Labirinto Tesourado

Hendersson Jairus Mendes Silva e Yogi Nam de Souza Barbosa

Descrição

Esta é a apresentação do domínio Labirinto Tesourado desenvolvido na disciplina "Fundamentos Lógicos em Inteligência Artificial" no segundo semestre de 2023. No Labirinto Tesourado o objetivo é saquear todos os tesouros do labirinto e achar a saída dele.

Introdução

O Labirinto Tesourado é baseado no gênero de jogos de labirinto, nos quais você parte de um ponto inicial e tem que achar a saída do labirinto, embora que aqui não é tão simples assim. Para início de conversa como alguém chega em um local desse tipo? Obviamente chegamos aqui porque estamos procurando tesouros valiosos e todo mundo sabe que labirintos são ótimos para achar uns tesouros brilhantes. No nosso labirinto temos, além dos tesouros (que vamos pegar todos), a possibilidade de achar uma picareta que nos permite destruir as paredes do labirinto, uma rara ferramenta, mas muito útil, ela não se desgasta então basta conseguir uma vez e podemos quebrar quantas paredes quiser. Temos também alguns buracos no decorrer desse percurso, que não conseguimos nem os pular pois o teto é bem baixo, para atravessá-los é necessário achar madeiras para tampar esses obstáculos, madeiras são gastas assim que usamos nos buracos. Além disso, chegamos aqui caindo do andar de cima, então nosso ponto inicial pode ser em qualquer lugar do labirinto, e nossa única saída é um portal mágico que também aparece em qualquer lugar.

Domínio

O domínio requer *:strips e :negative-preconditions* em PDDL, e *:hierarchy, :negative-preconditions, :method-preconditions* e *:universal-preconditions* em HDDL.

Predicados (sem parâmetros)

(em) representa a nossa posição, a posição da saída se dá por (saida). (livre) diz que a posição não é uma parede, mas pode conter objetos ou até mesmo um buraco. (buraco) diz que nessa posição há um buraco, (adjacente) representa que duas posições estão lado a lado, e, portanto, é possível andar de uma para a outra se forem livres. (tesouro) indica a posição de um tesouro, (madeira) a posição de uma madeira, (picareta) a posição de uma picareta, além disso, (tem-picareta) diz que já pegamos uma picareta. Em PDDL apesar do predicado (saida) estar presente, não o utilizamos, e temos um predicado (tem-madeira) para indicar que temos uma madeira devido a não apresentar uma ordem de ações.

Ações

A ação que mais utilizaremos é a (andar), com ela, se o destino for ao lado, livre e sem buracos conseguimos nos deslocar a ele. As ações de coleta são (pegar-tesouro), (pegar-madeira) e (pegar-picareta), são muito similares, e elas pegam o respectivo recurso se estivermos numa posição que tem esse recurso.

Tasks* e *Methods

Inspirado no domínio *Snake* por Mau Magnaguano, nossas *tasks* e *methods* tem uma estrutura recursiva com casos bases e casos recursivos.

A principal *task* é a (sair), ela é completada ou se já estivermos na saída (sair-b), ou ao pegar todos os tesouros e ir para a saída (sair-r). Para pegar todos os tesouros, temos a *task* (pegar-tesouros) que é completada ou se já pegamos todos os tesouros (pegar-tesouros-b), ou com o caso recursivo (pegar-tesouros-r) que consiste em localizar um tesouro, ir para ele, o pegar, e chamar (pegar-tesouros) para pegar os outros tesouros. Para ir aos lugares temos a *task* (ir-para), a qual é completada ou se já estivermos aonde queremos ir (ir-para-b), ou com os métodos (ir-para-n-r), com n indo de 0 a 3. Basicamente esses métodos são as diferentes formas de irmos ao destino. O primeiro é se não tiver buracos e a passagem está livre, bastando andar uma posição e fazer a chamada recursiva. O segundo, é se temos uma picareta, assim podemos ir destruindo as paredes. O terceiro é se não temos uma picareta, mas localizamos uma, assim iremos pegar a picareta no processo. Já o quarto é se houver um buraco no caminho, se tiver vamos localizar uma madeira, pegar ela, tampar o buraco e atravessar.

Problema

Nosso mapa é representado por células (p**y**-**x**), **y** sendo a linha e **x** a coluna, as células são nossos únicos objetos.

Em HDDL nosso objetivo é completar (sair), portanto a presença do predicado (saida) é de extrema importância, já em PDDL, o objetivo é dado sem o uso desse predicado.

O mapa é representado por uma matriz retangular, e as relações de adjacência entre células são bi direcionadas, apesar de não ser obrigatório.

Temos tesouros, madeiras, picaretas e buracos espalhados com quantidades arbitrárias pelo mapa, é possível acontecer mais de um plano para um problema e planos não solucionáveis, por exemplo se não houver acesso a madeiras e tiver um buraco entre a saída.

Posições com tesouros, madeiras e picaretas são posições livres, então é possível atravessar sem coletar. A posição dos buracos também é livre, apesar de que precondições não permitem atravessar buracos. A posição que estamos e a saída também são livres.

Gerador de Problemas

Nosso gerador de problemas foi feito em *Python* e recebe como parâmetros o comprimento e uma altura do labirinto, além de uma *flag* indicando se é um problema *PDDL* ou *HDDL*.

Com base nesses parâmetros ele irá criar um labirinto, a princípio solucionável, a partir do gerador de labirintos MMaze escrito também em Python, criado por MorvanZhou.

A partir daí, serão escolhidas posições aleatórias para a posição inicial, saída, além da quantidade de tesouros (quantidade máxima possível é igual à medida do comprimento), quantidade de buracos (quantidade máxima possível é igual à um terço da medida da altura arredondado

para baixo), além disso são gerados a mesma quantidade de madeiras, e até duas picaretas.

As posições também são escolhidas aleatoriamente entre as livres.

Por fim, teremos um arquivo de problema com a extensão escolhida a partir da flag, nomeada pb**w**-**h**-**t**-**b**.***ddl**, sendo **w**, **h**, **t** e **b** correspondente ao comprimento, altura, quantidade de tesouros e quantidade de buracos respectivamente.

É possível gerar problemas não solucionáveis! (devido à aleatoriedade do posicionamento dos buracos)

Podemos criar problemas retangulares.

Exemplo

python gerador.py 3 3 –hddl

Saída

pb3-3-2-0.hddl

Visualizador

O visualizador foi feito em *Python* com base no que tínhamos feito para o *Snake*, e recebe como parâmetros um arquivo de problema e o plano correspondente, a princípio ele só aceita planos com a formatação do planejador pandaPI, mas pode ser facilmente alterado.

Consegue visualizar planos com problemas retangulares.

Ele irá mostrar através da interface do *pygame* uma representação da solução encontrada, com a movimentação correspondente e ações realizadas.

Exemplo

python visualizador.py pb3-3-2-0.hddl plan.hddl

Saída: (print estático, no visualizador é possível ver a movimentação)

Ainda tem algumas coisas que podem ser melhoradas no visualizador, como por exemplo, ao andar por um recurso, o visualizador vai fazer o recurso desaparecer mesmo que não tenha sido pego, mas dá para ter uma ideia de como o plano se comportou.

Dificuldades

Dentre as diversas dificuldades que encontramos, estavam amarrar as relações entre *tasks* e *methods*, a organização de pensamento e do próprio código. A sintaxe nova, além das regras de métodos (por exemplo, um método sozinho consegue completar uma *task*, sem que os outros tenham completado)> A lógica de ordenação das *subtasks*, já que nosso domínio é ordenado, além de que acredito que não exista um depurador para identificar problemas na execução.

Benchmarks

Madagascar (Binário M, padrão)

Problema	Memória	Ações	Tempo (s)
pb3-3-1-0	347.00 MB	4	0.00
pb5-5-0-0	425.00 MB	11	0.00
pb8-8-5-1	533.00 MB	20	0.03
pb10-10-0-3	547.00 MB	12	0.09
pb10-10-5-3	1.065 GB	46	0.25
pb12-12-4-0	0.852 GB	54	0.33
pb15-8-14-2	1.465 GB	120	0.77
pb15-15-5-2	1.451 GB	72	2.09
pb20-20-7-6	2.106 GB	141	12.72
pb30-30-18-9	+8.214 GB	-	+3000.00

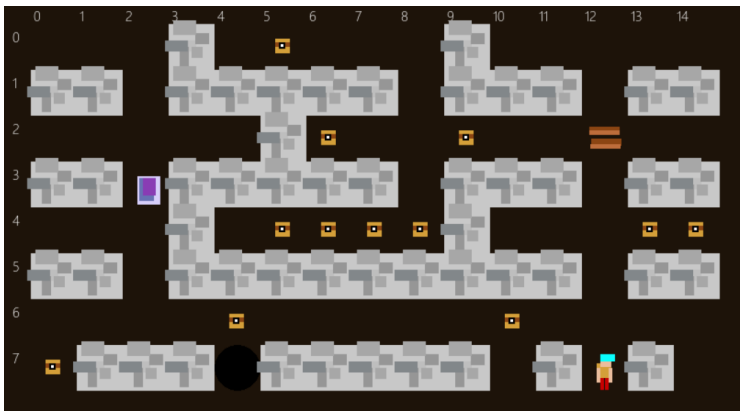
PandaPI (padrão)

Problema	Memória	Ações	Tempo (s)
pb3-3-1-0	-	4	0.07
pb5-5-0-0	-	7	0.07
pb8-8-5-1	-	35	0.19
pb10-10-0-3	-	12	1.26
pb10-10-5-3	-	-	-
pb12-12-4-0	-	64	60.00
pb15-8-14-2		-	-
pb15-15-5-2	-	-	-
pb20-20-7-6	-	-	-
pb30-30-18-9	-	-	-

Referências

MAGNAGUANO, Mau. Snake Domain. Disponível em: https://github.com/Maumagnaguagno/Snake

ZHOU, Morvan. MMaze. Disponível em: https://github.com/MorvanZhou/mmaze



pb15-8-12-1 (exemplo)