



UNIVERSIDADE FEDERAL DA BAHIA
DEPARTAMENTO DE CIÊNCIA DA COMPUTAÇÃO
PROGRAMA DE GRADUAÇÃO

ERIK VINICIUS GOMES ALMEIDA

**DESENVOLVENDO UM ROBÔ
COMPETITIVO PARA ROBOCODE**

Salvador - BA, Brasil

28 de Fevereiro de 2017

Erik Vinicius Gomes Almeida

Desenvolvendo um robô competitivo para Robocode

Monografia apresentada para obtenção do
Grau de Bacharel em Ciência da Computação
pela Universidade Federal da Bahia.

Universidade Federal da Bahia
Departamento de Ciência da Computação
Programa de Graduação

Orientador: Maurício Pamplona Segundo

Salvador - BA, Brasil
28 de Fevereiro de 2017

Erik Vinicius Gomes Almeida

Desenvolvendo um robô competitivo para Robocode/ Erik Vinicius Gomes Almeida.
– Salvador - BA, Brasil, 28 de Fevereiro de 2017-
63 p. : il. (algumas color.) ; 30 cm.

Orientador: Maurício Pamplona Segundo

Monografia – Universidade Federal da Bahia
Departamento de Ciência da Computação
Programa de Graduação, 28 de Fevereiro de 2017.

1. robocode. 2. inteligência artificial. 3. competição nacional. I. Maurício Pamplona Segundo. II. Universidade Federal da Bahia. III. Departamento de Ciência da Computação.

Erik Vinicius Gomes Almeida

Desenvolvendo um robô competitivo para Robocode

Monografia apresentada para obtenção do
Grau de Bacharel em Ciência da Computação
pela Universidade Federal da Bahia.

Trabalho aprovado. Salvador - BA, Brasil, 30 de abril de 2018:

Maurício Pamplona Segundo
Orientador

Professor
Convidado 1

Professor
Convidado 2

Salvador - BA, Brasil
28 de Fevereiro de 2017

Agradecimentos

Agradeço primeiramente e acima de tudo à santíssima trindade: minhas duas mães, Janice Gomes Cardoso e Marilise Barreto Moreira, e minha tia Jaildes Gomes Cardoso Lembrete. São provavelmente as pessoas mais influentes e importantes da minha vida. Minha mãe com um dos melhores exemplos de deboísmo e *good vibes* que eu conheço; minha madrinha na correria com o pé no chão; e a minha tia com a influência tecnológica geek que é hoje meu ganha-pão.

Em seguida aos familiares mais próximos que amo: minha irmã Carolina Gomes Souza e meu irmão e pai emprestados, Gutemberg Junior Silva Souza e Gutemberg Pereira Souza; e aos demais familiares que obviamente também tiveram participação importante na minha vida.

Também à UFBA e suas proporcionadas vicissitudes que por um lado castigaram a alma que estou prestes a recuperar, por outro me trouxe boas experiências e grandes amigos. As menções honrosas são de Mariana Uaquim e Rodrigo Correia, os amigos mais próximos que fiz aqui. Do período *early* UFBA posso citar Ari, Douglas, Gabriel, Genésio, e Ranieri. Do período *late* UFBA, Fernando e Matheus. As boas experiências atribuo aos, segundo minha memória e opinião pessoal e intrasferível, melhores professores que tive na Universidade: Aline, Eduardo, Fabíola, Flávio, Ricardo e Maurício.

Por fim, há naturalmente muito mais a agradecer deste período demasiadamente longo que passei aqui, mas eu invoco a carta de não ter espaço para agradecer a todos que merecem e deixo esta mensagem: para você que não foi mencionado aqui, mas participou positivamente da minha vida e por algum motivo está lendo isso (sério, por quê?) fica um vale-abraço. Pode cobrar.

*The woods are lovely, dark and deep,
But I have promises to keep,
And miles to go before I sleep,
And miles to go before I sleep.*
(FROST, Robert)

Resumo

Várias universidades têm tido iniciativas com torneios de Robocode para auxiliar no ensino de programação, uma vez que o engajamento proporcionado esses torneios subsidia um aprendizado efetivo.

Este trabalho tem por objetivo estudar as regras e funcionamento do Robocode de modo a desenvolver um robô de simples implementação, mas que seja competitivo e efetivo em campeonatos.

A edição de 2016 da Liga Brasileira de Robocode, realizada pelo Laboratório de Informática, Aprendizado e Gestão da Faculdade de Tecnologia da Universidade Estadual de Campinas, foi utilizada para avaliar a efetividade das estratégias propostas neste trabalho. O resultado obtido foi o primeiro lugar na liga.

Palavras-chave: inteligência artificial; robocode; wave surfing; linear targeting; guess factor targeting; my first robot; liga brasileira de robocode 2016.

Abstract

Many universities have been using Robocode tournaments to help teaching programming, since the engagement provided by those tournaments supports an effective learning.

The present work's goal is to study how Robocode works and its rules to develop a simple, but competitive robot, that is also effective in championships.

The 2016 edition of the Brazilian Robocode League, made by the Laboratory of Informatic, Learning and Management of the Faculty of Technology of the University of Campinas was used to evaluate the effectiveness of the strategies proposed in this work. The result obtained was the first place in the league.

Keywords: artificial intelligence; robocode; wave surfing; linear targeting; guess factor targeting; my first robot; liga brasileira de robocode 2016.

Lista de ilustrações

Figura 1 – Exemplo de batalha.	15
Figura 2 – Exemplo da tabela de pontuação em uma batalha de 10 rodadas. . . .	16
Figura 3 – Anatomia do robô.	17
Figura 4 – Visualização do radar.	19
Figura 5 – Algumas informações disponíveis no momento em que um inimigo é escaneado.	27
Figura 6 – <code>BasicRobot</code> com mira linear.	29
Figura 7 – Saldo de energia por turno por força de tiro	30
Figura 8 – Posicionamento lateral.	32
Figura 9 – Máximo ângulo de escape dividido em <i>guess factors</i>	33
Figura 10 – Disparos abstraídos em ondas com zonas de perigo.	39
Figura 11 – <code>MyFirstRobot</code> em 1v1 contra os robôs básicos	47
Figura 12 – Heurística de aproximação contra o <code>SpinBot</code>	48
Figura 13 – Diferentes períodos para troca de sentido contra o <code>SpinBot</code>	49
Figura 14 – <code>MyFirstRobot</code> em <i>melee</i> contra os robôs de exemplo	50
Figura 15 – BIRL em <i>melee</i> contra robôs de exemplo	51
Figura 16 – <i>Risk evaluation</i>	55
Figura 17 – Campeões da Liga Nacional de Robocode 2016.	60

Lista de tabelas

Tabela 1 – Pontuação das classificatórias por colocação	46
Tabela 2 – Chave das semifinais	47
Tabela 3 – Primeira rodada do torneio local	51
Tabela 4 – Segunda rodada do torneio local	52
Tabela 5 – Terceira rodada do torneio local	52
Tabela 6 – Quarta rodada do torneio local	52
Tabela 7 – Quinta rodada do torneio local	53
Tabela 8 – Chave das semifinais do torneio local	54
Tabela 9 – Resultado da segunda batalha das semifinais locais	54
Tabela 10 – Resultado da final regional	54
Tabela 11 – Campeões dos torneios locais.	56
Tabela 12 – Primeira rodada da liga nacional.	57
Tabela 13 – Segunda rodada da liga nacional.	58
Tabela 14 – Chave das semifinais da liga nacional.	59
Tabela 15 – Resultado BIRL contra Teleton Rejecteds.	59
Tabela 16 – Resultado da primeira batalha das semifinais nacionais.	59
Tabela 17 – Resultado da segunda batalha das semifinais nacionais	60
Tabela 18 – Final da liga nacional	60

Lista de códigos

2.1	O MyFirstRobot do Robocode, sem comentários e algumas quebras de linha.	20
2.2	MyFirstRobot adaptado para a classe AdvancedRobot	21
2.3	Estratégia básica - Escaneamento básico	24
2.4	Estratégia básica - Escaneamento preso	24
2.5	Estratégia básica - Escaneamento seguro	25
2.6	Estratégia básica - Mira simples	27
2.7	Estratégia básica - Mira linear	28
2.8	Estratégia básica - Heurísticas de tiro	30
2.9	Estratégia básica - Movimentação lateral	32
2.10	Estratégia avançada - Guess factor targeting - Selecionando o fator	34
2.11	Estratégia avançada - Guess factor targeting - Atualizando os fatores	35
2.12	Estratégia avançada - Wave surfing - Detectando o disparo	38
2.13	Estratégia avançada - Wave surfing - Armazenando as ondas	39
2.14	Estratégia avançada - Wave surfing - Atualizando as áreas de perigo	41
2.15	Estratégia avançada - Wave surfing - Surfando as ondas	43
3.1	Heurística de aproximação do alvo	48
3.2	Redução do período dos ciclos de movimentação	49

Sumário

	Lista de códigos	10
1	INTRODUÇÃO	13
2	ROBOCODE	15
2.1	Regras e funcionamento	15
2.1.1	Sistema de pontuação	16
2.1.2	A anatomia do robô	17
2.1.2.1	O corpo	17
2.1.2.2	O canhão	18
2.1.2.3	O radar	18
2.1.3	Os eventos	18
2.2	Robôs básicos	20
2.2.1	MyFirstRobot	20
2.2.1.1	MyFirstRobot avançado	21
2.2.2	Velocirobot	22
2.2.3	Crazy	22
2.2.4	SpinBot	23
2.2.5	Walls	23
2.2.6	Tracker	23
2.3	Estratégias básicas	23
2.3.1	Radar	23
2.3.1.1	Escaneamento básico	24
2.3.1.2	Escaneamento preso	24
2.3.1.3	Escaneamento seguro	25
2.3.2	Canhão	26
2.3.2.1	Mira simples	26
2.3.2.2	Mira linear	28
2.3.2.3	A força do tiro	29
2.3.3	Corpo	31
2.3.3.1	Movimentação lateral	31
2.4	Estratégias avançadas	32
2.4.1	<i>Guess factor targeting</i>	33
2.4.2	<i>Wave surfing</i>	38
2.5	Robôs avançados	45
2.5.1	DrussGT	45

2.5.2	Diamond	45
2.5.3	Neuromancer	45
3	ROBOCODE BRASIL	46
3.1	Regras	46
3.2	Etapa regional	47
3.2.1	Primeira rodada classificatória	51
3.2.2	Segunda rodada classificatória	51
3.2.3	Terceira rodada classificatória	52
3.2.4	Quarta rodada classificatória	52
3.2.5	Quinta rodada classificatória	53
3.2.6	Semifinais	53
3.2.7	Final	54
3.3	Etapa nacional	54
3.3.1	Primeira rodada	56
3.3.1.1	Equipe Rocket	56
3.3.1.2	Dragonborn	56
3.3.1.3	Skynet	56
3.3.1.4	RecrutaZero	56
3.3.1.5	Moto	57
3.3.1.6	Teleton Rejecteds	57
3.3.1.7	OutOfTheCage	57
3.3.2	Segunda rodada	58
3.3.2.1	Skynet	58
3.3.2.2	Dragonborn	58
3.3.2.3	RecrutaZero	58
3.3.2.4	Moto	58
3.3.3	Semifinais	59
3.3.4	Final	60
4	CONCLUSÃO	61
4.1	Desafios futuros	61
	REFERÊNCIAS	62

1 Introdução

Atualmente, instituições de ensino desenvolvem ações com jogos digitais nos processos de ensino-aprendizagem para incentivar o aprendizado de programação. Estudos demonstram que os alunos podem alcançar ganhos significativos de aprendizagem ao interagir com jogos educacionais em ambientes controlados (MEIRA, 2016).

Dos jogos clássicos *Asteroids* e *Minesweeper* (BECKER, 2001) aos mais recentes como *Angry Birds* (YOON; KIM, 2015) foram utilizados para demonstrar os jogos como ferramenta de aprendizagem, incluindo como introdutor, com alunos com pouca ou nenhuma experiência em programação (LEUTENEGGER; EDGINGTON, 2007).

O Robocode é um jogo de simulação programável no qual tanques de guerra virtuais batalham pela supremacia em uma arena, cujo propósito é de ensinar técnicas de programação Java, como a utilidade de herança e programação orientada a objetos em geral, além de prover uma introdução a programação orientada a eventos (HARTNESS, 2004). Quando usado em sala de aula, por trazer a percepção de uma aplicação no mundo real e aliado ao contexto social inerente à competitividade das batalhas, motiva os alunos a engajar nos desafios do aprendizado (LIU, 2008).

A competitividade de torneios Robocode é altamente engajante. Estudantes são energeticamente envolvidos e investidos no sucesso de seus robôs, o que alimenta diretamente o tipo de atenção, satisfação e comprometimento ao projeto que subsidia um aprendizado efetivo (GEORGAS, 2016). Pode compreender diversas técnicas de programação, como Algoritmo Genético (SHICHEL; ZISERMAN; SIPPER, 2005; HARPER, 2014), Aprendizado por Reforço (HARTNESS, 2004), Redes Neurais e Sistemas Classificadores (NIDORF; BARONE; FRENCH, 2010). Múltiplas técnicas podem ser aplicadas na confecção de um mesmo robô (LAI et al., 2006).

Várias universidades brasileiras têm tido iniciativas com torneios de Robocode dentro das disciplinas de programação ou em semanas de ciência e tecnologia, a exemplo do LIAG¹ da FT-UNICAMP², que promove torneios e ligas interinstitucionais de Robocode desde 2010 (MEIRA, 2016).

O objetivo deste trabalho é desenvolver um robô de simples implementação que seja efetivo em competições. Para tanto:

- Entender as regras e funcionamento do Robocode;
- Identificar as técnicas e estratégias existentes;

¹ Laboratório de Informática, Aprendizado e Gestão

² Faculdade de Tecnologia da Universidade Estadual de Campinas

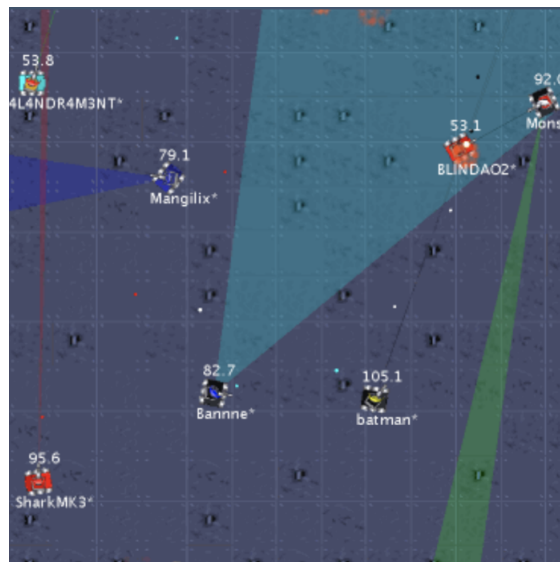
- Avaliar estratégias na construção de um robô;
- Elaborar um robô para competir em campeonato;
- Avaliar o robô durante a trajetória da edição de 2016 da Liga Brasileira.

Além do Capítulo 1, da introdução, este trabalho se divide em outros três capítulos. A apresentação da fundamentação teórica se encontra no Capítulo 2, onde se disserta sobre o que é o robocode, suas regras, robôs e estratégias básicas e robôs e estratégias avançadas. No Capítulo 3 será apresentado o caminho percorrido no campeonato Robocode Brasil realizado em 2016 bem com os resultados obtidos por seus robôs competidores. No Capítulo 4 é apresentada a conclusão do trabalho e são evidenciados os desafios futuros para evolução do robô.

2 Robocode

Criado por Mathew Nelson, o Robocode é um jogo de simulação de batalhas, no qual tanques de guerra virtuais disputam a obtenção de pontos. O jogador é o programador que desenvolve a inteligência artificial do robô controlando-o em uma arena, como ilustrado na Figura 1. Apesar das batalhas aparentarem ser em tempo real, o jogo aplica uma etapa bloqueante para processamento. Assim, é razoável encarar o comportamento dos robôs em termos de estados discretos com um número finito de estados sucessores; em outras palavras, um jogo baseado em turnos (HARTNESS, 2004).

Figura 1 – Exemplo de batalha.



2.1 Regras e funcionamento

As batalhas são divididas em rodadas e os tanques recebem pontos de acordo com um sistema de pontuação. Vence a batalha aquele que acumular mais pontos.

Os robôs são controlados por programas em Java ou .NET escritos pelos jogadores. Os programas têm à sua disposição a chamada inicial de execução do robô e outros eventos que são disparados quando suas condições forem atendidas; alguns exemplos são: seu robô é atingido por uma bala, seu tiro acerta um outro robô, algum robô da arena morre.

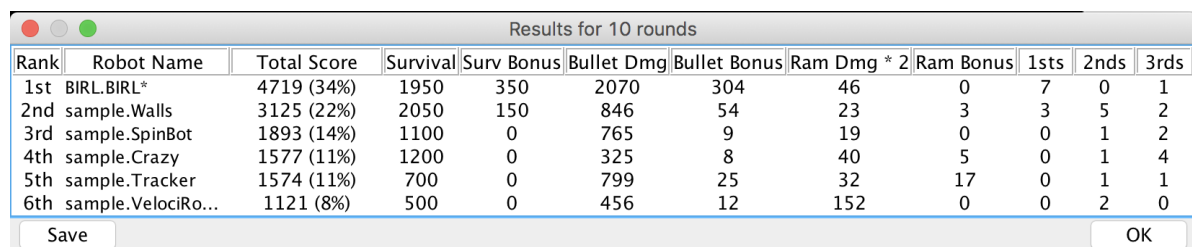
No começo de cada rodada os robôs são inicializados com uma certa quantidade de energia, que é perdida atirando balas, colidindo com algo ou sendo atingido por uma bala. Além disso, caso a batalha se prolongue por um certo período, todos os robôs passam a

perder energia continuamente até que a rodada termine. Quando um robô chega a 0 de energia ele morre e é eliminado da arena até começar a rodada seguinte, a não ser que ele tenha chegado a esse nível de energia atirando uma bala. Nesse caso, o robô é desativado, mas permanece na arena, imóvel. A única forma de recuperar energia é acertando um tiro em outro robô. A rodada termina quando restar no máximo um robô na arena.

2.1.1 Sistema de pontuação

Ao longo das batalhas, os robôs são recompensados com pontos de acordo com sistema de pontuação. Ao final de cada partida, as pontuações são compiladas e exibidas em uma tabela (ROBOWIKI, 2017h), a exemplo da Figura 2:

Figura 2 – Exemplo da tabela de pontuação em uma batalha de 10 rodadas.



Rank	Robot Name	Total Score	Survival	Surv Bonus	Bullet Dmg	Bullet Bonus	Ram Dmg * 2	Ram Bonus	1sts	2nds	3rds
1st	BIRL.BIRL*	4719 (34%)	1950	350	2070	304	46	0	7	0	1
2nd	sample.Walls	3125 (22%)	2050	150	846	54	23	3	3	5	2
3rd	sample.SpinBot	1893 (14%)	1100	0	765	9	19	0	0	1	2
4th	sample.Crazy	1577 (11%)	1200	0	325	8	40	5	0	1	4
5th	sample.Tracker	1574 (11%)	700	0	799	25	32	17	0	1	1
6th	sample.VelociRo...	1121 (8%)	500	0	456	12	152	0	0	2	0

As colunas da Figura 2 representam, respectivamente:

- **Colocação (Rank):** A ordem na qual os robôs se classificaram;
- **Nome do robô (Robot Name):** A identificação do robô;
- **Pontuação total (Total Score):** A soma dos pontos obtidos pelo robô na partida, seguido de seu percentual em relação a todos os pontos gerados na partida;
- **Sobrevivência (Survival):** Cada vez que um robô na arena é destruído, todos os outros ainda vivos recebem 50 pontos;
- **Bônus de sobrevivência (Surv Bonus):** O último robô vivo na arena recebe 10 pontos por cada robô destruído antes dele;
- **Dano de tiro (Bullet Dmg):** Cada unidade de dano causado pelo robô concede a ele 1 ponto;
- **Bônus de tiro (Bullet Bonus):** Quando um robô destrói um inimigo com um tiro, ele recebe um bônus de 20% em cima do dano infligido por ele no inimigo destruído;
- **Dano de colisão (Ram Dmg * 2):** Os robôs recebem 2 pontos para cada unidade de dano causado a outro por colisão;

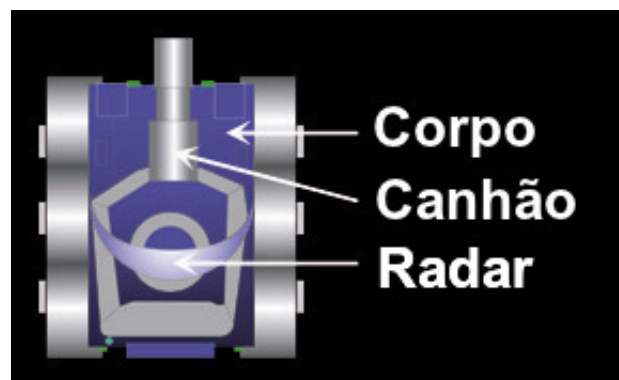
- **Bônus de colisão (Ram Bonus):** Caso um robô destrua outro por colisão, ele recebe um bônus de 30% em cima do dano infligido por ele ao destruído;
- **Em primeiro, em segundo, em terceiro (1sts, 2nds, 3rds):** Apenas informativo, relacionado ao tempo de sobrevivência de cada robô. No exemplo acima, o BIRL sobreviveu a todos os outros robôs em 7 batalhas, em nenhuma delas foi o penúltimo e foi o antepenúltimo sobrevivente em 1 batalha.

É relevante ponderar os custos da obtenção de cada ponto e daí extrair como cada um deve ser explorado. Por exemplo: os robôs são recompensados com pontos quando colidem entre si, mas ambos perdem energia. Caso apenas ambos estejam vivos na arena, explorar essa pontuação pode ser vantajoso para o que possui mais energia; em outras configurações, essa abordagem é perigosa.

2.1.2 A anatomia do robô

O robô é composto de 3 partes: radar, canhão e corpo ([ROBOWIKI, 2017g](#)), como ilustrado na Figura 3. Inicialmente cada parte é acoplada à imediatamente inferior, de modo que, caso o robô gire para a direita, ambos o canhão e o radar acompanharão. Se o corpo for desacoplado do canhão, seu movimento se torna independente dos outros, entretanto mover o canhão ainda ocasiona o giro do radar. Finalmente há como tornar as partes completamente independentes uma da outra.

Figura 3 – Anatomia do robô.



2.1.2.1 O corpo

O corpo é o atuador do robô responsável pela locomoção. É possível fazê-lo mover-se para frente ou para trás e girar no sentido horário ou anti-horário. O movimento do robô é acelerado, $1\text{pixel}/\text{turno}^2$, até atingir a máxima velocidade de $8\text{pixels}/\text{turno}$. A velocidade máxima do giro do robô é de $10^\circ/\text{turno}$.

Alguns métodos podem ser chamados para se obter informações sobre o próprio robô. Entre outros dados, é possível saber (CRESPO, 2017):

- Suas dimensões: 36 *pixels* de altura e de largura;
- Suas coordenadas na arena, sendo o ponto (0,0) o canto inferior esquerdo;
- Seu *heading*, valor entre 0 e 360¹ referente ao ângulo para o qual sua parte dianteira está direcionada, sendo 0 o norte e crescendo no sentido horário;
- Sua velocidade, positiva quando estiver se movendo no mesmo sentido do *heading*, negativa caso contrário;

2.1.2.2 O canhão

Outro atuador do robô, cujas ações permitidas são girar e atirar. A orientação do giro pode ser no sentido horário ou anti-horário, com velocidade de 20°/*turno*. Para disparar um projétil de força $f \in [0.1, 3.0]$, o robô gasta o mesmo valor f de energia. Caso algum inimigo seja atingido pelo projétil, ele receberá $f \times 4$ de dano, além de um adicional de $2(f - 1)$ caso $f > 1$, e o robô que atirou recuperará $f \times 3$ de energia.

Um disparo só pode ser realizado quando o canhão estiver "frio". No início de cada rodada os canhões começam com 3 unidades de calor. Cada tiro gera $1 + (f/5)$ de calor, dissipado em 0.1 por turno. O canhão é considerado "frio" quando possui 0 unidades de calor (ROBOWIKI, 2017f).

2.1.2.3 O radar

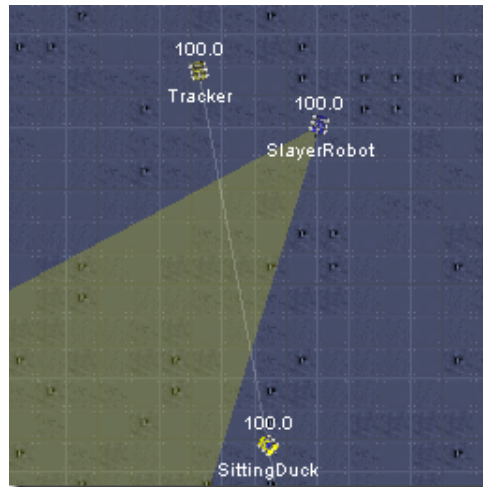
É o sensor capaz de identificar a presença dos inimigos na arena. É representado por uma linha partindo do sensor até uma distância máxima de 1200 *pixels*. É possível girar o radar nos sentidos horário e anti-horário com velocidade de 45°/*turno* (ROBOWIKI, 2017f). Em um turno que o radar tenha girado, ele terá formato representado por um cone, como pode ser observado na Figura 4, na qual o radar do SlayerRobot está cobrindo toda a área sombreada; já o radar do Tracker, que foi mantido parado, apresentou formato de linha.

2.1.3 Os eventos

Quando certa condição é atendida, um evento do tipo *CondiçãoEvent* será instanciado. Caso o jogador tenha implementado em seu robô o método *onCondiçãoEvent*, ele será chamado pelo simulador, recebendo por parâmetro a instância de *CondiçãoEvent* gerada. Por exemplo: no turno da Figura 4, caso o método *onScannedRobot* tenha sido

¹ Todos os métodos envolvendo ângulos possuem uma chamada alternativa que usa radianos.

Figura 4 – Visualização do radar.



implementado no robô Tracker, ele será chamado recebendo o evento `ScannedRobotEvent` por parâmetro, do qual pode se extrair informações sobre o SittingDuck.

Alguns dos eventos são (ZAMBIASI, 2010):

- **ScannedRobotEvent:** Argumentavelmente o evento mais importante do jogo. É através dele que o robô obterá informações sobre seus inimigos, dentre as quais destacam-se:
 - Seu nome;
 - Sua velocidade;
 - A quantidade de energia que possui;
 - A distância entre seu centro e a do robô que o escaneou;
 - Seu *bearing*, um ângulo $\alpha \in [-180^\circ, 180^\circ)$ formado entre sua posição e o *heading* do robô que o escaneou. Cresce positivamente no sentido horário;
- **BulletMissedEvent:** É gerado quando um tiro dado pelo robô atinge a parede;
- **BulletHitEvent:** É gerado quando um tiro dado pelo robô atinge um inimigo;
- **HitByBulletEvent:** É gerado quando o robô é atingido por um tiro;
- **HitRobotEvent:** É gerado quando o robô colide contra um inimigo;
- **HitWallEvent:** É gerado quando o robô colide contra a parede;
- **CustomEvent:** O jogador pode também criar suas próprias condições, como por exemplo o próprio nível de energia atingir um valor muito baixo, ou o canhão

terminou a rotação até o ângulo desejado, e adicioná-las em uma fila que será verificada a cada turno. Caso alguma condição elaborada seja atendida, o método *onCustomEvent* será chamado e a condição causadora será passada com o evento.

- **PaintEvent:** É gerado no momento em que o robô será impresso na arena. O jogador pode utilizá-lo para testar o funcionamento do robô imprimindo na tela, por exemplo, saídas reais e esperadas dos métodos implementados.

2.2 Robôs básicos

O Robocode fornece uma implementação bem simples de robô como ponto de partida do desenvolvimento, o `MyFirstRobot`. Outros robôs são disponibilizados em pacotes de exemplo e podem servir de inspiração na elaboração da estratégia do jogador.

2.2.1 MyFirstRobot

Ao longo deste capítulo, o `MyFirstRobot` será utilizado como base para ilustrar as estratégias básicas. Partindo do template apresentado na Listagem 2.1, cada iteração apresentará apenas as mudanças relevantes, de modo que cada resultado será funcional.

Listagem 2.1 – O `MyFirstRobot` do Robocode, sem comentários e algumas quebras de linha.

```
package mrcid;
import robocode.*;

public class MyFirstRobot extends Robot {
    public void run() {
        while (true) {
            ahead(100);
            turnGunRight(360);
            back(100);
            turnGunRight(360);
        }
    }
    public void onScannedRobot(ScannedRobotEvent e) {
        fire(1);
    }
    public void onHitByBullet(HitByBulletEvent e) {
        back(10);
    }
    public void onHitWall(HitWallEvent e) {
```

```
        back (20);  
    }  
}
```

A classe inicial estende `Robot` e implementa alguns métodos básicos. O método `run` será chamado no início de cada rodada e prenderá a execução do robô em um loop infinito no qual ele irá avançar 100 pixels, girar o canhão 360° no sentido horário, recuar 100 pixels e repetir o giro do canhão. Uma vez que o radar não foi desacoplado do canhão, ele será rotacionado ao mesmo passo.

A execução do loop será interrompida caso algum evento correspondente aos métodos implementados seja disparado. Caso algum robô inimigo cruze seu radar, um objeto da classe `ScannedRobotEvent` será instanciado com informações dele e será passado via parâmetro da função `onScannedRobot`. Quando isso acontecer, o robô gastará 1 ponto de energia para disparar um projétil de mesma força na direção e sentido para o qual o canhão está apontado. Em seguida, o robô retornará à execução do loop.

Quando o robô for atingido por um projétil, o método `onHitByBullet` será chamado e o fará retroceder 10 pixels. Quando o robô chocar-se contra a parede, o método `onHitWall` chamado o fará retroceder 20 pixels.

2.2.1.1 MyFirstRobot avançado

A classe `Robot` suporta apenas métodos síncronos: bloqueiam a execução do robô até completar sua ação. Assim, enquanto o robô estiver movendo para frente ou para trás, o canhão não irá realizar seu giro. Extendendo a classe `AdvancedRobot`, os métodos passam de ação para `setAção`, que retorna imediatamente e a ação é executada quando o método `execute` ou um método síncrono é chamado. O método `execute` retorna quando as ações são iniciadas. Desse modo é possível modificar o `MyFirstRobot`, do modo apresentado na Listagem 2.2, para que as ações de movimentação do robô e de giro do canhão aconteçam simultaneamente.

Listagem 2.2 – `MyFirstRobot` adaptado para a classe `AdvancedRobot`

```
public class MyFirstRobot extends AdvancedRobot {  
    public void run() {  
        setTurnGunRight (Double.POSITIVE_INFINITY);  
        int counter = 0;  
        while (true) {  
            if (counter < 16)  
                setAhead (100);  
            else  
                setBack (100);  
        }  
    }  
}
```

```
        counter = (counter + 1) % 32;
        execute ();
    }
}
public void onScannedRobot(ScannedRobotEvent e) {
    setFire (1);
}
}
```

Uma vez que o método `setTurnGunRight` não bloqueia a execução do código, pode-se comandar que o canhão gire infinitamente no sentido horário. Para obter movimentação similar à do robô anterior foi adicionado um contador, de modo a executar o `setAhead` em 16 turnos, depois `setBack` em 16 turnos. Desse modo, o robô se moverá 100 pixels em cada sentido.

O argumento passado no `setAhead` e no `setBack` não garante que o robô desenvolverá os 100 pixels. Não fossem os 16 turnos dedicados a cada sentido, controlado pelo contador, o robô moveria 1 pixel pra cima, outro para baixo.

O método `fire` foi substituído por `setFire` para que a movimentação não seja interrompida durante o disparo. Substituir o método `back` por `setBack` não manteria o comportamento anterior, uma vez que a instrução seria sobrescrita logo em seguida, quando a execução do código retornasse ao loop principal. Assim, os métodos `onHitByBullet` e `onHitWall` foram removidos.

2.2.2 Velocirobot

A movimentação deste robô é próxima à do `MyFirstRobot`: move-se 32 turnos para frente, depois 32 turnos para trás. O maior diferencial está no fato da velocidade ser alternada com base no sentido: $4\text{pixels}/\text{turno}$ para frente, $6\text{pixels}/\text{turno}$ para trás. Quando colide com a parede, inverte o sentido do movimento. Caso seja atingido por um projétil, seu corpo é rotacionado no sentido horário.

Seu canhão é mantido rotacionando no sentido horário e é disparado assim que um inimigo cruza o escanamento do radar acoplado a ele.

2.2.3 Crazy

O comportamento base deste robô é de mover-se continuamente para frente enquanto alterna sua rotação. Ao colidir com uma parede ou inimigo, seu sentido é alternado, retornando eventualmente ao comportamento base.

O canhão, como o radar, é acoplado ao corpo, e é disparado assim que algum robô inimigo é escaneado.

2.2.4 SpinBot

A dinâmica deste robô é mover-se para frente enquanto rotaciona no sentido horário com uma velocidade máxima de $5\text{pixels}/\text{turno}$, conseqüentemente fazendo-o girar em um círculo de raio menor. Ao colidir com um inimigo, ele rotaciona 10 graus a mais no próprio eixo e continua seu padrão.

Seu canhão e radar são controlados de forma equivalente ao **Crazy**.

2.2.5 Walls

Este robô move-se ao longo das paredes da arena no sentido horário. Caso colida com algum inimigo no meio do caminho, ele recua 10 pixels, rotaciona 90 graus, movimenta-se até a próxima parede e segue o curso.

O canhão é mantido perpendicular ao corpo do robô, e o radar acoplado a ele. Como os anteriores, dispara assim que um inimigo cruza seu escaneamento.

2.2.6 Tracker

O método deste robô é de rotacionar o radar, acoplado ao canhão, até encontrar um inimigo. Ao fazê-lo, aproxima-se dele e então dispara com força máxima. Caso um inimigo esteja muito próximo, ele recua.

2.3 Estratégias básicas

Para se desenvolver um robô competitivo, algumas estratégias podem ser adotadas. Este capítulo apresentará estratégias básicas classificadas de acordo com a parte da anatomia na qual ela será aplicada.

Algumas estratégias são mais ou menos eficientes a depender da quantidade de robôs na arena. A configuração na qual há mais de um inimigo na arena é chamada *melee*; já a com apenas um inimigo, *1v1*. Estratégias do tipo *1v1* serão abordadas neste capítulo.

2.3.1 Radar

O objetivo das estratégias de controle do radar é de escanear o mais frequentemente possível os robôs presentes na arena. Na implementação anterior, o radar foi mantido acoplado ao canhão, rotacionando $20^\circ/\text{turno}$ quando o mesmo é rotacionado. Uma vez que

a velocidade de giro do radar é de $45^\circ/\text{turno}$ quando desacoplado, é 125% mais eficiente controlá-lo separadamente.

2.3.1.1 Escaneamento básico

Uma forma simples de garantir que todos os robôs presentes na arena serão escaneados é fazê-lo girar indefinidamente, como apresentado na Listagem 2.3. O radar é desacoplado do canhão e do corpo, depois comandado que gire infinitamente no sentido horário (ROBOWIKI, 2017e).

Listagem 2.3 – Estratégia básica - Escaneamento básico

```
public class MyFirstRobot extends AdvancedRobot {
    public void run() {
        setAdjustRadarForGunTurn(true);
        setAdjustRadarForRobotTurn(true);
        setTurnRadarRight(Double.POSITIVE_INFINITY);
        int counter = 0;
        ...
    }
}
```

Por ora, o comando de rotação do canhão e o de atirar quando um inimigo for detectado foram removidos, já que uma vez que a velocidade de rotação do canhão é diferente da do radar, quando um inimigo for detectado, ele provavelmente não estará na direção do canhão.

2.3.1.2 Escaneamento preso

Com a velocidade de 45° , o radar demora 8 turnos para escanear toda a arena. Caso o robô esteja em uma batalha contra apenas um outro robô, o radar terá, em geral, eficiência de 12,5% adotando o escaneamento básico.

Alternar o sentido de rotação do radar toda vez que um robô é escaneado, fará com que o radar fique preso escaneando a mesma seção angular enquanto o inimigo estiver nessa área (ROBOWIKI, 2017e).

Listagem 2.4 – Estratégia básica - Escaneamento preso

```
public class MyFirstRobot extends AdvancedRobot {
    ...
    public void onScannedRobot(ScannedRobotEvent e) {
        setTurnRadarLeftRadians(getRadarTurnRemainingRadians());
    }
}
```

Como apresentado na Listagem 2.4, o primeiro comando dado ao radar foi de rotacionar infinitamente no sentido horário. Quando um robô é escaneado, o radar é comandado a girar no sentido anti-horário o que ainda falta para completar o giro do primeiro comando. A nova instrução será, portanto, girar infinitamente no sentido anti-horário. Caso o inimigo seja novamente interceptado pelo radar, uma vez que o radar está girando infinitamente no sentido anti-horário, o método `getRadarTurnRemainingRadians` retornará `Double.NEGATIVE_INFINITY`. Assim, o comando `setTurnRadarLeftRadians` recebendo `Double.NEGATIVE_INFINITY` fará o radar girar no sentido horário e assim sucessivamente, gerando o efeito de "ir e voltar".

2.3.1.3 Escaneamento seguro

Apesar do ganho significativo de eficiência, não é raro o inimigo sair da área varrida com o escaneamento preso. Para uma eficiência ainda maior, pode-se utilizar uma estratégia que consiste em calcular a distância do radar até o robô inimigo e adicionar uma distância segura para além do robô.

Uma vez que a manutenção da movimentação do radar não é mais feita com valores infinitos, pode ser observado na Listagem 2.5 a adição de uma condição no loop principal, para que o radar rotacione infinitamente sempre que não existir um movimento agendado para o próximo turno.

A distância segura a ser adicionada ao ângulo entre o radar e o inimigo é inversamente proporcional à distância que ele se encontra do robô, ou seja, quanto mais próximo o inimigo estiver do robô, mais aberto o ângulo do radar deve ser mantido. De modo que seja coberto as dimensões de um robô, 36 *pixels*, a mais de cada lado (ROBOWIKI, 2017e).

Listagem 2.5 – Estratégia básica - Escaneamento seguro

```

...
import robocode.util.Utils;

public class MyFirstRobot extends AdvancedRobot {
    public void run() {
        setAdjustRadarForGunTurn(true);
        setAdjustRadarForRobotTurn(true);
        int counter = 0;
        while (true) {
            if (getRadarTurnRemaining() == 0.0)
                setTurnRadarRightRadians(Double.POSITIVE_INFINITY);
            ...
        }
    }
}

```

```
public void onScannedRobot(ScannedRobotEvent e) {
    double angleToEnemy =
        getHeadingRadians() + e.getBearingRadians();
    double turnToEnemy = Utils.normalRelativeAngle(
        angleToEnemy - getRadarHeadingRadians()
    );
    double extraTurn =
        Math.atan(36.0 / e.getDistance())
        * (turnToEnemy >= 0 ? 1 : -1);

    setTurnRadarRightRadians(turnToEnemy + extraTurn);
}
}
```

2.3.2 Canhão

As estratégias de controle do canhão têm por objetivo acertar os inimigos com os tiros. Dado um alvo escolhido, a tarefa deste controlador é apontar o canhão numa direção tal que ao realizar um disparo, o alvo seja atingido.

A estratégia mais simples de controlar o canhão é a já implementada quando se cria o `MyFirstRobot`. O canhão é mantido acoplado ao radar, que estará sempre girando. Assim que um inimigo é detectado, o canhão é disparado.

Entretanto, como já discutido, o acoplamento do canhão com o radar afeta a eficiência deste. Logo, será apresentada aqui a estratégia de mira simples com o canhão desacoplado.

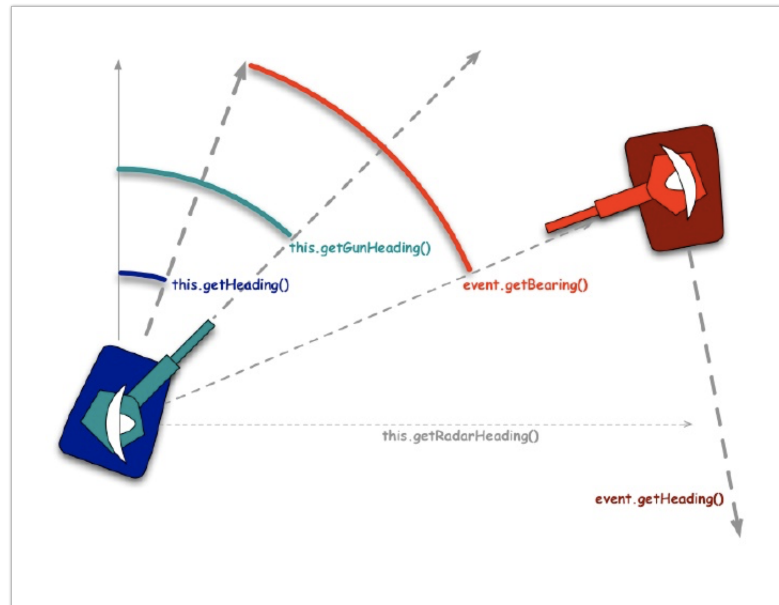
2.3.2.1 Mira simples

Uma vez que o canhão não está mais acoplado ao radar, é preciso apontar o canhão na direção do inimigo antes realizar o disparo. Quando o inimigo é escaneado, é chamada a função de atirar passando os dados capturados sobre ele, como visto na Listagem 2.6. Os ângulos trabalhados nesta estratégia podem ser vistos na Figura 5, onde observa-se da esquerda para a direita, respectivamente:

- `this.getHeading()`: A direção na qual o corpo do robô está apontado;
- `this.getGunHeading()`: A direção na qual o canhão do robô está apontado;
- `this.getRadarHeading()`: A direção na qual o radar do robô está apontado;
- `event.getBearing()`: O ângulo compreendido entre o *heading* do robô e o inimigo;

- `event.getHeading()`: A direção na qual o corpo do robô inimigo está apontado.

Figura 5 – Algumas informações disponíveis no momento em que um inimigo é escaneado.



Primeiro são somados o *bearing* do inimigo e o *heading* do robô. O valor resultante é um ângulo chamado *absolute bearing*, que representa o ângulo formado entre o norte do robô e a posição na qual o inimigo foi encontrado.

Em seguida é calculada a diferença entre o *absolute bearing* e o ângulo para o qual o canhão está apontando atualmente (*gun heading*). Essa diferença é justamente a rotação que fará o canhão apontar para o inimigo, mas antes de aplicá-la ela deve ser normalizada. A normalização faz com que uma rotação no sentido horário de mais de 180° seja feita no sentido anti-horário, aumentando assim sua eficiência (ROBOWIKI, 2013).

Listagem 2.6 – Estratégia básica - Mira simples

```
public class MyFirstRobot extends AdvancedRobot {
    ...
    public void onScannedRobot(ScannedRobotEvent e) {
        shoot(e);
        ...
    }
    public void shoot(ScannedRobotEvent e) {
        double absoluteBearing =
            e.getBearingRadians() + getHeadingRadians();
        double gunTurn = absoluteBearing - getGunHeadingRadians();
        setTurnGunRightRadians(Utils.normalRelativeAngle(gunTurn));
    }
}
```

```

        setFire (1.0);
    }
}

```

Os métodos chamados na Listagem 2.6 aparecem acrescidos do sufixo **Radians** em relação aos apresentados na Figura 5. As informações são equivalentes, nesta em graus; naquela, em radianos.

2.3.2.2 Mira linear

Ao escanear um inimigo o radar fornece, além de seu *bearing*, sua velocidade, direção e sentido. Com esses dados é possível projetar a posição do inimigo em um dado tempo futuro, assumindo que os valores não variem. A estratégia da mira linear consiste em apontar o canhão para essa posição na qual o inimigo foi projetado (ROBOWIKI, 2012a). como pode ser observado na Listagem 2.7

Listagem 2.7 – Estratégia básica - Mira linear

```

public class MyFirstRobot extends AdvancedRobot {
    ...
    public void shoot(ScannedRobotEvent e) {
        ...
        double gunTurn = absoluteBearing - getGunHeadingRadians();
        double future =
            e.getVelocity()
            * Math.sin(e.getHeadingRadians() - absoluteBearing)
            / Rules.getBulletSpeed(1);
        setTurnGunRightRadians(
            Utils.normalRelativeAngle(gunTurn + future)
        );
        setFire(1.0);
    }
}

```

Partimos do cálculo feito para a mira simples, que resulta na distância angular entre o canhão e a posição atual do inimigo, visto na Listagem 2.6. Uma vez que a movimentação do inimigo no eixo do canhão é irrelevante, já que isso não alteraria o ângulo da mira, só é preciso levar em conta sua movimentação lateral. Por isso é calculado o seno do ângulo para onde o inimigo está direcionado. Na Figura 6 pode ser observado o **BasicRobot** apontando o canhão na direção da linha pontilhada tendo previsto a movimentação do **MyFirstRobot** até o fim da mesma. A linha tracejada representa a movimentação lateral relevante para o cálculo do ângulo do canhão.

Figura 6 – BasicRobot com mira linear.



Agora basta multiplicar o seno pela velocidade do inimigo e depois dividir pela velocidade da bala que será atirada. Lembrando que a velocidade do projétil varia de acordo com sua força.

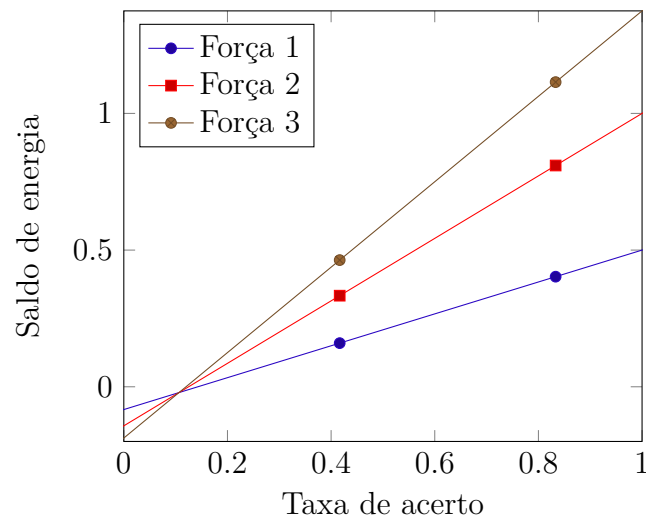
2.3.2.3 A força do tiro

Um aspecto que também deve ser levado em conta no controle do canhão é a força dos disparos. Gerenciar a própria energia é importante para sobrevivência enquanto o dano causado no adversário é relevante para a pontuação no jogo, assim, escolher a quantidade ótima de energia gasta em cada tiro não é uma tarefa trivial.

Até então, foram disparados tiros de força 1 em todas as ocasiões. Apesar de ser uma abordagem econômica, consumindo em média aproximadamente 0.08 de energia por turno, considerando uma taxa de acerto de 100%, causa apenas 0.33 de dano e recupera 0.25 de energia, em média, aproximadamente, por turno. Ou seja, um saldo de $0.33 + 0.25 - 0.08$, totalizando 0.5 pontos de energia por turno. Os saldos de energia a partir da taxa de acerto estão ilustrados na Figura 7 para diferentes tipos de força.

Como pode ser observado na Figura 7, para um inimigo que usa apenas tiros de força 2, uma taxa de acerto de 56.25% já é traduzida em saldo de 0.5 pontos de energia por turno, tendo por saldo máximo 1 ponto de energia por turno. De forma análoga, tiros de força 3 atingem o saldo de 0.5 pontos de energia por turno com taxa de acerto 44%, saldo máximo de 1.375. Por outro lado, taxas de acerto abaixo de 10.6% gera saldo negativo mais severo, para os tiros de força 2 e 3, que o equivalente em tiros de força 1. Para taxas de acerto de 0% teremos saldos de -0.08, -0.14 e -0.18 pontos de energia por turno, para tiros de força 1, 2 e 3, respectivamente.

Figura 7 – Saldo de energia por turno por força de tiro



Para gerenciar a força do tiro, as heurísticas abaixo, traduzidas em código na Listagem 2.8, podem ser utilizadas (ROBOWIKI, 2007):

- Em 1v1, tiro de força 2.0;
- Em melee, tiro de força máxima: 3.0;
- Quando o inimigo está muito distante, reduz a força; quando muito próximo, aumenta a força;
- Quando a própria energia estiver baixa, reduz a força;
- A força pode ser limitada pela quantidade estritamente necessária para destruir o inimigo.

Listagem 2.8 – Estratégia básica - Heurísticas de tiro

```

public class MyFirstRobot extends AdvancedRobot {
    ...
    public void shoot(ScannedRobotEvent e) {
        double firePower = decideFirePower(e);
        ...
        / Rules.getBulletSpeed(firePower);
        ...
        setFire(firePower);
    }
    public double decideFirePower(ScannedRobotEvent e) {

```

```
double firePower = getOthers() == 1 ? 2.0 : 3.0;

if (e.getDistance() > 400) {
    firePower = 1.0;
} else if (e.getDistance() < 200) {
    firePower = 3.0;
}

if (getEnergy() < 1) {
    firePower = 0.1;
} else if (getEnergy() < 10) {
    firePower = 1.0;
}

return Math.min(e.getEnergy() / 4, firePower);
}
}
```

2.3.3 Corpo

Ao contrário das estratégias de controle de canhão e do radar, cujas finalidades são, respectivamente, escolher e atingir um oponente e escanear o maior número de oponentes na menor quantidade de turnos, o objetivo por trás do controle do corpo não é trivial. Isso porque a lógica de movimentação pode ser baseada em diversos aspectos da batalha.

Observando os robôs de exemplo, é percebido:

- O movimento do **VelociRobot**, do **Crazy** e do **Spin** são independentes dos elementos externos;
- O **Walls** movimenta-se com base nas paredes da arena;
- O **Tracker** movimenta-se com base nos robôs inimigos;

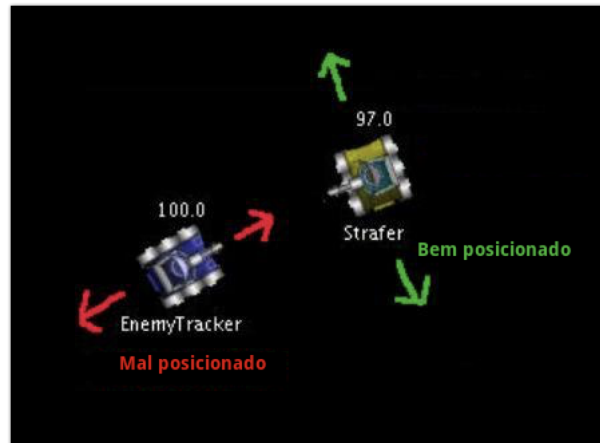
Quando algum evento de colisão é disparado, alguns dos robôs citados acima podem mover-se temporariamente com base na fonte da colisão.

2.3.3.1 Movimentação lateral

Como apresentado na seção 2.3.2.2, a movimentação do inimigo no eixo do canhão é desprezível no cálculo da mira, uma vez que o canhão não precisa rotacionar quando este é o caso. A Figura 8 ilustra bem a situação. O robô **EnemyTracker** está mal posicionado porque

não conseguirá fugir de um disparo do **Strafer** enquanto não rotacionar o corpo. Já o **Strafer** consegue fugir de disparos movimentando-se lateralmente, como está posicionado.

Figura 8 – Posicionamento lateral.



Uma estratégia de posicionamento do corpo é estar sempre perpendicular ao inimigo, explorando esse potencial de movimentação lateral (WHITLEY, 2003). Para tanto, basta rotacionar no sentido horário o *bearing* do inimigo mais 90 graus, como implementado na Listagem 2.9

Listagem 2.9 – Estratégia básica - Movimentação lateral

```
public class MyFirstRobot extends AdvancedRobot {
    ...
    public void onScannedRobot(ScannedRobotEvent e) {
        ...
        setTurnRadarRightRadians(turnToEnemy + extraTurn);
        setTurnRight(e.getBearing() + 90);
    }
    ...
}
```

2.4 Estratégias avançadas

Técnicas mais sofisticadas podem ser aplicadas no controle do canhão e do corpo do robô.

2.4.1 *Guess factor targeting*

Um conceito importante que serve de base para diversas estratégias tanto de movimentação quanto de mira é o do máximo ângulo de escape. No momento em que um tiro é disparado por um robô A , existe um setor tal que independente da movimentação, um inimigo B não conseguirá sair dele antes de uma bala, com velocidade v , percorrer a distância inicial.

Tomando como eixo central o raio que parte do robô A até o robô B , o arco do eixo até o fim do setor, projetado pelo máximo ângulo de escape, é a maior distância que o robô B pode percorrer no sentido horário ou anti-horário, conseguida quando este movimenta-se perpendicular ao robô A .

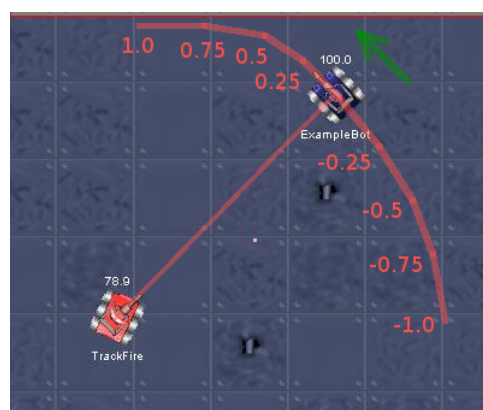
Uma vez que a velocidade máxima do robô é de $8\text{pixels}/\text{turno}$, o máximo ângulo de escape será $\arcsin(8/v)$.

Este método de controle do canhão consiste em dividir o máximo ângulo de escape em uma lista de setores, tratados como *guess factors*, e disparar um tiro na direção cujo setor tenha sido mais visitado com base nas experiências anteriores (ROBOWIKI, 2017c).

As visitas são contadas no momento t_f em que a distância viajada pela bala ultrapassa a distância entre o canhão e o inimigo no momento t_0 do disparo. Neste momento, a posição no vetor referente ao setor em que o inimigo se encontra é incrementado.

A lista tem, preterivelmente, tamanho ímpar, para que possa ser separada em 3 partes. A parte central é o setor no qual o inimigo se encontra em t_0 e as metades são referentes ao sentido no qual o inimigo se move em tal momento. A primeira metade é considerada quando, em t_f , o inimigo esteja na região contrária ao sentido que estava se locomovendo em t_0 . A segunda metade, quando o inimigo manteve-se na região do sentido em t_0 . A Figura 9 exemplifica isto com o máximo ângulo de escape do robô `ExampleBot` dividido em 9 setores: 4 positivos, 4 negativos e o central não explicitado.

Figura 9 – Máximo ângulo de escape dividido em *guess factors*.



Vamos começar criando a lista dos *guess factors* e selecionando o setor mais visitado para posicionar o canhão realizar o disparo, como na Listagem 2.10.

Listagem 2.10 – Estratégia avançada - Guess factor targeting - Selecionando o fator

```

public class MyFirstRobot extends AdvancedRobot {
    private GFGun gun = new GFGun();

    private static int GF_SIZE = 25;
    private static int GF_CENTER = (GF_SIZE - 1) / 2;
    private static int [] guessFactors = new int[GF_SIZE];
    ...
    public void onScannedRobot(ScannedRobotEvent e) {
        gun.onScannedRobot(e);
        double angleToEnemy =
            ...
    }
    public double maxEscapeAngle(double power) {
        return Math.asin(8 / Rules.getBulletSpeed(power));
    }
    public double getAbsoluteBearing(ScannedRobotEvent e) {
        return e.getBearingRadians() + getHeadingRadians();
    }
    public int getDirection(ScannedRobotEvent e) {
        double bearing = getAbsoluteBearing(e);
        double lateral = Math.sin(e.getHeadingRadians() - bearing);
        return lateral * e.getVelocity() < 0 ? -1 : 1;
    }
    public double decideFirePower(ScannedRobotEvent e) {
        ...
    }

    private class GFGun {
        public void onScannedRobot(ScannedRobotEvent e) {
            int direction = getDirection(e);
            double bearing = getAbsoluteBearing(e);
            double firePower = decideFirePower(e);
            double factor = getFactorFromIndex(getBestIndex());
            double angleOffset =
                direction * factor * maxEscapeAngle(firePower);
        }
    }
}

```

```

        double gunTurn = Utils.normalRelativeAngle(
            bearing - getGunHeadingRadians() + angleOffset
        );
        setTurnGunRightRadians(gunTurn);
        setFire(firePower);
    }
    private int getBestIndex() {
        int bestIndex = GF_CENTER;
        for (int i = 0; i < GF_SIZE; i++) {
            if (guessFactors[i] > guessFactors[bestIndex])
                bestIndex = i;
        }
        return bestIndex;
    }
    private double getFactorFromIndex(int index) {
        return (double) (index - GF_CENTER) / GF_CENTER;
    }
}
}

```

O antigo método `shoot` foi removido, e métodos utilitários foram adicionados. A lista de *guess factors* foi declarada estática para que as informações sejam mantidas entre uma batalha e outra.

Cada vez que o inimigo é escaneado, o setor mais visitado até então é selecionado e convertido em ângulo normalizado pelo máximo ângulo de escape.

Na Listagem 2.11 as informações do disparo são armazenadas para que no momento t_f seja possível resgatar as informações relevantes de t_0 . São elas: as coordenadas do canhão; o sentido do inimigo; a força do tiro; o *bearing* do inimigo e o momento do tiro.

Listagem 2.11 – Estratégia avançada - Guess factor targeting - Atualizando os fatores

```

import java.awt.geom.*;

```

```

public class MyFirstRobot extends AdvancedRobot {
    ...
    private static int [] guessFactors = new int [GF_SIZE];

    private Point2D targetLocation;
    ...
    public double getAbsoluteBearing(ScannedRobotEvent e) {

```

```
    ...
}
public double getAbsoluteBearing(Point2D source, Point2D target) {
    return Math.atan2(
        target.getX() - source.getX(), target.getY() - source.getY()
    );
}
public Point2D project(Point2D source, double angle, double length) {
    return new Point2D.Double(
        source.getX() + Math.sin(angle) * length,
        source.getY() + Math.cos(angle) * length
    );
}
...
private class GFGun {
    public void onScannedRobot(ScannedRobotEvent e) {
        ...
        double gunTurn = Utils.normalRelativeAngle(
            bearing - getGunHeadingRadians() + angleOffset
        );
        targetLocation = project(
            new Point2D.Double(getX(), getY()),
            bearing,
            e.getDistance()
        );
        setTurnGunRightRadians(gunTurn);
        if (setFireBullet(firePower) != null) {
            GFBullet gfBullet = new GFBullet(e, firePower);
            addCustomEvent(gfBullet);
        }
    }
    ...
}
private class GFBullet extends Condition {
    private Point2D gunLocation;
    private double bulletSpeed;
    private double bearing;
    private double lateralDirection;
    private double t0;
```

```

public GFBullet(ScannedRobotEvent e, double power) {
    this.bearing = getAbsoluteBearing(e);
    this.lateralDirection = getDirection(e);
    this.bulletSpeed = Rules.getBulletSpeed(power);
    this.gunLocation = new Point2D.Double(getX(), getY());
    this.t0 = getTime();
}
public boolean test() {
    double distanceTraveled = (getTime() - t0) * bulletSpeed;
    double limit = gunLocation.distance(targetLocation) - 18;
    if (distanceTraveled > limit) {
        updateFactor();
        removeCustomEvent(this);
    }
    return false;
}
private void updateFactor() {
    double currentBearing =
        getAbsoluteBearing(gunLocation, targetLocation);
    double angle =
        Utils.normalRelativeAngle(currentBearing - bearing);
    double normalizedAngle = angle / lateralDirection * GF_SIZE;
    int index = (int) Math.round(normalizedAngle + GF_CENTER);
    int safeIndex = Math.max(0, Math.min(index, GF_SIZE - 1));
    guessFactors[safeIndex]++;
}
}
}

```

Mais alguns métodos utilitários foram adicionados.

Duas alterações foram feitas no método `onScannedRobot` da class `GFGun`: a localização do inimigo será armazenada cada vez que ele é detectado e o método `setFire` foi substituído por `setFireBullet`. Esse método retorna a bala que foi disparada, ou `null` caso o canhão ainda esteja quente e o disparo não tenha sido realizado.

A classe `GFBullet` foi declarada como herdeira de `Condition`, uma vez passada como parâmetro de `addCustomEvent`, seu método `test` será chamado uma vez a cada turno, para verificar se a condição foi atendida. Daí a importância de garantir que apenas disparos bem sucedidos sejam lançados como condições a serem testadas.

O teste verifica se a distância viajada pela bala até então ultrapassou a distância entre a arma no instante t_0 e a posição atual do alvo menos a metade do tamanho do robô. Caso tenha ultrapassado, a lista de *guess factors* será atualizada com o ângulo do inimigo atual, relativo ao canhão em t_0 .

O teste sempre retorna false, para que o método `onCustomEvent` não seja chamado desnecessariamente.

2.4.2 Wave surfing

Esta técnica é uma estratégia de controle do corpo do robô cujo objectivo é desviar dos projéteis inimigos (ROBOWIKI, 2012b). O radar é incapaz de detectar balas no campo de batalha, tampouco consegue detectar caso um robô inimigo tenha realizado um disparo, mas isso pode ser inferido a partir da leitura contínua de sua energia.

Sabendo que um disparo gasta entre 0.1 e 3.0 pontos de energia do robô, quando uma baixa de tal magnitude for detectada em um inimigo, é provável este tenha realizado um disparo. Algumas alternativas geram falsos-positivos: colisão com parede ou outros robôs ou algum dano ocasionado por projétil com $f \leq 0.75$.

O escaneamento do inimigo do turno anterior será mantido para que seja possível detectar a queda de energia, como na Listagem 2.12.

Listagem 2.12 – Estratégia avançada - Wave surfing - Detectando o disparo

```
public class MyFirstRobot extends AdvancedRobot {
    private WaveSurfer body = new WaveSurfer ();

    ...
    public void onScannedRobot(ScannedRobotEvent e) {
        body.onScannedRobot(e);
        ...
    }
    ...
    private class WaveSurfer {
        private double lastScan = null;

        public void onScannedRobot(ScannedRobotEvent e) {
            if (lastScan != null) {
                double bulletPower =
                    lastScan.getEnergy() - e.getEnergy();
                if (0.1 <= bulletPower && bulletPower <= 3.0) {
                    System.out.println("Tiro detectado");
                }
            }
        }
    }
}
```

```

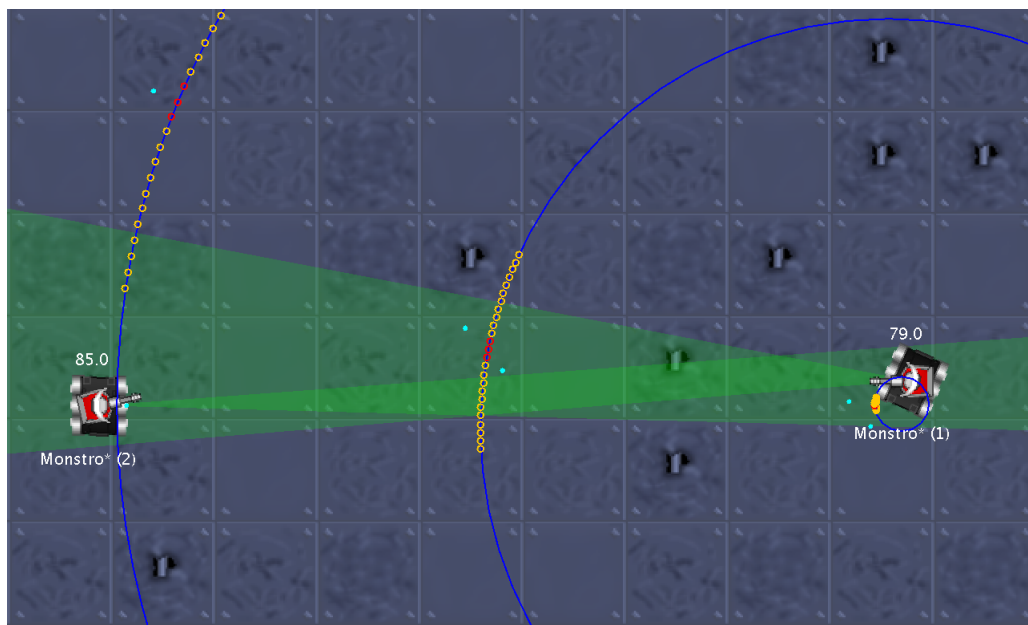
    }
  }
  lastScan = e;
}
}
}

```

Uma vez disparado, não sabemos a direção do tiro, mas sabemos o momento t_0 do disparo e sabemos a velocidade v da bala (calculada com base em sua força), ou seja, a bala pode estar em qualquer ponto de uma circunferência com centro na posição a qual o robô estava quando disparou o tiro e raio $(t - t_0) \times v$.

A circunferência em expansão é entendível como uma onda, sobre a qual existem regiões mais perigosas que outras. Na Figura 10 podem ser vistas 3 ondas referentes a disparos feitos pelo `Monstro*(1)`. O setor central da onda está marcado como perigoso e é evitado pelo `Monstro*(2)`. O efeito de movimentar-se de parte segura em parte segura de cada onda é abstraído como surfar essas ondas, daí o nome da estratégia. Essas ondas devem ser armazenadas, como na Listagem 2.13.

Figura 10 – Disparos abstraídos em ondas com zonas de perigo.



Listagem 2.13 – Estratégia avançada - Wave surfing - Armazenando as ondas

```

import java.util.ArrayList;

```

```

public class MyFirstRobot extends AdvancedRobot {
  ...

```



```
private Point2D targetLocation;
private ArrayList<EnemyWave> enemyWaves = new ArrayList();

public void run() {
    ...
    while(true) {
        ...
        body.updateWaves();
        execute();
    }
}
public void onScannedRobot(ScannedRobotEvent e) {
    ...
}
public Point2D getMyLocation() {
    return new Point2D.Double(getX(), getY());
}
...
private class WaveSurfer {
    public void onScannedRobot(ScannedRobotEvent e) {
        ...
        if (0.1 <= bulletPower && bulletPower <= 3.0) {
            enemyWaves.add(new EnemyWave(lastScan, bulletPower));
        }
        ...
    }

    public void updateWaves() {
        for (int i = 0; i < enemyWaves.size(); i++) {
            if (enemyWaves.get(i).test()) {
                enemyWaves.remove(i);
                i--;
            }
        }
    }
}
private class EnemyWave {
    private Point2D gunLocation;
    private double bulletSpeed;
    private double bearing;
```

```

private double lateralDirection;
private double t0;

public EnemyWave(ScannedRobotEvent e, double power) {
    this.bulletSpeed = Rules.getBulletSpeed(power);
    this.lateralDirection = getDirection(e);
    this.bearing = getAbsoluteBearing(e) + Math.PI;
    this.gunLocation = (Point2D.Double) targetLocation.clone();
    this.t0 = getTime() - 1;
}
public double distanceFromMe() {
    double radius = (getTime() - t0) * bulletSpeed;
    return gunLocation.distance(getMyLocation()) - radius;
}
public boolean test() {
    return distanceFromMe() < - 50;
}
}
}

```

Uma classe interna análoga à `GFBullet` foi adicionada, atributos similares são registrados na instanciação do objeto mudando a referência, uma vez que agora o disparo vem do inimigo.

Diferente da `GFBullet`, a `EnemyWave` não herda de `Condition`, uma vez que a lista de ondas será varrida de forma síncrona durante a movimentação do robô. Assim, em vez de usar o `removeCustomEvent` para remover a bala da pilha, o método `updateWaves` foi adicionado, para remover a onda da lista quando a mesma tiver ultrapassado o robô.

As regiões de perigo são construídas basicamente de forma análoga ao *guess factor targeting*. O máximo ângulo de escape será dividido em setores e armazenados como lista. Cada vez que o robô for atingido por uma bala, a posição na lista referente ao setor em que o robô estava sera incrementada, se tornando uma região mais perigosa. Essa atualização pode ser observada na Listagem 2.14.

Listagem 2.14 – Estratégia avançada - Wave surfing - Atualizando as áreas de perigo

```

public class MyFirstRobot extends AdvancedRobot {
    ...
    public static double[] dangerFactors = new double[GF_SIZE];
    ...
}

```

```

public void run() {
    ...
}
public void onHitByBullet(HitByBulletEvent e) {
    double speed = Rules.getBulletSpeed(e.getBullet().getPower());
    Point2D location = new Point2D.Double(
        e.getBullet().getX(), e.getBullet().getY()
    );

    EnemyWave hitWave = null;
    for (EnemyWave ew: enemyWaves) {
        boolean isClose = Math.abs(ew.distanceFromMe()) < 50;
        boolean hasSameSpeed = ew.bulletSpeed - speed < 0.001;
        if (isClose && hasSameSpeed) {
            hitWave = ew;
            break;
        }
    }
    if (hitWave != null) {
        body.updateFactors(hitWave, location);
        enemyWaves.remove(enemyWaves.lastIndexOf(hitWave));
    }
}
...
private class WaveSurfer {
    ...
    public void updateFactors(EnemyWave wave, Point2D location) {
        int index = wave.getFactorIndex(location);
        for (int i = 0; i < GF_SIZE; i++) {
            dangerFactors[i] += 1.0 / (Math.pow(index - i, 2) + 1);
        }
    }
}
private class EnemyWave {
    ...
    public int getFactorIndex(Point2D location) {
        double offset = Utils.normalRelativeAngle(
            getAbsoluteBearing(gunLocation, location) - bearing
        );
    }
}

```

```

        double factor =
            offset / maxEscapeAngle(bulletSpeed) * lateralDirection;
        int index =
            (int) Math.round(factor * GF_CENTER + GF_CENTER);
        return Math.max(0, Math.min(index, GF_SIZE - 1));
    }
}
}

```

No momento em que o robô é atingido por uma bala, a lista de ondas é varrida em busca de uma compatível em distância e em velocidade. Quando alguma é encontrada, os fatores de perigo serão atualizados de acordo com o ângulo da colisão relativo à posição do canhão do inimigo no instante t_0 . Na posição exata referente ao fator, a lista é incrementada em 1 unidade, aos imediatamente adjacentes é acrescido 0.5 e assim sucessivamente em queda exponencial.

Por fim, a lista será utilizada para guiar a movimentação do robô, que pode se dar de várias formas. Uma forma bem simples é a chamada *goTo*. Um fator seguro é selecionado da lista e então ele é projetado na distância do robô, esse ponto projetado será o destino do robô, como pode ser observado na Listagem 2.15.

Listagem 2.15 – Estratégia avançada - Wave surfing - Surfando as ondas

```

public class MyFirstRobot extends AdvancedRobot {
    ...
    public void run() {
        setAdjustRadarForGunTurn(true);
        setAdjustRadarForRobotTurn(true);
        while (true) {
            if (getRadarTurnRemaining() == 0.0)
                setTurnRadarRightRadians(Double.POSITIVE_INFINITY);
            body.updateWaves();
            body.surf();
            execute();
        }
    }
    ...
    public void onScannedRobot(ScannedRobotEvent e) {
        ...
        setTurnRadarRightRadians(turnToEnemy + extraTurn);
    }
    ...
}

```

```
private class WaveSurfer {
    ...
    public void updateWaves() {
        ...
    }
    public void surf() {
        EnemyWave closestWave = getClosestWave();
        if (closestWave != null) {
            goTo(closestWave.getSafestSpot());
        }
    }
    public EnemyWave getClosestWave() {
        double minDistance = Double.POSITIVE_INFINITY;
        EnemyWave closestWave = null;
        for (EnemyWave wave: enemyWaves) {
            double distance = wave.distanceFromMe();
            if (distance < minDistance
                && distance > wave.bulletSpeed) {
                closestWave = wave;
                minDistance = distance;
            }
        }
        return closestWave;
    }
    public void goTo(Point2D spot) {
        int x = (int) spot.getX() - (int) getX();
        int y = (int) spot.getY() - (int) getY();
        double turn = Math.atan2(x, y);
        setTurnRightRadians(Math.tan(turn - getHeadingRadians()));
        setAhead(Math.hypot(x, y) * Math.cos(turn));
    }
}

private class EnemyWave {
    ...
    public Point2D getSafestSpot() {
        int safestIndex = GF_CENTER;
        for (int i = 0; i < GF_SIZE; i++) {
            if (dangerFactors[i] < dangerFactors[safestIndex]) {
```

```

        safestIndex = i;
    }
}

double offset =
    (double) (safestIndex - GF_CENTER) / GF_CENTER;
double angle =
    lateralDirection * offset * Math.asin(8 / bulletSpeed);
double distance = gunLocation.distance(getMyLocation()) - 18;
return project(gunLocation, angle + bearing, distance);
}
}
}

```

2.5 Robôs avançados

É mantido pela RoboWiki um ranking mundial de robôs, que batalham nas diferentes categorias disponíveis, separadas por modos (*melee*, *1v1* e por time) e por tamanho de código.

2.5.1 DrussGT

Desenvolvido pelo usuário Skillgannon, atualmente o #1 no ranking mundial 1v1. Usa *wave surfing* para movimentação e *dynamic clustering* para o controle do canhão (ROBOWIKI, 2017b).

2.5.2 Diamond

No ranking mundial ocupa atualmente a segunda posição no 1v1 e terceira no *melee*, foi criado por *Voidious*, o atual responsável pela RoboWiki. Usa *Minimum Risk Movement* em *melee*, mudando para *Wave Surfing* com *Dynamic clustering* em 1v1. O canhão também utiliza *Dynamic Clustering* (ROBOWIKI, 2017a).

2.5.3 Neuromancer

Também desenvolvido pelo *Skillgannon* é o campeão mundial do modo *melee*. Controla o corpo com *wave surfing* tentando desviar de todas as balas (não apenas as direcionadas a ele); e o canhão com *dynamic clustering* (ROBOWIKI, 2017d).

3 Robocode Brasil

A Liga Brasileira, promovida pelo LIAG da FT-UNICAMP, é o torneio mais importante de Robocode no Brasil (ROBOCODEBRASIL, 2016). É dividida em duas etapas: o torneio local, uma etapa classificatória que ocorre nas instituições cadastradas como sede; e a liga nacional, disputada pelas equipes campeãs de cada torneio regional.

3.1 Regras

As instituições interessadas devem inscrever no mínimo quatro equipes de um a quatro participantes para sediarem um torneio local. Caso não consiga, será inscrita no torneio público.

Os torneios locais e a liga nacional ocorrem em duas etapas: a primeira etapa é classificatória; a segunda etapa é eliminatória, com semifinais e final.

Até no máximo um dia antes de cada rodada, o código de cada equipe deve ser submetido. Após a realização da rodada, os códigos das equipes ficaram disponíveis para livre acesso das outras.

Cada rodada é composta por 3 batalhas em uma arena $800px \times 800px$. Entre as batalhas, são dados intervalos de 10 minutos dentro dos quais as equipes podem realizar alterações no código do robô. As pontuações de cada batalha são somadas e definem o resultado da rodada.

Nas rodadas classificatórias, as batalhas são disputadas por todas as equipes daquela competição (liga ou torneio). Caso o número de equipes seja maior que 8, a arena terá dimensões $1200px \times 1200px$. Os resultados das rodadas pontuam as equipes participantes de acordo com a Tabela 1.

Tabela 1 – Pontuação das classificatórias por colocação

Colocação	Pontos
1° Colocado	10
2° Colocado	8
3° Colocado	6
4° Colocado	5
5° Colocado	4
6° Colocado	3
7° Colocado	2
8° Colocado	1
Demais	0

As pontuações das rodadas classificatórias são somadas para montar a chave das semifinais, seguindo a Tabela 2.

Tabela 2 – Chave das semifinais

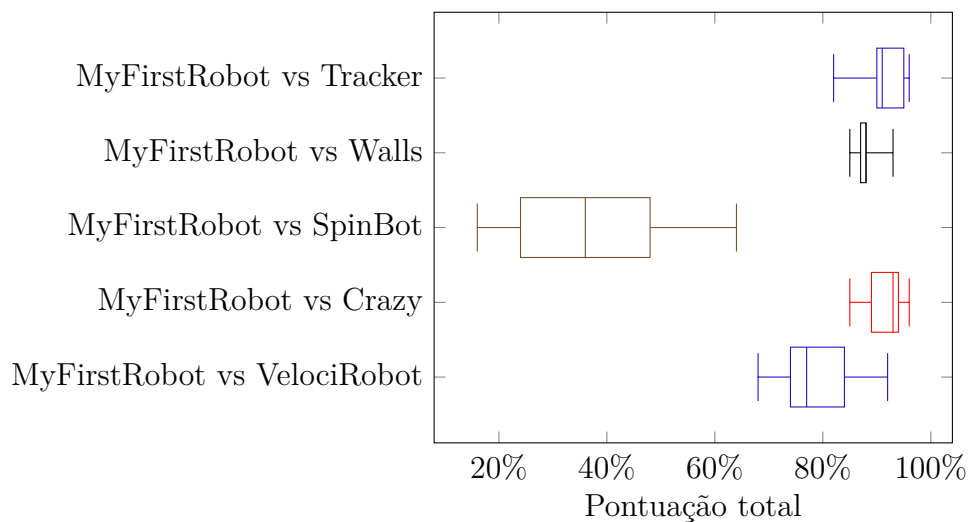
1ª Batalha	1º Colocado	4º Colocado
2ª Batalha	2º Colocado	3º Colocado

As campeãs de cada batalha disputam a rodada final. A rodada final do torneio local classifica seu campeão para a liga nacional. Caso o torneio tenha sido disputado por mais de 8 equipes, 2 serão classificadas.

3.2 Etapa regional

Os primeiros passos dados foram de implementar as estratégias básicas exploradas na Seção 2.3: Escaneamento seguro, Movimentação lateral e Mira linear. Rodando 25 batalhas 1v1 contra os exemplos citados na Seção 2.2 e seguindo as configurações do torneio, os resultados da Figura 11 foram obtidos. A exceção do SpinBot, não houve derrota alguma contra os outros robôs.

Figura 11 – MyFirstRobot em 1v1 contra os robôs básicos



Estratégias de movimentação contínua, mas curta, como a do próprio MyFirstRobot, do SpinBot e do VelociRobot são eficientes contra *linear targeting*. Ora, a projeção é feita com base na hipótese de que o robô manterá sua direção, velocidade e sentido nos próximos turnos. Quanto maior a distância, maior é o tempo necessário que o robô mantenha sua movimentação. O robô foi então alterado, como na Listagem 3.1, para aproximar-se do inimigo caso esteja muito distante, mas afastar-se caso esteja muito próximo. Seguindo a heurística básica do tiro, a movimentação do robô foi alterada para buscar a distância intermediária do intervalo separado para tiro de força 2: 300px.

Listagem 3.1 – Heurística de aproximação do alvo

```

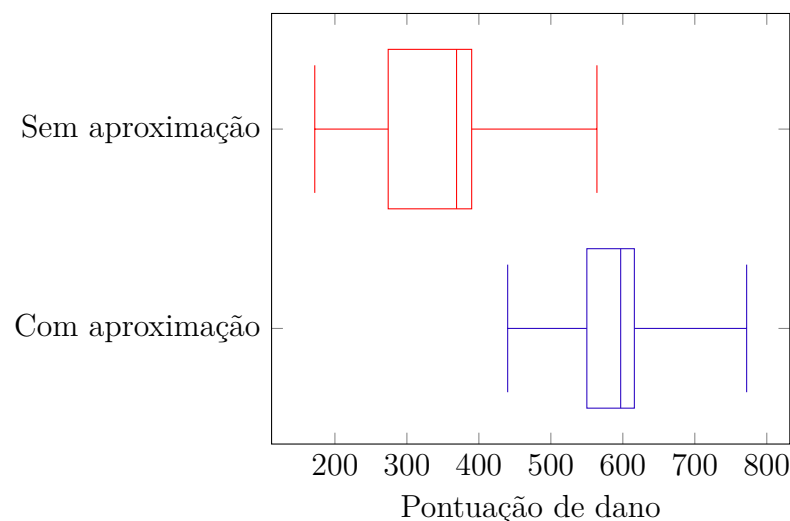
public class MyFirstRobot extends AdvancedRobot {
    ...
    public void onScannedRobot(ScannedRobotEvent e) {
        ...
        setTurnRadarRightRadians(turnToEnemy + extraTurn);
        double skew =
            (e.getDistance() - 300) / 5 * -Math.signum(getVelocity());
        setTurnRight(e.getBearing() + 90 + skew);
    }
    ...
}

```

Para centralizar o robô na distância de 300px do inimigo, uma leve inclinação foi adicionada ao ângulo de giro do corpo do robô. Caso o robo esteja muito próximo, o ângulo deve ser mais aberto, do contrário o ângulo deve ser mais fechado. Se o robô estiver se movendo no sentido contrário do seu *heading*, a inclinação deve ser contrária para surtir o mesmo efeito.

A consequência de aprimorar a movimentação em relação à mira do robô pode ser observada na Figura 12. Em 25 batalhas, o pior resultado foi superior a 75% dos resultados sem a estratégia. Apesar disso, o resultado geral foi afetado negativamente, como pode ser observado na Figura 13, uma vez que aproximar-se do SpinBot também facilita que este acerte tiros.

Figura 12 – Heurística de aproximação contra o SpinBot



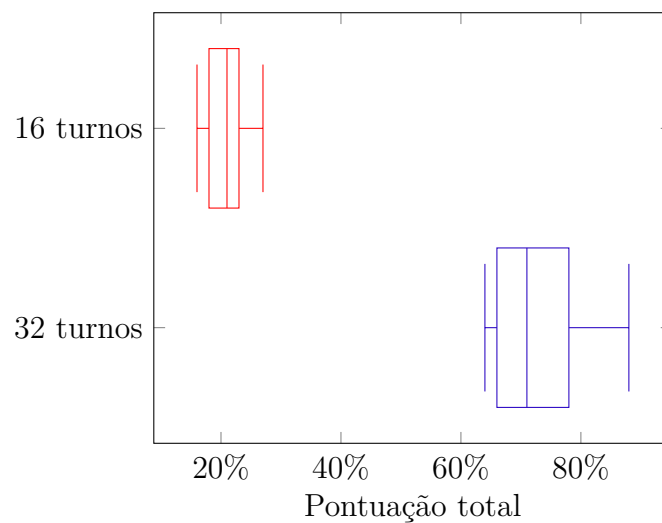
O problema da movimentação do robô é ser muito curta. Isso faz com que as estratégias mais básicas de tiro, disparar exatamente na direção a qual o inimigo foi encontrado, consigam acertar com boa frequência. Dobrando a duração do ciclo da movimentação,

como na Listagem 3.2, que inclusive é o mesmo período do `VelociRobot`, os resultados da Figura 13 foram obtidos.

Listagem 3.2 – Redução do período dos ciclos de movimentação

```
public class MyFirstRobot extends AdvancedRobot {
    public void run() {
        ...
        if (counter < 32)
        ...
        counter = (counter + 1) % 64;
        ...
    }
    ...
}
```

Figura 13 – Diferentes períodos para troca de sentido contra o SpinBot



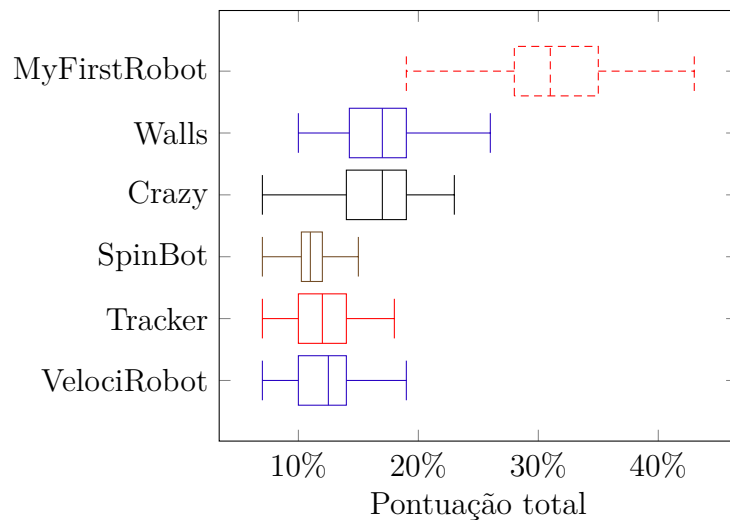
Nenhuma derrota foi registrada após a mudança no tamanho do ciclo. O pior resultado foi muito superior ao melhor resultado com a estratégia anterior, que foi derrotada em 100% dos casos.

Até então, os testes feitos com as estratégias foram todos em batalhas do tipo 1v1, mas as primeiras rodadas dos torneios são *melee*. O desempenho do robô em um *melee* contra os robôs de exemplo podem ser observados na Figura 14. Foram executadas 50 batalhas.

As seguintes heurísticas foram adicionadas para adaptar melhor o robô ao *melee*:

- O tamanho do ciclo foi tornado aleatório, de modo a equilibrar a esquiva contra mira básica com a mira linear: períodos maiores são alvos mais fáceis de uma mira linear;

Figura 14 – MyFirstRobot em melee contra os robôs de exemplo



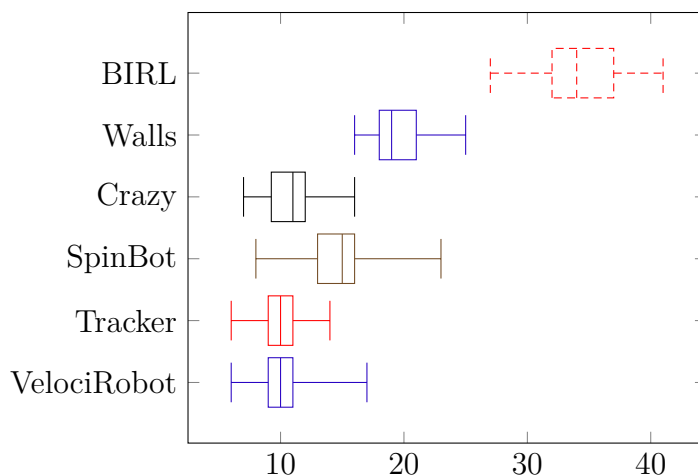
- No início de cada ciclo, o alvo do radar é alternado, a não ser que:
 - Seja o único inimigo restante na arena;
 - Esteja muito próximo;
 - Tenha pouca energia;
- A colisão do robô com o inimigo muda o alvo do radar para a fonte da colisão;
- Quando atingido por uma bala, o radar é apontado em sua direção, exceto quando:
 - O alvo atual tenha pouca energia;
 - O alvo atual seja o único inimigo restante na arena;
 - A distância angular do radar à direção da bala seja muito curta, o que significa que o inimigo provavelmente já é o alvo;
- Ao colidir com a parede, o sentido da movimentação é alternado.

Após a implementação das heurísticas, um teste de 50 batalhas contra os robôs de exemplo gerou o resultado da Figura 15: nenhuma derrota. A pior pontuação registrada foi superior à melhor pontuação de todos os inimigos. Estava pronto o robô, denominado BIRL, para a primeira rodada do torneio.

5 equipes foram inscritas para participar do torneio local sediado na Universidade Federal da Bahia: *BIRL - Beyond Intelligent Robotic Lifeform*, *Skynet UFBA*, Liga da Justiça, Orion e *WeDontHaveName*¹.

Foram disputadas 5 rodadas classificatórias, durante as quais nenhuma alteração foi feita no código do BIRL. Isso porque uma vez que o número de equipes participantes

¹ A equipe *WeDontHaveName* não participou de nenhuma das rodadas.

Figura 15 – BIRL em *melee* contra robôs de exemplo

da etapa local era igual ao número de vagas nas semifinais, a equipe já estava classificada para as rodadas eliminatórias. Além disso, o código seria disponibilizado para as demais equipes. Daí optamos por não mostrar as estratégias mais avançadas nesse momento da competição.

3.2.1 Primeira rodada classificatória

Os robôs submetidos pelas equipes para disputar a primeira rodada foram: **BIRL**, pela equipe BIRL; **Terminator**, pela *Skynet UFBA*; **THawk** pela Orion e **RoboC4** pela Liga da Justiça.

Analisando os códigos e comportamentos dos robôs adversários ao BIRL, não foi percebida implementação de nenhuma das estratégias básicas ou avançadas apresentadas neste trabalho. Dentre eles, a estratégia de controle de corpo mais elaborada foi a do RoboC4, uma máquina de estados baseada no histórico dos tiros disparados e recebidos.

Os resultados desta rodada podem ser observados na Tabela 3.

Tabela 3 – Primeira rodada do torneio local

Colocação	Equipe	Pontuação da rodada	Pontuação do torneio
1°	BIRL	6610	10
2°	Skynet UFBA	1625	8
4°	Liga da Justiça	1169	6
3°	Orion	1140	5

3.2.2 Segunda rodada classificatória

Os mesmos robôs da rodada anterior foram submetidos nesta rodada, a exceção da equipe Orion, que submeteu um novo robô de nome **Orion**. Neste foi percebida a implementação de um escaneamento preso, programado de modo que toda vez que o um

inimigo é escaneado, o robô aponta o radar em sua direção, mas as heurísticas de reação a tiro recebido, colisão com a parede e colisão com outro robô fazem chamadas bloqueantes de girar o corpo. Isso prejudica a execução do algoritmo do escaneamento.

Os resultados desta rodada podem ser observados na Tabela 4.

Tabela 4 – Segunda rodada do torneio local

Colocação	Equipe	Pontuação da rodada	Pontuação do torneio
1°	BIRL	5577	20
2°	Liga da Justica	1566	14
3°	Skynet UFBA	1546	14
4°	Orion	495	10

3.2.3 Terceira rodada classificatória

Nesta rodada apenas 3 robôs foram submetidos, o BIRL, o Terminator e o Orion. Nesta rodada foi possível detectar uma estratégia mais clara do Terminator, que aproximou-se da tática do robô Walls, já o Orion alterou as heurísticas de reação da rodada anterior, removendo as chamadas de movimentação bloqueante.

Os resultados desta rodada podem ser observados na Tabela 5.

Tabela 5 – Terceira rodada do torneio local

Colocação	Equipe	Pontuação da rodada	Pontuação do torneio
1°	BIRL	3986	30
2°	Skynet UFBA	1054	22
3°	Orion	841	16
4°	Liga da Justiça	0	14

3.2.4 Quarta rodada classificatória

Nesta rodada as equipes Orion e Liga da Justiça mantiveram os mesmos robôs, já a Skynet submeteu um novo robô chamado G2, com estratégia de movimentações circulares.

Os resultados desta rodada podem ser observados na Tabela 6.

Tabela 6 – Quarta rodada do torneio local

Colocação	Equipe	Pontuação da rodada	Pontuação do torneio
1°	BIRL	5962	40
2°	Skynet UFBA	2220	30
3°	Liga da Justiça	1284	20
4°	Orion	865	21

3.2.5 Quinta rodada classificatória

As equipes Liga da Justiça e Orion mantiveram os mesmos robôs, enquanto a equipe *Skynet* submeteu um novo chamado *Gêneseis*, de comportamento similar ao *G2*, mudando a heurística de força do tiro e realizando o disparo de forma assíncrona, ao contrário do anterior.

Os resultados desta rodada podem ser observados na Tabela 7.

Tabela 7 – Quinta rodada do torneio local

Colocação	Equipe	Pontuação da rodada	Pontuação do torneio
1°	BIRL	6448	50
2°	Skynet UFBA	2089	38
3°	Liga da Justiça	1224	26
4°	Orion	1055	26

3.2.6 Semifinais

Para as rodadas eliminatórias, *Wave Surfing* foi implementado com movimentação básica: em vez de escolher um ponto seguro na wave e deslocar-se até ele (como *goTo* explorado na Seção 2.4.2), o corpo é mantido perpendicular ao único inimigo na arena, como na Movimentação lateral e são feitas duas simulações de movimentação, uma no sentido horário, outra no anti-horário. Para as simulações são calculadas as posições do robô e das ondas nos turnos seguintes até o robô colida com a onda mais próxima². Quando a colisão ocorre, é verificado o nível de perigo naquele ponto da onda. Os níveis de perigo de cada simulação são comparados, e o robô se move no sentido menos perigoso.

A simulação de um ponto é calculada com base no ângulo perpendicular ao inimigo e uma distância referente à velocidade do robô. Entretanto, essa angulação pode ocasionar na colisão com a parede. Para evitar que isso ocorra, a técnica *Wall smoothing* foi adicionada à simulação: em vez de projetar a velocidade do robô no ângulo perpendicular ao inimigo, é projetada uma distância de segurança contra a parede. Caso o ponto projetado esteja fora dos limites da arena, a angulação é ajustada incrementalmente até que seja projetada para dentro.

Uma heurística foi adicionada ao *linear targeting* para alternar a angulação do tiro caso muitos disparos errassem o alvo.

Com o resultados das rodadas classificatórias, a chave para as semifinais foi configurada segundo a Tabela 8.

A equipe Orion não submeteu o código do robô, tendo sido desclassificada. Com isso, a equipe BIRL foi direto para a final. A equipe Liga da Justiça manteve o robô,

² É configurado um limite na distância em turnos da simulação, para o caso de não haver onda alguma.

Tabela 8 – Chave das semifinais do torneio local

1ª Batalha	BIRL	Orion
2ª Batalha	Skynet UFBA	Liga da Justiça

já a equipe *Skynet* UFBA submeteu um robô similar ao *SpinBot*, mas com velocidade máxima e heurísticas de reação a colisões diferentes, chamado novamente de *Terminator*. O resultado desta batalha pode ser observado na Tabela 9.

Tabela 9 – Resultado da segunda batalha das semifinais locais

Equipe	Pontuação da rodada
Skynet UFBA	2246
Liga da Justiça	827

3.2.7 Final

A equipe *Skynet* e a equipe BIRL submeteram robôs diferentes de todas as rodadas anteriores, respectivamente: *Terminator_T800* e *Monstro*. A estratégia de movimentação do *Terminator_T800* é de posicionar-se em um ângulo de 60° em relação ao inimigo e alternar o sentido da movimentação cada vez que detectar um disparo. O canhão foi mantido acoplado ao radar e é configurado para girar no sentido horário até que algum robô seja escaneado, quando então o sentido é alternado e um disparo é realizado.

De todas as rodadas, esta foi a mais equilibrada, pois a estratégia de movimentação implementada no *Terminator_T800* é bastante eficaz contra a mira linear implementado no *Monstro*. Apesar disso, a equipe BIRL venceu com a pontuação apresentada na Table 10 e foi classificada para a etapa nacional.

Tabela 10 – Resultado da final regional

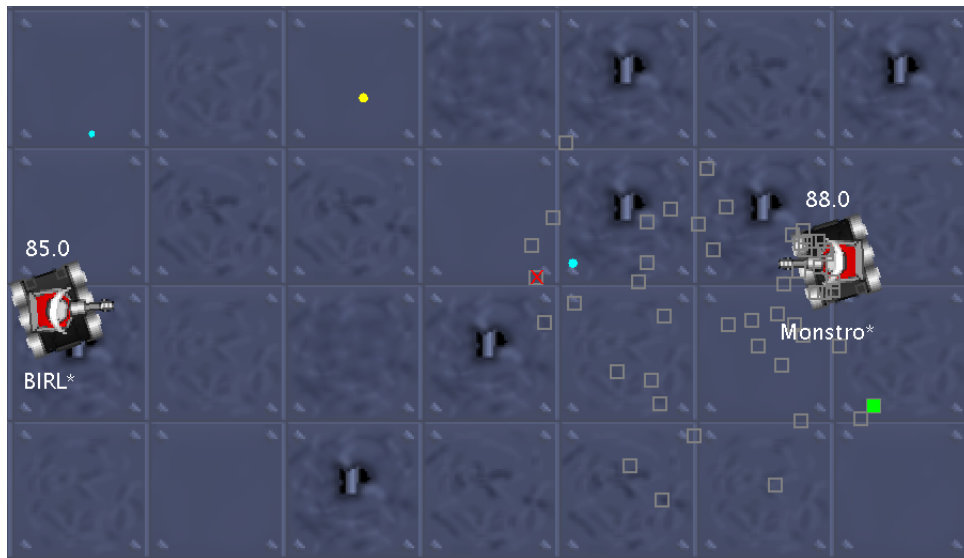
Equipe	Pontuação da rodada
BIRL	1502
Skynet UFBA	1073

3.3 Etapa nacional

As primeiras rodadas da liga nacional são disputadas em uma arena com todos os campeões dos torneios regionais. Para controlar a movimentação de modo mais adequado ao modo *melee*, a estratégia de movimentação *Risk Evaluation* foi implementada: vários pontos são sorteados nas proximidades do robô e então é escolhido o ponto que oferece menor risco, levando em consideração o perigo da ondas, distância dos inimigos, distância do centro da arena, e das paredes. A cada fator é atribuído um peso servindo de ponderação dos

riscos. A Figura 16 ilustra essa estratégia implementada no robô *Monstro*. Os quadrados representam os pontos sorteados nas proximidades do robô, o mais próximo do BIRL, marcado com um "x", foi calculado como ponto de maior risco; já o preenchido, pouco abaixo do *Monstro*, como de menor risco.

Figura 16 – *Risk evaluation*.



Para controle do canhão, foi implementado o *Guess Factor Targeting* com uma heurística para escolha do alvo, que pondera a distância até o robô, quantidade de energia e a distância angular do canhão. Além disso, mais dimensões foram adicionadas nos *guess factors*. As dimensões adicionadas foram: distância, velocidade atual do inimigo e velocidade anterior do inimigo. Isso quer dizer que a lista de fatores para o inimigo próximo é diferente da lista para quando o inimigo está distante, combinado com as possibilidades de estar movendo no sentido horário ou anti-horário.

O radar também foi aprimorado para cobrir uma maior quantidade de inimigos por escaneamento. Em cada varredura é calculado o ângulo do inimigo mais distante angularmente do radar. Caso a última varredura não tenha detectado todos os inimigos, o ângulo é aumentado.

Nas rodadas eliminatórias os pesos das ponderações, as segmentações do *guess factor targeting* e a heurística da força de tiro foram alterados.

9 campeões dos torneios locais foram classificados para a liga nacional e disputaram, com exceção da equipe desistente Dot GTX, as duas primeiras rodadas classificatórias para as semifinais, como dispostos na Tabela 11.

Tabela 11 – Campeões dos torneios locais.

Sede	Equipe
Sem sede (torneio público)	RecrutaZero
Unicamp	Teleton Rejecteds
Faculdade SATC	Skynet
Faculdade Anhanguera	Moto
IFSP - Campus Capivari	Dragonborn
Pontifícia Universidade Católica do Paraná	Dot GTX
Universidade Federal da Bahia	BIRL - Beyond Intelligent Robotic Lifeform
Instituto Federal Farroupilha	Equipe Rocket
Instituto Federal Farroupilha	OutOfTheCage

3.3.1 Primeira rodada

As estratégias implementadas por cada uma das equipes na primeira rodada classificatória foram as seguintes:

3.3.1.1 Equipe Rocket

O robô submetido pela equipe Rocket, chamado **Chupingole**, usa mira linear para controle do canhão, escaneamento preso para controle do radar, e uma heurística de aproximação tentando manter o robô na faixa de 140 pixels de distância do inimigo. Quando colide com a parede, o sentido da movimentação é alternado, afastando-o do inimigo cada vez mais, até colidir novamente contra a parede.

3.3.1.2 Dragonborn

A equipe Dragonborn submeteu o robô **SharkMK3**, que usa mira simples para controlar o canhão, escaneamento preso para o radar e movimentação lateral com uma leve inclinação extra para aproximar-se lentamente do inimigo.

3.3.1.3 Skynet

A equipe Skynet submeteu o robô de mesmo nome, cuja estratégia de movimentação é similar à do **SpinBot**, no radar utiliza escaneamento básico e no canhão, apesar do código aparentar implementar mira simples, seu comportamento não é compatível. Isso porque o canhão não foi desacoplado do corpo, e o ângulo de giro não é normalizado.

3.3.1.4 RecrutaZero

A equipe RecrutaZero submeteu um código idêntico ao do robô BIRL, submetido na primeira rodada do torneio local. As diferenças são, em sua quase totalidade, nomes das variáveis e dos métodos.

3.3.1.5 Moto

A equipe Moto submeteu o robô M4L4NDR4M3NT, cuja estratégia de controle do corpo é de mover-se até uma parede, e lá manter movimentação próxima à do MyFirstRobot inicial: move-se 100 *pixels* para frente, 100 *pixels* para trás, gira o canhão em 360 graus e atira quando escanear um inimigo, com força de tiro baseado em uma heurística de distância e quantidade de energia. Após isso, o canhão é direcionado à sua posição. Como o radar é mantido acoplado ao canhão, há o efeito momentâneo de escaneamento preso, que é perdido quando a movimentação é retomada.

3.3.1.6 Teleton Rejecteds

A estratégia do robô Banne, submetido pela equipe Teleton Rejecteds, é de mover-se alternando brevemente entre sentido horário e anti-horário perpendicularmente ao inimigo detectado com uma heurística de aproximação. No radar foi implementado escaneamento básico e no canhão, mira linear.

3.3.1.7 OutOfTheCage

O robô submetido pela equipe OutOfTheCage, de mesmo nome, tem por estratégia de controle do corpo mover-se perpendicularmente ao inimigo detectado até chegar próximo a alguma parede, quando muda o sentido do movimento. No radar foi implementado a variante de escaneamento preso que consiste em apontar o radar na direção do inimigo escaneado. No canhão foi implementada mira linear.

Os resultados desta rodada podem ser observados na Tabela 12.

Tabela 12 – Primeira rodada da liga nacional.

Colocação	Equipe	Pontuação da rodada	Pontuação do torneio
1°	OutOfTheCage	5811	10
2°	BIRL	5789	8
3°	Teleton Rejecteds	5210	6
4°	RecrutaZero	4328	5
5°	Dragonborn	3946	4
6°	Moto	3594	3
7°	Equipe Rocket	3002	2
8°	Skynet	1782	1

3.3.2 Segunda rodada

As equipes Teleton Rejecteds, OutOfTheCage e Equipe Rocket submeteram os mesmos robôs da rodada anterior. As estratégias das demais foram as seguintes:

3.3.2.1 Skynet

O robô submetido pela Skynet, de mesmo nome, tem por estratégia de controle de corpo movimentação lateral; controla o canhão com mira simples e o radar com uma variação do escaneamento seguro, cuja rotação extra é fixa em aproximadamente 22 graus.

3.3.2.2 Dragonborn

A equipe Dragonborn submeteu o robô **SharkMK4**, que, pela análise do código, aparenta ter utilizado algoritmo genético. Seu comportamento é semelhante ao do **SpinBot**, mas com velocidade máxima, mira simples e uma variação de radar preso.

3.3.2.3 RecrutaZero

O código do robô de nome **MegaBot**, submetido pela equipe RecrutaZero, é, em sua quase totalidade, idêntico ao submetido pela equipe OutOfTheCage na rodada anterior, removendo comentários e alterando o nome das variáveis.

3.3.2.4 Moto

O robô **M4L4NDR4M3NT**, submetido pela equipe Moto, é similar ao submetido na rodada anterior. A movimentação base é a mesma, exceto que o giro do canhão deixou de ser síncrono, fazendo com que o canhão gire enquanto o robô se movimenta. A heurística da força de tiro foi removida, dando lugar a escolher apenas tiros de força 3. Uma heurística de reação a colisão com inimigo foi adicionada.

Os resultados desta rodada podem ser observados na Tabela 13.

Tabela 13 – Segunda rodada da liga nacional.

Colocação	Equipe	Pontuação da rodada	Pontuação do torneio
1°	Moto	5529	13
2°	BIRL	4923	16
3°	Skynet	4303	7
4°	OutOfTheCage	4123	15
5°	RecrutaZero	3987	9
6°	Teleton Rejecteds	3980	9
7°	Dragonborn	3355	6
8°	Equipe Rocket	2240	3

3.3.3 Semifinais

Com o resultado das rodadas classificatórias, a chave para as semifinais foi configurada segundo a Tabela 14.

Tabela 14 – Chave das seminifinais da liga nacional.

1ª Batalha	BIRL	Teleton Rejecteds
2ª Batalha	OutOfTheCage	Moto

A equipe Teleton Rejecteds submeteu um robô de nome **Bannne**, que embora a partida tenha ocorrido com o resultado apresentado na Tabela 15, a equipe foi posteriormente desclassificada por plágio do robô **DrussGT**.

Tabela 15 – Resultado BIRL contra Teleton Rejecteds.

Colocação	Equipe	Pontuação da rodada
1º	BIRL	1072
2º	Teleton Rejecteds	597

Por consequência da desclassificação da equipe Teleton Rejecteds, a quinta colocada, **RecrutaZero**, foi classificada para as semifinais. O robô de nome **MasterBot**, submetido pela equipe **RecrutaZero**, sofreu poucas alterações em comparação à versão submetida na primeira rodada da liga nacional. As heurísticas de aproximação e movimentação básica foram alteradas. Nesta versão o robô tenta manter-se à distância de 200 pixels e o tamanho do ciclo de alternância do sentido é sorteado apenas uma vez antes de entrar no *loop* de execução principal.

O resultado da partida semifinal da equipe BIRL contra a equipe **RecrutaZero** foi o exposto na Tabela 16.

Tabela 16 – Resultado da primeira batalha das semifinais nacionais.

Colocação	Equipe	Pontuação da rodada
1º	BIRL	1663
2º	RecrutaZero	684

As equipes **Moto** e **OutOfTheCage** submeteram, para a segunda batalha das semifinais nacionais, os robôs **M4L4NDR4M3NT** e **OutOfTheCage** respectivamente. Neste não foi observada mudança alguma em relação à rodada anterior, naquele o robô deixou de mover-se até a parede, mantendo-se em vez disso, perpendicular ao inimigo, com a mesma lógica anterior do ciclo de 100 pixels para frente e para trás. A escolha da força do tiro foi implementada uma heurística com base na distância do inimigo relativa ao tamanho da arena.

O resultado da partida semifinal da equipe **Moto** contra a equipe **OutOfTheCage** foi o exposto na Tabela 17.

Tabela 17 – Resultado da segunda batalha das semifinais nacionais

Colocação	Equipe	Pontuação da rodada
1°	Moto	2170
2°	OutOfTheCage	263

3.3.4 Final

O robô M4L4NDR4M3NT, submetido pela equipe Moto, retomou a estratégia de mover-se até a parede, mas agora o tamanho dos ciclos de alternância de sentido são sorteados no final de cada um deles. A heurística de escolha de força de tiro passa a levar em conta também o próprio nível de energia.

Com o resultado da batalha final da liga nacional, exposto na Tabela 18, a equipe BIRL, da Figura 17, foi a campeã da Liga Nacional de Robocode.

Tabela 18 – Final da liga nacional

Colocação	Equipe	Pontuação da rodada
1°	BIRL	1438
2°	Moto	701

Figura 17 – Campeões da Liga Nacional de Robocode 2016.



Da esquerda para a direita: Erik Vinicius Almeida, Rodrigo Correia e Matheus Magalhães.

4 Conclusão

Construir um robô competitivo para o Robocode não é uma tarefa trivial. Apesar disso, as estratégias apresentadas neste trabalho podem ser encontradas em comunidades e fóruns e conseguem produzir um resultado satisfatório.

Na etapa regional, foi evidente a disparidade entre o BIRL e os outros robôs nas rodadas classificatórias, que não utilizavam nenhuma das estratégias básicas ou avançadas: em todas as rodadas, a soma das pontuações dos outros robôs foi inferior à do BIRL. Na rodada final, mesmo utilizando estratégias já mais avançadas, o robô **Monstro**, batalhou de forma nivelada contra o robô **Terminator_T800**, com estratégias básicas.

Os competidores da etapa nacional, com exceção do BIRL, implementaram em seus robôs apenas estratégias básicas. Aliado a isso, o modo *melee*, utilizado durante as rodadas classificatórias, tornou as batalhas mais niveladas, sem grandes disparidades na pontuação. Entretanto, nas rodadas eliminatórias, em modo *1v1*, ficou evidente a diferença de performance das estratégias avançadas contra as básicas.

Observando a trajetória da equipe BIRL durante o processo da Liga Brasileira, pode-se concluir que as estratégias básicas e avançadas são eficazes na construção de robôs competitivos.

4.1 Desafios futuros

A fim de tornar o **Monstro** um robô ainda mais competitivo, pode-se:

- Adicionar um mecanismo de aprendizado aos fatores de risco utilizados na estratégia *Risk Evaluation*, a fim de calibrar os coeficientes de acordo com a situação de cada batalha;
- Implementar *Dynamic Clustering* no controle do canhão: uma técnica que projeta possíveis posições futuras do inimigo com base em experiências similares anteriores, e dispara em um ângulo que atinja o maior número de projeções possível.
- Pesquisar melhores estratégias para controle do radar em situações *melee*.

Referências

- BECKER, K. Teaching with games: The minesweeper and asteroids experience. *J. Comput. Sci. Coll.*, Consortium for Computing Sciences in Colleges, USA, v. 17, n. 2, p. 23–33, dez. 2001. ISSN 1937-4771. Disponível em: <<http://dl.acm.org/citation.cfm?id=775339.775347>>. Citado na página 13.
- CRESPO, F. N. L. L. *Robocode class Rules*. 2017. Disponível em: <<http://robocode.sourceforge.net/docs/robocode/robocode/Rules.html>>. Acesso em: 06 de janeiro de 2018. Citado na página 18.
- GEORGAS, J. C. Teams battling teams - introducing software engineering education in the first year with robocode. 2016. Citado na página 13.
- HARPER, R. Evolving robocode tanks for evo robocode. *Genetic Programming and Evolvable Machines*, v. 15, n. 4, p. 403–431, Dec 2014. ISSN 1573-7632. Disponível em: <<https://doi.org/10.1007/s10710-014-9224-2>>. Citado na página 13.
- HARTNESS, K. Robocode: Using games to teach artificial intelligence. *J. Comput. Sci. Coll.*, Consortium for Computing Sciences in Colleges, USA, v. 19, n. 4, p. 287–291, abr. 2004. ISSN 1937-4771. Disponível em: <<http://dl.acm.org/citation.cfm?id=1050231.1050275>>. Citado 2 vezes nas páginas 13 e 15.
- LAI, T.-w. et al. Research and implementation of robocode decision-making system based on jess and machine learning. *Journal of System Simulation*, p. S2, 2006. Citado na página 13.
- LEUTENEGGER, S.; EDGINGTON, J. A games first approach to teaching introductory programming. *SIGCSE Bull.*, ACM, New York, NY, USA, v. 39, n. 1, p. 115–118, mar. 2007. ISSN 0097-8418. Disponível em: <<http://doi.acm.org/10.1145/1227504.1227352>>. Citado na página 13.
- LIU, P. L. Using open-source robocode as a java programming assignment. *SIGCSE Bull.*, ACM, New York, NY, USA, v. 40, n. 4, p. 63–67, nov. 2008. ISSN 0097-8418. Disponível em: <<http://doi.acm.org/10.1145/1473195.1473222>>. Citado na página 13.
- MEIRA, M. C. Aprendizagem de linguagem de programação com metodologia pbl em competições científicas com robocode. 2016. Citado na página 13.
- NIDORF, D. G.; BARONE, L.; FRENCH, T. A comparative study of neat and xcs in robocode. In: *IEEE Congress on Evolutionary Computation*. [S.l.: s.n.], 2010. p. 1–8. ISSN 1089-778X. Citado na página 13.
- ROBOCODEBRASIL. *Liga Brasileira de Robocode*. 2016. Disponível em: <<http://www.robocodebrasil.com.br/>>. Acesso em: 05 de janeiro de 2018. Citado na página 46.
- ROBOWIKI. *Selecting Fire Power*. 2007. Disponível em: <http://robowiki.net/wiki/Selecting_Fire_Power>. Acesso em: 06 de janeiro de 2018. Citado na página 30.
- ROBOWIKI. *Linear Targeting*. 2012. Disponível em: <http://robowiki.net/wiki/Linear_Targeting>. Acesso em: 06 de janeiro de 2018. Citado na página 28.

- ROBOWIKI. *Wave Surfing*. 2012. Disponível em: <http://robowiki.net/wiki/Wave_surfing>. Acesso em: 06 de janeiro de 2018. Citado na página 38.
- ROBOWIKI. *Head-On Targeting*. 2013. Disponível em: <http://robowiki.net/wiki/Head-On_Targeting>. Acesso em: 06 de janeiro de 2018. Citado na página 27.
- ROBOWIKI. *Diamond*. 2017. Disponível em: <<http://robowiki.net/wiki/Diamond>>. Acesso em: 06 de janeiro de 2018. Citado na página 45.
- ROBOWIKI. *DrussGT*. 2017. Disponível em: <<http://robowiki.net/wiki/DrussGT>>. Acesso em: 06 de janeiro de 2018. Citado na página 45.
- ROBOWIKI. *GuessFactor Targeting (traditional)*. 2017. Disponível em: <http://robowiki.net/wiki/Guess_Factor_Targeting>. Acesso em: 06 de janeiro de 2018. Citado na página 33.
- ROBOWIKI. *Neuromancer*. 2017. Disponível em: <<http://robowiki.net/wiki/Neuromancer>>. Acesso em: 06 de janeiro de 2018. Citado na página 45.
- ROBOWIKI. *One on One Radar*. 2017. Disponível em: <http://robowiki.net/wiki/One_on_One_Radar>. Acesso em: 06 de janeiro de 2018. Citado 2 vezes nas páginas 24 e 25.
- ROBOWIKI. *Robocode/Game Physics*. 2017. Disponível em: <http://robowiki.net/wiki/Robocode/Game_Physics>. Acesso em: 06 de janeiro de 2018. Citado na página 18.
- ROBOWIKI. *Robocode/Robot Anatomy*. 2017. Disponível em: <http://robowiki.net/wiki/Robocode/Robot_Anatomy>. Acesso em: 06 de janeiro de 2018. Citado na página 17.
- ROBOWIKI. *Robocode/Scoring*. 2017. Disponível em: <<http://robowiki.net/wiki/Robocode/Scoring>>. Acesso em: 06 de janeiro de 2018. Citado na página 16.
- SHICHEL, Y.; ZISERMAN, E.; SIPPER, M. Gp-robocode: Using genetic programming to evolve robocode players. In: KEIJZER, M. et al. (Ed.). *Genetic Programming*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005. p. 143–154. ISBN 978-3-540-31989-4. Citado na página 13.
- WHITLEY, M. *Robocode Lesson #5: Movement Basics*. 2003. Disponível em: <<http://mark.random-article.com/weber/java/robocode/lesson5.html>>. Acesso em: 06 de janeiro de 2018. Citado na página 32.
- YOON, D. M.; KIM, K. J. Challenges and opportunities in game artificial intelligence education using angry birds. *IEEE Access*, v. 3, p. 793–804, 2015. Citado na página 13.
- ZAMBIASI, S. P. *Robocode - Eventos*. 2010. Disponível em: <<http://www.gsigma.ufsc.br/~popov/aulas/robocode/eventos.html>>. Acesso em: 06 de janeiro de 2018. Citado na página 19.