# **Implementação de um Agente de Inteligência Artificial para a Resolução do Jogo Sokoban**

**Cesar Rafael Lopes**

Universidade Tecnológica Federal do Paraná – UTFPR

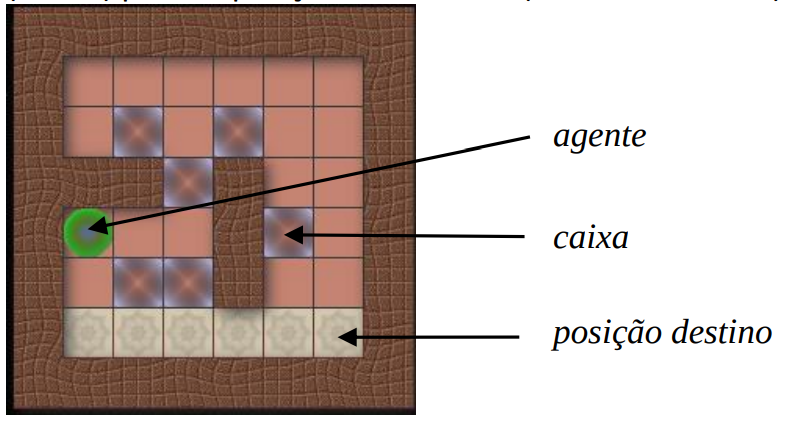
Avenida Sete de Setembro, 3165 - Curitiba/PR, Brasil - CEP 80230-901

ceslop84@gmail.com

**Resumo:** O objetivo deste trabalho é a implementação e análise do jogo Sokoban de forma a buscar uma solução ótima – ou seja, a de menor número de movimentos - empregando um agente de Inteligência Artificial.

# Introdução

Segundo [1] Sokoban (detentor do depósito. em japonês) é um quebra-cabeça de transporte no qual o jogador, ou agente, empurra caixas ao redor de um labirinto, visto de cima, e tenta colocá-las em locais designados, ou objetivos, conforme Figura 1. Apenas uma caixa pode ser empurrada por vez e as caixas não podem ser puxadas.



*Figura 1 - Representação básica de um jogo Sokoban*

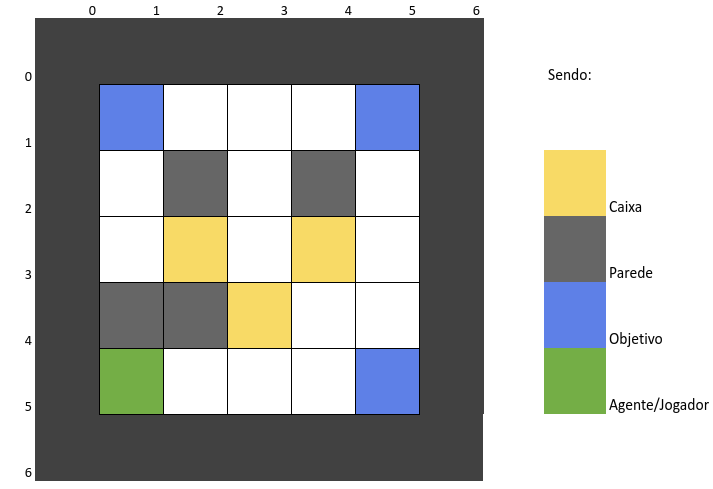
Segundo [2] Sokoban é um problema de espaço de estado em uma grade de labirinto, que é definida por quadrados ocupados por blocos imóveis (paredes) e quadrados livres. Existem k blocos móveis chamados caixas e k quadrados de objetivo. O homem (Sokoban) é um bloco móvel que pode atravessar quadrados livres e empurrar pedras para quadrados livres adjacentes. Uma solução de Sokoban é uma sequência de tais ações que move as pedras de suas posições iniciais para os quadrados de objetivo.

Ainda conforme [2] o Sokoban é um problema desafiador e um dos quebra-cabeças restantes que os humanos resolvem melhor que os computadores. Sokoban um problema do tipo PSPACE-completo [3] e é mais difícil de resolver do que outros problemas conhecidos de pesquisa de agente único, como o cubo mágico (Rubik’s Cube), devido ao seu grande fator de ramificação, maior tamanho da solução, maior tamanho do espaço de pesquisa e uma computação mais complexa do valor heurístico [4]. Características de problemas do mundo real, segundo [2], como a presença de impasses, estados que são mais complexos de representar e gerar e falta de simetria também contribuem para a dificuldade de resolver Sokoban. Por esses fatores a maioria das soluções para o Sokoban se preocupam apenas em encontrar uma solução sem garantias de otimização.

Uma solução ótima é encontrar uma solução que minimize o número de empurrões e movimentos do jogador. Neste trabalho estamos interessados ​​em uma implementação computacional, através de técnicas de inteligência artificial para encontrar uma solução ótima. Além disso. o presente trabalho buscou implementar duas estratégias para a busca da solução ótima visando realizar a comparação entre as mesmas.

# Abordagem e Metodologia

O jogo Sokoban para a finalidade deste trabalho foi definido de 5 x 5 espaços (ou 7 x 7 caso sejam consideradas as paredes externas), conforme exemplo na Figura 2.



*Figura 2 - Exemplo de espaço amostral*

Para tanto foi implementado um agente inteligente baseado em objetivos empregando técnicas de busca informada para a solução do problema, que será definido a seguir.

A modelagem do problema inicia-se a partir do seu estado inicial, Figura 2, sendo que um estado é definido com sendo a configuração composta pela posição em determinado instante do jogador/agente, das caixas e pares. As ações possíveis são *Mover(Direção)* ou *Empurrar(Direção)*, sendo as direções possíveis: norte, sul, leste e oeste. O modelo de transição, ou função sucessora é definido por um estado e uma ação. Por exemplo: *Sucessor(estado\_inicial, Mover(norte))*. O teste de objetivo é realizado verificando se todos os objetivos possuem um caixa sob sua posição. O custo de cada é igual, não havendo diferenciação entre uma ação de mover ou empurrar. A solução é dada pela sequẽncia de ações (mover ou empurrar) necessárias para levar os estado inicial até o estado objetivo, sendo a solução ótima aquela com o menor custo, ou seja, menor número de ações. O ambiente deste agente é categorizado como sendo completamente observável, mono agente, competitivo, determinístico, sequencial, estático e discreto.

O tamanho para o espaço de estados, para o tamanho de labirinto proposto, é composto pelos estados possíveis do jogador e de cada uma das caixas.

A quantidade estimada de estados possíveis para o jogador é igual à:

*Qtde estados jogador: Linhas x Colunas - Qtde de caixas - Qtde de Paredes*

Sendo, para o caso em estudo. igual à ordem de grandeza de:

*Qtde de estados para o jogador: 5 x 5 - 3 - 4 = 18*

A quantidade estimada de estados possíveis para cada caixa é igual à:

*Qtde estados caixas: Linhas x Colunas - (Qtde de caixas -1) - Qtde de Paredes - Jogador*

Sendo, para o caso em estudo. igual à ordem de grandeza de:

*Qtde estados caixas: 5 x 5 - (3 - 1) - 4 - 1 = 18*

Multiplicando as possibilidades para o jogador, para a caixa 1 e para a caixa 2 obtemos o total de estados possíveis para o problema:

*Qtde total estados: 18 \* 18 \* 18 \* 18 = 104.976*

Apenas a título comparativo, a quantidade estimada de estados possíveis para o exemplo da Figura 1 seria de 381.420.488 (27 \* 27 \* 27 \* 27 \* 27 \*27). Adicionalmente, cabe destacar que esses números são aproximados, para permitir uma análise comparativa do tamanho das ordens de grandeza do campo amostral para cada experimento. Os números acima, por exemplo, desconsideram repetições de estados ou ainda configurações infactíveis. Muito provavelmente uma análise mais refinada da quantidade de estados para cada cenário avaliaria os números abaixo daqueles indicados acima.

A seleção da configuração proposta, conforme Figura 2 faz-se necessário pois a eventual adoção da configuração da Figura 1 geraria tal quantidade de estados que inviabilizaria a realização de cálculos e comparações. Para a configuração da Figura 2 o tempo de pŕocessamento é de aproximadamente uma dezena de minutos, já para a Figura 1 as estimativas seriam de algumas centenas de horas. Assim, justifica-se a seleção da presente configuração, conforme Figura 2, sendo uma espaço de 5x5 com 3 caixas e 3 objetivos.

Foi implementada uma busca informada empregando o conceito matemático de função de busca pela melhor escolha A\*.

Assim:

*f(n) = g(n) + h(n)*

Sendo:

*f(n):* função de busca pela melhor escolha ou custo estimado total do caminho para chegar do estado inicial ao objetivo passando por *n;*

*g(n):* Custo do caminho do estado inicial até o nó *n*;

*h(n):* Custo estimado de n ao estado objetivo pelo caminho mais barato, através de uma heurística

Foram empregadas três abordagens para a busca da melhor escolha de estado possível: custo uniforme, menor distância ao objetivo (heurística 1) e graus de bloqueio (heurística 2). Ambas as heurísticas propostas respeitaram os requisitos de otimalidade, pois são admissíveis e consistentes.

Para o custo uniforme, não há distinção de custos para a tomada de decisão, sendo o único fator a profundidade do estado atual. Ou seja, é dada preferência para os nós mais rasos pois estes representam uma menor quantidade de movimentos necessários, e desconsidera a aplicação de uma heurística, ou seja, *h(n)=0*.



No método da menor distância ao objetivo, para cada estado é calculada qual é a menor distância entre cada caixa e cada objetivo, desconsiderando paredes e outras caixas, visando atingir o estado objetivo. Não é calculada a distância Euclidiana mas sim a distância física em termos de movimentos necessários para mover de um ponto ao outro empregando as regras do jogo, movimento incrementais de coluna e linha. Este cálculo também é conhecido como “*distância Manhattan*” [5]. Por exemplo, na Figura 2 a caixa na posição (4, 3), em notação do tipo (linha, coluna), é a mais próxima do objetivo em (5, 5) pois seriam necessários apenas 3 movimentos, sendo 2 em colunas e 1 em linha para chegar até lá, desconsiderando paredes ou outras caixas. Assim, para este estado, deveriam ser calculadas as menores distâncias para os demais objetivos e caixas e então somar todos para obter a menor distância em movimentos entre o estado atual e o estado objetivo. Este método foi selecionado por ser simples de ser calculado, e possuir um significado físico igualmente simples de ser avaliado. O agente buscará selecionar os estados que estiverem mais próximos, em termos de movimentos, da solução desejada. A idéia é que a cada movimento haja uma convergência das caixas para as posições de objetivo, e não uma dispersão. Desta forma, esta heurística irá tentar capturar o estado que estiver convergindo para o estado objetivo, através do menor número de movimentos.

Já no método de graus de bloqueio é realizada uma penalização do cálculo anterior, considerando a quantidade de lados que cada caixa possui bloqueados. Considerando que cada caixa possui 4 lados (norte, sul, leste e oeste), para cada posição que estiver bloqueada, por uma caixa ou parede, é deduzido um grau de liberdade desta caixa. Se os quatro lados estiverem livres, a caixa possuirá 4 graus de liberdade e consequentemente 0 graus de bloqueio. Para o exemplo da Figura 2 a caixa na posição (4, 3) possui 1 grau de bloqueio (uma parede à oeste). Ao calcular um índice entre a quantidade de graus de liberdade possíveis e a quantidade de graus de bloqueio existentes, obtém-se um valor entre 0 e 1 que é então subtraído da menor distância ao objetivo. A idéia por traś da concepção desta heurística é que o jogador deverá sempre escolher o estado que permitir o maior número de movimentos possíveis, evitando desta forma a geração de estados infactíveis (caixa presas ou emperradas). Assim, estados com menores graus de bloqueio são bonificados ao deduzir em até 1 movimento (caso todas as caixas tenham todos os lados livres) a distância mínima até o estado objetivo.

O código fonte com a implementação proposta pode ser consultado no repositório GitHub do projeto [6]. Foram realizados três experimentos. um com cada abordagem, sendo que as seguintes informações foram coletadas para cada execução: tempo de execução, total de movimento, custo *f(n)* da solução, número de nós gerados, número de nós descartados e número de nós de fronteira descartados.

# Resultados

Apresente e compare os resultados de desempenho obtidos pelas estratégias mediante os critérios definidos na metodologia. Apresente as soluções obtidas com os custos.

Segue tabela comparativa com os resultados obtidos para os três experimentos realizados.

|  |  |  |  |
| --- | --- | --- | --- |
| Descrição | Custo uniforme - CUNIF | Menor distância - MDIST | Graus de bloqueio - GBLQ |
| Tempo de execução (s) | 557,0703 | 536,1050 | 488,2545 |
| Total de movimentos | 63 | 63 | 63 |
| Custo *f(n)* da solução | 63 | 63 | 63 |
| Número de nós gerados | 10.785 | 10.364 | 10.353 |
| Número de nós descartados | 12.379 | 11.720 | 11.722 |
| Número de nós de fronteira descartados | 2.656 | 2.467 | 2.448 |

*Tabela 1 - Comparação dos resultados*

Com base nos resultados da Tabela 1 é possível observar que o método rápido foi a aplicação da heurística GBLQ, pois foi a de menor tempo de execução (12,35% mais rápido que CUNIF e 8,93% mais rápido que MDIST). Este método também foi aquele que gerou o menor número de nós (4,01% menor nós que CUNIF e 0,11% menor que MDIST).

A heurística GBLQ foi igualmente eficiente no quesito quantidades de nós descartados e nós de fronteira descartados também foram inferiores ou similares em desempenho que os demais métodos. Para a quantidade de nós descartados o método GBLQ foi 5,31% mais eficiente que CUNIF e 0,02% pior que MDIST (este é o único quesito que o método não performou melhor que os demais), e para a quantidade de nós de fronteira descartados o método GBLQ foi 7,83% mais eficiente que CUNIF e 0,77% mais performático que MDIST.

# Conclusão

Com base nos resultados da Tabela 1 é possível observar que o método mais eficiente foi a aplicação da heurística GBLQ, pois foi o método de execução mais rápido e de menor alocação de memória, pois gerou um quantidade menor de nós. Além disso, o bom desempenho deste método para os quesitos nós descartados e nós de fronteira descartados indicam que foi o método com a melhor heurística de busca, dentre aquelas testadas neste trabalho, para encontrar a solução ótima para o problema proposto. Ou seja, descartando menos nós indica que os estados sucessores gerados convergiam a cada iteração para a solução ótima, de forma mais eficaz que os demais métodos.

Muito embora a heurística GBLQ tenha apresentado um melhor desempenho em termos gerais, o seu desempenho em relação à MDIST é muito próximo em diversos quesitos. Isso se deve ao fato de que, por definição, ambas estão correlacionadas, sendo GBLQ na verdade uma pequena melhoria em relação à MDIST. Todavia, mesmo que as diferenças em quantidade de nós tenha sido relativamente pequena entre GBLQ e MDIST (apenas 0,11%) o resultado em tempo de processamento é longe de ser insignificante (8,93%).

Por fim, destaca-se que este trabalho concentrou-se num universos amostral bastante reduzido, e que os resultados podem ser comportar de forma adversa caso a quantidade amostral seja expandida. A bonificação por graus de bloqueio pode eventualmente se comportar de forma diferente quando ou a configuração das paredes crescer em complexidade ou quantidade de caxas crescer significativamente. Isso por que ao “premiar” o estado que possui mais graus de liberdade o agente pode estar sendo induzido para uma solução não otimizada. Contudo, para a avaliação deste comportamento novos experimentos devem ser considerados.

# Referências

[1] <http://game-sokoban.com> Visitado em 23/05/2020 às 14h53.

[2] Optimal Sokoban solving using pattern databases with specific domain knowledge. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0004370215000867> Visitado em 23/05/2020 às 15h08.

[3] Sokoban is PSPACE-complete. Disponível em: <https://pdfs.semanticscholar.org/7a73/f74c2943e5aafef364735302a36ee2f17b26.pdf?_ga=2.253989454.791335611.1590257833-944032958.1590257833> Visitado em 23/05/2020 às 15h20.

[4] Sokoban: Enhancing general single-agent search methods using domain knowledge. Disponível em: <https://reader.elsevier.com/reader/sd/pii/S0004370201001096?token=65D79FDEA2C495D545E0DF44ECAA8396CE1F9241280745DD2E2EDFB55A750DEACC632523FA2728E56C5335E20F1B4274> Visitado em 23/05/2020 às 15h25.

[5] Using an Algorithm Portfolio to Solve Sokoban. Disponível em: <https://baldur.iti.kit.edu/theses/SokobanPortfolio.pdf> Visitado em 26/05/2020 às 10h38.

[6] Repositório GitHub para o projeto. Disponível em: <https://github.com/ceslop84/ia_skb> Visitado em 26/05/2020 às 10h27.