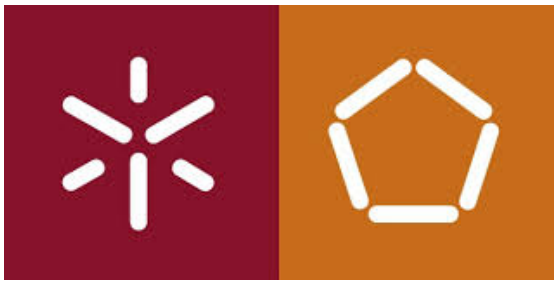

Universidade do Minho - Escola de Engenharia

Mestrado Integrado em Engenharia Informática

Sistemas Autónomos

Trabalho prático
RoboCode

Carlos José Gomes Campos a74745
José Pedro Ferreira Oliveira a78806
Vitor José Ribeiro Castro a77870

9 de Março de 2019

Resumo

Este relatório pretende descrever os passos necessários à resolução de vários desafios propostos pela equipa docente. Desde a arquitetura escolhida, até à implementação final, passando por todas as fases de teste, no desenvolvimento para o RoboCode, no âmbito da unidade curricular de Sistemas Autónomos, do perfil de Sistemas Inteligentes, do Mestrado Integrado em Engenharia Informática.

Conteúdo

1 Introdução

1.1 Contextualização

O RoboCode é um jogo de programação em que o objetivo é programar um robot de batalha para competir contra outros. O programador não tem qualquer impacto na ação do seu robot em tempo real, ou seja, deve desenvolver algum género de Inteligência Artificial que supere os oponentes.

Os robots podem atuar de forma individual ou em equipas, sendo que neste trabalho se pretendeu, na fase final, apresentar uma equipa com vários tipos de robot. A programação é, por isso, enquadrada na Unidade Curricular de Sistemas Autónomos.

1.2 Caso de Estudo

O projeto tem por base a conceção de soluções para a resolução de um conjunto de desafios propostos, que pretendem a integração de conceitos de agentes inteligente e sistemas multiagente, no desenvolvimento e programação de sistemas autónomos, utilizando o ambiente de programação RoboCode. Numa primeira fase, o desenvolvimento de um odómetro e contorno de obstáculos, que resulta numa última fase numa equipa completa.

1.3 Estrutura do Relatório

O relatório está dividido em três partes, correspondentes aos três desafios propostos.

Na secção 2 são explicadas as várias soluções idealizadas para a implementação de um odómetro para um robot. Na secção 3 será descrito o processo de construção da arquitetura que permite o contorno dos obstáculos com otimização do percurso percorrido. Na secção 4 serão explicadas as decisões acerca dos comportamentos dos robots que constituem a equipa, para ser usada em batalha, mais propriamente os algoritmos usados pelos mesmos face à necessidade de negociar/agir com outros elementos do sistema, quer sejam elementos da equipa ou inimigos.

O relatório termina com as conclusões na secção 5 onde é feita uma reflexão acerca do trabalho realizado bem como do trabalho futuro.

2 Odómetro

Numa implementação inicial do odómetro, a solução passa por reconhecer uma série de eventos que levam à alteração da posição do robot e, portanto, nessas situações a variável que guarda o valor da distância percorrida deverá ser atualizada. Os movimentos possíveis são alcançados através dos métodos *back()* que movimenta o robot para trás e *ahead()* que movimenta o robot para a frente.

Posto isto, quando estes métodos são invocados, o valor do odómetro é atualizado, através da formula de calculo da distância entre dois pontos, de acordo com o figura 1.

$$d = \sqrt{(x2 - x1)^2 + (y2 - y1)^2}$$

Figura 1: Distância entre dois pontos

Posteriormente, verificou-se que a implementação anterior era aceitável num Robot, no entanto, num AdvancedRobot podia não ser tão precisa, uma vez que é possível realizar movimentos em paralelo, ou seja, virar e deslocar em simultâneo, e assim realizar uma curva. Com isto, o grupo idealizou duas estratégias para resolver o desafio, uma utilizando a formula de cálculo em 1, e a outra utilizando a fórmula da velocidade de acordo com a figura 2, em que ambas seriam executadas num período de um *Tick/Turn* (unidade de tempo no RoboCode) levando a uma atualização periódica do valor do odómetro.

$$d = |v| * t$$

Figura 2

Ambas as soluções foram implementadas o que permitiu realizar testes para verificar qual delas é a mais precisa, verificando-se que a alternativa que utiliza o calculo através das coordenadas cartesianas é a que apresenta resultados mais consistentes.

A solução final passa por criar um *CustomEvent* com uma condição de teste *Timer* que é acionada de tick em tick, sempre que seja verificada são guardadas as coordenadas da posição atual do robot e é feito o calculo da distância entre essas coordenadas e as anteriores, através da fórmula anteriormente referida na figura 1, atualizando o valor do odómetro.

3 Contorno de obstáculos

O objetivo deste desafio passa por deslocar o robot segundo uma trajetória que permita contornar três obstáculos. O robot deve partir da posição (18,18) e iniciar um percurso próximo dos três robots dispersos por três quadrantes diferentes de acordo com a figura 3, voltando à posição inicial.

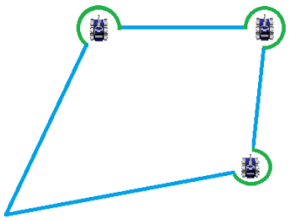

Figura 3: Percurso proposto

Depois de analisar as possíveis melhoras em relação à distância percorrida, foi fácil perceber que contornar os obstáculos podia causar bastante *overhead*.

De um modo geral, a solução idealizada por este desafio vai de encontro com a movimentação representada na figura 4, que consiste em fazer uma movimentação em forma de quadrilátero, de modo a conter todo o percurso, e assim diminuir o perímetro da figura, que representa a trajetória a executar.

Para que tal aconteça, é feita a deteção do ponto tangente a cada obstáculo, à medida que o robot vai executando o percurso, de maneira a que não haja colisão entre os dois robots. Para uma otimização final, é feita uma trajetória curvilínea nas mudanças de direção, de forma a conseguir melhorar ainda mais a distância percorrida, obtendo uma solução quase ótimo e sem recurso a grandes cálculos computacionais. É importante referir a questão do consumo de recursos computacionais, uma vez que quando o campo de batalha tem muitos robots pode tornar-se complicado executar todos os cálculos em tempo útil, resultando em comportamento inesperado observado pelo grupo nas suas experiências.

Figura 4: Movimentação idealizada para o desafio

Inicialmente, o robot nasce num local aleatório e é necessário desloca-lo para o ponto inicial do percurso através da função *void go(double x, double y)*.

Posteriormente, o robot é virado no sentido positivo do eixo do X e inicia viragens sucessivas para a direita até encontrar um robot utilizando o radar. Caso isso se verifique, o robot roda para a esquerda segundo um ângulo de getBearing()*2, e inicia uma deslocação em linha reta, percorrendo a distância até esse robot, de seguida efetua uma curva de 45º para a direita e inicia novamente a busca por outro robot.

O algoritmo termina quando encontrar os três robots, altura em que o robot se desloca para a posição inicial.

Foram também averiguadas situações em que existe uma colisão com uma parede, isto acontece quando um *RockQuad* está muito próximo de um limite do campo de batalha. Quando isto acontece, o robot volta-se para a parede, executa uma volta de 90º para a direita e segue em frente uma quantidade de pixeis suficientes para contornar o obstáculo.

4 Implementação da equipa

Para a implementação de uma equipa capaz de competir numa batalha, é necessário estabelecer um conjunto de sistemas de controlo, tanto a nível individual como coletivo.

Os comportamentos individuais têm como objetivo assegurar uma base de ação para um robot, face aos desafios que vai encontrando ao longo do duelo, sendo que, geralmente, estes comportamentos estão associados a robots com capacidade sensorial que lhes permite monitorizar o ambiente que os rodeia.

Os comportamentos coletivos surgem da necessidade de comunicação entre elementos da equipa para atingir um objetivo comum, mas também para assegurar um bom rendimento de robots com menor capacidade de monitorizar o ambiente.

Num cenário de batalha, o ambiente altera-se frequentemente, pelo que é necessário criar robots com capacidade de reconhecimento e de liderar, bem como robots com maior capacidade de luta.

A nossa estratégia consiste na combinação de uma arquitetura mais cooperativa, com a uma arquitetura mais individualista.

A equipa implementada é composta por três tipos de robots, dois robots *Estrelinha* e um robot *Fuinha* que são do tipo *TeamRobot*, e dois *Camões* que são do tipo *Droid*, com maior capacidade de combate. Estas escolhas foram baseadas na comparação de resultados que os robots de exemplo do robocode obtinham. O grupo concluiu, através de pesquisas junto de outros utilizadores do robocode, que os mais eficientes robots de equipa era aqueles que melhor se saiam contra muitos outros (*melee*). A imagem seguinte mostra os resultados desse comparativo.

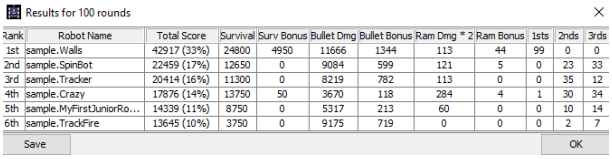

Figura 5: Top robots individuais

4.1 Robot Estrelinha

Este robot é o que exibe uma implementação mais complexa pois, para além de analisar o ambiente constantemente, possui uma capacidade de movimentação e disparo muito forte no combate individual. A arquitetura tem por base uma política *FeedBack*, os detalhes de implementação serão descritos de seguida.

Foi criada uma estrutura de dados auxiliar RobotData para salvaguardar a informação pertinente sobre um robot inimigo, bem como um LinkedHashMap para armazenar essas informações sobre cada elemento da equipa adversária.

```
    class RobotData {
final String name; // nome do robot
double scannedX; // coordenada x do robot atualizada
double scannedY; // coordenada y do robot atualizada
double scannedVelocity; // velocidade do robot
double scannedHeading; // heading do robot
double targetX; // coordenada x prevista para efetuar disparo
double targetY; // coordenada y prevista para efetuar disparo
...
}
```

O comportamento base deste robot, passa por, consecutivamente, executar as seguintes ações:

- Ajustar o radar de acordo com uma direção pré-calculada que será explicada de seguida, de modo a minimizar o espaço de procura por possíveis inimigos.
- Atualizar o alvo a abater, caso não tenha nenhum definido ou caso já tenha passado mais de vinte ticks desde a última atualização, aqui existe um processo de filtragem para averiguar quais os robots que estão fora do alcance e quais os que se encontram a menor distância, com o intuito de melhorar a eficiência dos disparos e prevenir ataques a colegas de equipa. De seguida, recalcular a posição da arma para disparar, através do calculo do ângulo entre a posição anterior da arma e a posição atual do alvo. Finalmente, verificar se o ângulo da arma é inferior ao ângulo que cobre o centro do robot até à borda, caso isso se verifique é bastante provável que o disparo atinja o alvo e, portanto, o mesmo é realizado.

- Movimentar-se após avaliar a situação em que se encontra. Em primeiro lugar é verificado se o robot tem a possibilidade de se movimentar verticalmente e/ou horizontalmente de acordo com a distância ao limite do campo de batalha. Com isto, calcula-se qual a menor distância a percorrer até ao alvo e avança-se na vertical ou horizontal, sendo que este robot mantem-se junto aos limites. Caso se encontre próximo do alvo, o robot deve assumir um movimento alternado, de maneira a ser mais difícil ser atingido pelo inimigo.

- Focar durante, pelo menos, 100 *ticks* num adversário, de modo a que não se mude constantemente de adversários e seja ineficiente. Para além disso, ajusta a potência da bala de acordo com a distância em relação ao alvo, de modo a ajustar a relação recompensa/risco associada a um disparo.

Para além dos comportamentos referidos anteriormente, existem eventos que despoletam ações como é o caso de:

- onScannedRobot

  A cada reconhecimento de um robot, caso não seja um membro da equipa, é feita uma inserção na estrutura de dados ou uma atualização com dados mais recentes acerca do robot detetado.

  Adicionalmente, é atualizada a direção para a qual deverá ser feito o próximo scan com o radar, de maneira a reduzir o espaço de busca, tendo por base a posição do robot que foi detetado e o heading do radar do nosso robot, tornando o scan mais eficiente.

  Para além disso, é também feita uma atualização às posições para onde se prevê que o disparo atinja um determinado robot com sucesso, para isso faz-se um calculo baseado em *Linear Targeting*.

  Finalmente, na ocorrência deste evento o robot *Estrelinha* auxilia o robot *Fuinha* na identificação do ambiente de batalha e divulga o nome e a posição do inimigo aos colegas de equipa.

- onRobotDeath

  A cada morte de um robot, é necessário verificar se se trata de um inimigo e nesse caso remove-se do LinkedHashMap, bem como das variáveis responsáveis por guardar as informações do alvo atual, caso seja o mesmo que acabou de ser destruido.

Este robot constitui a arquitetura mais individualista da nossa estratégia anteriormente mencionada.

4.2 Robot Fuinha

Este robot tem a função de monitorizar os inimigos no campo de batalha e fornecer informações. Deste modo, efetua um scan ao ambiente que o rodeia e em caso de detetar um inimigo envia o nome e a posição em broadcast aos colegas de equipa.

Para além disso, exibe uma estratégia de movimentação aleatória e dispersa, assegurando um maior tempo de vida em combate devido à dificuldade de ser atingido.

Esta mesma movimentação é regida através de uma variável, à qual é atribuído um valor aleatório, podendo ser 0 ou 1. Em cada iteração do ciclo de vida do robot, quando o valor da variável for 1, o robot executa uma movimentação em forma de "laço", por oposição a uma trajetória em forma de "S"quando a mesma é 0, tal como é demonstrado na figura 6.

Figura 6: Movimentação idealizada para o desafio

Para os eventos que afetem o movimento, como por exemplo o *HitByBulletEvent*, ou o *HitRobotEvent*, também é executado uma alteração do movimento que é aleatório.

4.3 Robot Camões

O robot Camões não possui radar e depende da informação fornecida pelos seus companheiros para desempenhar a sua função. Sempre que o evento *onMessageReceived* é chamado, o *Camões* faz a previsão da posição do inimigo que recebeu, efetua um disparo e no fim executa uma pequena movimentação de 100 pixeis na sua direção.

Esta pequena movimentação, que pode parecer muito simples e até desnecessária, tem o intuito de tornar a movimentação dos *Droids* o mais imprevisível possível no inicio da ronda, e à medida que os robots adversários vão sendo derrotados, os *Camões* movimenta-se em direção aos robots restantes, causando uma desvantagem numérica a favor da equipa.

Caso este robot não receba informação de nenhum robot da sua equipa, porque estes foram derrotados, ele executa uma movimentação aleatória idêntica à do robot *Fuinha*, sendo que também para além de se movimentar, vai disparando entre deslocações.

Estes dois últimos robots constituem a arquitetura cooperativa, anteriormente mencionada.

5 Conclusão

Em relação ao primeiro desafio, decidiu-se usar um método que se aproximasse o máximo possível do cálculo do comprimentos de curvas através de uma integração. Apesar de causar um aumento da carga computacional, visto que os cálculos são ligeiramente simples, decidiu-se continuar com essa decisão e, assim, conseguir obter resultados mais precisos do que o odómetro que foi fornecido para comparação.

No que consta ao segundo problema, foi adotada uma implementação que desse uma maior prioridade à autonomia do sistema, e menos prioridade à preocupação em fazer a menor distância possível ao percorrer o obstáculo. Concluindo que a solução apresentada para este desafio possui uma arquitetura de navegação à base do **planeamento episódico**, visto que existe um plano já prototipado antes da execução, mas caso existe alguma mudança inesperada, como o robot bater numa parede, ele é capaz de recalcular a rota.

Na questão da batalha, apesar do ser mais aconselhável fazer uma equipa de 5 robots do mesmo tipo, foi decidido fazer uma equipa diversificada, em que existem vários tipos de comportamentos sociais, como um robot mais covarde como o *Fuinha*, ou robots mais agressivos como o *Camões*, ou então robots mais calculistas como o *Estrelinha*. Aproveitando assim também para demonstrar o uso da arquitetura de Líder/Droids.

Para finalizar a análise do segundo desafio, é de frisar que a estratégia de controlo no robot *Camões* é **Open Loop**, já que o mesmo não possui radar, sendo que toda a informação que possui é enviada pelo seu líder, ou pelos seus sensores internos. A estratégia de controlo do robot *Fuinha* é **Feedforward** já que o mesmo utiliza o seu radar apenas para receber informações sobre o ambiente e envia-los para os *Droids*. Por último, o robot *Estrelinha*, possui uma estratégia de controlo de **Feedback** já que para além de monitorizar continuamente o ambiente à sua volta, o seu comportamento muda consoante a informação que é recebida. Em relação às arquiteturas de controlo, todas elas são tipo **Híbrida**, já que todos os robots possuem uma parte **Deliberativa** que já contém um plano para o desenvolvimento da batalha, mas também possui uma componente mais **Reativa**, que é disparada quando certos eventos acontecem, como o *onHitWallEvent*.

Foram encontradas diversas dificuldades no decorrer da elaboração do projeto, principalmente na última parte do mesmo. Fatores relacionados com a concorrência ao acesso em informações partilhadas não permitiam a correta remoção ou adição de inimigos encontrados. Adicionalmente, devido ao grande fluxo de comunicação e necessidade de atualizar os dados dos inimigos, houve momentos em que determinadas mensagens não chegavam a todos os robots. Posto isto, foram adotadas diversas estratégias que permitiram a resolução destes problemas, apesar de se acreditar que há limites impostos pelo poder computacional disponível.