Rush Hour

Group Assignment

Inteligência Artificial

Trabalho realizado por:

- Leonardo Almeida 102536
- Pedro Rodrigues 102778

Docentes:

- Diogo Gomes
- Luís Seabra Lopes

Aveiro, 9 de dezembro de 2022

Estrutura

O nosso agente está dividido em 3 entidades principais:

- KeyGenerator.py
- RandomCounter.py
- Agent.py

KeyGenerator:

- Serve para gerar a próxima key a ser enviada para o server.

RandomCounter:

- Serve para corrigir o caminho sempre que existir um movimento random.

Agent:

- É responsável por gerir as duas entidades supracitadas e executar a pesquisa para encontrar o caminho.

Pesquisa

Objetivo:

- Encontrar um caminho de modo a chegar à solução, no nosso caso este é uma lista de tuplos, onde estes são (‘letra da peça’, ‘direção’).
- Como a cada 100ms perdemos 1 ponto, é do nosso interesse fazer o número mínimo de movimentos do cursor, de modo a perder menos pontos possíveis.

Abordagens:

- Pesquisa em largura, é um método rápido para tabuleiros de 6x6, mas apenas devolve o número mínimo de carros a mover.
- Pesquisa Uniforme ou A*, estes métodos permitem obter o número mínimo de movimentos do cursor, mas neste caso é necessário considerar tabuleiros iguais com o cursor em sítios diferentes como estados diferentes.

A nossa abordagem

Ao considerar estados diferentes como referido anteriormente, são explorados muitos nós, o que fica bastante lento, mesmo usando programação dinâmica.

Por isso decidimos considerar estados diferentes apenas para tabuleiros diferentes, mas na função de custo usamos o cursor. Desta forma não obtemos o custo mínimo, mas um valor perto disso que é compensado pela velocidade ganha.

Portanto usámos pesquisa uniforme para níveis 6x6 e A* para níveis maiores.

Heurística usada em A*:

- *Número de carros que bloqueiam o carro A, tendo em conta se estes se encontram bloqueados por outros ou não.* Ou seja, se um carro que bloqueia A estiver completamente bloqueado, este terá um peso superior (igual a 2). No caso de um carro que bloqueie A tenha pelo menos um caminho livre, este terá um peso menor (igual a 1). A heurística corresponde à soma total destes valores.

Foram testadas outras heurísticas, no entanto sem sucesso, como por exemplo a distância do carro A à saída, o espaço livre ao redor de cada carro, o total de carros que bloqueiam o carro A, entre outras.

Algoritmo de pesquisa (detalhes)

De modo aumentar a velocidade do algoritmo de pesquisa criamos a classe Node.py, que é responsável por fazer a expansão da árvore.

Cada node tem um ponteiro para o pai, tabuleiro, peças, ação, custo, heurística e cursor.

De modo a poder comparar tabuleiros (para não repetir estados), decidimos usar Strings e para verificar estados usamos um dicionário onde a Key é a String to tabuleiro e o valor é o custo.

Como cada node filho precisa das peças (sendo estas uma lista de tuplos), o node filho apenas gera a peça que mudou e as restantes são ponteiros para o node original dessas peças.

Outra pequena otimização feita foi verificar estados repetidos antes de gerar o node, desta forma a função de pesquisa apenas recebe um gerador com nodes que vão ser usados.

Também usamos programação dinâmica para guardar características de nodes tal como o seu tabuleiro, peças e nodes filhos, de modo a melhorar a velocidade quando decidimos recalcular alguma parte do caminho ou quando passamos por estados repetidos com custo relevante.

Random Counter

Para corrigir o caminho sempre que existe um movimento aleatório esta entidade é invocada.

Começa por detectar a peça e direção desse movimento e de seguida segue um algoritmo que fizemos que consiste em:

- Percorrer a lista de movimentos, verificando se o movimento é possível
- Caso encontre um movimento com a mesma peça:
    - Se a direção é a mesma, quer dizer que o movimento foi benéfico e então retira este da lista
    - Se for a direção contrária, podemos corrigir a adicionar o movimento com direção oposta nesta posição da lista, aumentando o custo em 1 movimento do cursor
- Se o movimento não for possível, podemos adicionar o movimento oposto ao aleatório nesta posição da lista de modo a minimizar o movimento do cursor, visto que este estará na peça que foi bloqueada, ou então recalcular o caminho para obter menor custo
- Se percorreu a lista toda sem encontrar nenhum caso acima, quer dizer que o movimento aleatório não tem impacto na solução

Métodos de avaliação

Com o objetivo de avaliar as nossas soluções, fizemos ficheiros de testes onde podemos simular níveis e ver tempo demorado, número de movimentos de carros, número de movimentos de cursor.

Isto permitiu avaliar métodos de expansão de nós, diferentes pesquisas e heurísticas.

Além disso usamos um dataset de 1000 níveis 6x6 e fizemos um gerador de níveis para poder testar níveis maiores.

Também executamos o student.py múltiplas vezes com diferentes soluções de modo a identificar qual delas obtém um melhor resultado.

Nota: Não foram apresentados benchmark em concreto, de modo a manter o relatório sucinto, visto que foram feitas muitas mudanças ao longo do trabalho