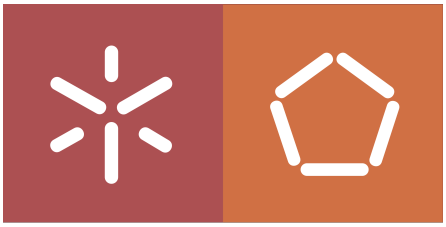

Universidade do Minho
Escola de Engenharia

RoboCode

Sistema Autónomos
MEI - 4º Ano - 2º Semestre
Grupo 4

PG42818 Carolina Marques
PG42820 Constança Elias
PG42844 Maria Araújo Barbosa
A86271 Renata Ribeiro

9 de abril de 2021

Conteúdo

Lista de Figuras

Capítulo 1

Introdução

1.1 Contextualização

Este trabalho foi desenvolvido no âmbito da unidade curricular de Sistemas Autónomos. O objectivo principal do projeto é desenvolver e programar sistemas autónomos, com particular ênfase no que respeita à construção de processos de simulação de comportamentos individuais, de grupo e sociais, no ambiente de programação RoboCode. Sucintamente, o RoboCode é um jogo de competição que permite aos jogadores programar tanques de batalha para destruir equipas adversárias.

1.2 Estrutura do Relatório

Este relatório divide-se em cinco partes principais:

- A primeira parte consiste na análise do problema e dos principais requisitos.
- Na segunda parte, é explicada a implementação da Circum-navegação e do Odómetro.
- A terceira corresponde à explicação dos comportamentos sociais de grupo.
- A quarta apresenta o trabalho realizado para preparar os robôs para a competição de equipa.
- Por fim é feita uma breve síntese do trabalho realizado, apresentando as principais conclusões do mesmo.

Capítulo 2

Análise e especificação do problema

2.1 Descrição informal do problema

Este projecto encontra-se dividido em três etapas. A primeira consiste na concepção e implementação de um odómetro e de técnicas de planeamento de trajectórias para realizar a circum-navegação de três objetos. Na segunda etapa, pretende-se criar equipas que apresentem diversos comportamentos sociais de grupo. Por último, deve ser preparada uma equipa de cinco elementos para realizar uma competição.

2.2 Especificação dos requisitos

Os requisitos principais passam por:

- Criar robôs;
- Desenvolver sistemas de controlo implementado diferentes estratégias;
- Criar diferentes arquiteturas de controlo;
- Garantir cooperação entre os robôs.
- Planear trajectórias;

Capítulo 3

Concepção da resolução

3.1 Fase 1

A primeira fase deste projeto, como já foi referido, envolve a conceção e implementação de um odómetro, para medir a distância percorrida por um robô em cada episódio (*round*) de uma batalha (*battle*), e a implementação de técnicas de planeamento de trajetórias para circum-navegação de 3 obstáculos.

Pretende-se que o robô comece a circum-navegação a partir da posição (18, 18), circule no sentido dos ponteiros do relógio por fora da área demarcada pelos 3 obstáculos e volte à posição de partida. O odómetro implementado será usado de forma a obter o valor da distância percorrida pelo robô, sendo que o objetivo passa por contornar os obstáculos percorrendo a menor distância possível.

3.1.1 Odómetro

O odómetro contém duas variáveis que determinam a condição de corrida. Estas variáveis irão determinar quando é que o odómetro começará a contar a distância percorrida.

- **is_racing**: *Boolean* que indica que o robô já começou o circuito quando está a *true*.
- **finished**: *Boolean* que indica o fim da corrida. Quando este se encontra a *true*, é retornada a distância total percorrida.

O cálculo da distância é feito através da Distância Euclidiana. Sabendo a posição anterior e atual do robô é possível ir atualizando este valor. A figura 3.1 apresenta a fórmula utilizada.

$$d_{AB} = \sqrt[2]{(x_B - x_A)^2 + (y_B - y_A)^2}$$

Figura 3.1: Fórmula euclidiana da distância entre dois pontos.

3.1.2 Circum-navegação

Nesta etapa, o primeiro passo consistiu em fazer com que o robô se deslocasse pelo mapa e ultrapassasse 3 obstáculos. Para isso, começámos por desenvolver a função *goTo(double x, double y)* que permite indicar qual o ponto para onde o robô deve seguir. Para isso, transformámos as coordenadas do nosso robô em vetores. De seguida calculámos o ângulo no qual o robô tinha de se virar para poder avançar em direção à posição final. A figura 3.2 esquematiza os ângulos calculados. Consoante o ângulo de viragem, quando a próxima execução tiver lugar, o robô irá avançar para a frente ou para trás, segundo uma distância que foi calculada através de *Math.hypot*, em *pixels*.

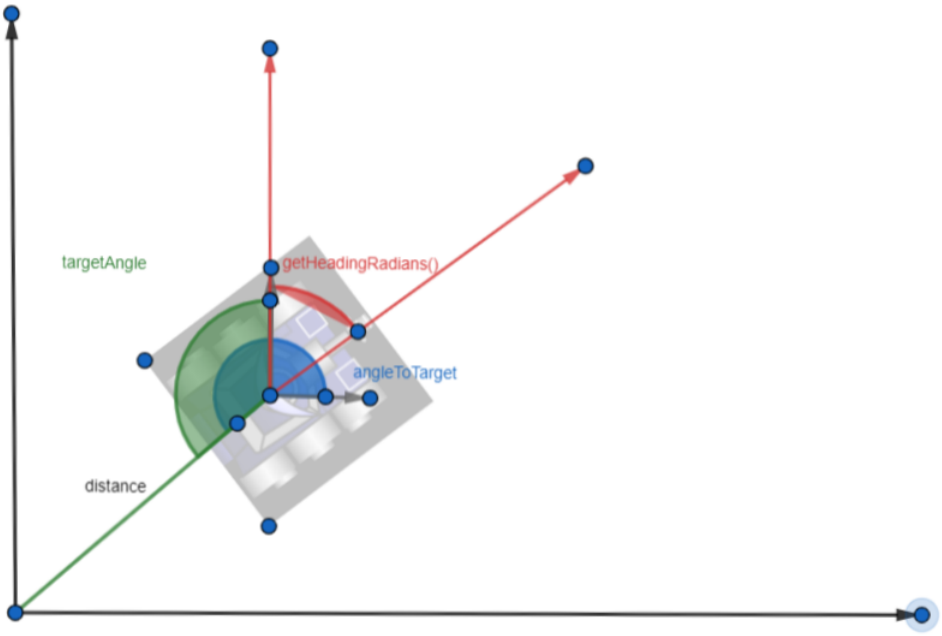

Figura 3.2: Cálculo do Ângulo

A solução proposta para esta etapa consiste em movimentar o robô numa espécie de quadrilátero (semelhante ao que está a ser usado no cálculo do perímetro pelo *Standard Odometer*) que abrange os três obstáculos no mapa. Para tal, é imperativo que o robô contorne os objetos mas que o faça percorrendo a menor distância possível.

Um *ScannedRobotEvent* é enviado quando o robô deteta outro robô e a função **onScannedRobot** contém as instruções que este deve efetuar no caso do evento acontecer. Para iniciar a trajetória, o robô tem de estar na posição inicial (18,18) e só a partir deste ponto é que as ações de como lidar com os eventos serão realizadas. O robô vai começar por se virar para a esquerda com o ângulo necessário para poder contornar o primeiro obstáculo sem colidir com o mesmo . Depois de colocado na direção correta, o robô vai percorrer em linha reta a distância dada por **e.getDistance()**. A esta distância, somamos metade do tamanho fixo (36 pixeis) do objeto para que este se posicione o mais próximo possível do mesmo. Para efetivamente contornar o objeto,

o robô vai virar 60º para a direita e percorrer 40 pixeis (comprimento do objeto de 36 pixeis somado com uma margem de erro de 4 pixeis). No final, a variável **turns** será incrementada para conter o número de objetos que já foram circum-navegados.

```
1  public void onScannedRobot(ScannedRobotEvent e) {
2          super.onScannedRobot(e);
3          if (scanningObject){
4              turnLeft(e.getBearing()*1.5);
5              ahead(e.getDistance()+18);
6              setTurnRight(60);
7              ahead(40);
8              this.turns++;
9          }
10     }
```

O método run() começa por colocar o robô na posição inicial e fá-lo esperar um certo número de *ticks* para que os outros robôs se coloquem em posição. Assim que a corrida começa, o robô procura identificar outros no mapa, e quando o faz, entra no evento *OnScannedRobot*. Quando a variável *turns* toma o valor 3 (ou seja, já contornou 3 robôs) pará de fazer *scan* e dirige-se para a posição inicial (18,18). Finalmente, a corrida termina e o odómetro imprime a distância total percorrida.

O robô apresenta, na distância percorrida, uma eficiência a rondar os 91% (relativamente ao perímetro ideal a percorrer).

3.2 Fase 2

Tal como havia sido indicado na secção anterior, nesta fase pretende-se conceber equipas de 5 elementos que demonstrem Comportamentos Sociais e de grupo e que permitam discutir e aplicar os conhecimentos adquiridos na UC de Agentes Inteligentes. Nomeadamente, pretende-se desenvolver a cooperação entre agentes num Sistema Multi-Agente, com o objetivo de resolver um problema comum.

Assim, de modo a ser possível experimentar várias possibilidades e treinar os vários conhecimentos, implementámos as três equipas abaixo apresentadas, que possuem comportamentos distintos.

3.2.1 SA Team

Esta equipa é constituída por um robô da classe *SA* (que funciona como líder), quatro robôs da classe *buddies* e tem uma utilidade meramente lúdica, uma vez que não permite o ataque. Nesta equipa, os quatro robôs *buddies* **seguem o seu líder** que os leva a "escrever"as letras *SA*, iniciais da UC que solicitou a realização deste trabalho, percorrendo para isso o caminho demarcado a vermelho (como mostra a figura 3.3).

Através da construção desta equipa, foi possível treinar a comunicação entre os vários Agentes (robôs) e a coordenação entre todos eles na procura da rota que lhes permite seguir o líder e evitar o choque.

O diagrama 3.4 esquematiza a comunicação envolvida entre os Agentes e as decisões tomadas para conseguir realizar o objectivo a que a equipa se propõe.

O processo começa quando o líder envia o seu nome aos membro da equipa. Estes, depois de receberem o nome do líder, enviam-lhe uma mensagem a identificarem-se.

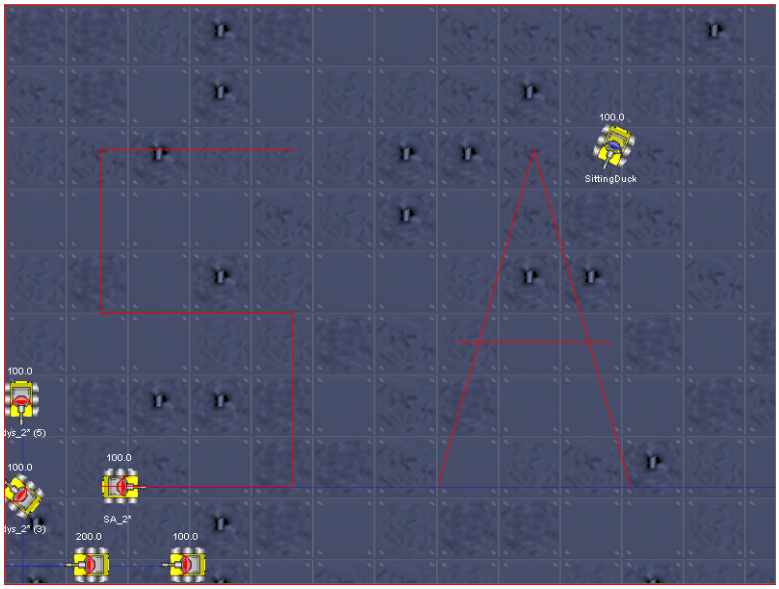

Figura 3.3: Caminho percorrido pelos robôs.

Em seguida o líder calcula as posições que cada elemento da equipa deve ocupar e envia-lhas por mensagem. Cada elemento lê a mensagem com as coordenadas que deve ocupar e dirige-se para a posição inicial. Quando a mesma é alcançada, enviam uma confirmação ao líder. Este, depois de receber as 4 confirmações dos elementos da sua equipa a indicar que já estão na posição inicial, dirige-se para a primeira posição e começa o percurso enviando aos restantes elementos uma mensagem para lhes dar autorização para seguir atrás dele. O trajeto termina quando o líder chega a posição final e todos os robôs buddies param atrás dele.

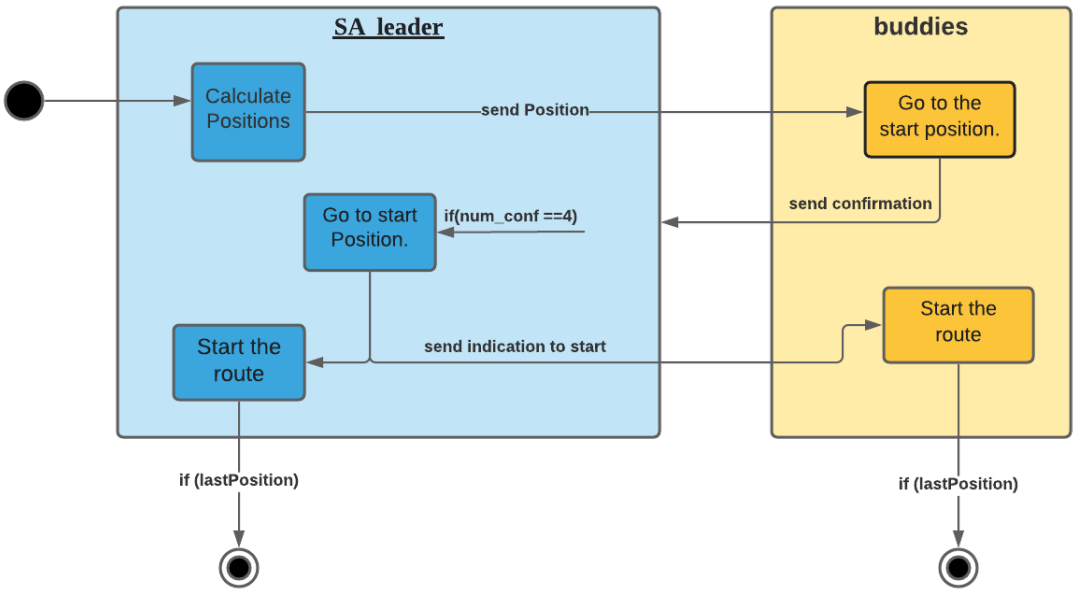

Figura 3.4: Diagrama de estados.

3.2.2 NSYNC

Esta equipa é constituída por um robô líder e quatro *droids*, da classe `JustinTimbs`, que cooperam para efetuar uma dança sincronizada, não se preocupando com os adversários. Na figura 3.6 podemos observar o diagrama de estados da interação entre os membros da equipa. Quando esta equipa é colocada em campo, o líder começa por se deslocar para o centro do mapa e mandar uma mensagem a cada membro da equipa, indicando a posição para a qual se devem deslocar. Quando os *JustinTimbs* já se encontram no local indicado e após o líder ter recebido a confirmação que todos estão no lugar, o líder dá ordem para todos iniciarem a dança da vitória do Robocode. A figura 3.5 ilustra uma das figuras geométricas que os robôs definem no mapa, para realizar a dança da vitória.

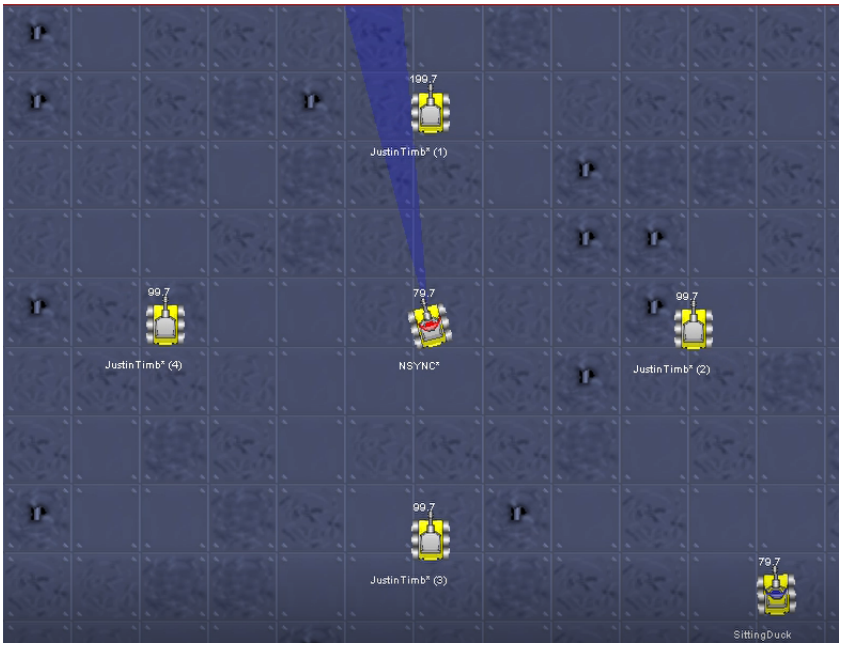

Figura 3.5: Comportamento da equipa *NSYNC*

Este processo repete-se até que os robôs passem por todas as posições da figura geométrica que estão a definir. Esta equipa, que coopera para obter o resultado visual pretendido, é centralizada uma vez que todos dependem das indicações fornecidas pelo líder.

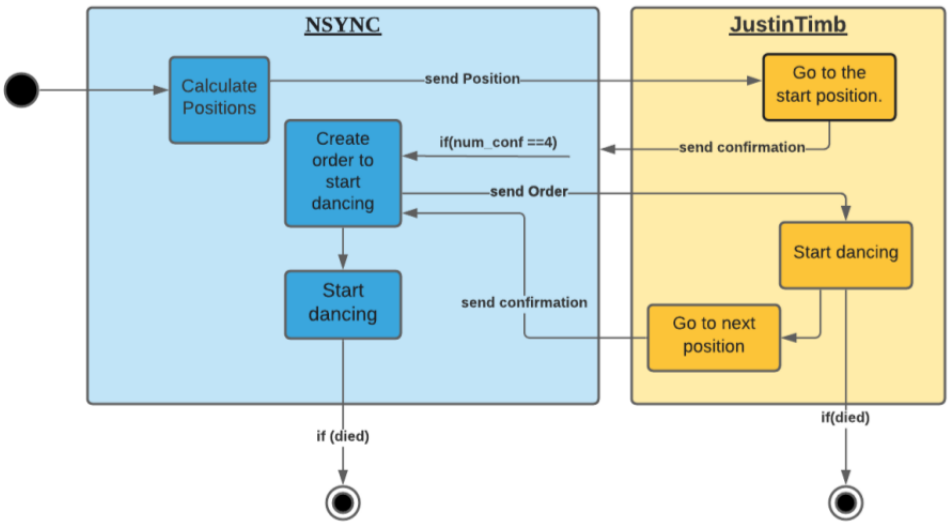

Figura 3.6: Diagrama de estados da equipa *NSYNC*.

3.2.3 Tracker Team

Esta equipa é constituída por um robô líder `TrackerLeader` e quatro robôs `TrackerSquad`. O objetivo desta é proteger o líder pelo que os restantes membros da equipa são colocados numa formação em semicírculo (ver figura 3.7).

O processo inicia-se com o *broadcast* de uma mensagem por parte do líder que tem como conteúdo o próprio nome. Esta mensagem será recebida por todos os robôs presentes no campo de batalha mas apenas os membros da equipa saberão lidar com a mesma. Assim que os *teammates* `TrackerSquad` recebem o nome do seu líder, respondem com o nome com que se identificam. No final desta breve trocas de mensagens, o líder vai saber a identificação dos restantes membros da equipa.

De seguida, o líder vai gerar o conjunto de posições iniciais que serão enviadas para os *teammates*. Estas posições vão colocar o TrackerLeader no topo do campo de batalha e os quatro TrackerSquads a rodear o mesmo, criando uma formação de modo a proteger o líder dos inimigos.

Assim que se encontrem nas suas posições, o líder vai iniciar o *scan* à procura do inimigo com menor energia. Caso os inimigos tenham todos a mesma energia, o primeiro a ser detetado é o escolhido. Após a deteção do inimigo, as suas informações (nome, posição, etc.) são enviadas para os *teammates*.

Como apresentado na figura 3.7, esta formação em particular divide o campo de batalha a meio e coloca dois robôs no lado esquerdo e dois robôs no lado direito criando assim dois quadrantes. Assim que os TrackerSquads receberem a informação do inimigo, vai ser verificado se este se encontra no quadrante respetivo, ou seja: caso o inimigo se encontre no quadrante da esquerda/direita, os dois TrackerSquads mais à esquerda/direita no campo vão atirar sobre ele enquanto os outros dois ficam em espera. Assim que o inimigo atravessar a "fronteira", os robôs que estavam previamente em espera irão agora abrir fogo.

Este método permite-nos evitar o conhecido "fogo amigo"visto que se um inimigo se encontrasse no quadrante esquerdo superior, os robôs mais à direita iriam acertar nos

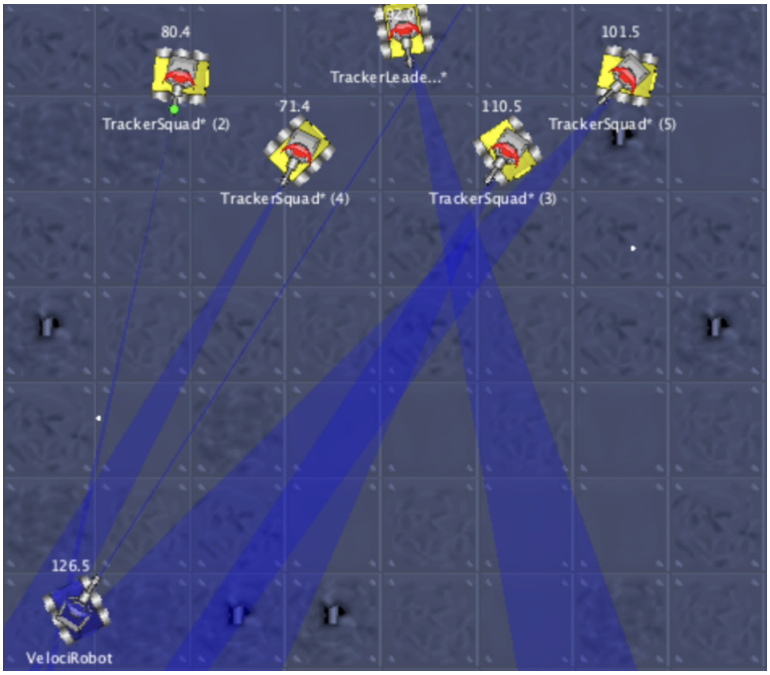

Figura 3.7: Formação em batalha da equipa *TrackerTeam*.

membros da sua equipa na tentativa de atingir o inimigo que foi identificado. Permite também controlar de certa forma o fogo que atinge um inimigo pois há mais probabilidade de atingir um alvo em movimento quando este se encontra numa trajetória mais próxima.

Caso o inimigo morra e ainda existam inimigos no campo de batalha, o líder vai identificá-los e enviar de novo a informação do inimigo com menor energia. Caso contrário, termina aqui a batalha. Na figura 3.8 encontra-se esquematizado o protocolo de comunicação e ações entre os membros da TrackerTeam que foi descrito anteriormente.

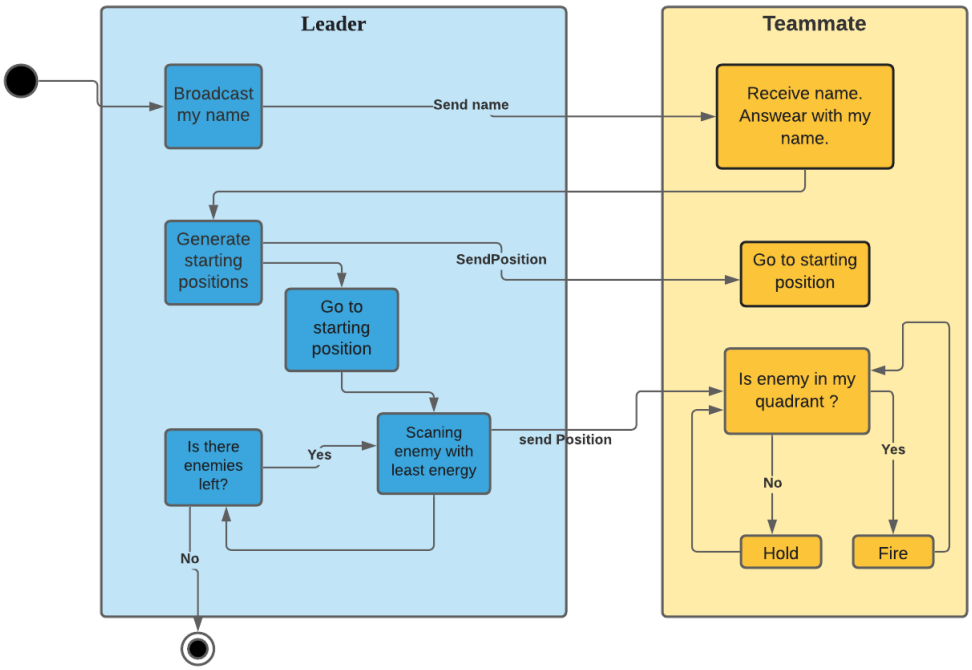

Figura 3.8: Diagrama de estados do funcionamento da comunicação entre membros da equipa TrackerTeam.

3.3 Fase 3

A terceira e última etapa deste trabalho consistiu na concepção e formação de uma equipa para posterior avaliação da sua performance em competição contra outra. A equipa criada possui 3 tipos de robôs diferentes, que apresentamos em seguida:

- **CaptainAmerica** - É do tipo *TeamRobot* e representa o líder da equipa, apresentando comportamentos defensivo e ofensivo.
- **Avenger** - Do tipo *TeamRobot*, com um comportamento de ataque direcionado.
- **Groot** Do tipo *Droid*, com um comportamente aletório e imprevisível mas também com ataque direcionado.

A equipa criada para combate é composta por um robô da classe *Captain America*, dois da classe *Avenger* e dois da classe *Groot*. Na figura 3.9, encontra-se esquematizado o protocolo de comunicação e as principais ações realizadas entre os membros da equipa no decorrer de uma batalha.

CaptainAmerica

Este robô é o líder da equipa apresentada. Como foi referido anteriormente, apresenta comportamento defensivo e ofensivo, sendo que o comportamento defensivo é o primeiro a ser adotado. O objetivo inicial do líder é esconder-se num canto, fazendo, a partir daí, um *scan* do campo e informando posições e/ou nomes dos inimigos aos seus colegas.

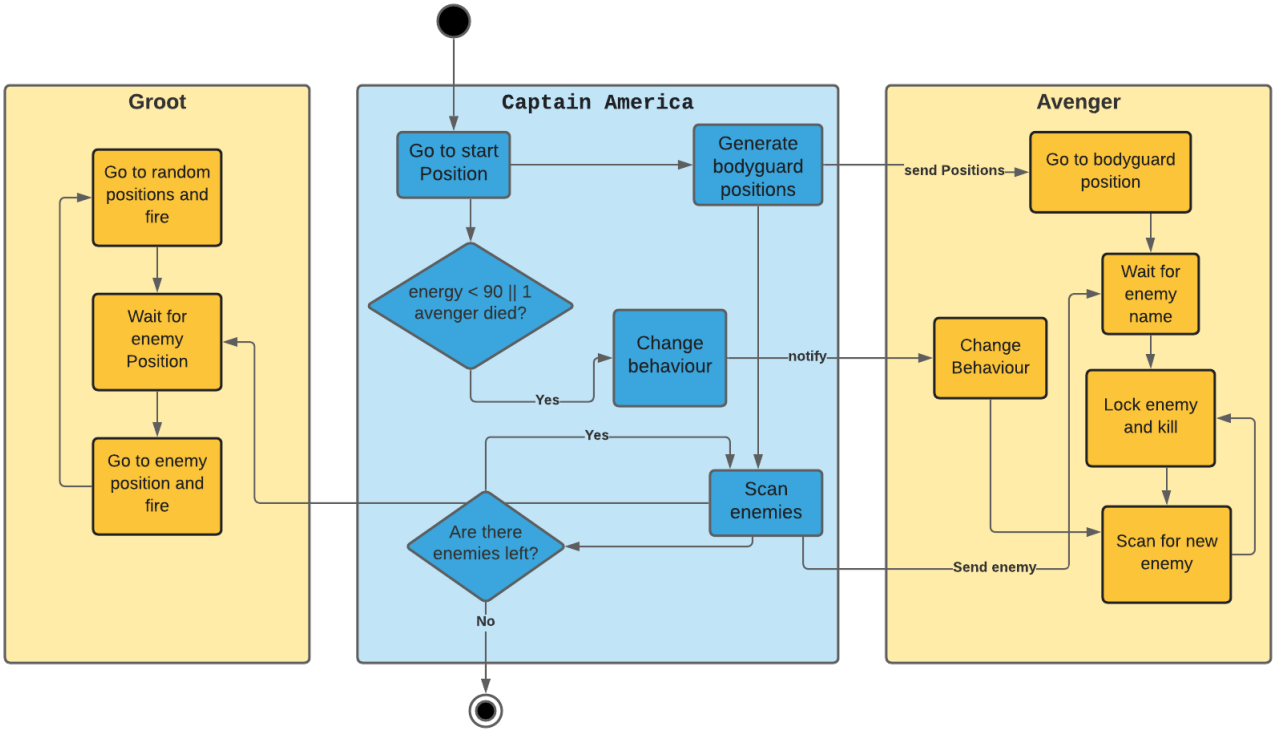

Figura 3.9: Diagrama de estados do funcionamento da comunicação entre membros da equipa.

Este robô começa por procurar qual dos cantos do campo de batalha está mais próximo de si e informa essa posição aos *Avengers*. Estes robôs, uma vez que desempenham a função de guardas do líder, irão deslocar-se e posicionar-se de forma a proteger o líder contra ataques inimigos.

O líder encontra-se responsável por delegar aos outros elementos da equipa o inimigo que cada um irá atacar. Uma vez que os *Groots* não possuem *scan*, o líder envia todas as posições do inimigo que encarregou a cada um destes robôs.

Este comportamento será adotado pelo *CaptainAmerica* até existir risco de ameaça. Na iminência de tal, o líder toma uma atitude ofensiva. Quando a sua energia é menor que 90 ou um *Avenger* morre (o líder fica exposto), o CaptainAmerica toma a estratégia do robô *Walls* [1] e percorre o perímetro do campo, atacando os inimigos que encontre e continuando a enviar a informação dos inimigos aos restantes colegas de equipa. O excerto de código 3.1 implementa a estratégia referida.

Fonte de código 3.1: Excerto do Evento *OnScannedRobot*, que lida com a estratégia ofensiva do *Captain America*

```
1   public void onScannedRobot(ScannedRobotEvent e) {
2
3       // Enter wallsMode and the scanned robot isnt a teammate
4       if (wallsMode && !teamsArray.contains(e.getName())) {
```

[1] Mais informação em `https://robowiki.net/wiki/Walls_(robot)` e `https://github.com/robo-code/robocode/blob/master/robocode.samples/src/main/java/sample/Walls.java`

```
5
6       //get the distance of the scanned robot
7       double distance = e.getDistance();
8
9       //Managing power according to distance to enemy
10      if(distance > 800)
11          fire(5);
12      else if(distance > 600 && distance <= 800)
13          fire(4);
14      else if(distance > 400 && distance <= 600)
15          fire(3);
16      else if(distance > 200 && distance <= 400)
17          fire(2);
18      else if(distance < 200)
19          fire(1);
20
21      if (peek) {
22          scan();
23          }
24      }
25
26      // Send enemys positions to Droids
27      if(!teamsArray.contains(e.getName())){
28
29          double enemyBearing = this.getHeading() + e.getBearing();
30          double enemyX = getX() + e.getDistance() *
            ↪  Math.sin(Math.toRadians(enemyBearing));
31          double enemyY = getY() + e.getDistance() *
            ↪  Math.cos(Math.toRadians(enemyBearing));
32          try {
33              broadcastMessage(new Enemy(e.getName(), enemyX,
                ↪  enemyY));
34          } catch (IOException ex) {
35              ex.printStackTrace(out);
36          }
37      }
38  }
```

Avenger

Estes robôs representa o guarda do *CaptainAmerica* e possui um comportamento de ataque direccionado. Inicialmente coloca-se nas posição atribuída pelo seu líder de forma a protegê-lo. Mal receba o nome do inimigo que deve atingir, começa a disparar até este morrer.

Morrendo o seu alvo inicial, toma a iniciativa de usar o seu *scanner* para procurar outros inimigos e assim ajudar os seus colegas.

Da mesma forma que acontece com o Capitão América, este robô apresenta duas estratégias: ser guarda do líder ou ser um mero atacante. O *trigger* que indica a mudança de estratégia depende do líder. Estes apenas mudam de comportamento

quando o líder também muda. Quando isso acontece, o *Avenger* abandona a sua posição (pois até agora se encontrava numa posição fixa de forma a receber e proteger o líder contra os disparos), e começa a perseguir o inimigo que encontra (no caso de o seu inimigo inicial ter morrido, ataca diretamente o primeiro que encontrar).

Groot

Este robô é um *Droid*. Apresenta um comportamento aleatório, de forma a não ser um alvo previsível. Este comportamento encontra-se definido na função *run()*, que é apresentada no excerto de código 3.2. A função define três tipos de rotas que o robô pode adotar (1, 2 ou 3). A rota que o robô vai tomar é escolhida aleatoriamente.

Fonte de código 3.2: Comportamento do *Groot*

```
fire(3);
Random r = new Random();
int option = r.nextInt(2);  // Defines the route that it is going to
↪  take.
if(option==0){
        setTurnRight(270);
        ahead(500);
        fire(3);
}else if (option == 1){
        setTurnLeft(180);
        ahead(400);
        setTurnRight(180);
        ahead(400);
        fire(3);
}
else {
        setTurnLeft(180);
        setTurnRight(180);
        back(400);
        fire(3);
}
```

Visto que não possui *scan* próprio, depende da passagem de informação das posições dos seus inimigos por parte do líder. Recebendo estas posições, calcula o ângulo até ao alvo referido, ataca e avança.

3.3.1 *Performance* da Equipa

Tal como foi referido anteriormente, o principal objetivo desta fase consiste em vencer os combates contra as equipas desenvolvidas pelos restantes grupos. Na aula do dia 22/03/2021 realizou-se o primeiro combate e a nossa equipa não passou dos quartos de final, ficando em sexto lugar num total de onze equipas. Chegou-se à conclusão que o comportamento dos *Groots* teria de ser melhor desenvolvido.

Capítulo 4

Conclusão

Ao longo da resolução deste trabalho foram surgindo várias dificuldades. Na primeira etapa, a definição da estratégia ótima para circum-navegação e a familiarização com o ambiente do Robocode foram os principais impasses. No entanto, conseguimos obter uma eficiência satisfatória na circum-navegação proposta, a rondar os 90%.

Na segunda fase, a maior adversidade passou pela transição da idealização das estratégias para a implementação. Surgiram vários erros, nomeadamente, na obtenção dos membros da mesma equipa com a utilização da função *isTeammate()* por esta se encontrar mal definida. Este pequeno detalhe forçou que fosse implementado um protocolo de comunicação adicional para permitir estabelecer a equipa em campo. Foram desenvolvidas duas equipas lúdicas que demonstraram claramente o funcionamento correto das comunicações entre os membros e comportamentos como a coordenação. A última equipa apresentou comportamentos ambos defensivos e agressivos, visto que o líder era inicialmente protegido e posteriormente alterava a estratégia para atacar.

Na última fase, foram implementadas estratégias de controlo *Feedforward* e *Feedback*. Com estas estratégias, foi possível tomar partido dos sensores embutidos nos robôs (radares) para poder monitorizar o ambiente que os rodeia (inimigos, paredes, entre outros.), de forma contínua, e ajustar as ações dos robôs de acordo com a interpretação dos dados do ambiente.

A estratégia e os comportamentos implementados não mostraram ser os mais eficientes. O comportamento aleatório dos *Groots* é agil mas não é competente o suficiente para obter resultados satisfatórios. A implementação de formações em batalha é igualmente pouco vantajosa.

O grupo considera, no entanto, ter cumprido os requisitos do projeto com sucesso e que o trabalho desenvolvido reflete não só o trabalho desenvolvido no decorrer da disciplina, mas também os objetivos delineados para o mesmo de uma forma fidedigna.