

UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ  
DIRETORIA DE PESQUISA E PÓS-GRADUAÇÃO  
PROGRAMA DE PÓS-GRADUAÇÃO EM COMPUTAÇÃO APLICADA (PPGCA)  
PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA E  
INFORMÁTICA INDUSTRIAL (CPGEI)

**AVANÇOS NA APLICAÇÃO DE FUTEBOL DE ROBÔS  
EM *FRAMEWORK* PON C++ 2.0 DO  
PARADIGMA ORIENTADO A NOTIFICAÇÕES – PON**

**Disciplinas:**

Tópicos Especiais em EC: Paradigma Orientado a Notificações - TEC0301 (CPGEI)  
Tópicos Avançados em Sistemas Embarcados 2 - CASE102 (PPGCA)

**Alunos:**

1. Anderson Eduardo de Lima
2. Felipe dos Santos Neves
3. Lucas Tachini Garcia
4. Luis Henrique Sant'Ana
5. Omero Francisco Bertol

CURITIBA  
2020

## LISTA DE FIGURAS

Figura 1 – Exemplo de uma <i>Rule</i> .....	8
Figura 2 – Oracle VM VirtualBox instalado .....	22
Figura 3 – Máquina virtual “Lubuntu+ROS_Kinetic” no Oracle VM Virtual Box .....	23
Figura 4 – Máquina virtual “Lubuntu+ROS_Kinetic” executando no Oracle VM Virtual Box .....	23
Figura 5 – Ambiente gráfico de simulação “grSim” em execução.....	24
Figura 6 – Futebol de Robôs rodando no ambiente gráfico de simulação “grSim” .....	24
Figura 7 – Robôs se movimentando após ações executadas no ambiente gráfico de simulação <i>Small Size League</i> .....	25
Figura 8 – Organização estrutural do projeto “RoboCup2012” .....	26
Figura 9 – Códigos-fontes do pacote “ControleF180” projeto “RoboCup2012”.....	27
Figura 10 – Diagrama de classes parcial do projeto “RoboCup2012” .....	29
Figura 11 – Softwares para partidas de Futebol de Robôs desenvolvido em Santos (2017) .....	48
Figura 12 – Diagrama de classes parcial do projeto “RoboCup2012” desenvolvido em Santos (2017).....	50
Figura 13 – Dimensão, em milímetros, do campo oficial da categoria <i>Small Size League</i> (SSL).....	51

## LISTA DE QUADROS

Quadro 1 – Exemplo da declaração de <i>Attributes e MethodPointers</i> .....	9
Quadro 2 – Exemplo da implementação de <i>MethodsPointers</i> .....	9
Quadro 3 – Exemplo da implementação de <i>Premises</i> .....	10
Quadro 4 – Exemplo da implementação de <i>Rules</i> .....	11
Quadro 5 – Conjunto de <i>Rules</i> implementadas na aplicação para controle do Futebol de Robôs.....	17
Quadro 6 – Principais códigos-fontes do pacote “ControleF180” projeto “RoboCup2012” .....	27
Quadro 7 – Conjunto de <i>Premises</i> do projeto “RoboCup2012” desenvolvido em Santos (2017) .....	31
Quadro 8 – Conjunto de <i>Instigations</i> do projeto “RoboCup2012” desenvolvido em Santos (2017).....	37
Quadro 9 – Conjunto de <i>Rules</i> do projeto “RoboCup2012” desenvolvido em Santos (2017) .....	39
Quadro 10 – Principais códigos-fontes do projeto “RoboCup2012” desenvolvido em Santos (2017).....	49
Quadro 11 – Partes relevantes do arquivo “StrategyPON.cpp” para a <i>Rule</i> “rlPassWhenOff”.....	68
Quadro 12 – <i>Method</i> “attackMove” presente nos diferentes códigos de acordo com o posicionamento do jogador.....	69
Quadro 13 – <i>Method</i> “defenceMove” presente nos diferentes códigos de acordo com o posicionamento do jogador.....	71
Quadro 14 – <i>Method</i> “moveToStopPosition”.....	73
Quadro 15 – Alterações relevantes no arquivo “StrategyPonDF.cpp” para as novas movimentações de goleiro .....	74

## SUMÁRIO

<b>1 INTRODUÇÃO .....</b>	<b>4</b>
1.1 CONTEXTUALIZAÇÃO .....	4
1.2 OBJETIVOS .....	4
1.3 ESTRUTURA DO TRABALHO .....	4
<b>2 PARADIGMA ORIENTADO A NOTIFICAÇÕES – PON.....</b>	<b>7</b>
2.1 ELEMENTOS FUNDAMENTAIS DO PON .....	7
2.2 ATTRIBUTES E METHODS .....	8
2.3 CONDITION E PREMISES .....	10
2.4 ACTION E INSTIGATIONS .....	10
2.5 RULES .....	11
2.6 FLUXO DE EXECUÇÃO NO PON .....	12
2.7 LINGUAGENS DE PROGRAMAÇÃO NO PON .....	12
<b>3 INFORMAÇÕES SOBRE O FUNCIONAMENTO DO FUTEBOL DE ROBÔS</b> <b>.....</b>	<b>13</b>
3.1 REQUISITOS FUNCIONAIS.....	13
3.2 REGRAS .....	14
3.3 NOTAS IMPORTANTES .....	15
<b>4 DESENVOLVIMENTO DA APLICAÇÃO PARA CONTROLE DO FUTEBOL</b> <b>DE ROBÔS.....</b>	<b>16</b>
4.1 ATIVIDADE PROPOSTA.....	16
4.2 DIFICULDADES ENCONTRADAS .....	16
4.3 TRABALHO DESENVOLVIDO .....	17
<b>5 CONCLUSÃO.....</b>	<b>20</b>
<b>REFERÊNCIAS .....</b>	<b>21</b>
<b>APÊNDICE A: UTILIZAÇÃO DO AMBIENTE DE FUTEBOL DE ROBÔS EM</b> <b>MÁQUINA VIRTUAL.....</b>	<b>22</b>
<b>APÊNDICE B: PROJETO “ROBOCUP2012” EM AMBIENTE DE SIMULAÇÃO</b> <b>.....</b>	<b>26</b>
<b>APÊNDICE C: PROJETO “ROBOCUP2012” DESENVOLVIDO EM SANTOS</b> <b>(2017) .....</b>	<b>30</b>
<b>APÊNDICE D: RULES IMPLEMENTADAS NO PROJETO “ROBOCUP2012”</b> <b>PELOS ALUNOS ANDERSON EDUARDO DE LIMA, FELIPE DOS SANTOS</b> <b>NEVES, LUCAS TACHINI GARCIA E OMERO FRANCISCO BERTOL.....</b>	<b>51</b>
<b>APÊNDICE E: RULES IMPLEMENTADAS NO PROJETO “ROBOCUP2012”</b> <b>PELO ALUNO LUIS HENRIQUE SANT’ANA .....</b>	<b>67</b>

## 1 INTRODUÇÃO

Nestas considerações iniciais serão apresentadas a contextualização do problema objeto deste trabalho acadêmico, os objetivos e a organização estrutural das seções que constituem este relatório técnico.

### 1.1 CONTEXTUALIZAÇÃO

Este trabalho tem por objetivo relatar o que foi colocado em prática no tocante aos tópicos abordados nas disciplinas referentes ao Paradigma Orientado a Notificações (PON) registradas como “Tópicos Especiais em EC: Paradigma Orientado a Notificações – TEC0301” no Programa de Pós-Graduação em Engenharia Elétrica e Informática Industrial (CPGEI) e como “Tópicos Avançados em Sistemas Embarcados 2 – CASE102” no Programa de Pós-Graduação em Computação Aplicada (PPGCA).

Para experimentar as técnicas explanadas nas disciplinas em questão em uma aplicação real, foi desenvolvido um projeto orientado a regras e baseado nos conceitos e propriedades do PON para um simulador de Futebol de Robôs (RoboCup) executado em máquina virtual.

O desenvolvimento deste estudo deve contemplar o regulamento de funcionamento de uma competição de Futebol de Robôs (RoboCup), um estudo teórico e prático dos elementos fundamentais da tecnologia PON, a configuração de um ambiente de simulação do Futebol de Robôs em máquina virtual (Apêndice A), a análise de *Regras (Rules)* implementadas em trabalhos existentes (Apêndices B e C) e o desenvolvimento no *Framework* PON C++ 2.0 de um conjunto de *Rules* para um software de controle de uma partida de Futebol de Robôs (Apêndices D e E).

### 1.2 OBJETIVOS

Com o objetivo de colocar em prática os conceitos vistos em classe no contexto das disciplinas supracitadas, foi proposto o desenvolvimento de um aplicativo de controle de robôs em tecnologia PON, em contexto de Futebol de Robôs, para execução em ambiente de simulação.

### 1.3 ESTRUTURA DO TRABALHO

Este relatório técnico está dividido em 5 seções e 5 apêndices. Nesta presente e primeira seção, nomeadamente “Introdução”, foi explicado o assunto tema do trabalho e também foram abordados a motivação, os objetivos e a estrutura geral do trabalho.

Subsequentemente, na segunda seção nomeadamente “Paradigma Orientado a Notificações – PON” será mostrada a estrutura do PON, seus elementos essenciais na forma de entidades *Atributos (Attributes)*, *Métodos (Methods)*, *Condição (Condition)*, *Premissas (Premises)*, *Ação (Action)*, *Instigações (Instigations)* e *Regras (Rules)*, o fluxo de execução no PON e a apresentação das linguagens de programação que materializam a tecnologia PON.

Por sua vez, na terceira seção, nomeadamente “Informações sobre o Funcionamento do Futebol de Robôs”, são apresentadas informações sobre as *Rules* de como funciona uma partida de Futebol de Robôs (RoboCup).

Notadamente, na quarta seção, nomeadamente “Desenvolvimento da Aplicação para Controle do Futebol de Robôs”, será apresentado o desenvolvimento do software de controle dos robôs visando a simulação de uma partida de Futebol de Robôs em ambiente de simulação, o qual foi disponibilizado em uma máquina virtual.

Por fim, na quinta e última seção, nomeadamente de “Conclusão”, serão apresentadas as considerações finais apontando as conclusões alcançadas com o desenvolvimento deste trabalho acadêmico.

Outrossim, no “Apêndice A: Utilização do Ambiente de Futebol de Robôs em Máquina Virtual”, será apresentado um tutorial com os passos para configuração da máquina virtual e a execução do ambiente de simulação do Futebol de Robôs.

Ainda, no “Apêndice B: Projeto RoboCup2012 em Ambiente de Simulação” será detalhado o projeto do aplicativo de controle do Futebol de Robôs “RoboCup2012” em tecnologia PON, para execução no ambiente de simulação e que foi disponibilizado para este estudo.

Por sua vez, o “Apêndice C: Projeto RoboCup2012 Desenvolvido em Santos (2017)”, será discutido o trabalho de Santos (2017) que realizou a proposição de uma nova versão da linguagem de programação específica para o Paradigma Orientado a Notificações (PON), nomeada LingPON. A versão 1.2 implementada da LingPON foi empregada no desenvolvimento de um software que controla partidas de Futebol de Robôs (RoboCup) e que foi então comparado com outros dois softwares equivalentes, a saber, desenvolvidos utilizando o *Framework* PON C++ 2.0 e a LingPON versão 1.0. Os softwares desenvolvidos em Santos (2017) nas diferentes linguagens de programação que materializam a tecnologia PON foram também disponibilizados para a realização deste estudo.

Em tempo, as *Rules* implementadas pelos discentes no Projeto “RoboCup2012” em ambiente de simulação serão apresentadas no Apêndice D as *Rules* implementadas pelos alunos Anderson Eduardo de Lima, Felipe dos Santos Neves, Lucas Tachini Garcia e Omero Francisco Bertol; e no Apêndice E as *Rules* implementas pelo aluno Luis Henrique Sant’Ana.

## 2 PARADIGMA ORIENTADO A NOTIFICAÇÕES – PON

Nesta segunda seção serão feitas as considerações iniciais sobre o Paradigma Orientado a Notificações (PON), seus elementos essenciais na forma de *Atributos* (*Attributes*), *Métodos* (*Methods*), *Condição* (*Condition*), *Premissas* (*Premises*), *Ação* (*Action*), *Instigações* (*Instigations*) e *Regras* (*Rules*), o fluxo de execução no PON e a apresentação das linguagens de programação que materializam a tecnologia PON.

### 2.1 ELEMENTOS FUNDAMENTAIS DO PON

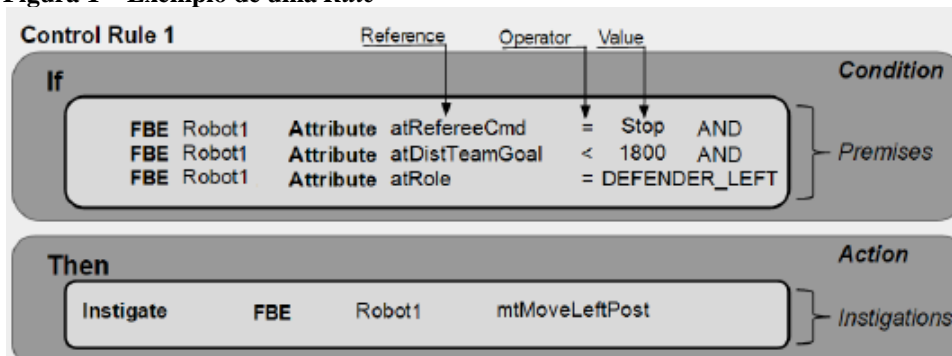
O Paradigma Orientado a Objeto (POO), classificado como subparadigma do Paradigma Imperativo, ou os Sistemas Baseados em Regras, englobados pelo Paradigma Declarativo, sofrem limitações intrínsecas de seus paradigmas (RONSZCKA, 2012, p. 27).

Esses paradigmas levam ao forte acoplamento em expressões lógico causais deste modo levando ao processamento desnecessário com redundâncias causais e/ou utilização custosa de estruturas de dados. Essas limitações com frequência comprometem o desempenho das aplicações. Nesse contexto, existem motivações para buscas de alternativas aos Paradigma Imperativo e Paradigma Declarativo, com o objetivo de diminuir ou eliminar as desvantagens desses paradigmas (RONSZCKA, 2012, p. 28).

Nesse contexto, o Paradigma Orientado a Notificações (PON) foi apresentado como uma alternativa. O PON pretende eliminar algumas deficiências dos paradigmas usuais em relação à avaliação de expressões causais desnecessárias e acopladas. Para tal, o PON faz uso de um mecanismo baseado no relacionamento de entidades computacionais que proporcionam um fluxo de execução de maneira reativa através de notificações precisas e pontuais (RONSZCKA, 2012, p. 28).

Na estrutura do Paradigma Orientado a Notificações (PON), as entidades computacionais que possuem *Atributos* (*Attributes*) e *Métodos* (*Methods*) são genericamente chamadas de *Fact Base Elements* (FBEs). Por meio de seus *Attributes* e *Methods*, as entidades FBEs são passíveis de correlação lógico-casual, normalmente proveniente de regras “se-então” (ou “If-Then”), por meio de entidades *Rules*, esquematizada na Figura 1, o que constituem elementos fundamentais do PON (SANTOS, 2017, p. 19; RONSZCKA, 2019, p. 28).

Figura 1 – Exemplo de uma *Rule*



Fonte: Santos (2017, p. 19).

Uma *Rule*, como mostra novamente a Figura 1, é decomposta em entidades *Condition* (*Condição*, trata da decisão da *Rule*) e *Action* (*Ação*, trata da execução das ações associadas a *Rule*). Por sua vez, a *Condition* é decomposta em uma ou mais entidades *Premises* (*Premissas*, realiza as verificações que definem a tomada de decisão) e uma *Action* é decomposta em uma ou mais entidades *Instigations* (*Instigações*, “instiga” *Methods* responsáveis por realizar serviços ou habilidades) (BANASZEWSKI, 2009, p. 10; SANTOS, 2017, p. 19; VALENÇA, 2012, p. 8-9; RONSZCKA, 2019, p. 28-29).

## 2.2 ATTRIBUTES E METHODS

Todo e qualquer recurso de uma aplicação expressa os estados ou valores de situações por meio de entidades chamadas de *Atributos* (*Attributes*), bem como disponibiliza seus serviços ou funcionalidades por meio de entidades chamadas de *Métodos* (*Methods*) (BANASZEWSKI, 2009, p. 9).

O conceito de *Atributos* (*Attributes*) e *Métodos* (*Methods*) representam uma evolução dos conceitos de *Atributos*, que definem a porção de dados; e *Métodos*, que implementam o comportamento, de classe do Paradigma Orientado a Objetos. A diferença, segundo Ronszcka (2012), está no desacoplamento implícito da classe proprietária e a colaboração pontual para com as entidades *Premises* e *Instigations*.

No contexto do desenvolvimento do aplicativo em tecnologia PON para controle de uma partida de Futebol de Robôs, tem-se como exemplo de *Attribute* PON o elemento “atClosestToGoal” (Quadro 1, Linha 4), implementado para definir se o jogador está próximo (*true*) ao gol ou não (*false*). Já no caso do *Attribute* “atPoxX” (Quadro 1, Linha 5), ele será utilizado para representar o valor da posição do robô em relação ao eixo “X” do gráfico cartesiano (corresponde a distância da linha de gol defendido, até a linha de gol atacado).



**Quadro 1 – Exemplo da declaração de *Attributes* e *MethodPointers***

```
// Declaração dos Attributes "atClosestToGoal" e "atPosX" na classe "RobotPON"
// no arquivo de cabeçalhos "RobotPON.h"
1. class RobotPON : public Robot, public FBE {
2. public:
3. ...
4. Boolean* atClosestToGoal;
5. Double* atPosX;
6. };
```

**Fonte: Autoria própria.**

Em tempo, também no contexto do aplicativo de controle de uma partida de Futebol de Robôs, tem-se como exemplos de *Methods* PON os elementos “*mtClosestToGoal*” e “*mtNotClosestToGoal*” (Quadro 2, Linhas 4-5). Estes elementos do tipo *MethodPointers* permitem que métodos implementadas em C++ sejam utilizadas diretamente como *Methods*. Eles são criados atrelando o método que modifica o estado de um jogador para indicar se ele está próximo ao gol (*Attribute* “*atClosestToGoal*” é setado com o valor *true*), em “*closestToGoal*” (Quadro 2, Linhas 7 e 15-17); ou não (*Attribute* “*atClosestToGoal*” é setado com o valor *false*), com o método “*notClosestToGoal*” (Quadro 2, Linhas 8 e 18-20).

**Quadro 2 – Exemplo da implementação de *MethodsPointers***

```
// Implementação das interfaces dos Methods na classe "StrategyPON"
// no arquivo de cabeçalhos "StrategyPON.h"
1. class StrategyPON : public Strategy, public FBE {
2. protected:
3. ...
4. MethodPointer<StrategyPON>* mtClosestToGoal;
5. MethodPointer<StrategyPON>* mtNotClosestToGoal;
6. ...
7. void closestToGoal();
8. void notClosestToGoal();
9. };

// Criação das Instigations "mtClosestToGoal" e "mtNotClosestToGoal"
// no arquivo de códigos-fontes StrategyPON.cpp Method que instância as Instigations
10. void StrategyPON::initMethodPointers() {
11. ...
12. mtClosestToGoal = new MethodPointer<StrategyPON>(this, &StrategyPON::closestToGoal);
13. mtNotClosestToGoal = new MethodPointer<StrategyPON>(this,
    &StrategyPON::notClosestToGoal);
14. }

// Implementação do comportamento dos Methods no arquivo de códigos-fontes StrategyPON.cpp
15. void StrategyPON::closestToGoal(){
16. this->robot->atClosestToGoal->setValue(true);
17. }

18. void StrategyPON::notClosestToGoal(){
19. this->robot->atClosestToGoal->setValue(false);
20. }
```

**Fonte: Autoria própria.**

## 2.3 CONDITION E PREMISES

As *Premissas (Premises)* que compõem a entidade *Condição (Condition)*, como apresentadas no exemplo de *Rules* ainda na Figura 1, são responsáveis pela realização das verificações que definem a tomada de decisão de uma *Rule*.

A implementação da tecnologia PON, no aplicativo de controle de uma partida de Futebol de Robôs, no que diz respeito as *Premises* está exemplificada no Quadro 3. No primeiro momento do exemplo, declara-se a *Premise* “prNotClosestToGoalX” como um ponteiro (\*) para objetos da classe “Premise” (Quadro 3, Linha 4). Na segunda etapa do processo no método “initPremises”, a *Premise* “prNotClosestToGoalX” é instanciada a partir da classe “Premise” no método “PREMISE” com o seguinte pseudocódigo da expressão lógica: *atPosX* <= 7985 (Quadro 3, Linha 8). Neste contexto, a *Premise* “prNotClosestToGoal” é responsável por verificar se um robô “não” está próximo a linha do gol atacado verificando se o estado/valor do *Attribute* “atPosX” (Quadro 1, Linha 5) é menor ou igual (SMALLERTHAN) ao valor *double* 7985 (*new Double(this, 7980)*).

**Quadro 3 – Exemplo da implementação de Premises**

```
// Declaração do Attribute “prClosestToGoal”, como um ponteiro para objetos da classe “Premise”,
// no arquivo de cabeçalhos “StrategyPON.h”
1. class StrategyPON : public Strategy, public FBE {
2. protected:
3. ...
4.   Premise *prNotClosestToGoalX;
5. };

// Method que instância as Premises: criação da Premise “prClosestToGoal”
6. void StrategyPON::initPremises() {
7. ...
8.   PREMISE(prNotClosestToGoalX, this->robot->atPosX, new Double(this, 7980),
        Premise::SMALLERTHAN, Premise::STANDARD, false);
9. }
```

**Fonte:** Autoria própria.

## 2.4 ACTION E INSTIGATIONS

As *Instigações (Instigations)*, que compõem a entidade *Ação (Action)*, como demonstradas no exemplo de *Rules* novamente na Figura 1, são responsáveis através de seus *Methods* pela realização dos serviços ou habilidades associadas a uma *Rule*.

A demonstração da forma de implementação das *Instigations*, novamente no aplicativo de controle da partida de Futebol de Robôs, pode ser observada no Quadro 2. A *Instigation* é criada e atrelada diretamente ao objeto *Rule* “rlNotClosestToGoalX” (Quadro 4, Linha 4) por meio do método INSTIGATION (Quadro 4, Linha 10). Esta

*Instigation* instigará o *Method* “mtNotClosestToGoal” (Quadro 2, Linha 5).

## 2.5 RULES

Uma *Rule* é decomposta em entidades *Condition* (condição, conjunto de *Premises*) e *Action* (ação, conjunto de *Instigations*), como pode ser observado ainda na Figura 1. A *Condition* é formada pelas *Premises* associadas à *Rule* de modo que a ativação delas define a execução ou não da *Action*. Cada *Rule* desempenha o papel de conjunção ou disjunção unicamente para todas as *Conditions* em que está atrelada.

Uma das *Rules* implementada no aplicativo de controle da partida de Futebol de Robôs (Apêndice D), pode ser observada no Quadro 4. Primeiramente, declaram-se a *Rule* “rlNotClosestToGoalX” como um ponteiro (\*) para objetos da classe “RuleObject” (Quadro 4, Linha 4). A seguir, na próxima etapa do processo no *Method* “initRules”, cria-se a *Rule* com 3 (três) linhas de implementações: a) *Rule* “rlNotClosestToGoalX” é instanciada a partir da classe “RuleObject” no *Method* “RULE” (Quadro 4, Linha 8); b) adiciona-se a *Premise* “prNotClosestToGoalX” (Quadro 3, Linha 4) na *Rule* “rlNotClosestToGoalX” (Quadro 4, Linha 9); e c) adiciona-se a *Instigation* criada a partir do *MethodPointer* “mtNotClosestToGoal” com o método INSTIGATION na *Rule* “rlNotClosestToGoalX” (Quadro 4, Linha 10).

### Quadro 4 – Exemplo da implementação de Rules

```
// Declaração da Rule “rlNotClosestToGoalX”, como um ponteiro para objetos da classe “RuleObject”
// no arquivo de cabeçalhos StrategyPON.h
1. class StrategyPON : public Strategy, public FBE {
2.   protected:
3.     ...
4.     RuleObject* rlNotClosestToGoalX;
5. };

// Method que instância as Rules: criação da Rule “rlNotClosestToGoalX”
6. void StrategyPON::initRules(){
7.   ...
8.   RULE(rlNotClosestToGoalX, scheduler, Condition::SINGLE);
9.   rlNotClosestToGoalX->addPremise(prNotClosestToGoalX);
10.  rlNotClosestToGoalX->addInstigation(INSTIGATION(this->mtNotClosestToGoal));
11. }
```

Fonte: Autoria própria.

A *Rule* “rlNotClosestToGoalX”, implementada novamente no Quadro 4, se comporta de forma a determinar se o robô “não” está próximo do gol em relação a coordenada “x”, valor do *Attribute* “atPosX” (Quadro 1, Linha 5), expressão lógica implementada na *Premise* “prNotClosestToGoalX” (Quadro 3, Linha 4). Se não está próximo ao gol, então define o estado do *Attribute* “atClosestToGoal” (Quadro 1, Linha

4) como *false*, comportamento implementação no *Method* “mtNotClosestToGoal” (Quadro 2, Linha 11).

## 2.6 FLUXO DE EXECUÇÃO NO PON

No Paradigma Orientado a Notificações (PON) o fluxo de execução de uma aplicação inicia-se a partir da mudança de estado de um *Attribute* que “notifica” todas as *Premises* pertinentes, a fim de que estas reavaliem seus estados lógicos. Caso o valor lógico de uma *Premise* se altere, a *Premise* colabora com a avaliação lógica de uma ou de um conjunto de *Conditions* conectadas, o que ocorre por meio da “notificação” sobre a mudança relacionada ao seu estado lógico (VALENÇA, 2012, p. 29; RONSZCKA, 2019, p. 29; BANASZEWSKI, 2009, p. 11).

Consequentemente, cada *Condition* notificada avalia o seu valor lógico de acordo com as notificações recebidas das *Premises* e com um dado operador lógico, sendo este em geral um operador de conjunção (*and*) ou disjunção (*or*). Assim, no caso de uma conjunção, por exemplo, quando todas as *Premises* que integram uma *Condition* são satisfeitas, a *Condition* também é satisfeita. Isto resulta na aprovação de sua respectiva *Rule* que pode “então” ser executada (*Action*) (VALENÇA, 2012, p. 29).

Ainda segundo Valença (2012, p. 29), com a *Rule* aprovada a sua respectiva *Action* é ativada. Uma *Action*, por sua vez, é conectada a um ou vários *Instigations*. As *Instigations* colaboram com as atividades das *Actions*, acionando a execução de algum serviço por meio dos seus *Methods*. Usualmente, os *Methods* alteram os estados dos *Attributes*, recomeçando assim o ciclo de notificações.

## 2.7 LINGUAGENS DE PROGRAMAÇÃO NO PON

Santos (2017, p. 21) relata que os conceitos do PON foram primeiramente materializados no Paradigma Orientado a Objetos, através de um *framework* desenvolvido na Linguagem de Programação C++ em 2007 pelo Prof. Jean Marcelo Simão e que teve posteriormente a versão 1.0 desenvolvida no trabalho de Banaszewski (2009). Em 2012, a nova versão nomeada de *Framework* PON C++ 2.0 foi implementada nos trabalhos de Valença (2012) e Ronszcka (2012).

No caso da categoria de linguagem de programação específica para o PON, nomeada de LingPON, a versão 1.0 com o respectivo compilador foi desenvolvida e apresentada no projeto de mestrado de Ferreira (2015). Posteriormente, o trabalho de Santos (2017) propôs a LingPON versão 1.2.

### 3 INFORMAÇÕES SOBRE O FUNCIONAMENTO DO FUTEBOL DE ROBÔS

Nesta segunda seção serão apresentadas informações de como uma partida de Futebol de Robôs é jogada. Tais informações são necessárias para que haja o entendimento de *Rules* implementadas.

#### 3.1 REQUISITOS FUNCIONAIS

Aqui, estão apresentados requisitos funcionais baseados na regulamentação atual da competição de Futebol de Robôs (RoboCup), documento “Rules of the RoboCup Small Size League” (RULES, 2020), a fim de simplesmente garantir uma partida dentro desta regulamentação. São elas:

- a) Quando em “fora de jogo”, a bola não pode ser manipulada, ou seja, um jogador não pode realizar *dribbling* ou *shooting*. *Dribbling* é o ato manipular a bola de maneira geral, podendo mover-se com ela.
- b) Não é permitido executar *dribbling* por mais de 1m, ou seja, manter contato com a bola (segurá-la) por mais de um metro.
- c) Após cobranças, o jogador que cobrará não pode encostar na bola após ela ter se movido 50mm (o que torna a bola em jogo) sem que outro jogador a tenha tocado após esse movimento. Essa infração é chamada de toque duplo e possui uma tolerância equivalente a movimentação para a bola estar em jogo, há uma tolerância de 50mm para o toque duplo. Ou seja, o jogador deve executar passe ou chute quando em cobranças.
- d) Quando em *stop*, os robôs devem se movimentar abaixo de 1,5m/s, devendo manter-se a 500mm de distância da bola, não podendo manipulá-la, e devem se posicionar a 200mm da área de defesa do oponente. Há uma tolerância de 2 segundos.
- e) Quando em *halt*, nenhum robô pode mover-se ou manipular a bola. Há uma tolerância de 2 segundos.
- f) Em *direct kicks* pode ser realizado gol com o chute. Os jogadores defensores devem se posicionar a pelo menos 500mm da bola e os jogadores atacantes podem se posicionar próximos à bola.
- g) O *indirect kick* possui regras semelhantes ao *direct kick*, mas não pode ser realizado gol com o chute (pelo menos um jogador atacante que não o *kicker*, após a cobrança, deve tocar na bola antes da validade de um gol).

- h) Após o comando *force start*, ambos os times podem manipular a bola.
- i) Após o comando de pênalti, o goleiro defensor deve tocar a linha do gol e um jogador atacante pode se aproximar da bola e será o cobrador. Todos os outros robôs devem se posicionar a uma distância mínima demarcada por uma linha a 400mm da marca de pênalti, paralela à linha do gol.
- j) Após o comando *normal start*, deve ser avaliado também se o estado anterior era pênalti, para execução de chute do cobrador.
- k) Após o comando *normal start*, deve ser avaliado também se o estado anterior era *kick-off*, para execução de chute ou passe do cobrador.
- l) De mesmo modo, observa-se que apenas o goleiro pode entrar na sua área de defesa.
- m) Um robô não pode executar “chute sem mira”.
- n) Um robô não pode chutar a bola de forma que esta ultrapasse a velocidade de 6,5m/s.
- o) Um robô não deve ir de encontro direto a outro robô para evitar “colisão” e “empurrão”.
- p) Quando em chute a gol, a bola não pode ser elevada mais do que 150mm, ou o gol é declarado como inválido.
- q) Quando a bola ultrapassa as linhas de campo, há cobranças como tiro de meta ou laterais.

### 3.2 REGRAS

A seguir estão apresentadas *Regras (Rules)*, *Premissas (Premises)* e *Métodos (Methods)* necessários para que o funcionamento dos robôs siga de acordo com o estipulado pelos requisitos apresentados na seção “2.1 Requisitos Funcionais”. São elas:

- a) Premissa que avalia se o comando atual é *normal start*, se o anterior era pênalti, se o jogador é cobrador e método que realiza chute.
- b) Limitar a velocidade de chute ou passe nos respectivos métodos.
- c) Impedir avanço sobre robô adversário, especialmente se este está se deslocando na direção do robô em questão. Limitação em premissa ou método de movimentação.
- d) Limitação no método de chute.
- e) Não há razão para um robô posicionar-se além das linhas de campo.

### 3.3 NOTAS IMPORTANTES

Não é necessário haver um goleiro segundo o documento “Rules of the RoboCup Small Size League”, nas seções “3.1 Number Of Robots” e “4.3.5 Choosing Keeper Id” (RULES, 2020).

Pode-se segurar a bola, desde que não haja restrição a todos os seus graus de liberdade, pelo menos 80% da área da bola (RULES, 2020) deve ser visível quando em vista superior, não eleve a bola do chão e seja possível que outro robô adquira a posse de bola. Uma forma de implementação, portanto, pode-se ser exercer força limitada na bola em direção ao robô com posse.

## 4 DESENVOLVIMENTO DA APLICAÇÃO PARA CONTROLE DO FUTEBOL DE ROBÔS

Nesta quarta seção, será feita a apresentação da proposta de trabalho de conclusão de disciplina que envolve o desenvolvimento de *Rules* no software de controle dos robôs disponibilizado para estudo (Apêndice B) visando a simulação de uma partida de Futebol de Robôs em ambiente de simulação.

### 4.1 ATIVIDADE PROPOSTA

A atividade proposta durante a disciplina era de realizar a implementação de um novo conjunto de *Rules* com base no trabalho desenvolvido na dissertação do então mestrando Leonardo Araujo Santos (SANTOS, 2017), afim de melhorar o comportamento dos robôs.

### 4.2 DIFICULDADES ENCONTRADAS

Durante o desenvolvimento do trabalho foram encontradas diversas dificuldades, principalmente no que se refere ao ambiente proposto.

No uso da máquina virtual disponibilizada (Apêndice A), em computadores com processador AMD ocorriam algumas falhas de execução, e foi necessário o *download* do código fonte e a recompilação da aplicação “grSim”. Esse problema foi causado pelo fato das bibliotecas já virem pré-compiladas, e ao utilizar um processador diferente as mesmas eventualmente vem a apresentar falhas.

Além desse problema inicial também foi constatado que a versão de código disponibilizado neste ambiente não correspondia à versão final da dissertação, mas sim uma versão mais antiga.

Por conta disso inicialmente foi feita uma tentativa de atualizar o código (Apêndice B) com as novas *Rules*, utilizando um outro código disponibilizado da versão final de Santos (2017) (Apêndice C), porém isso se mostrou bastante complicado pois havia uma diferença muito grande entre a estrutura base da interface dos robôs sob a qual essas versões estavam implementadas, conforme a regulamentação de competição de Futebol de Robôs (RoboCup) descrito no capítulo 3, portanto ao tentar compilar e executar o código resultava em muitos erros de compilação, que quando resolvidos apenas geravam erros de execução (*core dump*). Esses erros podem ter sido causados devido a diferenças nas bibliotecas sob as quais os mesmos foram implementados.



Não obtendo sucesso foi observado que não seria muito produtivo prosseguir nessa tentativa de recompilar o código. Desta forma o que se seguiu foi uma tentativa de se adaptar as *Rules* propostas por Santos (2017) (Apêndice C), no código funcional, porém mais uma vez a tarefa se mostrou complicada, devido estrutura geral do código e falta de algumas partes, tornou-se impossível a mesclagem dos códigos, funcional e de Santos (2017) (Apêndice C).

#### 4.3 TRABALHO DESENVOLVIDO

Considerando as dificuldades apresentadas foi decidido finalmente que cada um dos alunos ficaria responsável pela implementação de 5 *Rules* próprias (Apêndices D e E), de modo a exercitar melhor os conhecimentos individuais sobre o PON e também obter como resultado um código funcional no qual seria possível observar a aplicação das *Rules* implementadas durante uma partida do Futebol de Robôs.

O Quadro 5 apresenta as *Rules* implementadas pelos integrantes do trabalho. Neste quadro é descrito o nome da variável adicionada ao código para novas *Rules*, uma descrição de funcionamento dela e seu responsável. As *Rules* têm papel de modificar o comportamento dos jogadores durante a partida.

**Quadro 5 – Conjunto de *Rules* implementadas na aplicação para controle do Futebol de Robôs**  
(continua)

<i>Rule</i>	<b>Objetivo</b>	<b>Responsável</b>
rlPositionToShoot	Posicionar jogador para chutar ao gol quando estiver próximo.  Verifica <i>Attributes</i> de estado do jogo e executa o <i>Method</i> que seta <i>flag</i> “atReadyToShoot” para <i>true</i> .	Felipe dos Santos Neves (Apêndice D)
rlShootToGoal	Chutar ao gol após estar posicionado.  Verifica <i>Attributes</i> de controle de chute a gol e executa o <i>Method</i> que produz a ação de chute “mtShootToGoal”.	Felipe dos Santos Neves (Apêndice D)
rlResetShootFlag	Resetar <i>flags</i> de controle após realizar chute.  Verifica os mesmos <i>Attributes</i> de controle de chute e também posse de bola. O chute é considerado feito quando “atBallIsMine” é <i>false</i> . Então as <i>flags</i> de controle de chute são atribuídas como <i>false</i> .	Felipe dos Santos Neves (Apêndice D)
rlMaxDrible	Passar a bola após exceder determinado número de passes.  O <i>Attribute</i> “atDribleCount” é incrementado a cada toque e “atMaxDribleCount” é randomizado entre 1 e 6 a cada execução da <i>Rule</i> . O <i>Method</i> força um passe e o <i>reset</i> de “atDribleCount”.	Felipe dos Santos Neves (Apêndice D)

**Quadro 5 – Conjunto de *Rules* implementadas na aplicação para controle do Futebol de Robôs**  
(continua)

<b>Rules</b>	<b>Objetivo</b>	<b>Responsável</b>
rlPassGkCloseGoal	Forçar goleiro a passar a bola ao ficar com a posse dela muito próximo ao gol.  Verifica se a distância ao gol “atBallDistTeamGoal” é menor que 1800 e outros <i>Attributes</i> de controle de posse de bola. Força o passe a outro robô. Só é aplicada em robô do tipo defesa.	Felipe dos Santos Neves (Apêndice D)
rlPenalty	Posicionar o jogador atacante para a posição de pênalti do campo.  Tratamento para movimentar o jogador atacante para a posição de pênalti. Posição calculada com base nas dimensões do campo. O estímulo desta <i>Rule</i> é realizado pelo painel <i>Referee Box</i> no botão <i>Penalty</i>	Anderson Eduardo de Lima (Apêndice D)
rlYellowCardBlue	Captura do evento de cartão amarelo do time azul.  Realiza o tratamento do estímulo do painel <i>Referee Box</i> no botão <i>Yellow Card</i> para os jogadores do time azul.	Anderson Eduardo de Lima (Apêndice D)
rlRedCardBlue	Captura do evento de cartão vermelho do time azul.  Realiza o tratamento do estímulo do painel <i>Referee Box</i> no botão <i>Red Card</i> para os jogadores do time azul.	Anderson Eduardo de Lima (Apêndice D)
rlLimiteHorizontal	Evitar a saída de robôs do limite horizontal do campo (regra da categoria <i>Small Size League - SSL</i> ).  Verifica se o robô encontra-se fora do limite horizontal, pré-estabelecido, do campo.	Anderson Eduardo de Lima (Apêndice D)
rlLimiteVertical	Evitar a saída de robôs do limite vertical do campo (regra do SSL).  Verifica se o robô encontra-se fora do limite vertical, pré-estabelecido, do campo.	Anderson Eduardo de Lima (Apêndice D)
rlClosestToGoal	Determinar se o robô está próximo do gol.  Se está próximo ao gol (coordenadas “x” e “y”) então define o estado “atClosestToGoal” como <i>true</i> .	Omero Francisco Bertol (Apêndice D)
rlNotClosestToGoalX	Determinar se o robô “não” está próximo do gol em relação a coordenada “x”.  Se não está próximo ao gol na coordenada “x” então define o estado “atClosestToGoal” como <i>false</i> .	Omero Francisco Bertol (Apêndice D)
rlBallFar	Reposicionar o robô.  Redefinir chute e drible quando a bola estiver longe. Baseada na “Rule 5” de Santos (2017, p. 179).	Omero Francisco Bertol (Apêndice D)
rlNotClosestToGoalY	Determinar se o robô “não” está próximo do gol em relação a coordenada “y”.  Se não está próximo ao gol em uma das coordenadas “y” então define o estado “atClosestToGoal” como <i>false</i> .	Omero Francisco Bertol (Apêndice D)

**Quadro 5 – Conjunto de *Rules* implementadas na aplicação para controle do Futebol de Robôs (conclusão)**

<b>Rule</b>	<b>Objetivo</b>	<b>Responsável</b>
rlClosestToGoalKick	Determinar se o robô deverá realizar o chute a gol.  Se o jogador está pronto, está com a bola e está próximo ao gol então chuta para o gol.	Omero Francisco Bertol (Apêndice D)
rlRobotWantsToMoveX	Avalia necessidade de movimento em X.	Lucas Tachini Garcia (Apêndice D)
rlRobotWantsToMoveY	Avalia necessidade de movimento em Y.	Lucas Tachini Garcia (Apêndice D)
rlRobotWantsToMove	Confirma a necessidade de movimento em X ou Y.	Lucas Tachini Garcia (Apêndice D)
rlRobotNotRetrained	Esta <i>Rule</i> deve agrupar condições negadas que possam vir a restringir o movimento do jogador.	Lucas Tachini Garcia (Apêndice D)
rlRobotMoveGoalKeeper	Permite ou não que o movimento do goleiro seja executado.	Lucas Tachini Garcia (Apêndice D)
rlNextMoveToRestrictedArea	Avalia movimentação para áreas não permitidas.	Lucas Tachini Garcia (Apêndice D)
rlFixRobotNotGkmove	Modifica o movimento do jogador que não seja o goleiro.	Lucas Tachini Garcia (Apêndice D)
rlFixRobotGkmove	Modifica o movimento do goleiro.	Lucas Tachini Garcia (Apêndice D)
rlRedCardYellow	Aplica cartão vermelho para um jogador do time amarelo.	Lucas Tachini Garcia (Apêndice D)
rlBallOutsideEnemyTeam	Se a bola sai da área do jogo no campo adversário, força o jogador a voltar para sua posição inicial.	Lucas Tachini Garcia (Apêndice D)
rlPassWhenOff	Determinar se o robô deve passar a bola e retornar ao centro do campo.  Jogador com a bola, avançado no campo adversário e fora de centro, passa a bola e retorna ao centro.	Luis Henrique Sant’Ana (Apêndice D)

**Fonte: Autoria própria.**

A finalidade das *Rules* criadas foi presenciar a modificação no comportamento dos jogadores durante simulação causada pela adição das instruções PON. Como muitas das funcionalidades descritas no projeto “RoboCup2012” desenvolvido em Santos (2017) estavam faltando algumas partes do código, desta forma impossibilitando a compilação, algumas destas novas funcionalidades não estão adequadas às regras vigentes para uma partida de Futebol de Robôs Oficial (RoboCup).

## 5 CONCLUSÃO

Com o objetivo de aplicar na prática os temas abordados nas disciplinas com enfoque no Paradigma Orientado a Notificações (PON) foi proposto a aplicação da tecnologia PON, configuração do ambiente de simulação (Apêndice A), análise de *Rules* em trabalhos existentes (Apêndices B e C) e o desenvolvimento de novas *Rules* do software para o simulador visando controle do Futebol de Robôs (Apêndices D e E).

Notadamente, foram levantadas informações sobre o regulamento da competição do Futebol de Robôs (RoboCup), como é realizada a condução de uma partida, *Rules*, *Premises* e *Methods* necessários para que o funcionamento de um jogador robô siga em acordo com os requisitos funcionais estabelecidos no regulamento.

Subsequentemente, realizou-se a apresentação do Paradigma Orientado a Notificações (PON) como uma alternativa para outros paradigmas de programação, entre eles, o Paradigma Imperativo e Paradigma Declarativo. O estudo da tecnologia PON mostrou seus elementos fundamentais materializados na forma de seus elementos essenciais na forma de *Atributos* (*Attributes*), *Métodos* (*Methods*), *Condição* (*Condition*), *Premissas* (*Premises*), *Ação* (*Action*), *Instigações* (*Instigations*) e *Regras* (*Rules*). O fluxo de execução de aplicações no PON foi destacado mostrando o desacoplamento em expressões lógico-causais, inexistente em outros paradigmas, o que evita o processamento desnecessário com redundâncias causais e/ou utilização custosa de estruturas de dados. Importância também foi dada as linguagens de programação que materializam a tecnologia PON, são elas: a) *Framework* PON C++ 2.0, desenvolvido na Linguagem de Programação C++; e b) LingPON, linguagem específica para o PON.

Por fim, foi apresentado o desenvolvimento de novas *Rules*, com base no trabalho de Santos (2017), afim de melhorar o comportamento dos jogadores no aplicativo para controle de Futebol de Robôs, executado em ambiente de simulação configurado em máquina virtual. Tarefa essa que apresentou diversas dificuldades, principalmente, no que se refere ao ambiente proposto como por exemplo, na questão das bibliotecas já estarem pré-compiladas e se mostraram incompatíveis com processadores diferentes provocando falhas na sua utilização. A nível de códifos-fontes a versão disponibilizada para estudo não correspondia à versão final de Santos (2017), mas sim uma versão anterior. Considerando as dificuldades, foram então apresentadas as *Rules* que cada um dos alunos implementou aplicando os conhecimentos individuais adquiridos sobre a tecnologia PON.

## REFERÊNCIAS

BANASZEWSKI, Roni Fabio. **Paradigma Orientado a Notificações: Avanços e comparações**. 2009. 285 f. Dissertação de Mestrado – Programa de Pós-graduação em Engenharia Elétrica e Informática Industrial (CPGEI), Universidade Tecnológica Federal do Paraná (UTFPR). Curitiba, 2009. Disponível em: <<http://livros01.livrosgratis.com.br/cp087456.pdf>>. Acesso em: 16 fev. 2020.

FERREIRA, Cleverson Avelino. **Linguagem e compilador para o paradigma orientado a notificações (PON): avanços e comparações**. 2015. 227 f. Dissertação (Mestrado em Computação Aplicada) – Universidade Tecnológica Federal do Paraná, Curitiba, 2015. Disponível em: <<http://repositorio.utfpr.edu.br/jspui/handle/1/1414>>. Acesso em: 22 mai. 2020.

RONSZCKA, Adriano Francisco. **Contribuição para a concepção de aplicações no paradigma orientado a notificações (PON) sob o viés de padrões**. 2012. 224 f. Dissertação de Mestrado – Programa de Pós-Graduação em Engenharia Elétrica e Informática Industrial (CPGEI), Universidade Tecnológica Federal do Paraná (UTFPR). Disponível em: <[http://repositorio.utfpr.edu.br/jspui/bitstream/1/327/1/CT\\_CPGEI\\_M\\_%20Ronszcka,%20Adriano%20Francisco\\_2012.pdf](http://repositorio.utfpr.edu.br/jspui/bitstream/1/327/1/CT_CPGEI_M_%20Ronszcka,%20Adriano%20Francisco_2012.pdf)>. Acesso em: 23 mar. 2020.

RONSZCKA, Adriano Francisco. **Método para a criação de linguagens de programação e compiladores para o Paradigma Orientado a Notificações em plataformas distintas**. 2019. 375 f. Tese de Doutorado – Programa de Pós-graduação em Engenharia Elétrica e Informática Industrial (CPGEI), Universidade Tecnológica Federal do Paraná (UTFPR). Curitiba, 2019. Disponível em: <<http://repositorio.utfpr.edu.br/jspui/handle/1/4234>>. Acesso em: 25 fev. 2020.

RULES of the RoboCup Small Size League 2019. **RoboCup Federation**, 19 mar. 2020. Disponível em: <<https://robocup-ssl.github.io/ssl-rules/sslrules.html>>. Acesso em: 08 fev. 2020.

SANTOS, Leonardo Araujo. **Linguagem e compilador para o paradigma orientado a notificações: Avanços para facilitar a codificação e sua validação em uma aplicação de controle de Futebol de Robôs**. 2017. 293 f. Dissertação de Mestrado – Programa de Pós-graduação em Engenharia Elétrica e Informática Industrial (CPGEI), Universidade Tecnológica Federal do Paraná (UTFPR). Curitiba, 2017. Disponível em: <<http://repositorio.utfpr.edu.br/jspui/handle/1/2778>>. Acesso em: 16 fev. 2020.

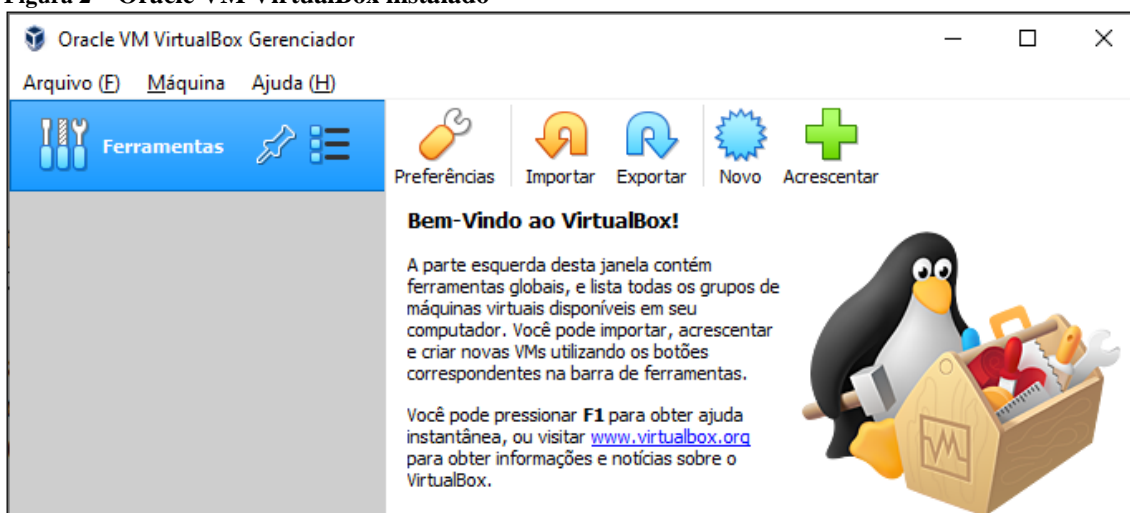
VALENÇA, Glauber Zárate. **Contribuição para a materialização do Paradigma Orientado a Notificações (PON) via framework e wizard**. 2012. 205 f. Dissertação (Mestrado em Computação Aplicada) – Programa de Pós-graduação em Computação Aplicada (PPGCA), Universidade Tecnológica Federal do Paraná (UTFPR). Curitiba, 2012. Disponível em: <<http://repositorio.utfpr.edu.br:8080/jspui/handle/1/393>>. Acesso em: 16 fev. 2020.

## APÊNDICE A: UTILIZAÇÃO DO AMBIENTE DE FUTEBOL DE ROBÔS EM MÁQUINA VIRTUAL

O procedimento necessário para utilização do ambiente de Futebol de Robôs em máquina virtual é detalhado a seguir e tem como base um tutorial fornecido pelo **Prof. Dr. João Alberto Fabro** (fabro@dainf.ct.utfpr.edu.br). Os 5 (cinco) passos deste procedimento estão aqui resumidos:

1. Instalar o VirtualBox<sup>1</sup>: VirtualBox 6.1.2 platform packages (Windows host). Disponível em: <<https://www.virtualbox.org/wiki/Downloads>>. Acesso em: 15 jan. 2020. Executar o instalador: VirtualBox-6.1.2-135662-Win, como administrador. O resultado da instalação do Oracle VM VirtualBox pode ser observado na Figura 2.

Figura 2 – Oracle VM VirtualBox instalado



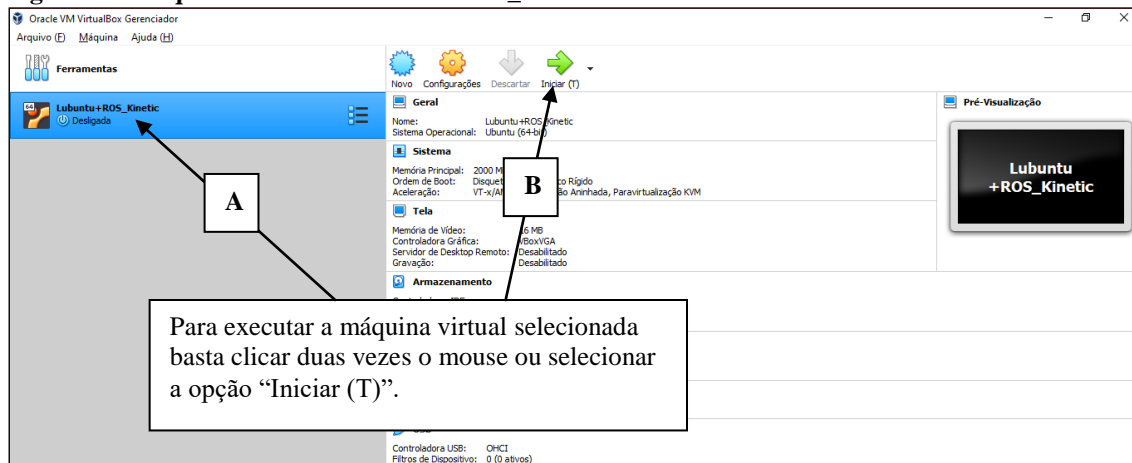
Fonte: Autoria própria.

2. Baixar o arquivo “LubuntuSimao.ova”<sup>2</sup>, este procedimento poderá demorar alguns minutos. Disponível em: <[www.dainf.ct.utfpr.edu.br/~fabro/download/LubuntuSimao.ova](http://www.dainf.ct.utfpr.edu.br/~fabro/download/LubuntuSimao.ova)>. Acesso em: 15 jan. 2020.
3. Na barra de menu do Oracle VM VirtualBox escolher a opção “Arquivo (F) | Importar Appliance...” e selecionar (📁) o arquivo “LubuntuSimao.ova” baixado no passo “2”. Como resultado da execução tem-se a criação da máquina virtual “Lubuntu+ROS\_Kinetic”, apresentada na Figura 3.

<sup>1</sup> O **VirtualBox** é um programa de virtualização que permite instalar e executar diferentes sistemas operacionais em um único computador. Fonte: **Oracle VM VirtualBox**. Disponível em: <<https://www.virtualbox.org/>>. Acesso em: 15 jan. 2020.

<sup>2</sup> A extensão de arquivo “**ova**” indica que se trata de um arquivo que possui descrições/informações de uma máquina virtual.

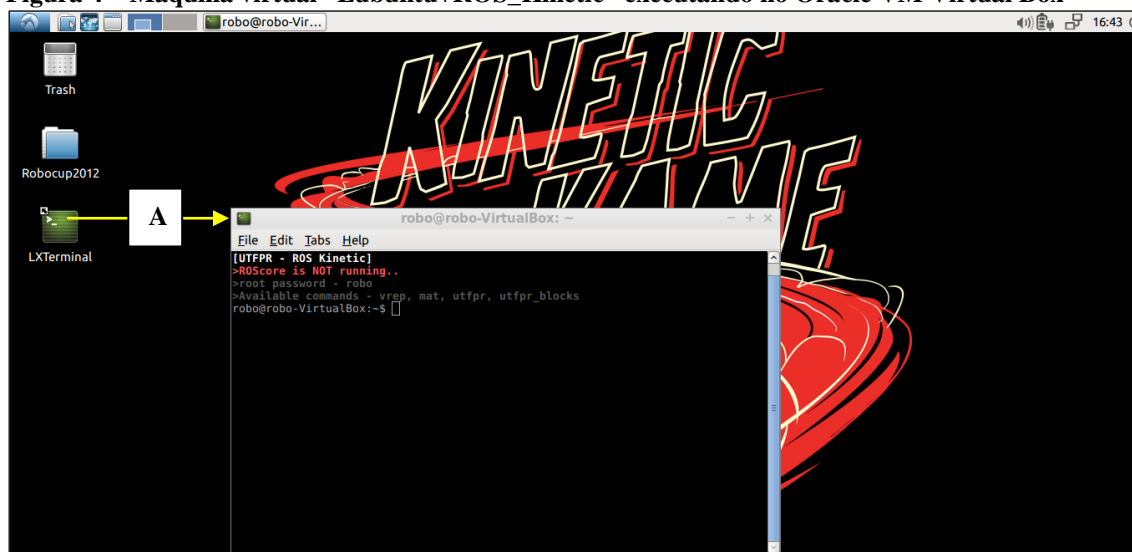
**Figura 3 – Máquina virtual “Lubuntu+ROS\_Kinetic” no Oracle VM Virtual Box**



Fonte: Autoria própria.

4. Executar a máquina virtual “Lubuntu+ROS\_Kinetic” no Oracle VM VirtualBox clicando duas vezes o mouse sobre a máquina – item A na Figura 3, ou ainda, escolhendo a opção “Iniciar (T)” – item B na Figura 3. A máquina “Lubuntu+ROS\_Kinetic” executando é apresentada na Figura 4.

**Figura 4 – Máquina virtual “Lubuntu+ROS\_Kinetic” executando no Oracle VM Virtual Box**



Fonte: Autoria própria.

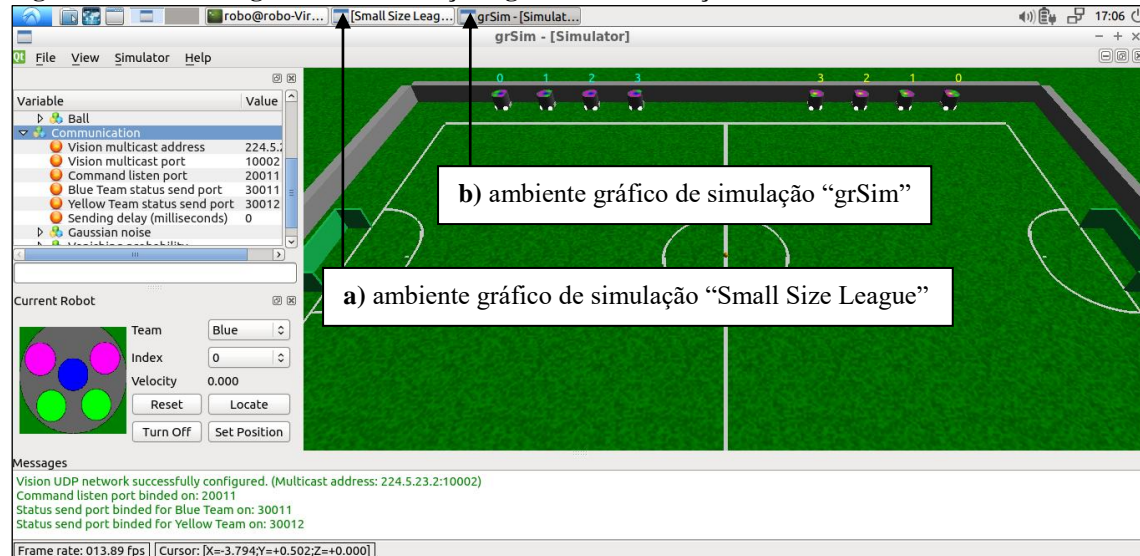
5. Após a execução da máquina virtual “Lubuntu+ROS\_Kinetic” no Oracle VM VirtualBox, para começar o Futebol de Robôs basta implementar os seguintes comandos no terminal<sup>3</sup>:
  - a. \$ cd Desktop/ para entrar na pasta “Desktop”
  - b. \$ cd Robocup2012/ para entrar na pasta “Robocup2012”

<sup>3</sup> Para abrir o terminal basta escolher a opção “LXTerminal” – item A na Figura 4.

c. `$ bash futebol.sh`

o comando “bash” é utilizado para executar a sequência de comandos do arquivo “futebol.sh”. Os comandos implementados neste arquivo executam os ambientes gráficos de simulação: a) *Small Size League* (uma das ligas do Futebol de Robôs); e b) grSim (usado para testar o software), como mostra a Figura 5.

**Figura 5 – Ambiente gráfico de simulação “grSim” em execução**



**Fonte: Autoria própria.**

d. `$ cd QtProjectsRobocup/`

e. `$ cd QtMainClient/`

f. `$ ./QtMainClient`

para entrar na pasta “QtProjectsRobocup”  
para entrar na pasta “QtMainClient”  
executa o programa para escolher a cor do time (1. Azul ou 2. Amarelo) e iniciar o futebol dos robôs. Neste momento, os robôs devem começar a “andar”, como mostra a Figura 6.

**Figura 6 – Futebol de Robôs rodando no ambiente gráfico de simulação “grSim”**

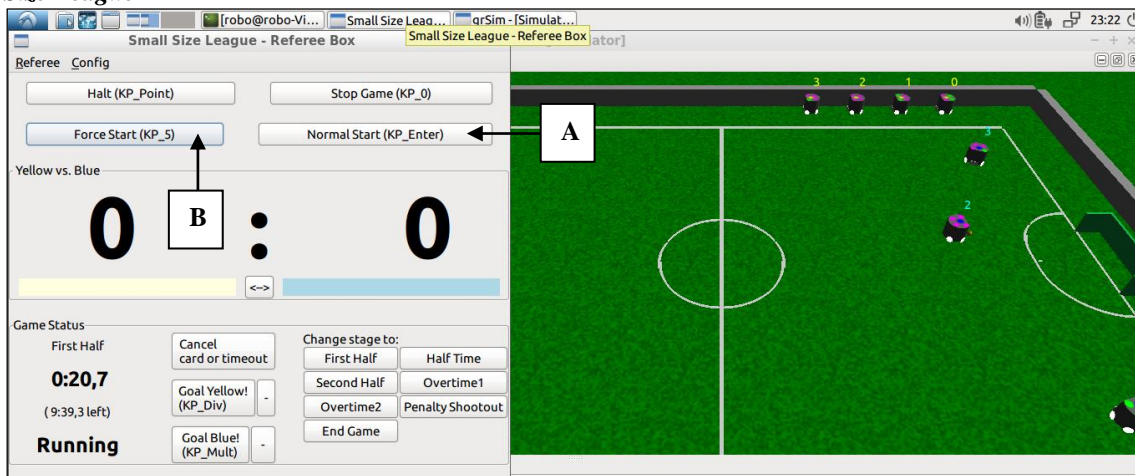


**Fonte: Autoria própria.**



- g. No ambiente gráfico de simulação “grSim” para posicionar a bola deve-se clicar como o botão direito, em qualquer lugar da janela, e escolher no menu a primeira opção identificada como “Locate ball here” – item A na Figura 6.
- h. Por último, no ambiente gráfico de simulação “Small Size League”, pressionar o botão “Normal Start (KP\_Enter)” – item A na Figura 7 e a seguir o botão “Force Start (KP\_5)” – item B na Figura 7. Esta sequência de ações fazem os robôs se movimentarem como mostra a Figura 7.

**Figura 7 – Robôs se movimentando após ações executadas no ambiente gráfico de simulação *Small Size League***



Fonte: Autoria própria.

Para manter as *Regras (Rules)*: alterando, eliminando ou adicionando; definindo a estratégia de jogo de um robô deve-se no terminal da máquina virtual “Lubuntu+ROS\_Kinetic” (para abrir um novo terminal: Ctrl + Shift + T) implementar:

- a. \$ cd para entrar no diretório raiz do usuário
- b. \$ cd Desktop/Robocup2012/ para entrar na pasta “Robocup2012”
- c. \$ cd ControleF180/ para entrar na pasta “ControleF180”
- d. StrategyPON.cpp código-fonte das *Rules*
- e. void StrategyPON::initRules() fazer as alterações desejadas no *Method* “initRules”

Alteradas as *Rules* no arquivo “StrategyPON.cpp” para realizar a instalação delas no ambiente, através do terminal deve-se:

- a. \$ cd para entrar no diretório raiz do usuário
- b. \$ cd Desktop/Robocup2012/QtProjectsRobocup/QtMainClient/ para entrar no diretório “QtMainClient”
- c. \$ ./my\_install.sh executa a sequência de comandos implementados no arquivo de scripts “my\_install.sh”

Fonte: Tutorial fornecido pelo **Prof. Dr. João Alberto Fabro** (<http://www.dainf.ct.utfpr.edu.br/~fabro/>, [fabro@dainf.ct.utfpr.edu.br](mailto:fabro@dainf.ct.utfpr.edu.br)).

## APÊNDICE B: PROJETO “ROBOCUP2012” EM AMBIENTE DE SIMULAÇÃO

O projeto do aplicativo “RoboCup2012” em tecnologia PON, em contexto de Futebol de Robôs, para execução no ambiente de simulação disponibilizado para estudo tem sua organização estrutural de pacotes<sup>1</sup> como mostra a Figura 8. Esta estrutura apresenta, por exemplo, os pacotes: a) ControleF180: classes que implementam a estratégia de jogo de um robô; e b) FrameworkPON: *Framework* C++ 2.0 que materializa o Paradigma Orientado a Notificações – PON.

**Figura 8 – Organização estrutural do projeto “RoboCup2012”**

.vscode	27/01/2020 22:39	File folder	
CommunicationSystemXBot	27/01/2020 22:39	File folder	
ControleF180	08/03/2020 18:50	File folder	
Diagramas	15/02/2020 21:22	File folder	
flie_bellator	27/01/2020 22:39	File folder	
FrameworkPON	27/01/2020 22:39	File folder	
librobot-1.1.0	27/01/2020 22:39	File folder	
MainClient	27/01/2020 22:39	File folder	
mani-monaj-grSim-0a3490a	27/01/2020 22:39	File folder	
QtProjectsRobocup	27/01/2020 22:39	File folder	
RedeNeural	27/01/2020 22:39	File folder	
SCISoccer1.1	27/01/2020 22:39	File folder	
scisoccer-1.3.5	27/01/2020 22:39	File folder	
ScriptFutebol	29/02/2020 10:46	File folder	
sslrefbox-2010.2	27/01/2020 22:39	File folder	
XBotCom	27/01/2020 22:39	File folder	
Como_Instalar	21/08/2017 19:57	Text Document	1 KB
executar	21/08/2017 19:57	File	1 KB
futebol	21/08/2017 19:57	Shell Script	1 KB
gamecontrol	21/08/2017 19:57	File	1 KB
joga	18/02/2020 11:01	Shell Script	1 KB
librobot-1.1.0.tar	21/08/2017 19:57	Arquivo do WinRAR	340 KB
logs2	21/08/2017 19:57	Text Document	88 KB
referee.sav	21/08/2017 19:57	SAV File	1 KB
Topicos Relatorio	21/08/2017 19:57	File	1 KB

Fonte: Autoria própria.

No pacote “ControleF180”, com a relação dos códigos-fontes apresentados na Figura 9, do projeto do aplicativo de controle de robôs “RoboCup2012”, tem-se a implementação das classes que implementam a estratégia de jogo de um robô.

<sup>1</sup> Os **pacotes** são utilizados para organizar um conjunto de classes relacionadas.

**Figura 9 – Códigos-fontes do pacote “ControleF180” projeto “RoboCup2012”**

Ball	Ball	Coach
Coach	CommunicationSystem	CommunicationSystem
ComSysSimulator	ComSysSimulator	ComSysXBot
ComSysXBot	DynamicElement	DynamicElement
Element	Element	EnemyRobot
exec	FactoryRobot	FactoryRobotPI
FactoryRobotPI	FactoryRobotPON	FactoryRobotPON
Game	Game	gamecontrol
GameFunctions	ListElement	ListEnemyRobots
ListEnemyRobots	Logger	Logger
main_teste	Makefile	MVector
MVector	Point	Point
referee.sav	RefereeBoxCommunication	RefereeBoxCommunication
RobocupController	RobocupController	Robot
Robot	RobotBehavior	RobotBehavior
RobotBehaviorGoalkeeper	RobotBehaviorGoalkeeper	RobotPI
RobotPI	RobotPON	RobotPON
SoccerEntity	SoccerEntity	Strategy
Strategy	StrategyPON	StrategyPON
StrategyPonAT	StrategyPonAT	StrategyPonDF
StrategyPonDF	StrategyPonLD	StrategyPonLD
StrategyPonLE	StrategyPonLE	Team
Team.d	Team	UtilGame
UtilGame		

**Fonte: Autoria própria.**

No Quadro 6 são apresentados os principais códigos-fontes, arquivos de cabeçalhos (\*.h) e de implementação (\*.cpp), do pacote “ControleF180” pertencente ao projeto do aplicativo “RoboCup2012”. Na primeira coluna tem-se a identificação do programa e na segunda coluna uma descrição da finalidade de implementação do respectivo programa.

**Quadro 6 – Principais códigos-fontes do pacote “ControleF180” projeto “RoboCup2012”**

(continua)

<b>Código-fonte</b>	<b>Finalidade da implementação</b>
Robot	<p>Classe “Robot” define a estrutura base do robô implementada com herança múltipla.</p> <p>Classes Pai: “SoccerEntity” e “DynamicElement”.</p> <p><i>Attributes</i> de identificação do robô: id; number; team.</p> <p><i>Attribute</i> com o comando do juiz: refereeCmd.</p> <p><i>Attributes</i> de movimento: angularVelocity; kick; dribble; orientation.</p> <p><i>Attributes</i> gerais: isBallMine; distanceForBall; PON.</p>
RobotPON	<p>Classe “RoboPON” define a estrutura complementar do robô implementada com herança múltipla.</p> <p>Classes Pai: “Robot” e “FBE”.</p> <p><i>Attributes</i> PON, como por exemplo: atIsReady, atBallIsMine, atCmdReferee, atLastCmdReferee, atPosX e atPosY.</p>

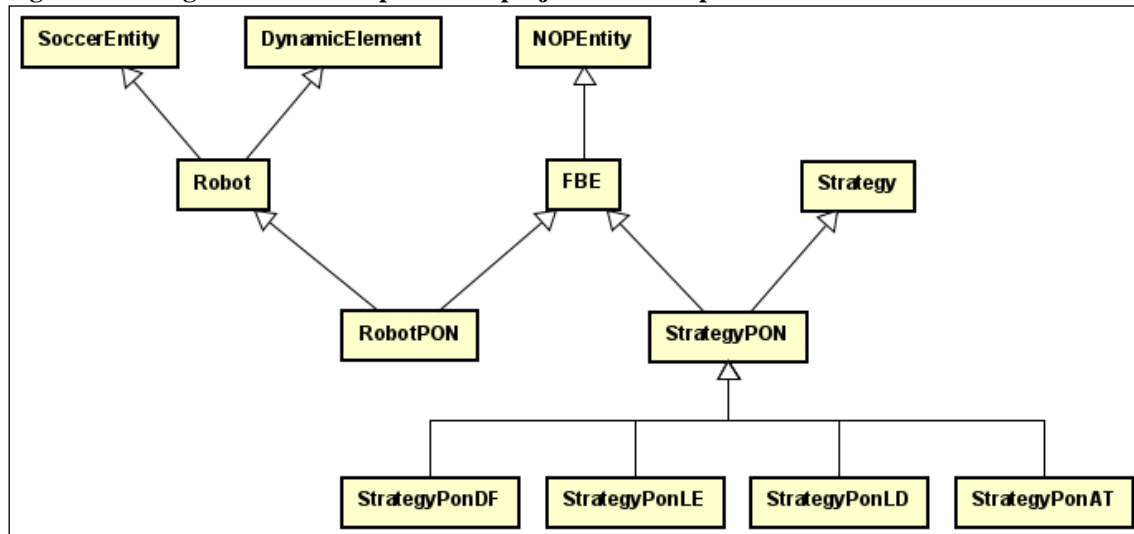
**Quadro 6 – Principais códigos-fontes do projeto “RoboCup2012” pacote “ControleF180”**  
(conclusão)

<b>Código-fonte</b>	<b>Finalidade da implementação</b>
StrategyPON	<p>Classe “StrategyPON” implementada com herança múltipla.</p> <p>Classes Pai: “Strategy” e “FBE”.</p> <p><i>Attributes:</i></p> <ul style="list-style-type: none"> <li>a) “MethodPointer” (<i>Instigation</i>), como por exemplo: mtExecuteMove; mtAngleMove; mtDribble; mtShootToGoal e mtReadyKickOff;</li> <li>b) “Premise”, como por exemplo: prRobotMoveX; prRobotMoveY; prRobotAngleMove; prBallIsMine; prRobotIsReady e prBallIsFar;</li> <li>c) “RuleObject”, como por exemplo: rlRobotMoveX; rlRobotMoveY; rlAngleMove; rlDribble; rlPassBal e rlShootToGoal.</li> </ul> <p>Principais <i>Methods</i>:</p> <ul style="list-style-type: none"> <li>a) initPremises(): cria as <i>Premises</i> através do <i>Method</i> “PREMISE”, por exemplo:  <pre>PREMISE(prRobotMoveX, this-&gt;robot-&gt;atPosX, this-&gt;robot-&gt;atPosToGoX, Premise::DIFFERENT, Premise::STANDARD, false);</pre> <pre>PREMISE(prRobotMoveY, this-&gt;robot-&gt;atPosY, this-&gt;robot-&gt;atPosToGoY, Premise::DIFFERENT, Premise::STANDARD, false);</pre> </li> <li>b) initInstigations(): cria as <i>Instigations</i> alocando um objeto “MethodPointer”, por exemplo:  <pre>mtExecuteMove = new MethodPointer&lt;PONRobot&gt;(this, &amp;BaseRobot::executeMove);</pre> </li> <li>c) initRules(): cria as <i>Rules</i>, por exemplo:  <pre>// Rule: Move to PosToGo (X axis) RULE(rlRobotMoveX, scheduler, Condition::CONJUNCTION); rlRobotMoveX-&gt;addPremise(prRobotMoveX); rlRobotMoveX-&gt;addPremise(prAllowMoveX); rlRobotMoveX-&gt;addInstigation(INSTIGATION(this-&gt;mtExecuteMove));</pre> <pre>// Rule: Move to PosToGo (Y axis) RULE(rlRobotMoveY, scheduler, Condition::CONJUNCTION); rlRobotMoveY-&gt;addPremise(prRobotMoveY); rlRobotMoveY-&gt;addPremise(prAllowMoveY); rlRobotMoveY-&gt;addInstigation(INSTIGATION(this-&gt;mtExecuteMove));</pre> </li> </ul>
StrategyPONAT	Classe “StrategyPonAT”. Implementa a estratégia dos robôs com as posições de atacante. A posição do robô é definida pelo tipo enumerado “PlayPosition”: <u>enum</u> PlayPosition{Defender, LeftSide, RightSide, Forward};
StrategyPonDF	Classe “StrategyPonDF”. Implementa a estratégia dos robôs com as posições de defensores (Defender).
StrategyPonLD	Classe “StrategyPonLD”. Implementa a estratégia dos robôs com as posições de defensores pelo lado direito do campo (RightSide).
StrategyPonLE	Classe “StrategyPonLE”. Implementa a estratégia dos robôs com as posições de defensores pelo lado esquerdo do campo (LeftSide).

**Fonte: Autoria própria.**

O diagrama de classes parcial do projeto “RoboCup2012”, apresentado na Figura 10, mostra as classes: “Robot”, “RobotPON”, “StrategyPON”, “StrategyPONAT”, “StrategyPonDF”, “StrategyPonLD” e “StrategyPonLE”, que estão descritas novamente no Quadro 6, e seus respectivos relacionamentos de herança ( $\rightarrow$ ).

**Figura 10 – Diagrama de classes parcial do projeto “RoboCup2012”**



Fonte: Autoria própria.

A proposta de trabalho de conclusão de disciplina apresentada neste relatório, envolveu o desenvolvimento de novas *Rules* no aplicativo “RoboCup2012” em tecnologia PON disponibilizado para estudo (Figura 10), tendo como base o trabalho desenvolvido em Santos (2017) (Apêndice C), afim de melhorar o comportamento dos robôs no contexto de uma partida de Futebol de Robôs (RoboCup). As *Rules* implementadas com este propósito estão relacionadas ainda no Quadro 5 da seção 4 nomeadamente de “Desenvolvimento da Aplicação para Controle do Futebol de Robôs” e com suas respectivas implementações descritas nos Apêndices D e E.

## APÊNDICE C: PROJETO “ROBOCUP2012” DESENVOLVIDO EM SANTOS (2017)

O trabalho de Leonardo Araujo Santos (SANTOS, 2017), propôs uma nova versão da linguagem de programação específica para o Paradigma Orientado a Notificações (PON), nomeada LingPON. A versão 1.2 da LingPON foi então empregada no desenvolvimento de um software (RoboCup2012) que trata do controle para partidas de Futebol de Robôs (RoboCup).

Como parte também do trabalho de Leandro Araujo Santos, o software desenvolvido na LingPON versão 1.2 (SANTOS, 2017, p. 106) foi comparado quantitativamente e qualitativamente como outros dois softwares equivalentes, a saber, um desenvolvido utilizando o *Framework* PON C++ 2.0<sup>2</sup> (SANTOS, 2017, p. 104); e outro desenvolvido na LingPON versão 1.0 (SANTOS, 2017, p. 105).

As comparações entre as aplicações de controle de Futebol de Robôs desenvolvidas em Santos (2017), são no tocante a medidas de complexidade de código, principalmente, a quantidade de linhas de códigos; e o nível de manutenibilidade.

Na avaliação da complexidade medindo a quantidade de linhas de código os resultados foram: 1º) a diferença do número de linhas de códigos da aplicação implementada na versão 1.2 da LingPON desenvolvida por Santos (2017), em relação a solução proposta na versão 1.0 da LingPON foi considerável: 3.357 na LingPON versão 1.2 contra 8.578 na LingPON versão 1.0 (SANTOS, 2017, p. 109); 2º) já na comparação com a solução desenvolvida no *Framework* PON C++ 2.0, está apresentou um número menor de linhas de código: 3.232 no *Framework* PON C++ 2.0 contra 3.357 na LingPON versão 1.2 (SANTOS, 2017, p. 109).

Já no caso da avaliação do nível de manutenibilidade as métricas utilizadas foram o tempo gasto para desenvolver o código necessário para implementar um novo requisito do software e a quantidade de linhas no código-fonte alteradas (SANTOS, 2017, p. 111). A comparação buscada de forma justa e imparcial foi realizada por três desenvolvedores de software, são eles: a) o autor do trabalho Leonardo Araujo Santos (SANTOS, 2017); b) o estudante de Engenharia da Computação André Luiz Costantino Botta<sup>3</sup>; e c) o professor Dr. João Alberto Fabro. Um resultado apresentado mostrou que na implementação de novas *Rule* foram necessárias 80 novas linhas ao código-fonte

---

<sup>2</sup> *Framework* PON C++ 2.0: materialização do Paradigma Orientado a Notificações (PON) na linguagem de programação C++.

<sup>3</sup> Currículo Lattes disponível em: <<http://lattes.cnpq.br/5735332297553810>>, acesso em: 06 mai. 2020.

LingPON versão 1.2, 34 novas linhas ao código-fonte *Framework PON C++ 2.0* e 534 novas linhas na aplicação desenvolvida utilizando a LingPON 1.0 (SANTOS, 2017, p. 118).

As conclusões gerais apresentadas em resumo no trabalho de Santos (2017) relatam que é possível desenvolver aplicações PON de forma mais fácil e com menor esforço utilizando a versão 1.2 da LingPON.

Outrossim, a análise da solução disponibilizada sobre o viés do Paradigma Orientado a Notificações (PON) é buscar no cerne de uma aplicação PON a construção de *Rules* a partir das *Conditions* com *Premises* e *Instigations* com *Actions* (SANTOS, 2017, p. 103).

## 1 PREMISES

As *Premissas (Premises)* são responsáveis pela realização das verificações que definem a tomada de decisão de uma *Rule*. A relação das 46 *Premises*, levantadas a partir da análise dos códigos-fontes do projeto desenvolvido em Santos (2017) e disponibilizado para estudo é apresentada no Quadro 7. O referido quadro tem na primeira coluna o nome de identificação da *Premise* e o código da sua expressão lógica e na segunda a relação das *Rules* as quais a respectiva *Premise* foi adicionada.

**Quadro 7 – Conjunto de *Premises* do projeto “RoboCup2012” desenvolvido em Santos (2017)**

(continua)

nome da <i>Premise</i> código da <i>Premise</i>	<i>Rules</i>
1. prActiveRole Robot.atRole != ""	5. rlBallFar 12. rlStartTargetToBall
2. prAngleMove Robot.atAngle != Robot.atAngleToGo	4. rlAngleMove
3. prBallCloseTeamGoal Robot.atBallDistanceToTeamGoal <= distanceMinStop	13. rlGoalkeeperStopCloseGoal 22. rlDefenderLeftStopBallClose 31. rlDefenderRightStopBallClose
4. prBallEnemyField Robot.atBallEnemyField == true	10. rlStartEnemyFieldPositionKick 11. rlStartEnemyFieldKick
5. prBallFarTeamGoal Robot.atBallDistanceToTeamGoal > distanceMinStop	14. rlGoalkeeperStopFarGoal 21. rlDefenderLeftStopBallFar 30. rlDefenderRightStopBallFar
6. prBallInsideGoalArea Robot.atBallDistanceToTeamGoal <= goalkeeperAreaRadius	15. rlGoalkeeperStartInsideAreaClosestBall 16. rlGoalkeeperStartInsideAreaClosestBallKick 17. rlGoalkeeperStartInsideArea
7. prBallIsFar Robot.atDistanceToBall >= 300	5. rlBallFar
8. prBallNotInsideGoalArea Robot.atBallDistanceToTeamGoal > goalkeeperAreaRadius	18. rlGoalkeeperStartOutsideArea

Quadro 7 – Conjunto de *Premises* do projeto “RoboCup2012” desenvolvido em Santos (2017)

(continua)

nome da <i>Premise</i> código da <i>Premise</i>	<i>Rules</i>
9. prBallTeamField Robot.atBallEnemyField == false	6. rlStartFreePartner 7. rlStartFreePartnerPass 8. rlStartNoFreePartner 9. rlStartNoFreePartnerKick
10. prClosestToBall Robot.atClosestToBall == true	6. rlStartFreePartner 7. rlStartFreePartnerPass 8. rlStartNoFreePartner 9. rlStartNoFreePartnerKick 10. rlStartEnemyFieldPositionKick 11. rlStartEnemyFieldKick 15. rlGoalkeeperStartInsideAreaClosestBall 16. rlGoalkeeperStartInsideAreaClosestBallKick
11. prFreePartner Robot.atPartnerFreeID >= 0	6. rlStartFreePartner 7. rlStartFreePartnerPass
12. prLastRefereeCmdKickoffBlue Robot.atLastRefereeCmd == KickOffBlue	1. rlMidfieldOnlyBlueReadyKickoffBlue
13. prLastRefereeCmdKickoffYellow Robot.atLastRefereeCmd == KickOffYellow	42. rlMidfieldOnlyYellowReadyKickoffYellow
14. prLastRefereeCmdPenaltyBlue Robot.atLastRefereeCmd == PenaltyBlue	51. rlMidfieldOnlyBlueReadyPenaltyBlue
15. prLastRefereeCmdPenaltyYellow Robot.atLastRefereeCmd == PenaltyYellow	52. rlMidfieldOnlyYellowReadyPenaltyYellow
16. prLinePlayerRole Robot.atRole != "GOALKEEPER"	6. rlStartFreePartner 7. rlStartFreePartnerPass 8. rlStartNoFreePartner 9. rlStartNoFreePartnerKick 10. rlStartEnemyFieldPositionKick 11. rlStartEnemyFieldKick
17. prMoreThanFourPlayers Robot.atNumPlayers > 4	40. rlMidfieldOnlyBlueKickoff 41. rlMidfieldOnlyYellowKickoff 43. rlMidfieldOnlyBlueDirectKick 44. rlMidfieldOnlyYellowDirectKick 76. rlDefenderOnlyStopFourPlayers
18. prNoEnemyOnLineGoal Robot.atEnemyOnGoalLine == false	10. rlStartEnemyFieldPositionKick 11. rlStartEnemyFieldKick
19. prNoFreePartner Robot.atPartnerFreeID <= 0	8. rlStartNoFreePartner
20. prNotClosestToBall Robot.atClosestToBall == false	12. rlStartTargetToBall 17. rlGoalkeeperStartInsideArea 29. rlDefenderLeftStartBallNotClose 38. rlDefenderRightStartBallNotClose 63. rlStrikerLeftStartBallNotClose 74. rlStrikerRightStartBallNotClose
21. prRefereeCmdDirectKickBlue Robot.atRefereeCmd == DirectFreeKickBlue	23. rlDefenderLeftBlueDirectKickBlue 32. rlDefenderRightBlueDirectKickBlue 43. rlMidfieldOnlyBlueDirectKick 55. rlStrikerLeftBlueDirectKick 66. rlStrikerRightBlueDirectKick
22. prRefereeCmdDirectKickYellow Robot.atRefereeCmd == DirectFreeKickYellow	24. rlDefenderLeftYellowDirectKickYellow 33. rlDefenderRightYellowDirectKickYellow 44. rlMidfieldOnlyYellowDirectKick 56. rlStrikerLeftYellowDirectKick 67. rlStrikerRightYellowDirectKick



Quadro 7 – Conjunto de *Premises* do projeto “RoboCup2012” desenvolvido em Santos (2017)

(continua)

nome da <i>Premise</i> código da <i>Premise</i>	<i>Rules</i>
23. prRefereeCmdIndirectKickBlue Robot.atRefereeCmd == IndirectFreeKickBlue	25. rlDefenderLeftBlueIndirectKickBlue
	34. rlDefenderRightBlueIndirectKickBlue
	45. rlMidfieldOnlyBlueIndirectDirectKick
	57. rlStrikerLeftBlueIndirectKick
	68. rlStrikerRightBlueIndirectKick
24. prRefereeCmdIndirectKickYellow Robot.atRefereeCmd == IndirectFreeKickYellow	26. rlDefenderLeftYellowIndirectKickYellow
	35. rlDefenderRightYellowIndirectKickYellow
	46. rlMidfieldOnlyYellowIndirectDirectKick
	58. rlStrikerLeftYellowIndirectKick
	69. rlStrikerRightYellowIndirectKick
25. prRefereeCmdKickoffBlue Robot.atRefereeCmd == KickOffBlue	40. rlMidfieldOnlyBlueKickoff
26. prRefereeCmdKickoffYellow Robot.atRefereeCmd == KickOffYellow	77. rlDefenderOnlyBlueKickoff
	41. rlMidfieldOnlyYellowKickoff
27. prRefereeCmdPenaltyBlue Robot.atRefereeCmd == PenaltyBlue	78. rlDefenderOnlyYellowKickoff
	20. rlGoalkeeperYellowPenaltyBlue
	28. rlDefenderLeftYellowPenaltyBlue
	37. rlDefenderRightYellowPenaltyBlue
	47. rlMidfieldOnlyBluePenaltyBlue
	50. rlMidfieldOnlyYellowPenaltyBlue
	59. rlStrikerLeftBluePenaltyBlue
	62. rlStrikerLeftYellowPenaltyBlue
28. prRefereeCmdPenaltyYellow Robot.atRefereeCmd == PenaltyYellow	70. rlStrikerRightBluePenaltyBlue
	73. rlStrikerRightYellowPenaltyBlue
	19. rlGoalkeeperBluePenaltyYellow
	27. rlDefenderLeftBluePenaltyYellow
	36. rlDefenderRightBluePenaltyYellow
	48. rlMidfieldOnlyBluePenaltyYellow
	49. rlMidfieldOnlyYellowPenaltyYellow
	60. rlStrikerLeftBluePenaltyYellow
29. prRefereeCmdStartGame Robot.atRefereeCmd == Ready	61. rlStrikerLeftYellowPenaltyYellow
	71. rlStrikerRightBluePenaltyYellow
	72. rlStrikerRightYellowPenaltyYellow
	1. rlMidfieldOnlyBlueReadyKickoffBlue
	6. rlStartFreePartner
	7. rlStartFreePartnerPass
	8. rlStartNoFreePartner
	9. rlStartNoFreePartnerKick
	10. rlStartEnemyFieldPositionKick
	11. rlStartEnemyFieldKick
	12. rlStartTargetToBall
	15. rlGoalkeeperStartInsideAreaClosestBall
	16. rlGoalkeeperStartInsideAreaClosestBallKick
	17. rlGoalkeeperStartInsideArea
	18. rlGoalkeeperStartOutsideArea
	29. rlDefenderLeftStartBallNotClose
	38. rlDefenderRightStartBallNotClose
	42. rlMidfieldOnlyYellowReadyKickoffYellow
	51. rlMidfieldOnlyBlueReadyPenaltyBlue
	52. rlMidfieldOnlyYellowReadyPenaltyYellow
	63. rlStrikerLeftStartBallNotClose
	74. rlStrikerRightStartBallNotClose

Quadro 7 – Conjunto de *Premises* do projeto “RoboCup2012” desenvolvido em Santos (2017)

(continua)

nome da <i>Premise</i> código da <i>Premise</i>	<i>Rules</i>
30. prRefereeCmdStop Robot.atRefereeCmd == Stop	13. rlGoalkeeperStopCloseGoal 14. rlGoalkeeperStopFarGoal 21. rlDefenderLeftStopBallFar 22. rlDefenderLeftStopBallClose 30. rlDefenderRightStopBallFar 31. rlDefenderRightStopBallClose 39. rlMidfieldOnlyStop 53. rlStrikerLeftStopTeamLeft 54. rlStrikerLeftStopTeamRight 64. rlStrikerRightStopTeamLeft 65. rlStrikerRightStopTeamRight 75. rlDefenderOnlyStopTwoPlayers 76. rlDefenderOnlyStopFourPlayers
31. prRobotIsNotReady Robot.atIsReady == false	6. rlStartFreePartner 8. rlStartNoFreePartner 10. rlStartEnemyFieldPositionKick 15. rlGoalkeeperStartInsideAreaClosestBall
32. prRobotIsReady Robot.atIsReady == true	7. rlStartFreePartnerPass 9. rlStartNoFreePartnerKick 11. rlStartEnemyFieldKick 16. rlGoalkeeperStartInsideAreaClosestBallKick
33. prRobotMoveX Robot.atPosX != Robot.atPosToGoX	2. rlRobotMoveX
34. prRobotMoveY Robot.atPosY != Robot.atPosToGoY	3. rlRobotMoveY
35. prRoleDefenderLeft Robot.atRole == "DEFENDER_LEFT"	21. rlDefenderLeftStopBallFar 22. rlDefenderLeftStopBallClose 23. rlDefenderLeftBlueDirectKickBlue 24. rlDefenderLeftYellowDirectKickYellow 25. rlDefenderLeftBlueIndirectKickBlue 26. rlDefenderLeftYellowIndirectKickYellow 27. rlDefenderLeftBluePenaltyYellow 28. rlDefenderLeftYellowPenaltyBlue 29. rlDefenderLeftStartBallNotClose
36. prRoleDefenderOnly Robot.atRole == "DEFENDER_ONLY"	75. rlDefenderOnlyStopTwoPlayers 76. rlDefenderOnlyStopFourPlayers 77. rlDefenderOnlyBlueKickoff 78. rlDefenderOnlyYellowKickoff
37. prRoleDefenderRight Robot.atRole == "DEFENDER_RIGHT"	30. rlDefenderRightStopBallFar 31. rlDefenderRightStopBallClose 32. rlDefenderRightBlueDirectKickBlue 33. rlDefenderRightYellowDirectKickYellow 34. rlDefenderRightBlueIndirectKickBlue 35. rlDefenderRightYellowIndirectKickYellow 36. rlDefenderRightBluePenaltyYellow 37. rlDefenderRightYellowPenaltyBlue 38. rlDefenderRightStartBallNotClose

Quadro 7 – Conjunto de *Premises* do projeto “RoboCup2012” desenvolvido em Santos (2017)

(continua)

nome da <i>Premise</i> código da <i>Premise</i>	<i>Rules</i>
38. prRoleGoalkeeper Robot.atRole == "GOALKEEPER"	13. rlGoalkeeperStopCloseGoal 14. rlGoalkeeperStopFarGoal 15. rlGoalkeeperStartInsideAreaClosestBall 16. rlGoalkeeperStartInsideAreaClosestBallKick 17. rlGoalkeeperStartInsideArea 18. rlGoalkeeperStartOutsideArea 19. rlGoalkeeperBluePenaltyYellow 20. rlGoalkeeperYellowPenaltyBlue
39. prRoleMidfieldOnly Robot.atRole == "MIDFIELD_ONLY"	1. rlMidfieldOnlyBlueReadyKickoffBlue 39. rlMidfieldOnlyStop 40. rlMidfieldOnlyBlueKickoff 41. rlMidfieldOnlyYellowKickoff 42. rlMidfieldOnlyYellowReadyKickoffYellow 43. rlMidfieldOnlyBlueDirectKick 44. rlMidfieldOnlyYellowDirectKick 45. rlMidfieldOnlyBlueIndirectDirectKick 46. rlMidfieldOnlyYellowIndirectDirectKick 47. rlMidfieldOnlyBluePenaltyBlue 48. rlMidfieldOnlyBluePenaltyYellow 49. rlMidfieldOnlyYellowPenaltyYellow 50. rlMidfieldOnlyYellowPenaltyBlue 51. rlMidfieldOnlyBlueReadyPenaltyBlue 52. rlMidfieldOnlyYellowReadyPenaltyYellow
40. prRoleStrickerLeft Robot.atRole == "STRIKER_LEFT"	53. rlStrikerLeftStopTeamLeft 54. rlStrikerLeftStopTeamRight 55. rlStrikerLeftBlueDirectKick 56. rlStrikerLeftYellowDirectKick 57. rlStrikerLeftBlueIndirectKick 58. rlStrikerLeftYellowIndirectKick 59. rlStrikerLeftBluePenaltyBlue 60. rlStrikerLeftBluePenaltyYellow 61. rlStrikerLeftYellowPenaltyYellow 62. rlStrikerLeftYellowPenaltyBlue 63. rlStrikerLeftStartBallNotClose
41. prRoleStrickerRight Robot.atRole == "STRIKER_RIGHT"	64. rlStrikerRighStopTeamLeft 65. rlStrikerRighStopTeamRight 66. rlStrikerRightBlueDirectKick 67. rlStrikerRightYellowDirectKick 68. rlStrikerRightBlueIndirectKick 69. rlStrikerRightYellowIndirectKick 70. rlStrikerRightBluePenaltyBlue 71. rlStrikerRightBluePenaltyYellow 72. rlStrikerRightYellowPenaltyYellow 73. rlStrikerRightYellowPenaltyBlue 74. rlStrikerRightStartBallNotClose
42. prTeamLeftSide Robot.atTeamSide == "LEFT"	53. rlStrikerLeftStopTeamLeft 64. rlStrikerRighStopTeamLeft
43. prTeamRightSide Robot.atTeamSide == "RIGHT"	54. rlStrikerLeftStopTeamRight 65. rlStrikerRighStopTeamRight
44. prTwoPlayers Robot.atNumPlayers == 2	75. rlDefenderOnlyStopTwoPlayers 77. rlDefenderOnlyBlueKickoff 78. rlDefenderOnlyYellowKickoff

**Quadro 7 – Conjunto de *Premises* do projeto “RoboCup2012” desenvolvido em Santos (2017)**  
(conclusão)

nome da <i>Premise</i> código da <i>Premise</i>	<i>Rules</i>
45. prTeamBlue Robot.atTeamColor == "BLUE"	1. rlMidfieldOnlyBlueReadyKickoffBlue
	19. rlGoalkeeperBluePenaltyYellow
	23. rlDefenderLeftBlueDirectKickBlue
	25. rlDefenderLeftBlueIndirectKickBlue
	27. rlDefenderLeftBluePenaltyYellow
	32. rlDefenderRightBlueDirectKickBlue
	34. rlDefenderRightBlueIndirectKickBlue
	36. rlDefenderRightBluePenaltyYellow
	40. rlMidfieldOnlyBlueKickoff
	43. rlMidfieldOnlyBlueDirectKick
	45. rlMidfieldOnlyBlueIndirectDirectKick
	47. rlMidfieldOnlyBluePenaltyBlue
	48. rlMidfieldOnlyBluePenaltyYellow
	51. rlMidfieldOnlyBlueReadyPenaltyBlue
	55. rlStrikerLeftBlueDirectKick
	57. rlStrikerLeftBlueIndirectKick
	59. rlStrikerLeftBluePenaltyBlue
	60. rlStrikerLeftBluePenaltyYellow
	66. rlStrikerRightBlueDirectKick
	68. rlStrikerRightBlueIndirectKick
	70. rlStrikerRightBluePenaltyBlue
	71. rlStrikerRightBluePenaltyYellow
	77. rlDefenderOnlyBlueKickoff
46. prTeamYellow Robot.atTeamColor == "YELLOW"	20. rlGoalkeeperYellowPenaltyBlue
	24. rlDefenderLeftYellowDirectKickYellow
	26. rlDefenderLeftYellowIndirectKickYellow
	28. rlDefenderLeftYellowPenaltyBlue
	33. rlDefenderRightYellowDirectKickYellow
	35. rlDefenderRightYellowIndirectKickYellow
	37. rlDefenderRightYellowPenaltyBlue
	41. rlMidfieldOnlyYellowKickoff
	42. rlMidfieldOnlyYellowReadyKickoffYellow
	44. rlMidfieldOnlyYellowDirectKick
	46. rlMidfieldOnlyYellowIndirectDirectKick
	49. rlMidfieldOnlyYellowPenaltyYellow
	50. rlMidfieldOnlyYellowPenaltyBlue
	52. rlMidfieldOnlyYellowReadyPenaltyYellow
	56. rlStrikerLeftYellowDirectKick
	58. rlStrikerLeftYellowIndirectKick
	61. rlStrikerLeftYellowPenaltyYellow
	62. rlStrikerLeftYellowPenaltyBlue
	67. rlStrikerRightYellowDirectKick
	69. rlStrikerRightYellowIndirectKick
	72. rlStrikerRightYellowPenaltyYellow
	73. rlStrikerRightYellowPenaltyBlue
	78. rlDefenderOnlyYellowKickoff

Fonte: Autoria própria.

## 2 INSTIGATIONS

As *Instigações (Instigations)* são responsáveis pela realização dos serviços ou habilidades associadas a uma *Rule*. A relação das 28 *Instigations*, levantadas a partir da

análise dos códigos-fontes do projeto desenvolvido em Santos (2017) e disponibilizado para estudo é apresentada no Quadro 8. O referido quadro tem na primeira coluna o nome da *Instigation* e *Method* e na segunda a relação das *Rules* as quais a respectiva *Instigation* foi adicionada.

**Quadro 8 – Conjunto de *Instigations* do projeto “RoboCup2012” desenvolvido em Santos (2017)**  
(continua)

nome da <i>Instigation</i>   <i>Method</i>	<i>Rules</i>
1. mtAngleMove   angleMove()	4. rlAngleMove
2. mtExecuteMove   executeMove()	2. rlRobotMoveX 3. rlRobotMoveY
3. mtGoalKeeperEnemyPenalty   behaviorGkPenalty()	13. rlGoalkeeperStopCloseGoal 17. rlGoalkeeperStartInsideArea 19. rlGoalkeeperBluePenaltyYellow 20. rlGoalkeeperYellowPenaltyBlue
4. mtGoalKeeperOutArea   behaviorGkBallOutsideArea()	14. rlGoalkeeperStopFarGoal 18. rlGoalkeeperStartOutsideArea
5. mtMarkEnemy   markEnemy()	76. rlDefenderOnlyStopFourPlayers
6. mtMoveGoalLeftPost   moveToLeftPost()	21. rlDefenderLeftStopBallFar 23. rlDefenderLeftBlueDirectKickBlue 24. rlDefenderLeftYellowDirectKickYellow 25. rlDefenderLeftBlueIndirectKickBlue 26. rlDefenderLeftYellowIndirectKickYellow 29. rlDefenderLeftStartBallNotClose
7. mtMoveGoalRightPost   moveToRightPost()	30. rlDefenderRightStopBallFar 32. rlDefenderRightBlueDirectKickBlue 33. rlDefenderRightYellowDirectKickYellow 34. rlDefenderRightBlueIndirectKickBlue 35. rlDefenderRightYellowIndirectKickYellow 38. rlDefenderRightStartBallNotClose
8. mtMoveIndirectKick   moveToIndirectKickPosition()	45. rlMidfieldOnlyBlueIndirectDirectKick 46. rlMidfieldOnlyYellowIndirectDirectKick
9. mtMovePenaltyDefenseCenter   movePenaltyDefenseCenterPosition()	48. rlMidfieldOnlyBluePenaltyYellow 50. rlMidfieldOnlyYellowPenaltyBlue
10. mtMovePenaltyDefenseLeft   movePenaltyDefenseLeftPosition()	27. rlDefenderLeftBluePenaltyYellow 28. rlDefenderLeftYellowPenaltyBlue
11. mtMovePenaltyDefenseRight   movePenaltyDefenseRightPosition()	36. rlDefenderRightBluePenaltyYellow 37. rlDefenderRightYellowPenaltyBlue
12. mtMovePenaltyOffensiveLeft   movePenaltyAttackLeftPosition()	59. rlStrikerLeftBluePenaltyBlue 61. rlStrikerLeftYellowPenaltyYellow
13. mtMovePenaltyOffensiveRight   movePenaltyAttackRightPosition()	70. rlStrikerRightBluePenaltyBlue 72. rlStrikerRightYellowPenaltyYellow
14. mtMovePositionPenaltyKick   positionKickPenalty()	47. rlMidfieldOnlyBluePenaltyBlue 49. rlMidfieldOnlyYellowPenaltyYellow
15. mtMoveStopPosition   moveToStopPosition()	39. rlMidfieldOnlyStop 75. rlDefenderOnlyStopTwoPlayers
16. mtMoveStopPositionAngleNeg   moveToStopPositionNegativeAngle()	53. rlStrikerLeftStopTeamLeft 65. rlStrikerRightStopTeamRight
17. mtPositionPassBallPartner   positionTopassBallPartner()	6. rlStartFreePartner
18. mtMoveStopPositionAngleNeg3pl   moveToStopPositionNegativeAngle3pl()	22. rlDefenderLeftStopBallClose
19. mtMoveStopPositionAnglePos   moveToStopPositionPositiveAngle()	54. rlStrikerLeftStopTeamRight 64. rlStrikerRightStopTeamLeft

**Quadro 8 – Conjunto de *Instigations* do projeto “RoboCup2012” desenvolvido em Santos (2017) (conclusão)**

nome da <i>Instigation</i>   <i>Method</i>	<i>Rules</i>
20. mtMovePositionToKick   moveToKickPosition()	8. rlStartNoFreePartner
	10. rlStartEnemyFieldPositionKick
	15. rlGoalkeeperStartInsideAreaClosestBall
	40. rlMidfieldOnlyBlueKickoff
	41. rlMidfieldOnlyYellowKickoff
	43. rlMidfieldOnlyBlueDirectKick
	44. rlMidfieldOnlyYellowDirectKick
	77. rlDefenderOnlyBlueKickoff
21. mtMoveStopPositionAnglePos3pl   moveToStopPositionPositiveAngle3pl()	78. rlDefenderOnlyYellowKickoff
	31. rlDefenderRightStopBallClose
22. mtMoveStrickerLeftTeamDirect   moveStrikerPositionLeft()	55. rlStrikerLeftBlueDirectKick
	56. rlStrikerLeftYellowDirectKick
	57. rlStrikerLeftBlueIndirectKick
	58. rlStrikerLeftYellowIndirectKick
	60. rlStrikerLeftBluePenaltyYellow
	62. rlStrikerLeftYellowPenaltyBlue
	63. rlStrikerLeftStartBallNotClose
23. mtMoveStrickerRightTeamDirect   moveStrikerPositionRight()	66. rlStrikerRightBlueDirectKick
	67. rlStrikerRightYellowDirectKick
	68. rlStrikerRightBlueIndirectKick
	69. rlStrikerRightYellowIndirectKick
	71. rlStrikerRightBluePenaltyYellow
	73. rlStrikerRightYellowPenaltyBlue
	74. rlStrikerRightStartBallNotClose
24. mtPassBallPartner   passBallPartner()	7. rlStartFreePartnerPass
25. mtReadyKickoff   readyKickoff()	1. rlMidfieldOnlyBlueReadyKickoffBlue
	9. rlStartNoFreePartnerKick
	11. rlStartEnemyFieldKick
	16. rlGoalkeeperStartInsideAreaClosestBallKick
	42. rlMidfieldOnlyYellowReadyKickoffYellow
26. mtReadyPenalty   readyPenalty()	51. rlMidfieldOnlyBlueReadyPenaltyBlue
	52. rlMidfieldOnlyYellowReadyPenaltyYellow
27. mtResetKick   resetKickAndDribble()	5. rlBallFar
28. mtTargetToBall   targetToBall()	12. rlStartTargetToBall

**Fonte: Autoria própria.**

### 3 RULES

Uma *Regra (Rule)* é decomposta em entidades *Condição (Condition)*, conjunto de *Premises* e *Ação (Action)*, conjunto de *Instigations*). A relação das 78 *Rules*, levantadas a partir da análise dos códigos-fontes do projeto desenvolvido em Santos (2017) e disponibilizado para estudo é apresentada no Quadro 9. O referido quadro tem na primeira coluna o nome da *Rule*, na segunda a entidade *Condition* com a relação das *Premises* associadas e na terceira a entidade *Action* com a *Instigation* pertinente.

Quadro 9 – Conjunto de *Rules* do projeto “RoboCup2012” desenvolvido em Santos (2017)

(continua)

nome da <i>Rule</i>	<i>Condition</i> e suas <i>Premises</i>	<i>Action</i> e sua <i>Instigation</i>
1. rlMidfieldOnlyBlueReadyKickoffBlue	prRefereeCmdStartGame Robot.atRefereeCmd == Ready prRoleMidfieldOnly Robot.atRole == "MIDFIELD_ONLY" prLastRefereeCmdKickoffBlue Robot.atLastRefereeCmd == KickOffBlue prTeamBlue Robot.atTeamColor == "BLUE"	mtReadyKickoff
2. rlRobotMoveX	prRobotMoveX Robot.atPosX != Robot.atPosToGoX	mtExecuteMove
3. rlRobotMoveY	prRobotMoveY Robot.atPosY != Robot.atPosToGoY	mtExecuteMove
4. rlAngleMove	prAngleMove Robot.atAngle != Robot.atAngleToGo	mtAngleMove
5. rlBallFar	prBallIsFar Robot.atDistanceToBall >= 300 prActiveRole Robot.atRole != ""	mtResetKick
6. rlStartFreePartner	prRefereeCmdStartGame Robot.atRefereeCmd == Ready prLinePlayerRole Robot.atRole != "GOALKEEPER" prBallTeamField Robot.atBallEnemyField == false prClosestToBall Robot.atClosestToBall == true prFreePartner Robot.atPartnerFreeID >= 0 prRobotIsNotReady Robot.atIsReady == false	mtPositionPassBallPartner
7. rlStartFreePartnerPass	prRefereeCmdStartGame Robot.atRefereeCmd == Ready prLinePlayerRole Robot.atRole != "GOALKEEPER" prBallTeamField Robot.atBallEnemyField == false prClosestToBall Robot.atClosestToBall == true prFreePartner Robot.atPartnerFreeID >= 0 prRobotIsReady Robot.atIsReady == true	mtPassBallPartner
8. rlStartNoFreePartner	prRefereeCmdStartGame Robot.atRefereeCmd == Ready prLinePlayerRole Robot.atRole != "GOALKEEPER" prBallTeamField Robot.atBallEnemyField == false prClosestToBall Robot.atClosestToBall == true prNoFreePartner Robot.atPartnerFreeID <= 0 prRobotIsNotReady Robot.atIsReady == false	mtMovePositionToKick
9. rlStartNoFreePartnerKick	prRefereeCmdStartGame Robot.atRefereeCmd == Ready prLinePlayerRole Robot.atRole != "GOALKEEPER" prBallTeamField Robot.atBallEnemyField == false prClosestToBall Robot.atClosestToBall == true prRobotIsReady Robot.atIsReady == true	mtReadyKickoff

Quadro 9 – Conjunto de *Rules* do projeto “RoboCup2012” desenvolvido em Santos (2017)

(continua)

nome da <i>Rule</i>	<i>Condition</i> e suas <i>Premises</i>	<i>Action</i> e sua <i>Instigation</i>
10. rlStartEnemyFieldPositionKick	prRefereeCmdStartGame Robot.atRefereeCmd == Ready prLinePlayerRole Robot.atRole != "GOALKEEPER" prBallEnemyField Robot.atBallEnemyField == true prNoEnemyOnLineGoal Robot.atEnemyOnGoalLine == false prClosestToBall Robot.atClosestToBall == true prRobotIsNotReady Robot.atIsReady == false	mtMovePositionToKick
11. rlStartEnemyFieldKick	prRefereeCmdStartGame Robot.atRefereeCmd == Ready prLinePlayerRole Robot.atRole != "GOALKEEPER" prBallEnemyField Robot.atBallEnemyField == true prNoEnemyOnLineGoal Robot.atEnemyOnGoalLine == false prClosestToBall Robot.atClosestToBall == true prRobotIsReady Robot.atIsReady == true	mtReadyKickoff
12. rlStartTargetToBall	prRefereeCmdStartGame Robot.atRefereeCmd == Ready prActiveRole Robot.atRole != "" prNotClosestToBall Robot.atClosestToBall == false	mtTargetToBall
13. rlGoalkeeperStopCloseGoal	prRefereeCmdStop Robot.atRefereeCmd == Stop prRoleGoalkeeper Robot.atRole == "GOALKEEPER" prBallCloseTeamGoal Robot.atBallDistanceToTeamGoal <= distanceMinStop	mtGoalKeeperEnemyPenalty
14. rlGoalkeeperStopFarGoal	prRefereeCmdStop Robot.atRefereeCmd == Stop prRoleGoalkeeper Robot.atRole == "GOALKEEPER" prBallFarTeamGoal Robot.atBallDistanceToTeamGoal > distanceMinStop	mtGoalKeeperOutArea
15. rlGoalkeeperStartInsideAreaClosestBall	prRefereeCmdStartGame Robot.atRefereeCmd == Ready prRoleGoalkeeper Robot.atRole == "GOALKEEPER" prBallInsideGoalArea Robot.atBallDistanceToTeamGoal <= goalkeeperAreaRadius prClosestToBall Robot.atClosestToBall == true prRobotIsNotReady Robot.atIsReady == false	mtMovePositionToKick
16. rlGoalkeeperStartInsideAreaClosestBallKick	prRefereeCmdStartGame Robot.atRefereeCmd == Ready prRoleGoalkeeper Robot.atRole == "GOALKEEPER" prBallInsideGoalArea Robot.atBallDistTeamGoal <= goalkeeperAreaRadius prClosestToBall Robot.atClosestToBall == true prRobotIsReady Robot.atIsReady == true	mtReadyKickoff



**Quadro 9 – Conjunto de *Rules* do projeto “RoboCup2012” desenvolvido em Santos (2017)**

(continua)

nome da <i>Rule</i>	<i>Condition</i> e suas <i>Premises</i>	<i>Action</i> e sua <i>Instigation</i>
17. rlGoalkeeperStartInsideArea	prRefereeCmdStartGame Robot.atRefereeCmd == Ready prRoleGoalkeeper Robot.atRole == "GOALKEEPER" prBallInsideGoalArea Robot.atBallDistTeamGoal <= goalkeeperAreaRadius prNotClosestToBall Robot.atClosestToBall == false	mtGoalKeeperEnemyPenalty
18. rlGoalkeeperStartOutsideArea	prRefereeCmdStartGame Robot.atRefereeCmd == Ready prRoleGoalkeeper Robot.atRole == "GOALKEEPER" prBallNotInsideGoalArea Robot.atBallDistanceToTeamGoal > goalkeeperAreaRadius	mtGoalKeeperOutArea
19. rlGoalkeeperBluePenaltyYellow	prRefereeCmdPenaltyYellow Robot.atRefereeCmd == PenaltyYellow prRoleGoalkeeper Robot.atRole == "GOALKEEPER" prTeamBlue Robot.atTeamColor == "BLUE"	mtGoalKeeperEnemyPenalty
20. rlGoalkeeperYellowPenaltyBlue	prRefereeCmdPenaltyBlue Robot.atRefereeCmd == PenaltyBlue prRoleGoalkeeper Robot.atRole == "GOALKEEPER" prTeamYellow Robot.atTeamColor == "YELLOW"	mtGoalKeeperEnemyPenalty
21. rlDefenderLeftStopBallFar	prRefereeCmdStop Robot.atRefereeCmd == Stop prRoleDefenderLeft Robot.atRole == "DEFENDER_LEFT" prBallFarTeamGoal Robot.atBallDistanceToTeamGoal > distanceMinStop	mtMoveGoalLeftPost
22. rlDefenderLeftStopBallClose	prRefereeCmdStop Robot.atRefereeCmd == Stop prRoleDefenderLeft Robot.atRole == "DEFENDER_LEFT" prBallCloseTeamGoal Robot.atBallDistanceToTeamGoal <= distanceMinStop	mtMoveStopPositionAngleNeg3pl
23. rlDefenderLeftBlueDirectKickBlue	prRefereeCmdDirectKickBlue Robot.atRefereeCmd == DirectFreeKickBlue prRoleDefenderLeft Robot.atRole == "DEFENDER_LEFT" prTeamBlue Robot.atTeamColor == "BLUE"	mtMoveGoalLeftPost
24. rlDefenderLeftYellowDirectKickYellow	prRefereeCmdDirectKickYellow Robot.atRefereeCmd == DirectFreeKickYellow prRoleDefenderLeft Robot.atRole == "DEFENDER_LEFT" prTeamYellow Robot.atTeamColor == "YELLOW"	mtMoveGoalLeftPost
25. rlDefenderLeftBlueIndirectKickBlue	prRefereeCmdIndirectKickBlue Robot.atRefereeCmd == IndirectFreeKickBlue prRoleDefenderLeft Robot.atRole == "DEFENDER_LEFT" prTeamBlue Robot.atTeamColor == "BLUE"	mtMoveGoalLeftPost
26. rlDefenderLeftYellowIndirectKickYellow	prRefereeCmdIndirectKickYellow Robot.atRefereeCmd == IndirectFreeKickYellow prRoleDefenderLeft Robot.atRole == "DEFENDER_LEFT" prTeamYellow Robot.atTeamColor == "YELLOW"	mtMoveGoalLeftPost

**Quadro 9 – Conjunto de Rules do projeto “RoboCup2012” desenvolvido em Santos (2017)**

(continua)

<b>nome da Rule</b>	<b>Condition e suas Premises</b>	<b>Action e sua Instigation</b>
27. rlDefenderLeftBluePenaltyYellow	prRefereeCmdPenaltyYellow Robot.atRefereeCmd == PenaltyYellow prRoleDefenderLeft Robot.atRole == "DEFENDER_LEFT" prTeamBlue Robot.atTeamColor == "BLUE"	mtMovePenaltyDefenseLeft
28. rlDefenderLeftYellowPenaltyBlue	prRefereeCmdPenaltyBlue Robot.atRefereeCmd == PenaltyBlue prRoleDefenderLeft Robot.atRole == "DEFENDER_LEFT" prTeamYellow Robot.atTeamColor == "YELLOW"	mtMovePenaltyDefenseLeft
29. rlDefenderLeftStartBallNotClose	prRefereeCmdStartGame Robot.atRefereeCmd == Ready prRoleDefenderLeft Robot.atRole == "DEFENDER_LEFT" prNotClosestToBall Robot.atClosestToBall == false	mtMoveGoalLeftPost
30. rlDefenderRightStopBallFar	prRefereeCmdStop Robot.atRefereeCmd == Stop prRoleDefenderRight Robot.atRole == "DEFENDER_RIGHT" prBallFarTeamGoal Robot.atBallDistanceToTeamGoal > distanceMinStop	mtMoveGoalRightPost
31. rlDefenderRightStopBallClose	prRefereeCmdStop Robot.atRefereeCmd == Stop prRoleDefenderRight Robot.atRole == "DEFENDER_RIGHT" prBallCloseTeamGoal Robot.atBallDistanceToTeamGoal <= distanceMinStop	mtMoveStopPositionAnglePos3pl
32. rlDefenderRightBlueDirectKickBlue	prRefereeCmdDirectKickBlue Robot.atRefereeCmd == DirectFreeKickBlue prRoleDefenderRight Robot.atRole == "DEFENDER_RIGHT" prTeamBlue Robot.atTeamColor == "BLUE"	mtMoveGoalRightPost
33. rlDefenderRightYellowDirectKickYellow	prRefereeCmdDirectKickYellow Robot.atRefereeCmd == DirectFreeKickYellow prRoleDefenderRight Robot.atRole == "DEFENDER_RIGHT" prTeamYellow Robot.atTeamColor == "YELLOW"	mtMoveGoalRightPost
34. rlDefenderRightBlueIndirectKickBlue	prRefereeCmdIndirectKickBlue Robot.atRefereeCmd == IndirectFreeKickBlue prRoleDefenderRight Robot.atRole == "DEFENDER_RIGHT" prTeamBlue Robot.atTeamColor == "BLUE"	mtMoveGoalRightPost
35. rlDefenderRightYellowIndirectKickYellow	prRefereeCmdIndirectKickYellow Robot.atRefereeCmd == IndirectFreeKickYellow prRoleDefenderRight Robot.atRole == "DEFENDER_RIGHT" prTeamYellow Robot.atTeamColor == "YELLOW"	mtMoveGoalRightPost
36. rlDefenderRightBluePenaltyYellow	prRefereeCmdPenaltyYellow Robot.atRefereeCmd == PenaltyYellow prRoleDefenderRight Robot.atRole == "DEFENDER_RIGHT" prTeamBlue Robot.atTeamColor == "BLUE"	mtMovePenaltyDefenseRight

**Quadro 9 – Conjunto de Rules do projeto “RoboCup2012” desenvolvido em Santos (2017)**

(continua)

<b>nome da Rule</b>	<b>Condition e suas Premises</b>	<b>Action e sua Instigation</b>
37. rlDefenderRightYellowPenaltyBlue	prRefereeCmdPenaltyBlue Robot.atRefereeCmd == PenaltyBlue prRoleDefenderRight Robot.atRole == "DEFENDER_RIGHT" prTeamYellow Robot.atTeamColor == "YELLOW"	mtMovePenaltyDefenseRight
38. rlDefenderRightStartBallNotClose	prRefereeCmdStartGame Robot.atRefereeCmd == Ready prRoleDefenderRight Robot.atRole == "DEFENDER_RIGHT" prNotClosestToBall Robot.atClosestToBall == false	mtMoveGoalRightPost
39. rlMidfieldOnlyStop	prRefereeCmdStop Robot.atRefereeCmd == Stop prRoleMidfieldOnly Robot.atRole == "MIDFIELD_ONLY"	mtMoveStopPosition
40. rlMidfieldOnlyBlueKickoff	prRefereeCmdKickoffBlue Robot.atRefereeCmd == KickOffBlue prRoleMidfieldOnly Robot.atRole == "MIDFIELD_ONLY" prTeamBlue Robot.atTeamColor == "BLUE" prMoreThanFourPlayers Robot.atNumPlayers > 4	mtMovePositionToKick
41. rlMidfieldOnlyYellowKickoff	prRefereeCmdKickoffYellow Robot.atRefereeCmd == KickOffYellow prRoleMidfieldOnly Robot.atRole == "MIDFIELD_ONLY" prTeamYellow Robot.atTeamColor == "YELLOW" prMoreThanFourPlayers Robot.atNumPlayers > 4	mtMovePositionToKick
42. rlMidfieldOnlyYellowReadyKickoffYellow	prRefereeCmdStartGame Robot.atRefereeCmd == Ready prRoleMidfieldOnly Robot.atRole == "MIDFIELD_ONLY" prLastRefereeCmdKickoffYellow Robot.atLastRefereeCmd == KickOffYellow prTeamYellow Robot.atTeamColor == "YELLOW"	mtReadyKickoff
43. rlMidfieldOnlyBlueDirectKick	prRefereeCmdDirectKickBlue Robot.atRefereeCmd == DirectFreeKickBlue prRoleMidfieldOnly Robot.atRole == "MIDFIELD_ONLY" prTeamBlue Robot.atTeamColor == "BLUE" prMoreThanFourPlayers Robot.atNumPlayers > 4	mtMovePositionToKick
44. rlMidfieldOnlyYellowDirectKick	prRefereeCmdDirectKickYellow Robot.atRefereeCmd == DirectFreeKickYellow prRoleMidfieldOnly Robot.atRole == "MIDFIELD_ONLY" prTeamYellow Robot.atTeamColor == "YELLOW" prMoreThanFourPlayers Robot.atNumPlayers > 4	mtMovePositionToKick
45. rlMidfieldOnlyBlueIndirectDirectKick	prRefereeCmdIndirectKickBlue Robot.atRefereeCmd == IndirectFreeKickBlue prRoleMidfieldOnly Robot.atRole == "MIDFIELD_ONLY" prTeamBlue Robot.atTeamColor == "BLUE"	mtMoveIndirectKick

**Quadro 9 – Conjunto de *Rules* do projeto “RoboCup2012” desenvolvido em Santos (2017)**

(continua)

nome da <i>Rule</i>	<i>Condition</i> e suas <i>Premises</i>	<i>Action</i> e sua <i>Instigation</i>
46. rlMidfieldOnlyYellowIndirectDirectKick	prRefereeCmdIndirectKickYellow Robot.atRefereeCmd == IndirectFreeKickYellow prRoleMidfieldOnly Robot.atRole == "MIDFIELD_ONLY" prTeamYellow Robot.atTeamColor == "YELLOW"	mtMoveIndirectKick
47. rlMidfieldOnlyBluePenaltyBlue	prRefereeCmdPenaltyBlue Robot.atRefereeCmd == PenaltyBlue prRoleMidfieldOnly Robot.atRole == "MIDFIELD_ONLY" prTeamBlue Robot.atTeamColor == "BLUE"	mtMovePositionPenaltyKick
48. rlMidfieldOnlyBluePenaltyYellow	prRefereeCmdPenaltyYellow Robot.atRefereeCmd == PenaltyYellow prRoleMidfieldOnly Robot.atRole == "MIDFIELD_ONLY" prTeamBlue Robot.atTeamColor == "BLUE"	mtMovePenaltyDefenseCenter
49. rlMidfieldOnlyYellowPenaltyYellow	prRefereeCmdPenaltyYellow Robot.atRefereeCmd == PenaltyYellow prRoleMidfieldOnly Robot.atRole == "MIDFIELD_ONLY" prTeamYellow Robot.atTeamColor == "YELLOW"	mtMovePositionPenaltyKick
50. rlMidfieldOnlyYellowPenaltyBlue	prRefereeCmdPenaltyBlue Robot.atRefereeCmd == PenaltyBlue prRoleMidfieldOnly Robot.atRole == "MIDFIELD_ONLY" prTeamYellow Robot.atTeamColor == "YELLOW"	mtMovePenaltyDefenseCenter
51. rlMidfieldOnlyBlueReadyPenaltyBlue	prRefereeCmdStartGame Robot.atRefereeCmd == Ready prRoleMidfieldOnly Robot.atRole == "MIDFIELD_ONLY" prLastRefereeCmdPenaltyBlue Robot.atLastRefereeCmd == PenaltyBlue prTeamBlue Robot.atTeamColor == "BLUE"	mtReadyPenalty
52. rlMidfieldOnlyYellowReadyPenaltyYellow	prRefereeCmdStartGame Robot.atRefereeCmd == Ready prRoleMidfieldOnly Robot.atRole == "MIDFIELD_ONLY" prLastRefereeCmdPenaltyYellow Robot.atLastRefereeCmd == PenaltyYellow prTeamYellow Robot.atTeamColor == "YELLOW"	mtReadyPenalty
53. rlStrikerLeftStopTeamLeft	prRefereeCmdStop Robot.atRefereeCmd == Stop prRoleStrickerLeft Robot.atRole == "STRIKER_LEFT" prTeamLeftSide Robot.atTeamSide == "LEFT"	mtMoveStopPositionAngleNeg
54. rlStrikerLeftStopTeamRight	prRefereeCmdStop Robot.atRefereeCmd == Stop prRoleStrickerLeft Robot.atRole == "STRIKER_LEFT" prTeamRightSide Robot.atTeamSide == "RIGHT"	mtMoveStopPositionAnglePos
55. rlStrikerLeftBlueDirectKick	prRefereeCmdDirectKickBlue Robot.atRefereeCmd == DirectFreeKickBlue prRoleStrickerLeft Robot.atRole == "STRIKER_LEFT" prTeamBlue Robot.atTeamColor == "BLUE"	mtMoveStrickerLeftTeamDirect

**Quadro 9 – Conjunto de *Rules* do projeto “RoboCup2012” desenvolvido em Santos (2017)**

(continua)

<b>nome da <i>Rule</i></b>	<b><i>Condition</i> e suas <i>Premises</i></b>	<b><i>Action</i> e sua <i>Instigation</i></b>
56. rlStrikerLeftYellowDirectKick	prRefereeCmdDirectKickYellow Robot.atRefereeCmd == DirectFreeKickYellow prRoleStrickerLeft Robot.atRole == "STRIKER_LEFT" prTeamYellow Robot.atTeamColor == "YELLOW"	mtMoveStrickerLeftTeamDirect
57. rlStrikerLeftBlueIndirectKick	prRefereeCmdIndirectKickBlue Robot.atRefereeCmd == IndirectFreeKickBlue prRoleStrickerLeft Robot.atRole == "STRIKER_LEFT" prTeamBlue Robot.atTeamColor == "BLUE"	mtMoveStrickerLeftTeamDirect
58. rlStrikerLefttYellowIndirectKick	prRefereeCmdIndirectKickYellow Robot.atRefereeCmd == IndirectFreeKickYellow prRoleStrickerLeft Robot.atRole == "STRIKER_LEFT" prTeamYellow Robot.atTeamColor == "YELLOW"	mtMoveStrickerLeftTeamDirect
59. rlStrikerLeftBluePenaltyBlue	prRefereeCmdPenaltyBlue Robot.atRefereeCmd == PenaltyBlue prTeamBlue Robot.atTeamColor == "BLUE"	mtMovePenaltyOffensiveLeft
60. rlStrikerLeftBluePenaltyYellow	prRefereeCmdPenaltyYellow Robot.atRefereeCmd == PenaltyYellow prRoleStrickerLeft Robot.atRole == "STRIKER_LEFT" prTeamBlue Robot.atTeamColor == "BLUE"	mtMoveStrickerLeftTeamDirect
61. rlStrikerLeftYellowPenaltyYellow	prRefereeCmdPenaltyYellow Robot.atRefereeCmd == PenaltyYellow prRoleStrickerLeft Robot.atRole == "STRIKER_LEFT" prTeamYellow Robot.atTeamColor == "YELLOW"	mtMovePenaltyOffensiveLeft
62. rlStrikerLeftYellowPenaltyBlue	prRefereeCmdPenaltyBlue Robot.atRefereeCmd == PenaltyBlue prRoleStrickerLeft Robot.atRole == "STRIKER_LEFT" prTeamYellow Robot.atTeamColor == "YELLOW"	mtMoveStrickerLeftTeamDirect
63. rlStrikerLeftStartBallNotClose	prRefereeCmdStartGame Robot.atRefereeCmd == Ready prRoleStrickerLeft Robot.atRole == "STRIKER_LEFT" prNotClosestToBall Robot.atClosestToBall == false	mtMoveStrickerLeftTeamDirect
64. rlStrikerRighStopTeamLeft	prRefereeCmdStop Robot.atRefereeCmd == Stop prRoleStrickerRight Robot.atRole == "STRIKER_RIGHT" prTeamLeftSide Robot.atTeamSide == "LEFT"	mtMoveStopPositionAnglePos
65. rlStrikerRighStopTeamRight	prRefereeCmdStop Robot.atRefereeCmd == Stop prRoleStrickerRight Robot.atRole == "STRIKER_RIGHT" prTeamRightSide Robot.atTeamSide == "RIGHT"	mtMoveStopPositionAngleNeg
66. rlStrikerRightBlueDirectKick	prRefereeCmdDirectKickBlue Robot.atRefereeCmd == DirectFreeKickBlue prRoleStrickerRight Robot.atRole == "STRIKER_RIGHT" prTeamBlue Robot.atTeamColor == "BLUE"	mtMoveStrickerRightTeamDirect

**Quadro 9 – Conjunto de *Rules* do projeto “RoboCup2012” desenvolvido em Santos (2017)**

(continua)

<b>nome da <i>Rule</i></b>	<b><i>Condition</i> e suas <i>Premises</i></b>	<b><i>Action</i> e sua <i>Instigation</i></b>
67. rlStrikerRightYellowDirectKick	prRefereeCmdDirectKickYellow Robot.atRefereeCmd == DirectFreeKickYellow prRoleStrickerRight Robot.atRole == "STRIKER_RIGHT" prTeamYellow Robot.atTeamColor == "YELLOW"	mtMoveStrickerRightTeamDirect
68. rlStrikerRightBlueIndirectKick	prRefereeCmdIndirectKickBlue Robot.atRefereeCmd == IndirectFreeKickBlue prRoleStrickerRight Robot.atRole == "STRIKER_RIGHT" prTeamBlue Robot.atTeamColor == "BLUE"	mtMoveStrickerRightTeamDirect
69. rlStrikerRightYellowIndirectKick	prRefereeCmdIndirectKickYellow Robot.atRefereeCmd == IndirectFreeKickYellow prRoleStrickerRight Robot.atRole == "STRIKER_RIGHT" prTeamYellow Robot.atTeamColor == "YELLOW"	mtMoveStrickerRightTeamDirect
70. rlStrikerRightBluePenaltyBlue	prRefereeCmdPenaltyBlue Robot.atRefereeCmd == PenaltyBlue prRoleStrickerRight Robot.atRole == "STRIKER_RIGHT" prTeamBlue Robot.atTeamColor == "BLUE"	mtMovePenaltyOffensiveRight
71. rlStrikerRightBluePenaltyYellow	prRefereeCmdPenaltyYellow Robot.atRefereeCmd == PenaltyYellow prRoleStrickerRight Robot.atRole == "STRIKER_RIGHT" prTeamBlue Robot.atTeamColor == "BLUE"	mtMoveStrickerRightTeamDirect
72. rlStrikerRightYellowPenaltyYellow	prRefereeCmdPenaltyYellow Robot.atRefereeCmd == PenaltyYellow prRoleStrickerRight Robot.atRole == "STRIKER_RIGHT" prTeamYellow Robot.atTeamColor == "YELLOW"	mtMovePenaltyOffensiveRight
73. rlStrikerRightYellowPenaltyBlue	prRefereeCmdPenaltyBlue Robot.atRefereeCmd == PenaltyBlue prRoleStrickerRight Robot.atRole == "STRIKER_RIGHT" prTeamYellow Robot.atTeamColor == "YELLOW"	mtMoveStrickerRightTeamDirect
74. rlStrikerRightStartBallNotClose	prRefereeCmdStartGame Robot.atRefereeCmd == Ready prRoleStrickerRight Robot.atRole == "STRIKER_RIGHT" prNotClosestToBall Robot.atClosestToBall == false	mtMoveStrickerRightTeamDirect
75. rlDefenderOnlyStopTwoPlayers	prRefereeCmdStop Robot.atRefereeCmd == Stop prRoleDefenderOnly Robot.atRole == "DEFENDER_ONLY" prTwoPlayers Robot.atNumPlayers == 2	mtMoveStopPosition
76. rlDefenderOnlyStopFourPlayers	prRefereeCmdStop Robot.atRefereeCmd == Stop prRoleDefenderOnly Robot.atRole == "DEFENDER_ONLY" prMoreThanFourPlayers Robot.atNumPlayers > 4	mtMarkEnemy

**Quadro 9 – Conjunto de *Rules* do projeto “RoboCup2012” desenvolvido em Santos (2017)**

(conclusão)
































<b>nome da Rule</b>	<b><i>Condition e suas Premises</i></b>	<b><i>Action e sua Instigation</i></b>
77. rlDefenderOnlyBlueKickoff	prRefereeCmdKickoffBlue Robot.atRefereeCmd == KickOffBlue prRoleDefenderOnly Robot.atRole == "DEFENDER_ONLY" prTeamBlue Robot.atTeamColor == "BLUE" prTwoPlayers Robot.atNumPlayers == 2	mtMovePositionToKick
78. rlDefenderOnlyYellowKickoff	prRefereeCmdKickoffYellow Robot.atRefereeCmd == KickOffYellow prRoleDefenderOnly Robot.atRole == "DEFENDER_ONLY" prTeamYellow Robot.atTeamColor == "YELLOW" prTwoPlayers Robot.atNumPlayers == 2	mtMovePositionToKick

**Fonte:** Autoria própria.

#### 4 DETALHES DA IMPLEMENTAÇÃO

A Figura 11 apresenta a relação dos códigos-fontes do projeto desenvolvido em Santos (2017) e disponibilizado para estudo que trata do controle para partidas de Futebol de Robôs (RoboCup). Foram três aplicações desenvolvidas com as seguintes materializações do PON, são elas: [A] LingPON versão 1.0; [B] LingPON versão 1.2; e [C] *Framework PON C++ 2.0*.

**Figura 11 – Softwares para partidas de Futebol de Robôs desenvolvido em Santos (2017)**

[A] LingPON versão 1.0	[B] LingPON versão 1.2	[C] <i>Framework PON C++ 2.0</i>
 BaseRobot  BaseRobot  EnemyRobot  robocup.pon  RobocupConsts  RobocupController  RobocupController  Team  Team	 BaseRobot  BaseRobot  EnemyRobot  robocup.pon  RobocupConsts  RobocupController  RobocupController  Team  Team	 BaseRobot  BaseRobot  EnemyRobot  FactoryRobot  FactoryRobotPON  FactoryRobotPON  PONRobot  PONRobot  RobocupConsts  RobocupController  RobocupController  Team  Team

**Fonte: Autoria própria.**

No Quadro 10 são apresentados os principais códigos-fontes/programas, arquivos de cabeçalhos (\*.h) e de implementação (\*.cpp), do projeto desenvolvido em Santos (2017) e disponibilizado para estudo. Na primeira coluna do referido quadro tem-se a identificação do programa e na segunda coluna uma descrição da finalidade de implementação do respectivo programa.



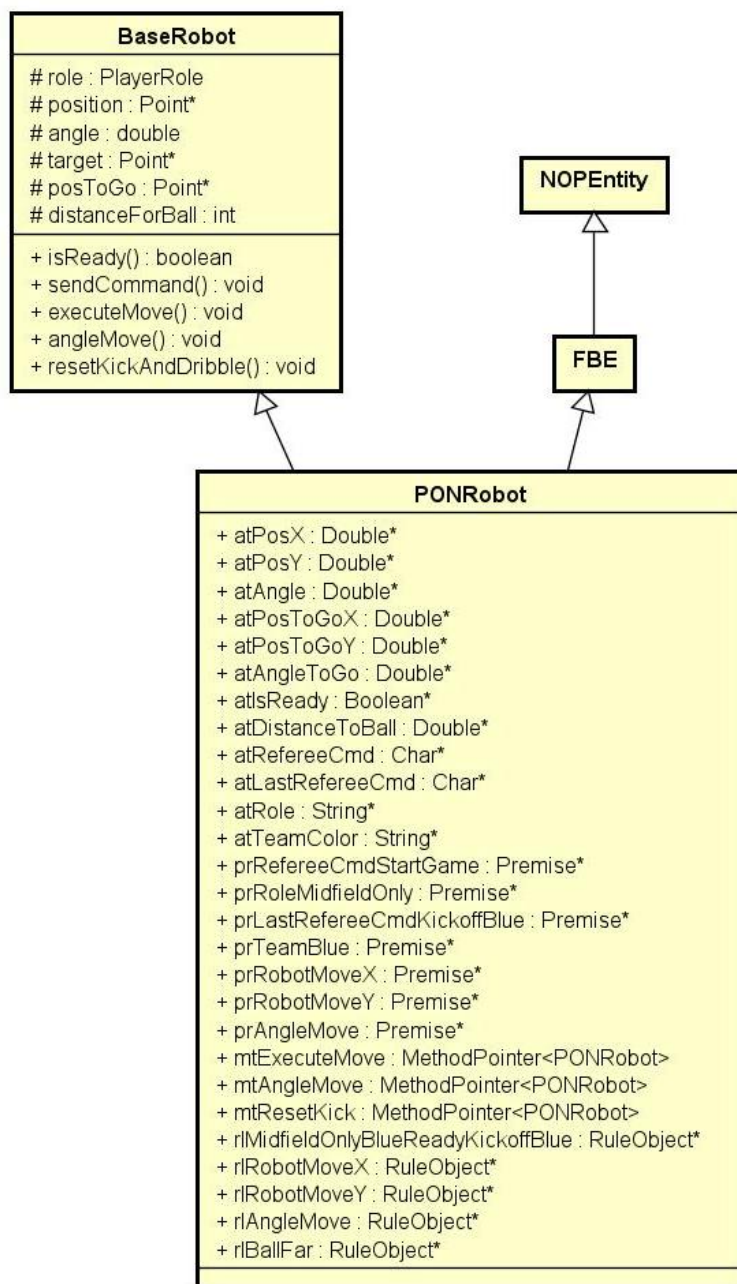
**Quadro 10 – Principais códigos-fontes do projeto “RoboCup2012” desenvolvido em Santos (2017)**

<b>código-fonte</b>	<b>finalidade da implementação</b>
BaseRobot	<p>Classe “BaseRobot” define a estrutura base do robô.</p> <p><i>Attributes</i> de identificação do robô: id; number; team; role.  <i>Attributes</i> de controle do robô: kick; dribble.  <i>Attributes</i> do jogador: teamColor; fieldSide; ownGoal; enemyGoal; ball.  <i>Attributes</i> de posição: position; angle; target; posToGo.  <i>Attributes</i> de configuração do campo de jogo: distanceForBall; distMinToOurGoal; distanceGoalMin; distanceMinStop; fieldHorizontalMin; fieldHorizontalMax; fieldVerticalMin; fieldVerticalMax; goalkeeperPenalty; penaltyMark.</p> <p><i>Methods</i> disparados pelas <i>Instigations</i>: executeMove(); angleMove(); moveToLeftPost; moveToRightPost(); moveStrikerPositionLeft(); moveStrikerPositionRight(); readyKickoff(); passBallPartner(); targetToBall().</p>
PONRobot	<p>Classe “RoboPON” define a estrutura complementar do robô implementada com herança múltipla. Classes Pai: “BaseRobot”, “FBE”.</p> <p><i>Attributes</i> que definem a posição atual do robô: atPosX; atPosY; atAngle.  <i>Attributes</i> com o comando do juiz: refereeCmd; atLastRefereeCmd. <i>Attribute</i> com o papel do jogador: atRole;  <i>Attributes</i> de identificação do time: atTeamSide; atTeamColor; atNumPlayers; atPartnerFreeID.  <i>Attributes</i> do tipo “Premise”: prRobotMoveX; prRobotMoveY; prRefereeCmdStop; prAngleMove; prRoleDefenderOnly; prRoleDefenderLeft; prRoleDefenderRight.  <i>Attributes</i> do tipo “MethodPointer&lt;PONRobot&gt;” (<i>Instigations</i>): mtExecuteMove; mtAngleMove; mtReadyKickoff; mtMoveGoalLeftPost; mtMoveGoalRightPost; mtMovePositionToKick.  <i>Attributes</i> do tipo “RuleObject”: rlRobotMoveX; rlRobotMoveY; rlBallFar; rlGoalkeeperBluePenaltyYellow; rlGoalkeeperYellowPenaltyBlue.</p> <p>Principais <i>Methods</i>:</p> <p>a) initAttributes(): inicializa os valores dos <i>Attributes</i> do robô;  b) initPremises(): cria as <i>Premises</i> através do <i>Method</i> “PREMISE”. Por exemplo:  PREMISE(prRobotMoveX, this-&gt;atPosX, this-&gt;atPosToGoX, Premise::DIFFERENT, Premise::STANDARD, false);  PREMISE(prRobotMoveY, this-&gt;atPosY, this-&gt;atPosToGoY, Premise::DIFFERENT, Premise::STANDARD, false);  c) initInstigations(): cria as <i>Instigations</i> alocando um objeto “MethodPointer”. Por exemplo:  mtExecuteMove = <u>new</u> MethodPointer&lt;PONRobot&gt;(this, &amp;BaseRobot::executeMove);  d) initRules(): cria as <i>Rules</i>. Por exemplo:  // Movement (X): Rule 2  RULE(rlRobotMoveX, scheduler, Condition::CONJUNCTION);  rlRobotMoveX-&gt;addPremise(prRobotMoveX);  rlRobotMoveX-&gt;addInstigation(INSTIGATION(this-&gt;mtExecuteMove));    // Movement (Y): Rule 3  RULE(rlRobotMoveY, scheduler, Condition::CONJUNCTION);  rlRobotMoveY-&gt;addPremise(prRobotMoveY);  rlRobotMoveY-&gt;addInstigation(INSTIGATION(this-&gt;mtExecuteMove));</p>

**Fonte: Autoria própria.**

O diagrama de classes parcial do projeto desenvolvido em Santos (2017) e disponibilizado para estudo, apresentado na Figura 12, mostra principalmente as classes: “BaseRobot” e “PONRobot”, que foram descritas novamente no Quadro 10, e seus respectivos relacionamentos de herança ( $\rightarrow$ ).

**Figura 12 – Diagrama de classes parcial do projeto “RoboCup2012” desenvolvido em Santos (2017)**



**Fonte: Autoria própria.**

Em tempo, volta-se a reforçar que a proposta de trabalho de conclusão de disciplina apresentada neste relatório, envolveu o desenvolvimento de *Rules* no aplicativo “RoboCup2012” para execução no ambiente de simulação disponibilizado para estudo (Apêndice B), a partir da análise desta implementação desenvolvida em Santos (2017). As *Rules* implementadas nesta proposta estão relacionadas novamente no Quadro 5 da seção 4, nomeadamente de “Desenvolvimento da Aplicação para Controle do Futebol de Robôs” e com suas respectivas implementações nos próximos Apêndices D e E.

**APÊNDICE D: RULES IMPLEMENTADAS NO PROJETO “ROBOCUP2012”  
PELOS ALUNOS ANDERSON EDUARDO DE LIMA, FELIPE DOS SANTOS  
NEVES, LUCAS TACHINI GARCIA E OMERO FRANCISCO BERTOL**

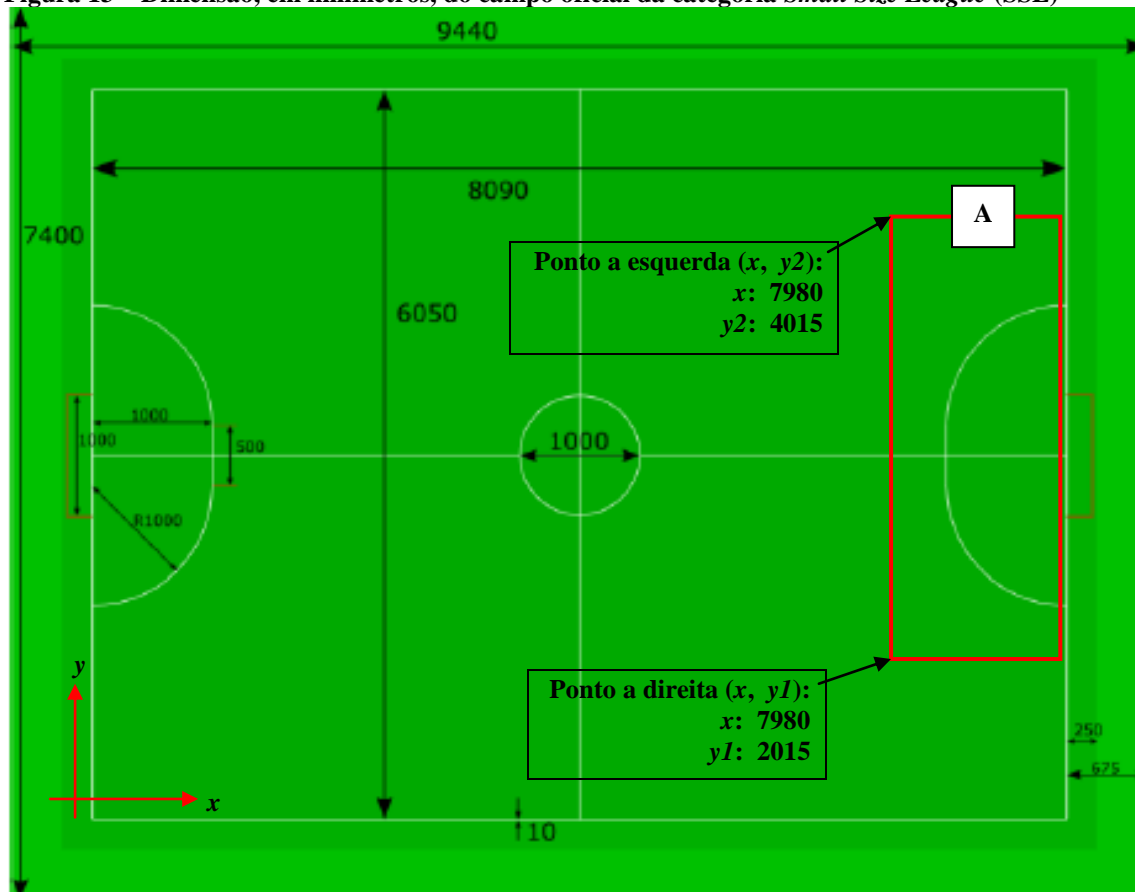
As implementações no projeto orientado a regras e baseado nos conceitos e propriedades da tecnologia PON para um simulador de Futebol de Robôs (RoboCup) executado em máquina virtual foram desenvolvidas no *Framework* PON C++ 2.0.

Segundo Santos (2017, p. 68), a Figura 13 apresenta as dimensões (em milímetros) do campo oficial da categoria *Small Size League* (SSL) das competições do Futebol de Rôbos (RoboCup).

As dimensões do retângulo [A] na Figura 13 foram consideradas como medidas da região próxima ao gol, são elas: a) Ponto à direita do gol ( $x, y1$ ) = (7980, 2015); b) Ponto à esquerda do gol ( $x, y2$ ) = (7980, 4015).

No caso das *Rules* implementadas pelo aluno Omero Francisco Bertol elas têm, basicamente, a semântica de analisar a posição do jogador e avaliá-la verificando se os valores dos *Attributes* “atPosX” e “atPosY” se encontram próximo ao gol ( $atPos > 7980$ ; e  $2015 < atPos < 4015$ ).

**Figura 13 – Dimensão, em milímetros, do campo oficial da categoria *Small Size League* (SSL)**



Fonte: Adaptado de Santos (2017, p. 68).

## Implementações realizadas

---

### 1. Na classe “RobotPON” declaração dos *Attributes*:

- “atClosestToGoal”: utilizado para definir se o jogador está próximo (*true*) ao gol ou não (*false*)  
Boolean\* atClosestToGoal;
- “atRole”: utilizado para definir a função/papel do jogador (“GOALKEEPER”, “DEFENDER\_LEFT”, “DEFENDER\_RIGHT”, “STRIKER”, “DEFENDER\_ONLY”);  
String \*atRole;
- “atGoalkeeper”: utilizado para definir se o jogador tem a função de goleiro  
Boolean\* atGoalkeeper;
- “atNextMoveToAttackArea”: utilizado para checar se próximo movimentação adentra a área de ataque adversária  
Boolean\* atNextMoveToAttackArea;
- “atNextMoveToDefenceArea”: utilizado para checar se o próximo movimento faz o jogador entrar na área de defesa do próprio time  
Boolean\* atNextMoveToDefenceArea;
- “atIsBlu”: utilizado para definir se a cor do time do jogador é azul  
Boolean\* atIsBlu;
- “atActiveRedCard”: utilizado para definir se o jogador recebeu um cartão vermelho  
Boolean\* atActiveRedCard;
- “atDribleCount”: utilizado para realizar a contagem de toques  
Integer \* atDribleCount;
- “atMaxDribleCount”: utilizado para definir o número máximo de toques  
Integer\* atMaxDribleCount;

### 2. Na classe “RobotPON” declaração e implementação do *Method* para “setar” um valor para o *Attribute* “atRole” (função/papel do jogador) e declaração para setar o *Attribute* “atActiveRedCard” (jogador recebeu um cartão vermelho):

```
void setatRole(const std::string& role);
```

```
void RobotPON::setatRole(const string& role) {  
    atRole->setValue(role, true);  
}
```

```
void RobotPON::setRedCard(const bool status) {  
    Robot::setRedCard(status);  
    atActiveRedCard->setValue(this->isRedCarded);  
  
    std::cout << "Took red card";  
}
```

### Implementações realizadas

---

3. No *Method* “initAttributes” da classe “RobotPON”:

- inicializa o valor do *Attribute* “atClosestToGoal” (próximo ao gol) como *false*:  
BOOLEAN(this, this->atClosestToGoal, false);
- inicializa o valor do *Attribute* “atRole” (função/papel do jogador) como “OFF”:  
STRING(this, this->atRole, “OFF”);
- inicializa o valor do *Attribute* “atGoalkeeper” (é goleiro) como *false*:  
BOOLEAN(this, this->atGoalkeeper, false);
- inicializa o valor do *Attribute* “atNextMoveToAttackArea” (movimento para área de ataque) como *false*:  
BOOLEAN(this, this->atNextMoveToAttackArea, false);
- inicializa o valor do *Attribute* “atNextMoveToDefenceArea” (movimento para área de defesa) como *false*:  
BOOLEAN(this, this->atNextMoveToDefenceArea, false);
- inicializa o valor do *Attribute* “atIsBlu” (cor do time é azul) como *false*:  
BOOLEAN(this, this->atIsBlu, false);
- inicializa o valor do *Attribute* “atActiveRedCard” (recebeu de cartão amarelo) como *false*:  
BOOLEAN(this, this->atActiveRedCard, false);
- inicializa o valor do *Attribute* “atDribleCount” (contador de toques) como 0:  
INTEGER(this, this->atDribleCount, 0);
- inicializa o valor do *Attribute* “atMaxDribleCount” (número máximo de toques) com um valor randomizado:  
INTEGER(this, this->atMaxDribleCount, 1+(rand()%5));

4. Na classe “StrategyPON” declaração dos *Attributes* (*Premises*, *Instigations* e *Rules*):

*// verifica se está próximo ao gol levando em consideração a coordenada X*

Premise \*prClosestToGoalX;

*// verifica se está próximo ao gol levando em consideração a coordenada Y (à*

*// direita do gol, valor Y1 na Figura 13)*

Premise \*prClosestToGoalY1;

*// verifica se está próximo ao gol levando em consideração a coordenada Y (à*

*// esquerda do gol, valor Y2 na Figura 13)*

Premise \*prClosestToGoalY2;

*// verifica se não está próximo ao gol levando em consideração a coordenada X*

Premise \*prNotClosestToGoalX;

*// verifica se a função/papel do jogador está ativo*

Premise \*prActiveRole;

## Implementações realizadas

---

### 4. Na classe “StrategyPON” declaração dos *Attributes* (*Premises*, *Instigations* e *Rules*):

*// verifica se excedeu o número máximo de toques na bola*

Premise \*prMaxDrible;

*// verifica se não está próximo ao gol levando em consideração a coordenada Y (à*

*// esquerda do gol, valor Y2 na Figura 13)*

Premise \*prNotClosestToGoalY2;

*// verifica se está próximo ao gol levando em consideração o Attribute*

*// “atClosestToGoal”*

Premise \*prClosestToGoal;

*// verifica se não está próximo ao gol levando em consideração a coordenada Y (à*

*// direita do gol, valor Y1 na Figura 13)*

Premise \*prNotClosestToGoalY1;

*// altera o valor do Attribute “atClosestToGoal” para true*

MethodPointer<StrategyPON>\* mtClosestToGoal;

void closestToGoal();

*// altera o valor do Attribute “atClosestToGoal” para false*

MethodPointer<StrategyPON>\* mtNotClosestToGoal;

void notClosestToGoal();

MethodPointer<StrategyPON>\* mtRedCardYellow;

void redCardYellowAction();

MethodPointer<StrategyPON>\* mtFixGkMove;

virtual void fixGeneralRobotNextMove();

MethodPointer<StrategyPON>\* mtFixRobotNotGkMove;

virtual void fixGkNextMove();

*// 1. marca o Attribute "atClosestToGoal" como true (está perto do gol)*

RuleObject\* rlClosestToGoal;

*// 2. marca o Attribute "atClosestToGoal" como false (não está perto do gol,*

*// levando em consideração o valor da coordenada “x”)*

RuleObject\* rlNotClosestToGoalX;

*// 3. O jogador estando pronto, com a bola e perto do gol então chuta para o gol*

RuleObject\* rlClosestToGoalKick;

*// 4. Redefinir chute e drible quando a bola estiver longe. Baseada na Rule 5*

*// (SANTOS, 2017, p. 179)*

RuleObject\* rlBallFar;

*// 5. Avalia necessidade de movimento em X*

RuleObject\* rlRobotWantsToMoveX;

## **Implementações realizadas**

---

4. Na classe “StrategyPON” declaração dos *Attributes* (*Premises*, *Instigations* e *Rules*):

*// 6. marca o Attribute "atClosestToGoal" como false (não está perto do gol,  
// levando em consideração o valor da coordenada “y”: direita do gol - valor y1  
// na Figura 13; esquerda do gol - valor y2 na Figura 13)*

RuleObject\* rlNotClosestToGoalY;

*// 7. Avalia necessidade de movimento em Y*

RuleObject\* rlRobotWantsToMoveY;

*// 8. Confirma a necessidade de movimento em X ou Y*

RuleObject\* rlRobotWantsToMove;

*// 9. Esta Rule deve agrupar condições negadas que possam vir a restringir o  
// movimento do jogador*

RuleObject\* rlRobotNotRetrained;

*// 10. Permite ou não que o movimento do goleiro seja executado*

RuleObject\* rlRobotMoveGoalKeeper;

*// 11. Permite o movimento de um jogador não goleiro seja executado*

RuleObject\* rlRobotMove;

*// 12. Avalia movimentação para áreas não permitidas*

RuleObject\* rlNextMoveToRestrictedArea;

*// 13. Modifica o movimento do jogador que não seja o goleiro*

RuleObject\* rlFixRobotNotGkmove;

*// 14. Modifica o movimento do goleiro*

RuleObject\* rlFixRobotGkmove;

*// 15. Aplica cartão vermelho para um jogador do time amarelo*

RuleObject\* rlRedCardYellow;

*// 16. Se a bola sai da área do jogo no campo adversário, força o jogador a voltar  
// para sua posição inicial*

RuleObject\* rlBallOutsideEnemyTeam;

*// 17. Chutar ao gol após estar posicionado*

RuleObject\* rlShootToGoal;

*// 18. Passar a bola após exceder determinado número de passes.*

RuleObject\* rlMaxDrible;

*// 19. Faz o goleiro passar a bola ao ficar com a posse dela muito próximo ao gol.*

RuleObject\* rlPassGkCloseGoal;

*// 20. Posicionar jogador para chutar ao gol quando estiver próximo.*

RuleObject\* rlPositionToShoot;

## Implementações realizadas

---

4. Na classe “StrategyPON” declaração dos *Attributes* (*Premises*, *Instigations* e *Rules*):  
*// 21. Resetar flags de controle após realizar chute.*

```
RuleObject* rlResetShootFlag;
```

5. void StrategyPON::initPremises():

```
// verificam se o jogador está próximo ao gol considerando os valores das  
// coordenadas "x" e "y"
```

```
PREMISE(prClosestToGoalX, this->robot->atPosX, new Double(this, 7980),  
        Premise::GREATEROREQUAL, Premise::STANDARD, false);  
PREMISE(prClosestToGoalY1, this->robot->atPosY, new Double(this, 2015),  
        Premise::GREATEROREQUAL, Premise::STANDARD, false);  
PREMISE(prClosestToGoalY2, this->robot->atPosY, new Double(this, 4025),  
        Premise::SMALLEROREQUAL, Premise::STANDARD, false);
```

```
// verificam se o jogador "não" está próximo ao gol considerando os valores das  
// coordenadas "x" e "y"
```

```
PREMISE(prNotClosestToGoalX, this->robot->atPosX, new Double(this, 7980),  
        Premise::SMALLERTHAN, Premise::STANDARD, false);  
PREMISE(prNotClosestToGoalY1, this->robot->atPosY, new Double(this, 2015),  
        Premise::SMALLERTHAN, Premise::STANDARD, false);  
PREMISE(prNotClosestToGoalY2, this->robot->atPosY, new Double(this, 4025),  
        Premise::GREATERTHAN, Premise::STANDARD, false);
```

```
// verifica se o jogador está próx. ao gol considerando o Attribute atClosestToGoal
```

```
PREMISE(prClosestToGoal, this->robot->atClosestToGoal, true,  
        Premise::EQUAL, Premise::STANDARD, false);
```

```
// verifica se bola fora da área do time adversário
```

```
PREMISE(prBallOutsideEnemyTeamArea,  
        this->robot->atBallOutsideEnemyTeamArea, true, Premise::EQUAL,  
        Premise::STANDARD, false);
```

```
// verifica se bola está dentro da área válida de jogo
```

```
PREMISE(prBallInsideValidGameArea,  
        this->robot->atBallOutsideEnemyTeamArea, false, Premise::EQUAL,  
        Premise::STANDARD, false);
```

```
// verifica se o movimento em X é permitido
```

```
PREMISE(prAllowMoveX, this->robot->atRobotGoingValidX, true,  
        Premise::EQUAL, Premise::STANDARD, false);
```

```
// verifica se o jogador não irá se mover par a área de ataque do goleiro adversário
```

```
PREMISE(prRobotNextMoveNotInsideAttackArea,  
        this->robot->atNextMoveToAttackArea, false, Premise::EQUAL,  
        Premise::STANDARD, false);
```

```
// verifica se o jogador está com uma função/papel ativo
```

```
PREMISE(prActiveRole, this->robot->atRole, new String(this, "OFF"),  
        Premise::DIFFERENT, Premise::STANDARD, false);
```



## Implementações realizadas

---

```
5. void StrategyPON::initPremises():
    // verificar se o jogador excedeu o número máximo de toques na bola
    PREMISE(prMaxDrible, this->robot->atDribleCount,
            this->robot->atMaxDribleCount, Premise::GREATERTHAN,
            Premise::STANDARD, false);

    // verifica se o movimento em Y é permitido
    PREMISE(prAllowMoveY, this->robot->atRobotGoingValidY, true,
            Premise::EQUAL, Premise::STANDARD, false);

    // verifica se o jogador é o goleiro
    PREMISE(prRobotGoalkeeper, this->robot->atGoalkeeper, true, Premise::EQUAL,
            Premise::STANDARD, false);

    // verifica se o jogador não é o goleiro
    PREMISE(prRobotNotGoalkeeper, this->robot->atGoalkeeper, false,
            Premise::EQUAL, Premise::STANDARD, false);

    // verifica se o jogador irá se mover par a área de ataque do goleiro adversário
    PREMISE(prRobotNextMoveInsideAttackArea,
            this->robot->atNextMoveToAttackArea, true, Premise::EQUAL,
            Premise::STANDARD, false);

    // verifica se o jogador não irá se mover para a área defesa do goleiro
    PREMISE(prRobotNextMoveNotInsideDefenceArea,
            this->robot->atNextMoveToDefenceArea, false, Premise::EQUAL,
            Premise::STANDARD, false);

    // verifica se o jogador irá se mover para a área defesa do goleiro
    PREMISE(prRobotNextMoveInsideDefenceArea,
            this->robot->atNextMoveToDefenceArea, true, Premise::EQUAL,
            Premise::STANDARD, false);

    // verifica se o jogador pertence ao time azul
    PREMISE(prTeamBlue, this->robot->atIsBlu, true, Premise::EQUAL,
            Premise::STANDARD, false);

    // verifica se o jogador pertence ao time amarelo
    PREMISE(prTeamYellow, this->robot->atIsBlu, false, Premise::EQUAL,
            Premise::STANDARD, false);

    // verifica se o jogador não recebeu um cartão vermelho
    PREMISE(prRobotNotRedCard, this->robot->atActiveRedCard, false,
            Premise::EQUAL, Premise::STANDARD, false);

    // verifica se o jogador recebeu um cartão vermelho
    PREMISE(prRobotRedCard, this->robot->atActiveRedCard, true,
            Premise::EQUAL, Premise::STANDARD, false);
```

## Implementações realizadas

---

6. `void StrategyPON::initMethodPointers():`  
*// altera o valor do Attribute "atClosestToGoal" para true*  
`mtClosestToGoal = new MethodPointer<StrategyPON>(this,  
 &StrategyPON::closestToGoal);`  
  
*// altera o valor do Attribute "atClosestToGoal" para false*  
`mtNotClosestToGoal = new MethodPointer<StrategyPON>(this,  
 &StrategyPON::notClosestToGoal);`  
  
*// altera as coordenadas para o movimento de um jogador normal*  
`mtFixRobotNotGkMove = new MethodPointer<StrategyPON>(this,  
 &StrategyPON::fixGeneralRobotNextMove);`  
  
*// altera as coordenadas para o movimento de um goleiro*  
`mtFixGkMove = new MethodPointer<StrategyPON>(this,  
 &StrategyPON::fixGkNextMove);`  
  
*// processa o recebimento de commando de cartão vermelho pelo time*  
`mtRedCardYellow = new MethodPointer<StrategyPON>(this,  
 &StrategyPON::redCardYellowAction);`
7. *Methods associados as Instigation:*  
`void StrategyPON::closestToGoal(){  
 this->robot->atClosestToGoal->setValue(true);  
}`  
  
`void StrategyPON::closestToGoal(){  
 this->robot->atClosestToGoal->setValue(true);  
}`  
  
`void StrategyPON::notClosestToGoal(){  
 this->robot->atClosestToGoal->setValue(false);  
}`  
  
`void StrategyPON::fixGeneralRobotNextMove() {  
 Point* pos = new Point(this->robot->posToGo->getPointX(),  
 this->robot->posToGo->getPointY());  
 if(std::abs(pos->getPointY()) < 1500) {  
 if(pos->getPointX() > 2400)  
 pos->setPointX(2400);  
 else if(pos->getPointX() < -2400)  
 pos->setPointX(-2400);  
 }  
 this->robot->setPosToGo(pos);  
}`

## Implementações realizadas

---

### 7. *Methods* associados as *Instigation*:

```
void StrategyPON::fixGkNextMove() {
    Point* pos = new Point(this->robot->posToGo->getPointX(),
        this->robot->posToGo->getPointY());
    if(std::abs(pos->getPointY()) < 1500) {
        if(this->robot->getFieldSide() == LEFT) {
            if(pos->getPointX() > 2400) {
                pos->setPointX(2400);
            }
        }
        else {
            if(pos->getPointX() < -2400) {
                pos->setPointX(-2400);
            }
        }
    }
    this->robot->setPosToGo(pos);
}

void StrategyPON::redCardYellowAction() {
    printf("Cartao Vermelho to yellow\n");
    this->robot->team->checkNewRedCard();
}

void StrategyPonAT::gkPenalty() {
    printf("Penalty\n");
    double y = this->robot->penaltyGoalPoint;
    double x = this->robot->getEnemyGoal()->getPointX();

    if (rand() % 2 )
        y *= -1;

    Point *target = new Point ( x, y );

    double angle = this->robot->getBallPosition()->calculateAngleTo (target);
    Point *position = new Point ( this->robot->getBallPosition()->getPointX() -
        ROBOT_RADIUS_ERROR * cos ( angle ),
        this->robot->getBallPosition()->getPointY() - ROBOT_RADIUS_ERROR *
        sin(angle));

    printf("id %d\n", this->robot->getId());
    printf("Pos 1 Penalty x %f, y %f\n", x, y);
    printf("Pos 2 Penalty x %f, y %f\n", position->getPointX(),
        position->getPointY());
    this->robot->setPosToGo(position);
    this->robot->setTarget ( target );
    this->robot->mapAngularVelocity();
    this->robot->sendCommand();
}
```

## Implementações realizadas

---

### 7. *Methods* associados as *Instigation*:

```
void StrategyPON::limiteHorizontal(){
    cout << "Limite horizontal do campo atingido - Robo:" << this->robot->getId()
    cout << endl;

    if (this->robot->getFieldSide() == LEFT) {
        this->robot->setPosToGo(new Point(0.0, 0.0));
    } else {
        this->robot->setPosToGo(new Point(0.0, 0.0));
    }

    this->robot->moveToPosToGo();
}

void StrategyPON::limiteVertical(){
    cout << "Limite vertical do campo atingido - Robo:" << this->robot->getId()
    cout << endl;

    if (this->robot->getFieldSide() == LEFT) {
        this->robot->setPosToGo(new Point(0.0, 0.0));
    } else {
        this->robot->setPosToGo(new Point(0.0, 0.0));
    }

    this->robot->moveToPosToGo();
}

void StrategyPON::gkYellowCardBlue(){
    printf("Cartao Amarelo\n");
    this->robot->resetRobot();
}

void StrategyPON::gkRedCardBlue(){
    printf("Cartao Vermelho to blue\n");
    this->robot->team->checkNewRedCard();
}

void StrategyPonAT::shootToGoal(){
    Logger::log(Logger::INFO, this, __PRETTY_FUNCTION__);
    double y = this->robot->getEnemyGoal()->getPointY();
    double x = this->robot->getEnemyGoal()->getPointX();

    // Reset dribble count when shooting ball
    this->robot->atDribbleCount->setValue(0);
    this->robot->setTarget ( this->robot->getEnemyGoal() );
    this->robot->setPosToGo(robot->getBallPosition());
    this->robot->calculateKickWithZ();
    this->robot->atHasShot->setValue(true);
}
```

## Implementações realizadas

---

```
8. void StrategyPON::initRules():
    // se está próximo ao gol (coordenadas “x” e “y”) então define o Attribute
    // “atClosestToGoal” como true
    RULE(rlClosestToGoal, scheduler, Condition::CONJUNCTION);
    rlClosestToGoal->addPremise(prClosestToGoalX);
    rlClosestToGoal->addPremise(prClosestToGoalY1);
    rlClosestToGoal->addPremise(prClosestToGoalY2);
    rlClosestToGoal->addInstigation(INSTIGATION(this->mtClosestToGoal));

    // se não está próximo ao gol na coordenada “x” então define o Attribute
    // “atClosestToGoal” como false
    RULE(rlNotClosestToGoalX, scheduler, Condition::SINGLE);
    rlNotClosestToGoalX->addPremise(prNotClosestToGoalX);
    rlNotClosestToGoalX->addInstigation(INSTIGATION(
        this->mtNotClosestToGoal));

    // se não está próximo ao gol na coordenada “y” então define o Attribute
    // “atClosestToGoal” como false
    RULE(rlNotClosestToGoalY, scheduler, Condition::DISJUNCTION);
    rlNotClosestToGoalY->addPremise(prNotClosestToGoalY1);
    rlNotClosestToGoalY->addPremise(prNotClosestToGoalY2);
    rlNotClosestToGoalY->addInstigation(
        INSTIGATION(this->mtNotClosestToGoal));

    // se estou pronto, a bola é minha e estou próximo ao gol então chuto para o gol
    RULE(rlClosestToGoalKick, scheduler, Condition::CONJUNCTION);
    rlClosestToGoalKick->addPremise(prReadyToShoot);
    rlClosestToGoalKick->addPremise(prBallIsMine);
    rlClosestToGoalKick->addPremise(prClosestToGoal);
    rlClosestToGoalKick->addInstigation(INSTIGATION(this->mtShootToGoal));

    // Reset Kick and Dribble when ball is far. “Rule 5” (SANTOS, 2017, p. 179).
    RULE(rlBallFar, scheduler, Condition::CONJUNCTION);
    rlBallFar->addPremise(prBallIsFar);
    rlBallFar->addPremise(prActiveRole);
    rlBallFar->addInstigation(INSTIGATION(this->mtResetKick));

    // confirma necessidade de movimentação em X
    RULE(rlRobotWantsToMoveX, scheduler, Condition::CONJUNCTION);
    rlRobotWantsToMoveX->addPremise(prRobotMoveX);
    rlRobotWantsToMoveX->addPremise(prAllowMoveX);

    // confirma necessidade de movimentação em Y
    RULE(rlRobotWantsToMoveY, scheduler, Condition::CONJUNCTION);
    rlRobotWantsToMoveY->addPremise(prRobotMoveY);
    rlRobotWantsToMoveY->addPremise(prAllowMoveY);
```

## Implementações realizadas

---

```
8. void StrategyPON::initRules():
    // confirma necessidade de movimentação
    RULE(rlRobotWantsToMove, scheduler, Condition::DISJUNCTION);
    rlRobotWantsToMove->addMasterRule(rlRobotWantsToMoveX);
    rlRobotWantsToMove->addMasterRule(rlRobotWantsToMoveY);

    // this Rule should be extended to add NEGATED cases that could restrain normal
    // movement
    RULE(rlRobotNotRetrained, scheduler, Condition::DISJUNCTION);
    rlRobotNotRetrained->addPremise(prRobotNotRedCard);

    // Goalkeeper move almost freely
    RULE(rlRobotMoveGoalKeeper, scheduler, Condition::CONJUNCTION);
    rlRobotMoveGoalKeeper->addMasterRule(rlRobotWantsToMove);
    rlRobotMoveGoalKeeper->addMasterRule(rlRobotNotRetrained);
    rlRobotMoveGoalKeeper->addPremise(prRobotNextMoveNotInsideAttackArea);
    rlRobotMoveGoalKeeper->addPremise(prRobotGoalkeeper);
    rlRobotMoveGoalKeeper->addInstigation(INSTIGATION(this->mtExecuteMove));

    //confirma movimentação para áreas restritas
    RULE(rlNextMoveToRestrictedArea, scheduler, Condition::DISJUNCTION);
    rlNextMoveToRestrictedArea->addPremise(prRobotNextMoveInsideAttackArea);
    rlNextMoveToRestrictedArea->addPremise(prRobotNextMoveInsideDefenceArea);

    // robot cant move inside areas of attack
    RULE(rlRobotMove, scheduler, Condition::CONJUNCTION);
    rlRobotMove->addMasterRule(rlRobotWantsToMove);
    rlRobotMove->addPremise(prRobotNotGoalkeeper);
    rlRobotMove->addPremise(prRobotNextMoveNotInsideAttackArea);
    rlRobotMove->addPremise(prRobotNextMoveNotInsideDefenceArea);
    rlRobotMove->addMasterRule(rlRobotNotRetrained);
    rlRobotMove->addInstigation(INSTIGATION(this->mtExecuteMove));

    // se um jogador normal se mova para uma área restrita então seu movimento será
    // ajustado
    RULE(rlFixRobotNotGkmove, scheduler, Condition::CONJUNCTION);
    rlFixRobotNotGkmove->addMasterRule(rlNextMoveToRestrictedArea);
    rlFixRobotNotGkmove->addPremise(prRobotNotGoalkeeper);
    rlFixRobotNotGkmove->addInstigation(
        INSTIGATION(this->mtFixRobotNotGkMove));

    // se um goleiro se mover para uma área restrita para ele então seu movimento será
    // ajustado
    RULE(rlFixRobotGkmove, scheduler, Condition::CONJUNCTION);
    rlFixRobotGkmove->addPremise(prRobotGoalkeeper);
    rlFixRobotGkmove->addPremise(prRobotNextMoveInsideAttackArea);
    rlFixRobotGkmove->addInstigation(INSTIGATION(this->mtFixGkMove));
```

## Implementações realizadas

---

```
8. void StrategyPON::initRules():
    // se um jogador do time amarelo receber o comando de cartão vermelho então
    // este avisará o controlador do time que distribuirá o cartão
    RULE(rlRedCardYellow, scheduler, Condition::CONJUNCTION);
    rlRedCardYellow->addPremise(prRedCardYellow);
    rlRedCardYellow->addPremise(prTeamYellow);
    rlRedCardYellow->addInstigation(INSTIGATION(this->mtRedCardYellow));

    // se a bola sair do campo no lado do time adversário então o jogador voltará para
    // sua posição inicial
    RULE(rlBallOutsideEnemyTeam, scheduler, Condition::CONJUNCTION);
    rlBallOutsideEnemyTeam->addPremise(prBallOutsideEnemyTeamArea);
    rlBallOutsideEnemyTeam->addInstigation(INSTIGATION(this->mtResetRobot));
    rlBallOutsideEnemyTeam->addInstigation(
        INSTIGATION(this->mtMoveStopPos));

    // mover atacante para a posição de pênalty
    RULE(rlPenalty, scheduler, Condition::SINGLE);
    rlPenalty->addPremise(prPenalty);
    rlPenalty->addInstigation(INSTIGATION(this->mtExecutePenalty));

    // tratamento de cartão vermelho do time azul
    RULE(rlRedCardBlue, scheduler, Condition::SINGLE);
    rlRedCardBlue->addPremise(prRedCardBlue);
    rlRedCardBlue->addInstigation(INSTIGATION(this->mtRedCardBlue));

    // tratamento de cartão amarelo do time azul
    RULE(rlYellowCardBlue, scheduler, Condition::CONJUNCTION);
    rlYellowCardBlue->addPremise(prYellowCardBlue);
    rlYellowCardBlue->addPremise(prTeamBlue);
    rlYellowCardBlue->addInstigation(INSTIGATION(this->mtYellowCardBlue));

    // tratamento do limite horizontal do campo
    RULE(rlLimiteHorizontal, scheduler, Condition::DISJUNCTION);
    rlLimiteHorizontal->addPremise(prLimiteHorizontalMax);
    rlLimiteHorizontal->addPremise(prLimiteHorizontalMin);
    rlLimiteHorizontal->addInstigation(INSTIGATION(this->mtLimiteHorizontal));

    // tratamento do limite vertical do campo
    RULE(rlLimiteVertical, scheduler, Condition::DISJUNCTION);
    rlLimiteVertical->addPremise(prLimiteVerticalMax);
    rlLimiteVertical->addPremise(prLimiteVerticalMin);
    rlLimiteVertical->addInstigation(INSTIGATION(this->mtLimiteVertical));

    // chuta bola em direção ao gol
    RULE(rlShootToGoal, scheduler, Condition::CONJUNCTION);
    rlShootToGoal->addPremise(prReadyToShoot);
    rlShootToGoal->addPremise(prBallIsMine);
    rlShootToGoal->addInstigation(INSTIGATION(this->mtShootToGoal));
```

## Implementações realizadas

---

```
8. void StrategyPON::initRules():
    // seta flag que está preparado para um chute ao gol quando está
    RULE(rlPositionToShoot, scheduler, Condition::CONJUNCTION);
    rlPositionToShoot->addPremise(prNotReadyToShoot);
    rlPositionToShoot->addPremise(prGameIsStarted);
    rlPositionToShoot->addPremise(prHasNotObstacleForward);
    rlPositionToShoot->addPremise(prBallIsMine);
    rlPositionToShoot->addPremise(prPositionedBehindBall);
    rlPositionToShoot->addPremise(prBallCloseToEnemyGoal);
    rlPositionToShoot->addPremise(prBallInsideValidGameArea);
    rlPositionToShoot->addInstigation(INSTIGATION(
        this->mtSetReadyToShootTrue));

    // reseta flag de chute ao gol
    RULE(rlResetShootFlag, scheduler, Condition::CONJUNCTION);
    rlResetShootFlag->addPremise(prReadyToShoot);
    rlResetShootFlag->addPremise(prBallIsNotMine);
    rlResetShootFlag->addInstigation(INSTIGATION(
        this->mtSetReadyToShootFalse));

    // verifica se o número máximo de toques foi excedido e força a passar a bola
    RULE(rlMaxDrible, scheduler, Condition::CONJUNCTION);
    rlMaxDrible->addPremise(prGameIsStarted);
    rlMaxDrible->addPremise(prBallIsMine);
    rlMaxDrible->addPremise(prMaxDrible);
    rlMaxDrible->addPremise(prRobotIsNotSetToPassBall);
    rlMaxDrible->addPremise(prBallInsideValidGameArea);
    rlMaxDrible->addInstigation(INSTIGATION(this->mtPositionPassBall));

    // passa a bola caso o jogador de defesa esteja com a posse muito próximo ao gol
    RULE(rlPassGkCloseGoal, scheduler, Condition::CONJUNCTION);
    rlPassGkCloseGoal->addPremise(prGameIsStarted);
    rlPassGkCloseGoal->addPremise(prBallIsMine);
    rlPassGkCloseGoal->addPremise(prBallCloseTeamGoal);
    rlPassGkCloseGoal->addPremise(prRobotIsNotSetToPassBall);
    rlPassGkCloseGoal->addInstigation(INSTIGATION(this->mtGkPassBall));
```



9. Definição da função/papel do jogador:

```
StrategyPON* RobotPON::defineStrategy(Robot::PlayPosition positionPlayer) {
    switch(positionPlayer) {
        case Robot::Defender:
            atGoalkeeper->setValue(true);
            setatRole("GOALKEEPER");
            return new StrategyPonDF(this);
        case Robot::LeftSide:
            setatRole("DEFENDER_LEFT");
            return new StrategyPonLE(this);
        case Robot::RightSide:
            setatRole("DEFENDER_RIGHT");
            return new StrategyPonLD(this);
        case Robot::Forward:
            setatRole("STRIKER");
            return new StrategyPonAT(this);
        default:
            setatRole("DEFENDER_ONLY");
            return new StrategyPonLE(this);
    }
}
```

10. Modificação na função drible para contar o número de toques:

```
void StrategyPON::dribble() {
    RobotPON* robot = this->robot;
    Point* ballPosition = robot->getBallPosition();
    Point* target = new Point(ballPosition->getPointX(),
                              ballPosition->getPointY());

    if (robot->getFieldSide() == LEFT) {
        target->setPointX(ballPosition->getPointX() + 200);
    } else {
        target->setPointX(ballPosition->getPointX() - 200);
    }

    robot->atDribleCount->setValue(1+robot->atDribleCount->getValue());
    robot->setTarget(target);
    robot->calculateKickWithZ();
    robot->calculateDribble();
    robot->setPosToGo(ballPosition);
    robot->atSetToReceiveBall->setValue(false);
}
```

11. Determinar se o robô deve passar a bola e retornar ao centro do campo.

*// Para Rule PassWhenoFF*

if (this->robot->getFieldSide() == LEFT)

    PREMISE(prAheadOpponentField, this->robot->atPosX, new Double(this, 1500),

    Premise::GREATERTHAN, Premise::STANDARD, false);

else PREMISE(prAheadOpponentField, this->robot->atPosX, new Double(this, -1500),

    Premise::SMALLERTHAN, Premise::STANDARD, false);

*// Para Rule PassWhenoFF*

PREMISE(prOutOfCenterUp, this->robot->atPosY, new Double(this, 300),

    Premise::GREATERTHAN, Premise::STANDARD, false);

PREMISE(prOutOfCenterDown, this->robot->atPosY, new Double(this, -300),

    Premise::SMALLERTHAN, Premise::STANDARD, false);

*// Para Rule PassWhenoFF*

*// Rule: check if robot is out of center*

RULE(rlOutOfCenter, scheduler, Condition::DISJUNCTION);

rlOutOfCenter->addPremise(prOutOfCenterUp);

rlOutOfCenter->addPremise(prOutOfCenterDown);

*// Rule PassWhenoFF*

*// Rule: pass ball when ahead of opponent field and off center*

RULE(rlPassWhenOff, scheduler, Condition::CONJUNCTION)

rlPassWhenOff->addMasterRule(rlOutOfCenter);

rlPassWhenOff->addPremise(prGameIsStarted);

rlPassWhenOff->addPremise(prBallIsMine);

rlPassWhenOff->addPremise(prAheadOpponentField);

rlPassWhenOff->addInstigation(INSTIGATION(this->mtPositionPassBall));

*// Rule: Ball is mine. Let's dribble!*

RULE(rlDribble, scheduler, Condition::CONJUNCTION);

rlDribble->addPremise(prBallIsMine);

rlDribble->addPremise(prGameIsStarted);

rlDribble->addPremise(prPositionedBehindBall);

rlDribble->addPremise(prHasNotObstacleForward);

rlDribble->addPremise(prRobotIsNotSetToPassBall);

rlDribble->addInstigation(INSTIGATION(this->mtDribble));

## APÊNDICE E: *RULES* IMPLEMENTADAS NO PROJETO “ROBOCUP2012” PELO ALUNO LUIS HENRIQUE SANT’ANA

Aqui estão relatadas as implementações realizadas pelo aluno “Luis Henrique Sant’Ana” de novas *Rules* ou comportamentos.

### 1 ROBÔ FINALIZADOR DIRIGE-SE AO MEIO DO CAMPO

Percebeu-se que robôs com posse de bola que se encaminham ao gol adversário não se posicionam adequadamente. Logo, buscou-se criar uma *Rule* para que o robô que estiver avançado no campo adversário, com posse de bola e fora do centro do campo, passe a bola para que tenha a chance de se reposicionar para possivelmente receber a bola novamente quando melhor posicionado para marcação de um gol.

A *Rule* “rlDribble” precisou ser alterada. A *Premisse* “prRobotIsNotSetToPassBall” foi adicionada, que verifica que o robô não está setado para passar a bola. Assim, se tiver que passar a bola, ele não irá continuar a conduzir a bola. Antes a alternância de uma tarefa para outra ocorria apenas pela presença de um obstáculo.

Adicionou-se então a *Rule* “rlPassWhenOff” com as seguintes *Premises*: “prGameIsStarted”, “prBallIsMine” e “prAheadOpponentField”. Esta última teve de ser criada. Ela checa, dependendo do lado ao qual o robô está jogando, se o seu posicionamento no eixo X é maior do 1500mm ou menor do que 1500mm.

Também foi necessário adicionar a *Rule* “rlPassWhenOff” uma regra mestra para acomodar a checagem de *Premises* através de disjunção. A *Rule* criada foi a “rlOutOfCenter”, que realiza operação lógica OU com as *Premises* “prOutOfCenterUp” e “prOutOfCenterDown”. Cada uma, respectivamente, checa se o robô está posicionado, no eixo Y, acima de 300mm ou abaixo de -300mm.

A *Instigation* consiste do *Method* “mtPositionPassBall”. Isso faz com que o robô passe a bola e tenha a chance de se reposicionar para receber a bola e marcar um gol depois.

Importante destacar que as constantes presentes nas *Premises* tiveram de ser criadas de tal maneira, como por exemplo: *new Double(this, -300)*. Caso contrário, o erro “segmentation fault (core dumped)” ocorre após a seleção de um time para o jogo. O código-fonte implementado está apresentado no Quadro 11.

**Quadro 11 – Partes relevantes do arquivo “StrategyPON.cpp” para a Rule “rlPassWhenOff”**

```
// Para Rule PassWhenoFF
if (this->robot->getFieldSide() == LEFT)
    PREMISE(prAheadOpponentField, this->robot->atPosX,new Double(this, 1500),
        Premise::GREATERTHAN,Premise::STANDARD,false);
else PREMISE(prAheadOpponentField, this->robot->atPosX,new Double(this, -1500),
    Premise::SMALLERTHAN,Premise::STANDARD,false);

// Para Rule PassWhenoFF
PREMISE(prOutOfCenterUp,this->robot->atPosY,new Double(this, 300),
    Premise::GREATERTHAN,Premise::STANDARD,false);
PREMISE(prOutOfCenterDown,this->robot->atPosY,new Double(this, -300),
    Premise::SMALLERTHAN,Premise::STANDARD,false);

// Para Rule PassWhenoFF
// Rule: check if robot is out of center
RULE(rlOutOfCenter, scheduler, Condition::DISJUNCTION);
rlOutOfCenter->addPremise(prOutOfCenterUp);
rlOutOfCenter->addPremise(prOutOfCenterDown);

// Rule PassWhenoFF
// Rule: pass ball when ahead of opponent field and off center
RULE(rlPassWhenOff,scheduler,Condition::CONJUNCTION)
rlPassWhenOff->addMasterRule(rlOutOfCenter);
rlPassWhenOff->addPremise(prGameIsStarted);
rlPassWhenOff->addPremise(prBallIsMine);
rlPassWhenOff->addPremise(prAheadOpponentField);
rlPassWhenOff->addInstigation(INSTIGATION(this->mtPositionPassBall));

// Rule: Ball is mine. Let's dribble!
RULE(rlDribble, scheduler, Condition::CONJUNCTION);
rlDribble->addPremise(prBallIsMine);
rlDribble->addPremise(prGameIsStarted);
rlDribble->addPremise(prPositionedBehindBall);
rlDribble->addPremise(prHasNotObstacleForward);
rlDribble->addPremise(prRobotIsNotSetToPassBall);
rlDribble->addInstigation(INSTIGATION(this->mtDribble));
```

**Fonte: Autoria própria.**

## 2 MOVIMENTAÇÃO DE DEFESA

Percebeu-se a necessidade de melhoria da movimentação dos jogadores de linha durante o ataque. As diferentes versões do *Method* “attackMove” presentes nos códigos “StrategyPonAT.cpp”, “StrategyPonLD.cpp” e “StrategyPonLE.cpp” foram modificados com essa finalidade.

Além da maior mobilidade proporcionada especialmente no eixo Y, foram feitas alterações de modo a evitar entrada nas áreas de gols quando este *Method* for executado. Os códigos-fontes das diferentes versões do *Method* “attackMove” podem ser observados no Quadro 12.

**Quadro 12 – Method “attackMove” presente nos diferentes códigos de acordo com o posicionamento do jogador**

(continua)

```
void StrategyPonAT::attackMove() {
    const int Y_DIST = 500;
    const int Y_AREA = 600;
    const int X_DIST = 500;
    const int X_AREA = 2500;
    int xPos;
    int yPos;
    Point* ballPosition = this->robot->getBallPosition();

    if (this->robot->getFieldSide() == LEFT) {
        xPos = ballPosition->getPointX() + X_DIST;
    } else {
        xPos = ballPosition->getPointX() - X_DIST;
    }

    xPos = xPos > X_AREA ? X_AREA : xPos;
    xPos = xPos < -X_AREA ? -X_AREA : xPos;
    yPos = ballPosition->getPointY();
    yPos = yPos > 0 ? yPos - Y_DIST : yPos + Y_DIST;
    yPos = yPos < -Y_AREA ? -Y_AREA : yPos;
    yPos = yPos > Y_AREA ? Y_AREA : yPos;

    this->robot->setPosToGo(new Point(xPos, yPos));
    this->robot->setTarget(this->robot->getBallPosition());
}

void StrategyPonLD::attackMove() {
    const int Y_DIST = 800;
    const int Y_MIN = 400;
    const int Y_AREA = 600;
    const int Y_MAX = 1800;
    const int X_DIST = 800;
    const int X_MIN = 2800;
    const int X_AREA = 2500;
    const int X_MAX = 2800;
    int xPos;
    int yPos;
    Point* ballPosition = this->robot->getBallPosition();

    if (this->robot->getFieldSide() == LEFT) {
        xPos = ballPosition->getPointX() + X_DIST;
        xPos = xPos > X_MAX ? X_MAX : xPos;
        xPos = xPos < -X_MIN ? -X_MIN : xPos;
        yPos = ballPosition->getPointY() - Y_DIST;
        yPos = yPos < -Y_MAX ? -Y_MAX : yPos;
        yPos = yPos > -Y_MIN ? -Y_MIN : yPos;
        if (xPos > X_AREA || xPos < -X_AREA) {
            yPos = yPos > -Y_AREA ? -Y_AREA : yPos;
        }
    }
}
```

**Quadro 12 – Method “attackMove” presente nos diferentes códigos de acordo com o posicionamento do jogador**

(conclusão)

```
else {
    xPos = ballPosition->getPointX() - X_DIST;
    xPos = xPos < -X_MAX ? -X_MAX : xPos;
    xPos = xPos > X_MIN ? X_MIN : xPos;
    yPos = ballPosition->getPointY() + Y_DIST;
    yPos = yPos > Y_MAX ? Y_MAX : yPos;
    yPos = yPos < Y_MIN ? Y_MIN : yPos;
    if(xPos > X_AREA || xPos < -X_AREA){
        yPos = yPos < Y_AREA ? Y_AREA : yPos;
    }
}

this->robot->setPosToGo(new Point(xPos, yPos));
this->robot->setTarget(this->robot->getBallPosition());
}

void StrategyPonLE::attackMove() {
    const int Y_DIST = 800;
    const int Y_MIN = 400;
    const int Y_AREA = 600;
    const int Y_MAX = 1800;
    const int X_DIST = 800;
    const int X_MIN = 2800;
    const int X_AREA = 2500;
    const int X_MAX = 2800;
    int xPos;
    int yPos;
    Point* ballPosition = this->robot->getBallPosition();

    if (this->robot->getFieldSide() == LEFT) {
        xPos = ballPosition->getPointX() + X_DIST;
        xPos = xPos > X_MAX ? X_MAX : xPos;
        xPos = xPos < -X_MIN ? -X_MIN : xPos;
        yPos = ballPosition->getPointY() + Y_DIST;
        yPos = yPos > Y_MAX ? Y_MAX : yPos;
        yPos = yPos < Y_MIN ? Y_MIN : yPos;
        if(xPos > X_AREA || xPos < -X_AREA){
            yPos = yPos < Y_AREA ? Y_AREA : yPos;
        }
    } else {
        xPos = ballPosition->getPointX() - X_DIST;
        xPos = xPos < -X_MAX ? -X_MAX : xPos;
        xPos = xPos > X_MIN ? X_MIN : xPos;
        yPos = ballPosition->getPointY() - Y_DIST;
        yPos = yPos < -Y_MAX ? -Y_MAX : yPos;
        yPos = yPos > -Y_MIN ? -Y_MIN : yPos;
        if(xPos > X_AREA || xPos < -X_AREA){
            yPos = yPos > -Y_AREA ? -Y_AREA : yPos;
        }
    }

    this->robot->setPosToGo(new Point(xPos, yPos));
    this->robot->setTarget(this->robot->getBallPosition());
}
```

**Fonte: Autoria própria.**

### 3 MOVIMENTAÇÃO DE DEFESA

Percebeu-se a necessidade de melhoria da movimentação dos jogadores de linha durante a defesa. As 3 (três) versões do *Method* “defenceMove” presentes nos códigos “StrategyPonAT.cpp”, “StrategyPonLD.cpp” e “StrategyPonLE.cpp” foram modificadas com essa finalidade. O *Method* não havia sido implementada no código do robô atacante, logo, foi criada.

Além da maior mobilidade proporcionada especialmente no eixo Y, foram feitas alterações de modo a evitar entrada nas áreas de gols quando este *Method* for executado. As diferentes versões do código-fonte do *Method* “defenceMove” podem ser observadas no Quadro 13.

**Quadro 13 – *Method* “defenceMove” presente nos diferentes códigos de acordo com o posicionamento do jogador**

(continua)

```
void StrategyPonAT::defenceMove() {
    const int Y_DIST = 500;
    const int Y_AREA = 600;
    const int X_DIST = 500;
    const int X_AREA = 2500;
    int xPos, yPos;
    Point* ballPosition = this->robot->getBallPosition();
    if (this->robot->getFieldSide() == LEFT)
        xPos = ballPosition->getPointX() - X_DIST;
    else xPos = ballPosition->getPointX() + X_DIST;
    xPos = xPos > X_AREA ? X_AREA : xPos;
    xPos = xPos < -X_AREA ? -X_AREA : xPos;
    yPos = ballPosition->getPointY();
    yPos = yPos > 0 ? yPos - Y_DIST : yPos + Y_DIST;
    yPos = yPos < -Y_AREA ? -Y_AREA : yPos;
    yPos = yPos > Y_AREA ? Y_AREA : yPos;
    this->robot->setPosToGo(new Point(xPos, yPos));
    this->robot->setTarget(this->robot->getBallPosition());
}

void StrategyPonLD::defenceMove() {
    const int Y_DIST = 800;
    const int Y_MIN = 400;
    const int Y_AREA = 600;
    const int Y_MAX = 1800;
    const int X_DIST = 800;
    const int X_MIN = 2800;
    const int X_AREA = 2500;
    const int X_MAX = 2800;
    int xPos, yPos;
    Point* ballPosition = this->robot->getBallPosition();
    if (this->robot->getFieldSide() == LEFT) {
        xPos = ballPosition->getPointX() - X_DIST;
        xPos = xPos > X_MAX ? X_MAX : xPos;
        xPos = xPos < -X_MIN ? -X_MIN : xPos;
        yPos = ballPosition->getPointY() - Y_DIST;
        yPos = yPos < -Y_MAX ? -Y_MAX : yPos;
```

**Quadro 13 – Method “defenceMove” presente nos diferentes códigos de acordo com o posicionamento do jogador**

(conclusão)

```

yPos = yPos > -Y_MIN ? -Y_MIN : yPos;
if(xPos > X_AREA || xPos < -X_AREA){
    yPos = yPos > -Y_AREA ? -Y_AREA : yPos;
}
} else {
    xPos = ballPosition->getPointX() + X_DIST;
    xPos = xPos < -X_MAX ? -X_MAX : xPos;
    xPos = xPos > X_MIN ? X_MIN : xPos;
    yPos = ballPosition->getPointY() + Y_DIST;
    yPos = yPos > Y_MAX ? Y_MAX : yPos;
    yPos = yPos < Y_MIN ? Y_MIN : yPos;
    if(xPos > X_AREA || xPos < -X_AREA){
        yPos = yPos < Y_AREA ? Y_AREA : yPos;
    }
}
this->robot->setPosToGo(new Point(xPos, yPos));
this->robot->setTarget(this->robot->getBallPosition());
}

void StrategyPonLE::defenceMove() {
    const int Y_DIST = 800;
    const int Y_MIN = 400;
    const int Y_AREA = 600;
    const int Y_MAX = 1800;
    const int X_DIST = 800;
    const int X_MIN = 2800;
    const int X_AREA = 2500;
    const int X_MAX = 2800;
    int xPos;
    int yPos;
    Point* ballPosition = this->robot->getBallPosition();
    if (this->robot->getFieldSide() == LEFT) {
        xPos = ballPosition->getPointX() - X_DIST;
        xPos = xPos > X_MAX ? X_MAX : xPos;
        xPos = xPos < -X_MIN ? -X_MIN : xPos;
        yPos = ballPosition->getPointY() + Y_DIST;
        yPos = yPos > Y_MAX ? Y_MAX : yPos;
        yPos = yPos < Y_MIN ? Y_MIN : yPos;
        if(xPos > X_AREA || xPos < -X_AREA){
            yPos = yPos < Y_AREA ? Y_AREA : yPos;
        }
    } else {
        xPos = ballPosition->getPointX() + X_DIST;
        xPos = xPos < -X_MAX ? -X_MAX : xPos;
        xPos = xPos > X_MIN ? X_MIN : xPos;
        yPos = ballPosition->getPointY() - Y_DIST;
        yPos = yPos < -Y_MAX ? -Y_MAX : yPos;
        yPos = yPos > -Y_MIN ? -Y_MIN : yPos;
        if(xPos > X_AREA || xPos < -X_AREA){
            yPos = yPos > -Y_AREA ? -Y_AREA : yPos;
        }
    }
    this->robot->setPosToGo(new Point(xPos, yPos));
    this->robot->setTarget(this->robot->getBallPosition());
}

```

**Fonte: Autoria própria.**



## 4 MOVIMENTAÇÃO DE PARADA

Percebeu-se que apenas o atacante respondia corretamente ao comando de parada. Logo, as funções para as posições laterais foram melhoradas, além da do atacante. As funções dos goleiros foram programadas como mostrado no Quadro 14.

Tomou-se o cuidado de não permitir que os jogadores de linha entrem em áreas de gol, sendo que os goleiros permanecem dentro da área. Com relação às linhas de campo, porém, é permitida a passagem, visto que a bola não está em jogo quando o jogo está parado, como apresentado no código-fonte apresentado no Quadro 14.

**Quadro 14 – Method “moveToStopPosition”**

```
void StrategyPonAT::moveToStopPosition() {
    Logger::log(Logger::INFO, this, "StrategyPonAT::moveToStopPosition");
    Point *ballPosition = this->robot->getBallPosition();
    double angle = this->robot->calculateAngleTo();
    Point *position = new Point();
    if (this->robot->getFieldSide() == LEFT) {
        position->setPointX(ballPosition->getPointX() -
            this->robot->RADIUS * cos ( angle ));
        position->setPointY(ballPosition->getPointY() -
            this->robot->RADIUS * sin ( angle ));
    } else {
        position->setPointX(ballPosition->getPointX() +
            this->robot->RADIUS * cos ( angle ));
        position->setPointY(ballPosition->getPointY() +
            this->robot->RADIUS * sin ( angle ));
    }
    this->robot->setKick(0);
    this->robot->setDribble(0);
    this->robot->setTarget ( ballPosition );
    this->robot->setPosToGo( position );
}

void StrategyPonLD::moveToStopPosition() {
    Logger::log(Logger::INFO, this, "StrategyPonLD::moveToStopPosition");
    const int X_POS = 1400;
    const int Y_POS = 1400;
    if (this->robot->getFieldSide() == LEFT)
        this->robot->setPosToGo(new Point(-X_POS, -Y_POS));
    else this->robot->setPosToGo(new Point(X_POS, Y_POS));
    this->robot->setTarget(this->robot->getBallPosition());
}

void StrategyPonLE::moveToStopPosition() {
    Logger::log(Logger::INFO, this, "StrategyPonLE::moveToStopPosition");
    const int X_POS = 1400;
    const int Y_POS = 1400;
    if (this->robot->getFieldSide() == LEFT) {
        this->robot->setPosToGo(new Point(-X_POS, Y_POS));
    } else this->robot->setPosToGo(new Point(X_POS, -Y_POS));
    this->robot->setTarget(this->robot->getBallPosition());
}
```

**Fonte: Autoria própria.**

## 5 POSIÇÕES DE GOLEIRO

Notou-se que uma possível boa estratégia é manter o goleiro sempre dentro da área por ser o único jogador com permissão para tocar a bola nessa região. Logo, inicialmente, foram ajustadas suas posições de apresentação (agora mais próxima à área) e a posição de parada dentro da área, de forma que haja espaço para a bola dentro da área à sua frente. Isso foi realizado alterando os valores de posição em X nos *Methods* “initPONStrategy” e “moveToStopPosition” no arquivo “StrategyPonDF.cpp”.

Em seguida, alterando os valores de X nos *Methods* “attackMove” e “defenceMove”, foi feito que o goleiro se posicione sobre a linha do gol quando a bola estiver com o time adversário e pouco mais à frente quando com o próprio time. Finalmente, foi feito que ele acompanhe a bola no eixo Y quando com o time adversário, mas permaneça no centro da área quando com o próprio time. As alterações relevantes no código presente em “StrategyPonDF.cpp” podem ser observadas no código-fonte apresentado no Quadro 15.

**Quadro 15 – Alterações relevantes no arquivo “StrategyPonDF.cpp” para as novas movimentações de goleiro**

(continua)

```
void StrategyPonDF::initPONStrategy() {
    const int X_POS = 1700;
    const int Y_POS = 0;

    if (this->robot->getFieldSide() == LEFT) {
        this->robot->setPosToGo(new Point(-X_POS, Y_POS));
    } else {
        this->robot->setPosToGo(new Point(X_POS, Y_POS));
    }

    this->robot->setTarget(this->robot->getBallPosition());
}

void StrategyPonDF::moveToStopPosition() {
    Logger::log(Logger::INFO, this, "StrategyPonLD::moveToStopPosition");
    const int X_POS = 2500;
    const int Y_POS = 0;

    if (this->robot->getFieldSide() == LEFT) {
        this->robot->setPosToGo(new Point(-X_POS, Y_POS));
    } else {
        this->robot->setPosToGo(new Point(X_POS, Y_POS));
    }

    this->robot->setTarget(this->robot->getBallPosition());
}
```

**Quadro 15 – Alterações relevantes no arquivo “StrategyPonDF.cpp” para as novas movimentações de goleiro**

(conclusão)

```
void StrategyPonDF::attackMove() {
    const int X_POS = 2700;
    const int Y_POS = 0;
    int xPos;
    int yPos;

    if (this->robot->getFieldSide() == LEFT) {
        xPos = -X_POS;
    } else {
        xPos = X_POS;
    }

    yPos = Y_POS;
    this->robot->setPosToGo(new Point(xPos, yPos));
    this->robot->setTarget(this->robot->getBallPosition());
}

void StrategyPonDF::defenceMove() {
    const int X_POS = 3000;
    const int Y_MAX = 240;
    int xPos;
    int yPos;
    Point* ballPosition = this->robot->getBallPosition();

    if (this->robot->getFieldSide() == LEFT) {
        xPos = -X_POS;
    } else {
        xPos = X_POS;
    }

    yPos = ballPosition->getPointY();
    yPos = yPos < -Y_MAX ? -Y_MAX : yPos;
    yPos = yPos > Y_MAX ? Y_MAX : yPos;
    this->robot->setPosToGo(new Point(xPos, yPos));
    this->robot->setTarget(this->robot->getBallPosition());
}
```

**Fonte: Autoria própria.**