Relatório Agente Inteligente

por

Gabriel Teixeira Júlio
Marcus Vinicius Nogueira Santos

Supervisor:
ALISSON MARQUES DA SILVA

CEFET-MG Campus-V - Divinópolis

1 de abril de 2024

Resumo

A atividade se basea em um agente inteligente onde dado um ambiente ele deverá limpa-ço da maneira mais eficiente possivel, possuindo restrições,penalidades e recompensas de acordo da ações feitas, ele deverá ser um algoritmo pseudo-aleatório e seu armazenamento deve ser apenas seus 3 ultimos movimentos, não podendo guardar mais informações ou posições.Para nosso algoritmo decidimos implementar que o agente irá se mover em "zigue-zague", potem ao encontrar uma parede ele deve escoler entre no maximo dois movimentos pois, em um vai está bloqueado por uma parede e outro será o local de onde ele veio não sendo um escolha inteligente.

Introdução

Foi implementado um agente inteligente, onde no caso desta atividade foi uma robô aspirador de pó que deveria analisar uma area de 4x4, que tem as seguintes funções: movimentar(cima, baixo, direita, esquerda), verificar o local se está limpo, caso não esteja ele deverá limpar o local, cada ação de movimento do robô tem uma penalidade de 1 ponto, ja quado ele limpa o quadrado tem um acrescimo de 3 portos, no final após a limpeza para cada quadrado sujo que restou ele tem uma penalidade de 20 pontos, seu objetivo é de limpar o ambiente e obter a maior pontuação possivel, o agente tem seu local de inicio aleatório igualmente com os locais que estarão sujos, o movimento do agente ao bater um uma das parades tem de ser pseudo-aleatório ou seja como ele não ira conseguir ir para frente, e voltar nçao seria a escolha mais inteligente, ele deve escolher entre as duas laterias quando possivel. Como armazenamento o agente só poderá armazenar os 3(três) ultimos movimentos (cima, baixo, direita, esquerda), não podendo guardar sua localização.

Lógica de Funcionamento

Sempre antes de mover o aspirador vai primeiro ver se precisa limpar a posição em que ele está. Na posição inicial do aspirador ele não terá uma direção definida em que ele tem que mover, logo ela é sorteada das direções em que o aspirador pode mover sendo elas U-cima, R-direita, D-baixo e L-esquerda. Depois de sortear a direção ele irá andar nesta direção até bater numa parede.

Quando o aspirador bater em uma parede, ele primeiro irá inverter a direção de movimento que ele estava fazendo, ou seja, se ele andava para direção D ele agora vai andar para direção inversa U. Após isso, ele irá sortear uma direção para andar para substituir o movimento em que foi bloqueado para definir as direções que serão sorteadas ele remove as que levaria ele voltar para a posição antes da batida e a que levou ele a bater. Depois deste movimento ele volta a seguir sua direção de movimento.

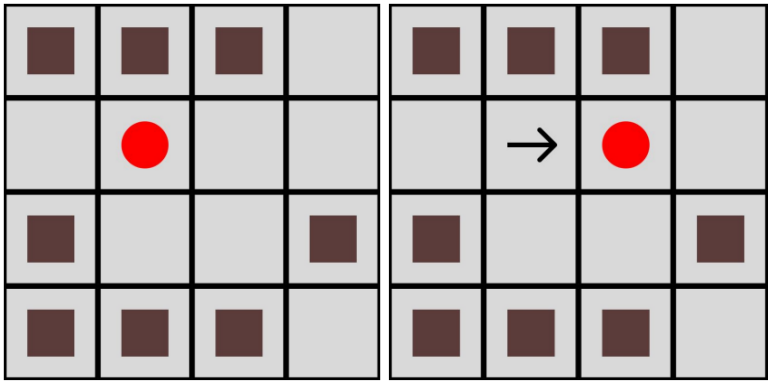

Figura 1: A esquerda Estado Inicial. A direita Depois de definir a direção de movimento o agente se movimenta

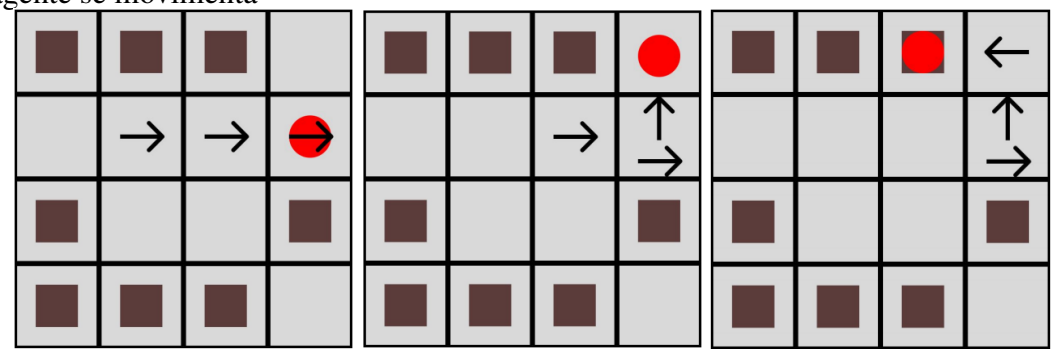

Figura 2: A esquerda Agente bateu na parede. No meio sorteou ir para cima dentre as opções entre cima ou baixo. A direita aspirador volta a andar utilizando sua direção de movimento

Quando o aspirador bater em duas paredes, em seguida ele chegou num canto, logo irá tentar se movimentar e mudar a direção de movimento para a direção sorteada das que sobram depois de remover a direção do quadrado que ele veio e as duas em que ele bateu. Caso a direção que ele tenha decidido resulte em uma batida novamente ele volta por onde ele veio ou testa a última direção possível, acontece só quando for seu primeiro movimento.

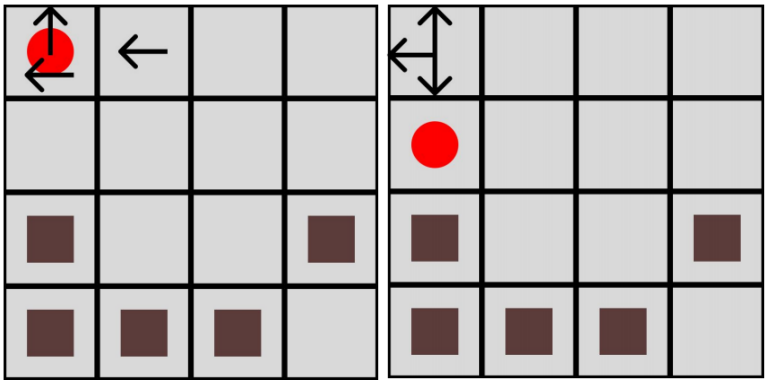

Figura 3: A esquerda Agente bateu em um cantol. A direita aspirador foi para única direção possível e continuará nela

O aspirador tem como condição de parada o esgotamento de sua bateria. A cada movimento realizado pelo agente, a bateria é reduzida em uma unidade.

Metodologia

O código foi feito no Python 3.12.0, a primeira coisa feita foi criar duas classes chamadas Ambiente e Aspirador, que tratam dos dados de seus respectivos nomes.

Ambiente

Os atributos utilizados em Ambiente foram:

- *Mapa* - contém as informações sobre o mapa em que o aspirador funiona, onde está limpo, sujo e onde são as paredes;
- *Posição* - guarda a posição do mapa em que o aspirador localiza-se;
- *Pontos* - contém a pontuação do aspirador, começa como 0.

Os métodos para o Ambiente são:

- *limpar_posicao* - utilizado para atualizar a posição atual do aspirador como limpa;
- *atualizarpontuacao* - usado para atualizar a pontuação do aspirador dependendo da ação que tenha sido passada para o método, aumenta os pontos em 3 se limpou e diminui em 1 se movimentou;
- *atualizar_posicao* - que serve para atualizar a posição do aspirador no mapa sendo a posição passada para o método.

Aspirador

Os atributos utilizados em Aspirador foram:

- *Memória* - armazaena os três últimos movimento em que eles são armazenado numa fila, ou seja, o o último movimento da fila é o movimento anterior do aspirador, para cada movimento é guardado a direção e o estado do movimento;
- *Direções* - contém todas as direções possíveis que o aspirador pode se mover;
- *Direção* - guarda a direção que o aspirador segue;
- *Bateria* - armazena a quantidade de bateria o aspirar possui.

Os métodos para o Aspirador são:

- *sensor_aspirador* - utilizado para retorna se a posição atual do aspirador está suja ou limpa;
- *acao_aspirador* - usado para limpar a posição atual do aspirador e depois atualizar a pontuação, o método recebe o retorno de *sensor_aspirador* para dcidir se o aspirador limpa ou não;
- *atualizar_dados* - serve para atualizar os dados depois de um movimento recebendo a direção e o status do movimento feito, ele atualiza a memória guardando o movimento feito, a pontuação e a bateria é diminuida em 1;
- *get_reverse* - serve para retornar a direção inversa da que foi passada ao método;
- *mover_aspirador* - utilizado para mover o aspirador nadireção passada pra ele, se o aspirador puder mover na direção é retorna 'L' de livre senão puder é retornado 'B' de bloqueado;
- *movimentar_aspirador* - usado para decidir para qual direção o aspirador vai se mover.

O método *movimentar_aspirador* obedece a seguinte lógica:

1) Guarda-se uma cópia da memória(*mem*) antes de mover o aspirador e uma cópia das direções possíveis(*dir*) do aspirador;
2) Pega-se o movimento anterior do aspirador, ou seja, o último movimento de *mem*;
3) Checa se existe um movimento anterior, ou seja, sé o o primeiro movimento do aspirador ou não;
4) Se existir anterior:
    a) Guarda a direção inversa do movimento anterior(*d_ant*)
    b) E tira *d_ant* das direções possíveis(*dir*), ou seja, remove a direção que leva para posição anterior
5) Se não existir anterior:
    a) Sorteia uma direção para mover de *dir* e guarda ela na direção de movimento do aspirador
6) Move o aspirador na direção de movimento do aspirador usando *mover_aspirador*e guarda o retorno do método em *status*
7) Atualiza os dados depois de fazer o movimento usando *atualizar_dados* passando a direção de movimento do aspirador e *status*;

8) Checa se *status* é ’L’ ou ’B’;

9) Se for ’L’, o aspirador foi para um espaço livre logo o método acaba;

10) Se for ’B’, o aspirador bateu em uma parede:

    a) Retira-se a direção de movimento do aspirador de *dir*;

    b) A direção de movimento do aspirador recebe o seu inverso;

    c) Sorteia outra direção de dir e guarda em *t_d*;

    d) Move o aspirador na direção *t_d* usando *mover_aspirador*e guarda o retorno do método em *status*

    e) Atualiza os dados depois de fazer o movimento usando *atualizar_dados* passando *t_d* e *status*;

    f) Verifica-se novamente se *status* é ’L’ ou ’B’;

    g) Se for ’L’, o aspirador foi para um espaço livre logo o método acaba;

    h) Se for ’B’, o aspirador bateu em outra parede logo está num canto:

        i) Retira-se *t_d* de *dir*;

        ii) Sorteia de novo um direção de *dir* e coloca em *t_d*

        iii) A direção de movimento do aspirador recebe *t_d*

        iv) Move o aspirador na direção *t_d* usando *mover_aspirador*e guarda o retorno do método em *status*

        v) Atualiza os dados depois de fazer o movimento usando *atualizar_dados* passando *t_d* e *status*;

        vi) Checa novamente se *status* é ‘L’ ou ‘B’;

        vii) Se for ’L’, o aspirador foi para um espaço livre logo o método acaba;

        viii) Se for ’B’, o aspirador bateu em mais uma parede logo está num beco:

            (1) Checa se *dir* está vazio:

                (a) Se está:

                    (i) Retira-se *t_d* de *dir*;

                    (ii) Sorteia de novo um direção de *dir* e coloca em *t_d*;

                    (iii) Move o aspirador na direção *t_d* usando *mover_aspirador*e guarda o retorno do método em *status*

                    (iv) Atualiza os dados depois de fazer o movimento usando *atualizar_dados* passando *t_d* e *status*;

                    (v) Checa se *status* é ‘B’;

                    (vi) Se for ’B’, o aspirador tentou ir em todas as direções e foi bloqueado nelas, logo atualiza bateria para 0.

(b) Senão está:

(i) Move o aspirador na direção *d_ant* utilizando *mover_aspirador* e guarda o retorno do método em *status*;

(ii) Atualiza os dados depois do movimento utilizando *atualizar_dados* passando *d_ant* e *status*

(iii) Checa se *status* 'B'

(iv) Se for 'B', o aspirador tentou ir em todas as direções e foi bloqueado nelas, logo atualiza bateria para 0.

Com as duas classes principais o aspirador segue a lógica a seguir enquanto tiver bateria disponível, sendo ela:

1. Chama-se *sensor_aspirador* e guarda o retorno em *status*
2. Chama-se *acao_aspirador* passando *status*
3. Chama-se *movimentar_aspirador*

Para avaliar sua eficácia, o aspirador foi testado em diversas condições de sujeira, variando entre 16, 12, 8 e 4 posições sujas. Cada configuração foi repetida 10 vezes, registrando a melhor e a pior pontuação, bem como a média e o desvio padrão das 10 repetições para cada quantidade de sujeira.

Os testes foram conduzidos em uma matriz de dimensão 4x4, onde uma penalização de -20 na pontuação foi atribuída para cada quadrado sujo após a conclusão da operação do aspirador. E a capacidade da bateria foi fixada em 40.

Resultados Obtidos

Estes foram os dados obtidos através de alguns testes realizadaos do algoritmo.

	16 Quadrados Sujos	12 Quadrados Sujos	8 Quadrados Sujos	4 Quadrados Sujos
Melhor	8	-4	-16	-28
Pior:	-84	-73	-39	-74
Média:	-31,2	-27,0	-20,7	-35,3
Desvio Padrão	38,51	30,86	9,15	14,70