UNIVERSIDADE FEDERAL DE PERNAMBUCO

CENTRO DE INFORMÁTICA

GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

LUCAS E SILVA DE SOUZA

Análise comparativa de técnicas de geração procedural em jogos de quebra-cabeça: Witchie e Sokoban

Recife

2025

LUCAS E SILVA DE SOUZA

Análise comparativa de técnicas de geração procedural em jogos de quebra-cabeça: Witchie e Sokoban

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Ciência da Computação do Centro de Informática da Universidade Federal de Pernambuco, como requisito parcial para obtenção do título de Bacharel em Ciências da Computação.

Orientador (a): Prof. Dr. Geber Lisboa Ramalho (glr)

Recife

2025

Ficha de identificação da obra elaborada pelo autor,
através do programa de geração automática do SIB/UFPE

Souza, Lucas e Silva de.
Análise comparativa de técnicas de geração procedural em jogos de quebra-cabeça: Witchie e Sokoban / Lucas e Silva de Souza. - Recife, 2025.
41 p. : il., tab.

Orientador(a): Geber Lisboa Ramalho
Trabalho de Conclusão de Curso (Graduação) - Universidade Federal de Pernambuco, Centro de Informática, Ciências da Computação - Bacharelado, 2025.
Inclui referências.

1. Geração Procedural de Conteúdo. 2. Jogos. 3. PCG. 4. Sokoban. 5. Técnicas de PCG. I. Ramalho, Geber Lisboa. (Orientação). II. Título.

000 CDD (22.ed.)

LUCAS E SILVA DE SOUZA

Análise comparativa de técnicas de geração procedural em jogos de quebra-cabeça: Witchie e Sokoban

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Ciência da Computação do Centro de Informática da Universidade Federal de Pernambuco, como requisito parcial para obtenção do título de Bacharel em Ciências da Computação.

Aprovado em: 09/04/2025

BANCA EXAMINADORA

_____

Prof. Dr. Geber Lisboa Ramalho (Orientador)

Universidade Federal de Pernambuco

_____

Prof. Dr. Filipe Carlos de Albuquerque Calegario (Examinador Interno)

Universidade Federal de Pernambuco

“Adeus foliões, adeus carnaval
É hora de se despedir
Juntos cantamos tanto
Por que findar em pranto
Nosso destino é cantar e sorrir
Fingindo alegria no rosto e partir”
(Getúlio Cavalcanti, 1998)

AGRADECIMENTOS

Gostaria de começar agradecendo a toda equipe da Witchie: Ernesto, Guilherme, Thales, Bruna e Barreto. Sem cada um de vocês a bruxinha nunca teria voado tão alto.

A todos os professores e mentores que me inspiraram e incentivaram a minha paixão por jogos: Geber, Giordano, Barroca e tantos outros.

A Bruna, minha noiva, que sempre esteve ao meu lado nas noites mal dormidas e nos momentos de vitórias e derrotas.

A meus pais, irmão e toda família, que sempre me apoiaram e estiveram torcendo pelo meu sucesso.

A Vivian, uma amiga querida que muito me ajudou com toda a sua experiência.

E, por fim, gostaria de deixar um agradecimento especial a meu avô Bartô e minha avó Jussara, que infelizmente nos deixaram mais cedo este ano. A vocês eu dedico esse trabalho, e agradeço por tudo que viveram comigo, tudo que me ensinaram e tudo que me amaram. Nos encontraremos novamente em outros carnavais.

RESUMO

Geração Procedural de Conteúdo (PCG) em jogos se refere à criação automática de conteúdo através de algoritmos, podendo ser aplicada na criação de dungeons, mapas, armas, vegetações e terrenos. Um exemplo desses jogos é o Sokoban, que tem utilizado amplamente diversas abordagens da Geração Procedural de Conteúdo para a criação automática de níveis. O Sokoban tem sido utilizado como referência no design de jogos de lógica e quebra-cabeça, influenciando novas criações, como *Witchie*, um jogo mobile desenvolvido por estudantes da UFPE, do qual o autor do presente estudo fez parte. *Witchie* é um jogo que reinterpreta o Sokoban ao introduzir um sistema de movimentação baseado em deslizamentos, adicionando uma nova camada de complexidade ao quebra-cabeça. Diante desse sistema de movimentação distinto, este estudo tem como objetivo investigar como diferentes técnicas de PCG se adaptam às especificidades do jogo *Witchie*. Para isso, foi empregada uma abordagem que trabalha "de trás para frente", e uma segunda versão adaptada, aplicada a ambos os jogos. A partir das análises, foi possível verificar que ambos os métodos são ferramentas úteis para a geração de níveis, especialmente quando empregados como suporte no processo de design.

Palavras-chaves: Geração Procedural de Conteúdo. Jogos. PCG. Sokoban. Técnicas de PCG.

ABSTRACT

Procedural Content Generation (PCG) in games refers to the automatic creation of content through algorithms, which can be applied to the generation of dungeons, maps, weapons, vegetation, and terrains. One example of such games is *Sokoban*, which has extensively utilized various Procedural Content Generation approaches for automatic level creation. *Sokoban* has been widely used as a reference in the design of logic and puzzle games, influencing new creations such as *Witchie*, a mobile game developed by students from UFPE, in which the author of this study participated. *Witchie* reinterprets *Sokoban* by introducing a sliding movement system, adding a new layer of complexity to the puzzle. Given this distinct movement system, this study aims to investigate how different PCG techniques adapt to the specificities of *Witchie*. To achieve this, a "backward" approach was employed, along with a second adapted version, applied to both games. Through the analyses, it was possible to verify that both methods are useful tools for level generation, especially when used as support in the design process.

Keywords: Procedural Content Generation, Games, PCG, Sokoban, PCG Techniques.

LISTA DE FIGURAS

LISTA DE QUADROS

LISTA DE ABREVIATURAS E SIGLAS

PCG	(*Procedural Content Generation)* Geração Procedural de Conteúdo
UFPE	Universidade Federal de Pernambuco
LLM	(*Large Language Models*) Modelo de Linguagem de Grande Escala
PRNG	(*Pseudo-random Number Generator*) Geradores de Números Pseudoaleatórios
GG	Gramáticas Generativas
L-systems	Sistemas Lindenmayer
AS	Algoritmos Espaciais
IA	Inteligência Artificial
ANN	(Artificial Neural Networks) Redes Neurais Artificiais

SUMÁRIO

1 INTRODUÇÃO

1.1 CONSIDERAÇÕES INICIAIS

A Geração Procedural de Conteúdo (PCG da sigla em inglês para *Procedural Content Generation*) tem sido amplamente utilizada na indústria de jogos para criar mundos, personagens e desafios de forma automatizada. Em jogos de quebra-cabeça, o uso de PCG é ainda mais relevante, pois acelera consideravelmente o tempo de criação pelos desenvolvedores, e aumenta o conteúdo fornecido ao jogador. Sokoban [11], um dos jogos de quebra-cabeça mais estudados em pesquisas de PCG, é conhecido por sua simplicidade e por exigir soluções otimizadas para problemas bem definidos.

Sokoban é um jogo de quebra-cabeça criado em 1981 por Hiroyuki Imabayashi, e lançado em 1982 pela sua empresa *Thinking Rabbit*. Nele o jogador assume o papel de um zelador de armazém encarregado de empurrar caixas até posições específicas (*goals*) dentro de uma *grid* retangular. O desafio do jogo reside em suas restrições fundamentais: o personagem se move um espaço por vez, e pode apenas empurrar as caixas, nunca puxá-las. Além disso, existem paredes, que funcionam como obstáculos, espalhadas pelo nível, fazendo com que cada movimento possa resultar em o jogador ficar “preso”, isto é, num estado onde não consegue mais colocar alguma caixa na posição necessária para completar o nível. Ao longo dos anos, Sokoban tornou-se uma referência no design de jogos de lógica e quebra-cabeça, inspirando diversas variações e influenciando a criação de novos títulos dentro do gênero.

Existem diversas técnicas de PCG documentadas na literatura aplicadas ao Sokoban. As abordagens mais fundamentais foram descritas por Yoshio e colaboradores [5] e mais tarde aprimoradas por Joshua e Parberry [7]. Outras abordagens mais diferenciadas são as de Graham e colaboradores [8] e Karman colaboradores [2]. No entanto, existem variações do Sokoban que introduzem mecânicas diferenciadas, onde o personagem pode não se mover passo a passo. Essas dinâmicas podem alterar substancialmente a maneira de solucionar os níveis e, por extensão, para gerá-los de maneira procedural.

Entre as variações do Sokoban, pode-se citar o jogo *Witchie*, desenvolvido por um time de estudantes da UFPE do qual o autor do presente estudo fez parte. *Witchie* é um jogo mobile que reinterpreta o Sokoban ao introduzir um sistema de movimentação baseado em deslizamentos. Diferente do jogo original, onde o

personagem se move um bloco por vez, em *Witchie* a protagonista desliza continuamente em uma direção até encontrar um obstáculo. Essa modificação fundamental adiciona uma nova camada de complexidade ao quebra-cabeça, tornando muito mais fácil que um jogador fique preso.

1.2 MOTIVAÇÃO

Sokoban tem sido amplamente explorado no campo da Geração Procedural de Conteúdo, com diversas abordagens sendo utilizadas para a criação automática de níveis. Técnicas como busca heurística, algoritmos genéticos, modelos de linguagem de grande escala (LLM da sigla em inglês para *Large Language Models*) e aprendizado de máquina têm sido aplicadas para gerar níveis que sejam solucionáveis. Como foi visto, o jogo *Witchie* compartilha a essência lógica de Sokoban, mas introduz um sistema de movimentação distinto, diante disso este estudo tem como objetivo investigar como diferentes técnicas de PCG se adaptam às especificidades do jogo *Witchie*. Assim, a pesquisa busca oferecer contribuições tanto para o desenvolvimento de jogos quanto para a comunidade acadêmica, fornecendo insights sobre como adaptar algoritmos consolidados para casos de uso específicos.

1.3 OBJETIVOS

Diante disso, o presente estudo tem como objetivo realizar uma análise comparativa de técnicas de Geração Procedural (PCG) aplicadas a Sokoban e Witchie, avaliando a viabilidade, qualidade e dificuldade dos níveis gerados. De modo específico, pretende-se identificar e selecionar diferentes técnicas de PCG documentadas na literatura, aplicar as técnicas selecionadas para gerar níveis para Sokoban, adaptar as técnicas selecionadas para gerar níveis para Witchie; avaliar os níveis gerados em termos de adaptabilidade, solubilidade e dificuldade; bem como comparar os resultados obtidos entre os dois jogos.

Para fins de teste dos níveis gerados, foi desenvolvido um protótipo de *Sokoban* e *Witchie* utilizando a engine PuzzleScript [6]. PuzzleScript é uma *engine open-source* de jogos de quebra-cabeça baseada em HTML5, desenvolvida por Stephen Lavelle. Ela foi escolhida pois permite uma implementação ágil das mecânicas necessárias para os experimentos. Todos os níveis desenvolvidos estão

disponíveis para testes públicos nos seguintes links:

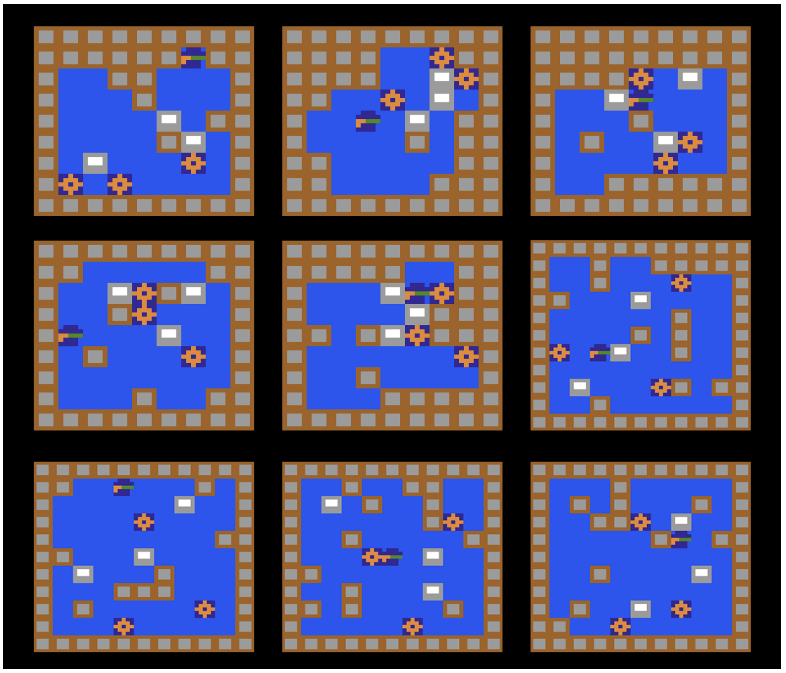
Witchie:

https://www.puzzlescript.net/play.html?p=17fd5a20c92d2b2febca87868fb1f7dd

Sokoban:

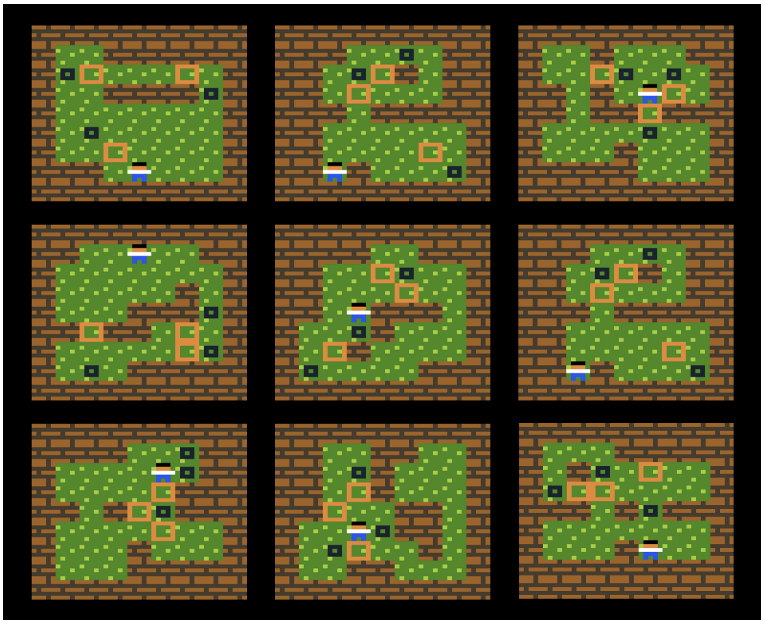
https://www.puzzlescript.net/play.html?p=3fbe6acf72b35aa4d35c3079d67bec1e

Figura 1. Alguns níveis do jogo *Witchie.*



Fonte: Autoria própria, 2025

Figura 2. Alguns níveis do jogo Sokoban



Fonte: Autoria própria, 2025

2 JUSTIFICATIVA PARA O USO DE TÉCNICAS DE GERAÇÃO PROCEDURAL

A utilização de técnicas de PCG em jogos de quebra-cabeça tem um potencial significativo para facilitar a criação de níveis, especialmente em projetos que buscam acelerar o desenvolvimento de conteúdo e oferecer uma experiência mais variada para os jogadores. Com o aumento da popularidade de sistemas que oferecem conteúdo diário, há uma crescente necessidade de ferramentas que permitam criar novos níveis com consistência e qualidade.

Uma ferramenta de auxílio baseada em PCG poderia não só acelerar o processo de criação de níveis, mas também oferecer a possibilidade de gerar grandes quantidades de conteúdo de uma só vez, sem a necessidade de desenvolver cada nível manualmente. Isso seria especialmente valioso em projetos que desejam implementar *features* como desafios diários ou eventos periódicos, proporcionando aos jogadores sempre algo novo para explorar.

Entretanto, a aplicação de técnicas de PCG nesse contexto apresenta alguns desafios. Dentre eles, o que diz respeito ao balanceamento da dificuldade do conteúdo gerado. Um sistema que apenas cria níveis de maneira automática pode acabar gerando apenas níveis excessivamente fáceis ou desproporcionalmente difíceis. Assim, para que o jogo proporcione uma experiência coerente, é necessário garantir uma boa variedade de níveis, desde os mais simples até desafios mais complexos.

Além disso, um bom nível deve ser divertido para o jogador, seja transmitindo um senso de realização após um nível difícil ou de progresso por vencer múltiplos níveis em sequência. Dessa forma, o desafio não se limita a criar níveis solucionáveis, mas a produzir conteúdo que seja verdadeiramente divertido e satisfatório para os jogadores.

Dessa forma, uma solução adequada deve atender a uma série de requisitos. Primeiramente, a ferramenta deve funcionar em sua essência como um facilitador para o trabalho do designer, acelerando o processo de criação sem sacrificar a qualidade. Além disso, deve permitir que o designer mantenha controle sobre o conteúdo gerado, possibilitando ajustes conforme necessário. Outro requisito fundamental é a capacidade de gerar níveis que apresentem uma boa variedade de dificuldades, algo que a experiência profissional do designer garante certa intuição. Por fim, para que essa solução seja eficaz, os níveis gerados devem ser divertidos,

proporcionando uma experiência positiva ao jogador.

A partir desses requisitos, para este trabalho, optou-se por adaptar o método de Taylor e Parberry, motivado pela abordagem, que utiliza a construção de salas vazias a partir de templates pré-definidos. O uso de templates facilita a definição de padrões básicos que podem ser manipulados e combinados para gerar uma variedade maior de níveis. Para *Witchie*, cujo a mecânica de movimentação deslizante eleva a dificuldade de posicionamento, essa abordagem se mostrou promissora por permitir a criação de templates específicos que facilitem a geração de níveis adequados de forma consistente.

3 GERAÇÃO PROCEDURAL DE CONTEÚDO

3.1 CONCEITO E VANTAGENS

Geração Procedural de Conteúdo (PCG, da sigla em inglês para *Procedural Content Generation*) em jogos se refere à criação automática de conteúdo através de algoritmos, podendo ser aplicada na criação de dungeons, mapas, armas, vegetações e terrenos. Além disso, essa criação pode ser aleatória ou se adaptar aos resultados gerados [4]. Assim, técnicas procedurais são uma alternativa para criar mundos de jogo complexos em um tempo limitado sem colocar um grande fardo nos designers de conteúdo do jogo. Todavia, para evitar perder o controle sobre o processo de design, os artistas e designers ainda podem influenciar o produto final ajustando os parâmetros do procedimento [1].

Existem vários motivos para a aplicação da PCG em um jogo visando auxiliar desde os desenvolvedores e artistas que trabalham na criação do jogo até os jogadores. Dentre esses pode-se citar: auxílio a artistas, rejogabilidade, economia de espaço em disco e satisfação do desejo dos jogadores em explorar [2].

Auxiliando Artistas - A quantidade de conteúdo dentro dos mundos está crescendo rapidamente, devido ao aumento do tamanho geral e dos detalhes dentro de um mundo. Alguns mundos exigem quantidades tão altas de conteúdo que os artistas não conseguem mais criar tudo manualmente. Diminuir a quantidade de tempo que um artista usa para criar conteúdo economiza os custos e o tempo de produção. Porém nem sempre as ferramentas simples de copiar e colar são suficientes. Assim, se utilizar apenas desses recursos poderia resultar em mundos repetitivos. Nesses casos, técnicas de PCG offline poderiam ser usadas para executar tarefas repetitivas, incluindo suas variações, e gerar ou alterar partes do mundo [2].

Aumentar o tempo de jogo - É possível aumentar o tempo que um jogo pode ser jogado usando PCG, seja através de rejogabilidade ou da criação de mundos infinitos. Um jogador pode passar mais tempo jogando o jogo se puder repetir o jogo inteiro ou partes do mesmo sem que a experiência se torne repetitiva. Para isso, a PCG pode ser usada, por exemplo, para gerar proceduralmente conteúdo exclusivo a cada vez que um jogador joga o jogo, incentivando-o a jogar mais de uma vez. Outro fator que pode levar o jogador a passar mais tempo jogando é se o mundo em que o jogo se passa tiver um tamanho infinito. Para gerar mundos que pareçam infinitos em tamanho é possível gerar uma pequena região do mundo ao redor do jogador à medida que ele

se move. Dessa forma, o jogador nunca pode atingir nenhum limite do espaço jogável, fazendo com que o mundo pareça infinito em tamanho [2].

Espaço em armazenamento limitado - A quantidade de espaço de armazenamento aumentou massivamente nos últimos anos, no entanto nem sempre foi esse o caso. Assim, a PCG era usada para limitar o uso de espaço em dispositivos móveis. Para isso, a PCG gerava proceduralmente o conteúdo apenas quando fosse necessário. Dessa forma, ao invés do jogo ser enviado com todo seu conteúdo, era enviado apenas os algoritmos para gerar esse conteúdo. Todavia, o tempo de carregamento seria mais longo [2].

Satisfazendo o desejo de explorar - A história da humanidade é repleta de descobertas motivadas pelo desejo dos humanos em explorar o desconhecido. Todavia, atualmente, explorar se tornou menos acessível à maioria das pessoas. Assim, para satisfazer o desejo de explorar, novas abordagens têm sido usadas. Os jogos podem levar os jogadores a explorar territórios desconhecidos e desfrutar de experiências emocionantes. Em alguns casos, a PCG é um fator propulsor por trás desses jogos [2].

3.2 TAXONOMIA

Visando melhor entender as diferenças e semelhanças entre as abordagens de métodos e algoritmos, será apresentado a seguir algumas distinções que Togelius e colaboradores [9] sugerem em seu estudo para caracterizar a PCG e algumas atualizações pontuadas por Melo [4].

1. **Online vs. Offline -** Técnicas de PCG podem ser usadas para gerar conteúdo à medida que o jogador avança, de forma online, em que o conteúdo é "gerado" de acordo com a forma como o jogador joga; offline, sugerindo conteúdo para o designer; ou mista, de modo que o algoritmo online sugere alguns comportamentos para o offline e vice-versa [9].
2. **Necessário vs. Opcional -** O conteúdo gerado pode ser definido como necessário quando o jogador precisa dele para progredir no jogo; ou opcional quando pode ser "evitado" ou ignorado, exemplo desse tipo de conteúdo são os itens e histórias que não afetem na narrativa principal do jogo [9].
3. **Genérico vs. Adaptativo -** O conteúdo genérico é gerado sem levar em conta o comportamento do jogador, que é o mais comumente usado; enquanto que o

adaptativo é personalizado e centrado no jogador, de modo que a interação do jogador é analisada e o conteúdo é gerado a partir disso [4].

4. **Estocástico vs. Determinístico -** A PCG determinística permite a "regeneração" de um mesmo conteúdo, dado que receba o mesmo ponto de partida e os mesmos parâmetros para seu método. Enquanto a PCG estocástica age da forma oposta, em que normalmente não é possível recriar um mesmo conteúdo dada uma mesma entrada [9].
5. **Sementes randômicas vs. Vetores Parâmetro -** A distinção dessas duas propriedades se encontra nos dados de entrada dos algoritmos. De modo que alguns só recebem um valor de "semente" de um gerador de número randômico, que criam aleatoriamente um conjunto de parâmetros que alimentam a geração; enquanto que outros podem receber valores que controlem os resultados as sementes criam aleatoriamente um conjunto de parâmetros que alimentam a geração, permitindo a modificação de cada parâmetro pelo usuário de forma individualizada [9].
6. **Construtivo vs. Gerar-e-Testar -** Os algoritmos construtivos geram o conteúdo uma única vez, havendo a necessidade de uma verificação do conteúdo gerando, já o algoritmo gerar-e-testar segue um critério previamente estabelecido e testa esse resultado para verificar se atingiu o padrão desejado [9].
7. **Geração Automática vs. Autoria Mista -** Na geração automática a entrada do usuário é limitada, enquanto que a autoria mista permite uma maior influência por parte do usuário, seja ele designer ou jogador [4].

3.3 CLASSES DE MÉTODOS EM PCG

Após décadas de pesquisa, existem muitos métodos para gerar proceduralmente muitos tipos de conteúdo de jogo. A Tabela 1 apresenta os métodos comuns para geração de conteúdo de jogo, que serão apresentados a seguir.

Quadro 1. Classes e métodos em PCG

Classes	Métodos
Geradores de números pseudoaleatórios (PRNG)	1. Ruído Perlin 2. Ruído Simplex
Gramáticas Gerativas (GG)	1. L-systems 2. Gramáticas Divididas 3. Gramáticas de Parede 4. Gramáticas de Forma 5. Gramática de Grafos Cíclicos
Algoritmos Espaciais (AS)	1. Ladrilhamento e Camadas 2. Subdivisão da Grade 3. Fractais 4. Diagramas de Voronoi
Inteligência Artificial (IA)	1. Algoritmos Genéticos 2. Redes Neurais Artificiais (ANN) 3. Satisfação de restrições e planejamento 4. Modelos de Linguagem de Grande Escala

Fonte: Autoria própria, 2025.

Geradores de números pseudoaleatórios (PRNG, da sigla em inglês para *Pseudo-random Number Generator*) podem ser usados para imitar a aleatoriedade encontrada na natureza, por exemplo, na forma de uma montanha, uma nuvem ou uma flor [1]. Para isso os PRNG geram sequências pseudo-aleatórias de números, que podem ser reproduzidos usando o mesmo número inicial (chamado de semente). Essas sequências aleatórias de números são um ruído aleatório, em que os números não têm relação entre si. Apesar de também existir formas de ruído aleatório em que os números se relacionam. Devido a sua aleatoriedade, os PRNG não são sofisticados o suficiente para gerar quebra-cabeças sozinho. No entanto, o PRNG pode auxiliar na geração de variabilidade de outras técnicas de geração de quebra-cabeças [2]. Ruído Perlin e Ruído Simplex são exemplos de PRNG:

1. **Ruído Perlin** é um algoritmo de ruído no qual os números têm uma relação entre si, através de uma transição suave entre os valores. Isso o torna extremamente adequado para geração de terreno, por exemplo, porque a altura do terreno também é uma forma de transição suave. O Ruído Perlin é usado principalmente em sua forma bidimensional, mas funciona para qualquer número de dimensões [2].

2. **Ruído Simplex** é uma outra forma de ruído suave, semelhante ao Perlin, mas que difere em algumas partes. As diferenças mais importantes são que o Ruído Simplex usa triângulos equiláteros, em vez de retângulos, resultando em artefatos de 60 graus em vez de artefatos de 90 graus, que são mais difíceis de detectar pelo olho humano. O Simplex também soma contribuições de cada canto e é um pouco mais rápido [2].

Gramáticas Gerativas (GG) são conjuntos de regras que operam a partir de palavras individuais, podendo gerar apenas sentenças gramaticalmente corretas. Elas podem ser usadas para criar objetos a partir de elementos codificados como letras/palavras. Assim, por serem feitas de elementos fixos, a GG é bem adequada para gerar quebra-cabeças [2]. L-systems, gramáticas divididas, gramáticas de parede e gramáticas de forma são exemplos de Gramáticas Gerativas que têm sido usadas para geração de conteúdo em entretenimento.

1. **Sistemas Lindenmayer (L-systems)** consistem em uma gramática que através de símbolos descrevem as características de um objeto. Uma string gerada pela gramática descreve a estrutura ou o comportamento de um objeto [1].
2. **Gramáticas Divididas -** Assim como o L-systems, Gramáticas Divididas funcionam em formas codificadas por string. A Gramática Dividida funciona aplicando regras de reescrita para transformar uma forma inicial em novas formas. Gramáticas divididas são livres de contexto, de modo que o processo de reescrita produzirá sempre o mesmo resultado dado um conjunto de regras de reescrita, independentemente da ordem em que os símbolos da gramática são processados, garantindo previsibilidade e consistência na Geração Procedural de Conteúdos [1].
3. **Gramáticas de Parede** são usadas especificamente para criar exteriores de edifícios, manipulando as formas semelhantemente às Gramáticas Divididas, mas podendo gerar formas mais avançadas, com formas tridimensionais complexas, criando sacadas e escadas de incêndio, por exemplo [1].
4. **Gramáticas de Formas**, diferentemente do Sistema L e da Gramática Dividida, são gramáticas sequenciais e sensíveis ao contexto. Assim, no seu processo de reescrita cada símbolo e seus vizinhos na sequência determinam seus os substitutos do símbolo original. Similarmente às Gramáticas de Parede, também podem gerar estruturas complexas [1].

Além desses quatro tipos apresentados por Hendrikx e colaboradores [1] e Karman [2] traz em seu trabalho o método **Gramática de Grafos Cíclicos**, que pode ser usado para gerar missões em um jogo. As abordagens tradicionais da Gramática de Grafos, o grafo se expande a partir de um ponto inicial, criando caminhos que eventualmente resultam em nós, que seriam um “beco sem saída”. Para corrigir isso, esses nós finais são conectados a outras partes do grafo. Enquanto na Gramática de Grafos Cíclicos esses problemas são evitados usando ciclos, permitindo a criação de caminhos mais interconectados. Essa abordagem é especialmente útil em jogos e simulações, pois oferece ao jogador rotas alternativas e múltiplas formas de navegação, tornando o design dos mapas e níveis mais dinâmico e realista.

Algoritmos Espaciais (AS) manipulam o espaço para gerar conteúdo de jogo. A saída é criada usando uma entrada com estrutura, por exemplo, uma grade ou autorrecorrência [1]. Algoritmos Espaciais pode ser muito útil ao gerar quebra-cabeças, visto que estes podem ser descritos como um problema abstrato em algum espaço de possibilidade [2]. A seguir, métodos de Algoritmos Espaciais são apresentados.

1. **Ladrilhamento e Camadas** são usadas para criar um espaço de jogo decompondo um mapa em uma grade e integrar várias dessas grades em um mesmo mapa. Essa abordagem permite gerar efeitos de sobreposição, como água corrente, e de espaço de jogo com aparência 3D usando apenas uma quantidade limitada de texturas de terreno de origem [1].
2. **Subdivisão da Grade** é uma técnica dinâmica para geração de objetos. Inicialmente, o objeto é dividido em uma grade uniforme, na qual são aplicadas as texturas apropriadas. Um algoritmo de subdivisão de grade é usado para adicionar progressivamente mais detalhes ao objeto. A parte dinâmica desta técnica é baseada no ponto de vista do observador, de modo que somente as células da grade mais próximas do ponto de detalhe atual devem ser subdivididas em células menores para criar o nível de detalhe necessário. Um exemplo usando esta técnica é a renderização de um terreno gerado proceduralmente, em que somente as células perto do jogador são detalhadas, enquanto o resto do terreno é mais grosseiro, economizando recursos computacionais [1].
3. **Fractais** são figuras recursivas formadas por cópias de si mesmas. Com fractais, alguns parâmetros controlam uma ampla gama de resultados possíveis. Uma de

suas vantagens é a capacidade de representar objetos extremamente detalhados de forma compacta, utilizando apenas uma função recursiva simples [1].

4. **Diagramas de Voronoi** são métodos de decomposição espacial que dividem um espaço em regiões baseadas na proximidade de pontos de referência, chamados de sementes. A decomposição estabelece fronteiras de pontos igualmente distantes dos pontos de semente mais próximos. Diversas abordagens foram desenvolvidas para torná-los eficientes, permitindo sua aplicação em tempo quase real [1].

Inteligência Artificial (IA) é um crescente ramo da ciência da computação usada para imitar a inteligência animal ou humana, desde tarefas simples como reconhecimento de fala até o planejamento e execução de tarefas físicas por robôs, por exemplo. Dentro do PCG, a IA é usada em Algoritmos Genéticos, Redes Neurais, Satisfação e Planejamento de Restrições e LLMs, que serão explicadas a seguir [2].

1. **Algoritmos Genéticos** são técnicas de otimização que imitam a evolução biológica. Cada solução possível é codificada como strings (cromossomos), e uma função de aptidão avalia sua qualidade. A geração de novas soluções ocorre por meio de duas operações: mutação, que altera uma solução existente para criar uma nova variação; e crossover, que combina partes dos cromossomos de diferentes soluções (pais) para gerar novas soluções (filhos). A frequência dessas operações são determinadas pela taxa de mutação e de crossover. Inicialmente, um conjunto de N soluções possíveis são geradas. Em seguida, novas soluções são geradas aplicando mutação e crossover sobre os cromossomos pais selecionados aleatoriamente com base em sua aptidão. Esse processo se repete até encontrar uma solução satisfatória ou até atingir um número pré-definido de iterações [1].
2. **Redes Neurais Artificiais (ANN**, da sigla em inglês para *Artificial Neural Networks*) são modelos computacionais capazes de aprender a relação entre uma entrada e uma saída, minimizando o erro entre a saída gerada e a saída esperada. Esse modelo pode ser usado para encontrar padrões e classificar, além de lembrar e estruturar dados. A ANN é composta por unidades computacionais chamadas neurônios, conectadas por arestas ponderadas. Cada neurônio pode receber múltiplas entradas, processá-las e decidir se será ativado. Se ativado, ele transmite um sinal combinado para os neurônios seguintes

através das conexões de saída. Durante o aprendizado, a rede ajusta os pesos das conexões com base no erro calculado entre a saída gerada e a saída esperada. Esse ajuste contínuo permite que a ANN aprimore seu desempenho ao longo do tempo [1].

3. **Satisfação de restrições e planejamento** envolve encontrar um caminho de um estado inicial para um estado final, aplicando uma sequência de ações. Assim, um problema de planejamento é constituído por em um estado inicial, um conjunto de ações e um teste de meta. Cada ação só pode ser executada quando sua condição inicial é satisfeita, desse modo seu efeito pode incluir adicionar ou remover variáveis, resultando em um novo estado. Para isso, algoritmos de busca de espaço de estado para frente podem iniciar o planejamento a partir do estado inicial. Enquanto, algoritmos de busca de espaço de estado para trás iniciam o planejamento a partir do estado final. Todavia, ainda assim, o planejamento geralmente é NP-difícil, o que reforça a importância da heurística no planejamento [1].
4. **Modelo de Linguagem de Grande Escala (LLM**, da sigla em inglês para *Large Language Models*) - são ferramentas capazes de escrever histórias, gerar código e responder perguntas, com aplicações diversas. No contexto da criação de níveis de jogos, especialmente para o *Sokoban*, os LLM podem gerar níveis funcionais, mesmo diante de restrições complexas e relações espaciais multidimensionais. A eficácia desses modelos na geração de níveis jogáveis melhora significativamente com o aumento do conjunto de dados de treinamento, tornando-os uma abordagem promissora. Além disso, os LLM se destacam por sua controlabilidade e capacidade de generalização, permitindo criar níveis com características específicas por meio de *prompts* em linguagem natural [8].

3.4 MÉTODO TESTADO NO *WITCHIE*

Considerando o contexto de jogos de quebra-cabeça como Witchie e Sokoban, buscou-se um método que operasse de forma *offline*, possibilitando a geração de níveis antes da execução do jogo; que seguisse uma lógica de gerar-e-testar, permitindo controle sobre a solubilidade dos níveis; e que favorecesse a autoria mista, oferecendo flexibilidade para que se possa ajustar os resultados conforme necessário.

Método de Taylor e Parberry

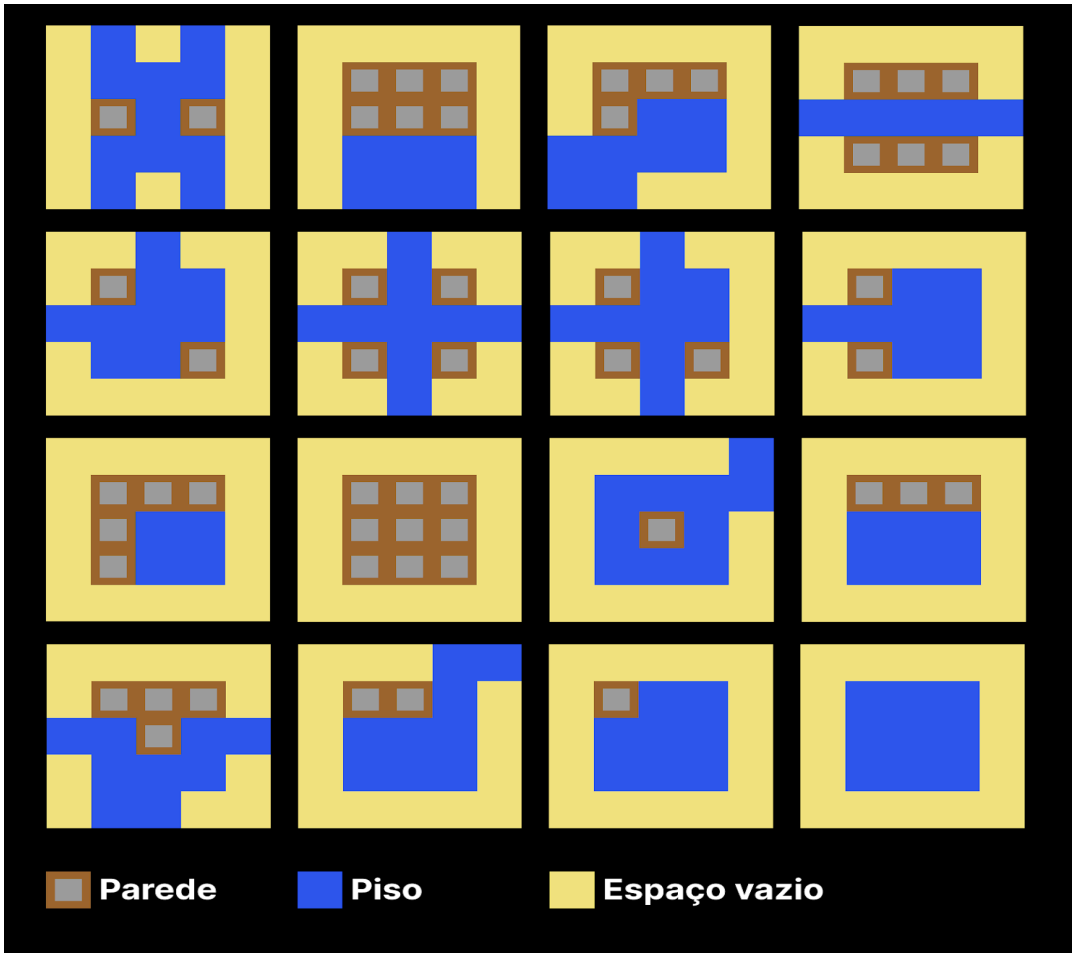
Com base nesses critérios, o método proposto por Taylor e Parberry [7] foi selecionado e adaptado para os propósitos deste trabalho, por apresentar uma estrutura compatível com essas exigências e, como será descrito mais a frente, pela estratégia de utilizar templates para gerar os níveis.

O método utilizado no estudo desses autores baseia-se na ideia de trabalhar “de trás para frente", ou seja, partindo do objetivo até o início, técnica essa já explorada anteriormente, mas usada apenas para resolver níveis existentes. Neste caso, essa abordagem foi aplicada para gerar novos níveis de forma procedural. Para isso, o algoritmo segue três etapas principais: 1. Construção de uma sala vazia, 2. Posicionamento dos objetivos (*goals*) na sala, e 3. Determinação do estado mais distante do estado final, descritas a seguir.

1. Construção de uma Sala Vazia

A sala vazia é gerada da seguinte maneira: Inicialmente, um tamanho de nível (largura e altura) é escolhido aleatoriamente, ou pelo usuário, dentro de um intervalo pré-definido. Em seguida, o espaço do nível é dividido em uma grade de blocos de 3×3. Cada bloco é preenchido utilizando um template aleatório, que pode ser rotacionado e espelhado. Os templates são padrões de blocos 3x3 que incluem paredes e pisos, cercados por espaços vazios, paredes ou pisos. Essa cerca faz com que haja sobreposição a blocos vizinhos, que será válida somente se um tile que não estiver vazio corresponder ao bloco que esteja tentando sobrepor, impedindo configurações ruins, como becos sem saída intransitáveis [7]. Os templates descritos na Figura 3 foram definidos por Taylor e Parberry [7].

Figura 3. Templates para os blocos 3x3 no Método 1



Fonte: Autoria própria, 2025

Se a geração de blocos resultar em uma configuração impossível (por exemplo, sem templates viáveis para preencher alguma célula), a tentativa é descartada e reiniciada. Assim, algumas verificações são feitas para garantir que o nível seja jogável, conferindo se há um caminho contínuo por todos os espaços de piso e espaço suficiente para todas as caixas, o jogador e pelo menos um espaço vazio. Caso algum desses critérios não seja atendido, o nível gerado é descartado e outra tentativa é feita [7].

2. Posicionamento dos Objetivos (*Goals*)

Os objetivos (onde as caixas precisam ser empurradas) são posicionados utilizando um método de "força bruta", testando todas as combinações possíveis de posicionamento. Para evitar tempos excessivos de execução, um timer monitora o processo e interrompe a busca se um tempo limite for ultrapassado, retornando a melhor configuração encontrada até aquele momento. Além disso, a ordem de teste

dos objetivos é verificada de forma aleatória para garantir que mesmo interrupções prematuras resultem em uma resposta razoável [7].

3. Determinação do Estado Mais Distante do Objetivo

Com os objetivos posicionados, o próximo passo é determinar o "estado inicial do nível", ou seja, a posição inicial do jogador e das caixas. O critério adotado para essa escolha é encontrar o estado mais distante do estado final, garantindo que o nível apresente um desafio significativo. Para isso, o algoritmo calcula a distância entre diferentes estados utilizando uma métrica específica de movimentação do Sokoban, chamada de *box lines*, que calcula quantas vezes um jogador empurra uma caixa. Porém, se uma caixa for empurrada várias vezes na mesma direção, conta como um único empurrão. Essa métrica foi considerada a mais eficaz, pois se correlaciona bem com a dificuldade real do nível [7].

O principal desafio desse processo é que os algoritmos tradicionais de busca (como BFS ou A*) não funcionam bem nesse contexto devido ao tamanho do espaço de estados. Para contornar isso, foi utilizada uma variação de BFS (*Breadth-first search*), que equilibra memória e desempenho. O processo envolve quatro funções principais:

- *MakeStartSet*: Define as posições iniciais do jogador considerando os espaços livres;
- *Try:* Tenta fazer a busca, e chama *Expand* sempre que precisa aumentar a profundidade.
- *Expand:* Calcula os próximos estados possíveis a partir do estado atual (Aumenta a profundidade).
- *Go:* Executa a busca iterativa em profundidade, chama *MakeStartSet* para gerar o estado inicial, e então chama *Try* para iniciar a busca, até que *Try* falhe, e retorna o estado anterior à falha.

Ao final desse processo, o estado inicial é definido como aquele mais distante do estado final, resultando em um nível que, teoricamente, exige uma solução desafiadora e bem estruturada [7].

Método adaptado para *Witchie*

Para adaptar o método de geração procedural de Taylor e Parberry [7] ao experimento deste estudo, algumas mudanças foram implementadas nos parâmetros de entrada, visando controlar o tempo de processamento e ajustar a complexidade dos níveis gerados. Essas mudanças serão pontuadas a seguir.

1. Tamanhos de Sala e Quantidade de Caixas

Os testes foram conduzidos com dois tamanhos fixos de sala: 6×6 e 9×9. Esse ajuste foi feito para garantir uma variação significativa na complexidade dos níveis sem comprometer a viabilidade da geração. Além disso, foram utilizadas duas configurações de número de caixas, sendo testados níveis com 2 caixas e 3 caixas.

2. Limitação da Profundidade Máxima na Busca Inversa

Para otimizar o tempo de geração, foi adicionada uma restrição no *depth* máximo durante a busca reversa para determinar o estado inicial mais distante. Essa limitação impede que o algoritmo explore excessivamente o espaço de estados, acelerando o processo sem comprometer a qualidade da geração. Os valores definidos para o *depth* máximo foram: para níveis 9×9 com 3 caixas: 20 a 30 passos e para os demais casos: 50 passos.

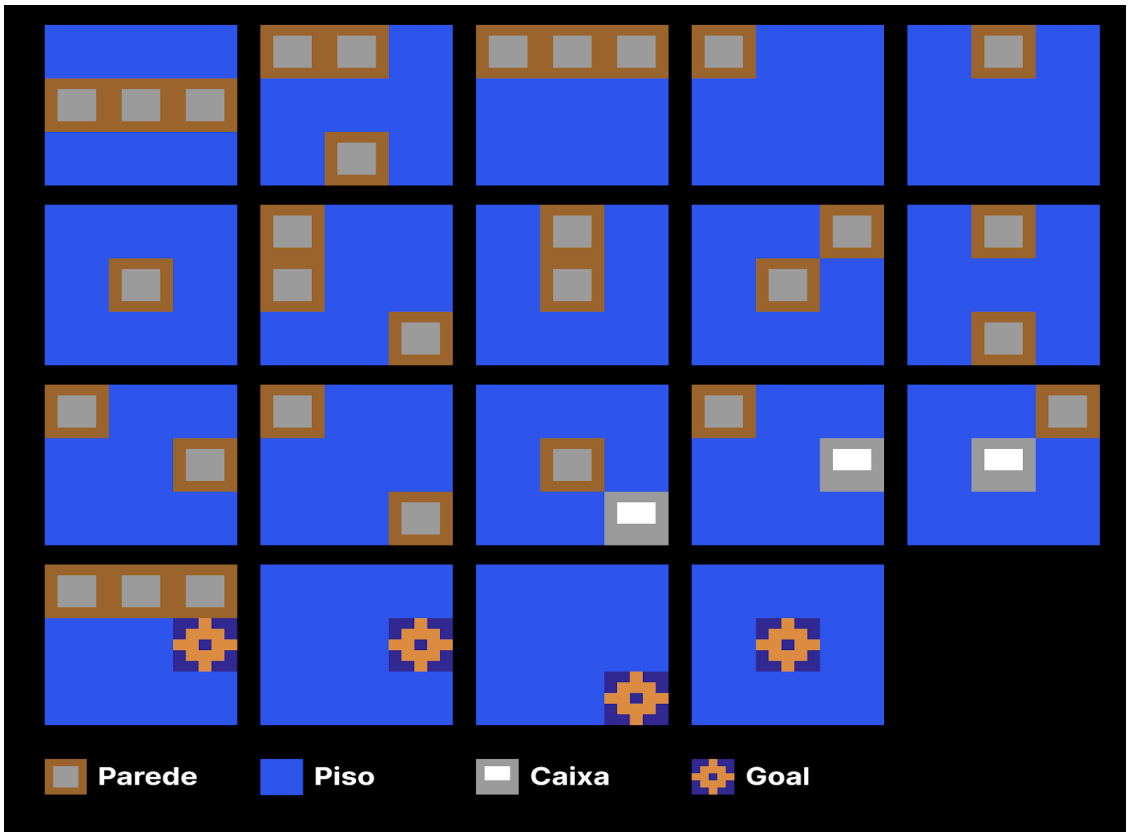
A redução do *depth* nos níveis maiores e mais complexos foi necessária para evitar tempos de execução excessivamente longos. Como a busca reversa cresce exponencialmente com o número de caixas e o tamanho da sala, limitar a profundidade permitiu encontrar um equilíbrio entre tempo de processamento e desafio do nível.

Construção da Sala Vazia com Templates Pré-Configurados

Assim como no primeiro método, a geração do nível começa com a criação de uma sala vazia, utilizando um sistema de templates baseados em blocos 3×3, cercados por tiles vazios, parede ou de pisos. No entanto, diferentemente do método original, os templates que projetamos já incluem as posições dos objetivos (*goals*) e

das caixas. Os templates que contém as caixas, vão contar com uma parede posicionada estrategicamente para servir de apoio, de maneira que permita a caixa ser empurrada (ver Figura 4).

Figura 4. Templates para os blocos 3x3 no Método 2



Fonte: Autoria própria, 2025

Para garantir controle sobre a complexidade e estrutura dos níveis, foi adicionada às seguintes restrições:

- O usuário deve definir previamente um número X de blocos que conterão objetivos e caixas;
- O algoritmo seleciona aleatoriamente esses blocos dentro da sala e os preenche com os templates apropriados;
- O restante do processo de montagem da sala segue as mesmas regras do primeiro método, garantindo conectividade e evitando padrões problemáticos, como blocos de chão cercados por três paredes.

Essa abordagem simplifica a etapa de posicionamento das caixas e objetivos, garantindo que as configurações iniciais respeitem padrões estruturais desejáveis, sem a necessidade de ajustes posteriores.

Como os templates já incluem os objetivos e caixas, não há mais necessidade das etapas de alocação de objetivos por força bruta, ou busca pelo *farthest state*, principais gargalos do processo. Dessa maneira é possível gerar níveis com mais caixas e maiores, sem se preocupar com tempo de execução.

Diferença entre as abordagens

Esse segundo método apresenta algumas vantagens em relação ao primeiro:

- **Maior controle sobre a estrutura dos níveis**, garantindo que os objetivos e caixas sejam posicionados em padrões específicos sem necessidade de ajustes posteriores. O que permite ao usuário ajustar para enquadrar melhor as peculiaridades do seu jogo

- **Redução no tempo de geração**, pois elimina a busca exaustiva por um estado inicial distante, tornando o algoritmo mais eficiente.
- **Facilidade na definição da dificuldade dos níveis**, uma vez que o usuário pode ajustar diretamente o número de objetivos e caixas nos templates.

Por outro lado, essa abordagem pode limitar a diversidade dos níveis gerados, já que os *layouts* dependerão da variedade de templates disponíveis. No entanto, essa limitação pode ser mitigada com um conjunto suficientemente grande de templates, permitindo uma ampla gama de variações.

4. ANÁLISE DOS RESULTADOS

Para avaliar a eficácia das técnicas de PCG aplicadas a *Sokoban* e *Witchie*, cada método foi utilizado para gerar 50 níveis, que passavam por um algoritmo de resolução para verificar se eram solucionáveis ou não. Na fase de definição, os níveis são inicializados e tem as posições de interesse destacadas (Paredes, Caixas e Objetivos), sendo definido o tipo de movimento (*Witchie* ou *Sokoban*), a heurística de prioridade a ser utilizada pelo A* (distância *Manhattan* entre cada Caixa e o Objetivo mais próximo) e um método para verificar *deadlocks* (Caixas em cantos, ou presas verticalmente/horizontalmente). Por fim, na fase de execução é utilizado o algoritmo A* para tentar resolver o nível, retornando o número de *BoxLine* e o caminho utilizado caso o nível seja solucionável, e 0 caso não seja. Após essa checagem, foram feitas duas análises.

4.1 PRIMEIRA ANÁLISE

A primeira análise visa comparar os níveis gerados para *Witchie* e *Sokoban*, a partir das métricas de Solubilidade e Adaptabilidade. A Solubilidade mede o nível de resolutividade antes de receber qualquer modificação, sendo atribuídos 1 ponto para níveis solucionáveis e 0 para os que exigem ajuste. Enquanto a Adaptabilidade quantifica as alterações feitas para tornar um nível solucionável, em que cada alteração é contabilizada.

Para consolidar os resultados, métricas gerais foram calculadas para cada método testado. A Solubilidade Geral é obtida através da média das Solubilidades de cada nível, nos dando um valor equivalente ao percentual de níveis solúveis inicialmente. A Adaptabilidade Geral é a média dos valores normalizados de (100 - Adaptabilidade). Por fim, a Efetividade do Método é calculada como uma média ponderada das duas métricas, os pesos atribuídos foram de 0,7 para Solubilidade e 0,3 para Adaptabilidade. A escolha dos pesos de deu a partir do objetivo principal dos métodos de gerar níveis solúveis, por isso esta obteve o maior peso, seguida pela de Adaptabilidade, pois é a métrica mais abstrata de se quantificar, uma vez que depende da escolha do usuário de como adaptar os níveis. Optamos pelos valores de 0,7 e 0,3 com o objetivo de equilibrar a ênfase entre as duas métricas, o que seria menos balanceado com pesos mais extremos.

Essa primeira análise permite uma comparação quantitativa entre os métodos de PCG testados, possibilitando a identificação das nuances de cada técnica em *Sokoban* e *Witchie*.

Figura 5. Desempenho dos Diferentes Métodos Para Cada Jogo

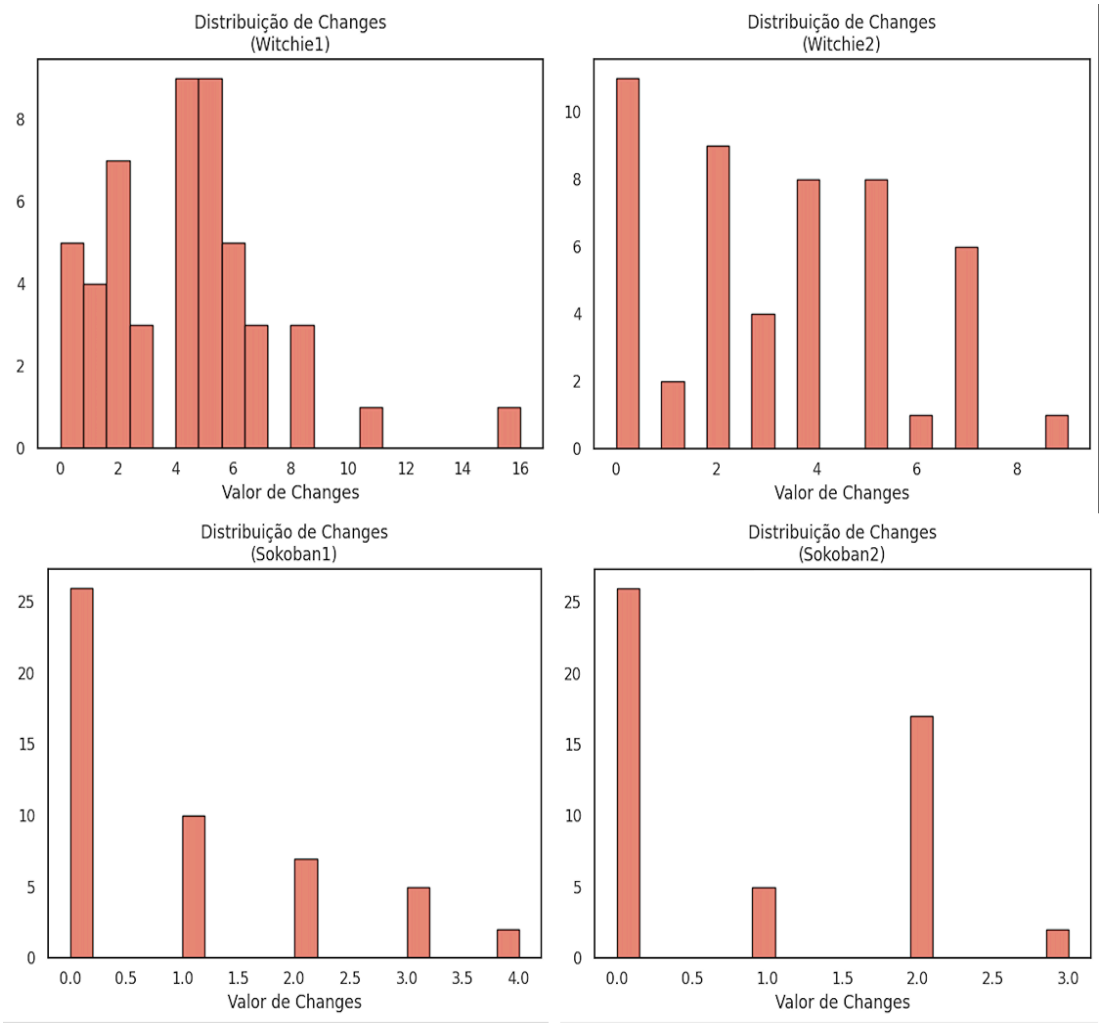
Desempenho dos Diferentes Métodos

Método	Solubilidade	Adaptabilidade	Níveis	Efetividade
Witchie1	14.00%	0.958	50	0.385
Witchie2	22.00%	0.968	50	0.444
Sokoban1	56.00%	0.991	50	0.689
Sokoban2	52.00%	0.991	50	0.661

Fonte: Autoria própria, 2025

Os resultados obtidos (Figura 5) demonstram que o Método 1 apresentou a maior efetividade geral, sendo especialmente eficiente para a geração de níveis de *Sokoban*. No entanto, ao analisar os dados separadamente para cada jogo, observou-se que o Método 2 foi mais adequado para *Witchie*, superando o Método 1 em todos os parâmetros.

Figura 6. Gráfico com as distribuições e valores de *Changes*



Fonte: Autoria própria, 2025

Changes é a variável que contabiliza o número de ajustes feitos para tornar um nível solúvel.. Dessa forma, constatou-se que os níveis gerados para *Sokoban* necessitam de menos ajustes do que os níveis de *Witchie* (Figura 6), o que indica que os métodos de PCG aplicados ao jogo original são mais estáveis e demandam menor intervenção para garantir a solubilidade.

4.2 SEGUNDA ANÁLISE

Em seguida, uma segunda análise foi realizada, focada especificamente no jogo *Witchie*, onde foi gerada uma amostra com 50 níveis para cada método. Nesta análise, quatro usuários, que já haviam jogado *Witchie* previamente, avaliaram a dificuldade de cada um dos 100 níveis de forma independente e responderam a seguinte pergunta na escala Likert [3]: O quão difícil você considerou o nível? (1 =

Trivial, 5 = Muito Difícil). Além de responder a pergunta, os usuários também forneceram a quantidade de vezes que ficaram presos num nível, e precisaram acionar um *reset*, para jogar novamente.

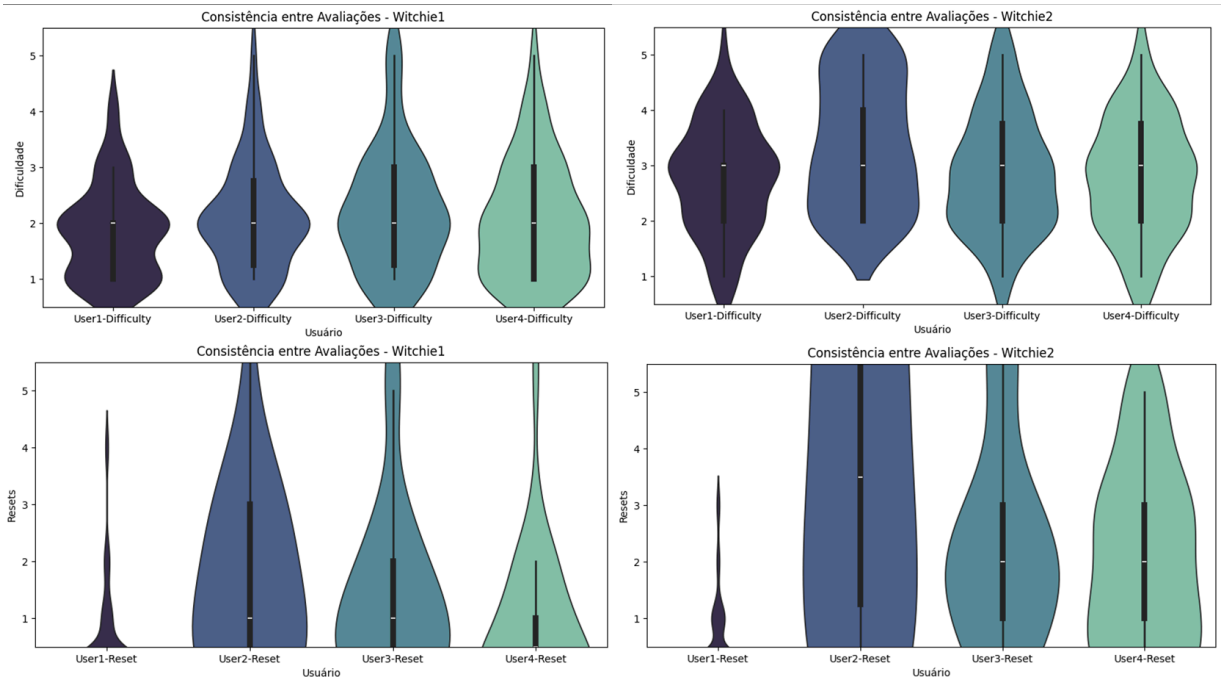
Para essa análise, além das métricas de Solubilidade e Adaptabilidade, foi criada uma nova métrica: Variabilidade. Ela foi proposta com o objetivo de avaliar se o método de PCG é capaz de produzir níveis com diferentes graus de dificuldade. Para isso, considerou-se a média das avaliações dos quatro usuários, essa média é então arredondada para o inteiro mais próximo e é usada para classificar o nível em uma das cinco faixas de dificuldade. Em seguida, calcula-se a distribuição dos níveis entre essas faixas e aplica-se a fórmula da entropia [10] para medir a diversidade dessa distribuição. A entropia real é normalizada pela entropia máxima (associada à distribuição normal), resultando em um valor entre 0 e 1. Quanto mais próximo de 1, maior a Variabilidade dos níveis gerados, o que indica um melhor desempenho do algoritmo no quesito diversidade de desafio.

Nesse caso, a Efetividade do Método é calculada como uma média ponderada das três métricas, os pesos atribuídos foram de 0,4 para Solubilidade, 0,4 para Variabilidade e 0,2 para Adaptabilidade. As justificativas para os pesos das primeiras métricas se mantém, com a adição de Variabilidade com uma importância equivalente a Solubilidade, uma vez que a diversidade na dificuldade dos níveis é de extrema importância para uma boa curva de dificuldade, e, consequentemente, entretenimento do jogador.

Essa abordagem permite uma comparação mais qualitativa entre os métodos de PCG testados, possibilitando a identificação de qual técnica funciona melhor para *Witchie* e aferir melhor a qualidade dos níveis gerados.

Contudo, antes de analisar as métricas de fato, foi feita uma avaliação de consistência entre os usuários.

Figura 7. Gráfico com a Análise de Consistência entre os Usuários



Fonte: Autoria própria, 2025.

Quanto à consistência nos valores de dificuldade atribuídos, observa-se que existe consistência na atribuição das notas. As principais variações foram o Usuário 1 que deu mais notas baixas e quase não teve *resets*, o que se justifica pela maior experiência do jogador em relação aos demais. Já o Usuário 2 apresentou maior dificuldade, dando mais notas altas, compatível com o seu número elevado de *resets*. Outro fator a ser destacado desses gráficos é que o método 2 teve mais níveis classificados com notas altas, indicando um maior grau de dificuldade.

Figura 8. Desempenho dos Diferentes Métodos Para Witchie

Desempenho dos Diferentes Métodos

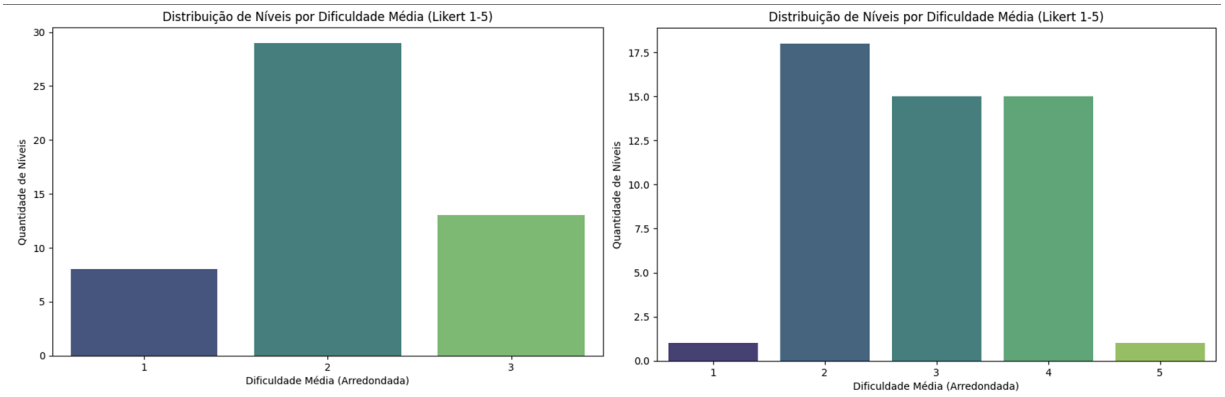
Método	Solubilidade	Variabilidade	Adaptabilidade	Níveis	Efetividade
Witchie1	14.00%	0.596	0.958	50	0.486
Witchie2	22.00%	0.775	0.968	50	0.591

Fonte: Autoria própria, 2025

Os resultados obtidos (Figura 8) demonstram que o método 2 foi mais efetivo para *Witchie*, superando o método 1 em todos os parâmetros. Assim, a Variabilidade destaca-se como principal responsável, apresentando um valor significativamente

maior que o do método 1, indicando bem mais diversidade em relação à dificuldade dos níveis gerados.

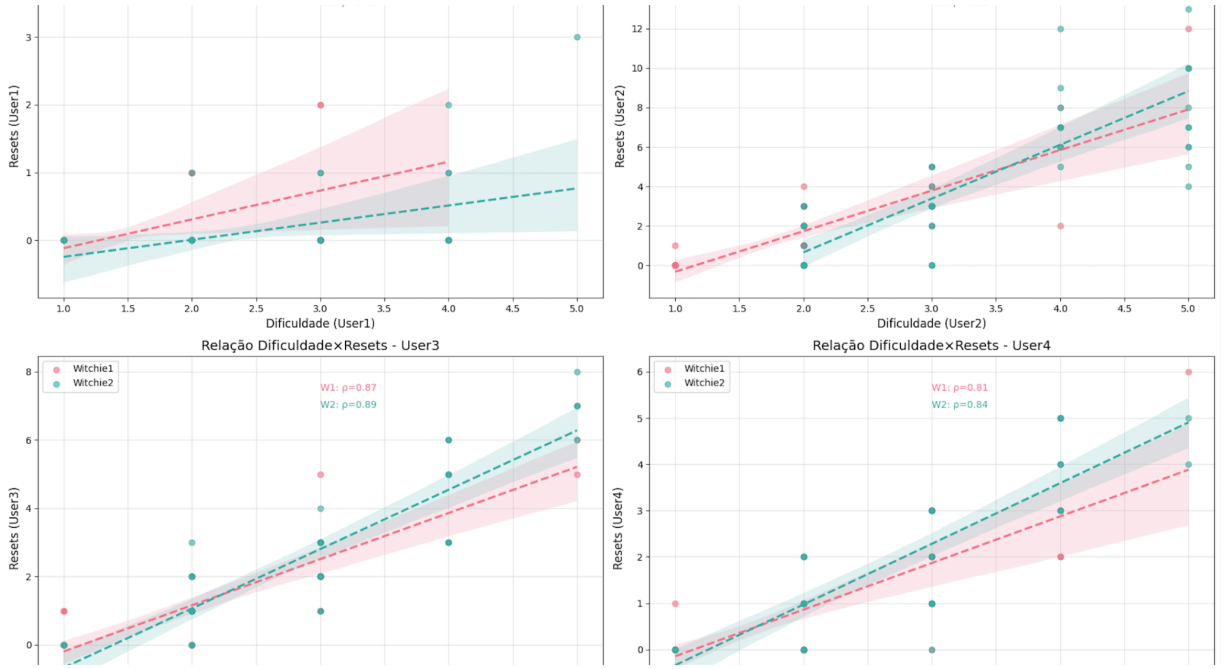
Figura 9. Distribuição dos Níveis por Dificuldade Média



Fonte: Autoria própria, 2025

Quanto à distribuição dos níveis por dificuldade (Figura 9), o método 1 não foi capaz de gerar níveis com dificuldades 4 e 5, demonstrando uma falha grave no quesito de Variabilidade. Já o método 2, apesar de gerar poucos níveis de dificuldades 1 e 5, tem uma distribuição muito mais equilibrada entre níveis fáceis, médios e difíceis, se mantendo consistente quanto ao valor obtido de sua Variabilidade.

Figura 10. Gráfico de Relação entre Dificuldade e *Reset*



Fonte: Autoria própria, 2025

Utilizando a regressão linear (Figura 10) analisamos a relação entre Dificuldade e *Resets*, confirmando, o esperado, de que os usuários tendem a resetar mais em fases mais difíceis. Porém, uma informação interessante é de que o método 1, apesar de ter menos níveis difíceis que o método 2, apresenta maiores taxas de *Reset* em seus níveis fáceis, indicando que tem níveis fáceis mais complexos que os do método 2.

5. CONCLUSÃO

5.1 CONSIDERAÇÕES FINAIS

Diante do que foi analisado, conclui-se que o método 1, baseado na proposta de Taylor e Parberry [7], se mostrou mais adequado para a geração de níveis de Sokoban. Esse método se apoia na construção de salas vazias a partir de templates, no posicionamento posterior dos objetivos e na definição do estado inicial mais distante. Ainda que apresente limitações em termos de escalabilidade, sobretudo quando aplicado a níveis de tamanho maior que 3x3, seu desempenho foi sólido no contexto do Sokoban.

Em contrapartida, o método 2 foi projetado para atender às particularidades do jogo *Witchie*, que possui uma mecânica de movimentação distinta baseada em deslizamento. A adaptação consistiu na utilização de templates que já incluem caixas e objetivos, dispensando a busca reversa pelo estado inicial e reduzindo significativamente o tempo de geração. Com isso, o método 2 apresentou melhor desempenho na geração de níveis para *Witchie*, se destacando por demonstrar uma maior variabilidade de dificuldades e consistência nos níveis gerados, desde que aplicado sobre um conjunto de templates bem elaborados. Dessa forma, se mostra o mais adequado para gerar níveis com uma boa curva de dificuldade para *Witchie*.

Por fim, verifica-se que ambos os métodos são ferramentas úteis para a geração de níveis, especialmente quando empregados como suporte no processo de design. Ao utilizá-los para criar bases bem estruturadas, o usuário pode aprimorar os níveis gerados, tornando-os ainda mais atrativos e desafiadores. Essa abordagem híbrida, combinando geração procedural e refinamento manual, se mostra promissora para o desenvolvimento de experiências de jogo mais ricas e equilibradas.

5.2 CONTRIBUIÇÕES

Este trabalho contribui com uma análise comparativa detalhada entre duas abordagens de Geração Procedural de Conteúdo aplicadas a jogos de lógica e quebra-cabeça. A principal contribuição reside na demonstração de que técnicas estabelecidas para jogos clássicos, como o Sokoban, podem ser adaptadas com

sucesso para jogos que apresentam variações mecânicas relevantes, como o *Witchie.*

Além disso, também contribui com a adaptação bem-sucedida de um método de PCG para o jogo *Witchie*, resultando na construção do método 2, uma abordagem capaz de gerar níveis jogáveis e de dificuldades variadas. Essa adaptação foi feita com base no método de Taylor e Parberry [7], mas incorporando modificações que o tornaram compatível com as especificidades de *Witchie*.

5.3 TRABALHOS FUTUROS

Algumas limitações deste estudo abrem espaço para investigações futuras que poderão aprofundar e expandir os resultados obtidos. Em primeiro lugar, sugere-se a realização de testes com salas de diferentes tamanhos, tanto menores quanto maiores que os utilizados neste experimento, com o objetivo de avaliar como a variação da escala influencia nas métricas de avaliação dos níveis. Além disso, uma expansão no número de templates disponíveis, bem como melhorias na qualidade dos mesmos, pode contribuir diretamente para o aumento da diversidade e riqueza dos níveis gerados, mitigando uma das limitações identificadas do método 2.

Outra direção promissora consiste na aplicação dos métodos de PCG a outros jogos derivados de *Sokoban* que introduzem variações em suas regras, como diferentes tipos de movimentação, obstáculos dinâmicos ou algum outro tipo de mecânica. Explorar essas variações permitiria testar a adaptabilidade e a robustez das abordagens propostas frente a novas dinâmicas.

Por fim, recomenda-se a investigação de outras técnicas de PCG não exploradas neste trabalho, com o intuito de comparar seus resultados e identificar novas soluções para a geração automática de níveis em jogos de quebra-cabeça. Essas possíveis extensões ampliam o escopo do estudo e podem fortalecer ainda mais a relação entre métodos procedurais e o design eficiente de jogos digitais.

REFERÊNCIAS

1. HENDRIKX, Mark et al. Procedural content generation for games: A survey. **ACM Transactions on Multimedia Computing, Communications, and Applications (TOMM)**, v. 9, n. 1, p. 1-22, 2013.

2. KARMAN, S. J. et al. **Generating sokoban levels that are interesting to play using simulation**. 2018. 56f. Dissertação (Mestrado em Game and Media Technology) – Utrecht University, Utrecht, Holanda, 2018.

3. LIKERT, R. **A technique for the measurement of attitudes**. 1932. 55f. Tese de (Doutorado - *New York University,* Nova York, 1932.

4. MELO C. M. P. **Ferramenta de visualização para exploração de uma taxonomia de geração procedural de conteúdo para jogos.** 2022. 48 f. Trabalho de Conclusão de Curso (Graduação em Sistema de Informação) – Universidade Federal de Pernambuco, Recife, 2022.

5. MURASE, Y.; MATSUBARA, H.; HIRAGA, Y.. Automatic making of sokoban problems. In: **PRICAI'96: Topics in Artificial Intelligence: 4th Pacific Rim International Conference on Artificial Intelligence Cairns, Australia, August 26–30, 1996 Proceedings 4**. Springer Berlin Heidelberg, 1996.

6. PUZZLESCRIPT . Puzzlescript: create puzzles. **Puzzlescript**. Disponível em: https://www.puzzlescript.net. Acesso em: 23 dez. 2024.

7. TAYLOR, J.; PARBERRY, I;. Procedural generation of sokoban levels. In: **Proceedings of the International North American Conference on Intelligent Games and Simulation.** 2011. p. 5-12.

8. TODD, G. et al. Level generation through large language models. In: **Proceedings of the 18th International Conference on the Foundations of Digital Games**. 2023.

9. TOGELIUS, J. et al. Geração de conteúdo procedural baseada em busca: Uma taxonomia e pesquisa. **IEEE Transactions on Computational Intelligence and AI in Games**, v. 3, n. 3, p. 172-186, 2011.

10. SEPÚLVEDA-FONTAINE, S. A.; AMIGÓ, J. M. Aplicações da Entropia em Análise de Dados e Aprendizado de Máquina: Uma Revisão. **Entropy** , v. 26, n. 12, p. 1126, 2024.

11. WIKIPEDIA. **Sokoban**. Wikipedia, The Free Encyclopedia. Disponível em: https://en.wikipedia.org/wiki/Sokoban. Acesso em: 23 dez. 2024.