1. Blockout - Codingame

A ideia básica do jogo é mover peças de formatos diferentes conforme elas descem em uma tela, para formar linhas completas, posteriormente ganhando pontos e avançando de nível no jogo. As formas usadas no Tetris são chamadas de *polyominoes*, que são generalizações do dominó. Cada ladrilho é construído colocando vários quadrados de ponta a ponta para criar formas. O jogo termina quando todo o campo de jogo é preenchido e nenhum outro movimento é possível.

Quanto maior se torna um problema de quebra-cabeça, mais difícil é resolvê-lo. A tática tradicional para resolver esses quebra-cabeças usa uma abordagem algorítmica de força bruta da ciência da computação chamada backtracking. O retrocesso cria soluções de forma incremental e, em seguida, descarta aquelas que resolvem o quebra-cabeça ou problema. Essa abordagem costuma consumir muito tempo e usar muitos dados.

Nesta primeira implementação foi utilizado é técnica de força bruta, juntamente com algumas estratégias para tentar alocar a peça no local mais edequado possível. Algumas obserções também são adicionadas como pros e contras.

2. Estratégia utilizada

Para resolver o problema, o algorítimo inicial se basea na técnica de força bruta. O algoritmo tenta primeiramente encontrar todas as possibilidades para preencher o cubo conforme a Figura 1, dado um bloco de entrada com dimensões (x, y, z) conforme a Figura 2.

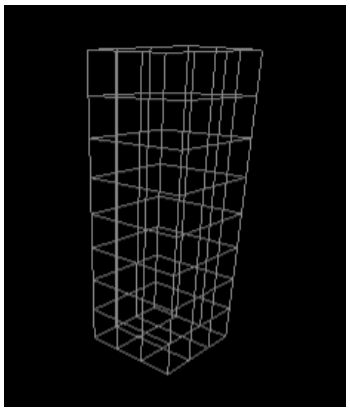

Figure 1. Cubo vazio

Por se tratar de um jogo em 3D, se fez necessário transformar primeiramente o ambiente em 2D, para trabalhar com matrizes. No exemplo da Figura 2, a primeira camada possui dimensões $(3, 3, 1)$, que corresponde a profundidade, largura e altura. Cada camada possui altura um, então para o cubo da figura, teremos 8 camadas, cada uma com dimensão $(3, 3, 1)$.

Para ficar fácil a manipulaçao, transmamos a matriz 3D $(3, 3, 1)$, no formato 2D, ficando assim um vetor $(9, 1)$. Então como resultado teremos uma matriz $(9, 8)$ onde cada linha corresponde a uma camada do cubo.

Utilizamos a matriz resultante para identificar todas as possibilidades de alocar o

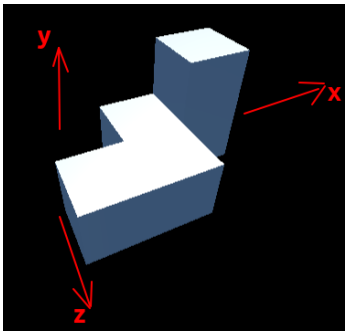

Figure 2. Bloco de entrada

bloco de entrada no cubo, que por sua vez essas possibilidades devem satisfazer a principal restrição (o bloco deve estar inteiramente dentro do cubo).

Uma vez encontrado o conjunto de possibilidade, o algoritmo divide todas em possibilidade em: **possibilidades boas** e **possibidades ruins**.

- As possibilidade boas consistem nos blocos que serão encaixados sem que haja espaço vazio em baixo; como o exemplo da Figura 3, onde o bloco na cor lilás é encaixado sobre o bloco verde sem que hava espaços vazios nas camadas inferiores.

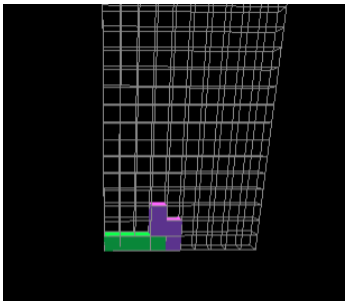

Figure 3. Bloco de entrada

  Não há critério para escolha da possibilidade apartir da lista obtida, sendo assim a primeira é escolhida.
- As possibilidade ruins consistem nos blocos que serão encaixados porém com espaço vazio em baixo; como o exemplo da Figura 4 onde o bloco de cor azul claro é encaixado porém deixando espaços vazios nas camadas inferiores.
  Da mesma forma, aqui também não há critério para escolha da possibilidade apartir da lista obtida, sendo assim a primeira é escolhida.

Apesar do algoritmo seguir o modelo de força bruta, não é feita uma busca recorrente dentro de todas possibilidades. Ao invés disso, é escolhido uma opção dentro do conjunto de possibilidades boas, e assim por diante.

A busca é feita a partir da camada mais inferior, caso não encontre possibilidades boas nela, então eleva-se mais uma camada até que um limite de camadas seja alcançado. Caso ainda não haja boas possibilidades, então uma opção é escolhida dentro do conjunto de possibilidades ruins.

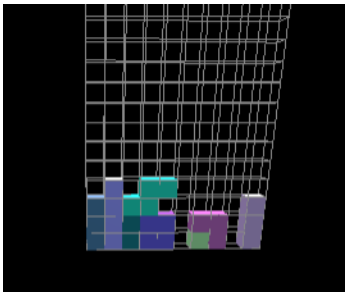

Figure 4. Bloco de entrada

Table 1. Resultados na plataforma codingame

Problema	Quantidade de jogadas	Sucesso
3x3x8 SIMPLE	245	Sim
10x1x20 TETRIS	54	Não
5x5x12 FLAT	35	Não
5x5x12 BASIC	30	Não
5x5x12 EXTENDED	14	Não
3x3x8 FLAT	15	Não
4x4x10 BASIC	10	Não
6x6x6 EXTENDED	2	Não

3. Resultados, prós e contras da estratégia

Após entender o problema, o método de força bruta geralmente é o mais fácil de ser implementado, pois vai consistir apenas em combinar todas as possibilidades e satisfazendo as possíveis restrições do problema. Apenas condicionais, loops, e alguns conceitos de estruturas de dados são necessários nessa abordagens.

Os contras da abordagem é a necessidade de muita memória e tempo computacional caso o problema seja grande, já que será necessário analisar todos os casos. Dependendo dos conjuntos de problemas, a implementação também pode ser tornar um gargalo, já que será necessário analisar muitos casos para implementar um algoritmo que cubra todos os casos.

Como resultado do primeiro experimento, é apresentado a Tabela 1.

Uma vantagem da estratégia adotada, como já mencionado, é uma subdivisão prévia entre boas e más possibilidades. O que reduz o tempo por buscar algo mais apropriado.

O algoritmo ainda precisa saber lida com casos onde haja pontos vazios nas camadas inferiores, assim tentar buscar uma melhor posição nas camadas superiores. Um exemplo desse cenário pode é visto na Figura 5.

4. Outras possíveis estratégias

Outras possíveis estratégias ainda podem ser adotadas. Ao invés de se basear na camada mais baixa, pode-se utilizar uma estratégia que tentar encaixar o bloco em uma camada mais a cima contanto que satisfaça a condição de não possuir espaço vazios nas camadas

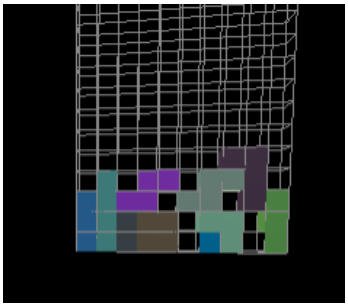

Figure 5. Cenário com espaços vazios nas camadas inferiores

inferiores.

Além disso, em [Gabillon et al. 2013] é apresentado um algoritmo baseado em programação dinâmica para resolver um problema Tetris, em [Burgiel 1997] também apresentado alguns conceitos e teoremas a respeito do problema. Algoritmos evolucionários também podem ser utilizando e em [Langenhoven et al. 2010] um algoritmo baseado em enxame de abelhas é apresentado.

Abordagens ainda relacionadas a aprendizado de máquina e inteligência artificial também podem ser aplicadas. Em [Nilsson 2021] e [Groß et al. 2008] é aplicado apredizando por reforço e técnicas baseadas em redes neurais para a escolha do bloco adequado.

References

Burgiel, H. (1997). How to lose at tetris. *The Mathematical Gazette*, 81(491):194–200.

Gabillon, V., Ghavamzadeh, M., and Scherrer, B. (2013). Approximate dynamic programming finally performs well in the game of tetris. *Advances in neural information processing systems*, 26.

Groß, A., Friedland, J., and Schwenker, F. (2008). Learning to play tetris applying reinforcement learning methods. In *ESANN*, pages 131–136.

Langenhoven, L., van Heerden, W. S., and Engelbrecht, A. P. (2010). Swarm tetris: Applying particle swarm optimization to tetris. In *IEEE Congress on Evolutionary Computation*, pages 1–8. IEEE.

Nilsson, A. (2021). Solving tetris-like puzzles with informed search and machine learning.