito maior num game engine comercial.

Antes dos game engines existirem, cada jogo tinha de ser escrito de raiz. Nos anos 80 os jogos arcade dominavam; cada máquina arcade tinha o seu próprio hardware, o que significava que, no final da produção de cada jogo, o código e a tecnologia gerada para concretizar o seu desenvolvimento, não voltavam a ser utilizados, sendo muitas vezes deitados fora. E só para exemplificar, era necessário criar de raiz os drivers para a correta utilização do ecrã para um jogo da Atari 2600. Os game engines foram criados para facilitar e simplificar o desenvolvimento dos jogos, ao tratarem dos aspetos comuns relativos ao desenvolvimento dos mesmos.

Gamemaker Studio 2

O Gamemaker Studio foi lançado em 1999. Este game engine é focado no desenvolvimento de jogos em 2D; tem suporte para vários tipos de gráficos, como sprites e vector art. Possui um IDE (Intergrated Development Environment) onde se podem desenvolver os projetos quase sem necessitar de outros programas; contudo, este não tem um digital audio workstation (para fazer a música). A linguagem de programação utilizada é o GameMaker Language que é semelhante a Javascript; alternativamente pode também utilizar-se programação visual onde se utilizam blocos para programar a lógica. Todos os projetos podem ser exportados para todos os dispositivos relevantes, tais como, computadores (Windows, Mac, Linux), telemóveis (Android, iOS), consolas (Playstation, Xbox, Nintendo) e também para a web. No entanto só se pode exportar consoante o tipo de

subscrição que se escolher[25]. Este game engine é pago, embora haja uma modalidade gratuita que permite experimentá-lo e desenvolver um jogo, embora este jogo só possa ser exportado para a plataforma GX games.

O gamemaker possui boa documentação de suporte estando também disponíveis tutoriais online.

Unity

O Unity é um dos game engines mais conhecidos, que permite o desenvolvimento de jogos em 2D e 3D. Como todos os game engines, este possui também um IDE para fazer o desenvolvimento do jogo; no entanto a programação não ocorre no IDE sendo necessário descarregar o Visual Studio para concretizar essa tarefa. A linguagem de programação é o C#, bastante utilizada também fora do desenvolvimento de jogos, como por exemplo, no desenvolvimento de aplicações de computador, telemóvel e também de servidores web. O Unity também oferece uma alternativa de programação visual chamada Bolt. Este game engine é pago embora tenha uma modalidade gratuita bastante boa, pois permite que se desenvolvam os jogos e permite exportá-los para computador e telemóvel sem pagar; já para exportá-los para consolas é necessário ter uma subscrição paga[26].

A documentação para este game engine é bastante completa, existindo imensos tutoriais que não só ensinam quase todos os aspetos do Unity, como também existem vários exemplos de como implementar as mais diversas funcionalidades.

Unreal Engine

O Unreal Engine é um game engine tão popular como o Unity. Este programa não só é utilizado para o desenvolvimento de jogos 3D, como também é utilizado na indústria cinematográfica, na indústria de transportes, em arquitetura e em simulação. O Unreal Engine tem várias tecnologias que o tornam extremamente bom na criação de jogos 3D, como por exemplo o Nanite, um sistema que permite ter mapas de grande dimensão nos jogos sem haver uma grande penalidade no desempenho do jogo. Outro exemplo é a tecnologia Temporal Super Resolution, que permite que o jogo seja renderizado numa menor resolução do ecrã sem afetar a fidelidade dos pixeis[27].

Em relação ao processo de desenvolvimento no Unreal Engine, tal como no Unity, tem de se utilizar o Visual Studio para efetuar a programação. A linguagem de programação é o C++, sendo que também existe uma linguagem visual de programação chamada de Blueprints. Num projeto, estas duas linguagens podem ser utilizadas em conjunto permitindo que pessoas que não tenham experiência, ou sistemas que não sejam críticos em relação ao desempenho, sejam programados em Blueprints mais facilmente e rapidamente. Os jogos podem ser exportados para todas as plataformas, isto é, para o Windows, macOS, Linux, Android, iOS, Nintendo Switch, Playstation, Xbox. O Unreal Engine também oferece uma boa documentação e tutoriais.

Quanto ao preço, este game engine é gratuito para todos os projetos que faturem até 1 milhão de dólares; caso esse valor seja ultrapassado, então a ter-se-á de pagar 5% de royalties, por trimestre, sobre a receita do projeto.

Godot

O Godot é um game engine open-source utilizado para o desenvolvimento de jogos em 2D e 3D. Ao contrário da maioria dos game engines este software é bastante leve, ocupando apenas 70mb de espaço. Este game engine aceita várias linguagens de programação como c-sharp, C/C++ e GDScript sendo esta última, a linguagem criada especificamente para o game engine. Os jogos desenvolvidos podem ser exportados para computadores e telemóveis[28].

Eu optei por escolher o Godot devido à sua forma única de desenvolvimento do jogo, com o seu sistema de scenes (cenas) e nodes (nós) e pela linguagem de programação que utiliza. O GDScript é uma linguagem muito parecida a Python sendo por isso fácil de adotar e programar. E por último, também possui uma boa documentação de suporte que, embora ainda possa melhorar, está extremamente bem integrada dentro do game engine, permitindo que não seja necessário abrir o web browser no caso de haver necessidade de a consultar.

2.3.2 Frameworks/Bibliotecas

As frameworks diferem muito dos game engines por não possuírem funcionalidades avançadas, mas são mais personalizáveis e adaptáveis ao desenvolvimento de um projeto. As frameworks não têm um IDE onde desenvolver o jogo, não têm uma janela para ver, por exemplo, a colocação dos gráficos (à exceção do RayLib) e costumam suportar só uma linguagem. Estas frameworks focam-se mais no desenvolvimento do projeto através da programação e acabam por ser mais maleáveis aos requisitos dos projetos e à forma de implementação das mecânicas que o developer desejar usar. Na tabela da figura 2.1, estão representadas as funcionalidades suportadas por cada game engine ou framework.

Monogame

O Monogame é uma framework baseada numa biblioteca da Microsoft chamada de XNA, que foi descontinuada em 2013. Esta biblioteca era muito apreciada pela comunidade de game developers, os quais, com base nela, criaram o Monogame[29]. Esta framework é para ser utilizada com a linguagem de programação C# e com o Visual Studio como ambiente de programação. O Monogame apenas oferece um template inicial para ajudar na criação inicial do jogo, embora, graças ao Visual Studio, possa oferecer uma boa consola de debugging (procura de erros ou comportamentos inesperados). Uma desvantagem bastante significativa é a falta de documentação; existem poucos tutoriais sobre como

	FEATURES						
GAME ENGINES / FRAMEWORKS	2D	3D	IDE	INTEGRAÇÃO CODE EDITOR	VERSÃO GRATUITA	EXPORTAÇÃO DE JOGOS PARA DIFERENTES PLANTAFORMAS	BOA DOCUMENTAÇÃO
Monogame	✔	✔			✔	✔	
Raylib	✔	✔			✔	✓	✔
Heaps	✔	✔			✔	✔	✔
Godot	✔	✔	✔	✔	✔	✓	✔
Unity	✔	✔	✔		✔	✔	✔
Unreal Engine		✔	✔		✔	✔	✔
GameMaker Studio 2	✔		✔	✔	✔	✔	✔

Tabela 2.1: Comparação das várias ferramentas

trabalhar com esta framework, embora o API esteja documentado. Esta framework é open-source e permite exportar para diversas plataformas, incluindo as consolas de jogos.

Raylib

O Raylib é uma biblioteca específica para fazer programação de jogos. Embora esta biblioteca seja escrita em C, existem dezenas de linguagens de programação que se podem usar, das mais conhecidas, às linguagens que ainda estão em fase de desenvolvimento[30]. Uma grande vantagem do Raylib é a sua cheatsheet, uma folha que detalha muito sucintamente, todas as funções e respetiva descrição. Existem também exemplos disponíveis no site para quem queira aprender. Quanto à exportação dos projetos, o Raylib suporta várias plataformas; no entanto tal depende da linguagem escolhida para desenvolver o projeto pois o suporte extra às linguagens é mantido por outras pessoas e dependente de outras tecnologias.

Heaps

O Heaps é uma framework que já foi utilizada em jogos comerciais, tais como o “Dead-cells”, “Northgard”ou “Evoland”e “Evoland 2”. Tal como as outras frameworks aqui enunciadas, esta também é open source[31]. A grande diferença é a linguagem de programação usada nesta framework, chamada de Haxe. Esta linguagem é uma linguagem orientada por objetos, tendo uma sintaxe semelhante a Java ou C#. Uma vantagem que o Haxe oferece é poder compilar o código para outras linguagens como C++, Javascript, ou Lua.

O Heaps permite desenvolver jogos tanto em 2D como em 3D, possui uma boa documentação e bons exemplos para se aprender. Todos os projetos desenvolvidos com o Heaps podem ser exportados para todas as plataformas.

2.3.3 Editor Gráfico

O editor gráfico que escolhi utilizar foi o Aseprite[32]. Esta é uma boa ferramenta para desenhar gráficos 2D do tipo pixel art. O Aseprite é utilizado por muitos profissionais, possuindo imensas funcionalidades que assistem no desenho de pixel art. Este programa é pago, estando disponível em várias lojas digitais, tais como a Steam, Itch.io e o Humble Bundle.

2.4 Sumário

Existem muitos algoritmos e técnicas diferentes para gerar conteúdos. Neste capítulo foram revistas e caracterizadas nove destas técnicas, as quais, apesar de possuírem capacidades diferentes, podem ser utilizadas na criação de algoritmos complexos de gestão procedimental de conteúdo.

Foi também analisado o trabalho concretizado por Joris Dormans no seu jogo "Unexplored", onde um jogador percorre os níveis procedimentalmente gerados por um algoritmo cíclico de grafos. A característica diferenciadora deste algoritmo é o facto de poder planear e gerar uma estrutura interligada fazendo com que o jogador não tenha que voltar atrás no caminho.

O jogo "Pokémon Mystery Dungeon"foi também abordado neste capítulo, por ser um jogo roquelike e utilizar a geração procedimental de conteúdos na construção das masmorras.

Em relação às ferramentas necessárias para o desenvolvimento do jogo desta dissertação, de todas as ferramentas analisadas, escolhi o Godot Engine e o Aseprite. Este game engine é muito bom para jogos 2D, proporcionando um ambiente de trabalho simples e responsivo com editor de código integrado; a linguagem GDScript é fácil de utilizar, sendo semelhante a Python, e o sistema de cenas e nós, embora seja menos intuitivo no inicio, é na verdade um sistema muito capaz e versátil.

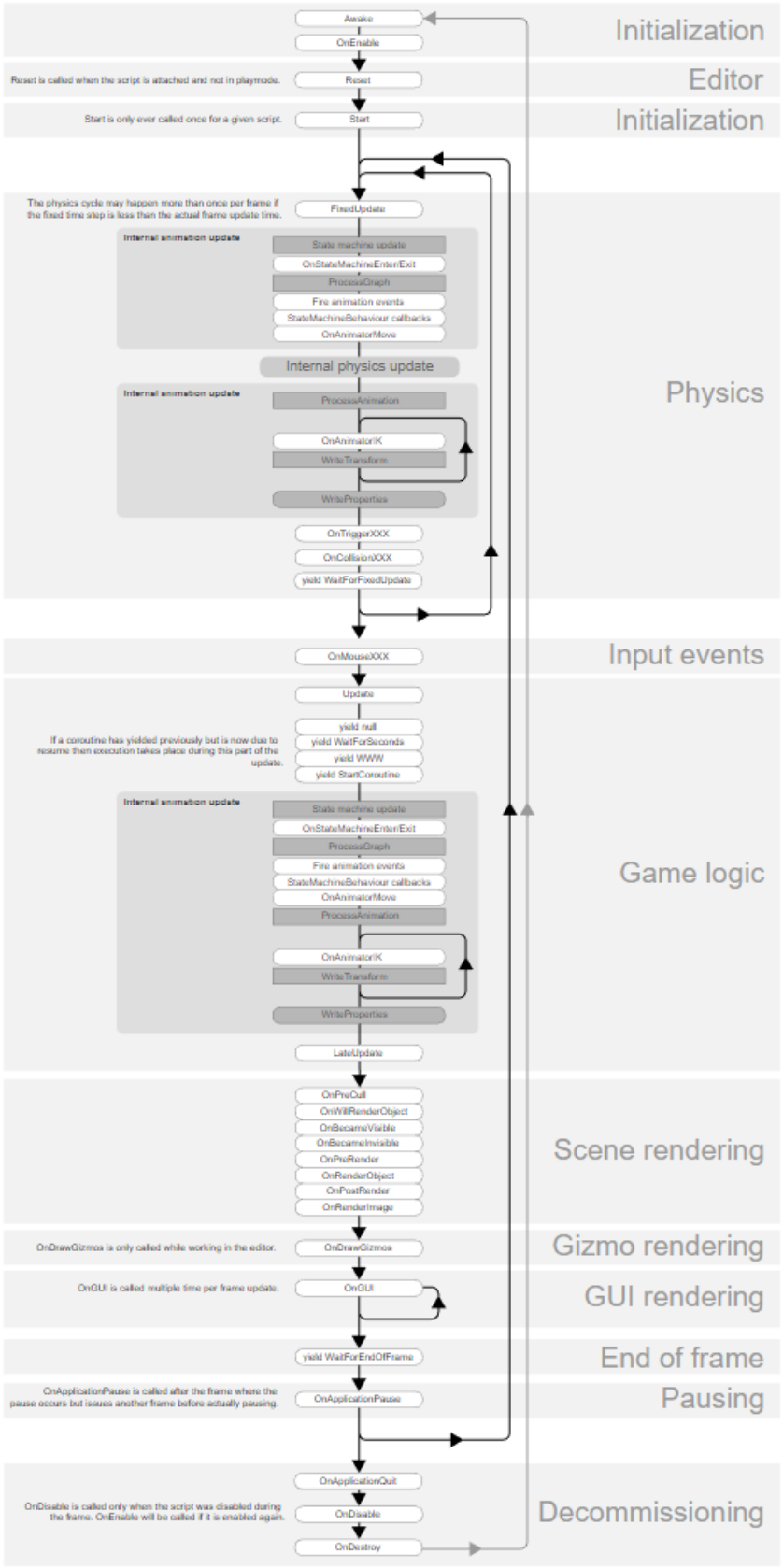

Figura 2.13: Ordem de Execução de um Script no Unity[7]

Capítulo 3

Análise

Neste capítulo irei explicar o processo de desenvolvimento do jogo e dos algoritmos utilizados, bem como analisar as vantagens e desvantagens encontradas na relação entre o algoritmo de grafos cíclico e os restantes. Também irei expor as dificuldades que tive no desenvolvimento do jogo.

3.1 O Jogo - Descrição Geral

Como referido anteriormente, este estudo incide sobre a implementação do algoritmo de grafos cíclico, no desenvolvimento de um jogo 2D roguelike, para o qual defini mecânicas simples, dado que, o que pretendo analisar, é o potencial do algoritmo. Neste sentido, identifico algumas das mecânicas planeadas:

- Geração do mapa, com algumas regras de transformação.
- Uma personagem para o jogador controlar.
- Um tipo só de inimigo.
- Mecânica de combate.
- O mecanismo "Lock and Key".

A construção do jogo inicia-se com a geração do nível da masmorra, com as suas divisões e corredores. A personagem do jogador irá nascer na divisão que for denominada como o início da masmorra e terá como objetivo ir passando sempre para o nível seguinte, sendo que terá de se colocar em cima do tile com escadas para o fazer. Para tal o jogador terá de ultrapassar quaisquer obstáculos que o algoritmo gere na masmorra, desde enfrentar inimigos até à mecânica "Lock and Key", o que inclui encontrar a chave para desbloquear o acesso à escada para passar de nível.

3.1.1 Explicação do jogo

Nesta secção irei explicar um pouco as mecânicas implementadas no jogo.

Personagem

Figura 3.1: O Jogador

A personagem implementada no jogo e controlada pelo jogador pode mover-se livremente pelo mapa, não podendo porém, ultrapassar as paredes e os obstáculos. A personagem possui inicialmente 4 valores de vida; caso sofra danos, será subtraído 1 valor de vida à sua vida total e se chegar a 0, então é ativada a função de criar um novo nível. Podem existir, no mapa, poções que aumentam a vida do jogador. O jogador pode atacar inimigos atirando facas; cada faca tira 1 unidade de vida aos inimigos.

Inimigos

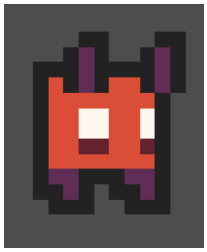

Figura 3.2: Inimigo

Foi implementado neste jogo, um único tipo de inimigo, cujo o objetivo é perseguir o jogador caso ele se aproxime demasiado. O inimigo tira um valor de vida ao jogador por cada toque que ocorra entre ambos.

Puzzle

O puzzle funciona da seguinte forma: quando o jogador se aproximar do puzzle, este irá mostrar uma sequência de cores que o jogador terá de copiar. Ao introduzir o código correto, o puzzle dará uma poção de vida. De referir, que o puzzle também poderia dar uma chave ou qualquer outro tipo de objeto embora isso não tenha sido implementado.

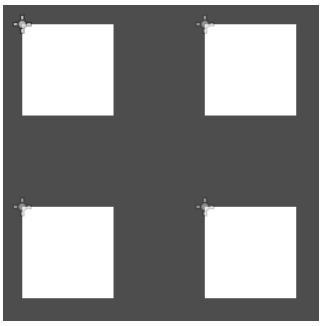

Figura 3.3: Puzzle

Tesouros

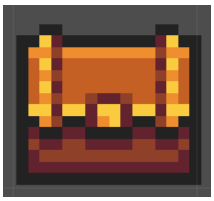

Figura 3.4: Tesouro

Foi construída uma sala com a função de atrair o jogador por caixas de tesouro onde irá receber um item por cada baú do tesouro; mas cuidado, pois estas salas podem ter alçapões escondidos, sendo que estes estão representados por quadrados pretos. Se o jogador cair no alçapão é criado um novo nível.

Lock and Key

Foi implementada uma versão simples do lock and key; esta versão apenas bloqueia as escadas para o próximo nível e para as desbloquear o jogador terá apenas que recolher a chave.

3.2 Algoritmos de procura

No desenvolvimento dos algoritmos para a geração da masmorra é necessário implementar um algoritmo que crie corredores para ligar as divisões. Em todas as implementações descritas nesta dissertação, apenas o Walker não necessita de um algoritmo para gerar esses corredores.
Para este jogo, utilizei dois algoritmos: o A* e o algoritmo de Prim.

A*

O A* (A-star) é um algoritmo utilizado para procurar caminhos entre dois pontos. Este algoritmo é uma extensão do algoritmo de Dijkstra, sendo que introduz uma heurística no

algoritmo, permitindo calcular uma estimativa do custo total do caminho. Esta diferenciação permite ao A* ser melhor que o Dijkstra, sempre que o espaço de procura da solução é grande para a tarefa de calcular o caminho mais curto entre dois pontos[33].

Para se averiguar o caminho mais curto de um trajeto, é associado um "custo"aos diversos pontos possíveis dos caminhos; desta forma o algoritmo pode avaliar as várias alternativas, isto é, selecionar o caminho com menor custo associado. A partir do ponto inicial, o A* testa todas as direções possíveis e seleciona, a partir do cálculo da heurística, o ponto adjacente com menor custo, integrando-o no caminho a percorrer, repetindo o processo até atingir o ponto final.

Neste algoritmo é utilizada uma fórmula muito importante: F=G+H

- A letra F representa o menor custo estimado entre o ponto inicial e o de destino. Traduz o caminho mais eficiente entre o ponto inicial e o final.
- A letra G representa o custo desde o ponto inicial até ao ponto atual.
- A letra H é o custo estimado do ponto atual até ao destino. É a heurística que estima o custo do caminho que falta percorrer.

O algoritmo A* segue os seguintes passos:

1. Criar duas listas.
2. Adicionar o ponto de partida à lista número 1.
3. Para todos os pontos adjacentes, encontrar o caminho com menor custo.
4. Na segunda lista, para todos os pontos adjacentes do ponto atual:
   (a) Se não for possível chegar ao ponto, então esse ponto é ignorado.
   (b) Caso seja possível e o ponto não estiver na lista número 1, então colocá-lo lá e calcular o F, o G, e o H desse ponto.
   (c) Caso seja possível e o ponto já estiver na lista número 1, então averiguar se o caminho a partir desse ponto possui um menor custo que o atual.
5. Repetir os passos 3 e 4 até:
   - Chegar ao destino
   - Verificar que não é possível chegar ao destino.

Algoritmo de Prim

O algoritmo de Prim é um algoritmo ganancioso[34], isto é, um algoritmo que segue a heurística de atingir a melhor opção, local e não global, em todas as fases do problema. A vantagem do algoritmo ser ganancioso é a de poder atingir um resultado próximo da melhor solução possível num período de tempo razoável. O algoritmo de Prim começa pelo ponto inicial do caminho, e vai-se movendo através dos vários pontos adjacentes, explorando todas as possibilidades de ligação. Funciona da seguinte forma[35]:

1. Escolhe-se o ponto inicial do mapa para começar o caminho.

2. O algoritmo começa a determinar os potenciais caminhos, ao escolher o ponto com o menor custo de ligação. Este custo de ligação é previamente definido manualmente. Caso haja dois caminhos com o menor valor então o caminho é escolhido aleatoriamente.

3. E assim sucessivamente até haver um mapa com as ligações entre os nós com o menor custo possível.

3.3 Implementação do Algoritmo de Grafos Cíclico

3.3.1 Geração da Topologia do Grafo

Para gerar um nível de uma masmorra utilizando grafos é preciso, em primeiro lugar, realizar a sua implementação. Os game engines e especialmente o que eu escolhi, o Godot, não incluem nenhum tipo de implementação ou pacote que facilite o desenvolvimento de grafos ou que traduza a topologia do grafo para o mapa.

Para iniciar a implementação dos grafos, é necessário ter em conta a forma como se irá guardar a informação dos nós que foram gerados e de como estão interligados. O método que desenvolvi acaba por ter algumas limitações, pois os nós são gerados em sequência, isto é, inicialmente gera-se a parte de cima e/ou de baixo da topologia do grafo sem quebrar a linearidade, como demonstrado na figura 3.5. No entanto, para colmatar esta limitação, efetuei algumas alterações de modo a que fosse possível não só quebrar o ciclo, como demonstrado na figura 3.6, mas também fosse possível a adição de um novo ciclo ao ciclo principal como demonstrado na figura 2.7.

Em relação ao processo de geração de nós para o grafo, este efetua-se através de um gerador de números pseudo-aleatórios, que dependendo da probabilidade dada, irá ou não gerar o novo nó e as suas ligações. Estes nós representam divisões que podem conter inimigos, baús de tesouro, armadilhas, puzzles. O gerador da topologia da masmorra foi implementado de forma a possibilitar padrões de desenho tais como os que estão representados na figura 3.7.

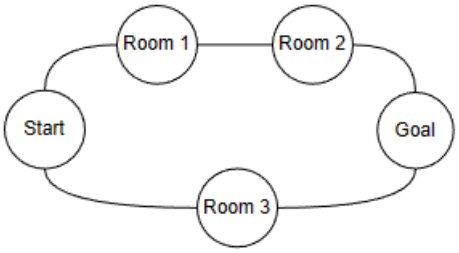

Figura 3.5: Exemplo de geração linear

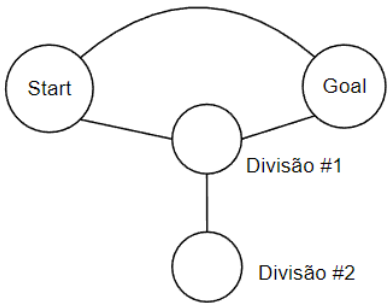

Figura 3.6: Exemplo de geração que quebra a linearidade

Contudo é necessário resolver um problema inerente a este método e que consiste em não existir qualquer relação entre a topologia do grafo e o mapa do jogo. Isto significa que é necessário abstrair e arranjar uma forma de converter o grafo para divisões concretas no mapa do jogo. Para resolver o problema adaptei a solução que Joris Dormans criou para o seu jogo: criei uma grelha de 4x3 onde ponho aleatoriamente no mapa, as divisões geradas do grafo.

3.3.2 Geração de corredores

A geração da estrutura das masmorras só acaba quando todas as divisões estiverem ligadas de acordo com a topologia. Tentei várias abordagens para gerar os corredores que ligam as diferentes divisões mas não obtive os resultados que pretendia, dado que parte dos corredores se cruzavam. Assim, optei por utilizar o algoritmo A*; este é um algoritmo de procura bastante conhecido e utilizado, que tem o propósito de encontrar os caminhos mais rápidos entre duas localizações. Com este algoritmo começa-se por criar o espaço do mapa, isto é, indica-se quais são os tiles que podem ser utilizados, e ao fornecer o ponto de partida e o de chegada, o A* irá encontrar o caminho mais rápido e dar uma lista com todos os pontos percorridos. De seguida, basta apenas colocar os tiles nas coordenadas dadas pelo algoritmo para desenhar os corredores.

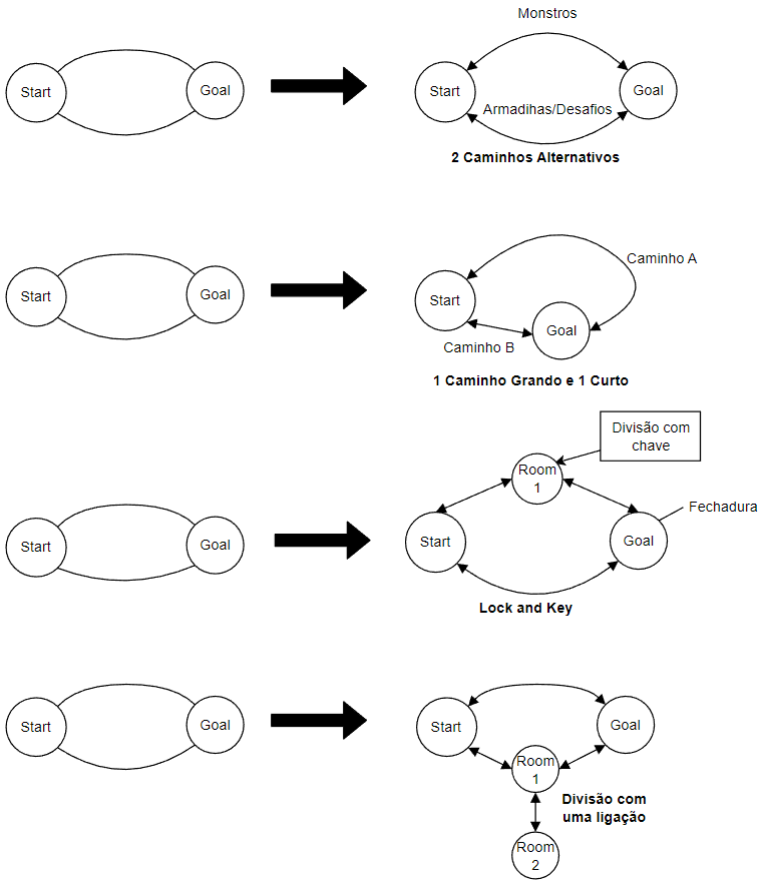

Figura 3.7: Padrões de Desenho

3.3.3 Geração dos objetos e entidades

Para completar a geração do cenário falta a etapa da geração dos objetos e entidades. Quando cada divisão é gerada, o algoritmo identifica o nome do nó para ativar e a correspondente função que gerará os objetos e/ou entidades. Em baixo estão identificados os nomes de cada nó e as respetivas funções.

Empty Room

A função do Empty Room é a de criar uma sala que, aleatoriamente, poderá ter uma poção ou um inimigo. Também é possível que tenha ambos ou que a divisão esteja mesmo vazia.

Monster House

A função do Monster House simplesmente gera entre 3 a 6 inimigos dentro da divisão.

Treasure Room

É gerado, pela função do Treasure Room, entre 1 a 5 baús do tesouro e 0 a 5 alçapões, que são colocados aleatoriamente na divisão.

Puzzle Room

A função do Puzzle Room é gerar o puzzle para a divisão.

Lock and Key

A função Lock and Key aceita as coordenadas do ponto a bloquear e o index dessa mesma divisão; é gerado um obstáculo nas coordenada do ponto, e é depois gerada uma chave numa divisão aleatória, embora sujeita a uma condição que previne a chave de ser gerada na divisão do obstáculo.

Trap Room

Existem duas gerações diferentes que podem acontecer com esta função. A primeira é a colocação de várias filas em paralelo com espaço entre elas e de um chão que arde intermitentemente. A segunda é a combinação desse chão com lançadores de setas que irão ficar no espaço referido do chão especial. Esta função tem em conta o formato (vertical ou horizontal) da divisão, isto é, se a divisão é horizontal ou vertical.

3.3.4 Exemplo

As figuras 3.8, 3.9, 3.10 e 3.11 representam dois exemplos de geração do algoritmo de grafos e a suas respetivas topologias. Pode observar-se nas figuras 3.9 e 3.11 a característica principal do algoritmo de grafos cíclico, isto é, todas as divisões apresentam dois caminhos para divisões diferentes. Embora a disposição das divisões seja diferente da apresentada nas topologias, todas as ligações correspondem ao que está representado nas figuras 3.8 e 3.10.

Estes exemplos apresentam também as mecânicas mencionadas tais como o lock and key, o puzzle, os inimigos, a trap room e o tesouro.

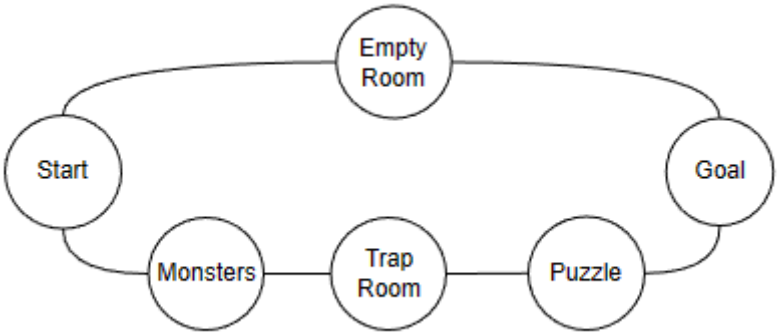

Figura 3.8: Topologia da masmorra da figura 3.9

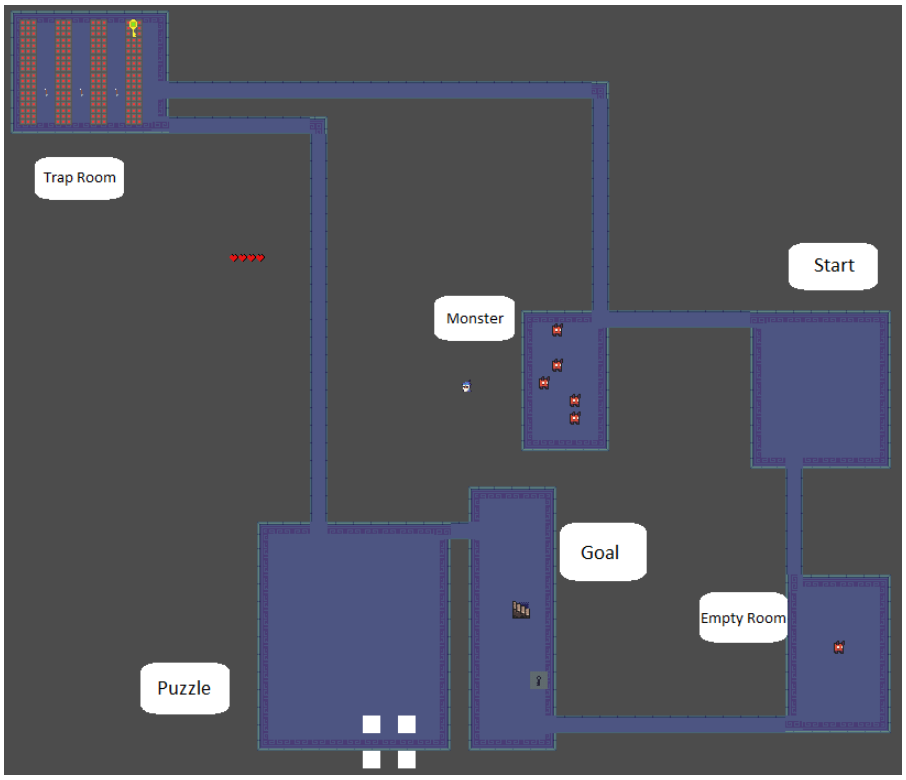

Figura 3.9: Primeiro exemplo de geração do algoritmo de grafos

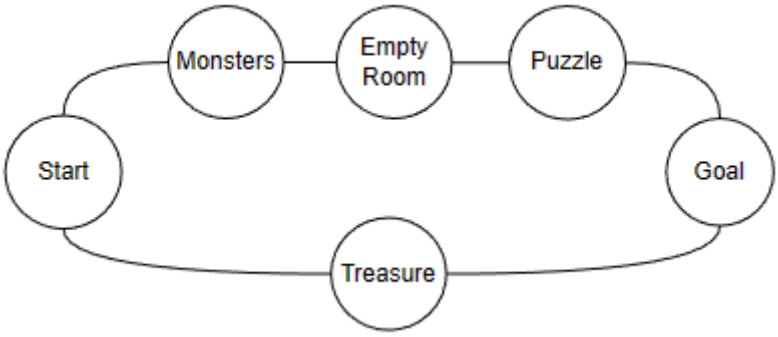

Figura 3.10: Topologia da masmorra da figura 3.11

3.4 Implementação dos Outros Algoritmos

Para analisar melhor o algoritmo de grafos cíclico, desenvolvi três algoritmos diferentes, o Walker, o Branching Trees e o algoritmo baseado no motor de física do game engine. Estes algoritmos são fundamentalmente diferentes uns dos outros; ao implementá-los foi possível comparar os diferentes resultados e as respetivas vantagens e desvantagens.

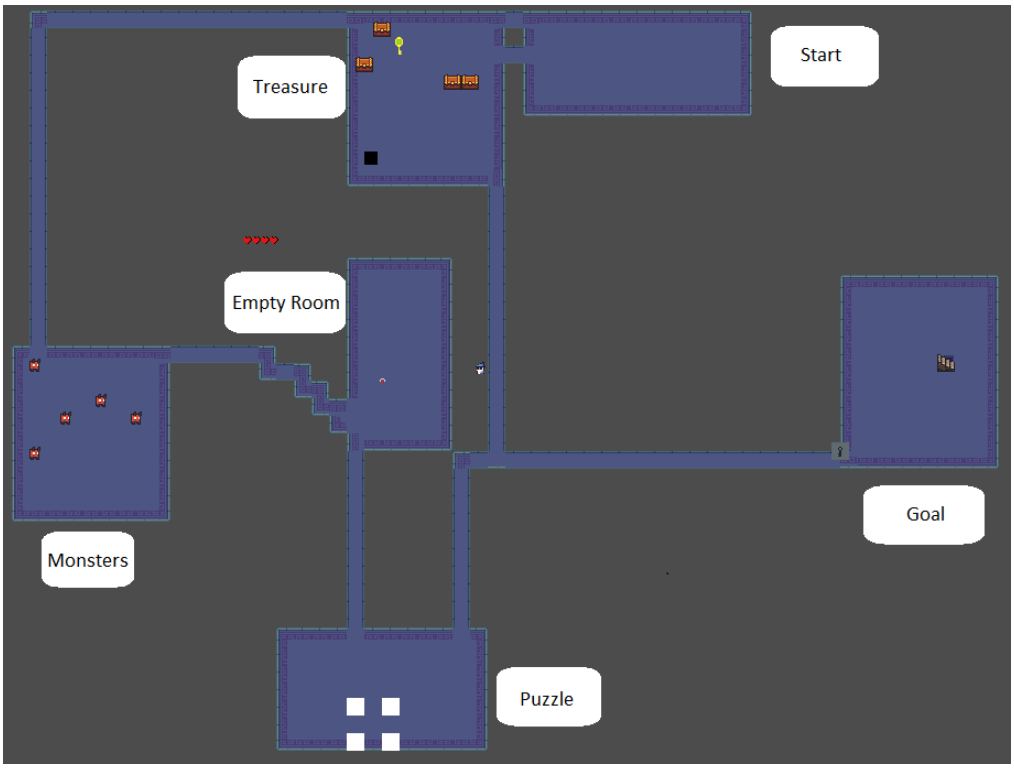

Figura 3.11: Segundo exemplo de geração do algoritmo de grafos

3.4.1 Walker

O Walker é um método de geração procedimental baseado em agentes, isto é, é criado uma entidade que irá andar pelo mapa, cumprindo as regras ou instruções definidas pelo developer para gerar o conteúdo, neste caso, a masmorra do jogo.

Esta implementação do Walker recorre muito à aleatoriedade do gerador de números pseudo-aleatórios para efetuar, ou não, determinadas ações. Funciona da seguinte forma:

- Em primeiro lugar é criado um agente numa localização inicial aleatória.
- O agente começa a deslocar-se em reta pelo mapa sendo a sua direção inicial aleatória, a cada 7 passos, o agente mudará de direção. O agente também mudará de direção caso se aproxime de um caminho por onde já passou ou de uma divisão já criada.
- Caso mude de direção, o agente terá a possibilidade de gerar uma divisão no local onde se encontra. A probabilidade de gerar a divisão é de 50% e o seu tamanho é aleatório.
- E assim sucessivamente até esgotar o número de passos que o agente pode dar (o número de passos é pré-definido).

- No final são colocados os objetos no jogo, isto é, os pontos de interesse, os inimigos e itens de acordo com a localização das divisões.

Este algoritmo pode apresentar resultados de geração muito diferentes uns dos outros, dependendo da definição de vários parâmetros que se podem alterar, tais como, a quantidade de passos que o agente pode dar, de que forma pode andar (por exemplo se muda de direção a cada 10 passos), o tamanho das divisões que deve gerar. Nas figuras 3.12 e 3.13 demonstra-se como a alteração desses parâmetros pode modificar os resultados gerados por este algoritmo. Na figura 3.13 e face à figura 3.12 foram alterados o número de passos do Walker, o número de passos consecutivos até virar de direção e uma probabilidade aleatória para gerar ou não a divisão.

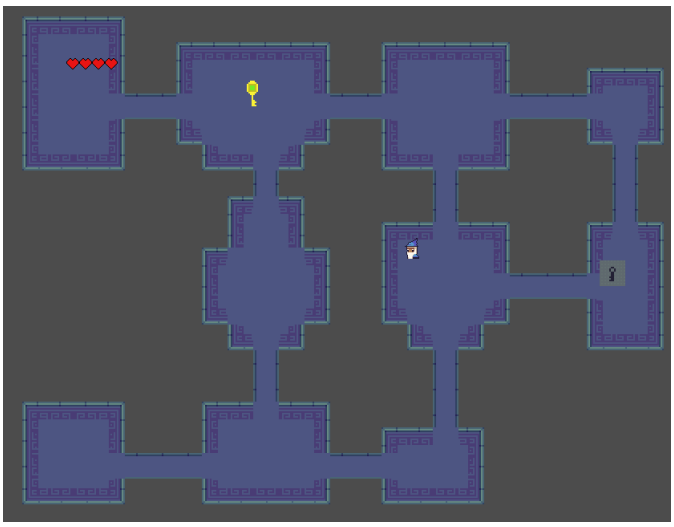

Figura 3.12: Exemplo de geração do Walker

Figura 3.13: Exemplo de geração do Walker com diferentes parâmetros.

3.4.2 Branching Trees

O algoritmo utilizado para implementar este método foi o "Simple Room Replacement". Este algoritmo, representado na figura 3.14, procede da seguinte forma:

- Em primeiro lugar é gerada uma divisão numa posição aleatória e com um tamanho aleatório. Como esta é a primeira divisão gerada, é colocada no mapa.
- O segundo passo é o de gerar uma outra divisão (que não é colocada no mapa); caso a posição da divisão se sobreponha a divisões já existentes, então é apagada e é gerada uma nova divisão. Quando se gerar uma divisão que não se sobreponha a nenhuma existente, então essa divisão é colocada no mapa.
- E assim sucessivamente até todas as divisões definidas estiverem criadas no mapa.
- Após as divisões estarem criadas é necessário conectá-las e para isso é usado o algoritmo de Prim.
- Finalmente são colocados os objetos e entidades no mapa.

É importante realçar que, para o algoritmo saber quantas divisões deve gerar, são criados os tipos de divisões existentes no nível e guardados numa lista, e quando essas divisões são colocadas de novo no mapa, é guardada a sua localização, para posteriormente, serem implementadas as funcionalidades respetivas das divisões.

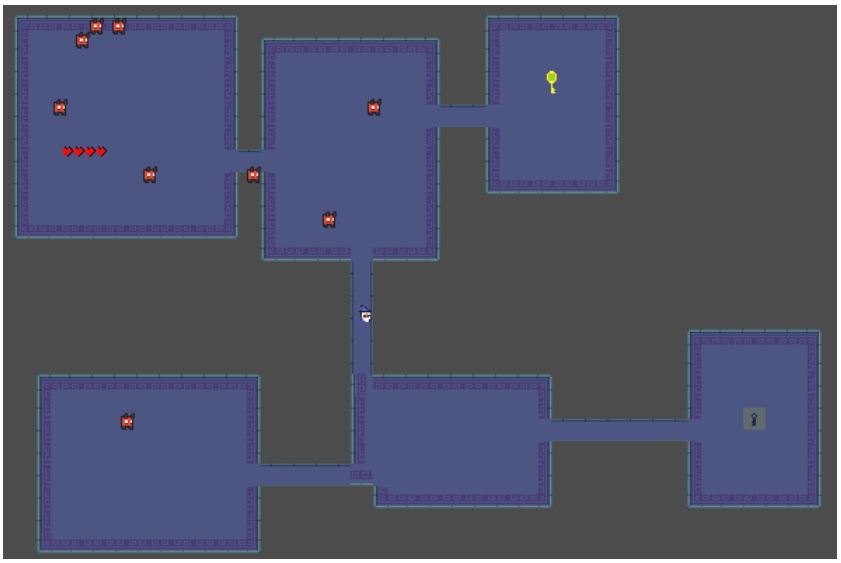

Figura 3.14: Exemplo de geração do Branching Trees

3.4.3 Algoritmo baseado no motor de física

Este método utiliza o motor de física do Godot de uma forma muito interessante:

- O algoritmo começa por gerar muitas divisões no mapa, com dimensões aleatórias, todas no mesmo sítio estando sobrepostas umas ás outras. Estas divisões são criadas com propriedades físicas para estarem sujeitas ao motor de física presente no game engine.
- O segundo passo é esperar que as divisões deixem de ficar sobrepostas. Como possuem propriedades físicas, o motor de física do Godot irá separar automaticamente essas divisões.
- Após as divisões estarem separadas, então o algoritmo irá apagar metade dessas divisões aleatoriamente; o resultado será que as maioria das divisões terá espaço entre si.
- De seguida, é utilizado o algoritmo de Prim que irá ligar as divisões no mapa.
- Por último, são colocados os objetos e entidades no mapa de forma aleatória, incluindo o "Lock and Key".

Podemos ver na figura 3.15 o resultado final da geração. Este algoritmo gerou uma estrutura bastante interessante, embora um pouco linear. De notar que é possível influenciar o resultado da geração da masmorra, para ter uma estrutura mais vertical (figura 3.16) ou horizontal (figura 3.17).

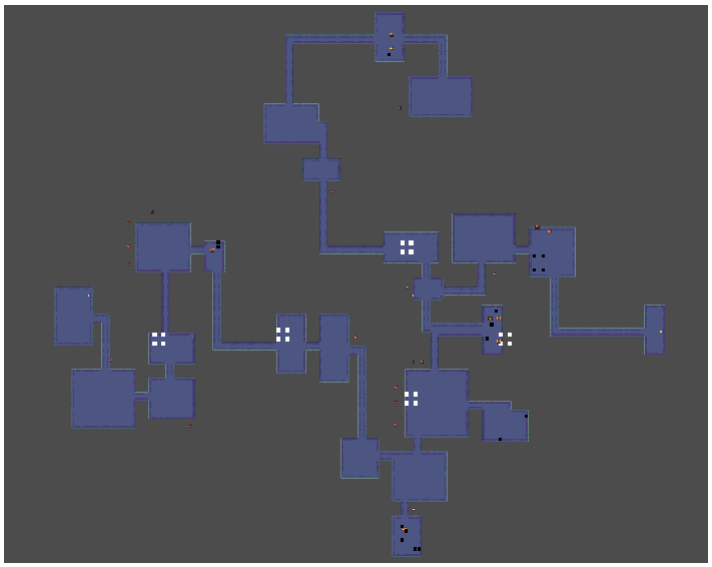

Figura 3.15: Exemplo de geração do algoritmo baseado em física

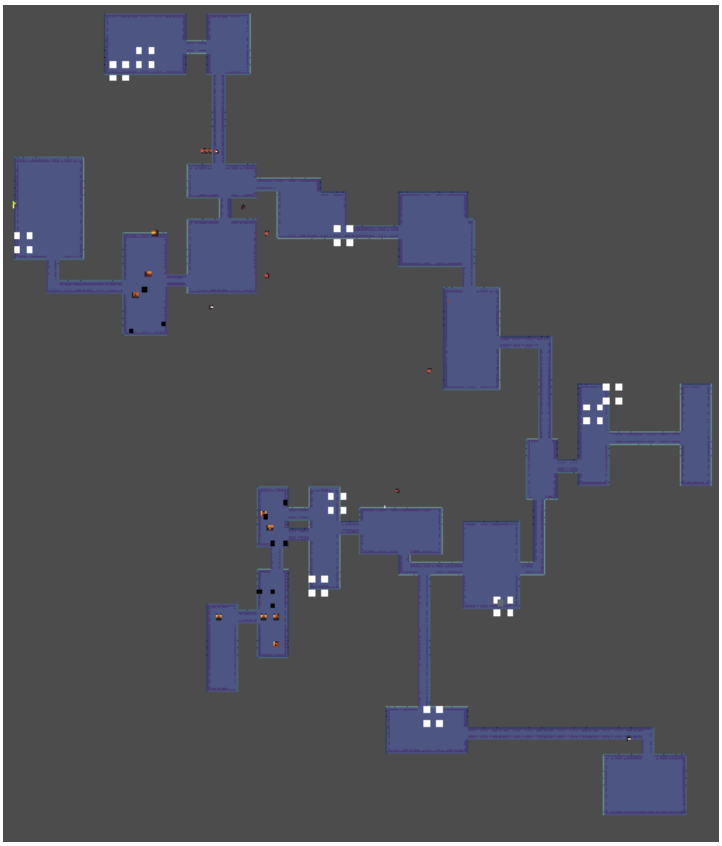

Figura 3.16: Geração do algoritmo com influência vertical

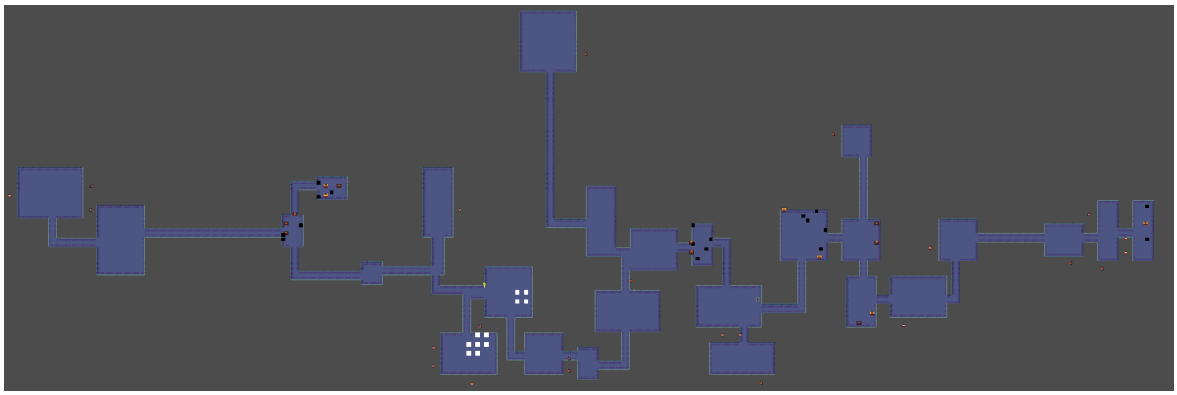

Figura 3.17: Geração do algoritmo com influência horizontal

3.5 Dificuldades no desenvolvimento

O desenvolvimento de um jogo é, por si só, um projeto bastante desafiante e senti algumas dificuldades, não só na implementação da estrutura do mesmo mas também pelo tipo de programação a desenvolver pois, por exemplo, desconhecia as funções e propriedades específicas do Godot Engine. Quase todas as dificuldades foram ultrapassadas (com exceção de duas); no entanto, estes obstáculos obrigaram-me a fazer alguns ajustamentos a certos aspetos do jogo devido aos atrasos que os mesmos causaram. Como exemplos

desta situação foram o movimento do jogador e dos inimigos, que originalmente era para ser baseado na grelha do jogo ao invés de ser movimento livre e o facto de o jogo não ser baseado em turnos, como tinha originalmente pensado.

A primeira dificuldade que não consegui ultrapassar está relacionada com as coordenadas dos objetos relativamente à posição do mapa; isto é especialmente visível na implementação do algoritmo Walker. Por vezes os objetos são gerados fora do mapa, não sendo acessíveis ao jogador, podendo ficar, por exemplo, parte do puzzle fora da divisão.

A segunda dificuldade que não consegui superar foi a geração dos corredores para ligar as divisões, pois muitas vezes, na geração desses corredores, estes cruzavam-se com outros. Para o mapa manter uma estrutura igual à da topologia do grafo era necessário os caminhos não se cruzarem. Tentei várias alternativas, umas criadas por mim e outras baseadas em tutoriais que vi na internet, mas por uma razão ou por outra, nunca consegui gerar os corredores sem que os mesmos se cruzassem. Isto também me impossibilitou de implementar determinados padrões de design de geração, especialmente aqueles que dependem da integridade dos corredores, por não conseguir controlar bem este aspeto da geração. Esta questão atrasou significativamente o trabalho, para além de que a incapacidade de resolver este problema também me impossibilitou de implementar algumas regras de transformação que tinha planeado fazer.

3.6 Conclusão

Apesar das dificuldades sentidas, o jogo foi desenvolvido com sucesso e dentro do que foi planeado. Foram implementados os quatro algoritmos de geração de cenários, com os respetivos algoritmos de procura para a geração dos corredores (A* e Prim), os atributos das divisões e a mecânica de jogo. Estas implementações funcionam de forma muito distinta, permitindo mostrar as respetivas capacidades e limitações de cada uma delas.

Capítulo 4

Resultados

Desenvolvidos os 4 tipos de algoritmos em estudo, de grafos cíclico, Walker, Branching Trees, e baseado no motor de física do game engine, impõe-se aplicá-los e analisar os resultados da geração, a influência dos parâmetros utilizados e quais os pontos fortes e fracos dos resultados da geração da masmorra. Para testar os algoritmos gerei 50 masmorras com cada um deles e apurei os resultados, por tipo de algoritmo, conforme os pontos 4.1 a 4.4 deste capítulo.

Efetuei também um pequeno inquérito para apurar os resultados do ponto de vista do jogador. Selecionei 6 masmorras criadas por cada algoritmo (24 no total) e os resultados desta avaliação estão refletidos no ponto 4.5.

4.1 Algoritmo de grafos cíclico

O algoritmo de grafos cíclicos é capaz de gerar masmorras interessantes, mesmo tendo em conta o problema constatado e não ultrapassado dos corredores cruzados. Contudo, o resultado da maioria das estruturas geradas, apresenta as divisões aleatoriamente espaçadas com os seus devidos corredores. Em relação a outros aspetos da geração, como a variedade de divisões, o algoritmo gera uma boa variedade de topologias diferentes embora, devido ao facto de alguns mapas apresentarem caminhos que se cruzam quando não era suposto, põem em causa, nesses casos, o propósito do planeamento da topologia da masmorra.

E ainda é importante referir que, por vezes, este algoritmo gera masmorras sem corredores ou com corredores parciais; estas ocorrências devem-se à implementação do algoritmo A* neste gerador; pode observar-se este comportamento na figura 4.1, onde apenas a parte inferior direita da masmorra tem corredores. Pode acontecer haver também divisões que se sobrepõem, como mostrado na figura 4.2, o que representa outro problema ao tornar irrelevante a mecânica do Lock and Key.

Relativamente aos becos sem saída, verifica-se que, de uma maneira geral, não ocorrem.

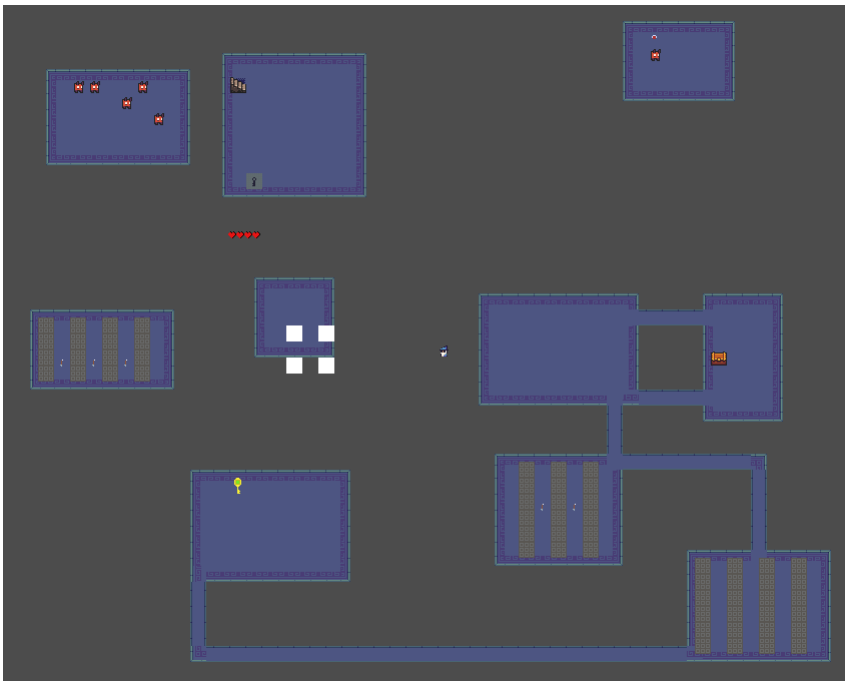

Figura 4.1: Masmorra parcialmente sem corredores

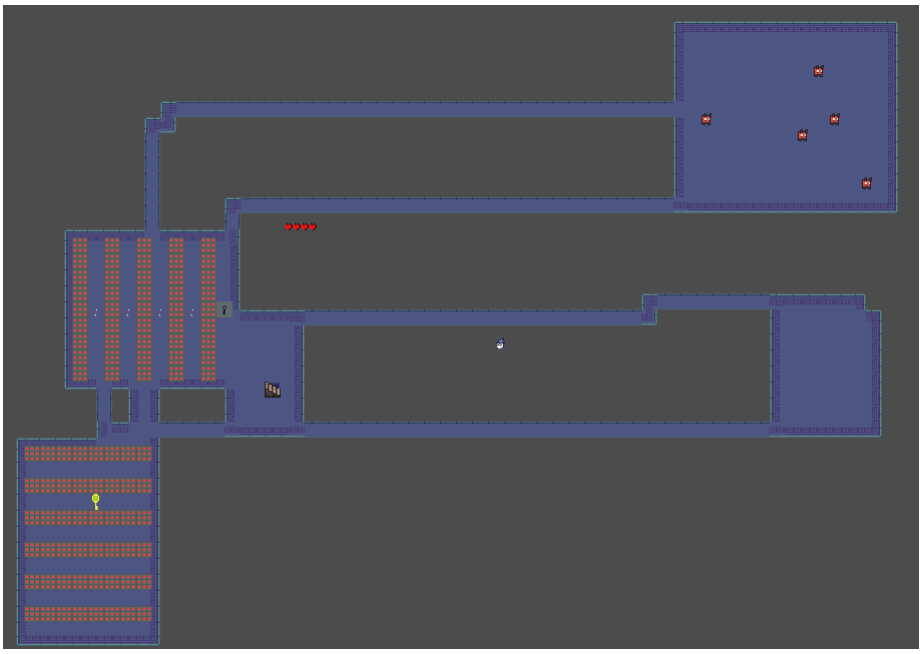

Figura 4.2: Masmorra com as divisões juntas

4.2 Walker

O algoritmo Walker gerou masmorras muito interessantes, com uma boa estrutura e distribuição de divisões e corredores. Verifiquei que é um algoritmo muito sensível aos parâmetros implementados, pois qualquer variação influencia significativamente o resul-

tado da geração da masmorra. É possível, como demonstrado nas figuras 3.12 e 3.13, que o resultado da geração de uma masmorra seja semelhante a uma grelha ou, caso se alterem alguns parâmetros, como o da quantidade de passos que o agente deve dar, existe a possibilidade de ser gerada ou não uma divisão e/ou a eventualidade de virar ou não de direção, originando um resultado da geração da masmorra completamente diferente do inicial, como se pode observar na figura 4.3. Este factor tanto funciona como uma vantagem como uma desvantagem pois é possível gerar masmorras completamente diferentes; no entanto, é necessário afinar bastante bem os parâmetros, sendo necessário testar a geração muitas vezes sempre que se estejam a experimentar novos parâmetros.

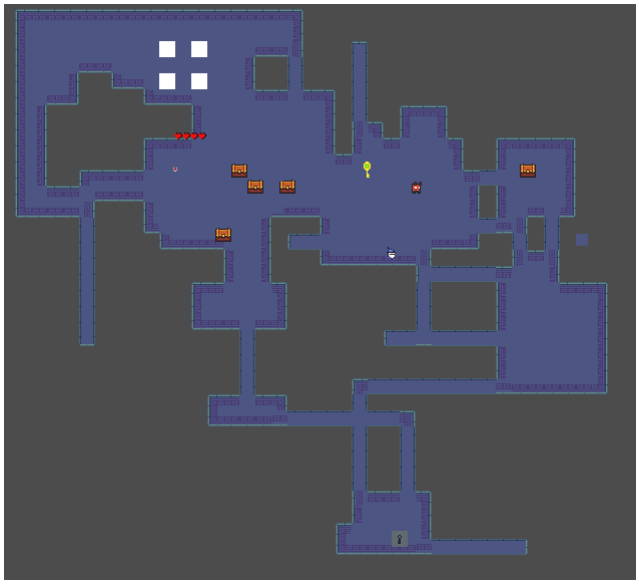

Figura 4.3: Geração Walker com parâmetros diferentes

Existe também outra vantagem: como o agente anda pelo mapa e coloca o chão da masmorra, não é necessário implementar um algoritmo de procura para gerar os corredores.

4.3 Branching Trees

O algoritmo Branching Trees com o "Simple Room Placement"sofre do mesmo problema que o algoritmo de grafos cíclico, pois os corredores das divisões intersetam-se. Podem ser geradas divisões coladas umas às outras, o que se deve ao autotiler. Como existem dois tiles de chão, um ao lado do outro, o autotiler liga esse chão um ao outro. Como podemos verificar na figura 4.4, as duas divisões centrais embora não estejam sobrepostas entre si, tornam essas duas divisões numa só.

Neste método, a geração das características técnicas das divisões da masmorra é genericamente mais limitada. Tal pode dever-se ao facto de que, atualmente, o algoritmo gera sempre masmorras com 10 divisões, e que a geração dessas características é aleatória.

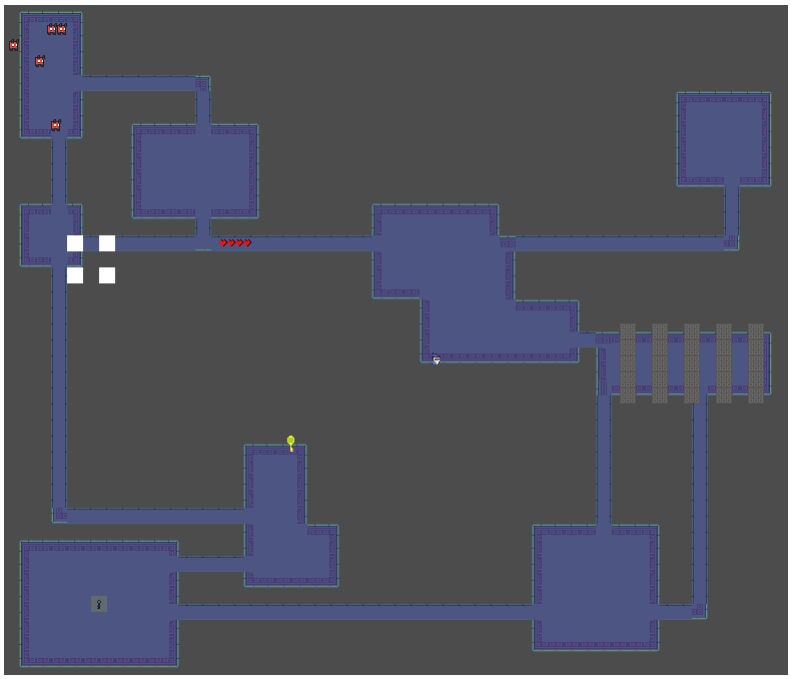

Figura 4.4: Branching Trees com divisões juntas

Experimentei construir um algoritmo que gerasse um número aleatório de divisões; no entanto os resultados foram semelhantes aos anteriores. Os resultados gerados por este método mostraram ser inferiores aos restantes algoritmos, devido à falta de planeamento do algoritmo, tendo uma excessiva aleatoriedade no processo de geração.

4.4 Algoritmo baseado no motor de física do game engine

Os resultados deste algoritmo foram surpreendentes pois este método apresenta estruturas da masmorra bem espaçadas e estruturalmente interessantes embora com becos sem saída que podem levar o jogador a fazer backtracking significativo. Devido ao facto de este algoritmo utilizar o algoritmo de Prim para ligar as divisões, não existem caminhos cruzados, como aqueles presentes tanto no algoritmo de grafos como no Branching trees. Verifica-se contudo que os cenários apresentam becos sem saída dependendo da localização inicial do jogador e dos objetivos no mapa.

4.5 Inquérito

No decorrer deste trabalho, resolvi efetuar um breve inquérito a um pequeno grupo de colegas, a quem mostrei o resultado da aplicação dos algoritmos que desenvolvi na criação de um jogo Roguelike, utilizando as 4 técnicas em estudo.

O objetivo foi de apurar qual a técnica que originava melhores resultados do ponto de vista do utilizador e qual o desempenho do algoritmo de grafos no conjunto dos cenários

apresentados e na construção de jogos desta natureza.

4.5.1 Caracterização do inquérito

O inquérito subdivide-se em duas partes: uma primeira onde, para além de se identificar o perfil genérico dos inquiridos (sexo, idade e formação académica) também são recolhidos dados mais específicos sobre o seu perfil enquanto jogadores, para determinação da sua capacidade na avaliação dos mapas de jogos que lhes foram apresentados (Com que frequência costumas jogar videojogos; quanto tempo jogas por semana, costumas jogar jogos de Roguelike?).

Na segunda parte pretende-se que cada participante avalie, individualmente, cada um dos 24 cenários criados (6 por algoritmo).

A avaliação é feita com base numa grelha de 5 parâmetros pré-definidos sendo cada um deles classificado de acordo com o interesse despertado, em Bom, Médio ou Fraco. Os parâmetros em questão são:

- Apreciação Geral: avaliar se globalmente o nível parece interessante.
- Estrutura e dimensão do mapa: avaliar a disposição, o layout e a dimensão do mapa.
- Corredores: avaliar se são suficientes e lógicos, e se no caso de se intercetarem se prejudicam a jogabilidade do cenário.
- Localização das divisões: avaliar se são em número adequado e bem distribuídas.
- Becos sem saída: existindo, se dificultam ou penalizam a experiência do jogador.

4.5.2 Participantes

Apresentei o inquérito a uma população de 8 universitários, licenciados, 7 do sexo masculino e 1 do sexo feminino e com idades compreendidas entre os 24 e 25 anos. Esta população apresenta experiência de jogo, sendo que todos eles jogam mais de 3 horas por semana. De referir que, no decorrer deste inquérito, não foi dado qualquer destaque às técnicas utilizadas na construção dos cenários, pretendendo-se que o enfoque fosse na imagem apresentada e na perspetiva do jogador.

4.5.3 Resultados

Os dados obtidos com base nas respostas atribuídas a cada imagem, foram analisados individualmente e também de forma agregada, sempre na perspetiva da técnica empregue, pelo que as tabelas foram construidas por tipo de algoritmo. Cada resposta foi contabilizada como 1 unidade, pelo que o total de respostas por cenário e por parâmetro é de 8.

No apuramento de resultados por metodologia (grupo de 6 cenários), o total de respostas por parâmetro é de 48.

Nas secções abaixo apresentam-se os resultados obtidos para cada metodologia de geração de cenários, apresentando-se os resultados, imagem em imagem.

Imagens originadas pelo algoritmo de Grafos

Imagens	G1			G2			G3			G4			G5			G6		
Avaliação / Parâmetros	Bom	Médio	Fraco	Bom	Médio	Fraco	Bom	Médio	Fraco	Bom	Médio	Fraco	Bom	Médio	Fraco	Bom	Médio	Fraco
Apreciação Geral	0	1	7	0	1	7	8	0	0	0	8	0	8	0	0	7	1	0
Estrutura e dimensão	0	1	7	0	0	8	8	0	0	1	7	0	8	0	0	8	0	0
Corredores	8	0	0	8	0	0	6	2	0	8	0	0	7	1	0	7	1	0
Localização das divisões	3	5	0	3	5	0	6	2	0	1	7	0	7	1	0	6	2	0
Becos sem saída	8	0	0	8	0	0	8	0	0	8	0	0	8	0	0	5	3	0

Tabela 4.1: Resultados da avaliação dos cenários construídos pelo algoritmo de Grafos

Os 6 cenários originados com esta metodologia são muito diferentes entre si, o que se reflete nas avaliações efetuadas pelos participantes deste inquérito, conforme se pode observar na tabela 4.1. Assim, a estrutura e dimensão dos cenários das imagens G1 e G2, foi considerada como fraca penalizando bastante a avaliação geral destes 2 mapas em relação aos 4 restantes, considerados, nesta matéria como Bons.

A localização dos corredores e divisões, assim como a inexistência de becos sem saída foram considerados fatores positivos nos 6 cenários, conforme se pode verificar na tabela 4.2 com valores globais (é atribuída avaliação de Bom a 91,7% dos corredores e 93,8% a becos sem saída).

Imagens	Global G			Distribuição (%)		
Avaliação / Parâmetros	Bom	Médio	Fraco	Bom	Médio	Fraco
Apreciação Geral	23	11	14	47,9%	22,9%	29,2%
Estrutura e dimensão	25	8	15	52,1%	16,7%	31,2%
Corredores	44	4	0	91,7%	8,3%	0,0%
Localização das divisões	26	22	0	54,2%	45,8%	0,0%
Becos sem saída	45	3	0	93,8%	6,2%	0,0%

Tabela 4.2: Resultado global da avaliação dos cenários construídos pelo algoritmo de Grafos

Em síntese, esta técnica gera automaticamente alguns cenários bastante bons (ex: imagens G3 e G5); contudo, algumas das gerações têm aspetos considerados fracos, como as

imagens G1 e G2, pelo que a ser utilizada na criação de cenários, tem que haver uma restrição na técnica para não apresentar cenários demasiado simples.

Imagens originadas pelos motor de física do game engine

Imagens	MF1			MF2			MF3			MF4			MF5			MF6		
Avaliação / Parâmetros	Bom	Médio	Fraco	Bom	Médio	Fraco	Bom	Médio	Fraco	Bom	Médio	Fraco	Bom	Médio	Fraco	Bom	Médio	Fraco
Apreciação Geral	7	1	0	7	1	0	7	1	0	8	0	0	7	1	0	7	1	0
Estrutura e dimensão	8	0	0	8	0	0	8	0	0	8	0	0	6	2	0	8	0	0
Corredores	8	0	0	8	0	0	8	0	0	8	0	0	8	0	0	8	0	0
Localização das divisões	7	1	0	7	1	0	7	1	0	7	1	0	7	1	0	7	1	0
Becos sem saída	6	1	1	5	2	1	5	1	2	5	2	1	7	1	0	5	1	2

Tabela 4.3: Resultado da avaliação dos cenários construídos pelo algoritmo baseado no motor de física

Os 6 cenários originados pelo algoritmo que utiliza o motor de física do game engine (tabelas 4.3 e 4.4), apesar de serem diferentes entre si, recolheram avaliações muito semelhantes, tendo 90% sido considerados bons, essencialmente devido à estrutura e dimensão dos mapas, aos corredores e à dimensão das divisões. No entanto estes cenários já apresentam becos sem saída, tendo alguns dos inquiridos considerado que estes poderiam afetar negativamente o desenvolvimento do jogo (15% consideram que era um aspeto fraco destes cenários).

Imagens	Global MF			Distribuição (%)		
Avaliação / Parâmetros	Bom	Médio	Fraco	Bom	Médio	Fraco
Apreciação Geral	43	5	0	89,6%	10,4%	0,0%
Estrutura e dimensão	46	2	0	95,8%	4,2%	0,0%
Corredores	48	0	0	100,0%	0,0%	0,0%
Localização das divisões	42	6	0	87,5%	12,5%	0,0%
Becos sem saída	33	8	7	68,8%	16,7%	14,5%

Tabela 4.4: Resultado global da avaliação dos cenários construídos pelo algoritmo de Grafos

Imagens originadas pelo algoritmo Walker

À exceção da imagem W5, conforme se pode observar na tabela 4.5, a avaliação dos restantes cenários gerados pelo algoritmo Walker é muito similar e genericamente bons. A imagem W5, que apresenta uma estrutura desorganizada e fraca disposição dos corredores no cenário, vem penalizar a avaliação global deste conjunto de cenários, como se pode verificar na tabela 4.6, onde a avaliação de fraco é dada apenas por esta imagem.

Imagens	W1			W2			W3			W4			W5			W6		
Avaliação / Parâmetros	Bom	Médio	Fraco	Bom	Médio	Fraco	Bom	Médio	Fraco	Bom	Médio	Fraco	Bom	Médio	Fraco	Bom	Médio	Fraco
Apreciação Geral	6	2	0	7	1	0	6	2	0	6	2	0	3	4	1	6	2	0
Estrutura e dimensão	6	2	0	7	1	0	5	3	0	7	1	0	1	5	2	4	4	0
Corredores	7	1	0	8	0	0	7	1	0	8	0	0	3	4	1	8	0	0
Localização das divisões	8	0	0	7	1	0	8	0	0	7	1	0	6	2	0	8	0	0
Becos sem saída	7	1	0	8	0	0	8	0	0	8	0	0	7	1	0	8	0	0

Tabela 4.5: Resultado da avaliação dos cenários construídos pelo Walker

Imagens	Global W			Distribuição (%)		
Avaliação / Parâmetros	Bom	Médio	Fraco	Bom	Médio	Fraco
Apreciação Geral	34	13	1	70,8%	27,1%	2,1%
Estrutura e dimensão	30	16	2	62,5%	33,3%	4,2%
Corredores	41	6	1	85,4%	12,5%	2,1%
Localização das divisões	44	4	0	91,7%	8,3%	0,0%
Becos sem saída	46	2	0	95,8%	4,2%	0,0%

Tabela 4.6: Resultado global da avaliação dos cenários construídos pelo Walker

Imagens originadas pelo algoritmo Branching Trees

Este algoritmo gerou o único mapa com avaliação de Bom em todos os parâmetros, a imagem BT6. As restantes imagens apresentavam avaliações muito irregulares pois os cenários são muito diferentes entre si. Contudo, verifica-se que, com exceção da imagem BT2, os corredores estão bem integrados e que os becos sem saída são inexistentes ou sem impacto negativo na qualidade do jogo. A localização das divisões e a estrutura e dimensão destes cenários de jogo, são considerados médios ou fracos, conforme se pode observar na tabela 4.7. Globalmente, como apresentado na tabela 4.8, os resultados deste algoritmo são inferiores aos restantes algoritmos, á exceção do mapa BT6 que apresentou um resultado perfeito.

Imagens	BT1			BT2			BT3			BT4			BT5			BT6		
Avaliação / Parâmetros	Bom	Médio	Fraco	Bom	Médio	Fraco	Bom	Médio	Fraco	Bom	Médio	Fraco	Bom	Médio	Fraco	Bom	Médio	Fraco
Apreciação Geral	0	8	0	0	3	5	3	5	0	7	1	0	7	1	0	8	0	0
Estrutura e dimensão	0	5	3	0	7	1	3	4	1	6	2	0	6	2	0	8	0	0
Corredores	8	0	0	0	3	5	7	1	0	7	1	0	8	0	0	8	0	0
Localização das divisões	1	6	1	2	5	1	5	3	0	7	1	0	6	2	0	8	0	0
Becos sem saída	8	0	0	8	0	0	2	6	0	8	0	0	8	0	0	8	0	0

Tabela 4.7: Resultado da avaliação dos cenários construídos pelo Braching Trees

Imagens / Avaliação / Parâmetros	Global BT			Distribuição (%)		
	Bom	Médio	Fraco	Bom	Médio	Fraco
Apreciação Geral	25	18	5	52,1%	37,5%	10,4%
Estrutura e dimensão	23	20	5	47,9%	41,7%	10,4%
Corredores	38	5	5	79,2%	10,4%	10,4%
Localização das divisões	29	17	2	60,4%	35,4%	4,2%
Becos sem saída	42	6	0	87,5%	12,5%	0,0%

Tabela 4.8: Resultado global da avaliação dos cenários construídos pelo Branching Trees

4.6 Conclusão

Todos os algoritmos apresentaram resultados satisfatórios na geração de masmorras. No entanto, todos eles, incluindo o algoritmo baseado no motor de física e no Walker que apresentaram os melhores resultados, possuem e demonstram as suas particularidades.

Em relação ao inquérito, este evidencia que o algoritmo baseado no motor de física do game engine e o algoritmo Walker originaram, globalmente, os melhores resultados, como se pode observar na tabela 4.9.

Algoritmos / Parâmetros	Grafos	M. Física	Walker	B. Trees
Apreciação Geral	9	43	33	20
Estrutura e dimensão	10	46	28	18
Corredores	44	48	40	33
Localização das divisões	26	42	44	27
Becos sem saída	45	26	46	42
Global	134	205	191	140

Tabela 4.9: Resultado global da avaliação dos algoritmos, para ponderação 1 da avaliação "Bom", 0 de "Médio" e -1 de "Fraco"

Verifica-se ainda que os jogadores são mais sensíveis, no momento de avaliação, à qualidade e estrutura e dimensão dos mapas em detrimento da localização das divisões ou da inexistência dos corredores cruzados. Os becos sem saída não mostraram ser um problema desde que os jogadores não tenham de recuar muito no nível do jogo, fator que pesou negativamente no algoritmo do motor de física, com resultados baixos neste parâmetro.

Relativamente ao algoritmo de grafos, além das imagens G1 e G2, cujos resultados penalizaram bastante a avaliação global deste algoritmo, é de notar que a localização das divisões poderia ainda assim ser melhorada. Este algoritmo é eficaz a gerar masmorras sem becos sem saída, embora eles ocorram pontualmente por erro na geração dos corredores; contudo, na avaliação das imagens, estas ocorrências não foram valorizadas negativamente. Por último, relativamente ao algoritmo de Braching Trees, apesar de ter originado o cenário com melhor avaliação, não agradou particularmente, devido essencialmente à aleatoriedade da geração.

Capítulo 5

Conclusão

Neste capítulo apresento as minhas conclusões sobre o algoritmo de grafos cíclico, bem como algumas ideias para o trabalho que futuramente poderá ser desenvolvido.

5.1 Conclusão

Esta dissertação visa analisar se o algoritmo de grafos cíclico é um método útil e eficaz na criação de cenários de jogo. Para efetuar este estudo criei um jogo 2D roguelike, utilizando um algoritmo de grafos cíclico, com o objetivo de comparar o processo de construção do algoritmo de grafos cíclico e os resultados obtidos, com os decorrentes da utilização de outros 3 algoritmos na construção de cenários de jogos. Os algoritmos utilizados para a comparação foram o Walker, o Branching Trees e um algoritmo baseado no motor de física do game engine.

Embora tenha tido algumas dificuldades no desenvolvimento do jogo, nomeadamente com a geração dos corredores, prejudicando a geração dos cenários, considero que os objetivos foram cumpridos e permitem a retirada de algumas conclusões importantes sobre a utilização destes algoritmos na geração de jogos 2D.

Concluí que este algoritmo é uma ferramenta viável na geração de masmorras em jogos 2D. Tem capacidade de estruturar eficazmente uma masmorra, implementando mecânicas que permitem obter resultados excelentes na geração de níveis de jogo. Quando o algoritmo de grafos cíclicos é bem desenvolvido, tem a capacidade de gerar automaticamente cenários muito realistas, ao ponto de os níveis parecerem ter sido desenvolvidos por level designers. Tal também se deve à sua característica de implementação de padrões de desenho. Estes resultados são de tal forma bons que os críticos elogiaram os níveis gerados no jogo "Unexplored"pois pareciam ter sido criados por técnicos.

No entanto esta metodologia, possui aspetos que dificultam a sua aplicação, como a necessidade de desenvolver a abstração da topologia do grafo; isto é especialmente verdade se o requisito passar por obter um nível de realismo elevado nos cenários.

A implementação do algoritmo de grafos cíclico, apresenta logo na sua concepção,

diferenças bastante significativas em relação aos outros métodos que utilizei.

A primeira que devo assinalar pela sua importância na geração de cenários é que a topologia do grafo não possui qualquer relação com o mapa do jogo sendo necessário criar essa camada de abstração para se conseguir gerar conteúdo. Dependendo da quantidade de funcionalidades ou das regras de geração que se pretendam aplicar, este passo poderá tornar-se bastante complexo. Para exemplificar o nível de complexidade, podemos referir novamente o caso de Joris Dormans que necessitou de criar um programa (à parte do jogo) para o ajudar a programar e a corrigir o seu algoritmo de grafos cíclico. De referir que a camada de abstração que criou para o referido jogo, não conseguia garantir perfeitamente que todos os níveis pudessem ser concluídos, pois havia casos em que a chave ou alavanca que possibilitava a passagem para o nível seguinte, ficava atrás de uma parede ou inacessível por outros motivos.

Outra desvantagem verifica-se quando, comparado com o algoritmo "Walker"; o algoritmo de grafos cíclico necessita da implementação adicional de um algoritmo de procura para gerar os corredores que ligam as divisões.

Por último, em comparação com o algoritmo baseado no motor de física do game engine, o algoritmo principal deste estudo precisa de significativamente mais horas trabalho, para apresentar resultados semelhantes. Nomeadamente, necessita de uma melhor camada de abstração entre a topologia do grafo e o mapa do jogo e de uma melhor implementação do algoritmo de procura.

Em relação à sondagem que realizei aos meus colegas, esta permitiu constatar que as masmorras que mais apelavam aos inquiridos eram as masmorras com uma maior dimensão, como as representadas pelo algoritmo baseado no motor de física do game engine. Observei também que preferiam as masmorras que parecessem ter tido algum planeamento ou estrutura, isto é, sempre que era evidenciada demasiada aleatoriedade na localização das divisões, os resultados eram inferiores. Logo, é possível concluir que caso os desafios mencionados anteriormente fossem ultrapassados, então o algoritmo de grafos cíclico poderia ser o melhor método para gerar as masmorras.

No entanto, é minha opinião, que apenas se deverá optar por este algoritmo, no caso de estar associada ao desenvolvimento do jogo ou do projeto, uma equipa de uma empresa ou de um estúdio que possa trabalhar no algoritmo, pois a tarefa de desenvolvimento do mesmo é mais complexa que o algoritmo aqui analisado, requerendo mais tempo de desenvolvimento, especialmente se for necessário um nível alto de realismo por parte do algoritmo de geração.

As outras implementações, especialmente as decorrentes dos algoritmos Walker e do algoritmo baseado no motor de física do game engine, vieram demonstrar que, sendo estes algoritmos mais simples, conseguem atingir resultados bons num menor espaço de tempo e com menos trabalho, fazendo com que estes sejam ideais para serem utilizados por equipas pequenas ou singulares, pois importa relembrar que o desenvolvimento do jogo é

um processo complexo que requer talento multi-disciplinar para criar tanto as mecânicas do jogo, como os gráficos, a música e a história.

Resumindo, concluo que este algoritmo é uma boa alternativa a outros graças à capacidade de planeamento da estrutura do cenário e aos padrões de desenho que pode implementar, embora apenas sugira que seja uma equipa com experiência em geração procedimental de conteúdo a adotar este método e não apenas uma ou duas pessoas.

Por último, este algoritmo é mais indicado para ser usado em jogos de exploração, roguelike, hack-and-slash, de sobrevivência ou de simulação.

5.2 Análise de Contribuições

O objetivo desta dissertação foi a implementação de um algoritmo de grafos cíclico, na criação de jogo 2D Roguelike, e comparar esse algoritmo com outros 3 para analisar não só o processo de construção mas as vantagens e desvantagens de cada um dos algoritmos e os desafios encontrados durante o respetivo desenvolvimento.

Assim, considero que as principais contribuições deste trabalho são a identificação da viabilidade do algoritmo de grafos cíclico para a geração de cenários, as capacidades e desafios inerentes ao desenvolvimento deste algoritmo e também como este se compara com outros algoritmos de geração de cenários permitindo um melhor entendimento das necessidades referentes aos projetos que se queiram desenvolver. O código deste trabalho está disponível no github[17].

5.3 Trabalho futuro

Como referido anteriormente, a especificidade do algoritmo de grafos cíclico passa pela abstração da topologia do grafo. Como tal, seria interessante que fosse desenvolvido um estudo sobre este tema, com o objetivo de serem criadas formas eficazes de se relacionar a topologia do grafo com o mapa do jogo.

Existem também outros dois aspetos que seriam interessantes de investigar: a criação de um padrão para o desenvolvimento das regras de geração de grafos, pois, como foi demonstrado, pode ser um processo bastante complexo, e investigar de que forma poderia um algoritmo analisar as masmorras geradas por algoritmos GPC de forma automática, verificando se cumpre os requisitos previamente definidos e quantificando a sua qualidade.

Bibliografia

[1] TileStats. (2023, 07) Cellular automata tutorial - the basics. [Online]. Available: https://www.youtube.com/watch?v=2nE0z_9-fCY

[2] S. Software. (2022, 11) Voronoidiagrammer. [Online]. Available: https://www.safe.com/transformers/voronoi-diagrammer/

[3] T. G. Xingze Tian. (2022, 12) A survey of smooth vector graphics: Recent advances in representation, creation, rasterization and image vectorization. [Online]. Available: https://www.researchgate.net/figure/Three-different-types-of-distortion-applied-to-a-Perlin-noise-field-top-left_fig6_332164251

[4] scratchapixel. (2022, 12) Perlin noise: Part 2. [Online]. Available: https://www.scratchapixel.com/lessons/procedural-generation-virtual-worlds/perlin-noise-part-2/perlin-noise-terrain-mesh.html

[5] T. Short and T. Adams, *Cyclic Generation*, 1st ed. A K Peters/CRC Press, 2017, pp. 83–95.

[6] A. Wiltshire. (2017, 03) How unexplored generates great roguelike dungeons. [Online]. Available: https://www.rockpapershotgun.com/how-unexplored-generates-great-roguelike-dungeons

[7] Unity. (2022, 10) Manual: Order of execution. [Online]. Available: https://docs.unity3d.com/Manual/ExecutionOrder.html

[8] Wikipedia contributors, "Roguelike," 07 2022. [Online]. Available: https://en.wikipedia.org/wiki/Roguelike

[9] R. Parreira. (2021, 06) Há 2,7 mil milhões de gamers no mundo: indústria de gaming vale mais de 300 mil milhões de dólares. [Online]. Available: https://tek.sapo.pt/noticias/negocios/artigos/ha-27-mil-milhoes-de-gamers-no-mundo-industria-de-gaming-vale-mais-de-300-mil-milhoes-de-do

[10] B. Skwarczek. (2021, 06) How the gaming industry has leveled up during the pandemic. [Online]. Available: https://www.forbes.com/sites/forbestechcouncil/2021/06/17/how-the-gaming-industry-has-leveled-up-during-the-pandemic/?sh=6ced97fb297c

[11] Werner Ballhaus, Wilson Chow, Emmanuelle Rivet. (2022, 06) Perspectives from the global entertainment and media outlook 2022–2026. [Online]. Available: https://www.pwc.com/gx/en/industries/tmt/media/outlook/outlook-perspectives.html

[12] Wikipedia contributors, "Video game," 07 2022. [Online]. Available: https://en.wikipedia.org/wiki/Video_game

[13] W. contributors, "List of most expensive video games to develop," 07 2022. [Online]. Available: https://en.wikipedia.org/wiki/List_of_most_expensive_video_games_to_develop

[14] Wikipedia contributors. (2022, 11) Game engine. [Online]. Available: https://en.wikipedia.org/wiki/Game_engine

[15] W. contributors, "Pokémon mystery dungeon — Wikipedia, the free encyclopedia," 2022, [Online; accessed 31-October-2022]. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Pok%C3%A9mon_Mystery_Dungeon&oldid=1119031983

[16] B. . Games. (2023, 08) Dwarf fortress features. [Online]. Available: http://www.bay12games.com/dwarves/features.html

[17] G. Amaral. (2023, 02) Tese - algoritmo de grafos. https://github.com/Kerium/Tese-Algoritmo-Grafos. [Online]. Available: https://github.com/Kerium/Tese-Algoritmo-Grafos

[18] N. Shaker, J. Togelius, and M. Nelson, *Procedural Content Generation in Games (Computational Synthesis and Creative Systems)*, 1st ed. Springer, 2016.

[19] Wikipedia contributors, "Diffusion-limited aggregation," 07 2021. [Online]. Available: https://en.wikipedia.org/wiki/Diffusion-limited_aggregation

[20] F. Bellelli. (2022, 04) The fascinating world of voronoi diagrams. [Online]. Available: https://towardsdatascience.com/the-fascinating-world-of-voronoi-diagrams-da8fc700fa1b

[21] Marchete. (2017, 06) What are voronoi diagrams? [Online]. Available: https://www.codingame.com/playgrounds/243/voronoi-diagrams/what-are-voronoi-diagrams

[22] H. Wolverson. (2023, 02) Procedural map generation techniques. [Online]. Available: https://www.youtube.com/watch?v=TlLIOgWYVpI&t=866s

[23] W. contributors. (2022, 11) Perlin noise. [Online]. Available: https://en.wikipedia.org/wiki/Perlin_noise

[24] G. N. Y. Amy K. Hoover, Julian Togelius, "Composing video game levels with music metaphors through functional scaffolding," 2015. [Online]. Available: http://julian.togelius.com/Hoover2015Composing.pdf

[25] (2022, 10) Make 2d games with gamemaker. [Online]. Available: https://gamemaker.io/en/

[26] Unity. (2022, 10) Unity real-time development platform. [Online]. Available: https://unity.com/

[27] I. Epic Games. (2022, 12) Features unreal engine. [Online]. Available: https://www.unrealengine.com/en-US/features

[28] Godot. (2022, 10) Godot engine - features. [Online]. Available: https://godotengine.org/features

[29] T. M. Team. (2022, 11) Introduction — monogame documentation. [Online]. Available: https://docs.monogame.net/index.html

[30] raysan5. (2022, 12) raylib — a simple and easy-to-use library to enjoy videogame programming. [Online]. Available: https://www.raylib.com/

[31] HeapsIO. (2022, 12) Heaps - haxe game engine - heaps.io game engine. [Online]. Available: https://heaps.io/

[32] I. S. S.A. (2023, 02) Aseprite - animated sprite editor and pixel art tool. [Online]. Available: https://www.aseprite.org/

[33] Karleigh Moore, Thaddeus Ably, Hannah Pang, Beakal Tiliksew, Jimin Khim. (2022, 10) A-Star Search. [Online]. Available: https://brilliant.org/wiki/a-star-search/

[34] GeeksforGeeks. (2022, 08) Prim's Minimum Spanning Tree. [Online]. Available: https://www.geeksforgeeks.org/prims-minimum-spanning-tree-mst-greedy-algo-5/

[35] (2022, 08) Prim's Algorithm. [Online]. Available: https://www.programiz.com/dsa/prim-algorithm

[36] G. Yannakakis and J. Togelius, *Artificial Intelligence and Games*, 1st ed. Springer, 2018.

[37] G. Smith. (2015, 07) Interview: How does level generation work in brogue? [Online]. Available: https://www.rockpapershotgun.com/how-do-roguelikes-generate-levels

[38] TheZZAZZGlitch. Pokémon mystery dungeon - deconstructing the dungeon generation algorithms. [Online]. Available: https://www.youtube.com/watch?v=fudOO713qYo&list=WL&index=171

[39] C. Bradfield. (2018, 12) Procedural generation in godot: Dungeon generation. [Online]. Available: https://www.youtube.com/watch?v=G2_SGhmdYFo&list=PLsk-HSGFjnaH82Bn6xbQNehatj3sIvtMQ&index=6