Inteligência Artificial (LECI)

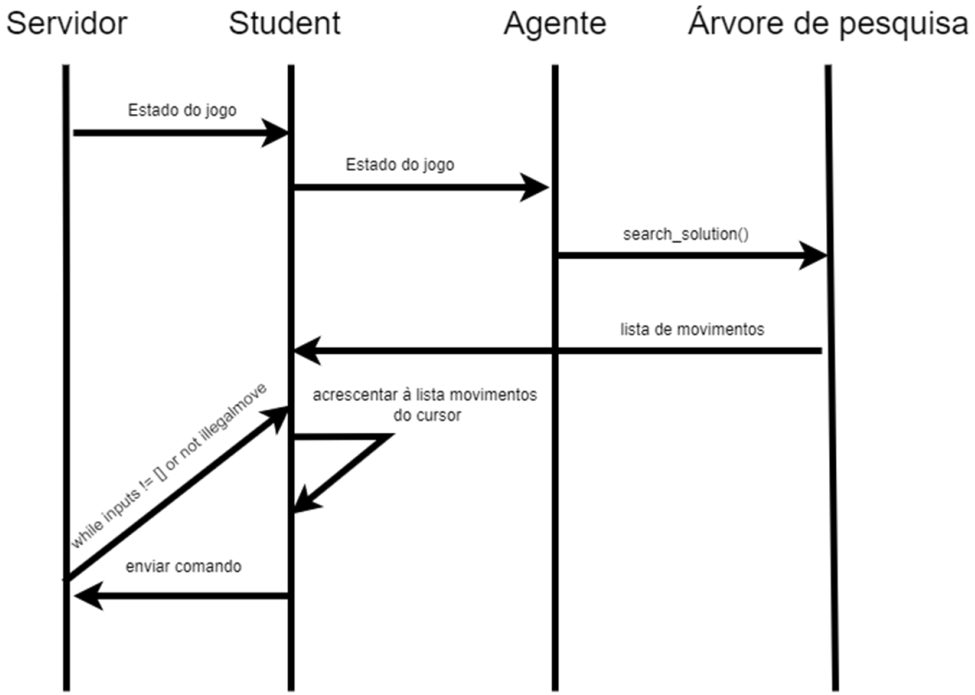
RUSH HOUR

- *Rafael Pinto 103379*
- *Adalberto Rosário 105589*

universidade de aveiro

Agente

- O nosso agente (agent.py) interage com o server através do student.py.
- Após receber as informações do nível, o student chama o agente enviando o estado atual do nível. O agente depois começa a pesquisa em árvore, iniciando o módulo implementado em tree_search.py.
- Quando é descoberta uma solução na árvore de pesquisa é enviado uma lista de tuplos do tipo "(carro a mover, movimento)" com os movimentos necessários para passar o nível para o student.
- Depois, no student, a lista com os movimentos é percorrida e a cada passo são calculadas as ações necessárias para chegar com o cursor até ao carro que se pretende mover.
- Quando acontece um crazy car e isso implicar que a solução anterior não consiga ser concretizada, os movimentos restantes são descartados e é recomeçada a pesquisa.



Algoritmo utilizado e Replaneamento

Algoritmo utilizado

- Para a procura da solução do problema usámos um modelo de árvore de pesquisa idêntica à desenvolvida nas aulas práticas e utilizou-se a pesquisa gulosa como estratégia de pesquisa.
- Escolhemos usar a estratégia de pesquisa gulosa, uma vez que mostrou ter um melhor desempenho que as restantes estratégias de pesquisa que implementou-se ao longo do projeto.

Replaneamento

- O agente executa todos os comandos que calculou no estado inicial e só vai calcular uma nova solução se um dos movimentos anteriores for bloqueado por um crazy car.
- Temos uma condição no student.py que deteta se o movimento executado foi legal ou não. Se o movimento não for legal o agente recomeça a pesquisa por uma solução nova.

Heurística

- Para a heurística, efetuam-se duas contagens:
  - Carros a bloquear o caminho do carro A
  - Número mínimo dos carros que estejam a bloquear estes últimos, caso estejam na vertical (mínimo entre os carros a bloquear movimentos para cima e carros a bloquear movimentos para baixo).
- Para a segunda contagem só temos em consideração os carros verticais, pois não faria sentido contar carros que já teriam sido contados quando se efetuou a contagem dos que bloqueavam o caminho ao carro A.
- Na figura ao lado temos 2 carros a bloquear (círculo azul) o carro A (vermelho) (C1), temos 1 carro a bloquear (círculo laranja) o primeiro carro que bloqueia A (C2a) e 2 carros a bloquear (círculos castanhos) o segundo carro que bloqueia A (C2b).
- A heurística da figura vai ser dada por:
  - H = C1 + min(C2a, C2b) = C1 + C2a = 2+1 = 3
- Permite assim dar prioridade aos nós que possuem menos carros a bloquear o caminho ao carro A e os seus carros carros bloqueadores.

Avaliação do agente

- Após executar o jogo várias vezes e cronometrar o tempo que o agente demorava a encontrar uma solução, conseguimos recolher dados suficientes para construir o gráfico ao lado.
- Para a construção do gráfico a versão mais recente do ficheiro com os níveis (levels.txt), até à data da entrega final
- Com este gráfico podemos concluir que até ao nível 25 o nosso agente não enfrenta dificuldades a rapidamente chegar a uma solução.
- O nosso agente apresenta grande dificuldade nos níveis 25 e 39.
- Como o nosso agente precisa de recorrer ao módulo tree_search.py cada vez que um crazy car bloqueia o movimento de um carro, nos níveis mais complexos fica difícil de completar os níveis devido à demora da pesquisa.
- Grande parte das vezes o nosso agente consegue concluir todos os níveis, porém, nos níveis mais complexos fica dependente da dificuldade que os crazy cars impõem no agente.