SOKOBAN – IIA

João Gameiro – 93097
DETI, UA Dezembro 2020

Organização do código

- Student.py
  Comunicação com o servidor, inicialização dos niveis e do agente.

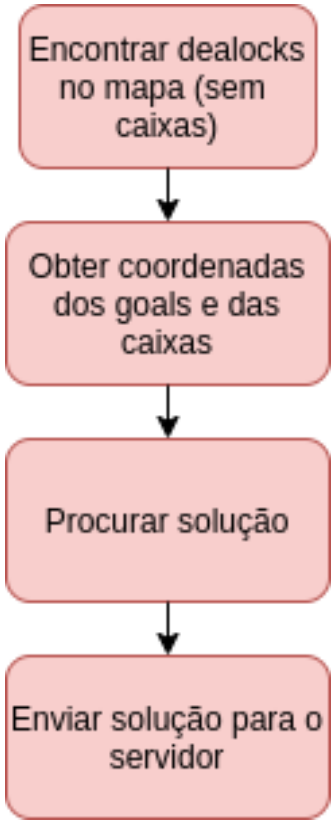
- StudentFunctions.py
  Funções auxiliares para auxiliar o processamento e resolução de um mapa:
  - dead_lock, para verificar se uma caixa vai ser empurrada para um deadlock.
  - deadlock_finder, para procurar todos os dealocks do mapa (sem caixas) no inicio do jogo.
  - dead_lock_line_finder, para procurar linhas de deadlocks (função auxiliar a ser usada dentro da função deadlock_finder).
  - has_obstacle, para verificar se existe algum obstáculo (parede ou caixa) na direção do movimento.
  - get_directions, dada uma lista de coordenadas, devolve uma lista de direções.
  - move_box, para mover uma caixa na direcção indicada.
  - in_goal, para verificar se uma caixa vai ser movida para um goal.
  - can_move, para verificar quais as direções em que uma caixa se pode mover (cima, baixo, esquerda, direita).

- TreeSearch.py
  Pesquisa em árvore para encontrar o caminho mais curto entre duas posições utilizando pesquisa a*.
  Um nó nesta árvore é caracterizado pelos atributos: state (posição em coordenadas), parent e cost (custo associado ao nó).

- TreeSolver.py
  Pesquisa em árvore para encontrar a solução do mapa.
  Fortemente baseado no anterior, diferindo em alguns métodos para suportar a nova pesquisa e forma de representação de dados.
  Um estado é caracteriza por uma lista de caixas, posição do keeper, caminho percorrido pelo keeper uma matriz de números aleatórios para implementação de uma função de hash.

Estrutura do código

Estrutura Geral



Deadlocks

Antes de se proceder à pesquisa, são encontrados todos os deadlocks do mapa e guardados numa lista. Durante a pesquisa são também detetadas situações adicionais de deadlocks que envolvem caixas que não estão presentes na lista de deadlocks.

Procura de uma solução

É usado o algoritmo de pesquisa em árvore. Considera-se que o nó inicial é constituído pelo keeper e pelas caixas nas suas posições inicias, e a partir deste ocorrem sucessivas expansões até ser encontrada a solução.

Expansão de um nó

Para cada caixa presente na lista de caixas são analisadas as direções nas quais a mesma se pode mover (Up, Down, Left, Right). Se a posição destino for acessível pelo keeper e não for um dealock, ou uma parede ou outra caixa, é criado um novo estado com a nova posição do keeper, a atualização do seu path e uma nova lista de caixas em que a posição da caixa em questão foi atualizada.

Heurística e custo

No caso do TreeSolver, a heurística consiste no cálculo de uma atribuição de cada caixa a cada goal, de modo a que a soma das distâncias (valor que vai ser retornado) entre as caixas e os goals que lhe foram atribuídos seja a menor possível.
O custo é dado pelo número de passos dados pelo keeper até ao momento.
No caso do TreeSearch a heurística é representada pela distância de manhattan entre a posição atual do keeper e o objetivo.

Resultados

Os resultados apresentados na tabela seguinte refletem o desempenho do agente até ao nível 106 (nível máximo que o agente consegui resolver).

A*			
Número total de Nós	Tempo total	Número médio de nós p/nível	Tempo médio p/ nível
11123	510.74990	1083.41904	4.8642848
Número total de passos	10916		
Greedy			
Número total de Nós	Tempo total	Número médio de nós p/nível	Tempo médio p/ nível
110200	348.00453	1049.52381	3.31432884
Número total de passos	12321		

Como podemos observar pelos resultados obtidos a pesquisa Greedy encontra uma solução em menos tempo, e expandindo um número menor de nós. A pesquisa A* demora mais tempo e expande mais nós no entanto a solução encontrada por esta técnica é melhor pois o número de passos é menor, contrariamente à Greedy que apresenta um número de passos mais elevado.

Conclusão

Optimizações que ficaram por implementar:

- Melhorar a detação de deadlocks para que o número de estados gerados seja menor
- Implentar a deteção de corrals (áreas às quais o keeper não consegue chegar).

Optimizações efectuadas que tiveram impacto positivo:

- Verificação de estados repetidos;
- Optimização da heuristica e criação da função de hash;
- Optimização da deteção de deadlocks (line_deadlocks e dealocks entre caixas);
- Utilização de sets e de uma heap, para aumentar a velocidade da pesquisa.

Consideração final:

Considero que este foi um trabalho bastante interessante e que contribuiu positivamente para a valorização do meu percurso académico e conhecimento introdutório na área da inteligência artificial.

Principal Website consultado durante o desenvolvimento do código

http://www.sokobano.de/wiki/index.php?title=Solver