Assignment 1 - RMI

Diogo Mendes (88801)
Raquel Pinto (92948)

Challenge 1

```
public void wander(boolean followBeacon) {
    if(irSensor0>1) {
        if(Math.max(irSensor1,irSensor2)==irSensor1) cif.DriveMotors(0.15,-0.05);
        else cif.DriveMotors(-0.05,0.15);
    }
    else if(irSensor1>2.1) cif.DriveMotors(0.15,0.1); //esquerda
    else if(irSensor2>2.1) cif.DriveMotors(0.1,0.15); //direita

    else cif.DriveMotors(0.15,0.15);
}
```

Challenge 2

Visualizamos o mapa como se estivesse dividido em quadrados.
O objetivo é o robô estar no centro de um quadrado para mapear as 4 direções
Desenvolveu-se uma classe vetor
Criamos um vetor “next” e trabalhamos com ele como o objetivo.

- Máquina de estados: GA, RL, RR, INV, END
  - Se TargetReached() então chama mappingDecode (escolher novo objetivo)
  - Se “caminho” não estiver vazio então chama a runCaminho()
  - Caso nenhuma das situações de cima se confirmem então faz as funções de movimento
  - Estado END - writeMap(), termina simulador e exit do java.
- Função Estados que define o estado consoante os valores do compass e do compass_goal.

Challenge 2 - Funções Movimento

- Abordagem com utilização de PID, mais especificamente a parte proporcional.
- Consoante o eixo do movimento o erro é calculado com um delta diferente.
- O “nivel” corresponde ao sentido (positivo ou negativo).
- Para rotação usou-se o compass_goal e o compass atual para calcular o erro.

```
public void goAhead(){ //yr -> y inicial -> funcao de andar para a frente
    double deltaY = next.getY() - y; //Erro do Y
    double deltaX = next.getX() - x; //Erro do X
    double l, r;
    double k = 0.75;

    if(Eixo()){ // ele esta virado na horizontal
        l =0.1 - k * deltaY * nivel(); //nivel corresponde a ser + ou - no eixo
        r =0.1 - k * -deltaY * nivel(); // Valor maximo de velocidade menos
    }
    else{ //robo no eixo veritcal
        l =0.1 - k * -deltaX * nivel(); //nivel corresponde a ser + ou - no eixo
        r =0.1 - k * deltaX * nivel(); //
    }
    cif.DriveMotors(l, r);
}
```

Challenge 2 - mappingDecode()

- Adicionar as coordenadas atuais a uma LinkedList que guarda as coordenadas visitadas.
- Remover as atuais da LinkedList com coordenadas visitáveis.
- Mapear à sua volta com os sensores e guardar num String[][] que tem o tamanho do mapa.
- Adicionar o que não for parede e não visitado às visitáveis.
- Calcular os vizinhos que são visitáveis.
- Escolher um deles como next e calcular o compass_goal consoante esta escolha.
- Caso não existam mais visitáveis (isEmpty()) colocar um boolean “end” como true.
- Caso não haja vizinhos locais chama a função setCaminho()

Challenge 2 - setCaminho() e runCaminho()

setCaminho()

- Escolhe as próximas coordenadas do next como objetivo (visitaveis.next()).
- Usando a posição atual como partida
- Utiliza o A* para calcular o caminho do objetivo para a partida
- Inverte o caminho (partida -> objetivo)
- Chama a runCaminho()

runCaminho()

- define o primeiro vetor da LinkedList “caminho” como o next.
- Ajusta o compass_goal para este next
- Elimina o primeiro vetor do “caminho”

Challenge 2 - A*

Código fonte retirado de: https://stackabuse.com/graphs-in-java-a-star-algorithm/
Principais adaptações:

- Adição de um valor vetor aos nós
- Reformulação de algumas partes para aceitarem os novos nós
- Adição de campo “filhos” à classe vetor para guardar os vizinhos com movimentos possíveis.

Challenge 2 - Outras funções

- WriteMap() - Escreve o String [][] num ficheiro txt
- AddToMap() - Adiciona ao String[][]
- OnMap() - verificar se coordenadas já foram preenchidas com algo diferente de “ “
- FillMap() - encher mapa com “ “
- Funções para obter coordenadas nas 4 direções, para k+1 e k+2
- Funções para arredondar a bússola e a bússola objetivo
- Verificar se é parede nas 4 direções

Challenge 3

- Código igual ao Challenge 2
- Em vez de ter a função WriteMap temos a WritePath que escreve o path obtido num txt
- Adição de um array vetor[getNumBeacons()] para guardar as coordenadas dos targets
- Adição de uma função para calcular o caminho para os targets - usa o A* para calcular o caminho do (0->1->2->...->0), soma tudo na mesma LinkedList para ser escrito.