

Manual Instruções do mundo agent0

Para correr o nosso programa, primeiro é necessário correr o ficheiro “main_server_zmq.py”, este programa serve para ligar o mundo dos agentes(servidor) e para adicionar agentes ao mundo é necessario correr “main_client_zmq.py”.

O nosso mundo aceita múltiplos agentes, sempre que um agente se move é mandada uma mensagem broadcast para todos os agentes ativos no mundo a dizer que um agente moveu-se para uma casa nova(no manual movement).

Temos um simples menu com três opções no lado do cliente(“main_client_zmq.py”), que servem para o utilizador escolher qual dos algoritmos pretende utilizar.

Pesquisa em profundidade:

A primeira opção do menu é o algoritmo de pesquisa em profundidade.

“Um algoritmo de busca em profundidade realiza uma busca não informada que progride através da expansão do primeiro nó filho da árvore de busca, e se aprofunda cada vez mais, até que o alvo da busca seja encontrado ou até que ele se depare com um nó que não possui filhos (nó folha). Então a busca retrocede (backtrack) e começa no próximo nó.”

-https://pt.wikipedia.org/wiki/Busca_em_profundidade

No nosso programa só temos que inserir o nível de profundidade que o utilizador deseja que o algoritmo percorra e este percorre todas as casas que estão ao alcance deste nível e verifica se existe o “GOAL” dentro dessa pesquisa.

Caso o goal seja encontrado, o algoritmo irá guardar numa lista todas posições em que se movimentou desde a sua posição inicial até ao goal, e de seguida manda essa lista para a função “follow_road()” que simplesmente faz o agente caminhar novamente da posição inicial até á posição do goal seguindo o caminho encontrado pelo algoritmo de pesquisa em profundidade da função “depth_search”.

Trepa colinas:

O segundo algoritmo é o trepa colinas.

O trepa colinas é um algoritmo pouco inteligente que sabe a posição do “GOAL” desde o início, e tenta chegar lá pelo caminho mais curto, ou seja ele vai tentar igualar as suas coordenadas com as coordenadas do goal, e contém um elemento random em que ele em cada jogada decide aleatoriamente se vai tentar igualar a coordenada X ou a coordenada Y.

Este algoritmo não tem em atenção se existem obstáculos à frente ou não. Se houverem obstáculos à frente dele, o agente tenta se desviar para as casas ao lado numa tentativa de se aproximar do objetivo. Caso ele entre num beco sem saída, ele não consegue sair.

Manual movement:

Nesta opção temos o utilizador tem controlo sobre o agente.

Ele tem que inserir algum comando para o agente se mover ou fazer alguma ação. No início ele pede um raio de visão que vai ser usado para decidir o quão á frente o agente consegue ver, ou seja, se o raio de visão for “1” o agente só consegue ver a posição de uma casa á frente dele, se o raio for “3” ele irá conseguir ver três casas á frente e por ai em diante.

Os comandos são compostos sempre por duas partes as “Headers” e os “Values”.

O nosso programa contém 3 headers, sendo estes o “info” (que serve para ir buscar alguma informação do servidor), “command”(que serve para interagir com os agentes no servidor) e “raio”(Que serve para definir o raio de visão).

Cada um destes “Headers” contêm os seus “values” específicos, abaixo temos os comandos que são aceites:

“command north” : Move-se para a casa a cima

“command south” : Move-se para a casa abaixo

“command west” : Move-se para a esquerda

“command east”: Move-se para a direita

“command set_steps”: Ao fazer este comando vai começar a pintar as casas onde esteve.

“info north”: Mostra informação a norte do agente(dependendo do valor do raio)
“info south”: Mostra informação a sul do agente(dependendo do valor do raio)
“info west”: Mostra a informação a este do agente(dependendo do valor do raio)
“info east”: Mostra a informação a oeste do agente(dependendo do valor do raio)
“info map”: Mostra várias listas que corresponde a cada linha, dentro dessa lista tem 1 e 0, o 0 corresponde não há obstáculos e o 1 diz que naquela linha e posição x tem um obstáculo.
“info position”: Mostra a posição atual do agente.
“info goal”: Mostra a posição goal

“raio x”: X pode ser um valor qualquer , que irá determinar o raio de visão do agente.

Nesta opção (manual_movement) utilizamos threads para interagir com o servidor e para estar á espera de mensagens do servidor ou seja utilizamos o

“Publisher/Subscriber” em que sempre que um agente se mover na board, os outros agentes todos receberão uma mensagem a dizer que o “agenteX” se movimentou para as coordenadas X, e também caso algum dos agentes ligados ao servidor chegue ao “GOAL”, será enviada também uma mensagem para os restantes agentes a dizer que o “agenteX” chegou ao goal.

Atenção: Após ser inserido um comando o servidor demora alguns segundos a responder, por exemplo, se o utilizador inserir “command west” irá demorar por volta de 3 segundos para que o agente se mova e os restantes agentes recebam a mensagem que o “agenteX” se mexeu. (As respostas do servidor estavam bem mais rápidas em commits anteriores, mas como agora nos foi pedido para nos focarmos mais no “publisher/subscriber” do “manual_movement”, o servidor demora alguns segundos a fazer broadcast das mensagens que quer enviar.)

Obstáculos:

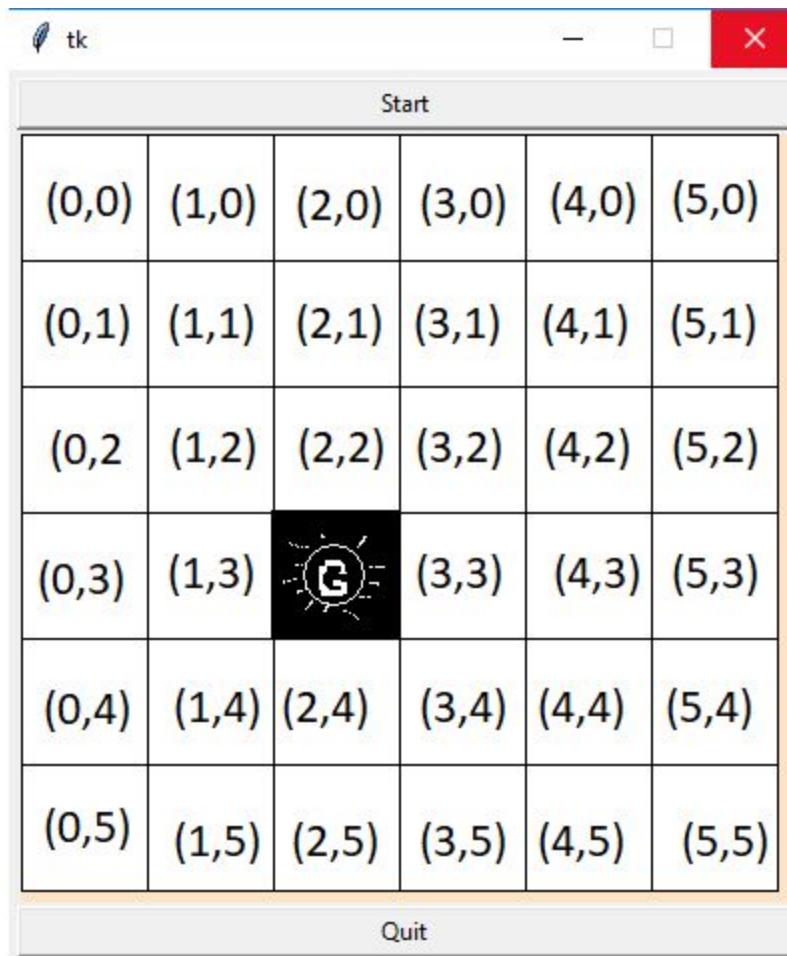
O nosso programa começa com alguns obstáculos em certas posições, posições estas que podem ser mudadas no ficheiro “main_server_zmq.py” como é demonstrado na imagem abaixo.


```
if name == "main":  
    print("Starting the Game Board")  
  
    root = tk.Tk()  
    board = gb.GameBoard(root, CONST_BOARD_ROWS, CONST_BOARD_COLUMNS)  
    board.pack(side="top", fill="both", expand="true", padx=4, pady=4)  
    # BOARD BOARD:  
    initialize_obstacles(CONST_IMAGE_DIR, [(1,1), (2,1), (3,1)])  
    initialize_goal(CONST_IMAGE_DIR, (CONST_GOAL_COORD_X, CONST_GOAL_COORD_Y))  
    initialize_bomb(CONST_IMAGE_DIR, [], CONST_BOARD_ROWS, CONST_BOARD_COLUMNS)  
    root.update()  
    # Loop  
    loop()
```

Seguindo o código da imagem acima, serão criados 3 obstáculos nas posições (1,1), (2,1) e (3,1).

É possível adicionar mais obstáculos ao mundo ou nenhum.

Coordenadas da Board



(0,0)	(1,0)	(2,0)	(3,0)	(4,0)	(5,0)
(0,1)	(1,1)	(2,1)	(3,1)	(4,1)	(5,1)
(0,2)	(1,2)	(2,2)	(3,2)	(4,2)	(5,2)
(0,3)	(1,3)		(3,3)	(4,3)	(5,3)
(0,4)	(1,4)	(2,4)	(3,4)	(4,4)	(5,4)
(0,5)	(1,5)	(2,5)	(3,5)	(4,5)	(5,5)

Na imagem acima está demonstrado o tabuleiro e como funcionam as coordenadas na board.

Trabalho realizado por:
-Hélder Braga
-João Sousa
-Leandro Branco

