Inteligência Artificial

Diogo Nuno Pereira Gomes
Luís Filipe de Seabra Lopes

Sokoban Solver

Hugo Paiva, 93195
Carolina Araújo, 93248

DETI
Universidade de Aveiro
11-12-2020

1 Introdução

Este trabalho prático foi desenvolvido com o objetivo de implementar um **agente inteligente** capaz de resolver diversos níveis do jogo **Sokoban**, sem necessitar de conhecimento *a priori* dos mesmos.

2 Arquitetura

A solução desenvolvida utiliza o *solver* de problemas do Standford Research Institute (**STRIPS**), que é uma técnica de planeamento automático, descrevendo primeiro o mundo. Isto é feito fornecendo objetos, ações, pré-condições e efeitos.

A imagem seguinte fornece uma ideia geral de toda a implementação:

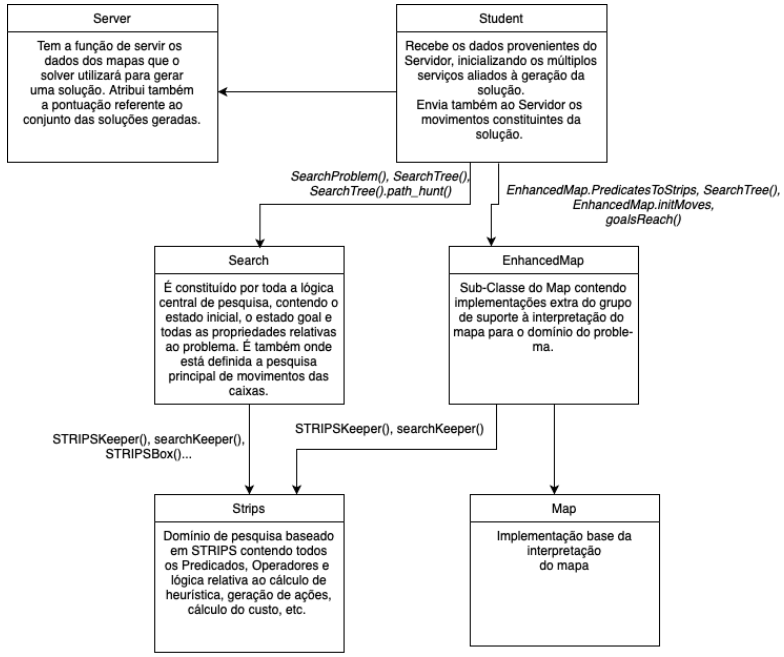

Figure 1: Arquitetura do trabalho prático desenvolvido

3 Sokoban Solver

Inicialmente o grupo debruçou-se sobre a pesquisa de quais os movimentos possíveis em cada posição do mapa, bem como a pesquisa de um caminho desde a posição do *keeper* até uma determinada caixa. Com isto foi possível avançar o suficiente para se chegar a níveis onde não se encontrava a tempo uma solução, visto que muitas das possibilidades que estavam a ser exploradas entravam em situação de *deadlock*. O próximo passo foi, então, prevenir o contínuo estudo de jogadas impossíveis, pelo que se implementaram funções para detetar *deadlocks*, como *simple deadlocks*, *freeze deadlocks* e *dead square deadlocks*.

Com isto, foi possível atingir patamares muito mais elevados, quando os algoritmos de deteção de *deadlocks* estavam já aprumados, no entanto faltava otimização. A título de exemplo, um dos erros, por assim dizer, cometido

a ínicio, e que estava a atrasar bastante o processo de pesquisa, era o facto apenas se estar a verificar a existência de um determinado nó em todos os seus nós superiores até à raiz. Ao fazer isto, estavam-se a calcular múltiplos estados já calculados previamente. A solução foi utilizar um *set* para guardar todos os estados já criados.

Algo também muito batalhado foi a escolha da heurística para o cálculo estimado do tamanho do caminho desde uma caixa até um *goal*. Várias implementações foram exploradas, uma delas envolvendo a própria pesquisa de um caminho desde a caixa até um goal, o que demorava demasiado tempo, pelo que o grupo acabou por escolher outra implementação. Aqui considerava-se que eram prioritários os estados onde mais caixas se encontrassem já em *goals*, bem como os estados onde as caixas já se encontravam na mesma linha ou coluna que os *goals*. Quanto à heurística usada na pesquisa dos caminhos das caixas até aos *goals*, escolheu-se uma simples que calculava a distância de Manhattan, em vez da distância Euclidiana, visto que a primeira, embora não tenha em consideração paredes e caixas que possam estar no caminho, e, por isso, provocar desvios, é mais realista do que a distância linear entre os pontos, visto que o *keeper* nem pode andar na diagonal.

É de notar que a explicação de todos os algoritmos se encontra em forma de comentário no código, facilitando não só a leitura do mesmo mas também o raciocínio por detrás de algumas das escolhas feitas.

4 Estratégia de Validação da Solução

Em termos de validação da solução, eram realizadas propostas de algoritmos que facilitassem a pesquisa e colocava-se a correr o programa, por vezes num nível em concreto, e, com base na medição do tempo que cada parte do código demorava, era possível chegar a diferentes conclusões, *i.e*, havia uma melhor perceção de qual o excerto de código que poderia estar a causar problemas. Com base nisto, eram realizadas otimizações, como por exemplo garantir que apenas era realizada a quantidade estritamente necessária de ciclos *for* e *while*, que não eram realizadas chamadas a funções fundamentais e que pudessem ser substituídas por outra solução mais direta, *etc*.

Usava-se, também, a tabela fornecida pelos docentes como forma de entender onde o grupo se encontrava em comparação com outros grupos, e, portanto, o quão próximos estariamos de uma solução plausível.

4.0.1 Estatística

Com o objetivo de analisar os resultados do trabalho prático, foram gerados dados relativos ao processamento de cada solução de modo a possibilitar a criação de gráficos. A implementação desta geração de dados encontra-se na *branch generate_graph_data*.

Apesar de os dados terem sido gerados, devido à escassez de tempo, não foi possível criar visualizações dos mesmos. Apesar disto, os dados encontram-se na pasta *stats* do repositório.

5 Conclusão

Em tom de remate, de acordo com as metas especificadas pelos docentes, pensa-se que o trabalho foi bem sucedido. Uma das dificuldades mais sentidas pelo grupo foi perceber o que podia estar a impedir que fosse possível encontrar uma solução antes de se ocorrer *timeout* e quais os *deadlocks*, ou outros pequenos pormenores, que podiam estar a interferir com o tempo que demorava a pesquisar por uma solução. Pensa-se ter alcançado bons resultados, visto que é possível passar com sucesso todos os níveis até ao 130, inclusive.

Considera-se que este trabalho prático ajudou a aprofundar conhecimento prévio sobre estruturas de dados e quais os tipos de pesquisa que é possível realizar sobre as mesmas, nomeadamente em árvores, bem como adquirir novos conhecimentos relacionados.

Todavia, o grupo considera que, embora se considere que tenham sido obtidos bons resultados, a primeira entrega não correu como esperado, sendo que se desejava ter conseguido solucionar mais níveis. Ainda assim, pensa-se que a evolução entre a primeira e a segunda entrega foi notável e já demonstrou melhor o empenho e tempo que foram colocados no trabalho.

6 Referências

Foi interpretada bastante informação relativa aos problemas e métodos de solução do *Sokoban* através de artigos e relatórios de estudos encontrados na internet. No entanto, para toda a implementação mais importante foi usada como base de pesquisa a *Sokoban Wiki*.

A nível de otimização de código, consoante o que o grupo verificava que estava a usar mais tempo de processamento foram sendo pesquisadas diversas soluções. A implementação de *Heap* em *Python* é um dos exemplos do resultado destas pesquisas.

Foram debatidas ideias com o grupo do aluno Lucas Botto 93019, e Rui Fernandes 92952.