universidade de aveiro
theoria poiesis praxis

Departamento de Eletrónica, Telecomunicações e Informática

Curso 8316 – Licenciatura em Engenharia de Computadores e Informática

Disciplina 40846 – Inteligência Artificial

Ano letivo 2022/2023

Trabalho Prático de Grupo - Rush Hour

Autores:

89078 Luís Filipe Correia do Couto – 50% do trabalho

103248 José Miguel Guardado Silva – 50% do trabalho

Turma P4

Regente Luís Filipe de Seabra Lopes

Arquitetura do Agente

O agente é composto por dois ficheiros desenvolvidos pelos estudantes:

- student.py
- pathfinding.py

student.py **-** ficheiro principal do agente. Aqui é criada a árvore de pesquisa e chamado o método para solucionar o problema caso não existam movimentos a fazer. Caso já existam, envia a cada 100ms a chave correspondente ao movimento a realizar. Também é aqui gerido o problema de sincronização, de modo ao jogo não ficar fora de *synch* com o servidor.

pathfinding.py **-** ficheiro que contém as classes com todos os métodos necessários para realizar a pesquisa.

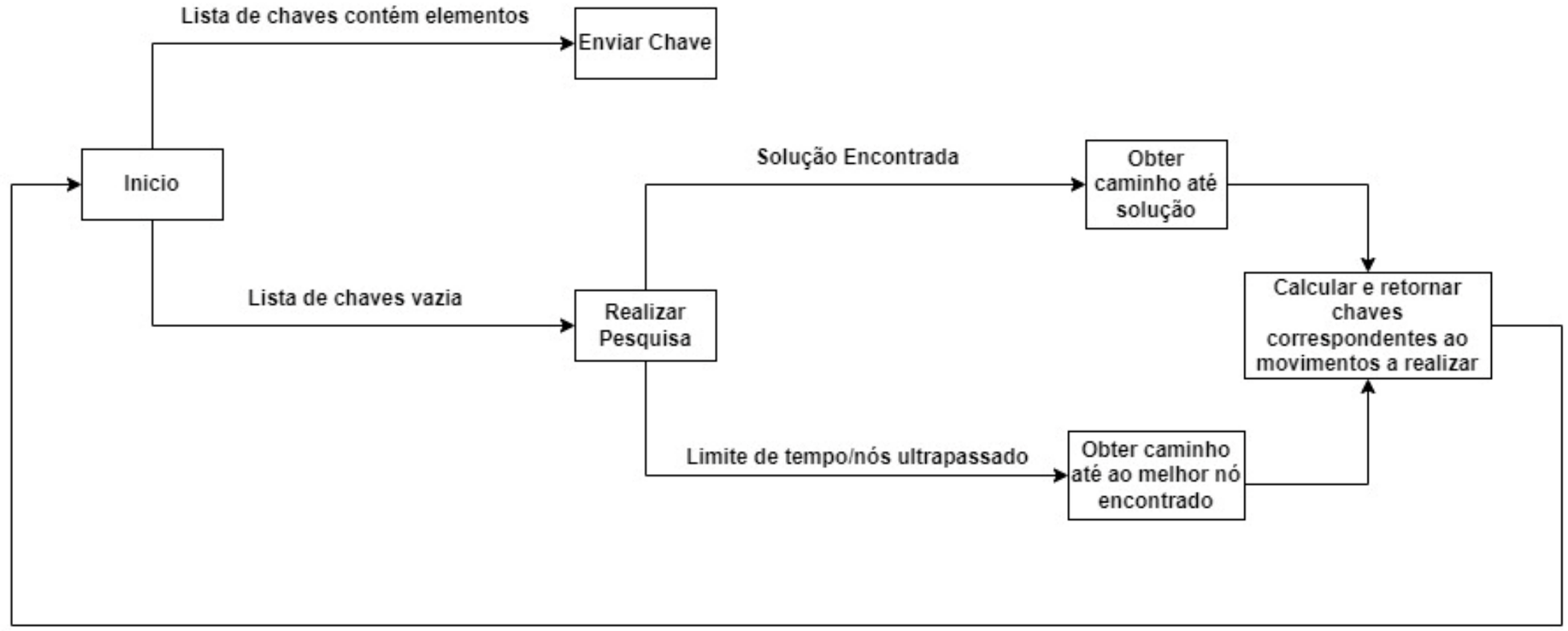
```
class SearchProblem - Classe que contém métodos para resolver operações específicas ao problema de pesquisa (verificar se um estado é solução, calcular heurística de um estado, calcular movimentos possíveis num estado, realizar movimento num estado, etc..)

class Node - Estrutura de dados para representar um nó da árvore de pesquisa.

class SearchTree - Estrutura de dados para representar a árvore de pesquisa. Contém também os métodos necessários para obter a solução do agente (Pesquisa A*, obter caminho até um dado nó, obter chaves correspondes aos movimentos a realizar, etc...)
```

Análise do Algoritmo

De uma maneira simplificada, o algoritmo utilizado pode ser descrito da seguinte maneira:



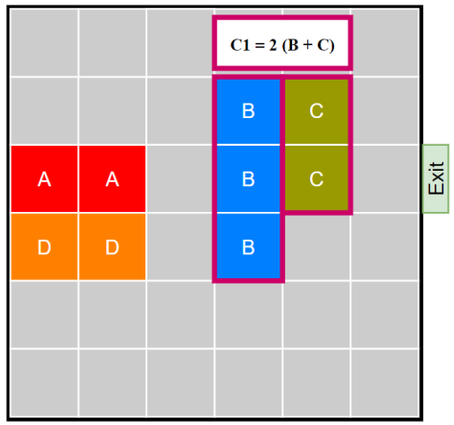
Heurística

A fórmula para calcular a heurística de um novo estado é composta pela soma de duas diferentes características do mapa desse novo estado:

$$H(n) = C_1(n) + C_2(n)$$

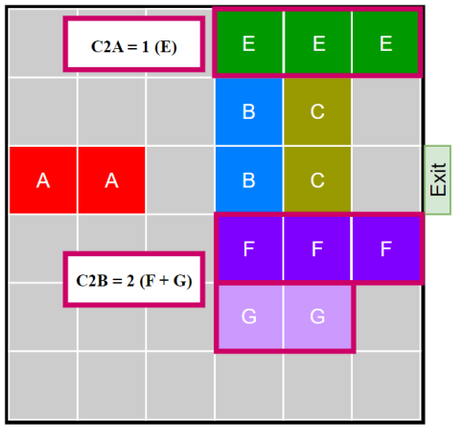
onde

$C_1(n)$ - número de carros a bloquear o carro alvo
$C_2(n)$ - número mínimo de carros a bloquear os carros que bloqueiam o carro alvo



$C_1(n) = 2$

O agente deve procurar, sempre que possível, minimizar este valor



$C_2(n) = min(C_2A, C_2B) = 1$

O agente deve procurar, sempre que possível, minimizar este valor

Pathfinding

O algoritmo de pesquisa utilizado neste projeto foi o algoritmo A*, utilizando a heurística descrita no slide anterior. Foram também introduzidas as seguintes alterações, de modo a melhorar a implementação e desempenho geral:

* Lista de nós abertos guardada numa min-heap, através da estrutura de dados **heapq**, usando para ordenação de inserção uma função de avaliação dinâmica dada por:

$$eval(n) = custo(n) + heurística(n) \text{ if } custo(n) < heurística(n) \text{ else } (custo(n) + (2*1.5 - 1) * heurística(n)) / 1.5$$

* Prevenção de ciclos através da implementação de um dicionário (*key*=str*(*estado), *value*=nó) de nós já visitados.

* Introdução de um tempo de pesquisa máximo e número máximo de nós já abertos, para prevenir tempos de pesquisa demasiado longos. Se um destes valores for ultrapassado e ainda não tenha sido encontrada a solução, a pesquisa termina e retorna o caminho até ao melhor nó encontrado até ao momento.
Caso isto aconteça, a árvore de pesquisa mantém todos os valores do dicionário de nós já visitados, de modo a melhorar o desempenho das próximas pesquisas.
Quando a solução é encontrada, são apagados todos os valores do dicionário de nós já visitados.

* Todas as operações são realizadas tendo em conta o tamanho do mapa, de modo a suportar mapas de qualquer tamanho

Fontes Consultadas

No desenvolvimento do projeto foram consultadas as seguintes fontes:

- Slides das aulas teóricas
- https://docs.python.org/3/library/heapq.html
- https://docs.python.org/3/library/time.html
- http://abbashommadi.github.io/AI-for-Rush-Hour-Game/
- https://www.cs.huji.ac.il/w~ai/projects/2015/RushHour/files/report.pdf