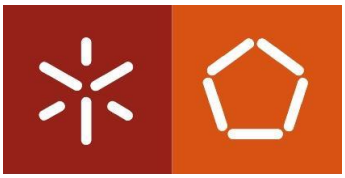

Universidade do Minho

Departamento de Informática

Robocode
Sistemas Autónomos

Bruno Martins (a80410) Catarina Machado (a81047)
Filipe Monteiro (a80229) Jéssica Lemos (a82061)

8 de Março de 2020

Conteúdo

Resumo

Este relatório tem como objetivo descrever todo o processo de tomada de decisão bem como da implementação de soluções às duas fases do problema apresentado pelos docentes da Unidade Curricular de Sistemas Autónomos do perfil de Sistemas Inteligentes no âmbito do projeto de programação de *robots* na competição *Robocode*.

1 Introdução

O *Robocode* é um jogo de competição que permite aos jogadores programarem tanques de batalha (individuais ou equipas) com o objetivo de destruir as equipas adversárias. No presente relatório irão ser abordadas ambas as fases propostas no âmbito do trabalho prático da Unidade Curricular de Sistemas Autónomos. Numa primeira instância irá ser descrito o processo de cálculo da distância percorrida pelos *robots* no mapa de combate.

De seguida irá ser descrito o método utilizado para que seja possível uma circum-navegação de pontos no mapa representados por outros *robots*.

Por fim irá ser descrito o algoritmo de combate entre equipas de *robots* bem como a comunicação feita entre eles para maximizar o dano causado e algumas conclusões a retirar deste projeto.

2 Odómetro

Para a implementação do odómetro começamos por delinear a estratégia na qual temos de ter em consideração que o *round* inicia quando o robot parte da posição (18,18) e termina após a circum-navegação dos 3 objetos regressando a esta mesma posição. Desta forma, o solução desenvolvida passou por calcular a distância percorrida através da fórmula da distância entre dois pontos, ou seja:

$$d_{AB} = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

Figura 1: Fórmula da distância entre dois pontos

Através de uma variável que guarda o valor da distância percorrida, e sabendo a posição anterior e atual do robot é assim possível atualizar este valor. Para tal, tornou-se necessária a criação de um *CustomEvent* cuja condição de teste é acionada de *tick* em *tick*. Sempre que é verificada é então guardada a posição atual do robot e atualizada a anterior sendo assim possível calcular o valor da distância através da fórmula referida na Figura 1.

3 Circum-navegação

Numa primeira fase tivemos de desenvoler uma estratégia de deslocação de forma a circum-navegar os 3 *robots* dispostos no mapa. Devido à natureza do problema, tendo apenas como objetivo percorrer os 3 *robots* da forma mais eficiente possível (menor distância alcançável), nunca considerámos criar um *robot* que consegui-se circum-navegar um grupo de *robots*, a não ser na disposição fornecida pelos professores.

3.1 V1 do Movimento

Numa primeira fase, pensámos que a maneira mais eficiente de percorrer os 3 obstáculos seria deslocar-se em linhas retas, para os vértices "perfeitos"do quadrado mais pequeno que englobe um *robot*. Este vértice variava consoante a posição do nosso *robot* em relação ao próximo a cirum-navegar.

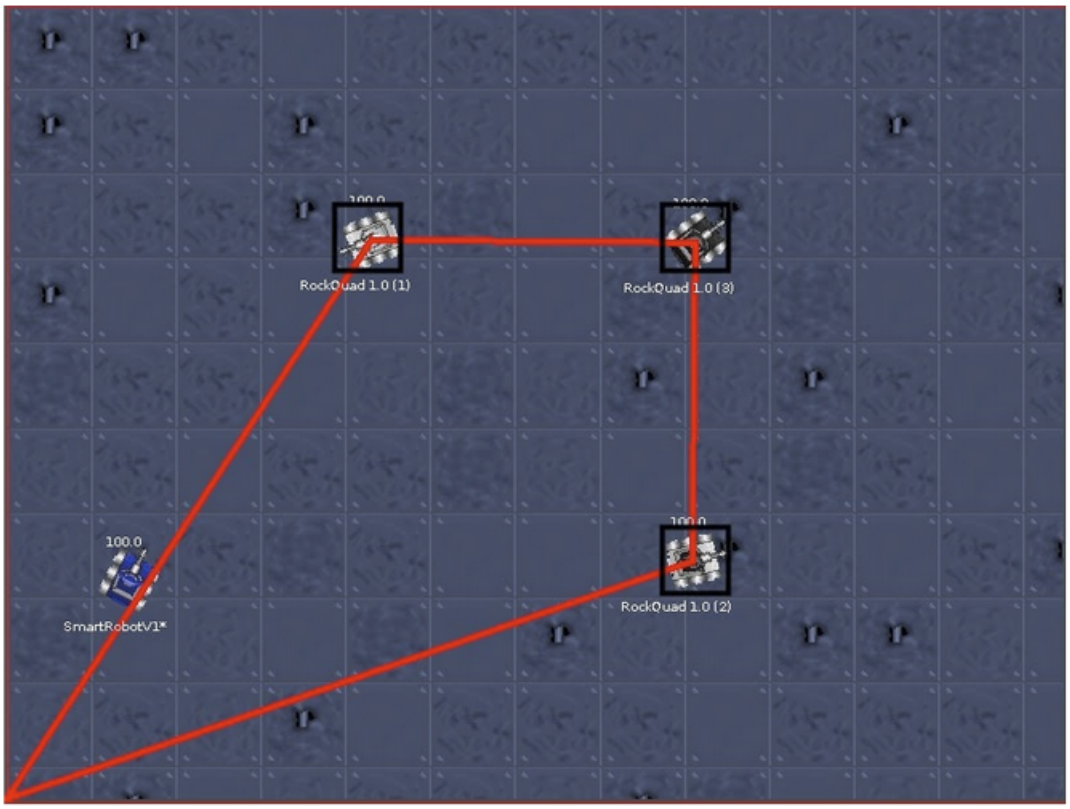

Figura 2: Quadrados envolvendo os robots.

Este quadrado era calculado utilizando o tamanho do *robot* (valor fixo de 36 pixeis) e *offsets* consoante a rotação do robot para evitar eventuais colisões. Depois de calcular o ângulo necessário a rodar para ir para a posição (x,y), este inica o deslocamento, andando no fim um pequeno extra pixeis. Este extra foi necessário porque, se o *robot* fosse estritamente para a posição indicada, consoante a posição do próximo objeto a circum-navegar, podia haver colisão. Este valor foi calculado com base em testes, tendo acabado em 12 pixeis para nunca colidir. Este movimento é por isso realizado deslocando-se:

- No primeiro obstáculo, para o vértice superior esquerdo;
- De seguida para o vértice superior direito;
- No ultimo obstáculo para o vértice inferior direito;
- Por fim deslocar-se para a posição (18,18).

Ao fim de 5 execuções, o *robot* percorreu uma média de 0.9027.

3.2 V2 do Movimento

Como havia tempo para explorar outras estratégias de deslocação, desenvolvemos uma outra onde o nosso *robot* percorria as arestas dos quadrados "perfeitos"mencionados em cima. Pensámos que iria ser mais bem-sucedido do que a versão apresentada anteriormente por causa de haver menos *offsets* e *magic numbers* no cálculo das posições a percorrer, mas no fim concluimos que a performance piorou substancialmente, apesar de ser mais segura.

No fundo, o nosso *robot* deslocava-se, no primeiro objetivo, em linha reta para o meio da aresta de baixo, percorrendo até ao meio da aresta esquerda, continuando para os outros obstáculos,

ajustando as arestas a percorrer. O cálculo dos pontos a deslocar-se foi realizado utilizando uma função que calculava 8 pontos do quadrado: 4 vértices e 4 pontos a meio de cada aresta, baseado no facto dos robos terem 36 pixeis de tamanho.

Comparativamente com a versão desenvolvida anteriormento, o *robot* percorreu uma média de 0.8467. Uma performance bastante inferior.

4 Combate

4.1 Robots

4.1.1 Bully

Este *robot* foi realizado inspirando-se numa equipa famosa do *Robocode* chamada de *PolyLunar* - cuja estratégia era, em pares, atacarem um inimigo cercando-o usando *close-up combat*. Achando que esta era uma estratégia muito agressiva e promissora, decidimos então implementar uma versão nossa.

Para facilitar as tomadas de decisões de quem atacar, quem notifica quem no momento de ataque, decidimos dividir este *Bully*: BullyLeader - cuja função é decidir o inimigo a atacar e notificar um amigo para iniciar o ataque - e o BullySlave - que espera pela mensagem de ataque. À parte destas diferenças, na íntegra pensam e agem de igual forma.

```
private Map<String, RobotData> bots = new HashMap();
private List<String> not_hit = new ArrayList<>();
private ACTION action = ACTION.THINKING;

private String attacking = "";
private String buddy = "";
private byte radarDirection = 1;
private boolean rogue = false;
private boolean tmp_leader = false;
```

Listing 1: Variáveis internas mais importantes do Bully

Estruturalmente, a estratégia de controlo de *flow* do *robot*, foi utilizado uma série de estados que mudam o comportamento deste:

- **Thinking**: neste estado este roda o *scanner* 360º para analisar o mapa, guardando o estado de cada *robot*, e por fim escolher o *robot* a atacar e o *robot* amigo a notificar;
- **Attacking**: aqui, o *robot* move o *scanner* para não perder de vista o inimigo, movendo-se de seguida e atacando caso esteja já numa distância mínima (73 pixeis, sendo a força do disparo maior quanto mais perto estiver).

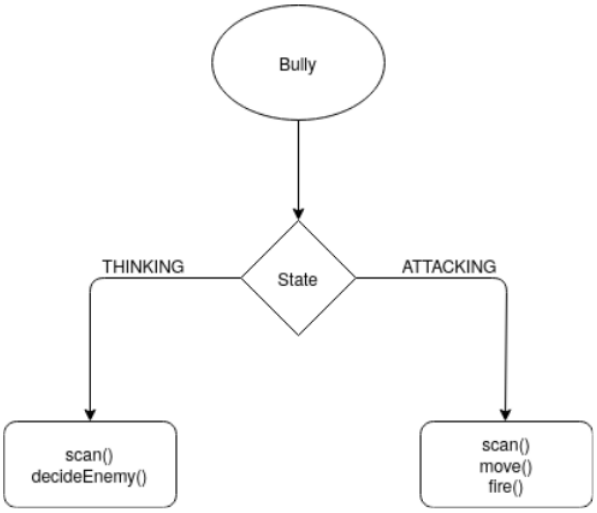

Figura 3: Controlo de *flow* nos diferentes estados

Como este *robot* necessita de um outro amigo para atacar, caso não hajam "Lideres"ou "Slaves"estes entram em modo *rogue*, ou seja, analisam o mapa por alguém vulnerável e iniciam um ataque imediatamente. Este utiliza uma estratégia de scan para seguir o movimento do inimigo que está a atacar chamada de *Oscillating Radar*, executada movendo o radar +/- X graus, de forma a criar um cone de segurança para o localizar. Assim, diminui a possibilidade de o perder de vista e consegue ter uma maior perceção do mundo à volta (não fica restringido apenas a "olhar"para o inimigo alvo). No que toca a movimento, este desloca-se seguindo o inimigo, tendo em conta a posição deste, sem prever colisões/fazer desvios. Caso bata de facto numa parede ou num outro *robot*, este tem um comportamente reativo e, roda 180 graus antes de continuar o deslocamento e anda pra trás 35 pixeis rodando também 45 graus, respetivamente. Caso bata no *robot* que está a atacar, não se desloca, visto beneficiar de estar o mais próximo possível deste. Por fim, de forma a evitar ser acertado por fogo amigo, estes, antes de iniciar um ataque, notifica todos os outros *robots* para não atacarem aquele inimigo.

Um pormenor importante é, no caso da equipa ser composta por um *Hulk* e *JBourne*, na eventualidade do *JBourne* morrer, a resposabilidade de controlar o *Hulk* recai sobre um *BullyLeader* e, caso este já não se encontre vivo, recairá sobre um *BullySlave*.

4.1.2 JBourne

A estratégia deste *robot* baseia-se em dois fatores: manter uma distância mínima entre o robot que está a atacar no momento e só atacar quando a probabilidade de o inimigo estar numa posição no momento seguinte for bastante elevada.

Numa primeira fase o *robot* dá scan a todo o mapa à procura de inimigos. Assim que deteta outro *robot* armazena e calcula todos os seus atributos possíveis: ângulo da arma, posição no mapa, energia, velocidade, ângulo de deslocamento e o tempo atual.

De seguida verifica se o *robot* é amigo ou inimigo e, se inimigo, envia uma mensagem ao *Hulk* a notificá-lo acerca da informação do inimigo.

Após a deteção é calculada, tendo em conta a velocidade e a direção do movimento anterior, uma possível posição futura do *robot* inimigo que foi detetado nessa iteração. É também calculada a potência do disparo variando linearmente com a distância ao *robot* que queremos atacar. Apenas é disparado um tiro se a potência do disparo for acima de 2.3 unidades para que o dano causado seja maximizado, minimizando assim a energia desperdiçada em disparos de longas distâncias.

O movimento deste *robot* tem em conta a posição do inimigo que está a tentar atacar no momento. Este mantém-se sempre a uma distância mínima de 200 unidades e movimenta-se tendo em conta um ângulo de *offset* de 45 graus. A distância percorrida em cada iteração também é dependente da distância ao inimigo, quanto maior, maior o deslocamento numa só iteração.

Este *robot* encontra-se equipado com outros mecanismos, sendo esses de reatividade, no caso de bater numa parede ou num *robot*. No caso de bater num *robot*, se este for um amigo vira 45 graus e distancia-se algumas unidades dele e caso este seja um inimigo antes de se afastar dispara um tiro para o mesmo. No caso de bater numa parede apenas gira para que no próximo movimento não se desloque em direção da parede.

Um facto importante de referir é que este *robot* recebe mensagens dos outros *robots* para que não ataquem o mesmo inimigo de modo a minimizar o dano aos membros da mesma equipa.

Na lista seguinte podemos verificar a sequência de ações, de forma resumida, que este robot executa durante o seu processo de ataque:

- Roda o radar em busca de inimigos;
- Encontra inimigo;
- Atualiza variaveies do inimigo a atacar;
- Envia mensagem para *Hulk*;
- Calcula possíveis coordenadas futuras desse inimigo;
- Averigua se compensa atacar inimigo;
- Dispara se condição anterior for verdadeira;
- Movimenta-se;

É também necessário referir que todas estas ações são não bloqueantes fazendo com que, o movimento e disparo, possam ser executados no mesmo instante de tempo.

4.1.3 Hulk

Este robot possui uma estratégia de movimentação aleatória e dispersa de modo a diminuir a probabilidade de ser atinguido e como tal um maior tempo de vida.

A estratégia adoptada para a movimentação passou por definir um limite do terreno no qual pretendimos que o *robot* executasse o movimento aleatóriamente, sendo que este tem por base a geração de um número aleatório que irá definir a direção do desvio de 180 graus. De forma semelhante foi aplicado aquando do embate com outro *robot*, que difere na medida em que neste o desvio é mais pequeno, sendo apenas de 45 graus, e ainda é necessário adaptar a estratégia tendo em conta as posições relativas dos mesmos. Neste caso, optamos ainda por efetuar um pequeno ataque caso o outro *robot* seja um inimigo. Contudo, ainda era necessário solucionar o problema do embate contra a parede pelo que optamos apenas por fazer um pequeno desvio da mesma uma vez que sempre que este se desvia do limite definido a estratégia passa por se dirigir para uma zona mais central. Desta forma, é assegurado que o mesmo não se encontra muito afastado dos inimigos encontrando-se assim mais próximo dos mesmos sendo mais eficaz no ataque. Torna-se, assim, percetível que o *Hulk* possui comportamentos reativos.

Toda esta estratégia de movimentação deve-se ao facto de termos planeado este *robot* para atacar os inimigos a pedido do *JBourne*. Assim, este encontra-se preparado para receber mensagens do mesmo com as posições que deve atacar, ou seja, para onde deve atirar a próxima bala. Assim, apenas precisa verificar se não irá atinguir um colega de equipa que se encontre na zona de ataque.

Tendo por base toda a estratégia desenvolvida, este *robot* consegue tirar um grande proveito do facto de ser um *Droid* visto que não necessita de um scanner e assim possui vida extra.

4.2 Estratégia

Quanto à estratégia, numa fase inicial foi pensada a utilização do tipo de *robot Bully* para uma abordagem agressiva nos adversários mantedo o *JBourne* um pouco mais distante dos combates de proximidade apenas enviando informações para o *Hulk*. Deparamos-nos com algumas dificulades de implementação e o *Jbourne* tornou-se num *robot* mais agressivo mas que enviava na mesma as informações necessárias ao *Hulk*. Este facto fez com que por vezes existisse *friendly fire* que foi resolvido com o *Bully* a enviar mensagens para o *JBourne* de modo a não atacar o mesmo inimigo, melhorando substancialmente os resultados de combate.

Após alguns testes de combate foi inferido que *Hulk* estava a criar um *bottleneck* na estratégia da equipa e, quando substituido por mais um *robot* do tipo *JBourne* a *performance* geral melhorava substancialmente.

Noutro contexto foi detetado ainda que para certas situações a presença de todos os *robots* do tipo *JBourne* eram uma soluçao bastante viável para o combate em equipas devido à não perda de tempo com mensagens entre colegas de equipa aliada a uma estratégia bastante agressiva para com a equipa inimiga.

4.3 Comunicação

Baseando-nos na unidade curricular do semestre passado, Sistemas Inteligentes, onde aprendemos a utilizar JADE como plataforma de criação de agentes e gestão da comunicação destes mesmos, criámos um sistema de mensagens semelhante ao aprendido na mesma. Desenvolvemos por isso diferentes classes que contém a informação necessária para cada mensagem, sendo que em cada tipo de *robot*, modificámos o método *onMessageReceived* de forma a apenas tratar as mensagens que cada um deveria receber/tratar. Isto permitiu uma maior facilidade na gestão, tratamento e adição/remoção de mensagens. No total, foram desenvolvidos 5 tipos de mensagens:

- *AttackEnemy*: responsável por pedir a um *BullySlave* para iniciar o ataque a um inimigo, sendo utilizada também como resposta para o *BullyLeader*, positiva ou negativa;
- *AttackingEnemy*: contém informação do *robot* a ser atacado por um par de *Bully's*;
- *DeathOfRobot*: notifica todos os amigos que robot X morreu. Esta mensagem apenas existiu no fim da implementação da solução, quando reparamos apenas haver um *robot* na equipa a receber o evento *RobotDeathEvent* estragando o estado interno dos *robots*. Como solução rápida implementou-se esta mensagem;
- *EnemyStatus*: mensagem enviada por quem for responsável pelo *Hulk* para o informar do estado de um robot no mapa. Esta é enviada sempre que o evento *ScannedRobotEvent* é ativado;
- *TakeAction*: também enviada pelo responsável do *Hulk*, contém a ordem a tomar naquele instante.

5 Conclusão

A primeira fase de construção do odómetro e circum-navegação foi conseguida com sucesso. Apenas existiram algumas diculdades na circum-navegação devido ao espaço que os robots ocupam para além da sua posição. A segunda fase, que consistiu na construção de *robots* para combate foi um pouco mais complexa. Isto deveu-se ao facto de ser necessário criar uma estratégia antes de iniciar a programação dos tanques. Após o começo verificou-se uma dificuldade de grau elevado na implemetação da mesma e isso levou a que fossem feitos alguns ajustes de forma a que a estratégia geral não fosse comprometida. No entanto, ficou vincado que nesta implementação a comunicação entre *robots* não era a estratégia mais eficiente devido à perda de tempo de jogo na comunicação e processamento nas mensagens permitindo ao adversário fazer danos nesse intervalo de tempo.

Outra das dificuldades encontradas foi a integração na estratégia de jogo o *Droid Hulk* devido à limitação de não possuir um radar mas em contrapartida conter mais energia.

Quanto ao combate após alguns testes foi notada uma agressividade, muitas vezes em demasia por parte dos *robots Bully* e *JBourne*. Isso fazia com que fosse bastante eficaz numa primeira parte do combate mas comprometia o final se a equipa adversária protege-se o seu lider (*robot* que inicia o jogo com mais energia). O que acontecia muitas vezes e teve que ser corrigido foi a ocorrência de *friendly fire*.

Concluimos desta forma que no contexto da Unidade Curricular este trabalho contribuiu para uma melhor compreensão dos vários tipos de Arquiteturas existentes para serem aplicados no caso de sistemas autónomos, sendo que na nossa implementação foi utilizada uma arquitetura híbrida e, no que toca a estruturas de controlo, foi utilizada uma estratégia *feedfoward*.