UNIVERSIDADE DO ESTADO DA BAHIA
DEPARTAMENTO DE CIÊNCIAS EXATAS E DA TERRA
COLEGIADO DE SISTEMA DE INFORMAÇÃO

RAONI SALES DE OLIVEIRA

GERAÇÃO DE COMPORTAMENTO PARA ROBÔS DA LIGA DE SIMULAÇÃO 3D

SALVADOR
2018

RAONI SALES DE OLIVEIRA

GERAÇÃO DE COMPORTAMENTO PARA ROBÔS DA LIGA DE SIMULAÇÃO 3D

Monografia apresentada ao curso de Sistemas de Informação da Universidade do Estado da Bahia – UNEB, como requisito parcial para obtenção do grau de bacharel em Sistemas de Informação, área de concentração: Sistemas Multiagente.

Orientador: Profª. Drª. Ana Patrícia Magalhães.

SALVADOR

2018

RAONI SALES DE OLIVEIRA

GERAÇÃO DE COMPORTAMENTO PARA ROBÔS DA LIGA DE SIMULAÇÃO 3D

Monografia apresentada ao curso de Sistemas de Informação do Departamento de Ciências Exatas e da Terra da Universidade do Estado da Bahia – UNEB, como requisito parcial à obtenção do grau de bacharel em Sistemas de Informação, área de concentração: Sistemas Multiagente.

Aprovada em:

COMISSÃO EXAMINADORA

Profa. Dra.Ana Patrícia F. Magalhães Mascarenhas (Orientadora)
Universidade do Estado da Bahia – UNEB

Prof.Msc. Marco Antônio Costa Simões
Universidade do Estado da Bahia – UNEB

Prof.Ph.D Josemar Rodrigues de Souza
Universidade do Estado da Bahia– UNEB

AGRADECIMENTOS

Sou grato à aquele que me deu vida.Em minhas limitações e no máximo de minhas capacidades,posso transformar a realidade, porém aquele que cria com perfeição está muito além do meu alcance, e jamais vou compreender totalmente Sua magnitude. Que em sua bondade, Ele possa me ensinara conhecê-lo, pois somos Dele, viemos Dele, e voltaremos à Ele.

Gostaria de agradecer de forma especial às orientações da Professora Drª. Ana Patrícia Magalhães, pois sem ela, nada seria possível.

Agradeço a meu pai, que nunca desistiu de mim, e a minha mãe, que dedicou a vida à criação de seus filhos.

Agradeço a minha esposa amiga, companheira e serena sempre acho ela mais sábia do que imagina.

Resumo

A robótica e a inteligência artificial são áreas de pesquisa de grande interesse na atualidade, pois suas propostas visam, dentre outras coisas, substituir o indivíduo por robôs nas tarefas onde o fator humano pode representar alguma limitação ou condição de risco. Para fomentar as pesquisas nestas áreas, iniciativas tais como competições de futebol de robôs propõe desafios que estimulam pesquisadores e estudantes a desenvolverem projetos com potencial para serem depois aproveitados em cenários reais. Um exemplo de competição é a Robocup, o maior desafio que abrange às áreas de robótica e inteligência artificial da atualidade. Nele atua o time BahiaRT criado pelo centro de pesquisa ACSO/UNEB, cujo diferencial está na coordenação multiagente, em específico no que tange habilidades de marcação e passe de bola, se posicionando entre os 4 melhores do mundo na liga de simulação 3D nos últimos anos. Nesse sentido, o time BahiaRT tem realizado uma tarefa dispendiosa: criar novos comportamentos para os agentes, a serem utilizados nas ferramentas de apoio estratégico adotadas - *Framework* FCPortugal *Setplays* e SPlanner -. Esse trabalho propõe uma solução que dê suporte à criação de novos comportamentos para as ferramentas de apoio estratégico do time BahiaRT. A solução compreende uma interface para definição do comportamento e um gerador de código para integrar o novo comportamento gerado às ferramentas de apoio estratégico utilizadas pelo time BahiaRT. A solução apresentada neste trabalho foi testada no laboratório do ACSO e se mostrou viável para utilização.
Palavras-chave: Comportamento. Multiagente. ACSO. BahiaRT. Robocup.

Abstract

Robotics and artificial intelligence are areas of research of great interest today, because its proposals aim, among other things, to substitute the individual for robots in the tasks where the human factor can represent some limitation or condition of risk. To foster research in these areas, initiatives such as robot soccer competitions pose challenges that encourage researchers and students to develop projects with potential to be harnessed in real-life scenarios. An example of competition is Robocup, the biggest challenge in the areas of robotics and artificial intelligence today. In it, the BahiaRT team created by the ACSO / UNEB research center, whose differential is in multiagent coordination, specifically in marking and passing skills, ranking among the 4 best in the world in 3D simulation league in recent years. In this sense, the BahiaRT team has performed a costly task: to create new behaviors for the agents, to be used in the strategic support tools adopted - Framework FCPortugal Setplays and SPlanner -. This work proposes a solution that supports the creation of new behaviors for the BahiaRT team's strategic support tools. The solution comprises a behavior definition interface and a code generator to integrate the new behavior generated to the strategic support tools used by the BahiaRT team. The solution presented in this work was tested in the ACSO laboratory and proved to be feasible for use.
Key words: Behavior. Multiagent. ACSO. BahiaRT. Robocup.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVEATURAS E SIGLAS

ACSO	Centro de Arquitetura de Computadores e Sistemas Operacionais
BahiaRT	*Bahia Robotics Team*
DAI	*Distributed Artificial Inteligence*
DDM	Desenvolvimento Dirigido a Modelo
FFS	FCPortugal *Framework Setplay*
FIFA	Federação Internacional de Futebol Associação
FIPA-ACL	*Agent Communication Language of Foundation for Intelligent Physical Agents*
IA	Inteligência Artificial
JE	Jogada Ensaiada
M2M	*Model-to-Model*
M2T	*Model-to-Text*
MOF	*Meta-Object Facility*
MSL	*Middle Size League*
RoboCup	*Robot World Cup Initiative*
SMA	Sistemas Multiagente
SBSP	*Situation Based Strategic Positioning*
SL	*Semantic Language*
SPL	*Standard Platform League*
TCP	*Transmission Control Protocol*
UML	*Unified Modeling Language*
UNEB	Universidade do Estado da Bahia
XABSL	*Extensible Agent Behavior Specification Language*

SUMÁRIO

1. INTRODUÇÃO

A robótica e a inteligência artificial são áreas de pesquisa de grande interesse na atualidade, pois suas propostas visam, dentre outras coisas, substituir o indivíduo por robôs nas tarefas onde o fator humano pode representar alguma limitação ou condição de risco. Robôs podem executar tarefas de maneira individual ou coordenada com outros robôs. A coordenação de tarefas entre robôs caracterizam-se pelos Sistemas Multiagentes (SMA) [16], onde os robôs são considerados agentes autônomos dotados de inteligência artificial (IA) [28]. Neste contexto, pode-se citar como exemplo de pesquisas sendo desenvolvidas, os sistemas de veículos autônomos para cidades inteligentes e nano robôs que auxiliam em procedimentos medicinais.

Para fomentar as pesquisas nas áreas de IA e Robótica, iniciativas tais como competições de futebol de robôs propõe desafios que estimulam pesquisadores e estudantes a desenvolverem projetos com potencial para serem depois aproveitados em cenários reais, Um exemplo de competição é a Robocup [1], maior evento de futebol de robôs do mundo.

A Robocup reúne pesquisadores da área de robótica e IA, em uma competição anual de robôs autônomos, aberta à participantes de todo o mundo. Foi criada em 1997, conhecido como um ano de ruptura, marcado pelo triunfo no desafio de xadrez e pela realização da primeira edição oficial da Robocup. Essa ruptura representa o final de um desafio de 40 anos, cujo marco foi a vitória do computador *Deep Blue* da IBM, no jogo contra o campeão mundial de xadrez Garry Kasparov, e o início de um novo desafio para as próximas décadas, dirigido pela Robocup [1]. Face às possibilidades de atuação das tecnologias a serem desenvolvidas nas competições, atualmente a RoboCup possui uma divisão em ligas: *RoboCup Soccer, RoboCup Industrial, RoboCup Rescue, RoboCup@Home e RoboCup Junior* [3].

Dentro desse contexto, o Bahia *Robotics Team* (BahiaRT), criada pelo Centro de Arquitetura de Computadores e Sistemas Operacionais (ACSO, atua em dois dos principais eixos de competição da RoboCup: *RoboCup @Home* e *3D Soccer Simulation* (*Robocup Soccer*) [5].

O BahiaRT é um grupo de cooperação científica que tem como objetivo agregar pesquisadores, grupos de pesquisa do estado da Bahia e estudantes

interessados em pesquisas científicas na área de Robótica e Inteligência Artificial, e participar ativamente da iniciativa de pesquisa internacional conhecida como RoboCup [5].

A liga *RoboCup @Home* tem como objetivo desenvolver serviços e tecnologia de robô assistivo com alta relevância para futuras aplicações domésticas pessoais. As categorias de Robocup *Soccer* simuladas foram criadas originalmente para promover pesquisas sobre jogos de futebol em um nível mais alto do que era possível no mundo real, devido às restrições contemporâneas da tecnologia robótica, e trabalhar técnicas que posteriormente fossem aplicadas a liga de robôs físicos. A atual *3D Soccer Simulation*, em que o BahiaRT está inserido, é uma evolução da categoria 2D de futebol simulado e faz parte da liga *Robocup Soccer*.

Na *Robocup Soccer,* cada robô que integra o time é visto como um agente, que deve contribuir, em conjunto com os demais que compõem o time, para atingir um objetivo em comum. Esse comportamento, dentro do domínio de inteligência artificial é denominado sistema multiagente [1]. Para atingir esse resultado, cada robô em campo é dotado com uma série de sensores, capaz de avaliar a situação atual do mundo, e definir qual é a melhor estratégia a ser tomada pelo time. A ação de cada agente deve estar coordenada com outros em campo, para que interesses individuais não entrem em conflito e possam atingir o propósito geral da equipe que pode ser, por exemplo, interceptar um ataque, realizar uma jogada ou fazer o gol.

Ações multiagente coordenadas, denominadas *setplay,* expressam claramente o esforço conjunto de uma equipe em obter um resultado em comum. Por exemplo, o movimento colaborativo de um conjunto de carros autônomos ou de um conjunto de nano robôs de medicina poderia ser realizado utilizando *setplays*. Em analogia ao futebol convencional de seres humanos, para os robôs da Robocup Soccer um *setplay* pode ser visto como uma jogada ensaiada (JE) entre os participantes de um time.

O trabalho desenvolvido em [6], buscou fortalecer a capacidade do ACSO de elaborar *setplays* integrando o código do time BahiaRT ao FCPortugal *Framework Setplay* (FFS) em conjunto com seu módulo gráfico de JEs, o SPlanner, dentro do domínio de futebol de robôs. Pensando no s*etplay* como um conjunto de ações que buscam atingir um objetivo, um comportamento pode ser visto como uma das ações

que irão compor esse conjunto, por exemplo, correr, driblar e chutar. Atualmente a criação de novos comportamentos para os agentes do time BahiaRT é uma tarefa dispendiosa, pois envolve modificar uma série de códigos dos arquivos que compõem o FFS e a ferramenta gráfica SPlanner.

Face à integração realizada em [6], e aos trabalhos futuros descritos em [7], uma nova demanda foi identificada: a necessidade de facilitar a criação de novos comportamentos para os agentes do time BahiaRT, tendo em vistas uma adaptação maior às inovações tecnológicas, que permitem que o robô realize uma variedade maior de movimentos em campo. Para contribuir com essa demanda, este trabalho propõe uma solução que auxilie a definição de novos comportamentos para o FFS e seu módulo de JEs SPlanner, que compreende identificar a forma como o comportamento deve ser modelado, os códigos envolvidos em sua definição, bem como a construção de uma ferramenta que automatize esse processo, tornando-o mais eficiente e seguro.

Aplicar uma solução que auxilie a criação de novos comportamentos por meio de geração de código automática, tende a diminuir drasticamente o tempo de desenvolvimento, independente do grau de maturidade que o usuário possua com relação ao ambiente do time BahiaRT.

Em geral, o futebol de robô no BahiaRT é conduzido por alunos de graduação que, normalmente, permanecem entre 1 e 2 anos no projeto. Sempre que um novo aluno integra o grupo é preciso dedicar tempo em aquisição de conhecimento. A solução aqui proposta visa, dentre outros benefícios, potencializar ganhos de produtividade e minimizar a *experise* necessária nas ferramentas envolvidas (FFS e SPlanner) no desenvolvimento de inovações voltadas para a coordenação dos agentes em campo, análise e desenvolvimento de novas JEs.

Elevando o nível de abstração do processo de definição dos novos comportamentos que comporão as jogadas, é possível diminuir o tempo de treinamento dos alunos para essa tarefa - que atualmente tem demonstrado alta curva de aprendizado -, por meio de um sistema que gere código automaticamente, abrevie o tempo de desenvolvimento e produza códigos de maior qualidade. Adicionalmente, o projeto deve contribuir, por meio de uma ferramenta de *log*, com o entendimento de todo o processo realizado pelo sistema, detalhando e

transparecendo as linhas de inserção de código, local dos arquivos alterados, quantidade de métodos criados, dentre outros elementos essenciais à didática de aprendizado dos alunos.

Este trabalho se divide em 9 capítulos: O capítulo 2 apresenta a Robocup e suas ligas. O capítulo 3 contextualiza estratégia, abordando sistemas multiagente e estratégia em futebol de robôs. O capítulo 4 apresenta as ferramentas de JEs utilizadas pelo time BahiaRT. No capítulo 5, são expostos os trabalhos relacionados. O capítulo 6 explica a metodologia de desenvolvimento do trabalho. O capítulo 7 detalha a solução proposta. O capítulo 8 apresenta a validação da solução proposta. No capítulo 9 são feitas as considerações finais.

2. A ROBOCUP

A idéia inicial de utilizar robôs no jogo de futebol partiu do professor Alan Mackworth, da Universidade de British Columbia, Canadá, em seu artigo "*On Seeing Robots*" de 1992 [2]. Após passar por anos de maturação da idéia, a primeira *Robot World CupInitiative* (RoboCup) oficial foi realizada em 1997, com a intenção inicial de utilizar o futebol na promoção de ciência e tecnologia, e dentro desse domínio, tratar grandes desafios em inteligência artificial de forma mais atraente ao público [9]. O jogo de xadrez também foi utilizado para tratar grandes desafios em inteligência artificial de uma maneira mais popular e amigável, e dentro das nuanças desse desafio, muitos algoritmos de busca foram avaliados e desenvolvidos, com ganhos dentro e fora do domínio de jogos de xadrez [2].

O objetivo de criar robôs que joguem *football* contra seres humanos vai muito além do estado atual da arte, no entanto a meta foi estipulada para as próximas cinco décadas [52]. O projeto cresce e a cada ano recebe mais participantes, com desdobramentos em outras áreas de atuação as atividades de pesquisa estão divididas em cinco eixos principais: RoboCup *Soccer*, RoboCup *Industrial*, RoboCup *Rescue*, RoboCup@*Home* e RoboCup *Junior* [3]. Cada eixo abrange um tópico central de pesquisa, que vai desde o desenvolvimento de soluções para o segmento industrial até simulação de técnicas que serão posteriormente aplicadas a robôs físicos [4].

Para garantir destaque no evento, áreas dentro e fora dos domínios de competição precisam ser desenvolvidas pelas equipes participantes, promovendo tópicos de pesquisa que contemplam comportamento reativo, aquisição de estratégia, aprendizado de máquina, planejamento em tempo real, sistemas multiagente, controle de sensores [9], visão computacional, sistemas de comunicação e locomoção, além de diversas aplicações diretas das tecnologias desenvolvidas, como por exemplo, em prótese, órtese e vigilância [1].

A cada ano é definido um país para a realização do evento. Dentre os lugares que já sediaram competições podemos citar as cidades Melbourne na Austrália (2000), Lisboa em Portugal (2004) e João Pessoa no Brasil (2014), recebendo, por evento, mais de 4.000 pessoas de mais de 40 países, com cerca de 500 times nos últimos anos [8].

2.1 Robocup *Soccer*

Visto como o principal eixo das competições da Robocup, e a que atrai um maior número de fãs, a Robocup *Soccer* é o projeto pioneiro desse evento internacional, assumindo uma meta bastante desafiadora: em meados do século 21 uma equipe totalmente autônoma de robôs humanóides jogadores de futebol deve vencer um jogo contra o time de humanos campeão da última Copa do Mundo da FIFA, utilizando as regras da FIFA [2].

A Figura 1 mostra uma partida da liga *Standard Platform League* (SPL), que passou a utilizar robôs humanóides a partir de 2008 [57]. Até então, as equipes da liga SPL competiam utilizando robôs Sony AIBO [58] de quatro pernas. Essa mudança mostrou um avanço tecnológico importante, em direção à meta desafiadora da Robocup. A característica principal da liga SPL é a utilização de robôs idênticos em todas as equipes.

Figura 1 - Campeonato Robocup Soccer 2017.

Fonte: Site oficial da Robocup [49].

A RoboCup *Soccer* é o projeto mais conhecido e o foco desse trabalho de conclusão de curso. As pesquisas desse eixo tratam de robôs autônomos, ou seja, aqueles que tomam decisões com base em um sistema de inteligência artificial [1]. O comportamento de agir ou executar ações, caracteriza cada robô um agente [62]. Na

Robocup *Soccer*, cada agente interage e coopera com os outros robôs em campo, compondo um sistema multiagente, que busca atingir um objetivo em comum: vencer o time adversário.

Existem características específicas, que apontam singularidades próprias à cada uma das ligas da Robocup *Soccer*. São divididas em cinco, que abordam questões distintas em robótica e inteligência artificial:

- *Small Size League*
- *Middle Size League* (MSL)
- *Standard Platform League* (SPL)
- *Humanoid League*
- *Soccer Simulation League*

A *Soccer Simulation League* é dividida em subligas de simulação 2D e 3D. A subliga 2D possui a mesma quantidade de jogadores da 3D, contudo os robôs 2D são mais simples - no formato de esferas bidimensionais - e muitos atributos mecânicos não são representados em seus movimentos, como por exemplo, o ângulos de rotação das articulações dos membros inferiores dos robôs bípedes da 3D, que podem gerar perda de equilíbrio e queda do robô. A subliga 3D, que o time BahiaRT do ACSO atua será melhor abordada na seção 2.2.

2.2. A subliga de Simulação 3D

Este trabalho tem como objetivo contribuir com atividades de pesquisa e desenvolvimento, voltadas para competições da categoria *3D Soccer Simulation*. Criada em 2004 [6], a subliga 3D trouxe um realismo maior para a categoria de futebol simulado. Isto se deve ao fato de ser adicionada a terceira dimensão, criando mais similaridades com o futebol real, surgindo diferentes estratégias e abordagens que em 2D eram impossíveis [11].

No entanto, até o ano de 2006 a representação dos agentes 3D simulados se limitava a esferas 3D no campo, o principal marco desse ano foi a simulação de agentes 3D humanóides, com modelos simples do Fujitsu HOAP-2 [12]. Mais tarde, em 2008, os robôs Fujitsu HOAP-2 foram substituídos por modelos do robô NAO, criado pela *Aldebaran Robotics* [13], o mesmo utilizado na SPL. A Figura 2 ilustra

uma partida da subliga de simulação 3D, que utiliza o mesmo modelo de robô ilustrado na Figura 1.

Figura 2 - 3D Soccer Simulation 2017.

Fonte: Site do Austin Villa *team* [50].

Inicialmente a liga de simulação surgiu com a intenção de abstrair problemas de *hardware*, resolvendo também o problema com o custo de aquisição de robôs para participar das ligas de robôs físicos. Atualmente, a simulação 3D se aproxima bastante da realidade, permitindo desenvolver e testar, além dos códigos de IA, códigos de *hardware*, como por exemplo, o controle de movimentação, que posteriormente podem ser aplicados de forma direta aos robôs NAO da SPL, que utiliza robôs similares à 3D [48].

A interação entre simulador e jogador (agente) se dá por meio de uma arquitetura cliente servidor, na qual cada agente em campo é um cliente que se conecta ao servidor por meio de *sockets* TCP (*Transmission Control Protocol*) [63][64]. Por ser um sistema em tempo real o simulador trabalha com intervalos de tempo uniformes também chamados ciclos [44]. Na execução de um ciclo o programa cliente especifica as ações que o jogador deve executar e em retorno recebe as informações oriundas dos sensores [15].

Na Robocup um jogador possui três sensores, o aural, que detecta mensagens enviadas pelos jogadores e pelo árbitro, o visual que detecta as informações visuais do campo e o corporal que detecta o estado físico do jogador, como o ângulo da cabeça e a velocidade [44].

O servidor *Soccer Server* garante um ambiente de simulação multiagente, suportando 22 agentes autônomos interagindo em uma partida de futebol em tempo real [14]. Inclui também alguns elementos como ruído, movimentação de objetos, e comunicação limitada, com intuito de tornar o ambiente mais próximo das complexidades do mundo real [56].

3. COORDENAÇÃO E ESTRATÉGIA EM FUTEBOL DE ROBÔS

Coordenação multiagente e planejamento estratégico são dois dos maiores tópicos de pesquisa dentro do contexto da Robocup [16]. Esta seção faz uma breve descrição desses tópicos: a seção 3.1 trata de sistemas multiagente, e a seção 3.2 trata de estratégia dentro do domínio da Robocup *Soccer*.

3.1 Sistemas Multiagente

Artigos publicados na década de 70 já tratavam de uma área específica de inteligência artificial denominada *distributed artificial inteligence* (DAI) [17]. Mais tarde, na década de 80, surgiram pesquisas [18][19] no campo de Sistemas Multiagente (SMA), uma abordagem mais moderna da DAI.

Tecnicamente, SMA é uma subárea da DAI, o principal motivo dessa divisão se dá ao fato da DAI possuir uma abrangência maior em seu espectro de aplicação, possuindo dois tipos principais de sistemas: Sistemas de resolução de problemas distribuídos, que foca na resolução de problemas complexos, decomposição de tarefas e síntese de solução [20]; e SMA, que focam na coordenação comportamental.

Métodos de coordenação em SMA são aplicados devido às relações de dependência entre as tarefas dos agentes [21]. Segundo [32], Coordenação é necessária ou desejada por várias razões possíveis:

- Relações de dependência entre agentes;
- Conhecimento e recursos, ou seja, necessidade de coordenar ações, pois nenhum agente possui todo o conhecimento ou recursos para resolver a tarefa;
- Restrições globais como tempo, custo e recursos.

Uma vez que tentam alcançar seus próprios objetivos com a ajuda ou o impedimento de outros [28], cada agente deve planejar quando e como interagir com quem [20]. Esse planejamento, em SMA, pode ser centralizado, nos casos em que um agente central desenvolve planos para um grupo que executa as tarefas; ou distribuído, quando um grupo de agentes é responsável por fazer esse planejamento [27].

Todos os agentes que operam por esse viés abandonam as impressões de um agente único - baseadas em informações usuais sobre objetos no mundo, pré-condições e efeitos apenas de suas próprias ações -, e passam a operar também sobre o que outros agentes acreditam e sobre o que eles podem fazer [19].

Tarefas de natureza distribuída podem ser otimizadas por um sistema cooperativo, que gera e executa planos em conjunto, fazendo inferências e estabelecendo comunicação mútua [19]. Para isso, o agente deve ser capaz de incluir em seu raciocínio, de maneira apropriada, conhecimento e planos de outros agentes [18], utilizando uma base de conhecimento em comum, assim como um protocolo de comunicação adequada [17].

Levando em conta a necessidade coordenação multiagente, a ser alcançada através de negociação ou cooperação para atingir objetivos em comum, objetivos distintos ou objetivos opostos [25], ao longo dos anos trabalhos importantes foram realizados desenvolvendo algumas linguagens padronizadas de comunicação multiagente, a exemplo da linguagem *Agent Communication Language* da *Foundation for Intelligent Physical Agents* (FIPA-ACL) [59], que possui uma série de protocolos que especificam, por exemplo, negociação, requisição e busca de informações [36].

3.2 Estratégia em futebol de robôs

A estratégia faz parte do mundo desportivo há bastante tempo. Tão relevante quando jogadores habilidosos sem ela um time de estrelas pode perder facilmente para outro qualquer. Cada estilo de jogo precisa adotar uma estratégia diferente, assimilando e incorporando regras, objetivos e quantidade de jogadores. No futebol de robôs não é diferente, pois é dividido em ligas e apesar de objetivos similares - fazer o gol -, cada uma possui particularidades que influenciam no comportamento do time e na estratégia durante a partida.

Algumas ligas possuem características que tornam o desenvolvimento estratégico primário [11], como por exemplo, a subliga *Adultsize* da *humanoid league*, que realiza partidas de um contra um [37]. Poucos agentes no campo em conjunto com dificuldades sensoriais e motoras, tendem a dificultar táticas ofensivas, como ocorre em muitas partidas SPL [11]. Ao abstrair algumas dessas dificuldades

mecânicas e realizar partidas 11 contra 11, as ligas simuladas tem alcançado resultados melhores em desenvolvimento estratégico e iteração multiagente.

Para desenvolver melhor as habilidades estratégicas do time, métodos como o de posicionamento espacial SBSP, de definição de comportamento, como a linguagem XABSL, e de suporte tático, como agente *coach* foram elaborados.

O método *Situation Based Strategic Positioning* (SBSP), calcula o posicionamento estratégico dos jogadores em campo, nos momentos em que o agente percebe que não está, ou não irá entrar em uma situação de comportamento ativo em breve [34]. Esse método compõe o mecanismo de distinção entre situações ativas e estratégicas da equipe de simulação FCPortugal [39], podendo ser observado nas descrições mais recentes do time para a liga de simulação 3D [38], onde o jogador só abandona a posição estratégica quando entra em um comportamento crítico, como posse e recuperação de bola. Uma adaptação desse método também pôde ser observada nas descrições mais recentes do time CAMBADA, da liga de robôs físicos *MiddleSize* [39].

A *Extensible Agent Behavior Specification Language* (XABSL) é uma linguagem que descreve comportamentos de agentes autônomos, utilizando um grafo de opções. Partindo de uma opção denominada *root* muitas outras opções são definidas e cada uma representa uma máquina de estados finitos, onde cada estado consiste em uma ação a ser tomada [41]. As hierarquias das máquinas de estado finito tornam o sistema modular, e asseguram o reuso de comportamentos em diferentes contextos [42]. Recentemente, uma variação da XABSL, a linguagem C-*based Agent Behavior Specification Language* (CABSL) [40], baseada em C++ [41], foi utilizada por uma das campeãs da SPL, a B-*Human.*

O conceito de agente *coach* pode ser citado como um método similar ao papel do treinador nas partidas reais de *football* [31], o *coach* é caracterizado como um agente autônomo, fora do campo, com capacidade de observar e analisar o jogo, sem controlar nenhum agente, ele desempenha o papel de conselheiro do time [35]. Em 1999, as primeiras equipes [45] começaram a utilizar um agente *coach* dando suporte tático. Dentre outras, essas são algumas de suas atribuições [44][35][43]:

- Receber informações globais sem erros, sobre os objetos em campo;
- Enviar mensagens táticas para os jogadores;

- Sugerir papéis;
- Ajustar a formação do time.

Um grande desafio superado dentro desse contexto foi a definição de uma linguagem de comunicação que permite ao agente *coaching*, de forma simples, comunicar-se com a sua equipe, possibilitando alterar a tática do jogo de acordo com sua análise estratégica [32].

A linguagem *Clang* [32], desenvolvida pela comunidade internacional da Robocup, denominada linguagem *standard* [34] utilizada pelo *soccerserver*, incorporou uma série de conceitos originais da linguagem de alto nível *Coaching Unilang* [46], como regiões e comportamentos. Atualmente as duas linguagens, em suas versões mais recentes, são bastante semelhantes [33].

4. FERRAMENTAS DE JOGADAS ENSAIADAS

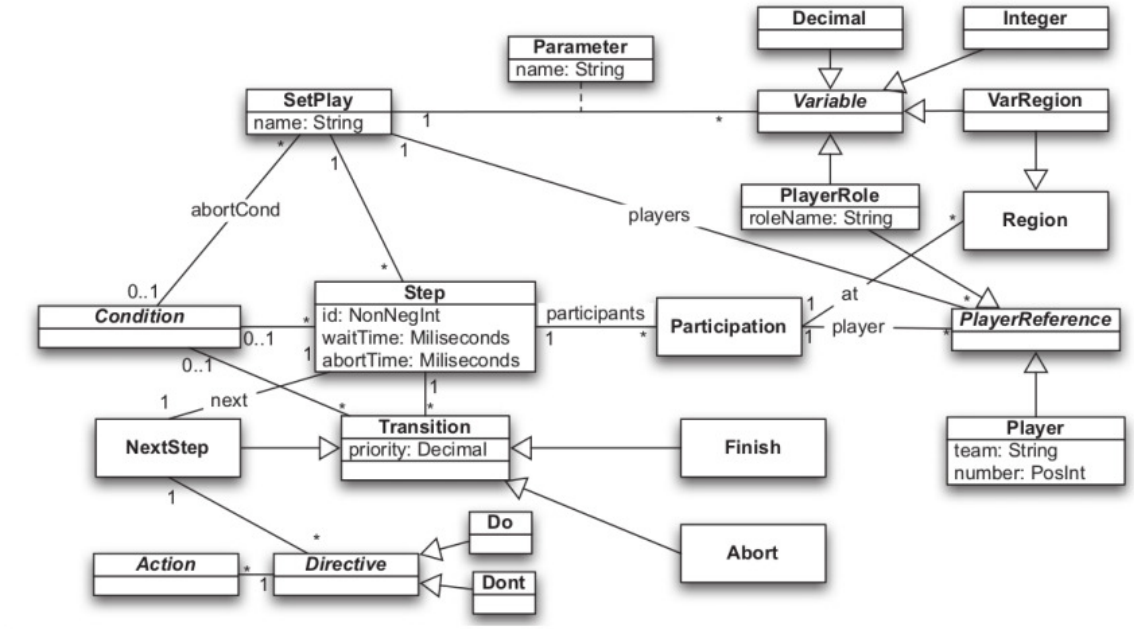
Atualmente o ambiente do time BahiaRT utiliza duas ferramentas principais na criação de jogada ensaiada (JE), o FFS, a ser explicado na seção 4.1, e seu módulo gráfico SPlanner, a ser explicado na seção 4.2.

4.1 Framework FCPortugal Setplays

Mota e Reis [16] ressaltam a falta de pesquisas nos tópicos estratégia e coordenação multiagente, por iniciativas que busquem atender de forma mais ampla as diferentes ligas presentes na Robocup, uma vez que foi percebida a tendência das tecnologias desenvolvidas serem aplicáveis a apenas uma liga específica. O *Framework* FCPortugal *Setplays* (FFS), surge então com a motivação de integrar estratégias multiagente desenvolvidas na Robocup, para que uma jogada criada possa ser compartilhada seguindo um padrão de desenvolvimento que possibilite a interpretação e execução por qualquer jogador de forma flexível e independente de liga, por meio de um *Framework* de definição e execução de *Setplays*.

O *framework* foi inicialmente desenvolvido na equipe FC Portugal e disponibilizado como *software* livre. Atualmente está disponível na *Source Forge* [55] como projeto de nome *fcportugalsetplays*. O FFS possui um conjunto completo de conceitos e mecanismos de execução necessários para atingir de forma efetiva comportamentos colaborativos multiagente [22].

Figura 3 - Estrutura geral de um *setplay* no FFS.



Fonte: Mota e Reis [22].

As definições a seguir estão de acordo com uma das principais fontes relacionadas ao FFS [22]. Conforme mostra a estrutura representada na Figura 3:

Um *setplay* é definido por um nome e possui uma lista de jogadores participantes (*PlayerReference*), identificando aqueles que fazem parte do *setplay*, podendo referenciar diretamente cada jogador (*Player)* ou um papel (*PlayerRole),* por exemplo, atacante e defensor. Possui também uma lista de passos (S*tep*), identificados por um *id*, um número inteiro não negativo. As variáveis *waitTime* e *abortTime* fazem parte do controle de execução: *waitTime* corresponde ao tempo que o jogador precisa esperar em um passo para passar para o passo seguinte, *abortTime* é o tempo limite para abortar o *setplay*, caso não seja possível passar para o próximo passo.

Os parâmetros (*Parameter*) de um *setplay* podem conter dados simples, como um inteiro ou decimal ou dados espaciais, como pontos e regiões. *Condition* representa as condições que precisam ser satisfeitas para entrar em um *step* e *Transition*, condições para passar de um *step* para outro (*NextStep*), finalizar (*finish*) ou abortar (*abort*) o *setplay*. Caso uma *Trasition* seja *NextStep*, deve possuir diretivas *Do* e *Dont*, representando ações (*Action*) que o jogador deve ou não

executar. *Action* representa conceitos abstratos que modelam comportamentos que um jogador pode executar, como por exemplo, driblar, receber a bola e marcar.

As classes *Actions, Conditions* e *Regions* utilizados no FS foram importados da linguagem Clang [32]. Na classe *Actions*, por exemplo, que representa os comportamentos do agente em campo, todas as ações definidas na *Clang* foram importadas com seus nomes originais e sintaxe similar [7][34]. A Tabela 1 mostra um panorama dessas ações e suas funcionalidades:

Tabela 1 - Ações importadas da linguagem *Clang*.

Ação	**Função**
Position	toma uma região como argumento. O executor deve se posicionar naquela região.
Forwardtoregion	toma uma região como argumento. A bola deve ser encaminhada para aquela região.Existe um argumento opcional, chamado *tipo*, com valores *normais*, *rápidos* e *lentos* quanto possível, descrevendo o tipo de chute que deve ser empregado.
Passto player	toma a referência de um jogador como argumento. Instrui o executor a passar a bola para aquele jogador. Da mesma forma que a ação de encaminhamento, existe um argumento opcional, *tipo*, com valores semelhantes aos descritos na ação anterior.
Mark player	toma a referência de um jogador como argumento, que deve ser marcada pelo executor.
Mark pass line to player	toma a referência de um jogador como argumento. Instrui o executor a marcar a linha de passe para aquele jogador.
Mark pass line to region	toma uma região como argumento. O executor deve marcar a linha de passagem para aquela região.
Set Offside line	toma uma região como argumento. Instrui o executor a definir uma linha de impedimento naquela região.
Dribble	toma uma região como argumento. Instrui o executor a driblar a bola para aquela região.
Clear	toma uma região opcional como argumento. Instrui o executor a limpar a bola daquela região.
Shoot	sem qualquer argumento. Instrui o executor a chutar no gol.
Hold	Sem qualquer argumento. Instrui o executor a segurar a bola no local atual.
Intercept	sem qualquer argumento, instrui o executor a interceptar ativamente a bola.

Tackle	pega uma referência de jogador (oponente) como argumento, que presumivelmente segura a bola e deve ser abordada pelo executor.

Fonte: Tabela adaptada ao conteúdo de [34].

Adicionalmente, as ações da Tabela 2 tiveram que ser criadas pelo autor do FFS, para suprir as necessidades das diferentes ligas da Robocup [34][7]:

Tabela 2 - Ações criadas pelo autor do *framework*.

Ação	Função
ReceiveBall	Recebe um passe de outro jogador, que é, implicitamente, o dono da bola. Não aceita argumentos.
AttentionTo	Direciona a atenção visual e/ou auditiva a um determinado objeto ou região, tomada como argumento.
Sequence (sec)	É simplesmente um agregador de ações, que são dadas como argumentos, e devem ser executadas em sequência.
MarkGoal	Posicionar em um local adequado, para evitar que chutes do adversário entrem no gol.
MoveToOffSideLine	Mover-se perto da linha de impedimento, evitando ao mesmo tempo cruzá-la. O argumento indica a coordenada y do local desejado.
Stop	Parar o mais rápido possível, certificando-se de que qualquer inércia restante seja adequadamente contrabalançada.

Fonte: Tabela adaptada ao conteúdo de [7].

4.1.1 Integração com o time BahiaRT

A criação da interface necessária à integração dos códigos do time BahiaRT com os componentes do FFS, foi realizada em [6], de acordo com a metodologia para times que desejam utilizar o *framework* estabelecida em [16]. As etapas seguidas para a criação da interface de integralização, de acordo com [6], foram:

- Implementação das classes abstratas *Conditions* e *Actions* no código existente dos agentes do time com intuito de incorporar ao *framework* as informações mapeadas no modelo de mundo do agente, assim como, suas limitações no que diz respeito a habilidades e ações;

- Alteração na mensagem atual de comunicação entre os agentes para incluir os elementos para controle da execução da jogada;
- Implementação do gerenciador de execução do *setplay*.

4.2 SPlanner

A ferramenta SPlanner foi desenvolvida em 2011, fruto da tese de mestrado de João Cravo, pela Faculdade de Engenharia da Universidade do Porto, em parceria com Luís Reis [11], seu orientador e co-autor do FFS. O nome escolhido para a aplicação, SPlanner, deriva de *Strategy Planner*, ou seja, um planejador de estratégia. Splanner surge então tencionando atingir os alguns objetivos [11]:

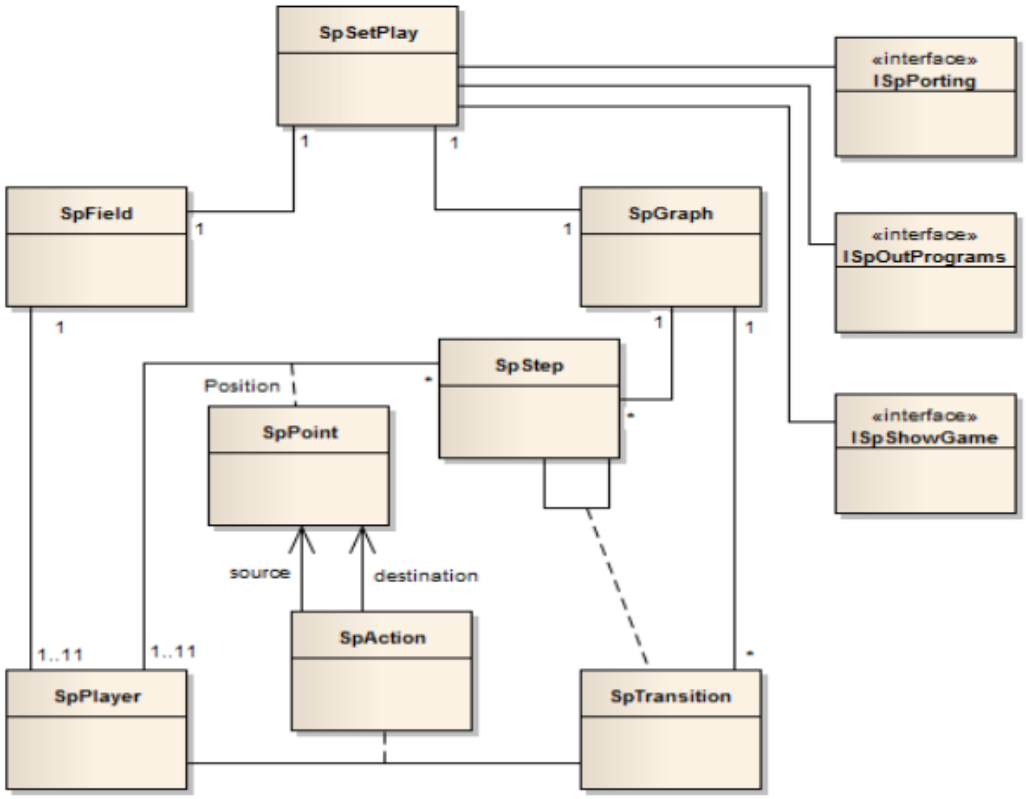
- Desenhar JEs que respeitem a gramática do FFS;
- Prover uma Interface limpa, intuitiva, favorecendo o recurso *drag-and-drop*;
- Importar e exportar JEs respeitando a gramática do FFS;
- Efetuar facilmente teste e *debug*, inicialmente para a liga de simulação 2D;
- Disponibilizar a visualização de um *log* de jogo ao mesmo tempo em que se desenha uma JE;
- Prover integração fácil com os módulos do FFS;
- Prover extensão para outras ligas além da 2D;
- Ser funcional em Linux.

Experimentos foram realizados no ACSO usando o SPlanner, com resultados relatados em [6], graças a interface de integração apresentada em 4.1.1, que permitiu que as JEs modeladas no módulo gráfico do FFS, o SPlanner, sejam automaticamente reconhecidas e utilizadas pelo time BahiaRT.

4.2.1 Arquitetura do Splanner

A arquitetura do SPlanner obedece alguns requisitos: extensível, modular, livre de conexão com servidores ou internet [11], além de ser independente do FFS, permitindo que outras equipes da Robocup ajustem-na de acordo com suas necessidades.Uma versão adaptada do SPlanner pode ser observada em [53]. O diagrama de classes pode ser visualizado na Figura 4:

Figura 4 - Diagrama de classes do módulo de desenho de JEs.



Fonte: João Cravo [11].

Com relação aos conceitos expostos na Figura 3, entre a ferramenta SPlanner e o FFS é possível fazer as seguintes analogias conceituais, de acordo com a Tabela 3:

Tabela 3 - Relações conceituais entre as classes do *FFS* e da ferramenta *SPlanner*.

FFS	SPlanner	Descrição
Setplay	SpSetplay	Representa e conserva todos os elementos pertencentes a uma JE.
PlayerReference	SpPlayer	Representação do jogador. No SPlanner têm ainda o encargo de desenhar o jogador no terreno de jogo.
Step	SpStep	Representação de um passo da JE. No SPlanner têm ainda o encargo de desenhar o passo no grafo.
Transition	SpTrasition	Representação da transição entre passos (SpSteps)

		da JE. No SPlanner têm ainda o encargo de desenhar a transição no grafo.
Action	SpAction	Representação de uma ação do jogador. No SPlanner têm ainda o encargo de desenhar a ação no terreno de jogo.
Region	SpPoint	Representação da posição de jogadores ou ações.

Fonte: João Cravo [11].

As outras classes do diagrama seguem as seguintes atribuições:

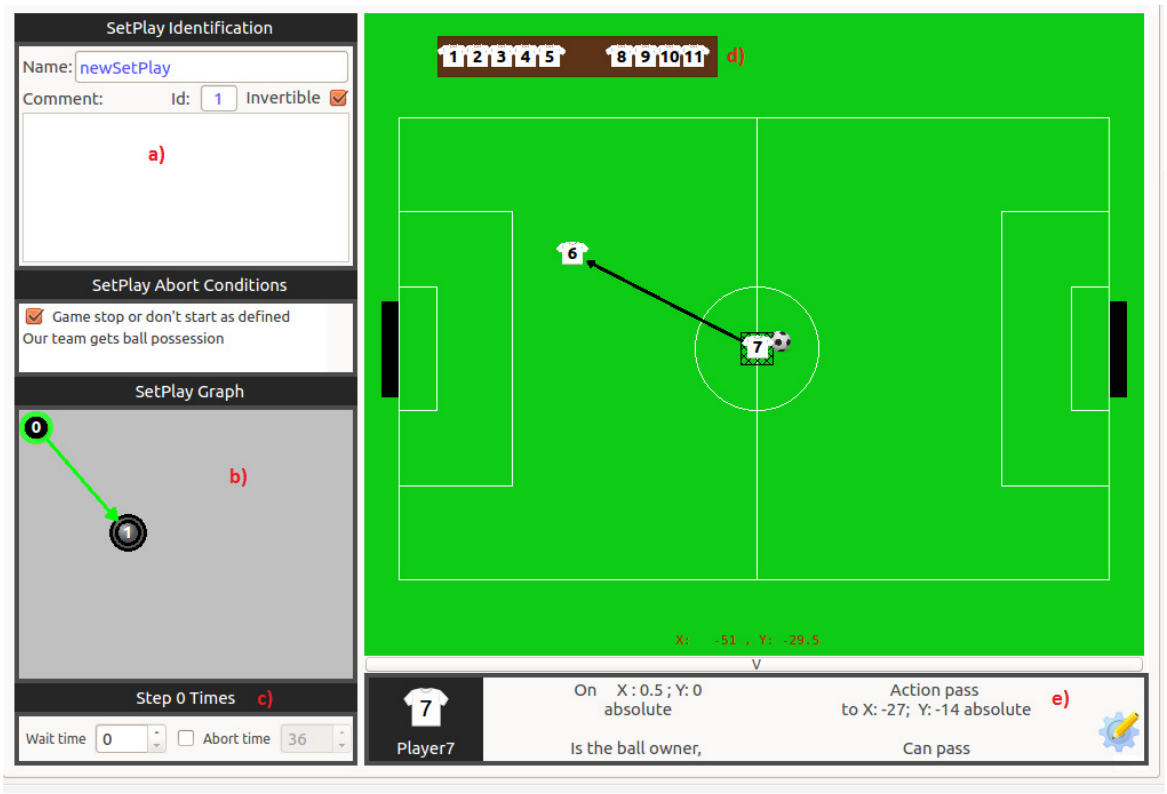
- SpField - Classe responsável por desenhar o terreno do jogo, o banco de jogadores reserva e ainda disponibilizar a interface necessária para poder arrastar os jogadores e criar as suas ações;
- SpGraph - Classe responsável por desenhar um painel e integrar os passos e as transições;
- IspPorting - Interface para importação e exportação de JEs;
- IspOutProgram - Interface para conexão com programas exteriores ao SPlanner;
- ISpShowGame - Interface para se poder ver jogos no terreno de jogo onde se define as JEs

O autor ressalta ainda que existem outras classes de menor importância, pertencentes ao SPlanner, no nível arquitetural, servindo apenas como caixas de diálogo ou janelas gráficas de interação com o usuário, para definição ou configuração de variáveis.

4.2.2 Interface

A interface gráfica do planejador estratégico apresenta cinco áreas distintas [11], como pode ser visualizado na Figura 5: A) Quadro de informações da jogada, b) Grafo de etapas da jogada, c) Quadro de informações do passo, d) Banco de jogadores não participantes, e) Quadro de informações do jogador.

Figura 5 - Interface principal do SPlanner, mostrando a modelagem de uma JE.



Fonte: Adaptado da ferramenta SPlanner [11].

É possível atribuir um nome ao *setplay* e um comentário para guardar alguma observação (Figura 5, item a). O grafo de etapas da jogada (Figura 5, item b), possibilita ver a quantidade e a identificação numérica dos passos que o *setplay* possui. Por convenção, o passo zero é o inicial de uma jogada.

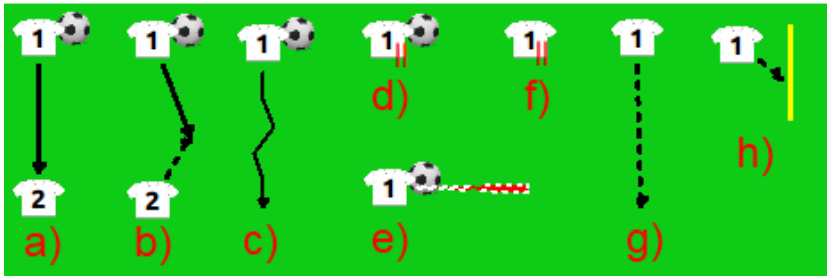
O atributo *Wait time* corresponde ao tempo que o jogador precisa esperar em um passo, para passar para o passo seguinte, e *Abort time*, representa o tempo limite para abortar o *setplay*, caso não seja possível passar para o próximo passo (Figura5, item c).

É possível verificar também os jogadores que não fazem parte da jogada (Figura 5, item d). Informações sobre o jogador selecionado, como as coordenadas de sua posição no campo, a ação que foi atribuída a ele, a condição para executá-la,

as coordenadas resultantes dessa ação e a posse de bola ficam disponíveis no separador inferior ao campo (Figura 5, item e).

Atualmente existem oito ações disponíveis no SPlanner (Figura 6), todas elas possuem uma representação gráfica a ser desenhada no campo quando selecionada, para ser melhor assimilada durante a modelagem de uma JE. As ações ilustradas na Figura 6 são: a) Passe, b) Passe em profundidade, c) Driblar, d) Guardar a bola, e) Rematar, f) Esperar, g) Correr, h) Posicionar-se na linha de fora de jogo.

Figura 6 - Ações possíveis na definição de uma JE pelo SPlanner.



Fonte: João Cravo [11].

Cinco dessas ações são realizadas com a posse de bola: passe, passe em profundidade, driblar, guardar a bola e rematar; e três sem a sua possessão: esperar, correr, e posicionar-se na linha de fora de jogo.

5. DESENVOLVIMENTO DIRIGIDO A MODELO

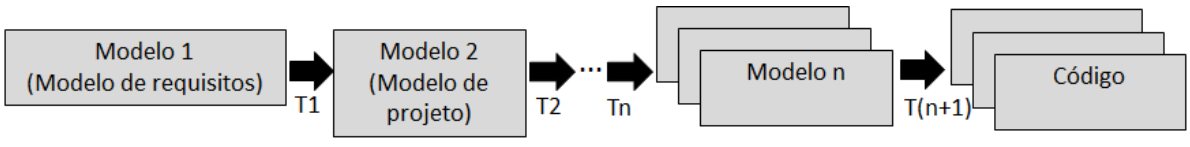
A utilização de modelos para representar sistemas complexos é uma técnica bastante utilizada em diversas áreas da engenharia, a exemplo da engenharia civil, que utiliza modelos para representar a arquitetura de uma construção. A utilização dessa técnica na Engenharia de *Software* foi amplamente disseminada com o advento do paradigma das linguagens de programação orientadas a objeto, e com a popularização da *Unified Modeling Language* (UML) [67], criada para modelar sistemas dentro desse paradigma.

Nesta direção, o Desenvolvimento Dirigido a Modelo (DDM) propõe construir modelos em alto nível de abstração e utilizá-los como artefatos principais nas etapas de desenvolvimento. Esses modelos são convertidos de forma (semi) automática através de uma cadeia de transformação, gerando outros modelos até atingir o código da aplicação [66].

No DDM, modelos devem ser especificados de acordo com uma linguagem de modelagem que possua sintaxe e semântica bem definida. Desta forma, os modelos deixam de ser uma simples documentação e passam a ser o veículo principal pelo qual as aplicações são criadas [66], sendo o insumo que vai gerar seu código final.

O desenvolvimento com DDM compreende dois elementos essenciais, os modelos, que representam um sistema em diversos níveis de abstração, e as transformações, sistemas que automatizam o processo de desenvolvimento convertendo os modelos até o código. A Figura 7 apresenta uma visão geral do desenvolvimento usando DDM. Os modelos (ex. Modelo 1 e Modelo n) e as transformações (ex. T1, Tn).

Figura 7 - Visão Geral do desenvolvimento usando DDM



A classificação da transformação varia de acordo com o artefato produzido em seu *output*, como resultado da transformação do modelo. Transformações M2M (*Model-to-Model*) referem transformações de um modelo para outro modelo, M2T

(*Model-to-Text*) referem transformações de um modelo para o código fonte da aplicação.

A solução proposta nesse trabalho utiliza DDM para criação de novos comportamentos, no entanto não realiza transformações do tipo M2M. A abordagem de transformação M2T utilizada na solução proposta será detalhada na seção 5.1.

5.1 Model To Text Standard

A abordagem de geração de código adotada na Solução proposta é similar à *template-based* utilizada pela *Meta-Object Facility* (MOF) *Model to Text standard* [65]. A partir dessa abordagem modelos podem ser convertidos em uma série de artefatos textuais, como códigos, documentações e especificações. Para isso, uma série de *templates* é parametrizada com os elementos de um modelo.

Cada *template* especifica um texto contendo espaços reservados para dados a serem extraídos de um modelo. Esses dados são convertidos em fragmentos de texto utilizando uma linguagem com suporte à manipulação de strings.

Para ilustrar um exemplo de como isso acontece, o *template* especificado na Figura 8 gera a definição de uma classe em Java. As *tags* “[template]” e “[/template]” limitam o texto definido para o *template public classToJava*. Dentro dessas *tags*, todo o texto escrito resultará no código gerado, exceto pelos espaços em que ficaram definidas as *tags* “[c.name/]”, que serão substituídas pelo conteúdo que corresponde ao nome da classe definido no modelo criado para representá-la.

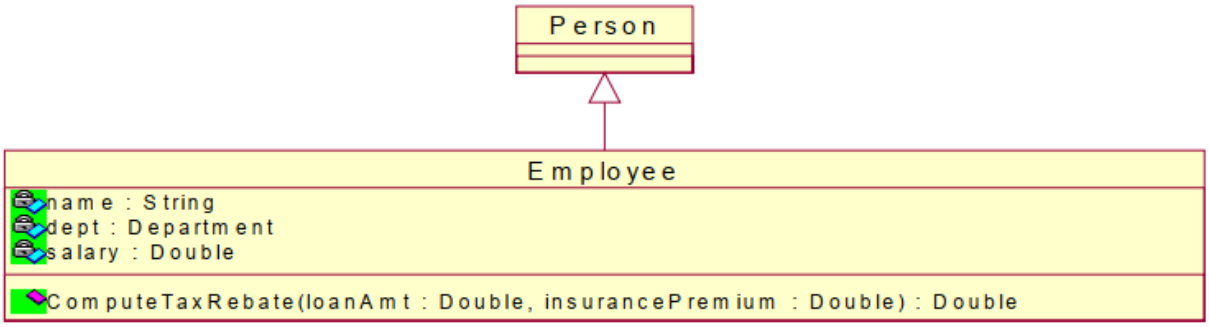
Figura 8 - Exemplo *Template*

```
[template public classToJava(c : Class)]
class [c.name/]
{
    // Constructor
    [c.name/]()
    {
    }
}
[/template]
```

Fonte: *Object Management Group* [65].

De forma resumida, na transformação M2T realizada no modelo correspondente ao diagrama da Figura 9 que contém a classe *Employee*, o *template* da Figura 8 irá gerar o código descrito na Figura 10.

Figura 9 - Exemplo de Modelo



Fonte: *Object Management Group* [65].

A transformação de código da classe *Emplyee* foi simplificada para facilitar o entendimento da abordagem, no entanto, é possível utilizar *templates* para compor requisitos mais complexos, que gerem códigos de sistemas maiores e mais robustos. Por exemplo, no próprio modelo exposto na Figura 9, outros elementos poderiam ser parametrizados em *templates*. Os atributos e métodos da classe *Empolyee*, bem como a criação da super classe *Person* poderiam fazer parte da etapa de transformação e geração de código.

Figura 10 - Exemplo de Código gerado

```
class Employee
{
        // Constructor
        Employee()
        {
        }
}
```

Fonte *Object Management Group* [65].

Todos os elementos dispostos no modelo podem ser convertidos em parâmetros para compor fragmentos de texto por meio de manipulação de strings, podendo criar documentos de natureza diversa, como códigos, documentações e especificações.

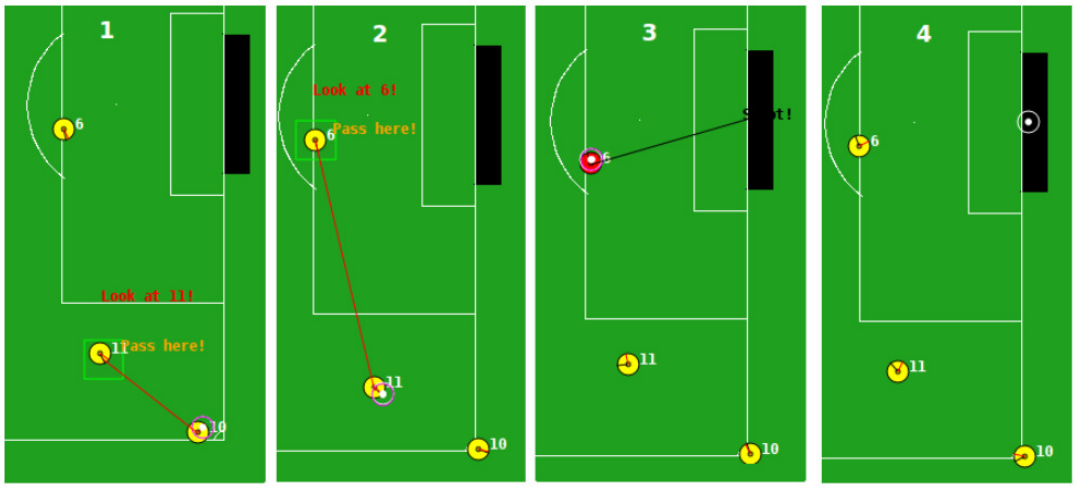
6. TRABALHOS RELACIONADOS

A adoção de ferramentas gráficas de suporte à criação de JEs tem se mostrado um recurso poderoso na Robocup *soccer* ao longo dos anos e um grande esforço tem sido empregado para torná-las cada vez mais versáteis e intuitivas. A intenção aqui é elevar o nível de abstração, à ponto de voltar o foco ao processo de aquisição de estratégia, deixando de lado detalhes técnicos de linguagem de programação.

6.1 *PlayMaker*

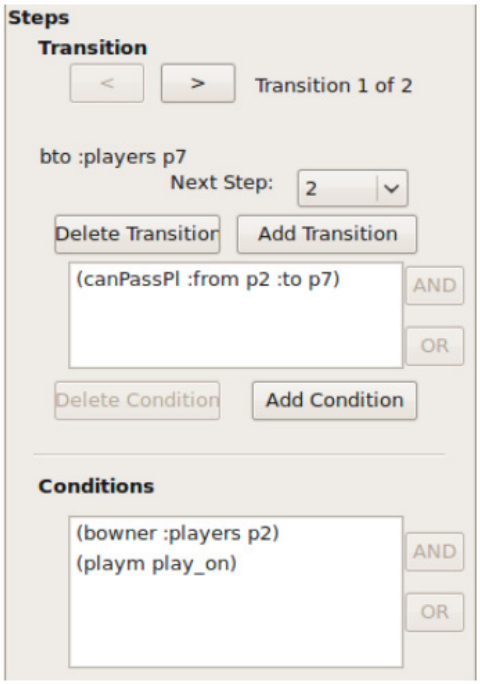
A ferramenta *PlayMaker*, apresentada em [23], surgiu como esforço para desenvolver, de forma rápida, JEs mais robustas e efetivas, possibilitando, através de uma interface amigável, interações com especialistas de futebol nesse processo [24].

Figura 11 - Cobrança de escanteio



Fonte: Mota e Reis [23].

Figura 12 - Interface de edição de *steps* do *setplay*.



Fonte: Mota e Reis [23].

A Figura 11 mostra o planejamento passo a passo de uma jogada de escanteio, desde o toque inicial, no 1º quadrante, até o chute à gol, com resultado previsto no 4º quadrante. A Figura 12 mostra os campos necessários para editar um *step* da jogada, incluindo condições, transições e próximo passo – que mostra qual a próxima etapa da jogada. De acordo com os artigos [11][23], as funcionalidades da ferramenta incluem:

- Definição de formação.
- Definição de JEs.
- Simetria

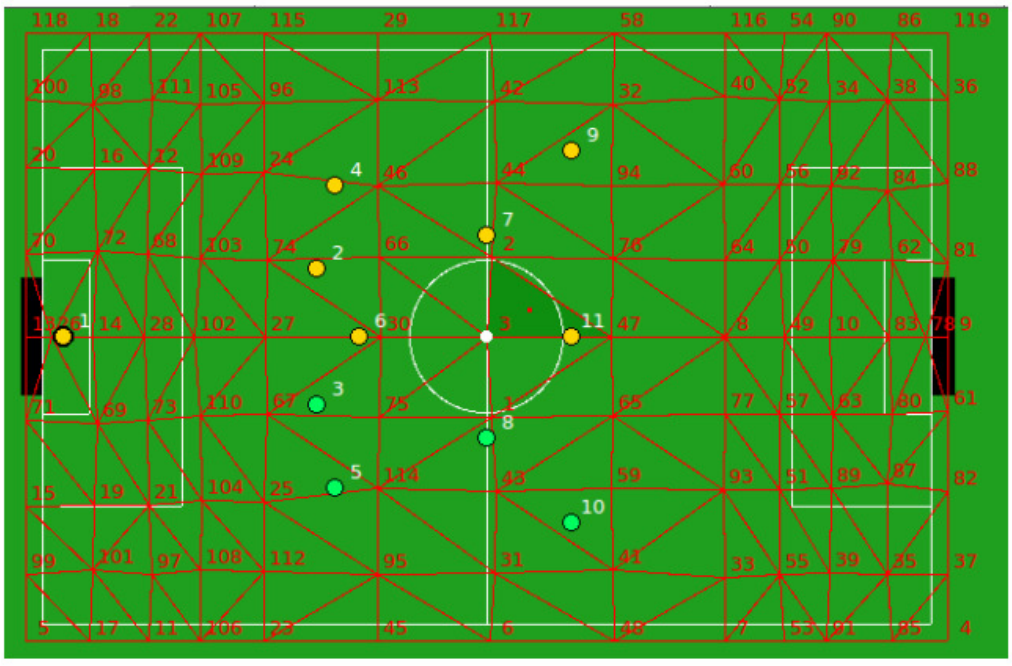
Esses três recursos possibilitam a modelagem de jogadas mais robustas, uma vez que a criação manual de um *setplay*, com mais de 4 passos, converge em uma tarefa difícil, com alta possibilidade de gerar erros no processo [23]. A opção de simetria facilita a replicação da mesma jogada aos dois lados do campo, como resposta rápida ao que se tornaria uma tarefa repetitiva. Nas referências associadas à ferramenta *PlayMaker*, não foi encontrado nada que desse suporte à criação de novos comportamentos.

6.2 *Matchflow*

A ferramenta *Matchflow* [25] foi originalmente criada por Hidehisa Akiyama [29], e posteriormente foi aperfeiçoada pela equipe FCPortugal, com a intenção de criar estratégias aplicáveis a mais de uma liga ao mesmo tempo [25]. Para tal, outros módulos responsáveis pela integração entre a ferramenta e o *framework* dos times das diferentes ligas foram criados, além das melhorias, que proporcionaram funcionalidades extras, por exemplo, o item 2 da lista de funcionalidades da ferramenta:

1. Definir formações de ataque e defesa.
2. Definição de valores de fluxo para regiões do campo.
3. Função de simetria na formação, o que poupa o trabalho de reproduzir posições semelhantes em regiões simétricas no campo.

Figura 13 - Triangulação de Delaunay da ferramenta *Matchflow*.



Fonte: Marco Simões [51].

A ferramenta *Matchflow* (Figura 13) adota um mecanismo baseado na Teoria de Triangulação de Delaunay [25], que facilita o posicionamento dos agentes no campo em função da posição da bola [26]. *Matchflow* foi utilizada também pelo time BahiaRT, do centro de pesquisa ACSO, na criação de novas formações, com

melhoras significantes nos quesitos aproveitamento de escanteio (20% a mais pelo lado direito e 16% a mais pelo lado esquerdo) e colisão entre agentes [51].

Apesar das funcionalidades que dão suporte à aquisição de estratégia do time, *Matchflow* lida apenas com conceitos de fluxos, papéis e formações, portanto se diferencia da proposta desta monografia porque não utiliza comportamentos na modelagem das jogadas.

De acordo com [11], as ferramentas acima possuem expressões próprias e de fácil interpretação para as equipes que as desenvolveram e difícil para outras equipes inexperientes, recorrem muito a caixas de seleção e botões na configuração das estratégias, tornando a ferramenta desconfortável de usar, comparando com outras presentes no mundo dos jogos [30], mais intuitivas e amigáveis. Geralmente as aplicações são dirigidas para ligas específicas da Robocup, podendo depois de algumas adaptações serem utilizadas nas demais.

Em contrapartida a esse cenário exposto, foi desenvolvida a especificação da ferramenta SPlanner, apresentada na seção 4.2, em conjunto com alguns de seus detalhes técnicos. Adicionalmente em [11], com relação ao SPlanner foram identificadas trabalhos futuros necessários, como por exemplo:

- Suporte à criação de *setplays* defensivos;
- Criação de novos comportamentos para dar suporte aos *setplays* defensivos.

Atualmente o time BahiaRT utiliza o FFS e seu módulo SPlanner como ferramentas para definição de estratégia. No entanto, o time tem apresentado demandas para criar novos comportamentos, necessários à modelagem das JEs. Essa tarefa tem se mostrado complexa e dispendiosa, uma vez que envolve códigos que compõe a estrutura de duas ferramentas envolvidas nesse planejamento estratégico, o FFS e o SPlanner.

Com base nas necessidades do time BahiaRT e nos trabalhos futuros apontados em [11], o foco desse trabalho é voltado para o desenvolvimento de um sistema que facilite o processo de criação de novos comportamentos, agilizando essa etapa de desenvolvimento, que acaba se potencializando quando os usuários são novos alunos integrantes do centro de pesquisa ACSO. Nesses casos, como resultado esperado, é buscada uma significativa diminuição da curva de

aprendizado, referente ao processo de criação dos novos comportamentos para os agente em campo.

Foi tensionado, no desenvolvimento da interface gráfica da solução proposta, elevar o nível de abstração do problema e facilitar a coleta de parâmetros, por meio de uma interface simplificada. É esperado que a validação da ferramenta corrobore com essa expectativa.

7. METODOLOGIA

A metodologia de desenvolvimento deste trabalho tem caráter exploratório e aplicado. Inicialmente foi feita uma análise em conjunto com o pessoal do centro de pesquisa ACSO-UNEB, para avaliar as possibilidades de atuação e estabelecer o domínio de pesquisa, o que definiu o plano de estudo, envolvendo o ambiente de desenvolvimento e testes do time BahiaRT, que atua na subliga de Simulação da Robocup *Soccer*. A metodologia está dividida em sete etapas, conforme descrito a seguir.

Etapa 1. Esta etapa teve como foco o estudo do ambiente de instalação do FFS e do módulo gráfico de criação de JEs, o SPlanner, suas bibliotecas e a forma são compilados, uma vez que buscamos alterar arquivos que compõem a estrutura dos dois, recompilar os códigos modificados é uma passo muito importante.

Etapa 2. Após montar um ambiente para desenvolvimento e teste, tendo conhecimento dos elementos da primeira etapa, buscamos entender os componentes de interface com o usuário do SPlanner, e a estrutura do FFS. A linguagem utilizada para compor o FFS e o SPlanner é C orientada a objetos (C++), então foi necessário estudar as classes e métodos que são invocados,à medida que uma nova jogada vai sendo modelada.

Etapa 3. As duas etapas anteriores garantiram uma análise de viabilidade da solução, pois com esse conhecimento foi possível fazer um levantamento dos elementos reutilizáveis nos códigos que compõem o FFS e o SPlanner, de que forma a inserção de código seria realizada e documentada, quais elementos do comportamento deveriam ser coletados para a geração de código, e a forma como seria feita essa coleta na interface com o usuário. Essa etapa definiu melhor o escopo do projeto.

Etapa 4. Nesse momento foi definida qual linguagem seria utilizada no protótipo e, com base nos estudos realizados na terceira fase, foi desenvolvida a arquitetura da solução.

Etapa 5. Na quinta etapa foi criado o protótipo, essencial para uma apresentação inicial do projeto ao ACSO, para coleta de sugestões e análise de impacto.

Etapa 6. Após a validação do protótipo e da arquitetura da solução foi possível

recolher as especificações necessárias para desenvolver a solução proposta da forma como está descrita na seção 8.

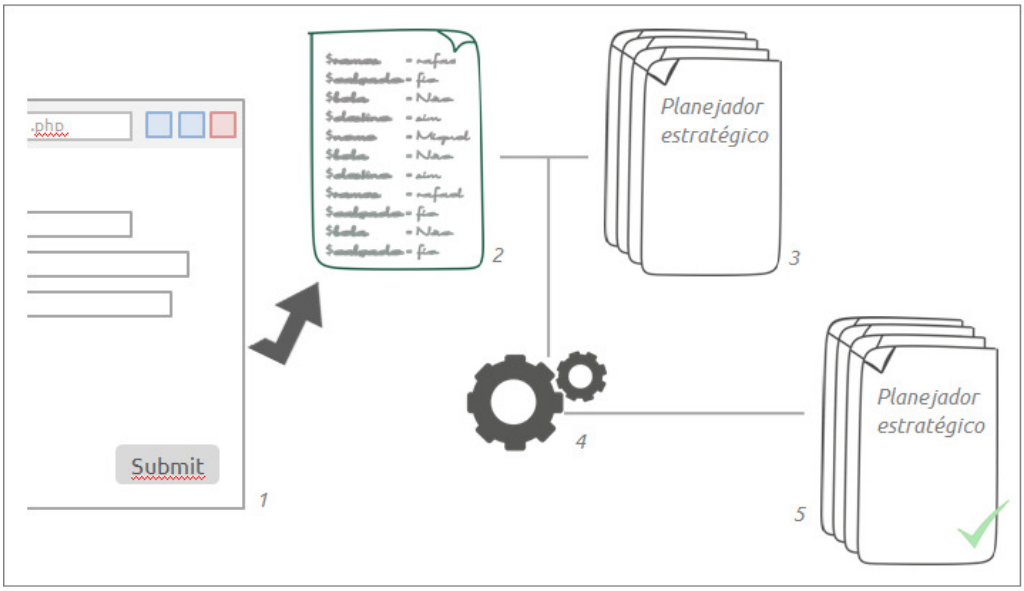
Etapa 7. Após a etapa de desenvolvimento foi realizada a validação da solução proposta, submetendo a métricas de avaliação do seu desempenho na criação de um novo comportamento. Essa etapa de validação está descrita passo a passo na seção 9.

8. SOLUÇÃO PARA CRIAÇÃO DE COMPORTAMENTOS

Este capitulo apresenta a solução proposta para criação de comportamentos de JE no futebol de robô. A solução utiliza um mecanismo de geração e inserção automática do código dentro dos arquivos do planejador estratégico do time BahiaRT, composto pelas ferramentas FFS e SPlanner.

A Figura 14 demonstra uma visão geral do processo de geração de comportamento proposto na solução. O primeiro passo (1 na figura) representa a coleta de dados realizada através de uma interface gráfica com o desenvolvedor. O conjunto de dados coletados nesta interface será utilizado como parâmetros para a geração do comportamento (2). Como dito anteriormente, para a criação do comportamento é necessário incluir ou alterar o código dos arquivos já existentes na ferramenta FCC e SPlanner (3). Portanto, o gerador de código (4) recebe como entrada os parâmetros informados pelo desenvolvedor e os arquivos a serem alterados e gera como saída os arquivos modificados (5) com o código necessário para o novo comportamento.

Figura 14 - Solução proposta



Fonte: O autor.

O restante deste capítulo detalha cada elemento da solução proposta apresentado na Figura 14: a seção 8.1 aborda os requisitos da solução proposta, a seção 8.2 aborda a interface gráfica com o desenvolvedor do comportamento, a

seção 8.3 define os parâmetros utilizados para geração de comportamento, a seção 8.4 demonstra os arquivos envolvidos na geração de comportamento, a seção 8.5 aborda o mecanismo gerador ilustrado no item 4 da Figura 14, a seção 8.6 apresenta a arquitetura de alto nível da solução e por fim a seção 8.7 define um passo a passo para a preparação do ambiente de execução da ferramenta.

8.1 Requisitos da solução

Para atender à solução proposta os seguintes requisitos foram levantados em acordo com os objetivos do time Bahia RT:

R1 A solução deve gerar um comportamento a ser identificado pela ferramenta SPlanner.

R2 A solução deve prover um recurso de *log*, com as seguintes características:

R2.1: Localização dos arquivos alterados;

R2.2: Localização das linhas alteradas nos arquivos;

R2.3: Contagem do total de linhas inseridas;

R2.4: Contagem do total de métodos criados.

R3 A solução deve ser construída para a plataforma Linux;

R4 A interface da solução deve explicar claramente quais os dados necessários para a geração do comportamento, de forma que desenvolvedores sem experiência com o código das ferramentas SPlanner e FCC sejam capazes de gerar um novo comportamento.

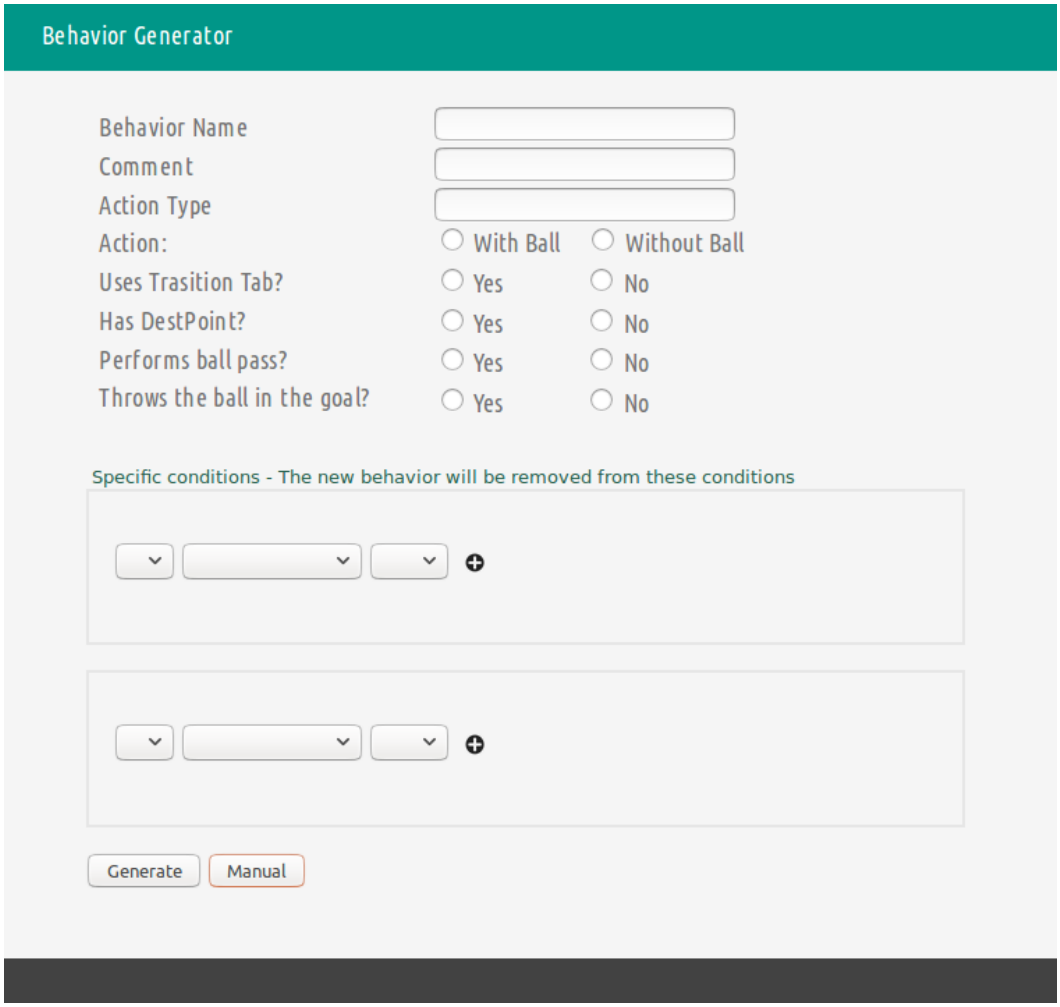
R5 A solução deve ser aplicável a outros times da Robocup, ou seja, não deve possuir limitações que permitam apenas geração de comportamentos para o time BahiaRT.

À ferramenta de *log* caberá fornecer recursos para a validação da ferramenta, bem como servir de consulta aos alunos, que desejem verificar de forma mais transparente as etapas do processo de geração de comportamento.

8.2 Interface gráfica da Solução

A interface de entrada de dados do Gerador foi construída utilizando as linguagens Php e HTML. A linguagem Php foi escolhida devido à compatibilidade com múltiplas plataformas e possibilita que futuramente o Gerador possa ser hospedado em um servidor com suporte à requisições cliente-servidor, para acesso WEB. A interface possui um formulário com 11 itens de entrada de dados. Esses dados são classificados em dados simples, que utilizam campos de texto e campos de seleção para informação de valores, e blocos de condições, que permitem definir condições associadas ao comportamento, indicando situações de uma JE em que o comportamento não poderá ser utilizado. Existem comportamentos que, por exemplo, só são permitidos no passo inicial (*step* zero) de uma jogada, e outros que só podem ser executados após esse passo (*step*). É possível visualizar dois botões na Figura 15, o botão "*Generate*" que encaminha os parâmetros do formulário para as páginas que irão criar o comportamento e o botão "*Manual*", que encaminha para a página do manual de utilização da ferramenta.

Figura 15 - Página de interface com o desenvolvedor.



Fonte: O autor.

O manual da ferramenta, conforme apresentado no Apêndice C, é composto pelas informações dispostas na Tabela 4, possui o significado de cada parâmetro solicitado no formulário de entrada de dados, na intenção de facilitar o entendimento do desenvolvedor, a respeito dos elementos associados à geração do comportamento.

8.3 Parâmetros necessários à geração

A escolha dos parâmetros para geração de código foi realizada com base na análise do FCC e do módulo gráfico SPlanner, código a código, buscando identificar comportamentos existentes e seus padrões na utilização de classes e métodos. A relação entre cada parâmetro identificado, que é solicitado no formulário da interface

com o desenvolvedor (Figura 15), e sua finalidade na geração de código pode ser visualizada na Tabela 4.

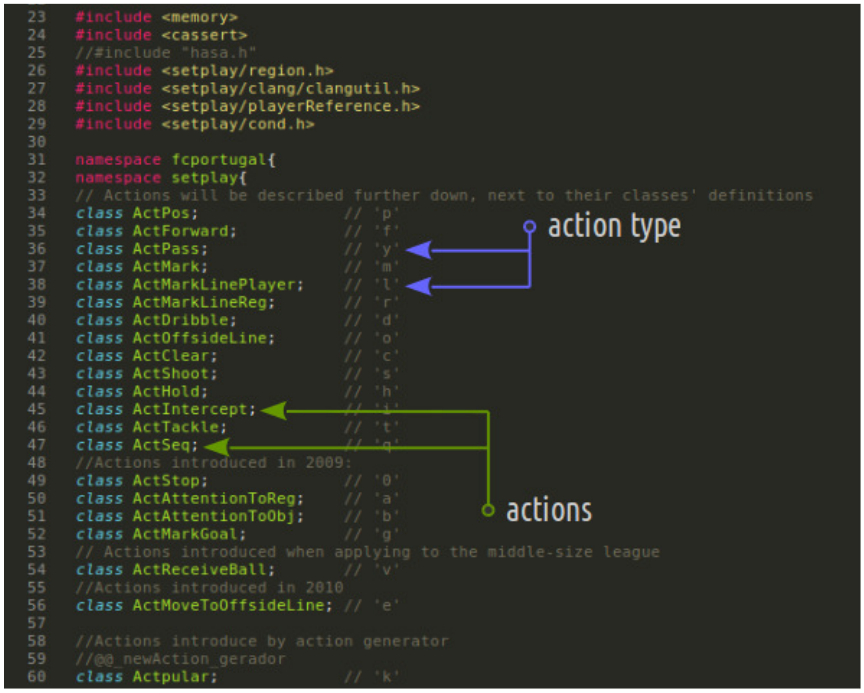
Tabela 4 - Definição dos parâmetros da solução

Item	Parâmetro	Finalidade
1	*Bahavior Name*	Define o nome do comportamento.
2	*Comment*	Registra um comentário para ser acrescentado no código, em pontos de definição do comportamento.
3	*ActionType*	Preenche um atributo da classe *clangaction*, que será usado por outras classes como um identificador único do comportamento. Pode ter no máximo dois caracteres. É necessário verificar no cabeçalho da biblioteca *clangaction.h* qual identificador ainda não existe para poder ser utilizado (Figura 16).
4	*Action*	Possui dois valores possíveis: com bola e sem bola. Estabelece se a ação será realizada com posse de bola ou não.
5	*Uses Transition Tab?*	Indica se no momento de seleção do comportamento na ferramenta SPlanner, a tela de transição (Figura 17) será aberta ou não automaticamente.
6	*Has DestPoint*?	A presença do atributo *DestPoint* na ação representa se a ação realiza movimentação da bola ou do jogador no campo. Ex. Correr (jogador), Driblar (jogador e bola), Tocar (bola), Esperar (não possui *DestPoint*). Possui uma exceção: Ações de lance à gol não possuem *Destpoint*.
7	*Performs ball pass*?	Determina se a ação realiza passe de bola.
8	*Throws the ball in the goal*?	Representa uma ação que lança a bola no gol. Ações de lance à gol não possuem *DestPoint*, pois nesses casos o SPlanner direciona automaticamente o destino da ação para o centro do gol.

Fonte: O autor

Conforme indicado no item 3 da tabela 4, a Figura 16 ilustra o local do arquivo *clangaction.h* onde todas as *actions* existentes e suas respectivas *actions type* ficam registradas.

Figura 16 – Localização da lista de *actions type* utilizadas.

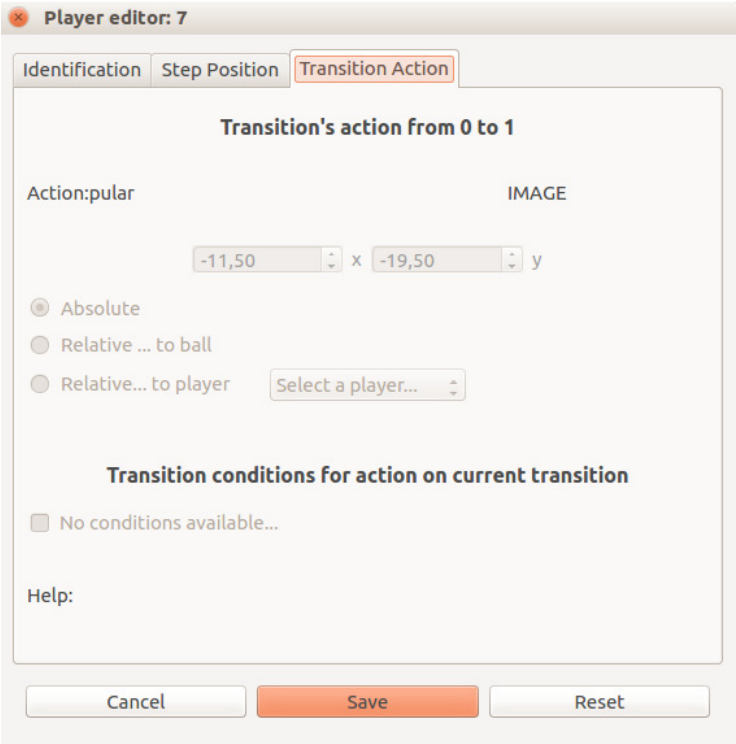



```
23  #include <memory>
24  #include <cassert>
25  //#include "hasa.h"
26  #include <setplay/region.h>
27  #include <setplay/clang/clangutil.h>
28  #include <setplay/playerReference.h>
29  #include <setplay/cond.h>
30
31  namespace fcportugal{
32  namespace setplay{
33  // Actions will be described further down, next to their classes' definitions
34  class ActPos;                // 'p'
35  class ActForward;            // 'f'
36  class ActPass;               // 'y'
37  class ActMark;               // 'm'
38  class ActMarkLinePlayer;     // 'l'
39  class ActMarkLineReg;        // 'r'
40  class ActDribble;            // 'd'
41  class ActOffsideLine;        // 'o'
42  class ActClear;              // 'c'
43  class ActShoot;              // 's'
44  class ActHold;               // 'h'
45  class ActIntercept;          // 'i'
46  class ActTackle;             // 't'
47  class ActSeq;                // 'q'
48  //Actions introduced in 2009:
49  class ActStop;               // '0'
50  class ActAttentionToReg;     // 'a'
51  class ActAttentionToObj;     // 'b'
52  class ActMarkGoal;           // 'g'
53  // Actions introduced when applying to the middle-size league
54  class ActReceiveBall;        // 'v'
55  //Actions introduced in 2010
56  class ActMoveToOffsideLine; // 'e'
57
58  //Actions introduce by action generator
59  //@@_newAction_gerador
60  class Actpular;              // 'k'
```

Fonte: Adaptação de [55] realizada pelo autor.

O item 5 da tabela 4 menciona a tela de transição de uma ação no SPlanner, que é utilizada quando o desenvolvedor deseja adicionar maiores detalhes sobre a transição de uma ação para o próximo passo (*step*). A Figura 17 ilustra essa tela.

Figura 17 - Tela de transição do SPlanner



Fonte: Ferramenta SPlanner [11]

O preenchimento dos blocos de condição específica se baseia em um conjunto de regras identificadas no código do SPlanner, descritas a seguir.

Os dois blocos de condições específicas obedecem a gramática disposta no Apêndice D, e funcionam da seguinte forma: de acordo com a Figura 18, são 5 campos possíveis de serem preenchidos com valores do tipo *combo box*, o principal é o campo indicado na Figura 18 com o número 2, pois os valores dos outros campos são filtrados a partir dele.

Caso a opção escolhida no campo 2 seja "*play mode*", a condição resultante deverá ser igual "=" ou diferente "!=" (campo 3) a qualquer um dos seguintes valores que estarão disponíveis no campo 4:

- *play on*
- *kick off*
- *throw in*
- *direct free kick*
- *indirect free kick*
- *corner kick*

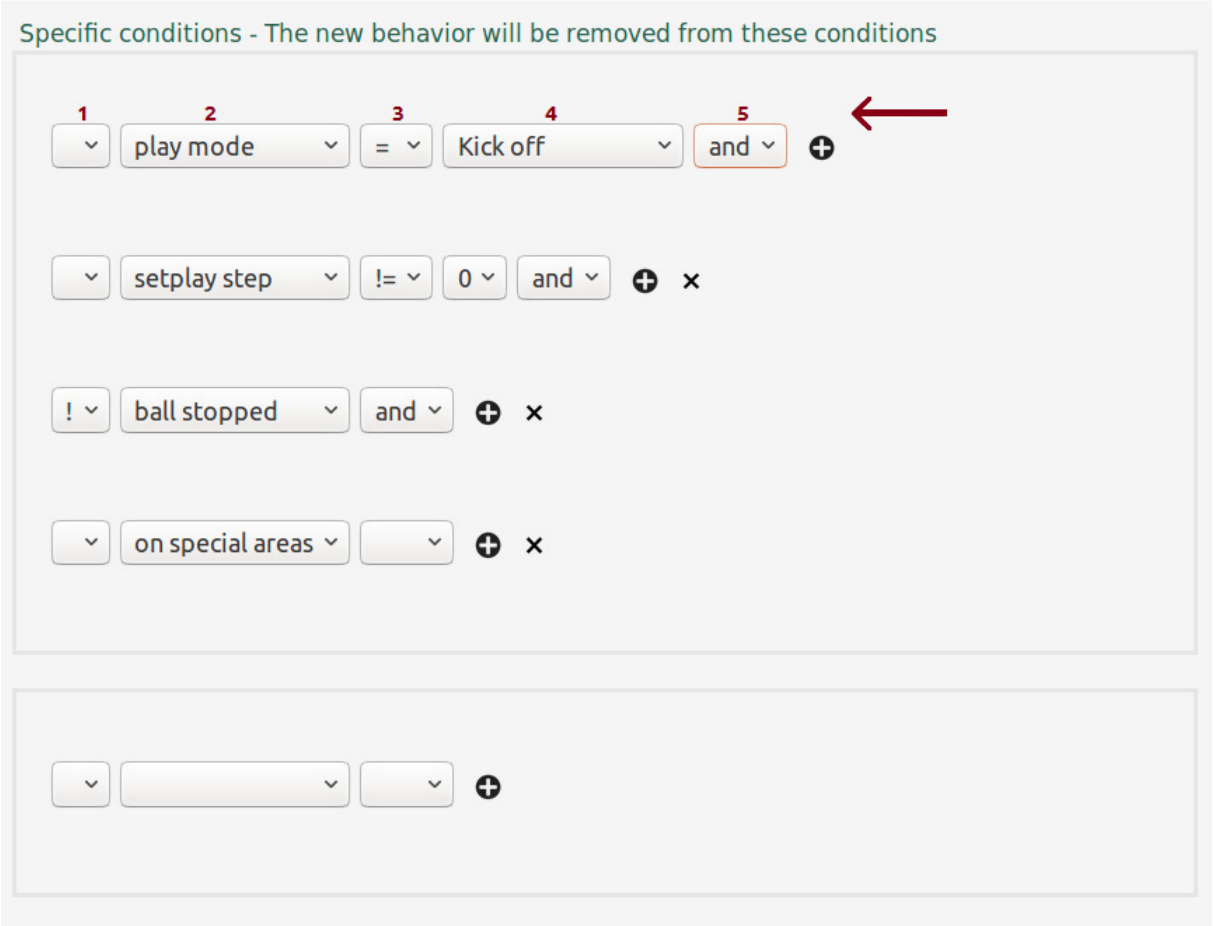
- *goal kick*
- *keeper catch*

Caso a opção escolhida no campo 2 seja “*setplay step*”, a condição resultante deverá ser igual “=” ou diferente “!=” (campo 3) ao valor “0” do campo 4.

Caso a opção escolhida no campo 2 seja “*ball stopped*” ou “*on special areas*”, o campo 1 poderá negar “!” ou não “ “ a condição resultante. Nesse caso os campos 3 e 4 não ficarão visíveis.

No campo de numero 5, o desenvolvedor sempre poderá selecionar o operador lógico “*and*” ou “*or*”, para adicionar novas condições à clausula de condições específicas.

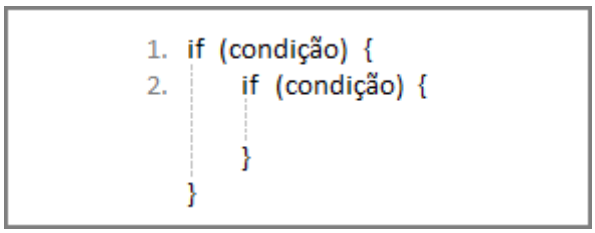
Figura 18 – Blocos de condições específicas.



Fonte: O autor.

São dois blocos de condição específica, conforme mostra a Figura 19, que funcionam da seguinte forma: o primeiro bloco indica as condições do primeiro "*if*", ao passo que o segundo bloco indica as condições do segundo "*if*".

Figura 19 - Configuração das cláusulas *if*.



Fonte: O autor.

O novo comportamento será removido do menu de seleção das *actions* no SPlanner, quando forem satisfeitas as condições selecionadas nos blocos de condição específica do Gerador de Comportamento.

8.4 Arquivos envolvidos na geração

No total, 9 arquivos são alterados na geração de código, como pode ser visto na Tabela 5, eles fazem parte da estrutura do FFS e do módulo gráfico SPlanner, com exceção dos arquivos *setplayaction*, que fazem parte da camada de interface entre o FFS e o time BahiaRT, apresentada na seção 4.1.1.

Tabela 5 – Lista de arquivos envolvidos na geração de comportamento.

	SPlanner	**FFS**	**Interface BahiaRT**
clangaction.h	✓	✓	
clangaction.cc	✓	✓	
spaction.h	✓		
spaction.cpp	✓		
spfcportugalporting.cpp	✓		
spfield.cpp	✓		
spplayer.cpp	✓		
setplayaction.cpp			✓
setplayaction.h			✓

Fonte: O autor.

Cada arquivo desempenha uma função na estrutura do FFS ou do SPlanner, como pode ser visto na Tabela 3. No entanto, existem outras finalidades mais específicas ao contexto da criação de um comportamento, associadas aos métodos

que são alterados ou criados pelo processo de geração de código. Essas relações específicas podem ser visualizadas na Tabela 6.

Tabela 6 - Finalidade da alteração dos arquivos.

Arquivo	**Modificações**
spaction	Cria um método de desenho específico para a ação na tela do SPlanner, possui o método construtor das ações.
spfcportugalporting	Cria a assinatura e a chamada sem implementação, do método referente ao novo comportamento que fará parte do mecanismo de importação e exportação de JEs. Indica sua localização para que possa ser acessado e implementado manualmente.
Spfield	Ações de passe devem aceitar apenas outro jogador como destino da ação, para garantir isso existe dois métodos nessa classe que devem ser atualizados. Atualiza também o desenho da ação que liga o jogador com posse de bola ao jogador que receberá o passe.
spplayer	Altera o método que cria o menu de seleção de ações, adicionando a ele o novo comportamento, cria a chamada da função que desenha a nova ação selecionada na tela. Altera métodos que modificam a posse de bola dos jogadores em campo.
Setplayaction	Cria a assinatura e a chamada sem implementação do método que mapeia o novo comportamento do FFS para o time BahiaRT, indicando o local que deverá ser implementado manualmente.
Clangaction	Cria assinaturas e chamadas de métodos sem implementação, indicando os locais que deverão ser implementados manualmente com definições específicas ao novo comportamento, que serão compartilhadas entre o FFS e o SPlanner.

Fonte: O autor

Nas classes *clangaction, setplayaction e spporting*, serão geradas assinaturas e chamadas de métodos sem implementação, que deverão ser preenchidos com códigos a serem desenvolvidos pela equipe do time BahiaRT. A implementação desses métodos não pôde ser incluída na etapa de geração de código, pois eles lidam com a base de conhecimento do time e variam de acordo com a capacidade e a inteligência dos agentes, levando a um grau de especificidade que aumenta a

complexidade da previsão de seus conteúdos, fugindo à proposta de uma ferramenta de auto nível de abstração.

8.5 Mecanismo gerador de comportamento

O gerador de comportamento é composto por 9 páginas Php, cada uma contendo *templates* de geração de código específicos a um arquivo correspondente na lista da Tabela 5. Um *template* é um modelo de código extraído do FFS ou do SPlanner, contendo os trechos que serão substituídos pelos parâmetros coletados na interface com o desenvolvedor. Os *templates* em conjunto com os parâmetros adicionam as características do novo comportamento aos códigos do FFS e do SPlanner, ao passo que seus arquivos vão sendo atualizados pelo Gerador.

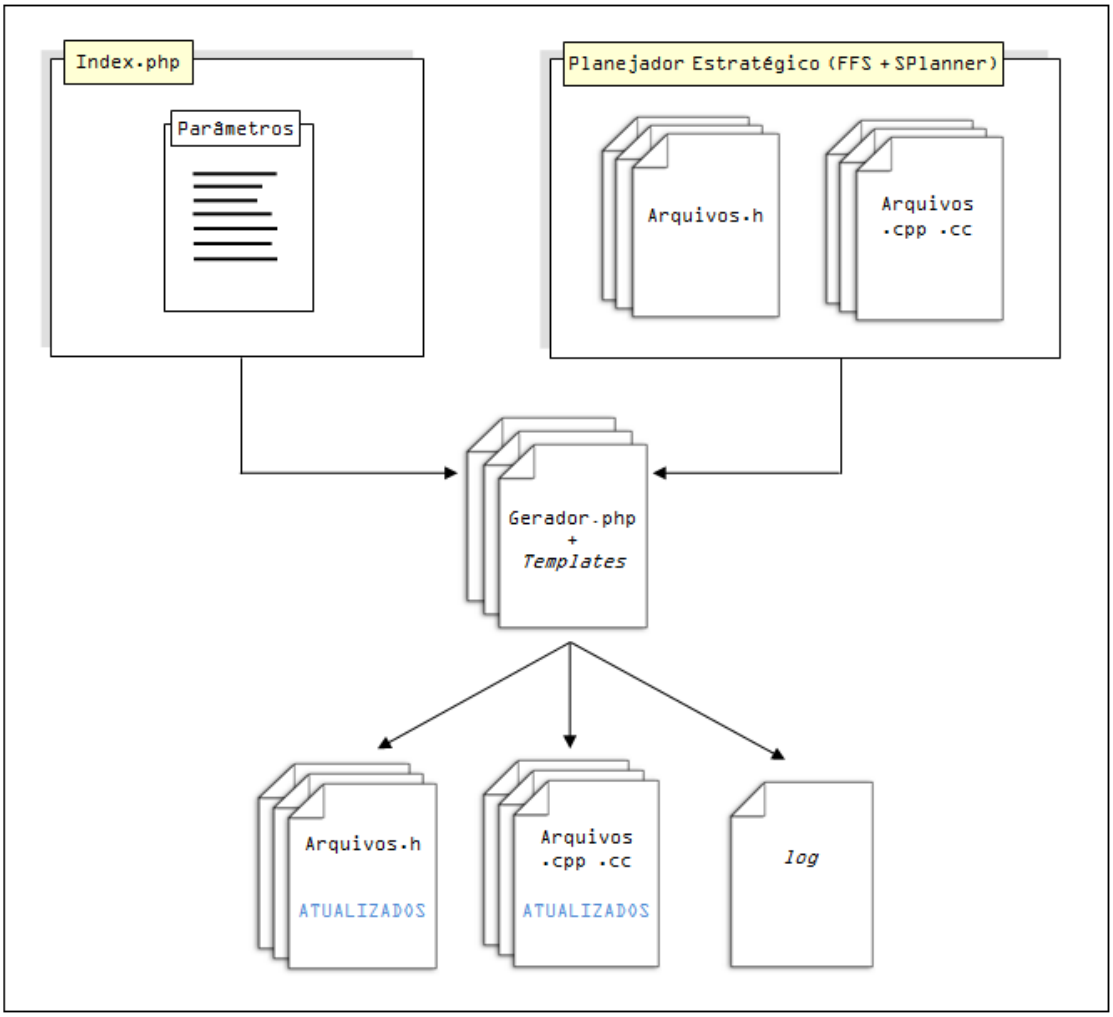
Pontos de inserção de código foram criados nos arquivos originais do FFS e do SPlanner utilizado *tags* em locais específicos. Dessa forma, o comportamento resultante varia de acordo com a lógica extraída do formulário, o que resultará na seleção dos *templates* compatíveis com essa lógica e na utilização de suas respectivas *tags* como pontos de inserção no código fonte.

Utilizando funções de manipulação de arquivo do Php, o Gerador foi configurado para acessar os arquivos em suas pastas originais e modificá-los sem qualquer intervenção do usuário da ferramenta, compondo um *log* contendo as informações dispostas na seção 8.1.

8.6 Arquitetura da solução

A arquitetura em alto nível da solução está representada na Figura 20 da seguinte forma: a página Index.php é responsável pela coleta dos parâmetros informados pelo desenvolvedor do comportamento. Os parâmetros coletados são passados à página Gerador.php, que irá criar o novo comportamento. Para isso, processa os arquivos que serão atualizados e executa uma série de *templates*, cada um configurado para modificar um determinado arquivo do planejador estratégico.

Figura 20 - Arquitetura da solução



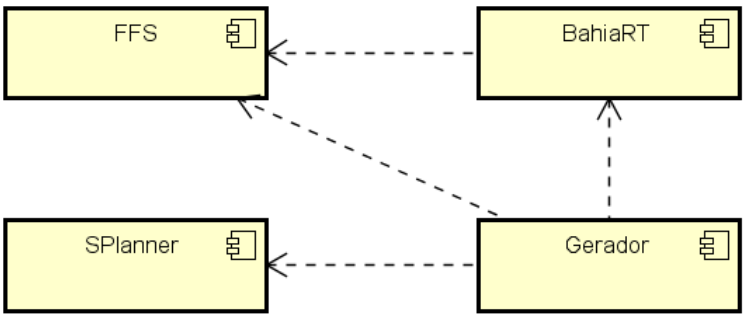
Fonte: O autor.

O gerador representa esse mecanismo de manipulação dos arquivos, realizado com base nos *templates* de geração de código e nos parâmetros coletados na interface com o usuário. A solução deve atualizar os arquivos do planejador estratégico e gerar um *log* conforme apresentado na seção 8.1.

A Figura 21 representa o diagrama de componentes da solução. O Gerador necessita acessar arquivos do FFS, SPlanner e do time BahiaRT para executar o processo de geração de comportamento, criando uma relação de dependência com esses três componentes. Os códigos do time BahiaRT que o Gerador acessa fazem parte da interface de comunicação do time com o FFS. Essa interface de comunicação foi apresentada na seção 4.1.1 e os arquivos correspondentes a ela estão dispostos na Tabela 5, na coluna denominada “interface BahiaRT”. Embora o

SPlanner possa criar e exportar jogadas ensaiadas para o FFS, não existe relação de dependência entre os dois, pois o SPlanner é um módulo independente que pode ser executado sem vínculo direto com o FFS. O time BahiaRT precisa acessar os conceitos e mecanismos que o FFS possui (Figura 3) para executar as jogadas ensaiadas, essa relação de dependência pode ser visualizada na Figura 21.

Figura 21 - Diagrama de componentes do Gerad**or**



Fonte: O autor.

O SPlanner possui o encargo adicional de desenhar as ações na tela, o que resultou em uma demanda demasiadamente específica para a ferramenta de geração de código, pois cada comportamento possui sua respectiva representação gráfica (Figura 6). Como solução para esse problema, foi criado um padrão genérico de desenho, podendo ser modificado posteriormente, de acordo com a necessidade de cada ação.

Além da criação de novos comportamentos, existe uma demanda emergente no ACSO-UNEB: habilitar no SPlanner, a opção de modelagem de *setplays* defensivos. Assim como foi dito pelo autor da ferramenta, em [11], é preciso criar comportamentos que dêem suporte à essa modalidade de JE, basta notar que ações de marcação, por exemplo, que estão previstas na Tabela 1, do FFS, ainda não estão previstas na Figura 6 do SPlanner. Essa diferença se dá ao fato de que, inicialmente a ferramenta SPlanner foi desenvolvida apenas com a opção de modelar *setplays* ofensivos [11].

8.7 Preparação do ambiente

O principal recurso utilizado na solução foram as funções de manipulação de arquivo da linguagem Php, o que implica na utilização de um

ambiente que dê suporte à interpretação da página do Gerador de interface com o usuário. O ambiente escolhido para executar a solução foi o XAMPP da plataforma Linux, que dá suporte à interpretação e execução de códigos Php, além de possuir um procedimento de instalação bastante simples. A versão do XAMPP utilizada para foi a 7.2.9. É importante mencionar a versão do XAMPP utilizada, pois ela possui a versão do Php utilizada no Gerador. O passo a passo para a preparação do ambiente do Gerador de Comportamento pode ser visualizado no Apêndice B.

A solução proposta neste trabalho modifica arquivos de código fonte já existentes nas ferramentas FFS e SPlanner, acrescentando o código necessário à criação de novos comportamentos. Portanto, após a atualização do código fonte das ferramentas por meio da execução do Gerador de Comportamentos, elas precisam ser novamente compiladas para que o comportamento se torne disponível para uso.

9. VALIDAÇÃO

Este capítulo apresenta a validação da solução proposta para criação de comportamento. A validação foi realizada no formato de estudo de caso, com o objetivo de avaliar sua viabilidade na geração de novos comportamentos para o planejador estratégico do time BahiaRT, a saber, o FFS e o seu módulo gráfico SPlanner. O objetivo da avaliação foi definido de acordo com o padrão GQM [61] conforme ilustrado na Figura 22.

Figura 22- Objetivo da Avaliação

Analisara solução Gerador de Comportamento **Com o propósito de** avaliar a viabilidade da geração de código **Com respeito a** amplitude do código gerado e a performance do desenvolvimento. **Na perspectiva do** desenvolvedor de software **No contexto de** estudantes de graduação do centro de pesquisa ACSO e especialista do ACSO

Fonte: *Template* GQM [61].

Para nortear a avaliação de validação da ferramenta de geração de comportamento, as seguintes questões de pesquisa foram definidas:

Q1 A solução gerou o código necessário para incluir um novo comportamento?

Q2 A solução diminuiu o tempo necessário para se desenvolver um novo comportamento?

Para avaliar essas questões de pesquisa foram definidas a métrica M1 e a métrica M2, que devem ser calculadas de acordo com os resultados obtidos na execução do estudo de caso.

Métrica M1: Amplitude do código gerado. Usará como parâmetro a quantidade de linhas envolvidas no processo de geração de código conforme demonstrado na Figura 23:

Figura 23 - Fórmula para o cálculo da amplitude.

$$\text{Amplitude} = \frac{\text{total de linhas geradas}}{\text{total necessário}}$$

Fonte: O autor.

Se a amplitude = 1, nada precisou ser feito para compor o novo comportamento além do que foi gerado pela solução. À medida que o valor resultante assume valores menores que 1, se torna um indicativo que linhas de código precisaram ser inseridas para terminar de criar o novo comportamento. Sendo assim, quanto mais próximo de 1 for o resultado, melhor é a avaliação da solução.

Métrica M2: Performance do desenvolvimento. Usará como parâmetro a quantidade de minutos necessária para desenvolver um novo comportamento com a solução.

9.1 Planejamento do experimento

Para realizar a validação foi utilizado o cenário que consiste em um comportamento específico, já existente, sendo recriado de forma similar, utilizando a solução proposta. Esse desenvolvimento inclui a etapa de geração automática, que é própria da solução, e a etapa de adição dos códigos que não são gerados e que são específicos caso a caso.

O comportamento existente escolhido foi o *Dribble*, que consiste em uma ação com posse de bola, e implica na tentativa de movimentação do robô de um ponto de origem a um ponto de destino, ou *destpoint*, sem que haja a perda dessa posse.

A coleta de dados foi realizada de forma direta, através do questionário localizado no Apêndice A, aplicado durante o estudo de caso, e de forma indireta, através da análise dos arquivos gerados pelos participantes.

Foram selecionados para participar do estudo alunos que pertencem ao grupo ACSO, e tem experiência no FFS e em seu módulo de gráfico Splanner ou profissionais que já tenham trabalhado recentemente (até 3 anos) no grupo ACSO, no time de futebol de robôs BahiaRT.

9.2 Execução do estudo de caso

O estudo foi executado com duas replicações, a primeira foi aplicada a um participante especialista, que realizou o estudo de caso e respondeu questões específicas adicionais, compatíveis com um nível de conhecimento maior sobre o FFS e o SPlanner. A segunda replicação foi aplicada a dois alunos atuais do ACSO, que durante o estudo de caso responderam ao mesmo questionário do participante especialista sem as questões específicas. No total 1 pessoa participou da primeira replicação e 2 pessoas participaram da segunda, como pode ser visto na tabela 7.

Tabela 7 - Participantes do estudo de caso

Etapa	Integrante	Perfil
1ª replicação	Participante 1	Especialista
2ª replicação	Participante 2	Aluno
	Participante 3	Aluno

Fonte: O autor

Exceto pelas questões específicas direcionadas apenas ao participante 1, as duas replicações seguiram o mesmo roteiro de validação:

1. Abrir a página do gerador no navegador WEB.
2. Ler as informações do manual da ferramenta.
3. Preencher o formulário com os dados do comportamento.
4. Gerar o comportamento.
5. Inserir os códigos que a solução não gera (O autor).
6. Recompilar o SPlanner.
7. Executar o SPlanner.
8. Criar uma JE utilizando o novo comportamento gerado, realizar a exportação da JE.
9. Responder o questionário.
10. Realizar um teste prático de execução da JE criada no time BahiaRT (O autor).

Após a execução dos passos 1 a 9 do roteiro acima, coube ao desenvolvedor da solução proposta, o mesmo que aplicou o estudo de caso, realizar o passo 10 e verificar a alteração dos seguintes arquivos:

- clangaction.h
- clangaction.cc
- spaction.h
- spaction.cpp
- spfcportugalporting.cpp
- spfield.cpp
- spplayer.cpp
- setplayaction.cpp
- setplayaction.h

Para cada replicação do estudo de caso, após a criação de um comportamento utilizando o Gerador, ele precisa passar por 3 etapas de verificação, como pode ser visto na Tabela 8:

Tabela 8 - Etapas de verificação do comportamento criado.

Etapa 1	Análise arquivo por arquivo, constatando a completude do código gerado.
Etapa 2	Verificação do comportamento reconhecido e exportado pelo SPlanner sob o formato de uma JE.
Etapa 3	Teste Prático da JE exportada pelo SPlanner sendo executada pelo time BahiaRT.

Fonte: O autor.

9.3 Coleta e análise dos dados

A resposta dos questionários pelos participantes do estudo de caso foi variada, no entanto foi possível perceber uma influência considerável do perfil do usuário nas respostas e no tempo de realização da tarefa.

O participante 1 desenvolveu atividades no ACSO ao longo de 4 anos, atuou no time BahiaRT, e criou em seu trabalho de conclusão de curso a interface que permitiu ao time utilizar os recursos do FFS e do SPlanner. Seu trabalho foi diretamente ligado ao desenvolvimento das capacidades de coordenação comportamental multiagente, e para tal, alcançou um conhecimento sólido a respeito da complexidade que existe em adicionar um novo comportamento à base de inteligência artificial dos agentes em campo. O participante 1 não realizou durante suas atividades no ACSO modificações na interface gráfica da ferramenta SPlanner.

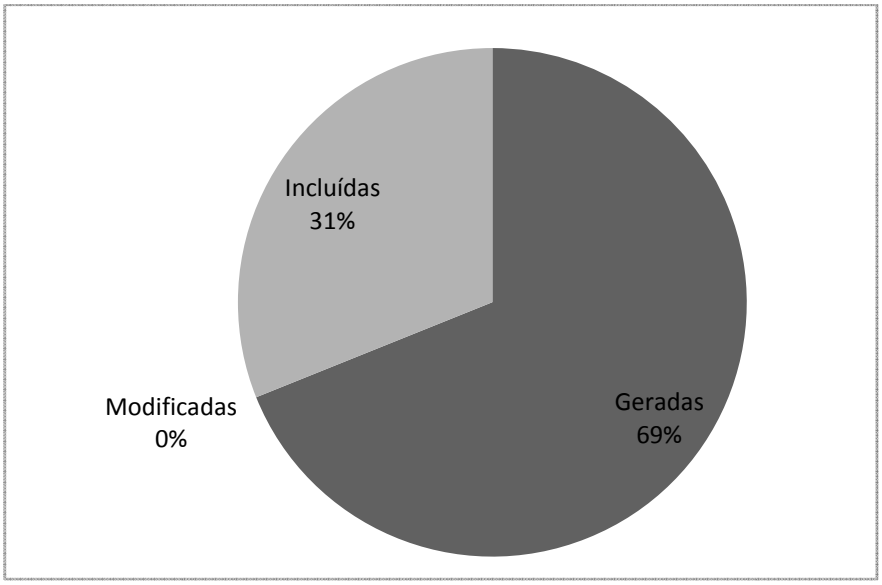
O participante 2 integra a equipe do ACSO há 2 anos e meio, no entanto apenas recentemente entrou para a equipe do time BahiaRT que atua na subliga de simulação 3D, tem experiência com o SPlanner. O participante 3 entrou no ACSO há 9 meses e começou desde então a atuar no time BahiaRT, na subliga de simulação 3D, tem experiência com o FFS.

Segundo o participante 1, a ferramenta cumpriu o propósito de gerar o comportamento e torná-lo acessível ao SPlanner, facilitando essa etapa de desenvolvimento (SPlanner), contribuindo com o entendimento a respeito do fluxo de criação de um comportamento. No entanto não foi possível utilizar o comportamento criado em uma JE e não diminuiu a complexidade de desenvolvimento, por não implementar os métodos das classes *Spporting, ClangAction* e *SetplayAction*, que lidam diretamente com as capacidades e com a base de conhecimento do agente.

Segundo os participantes 2 e 3, foi possível diminuir a complexidade do desenvolvimento de um novo comportamento. O participante 2, por exemplo, está mais familiarizado com a complexidade de edição da interface gráfica da ferramenta SPlanner, visto que seu trabalho atualmente no ACSO lida com uma demanda dessa natureza. Ainda segundo os participantes 2 e 3, foi possível utilizar o novo comportamento em uma JE, pois apesar de não implementar as classes *spfcportugalporting*, *clangaction* e *setplayaction*, utilizando nesses métodos códigos de um comportamento existente, foi possível exportar do SPlanner um arquivo de JE, que pode ser lido pelo FFS.

O gráfico da Figura 24 expõe o resultado da criação de comportamento utilizando a solução proposta, que apresentou uma média total de 133 linhas de código geradas, em contrapartida às 60 linhas de código que ainda precisaram ser acrescentadas para completar a criação do comportamento. Para responder a questão de pesquisa 1, foi aplicada a métrica da Figura 23 ao resultado da contagem de linhas geradas e adicionadas, obtendo a amplitude da ferramenta com o valor final 0,68.

Figura 24 - Linhas de código



Fonte: O autor.

Em relação à questão de pesquisa 2, não foi possível comparar os tempos do desenvolvimento de um novo comportamento com e sem a solução proposta, pois foram utilizados códigos existentes na etapa de adicionar os códigos que a ferramenta não gera. Contudo o tempo gasto pelos participantes na primeira etapa (geração automática) apresentou uma variação de 3 a 10 minutos, dependendo do nível de familiaridade do entrevistado com os parâmetros solicitados pelo Gerador, resultando em uma média final de 5 minutos e meio, um tempo razoavelmente baixo para programação de 133 linhas de código.

A etapa da criação do comportamento que consiste em acrescentar as linhas de código que não são geradas de forma automática, foi realizada pelo desenvolvedor da solução proposta, o mesmo que aplicou o estudo de caso e apresentou uma média de tempo de 4 minutos. Nessa fase foram acrescentados os códigos do comportamento *Dribble*, já existente nas ferramentas FFS e SPlanner. Portanto, não foi possível precisar o tempo que seria gasto para desenvolver este código do zero.

Não foi necessário alterar os códigos gerados, apenas acrescentar os que a ferramenta não gera, que corresponde à implementação dos métodos das classes *spfcportugalporting*, *clangaction* e *setplayaction*.

As 3 etapas de verificação da Tabela 8 foram realizadas após a criação dos comportamentos, submetendo-os à análise de seus arquivos código a código (etapa 1) e aos testes práticos na ferramenta SPlanner (etapa 2) e na execução do time BahiaRT (etapa 3). Para a realização das etapas 2 e 3 foi necessário recompilar os códigos do SPlanner e do time BahiaRT, para que o novo comportamento pudesse ser reconhecido por ambos.

A etapa 2 consistiu em criar uma JE utilizando o novo comportamento na ferramenta SPlanner. O SPlanner só exporta o arquivo da JE nos casos em que os códigos dos comportamentos utilizados nessa jogada estejam sintaticamente corretos. Passando pela exportação do arquivo com sucesso, a JE pôde passar para a etapa 3 de verificação.

Na etapa 3 de verificação a JE foi testada na execução do time BahiaRT. Para realizar essa verificação, foi necessário adotar a mesma versão do time BahiaRT utilizada em [6], pois essa versão foi configurada para alterar o fluxo de jogadas padrão dos agentes do time para um conjunto controlado de casos de teste. O time executou com sucesso a JE criada utilizando o novo comportamento, garantindo a validação semântica dos códigos gerados pela solução proposta.

Um conjunto completo de casos de teste utilizando todas as combinações possíveis dos parâmetros solicitados na interface com o desenvolvedor (Figura 15) pode ser visualizado no Apêndice E. O conjunto de testes do Apêndice E foram realizados fora do estudo de caso, validados apenas no SPlanner. Com relação à saída esperada (desenho na tela do SPlanner, condição para seleção do comportamento no menu de seleção do SPlanner e resultado da ação no campo), todos os resultados foram positivos.

10. CONSIDERAÇÕES FINAIS

Este trabalho propôs um gerador de comportamento para auxiliar o desenvolvimento de jogadas ensaiadas para o time BahiaRT nas competições da Robocup. O gerador permitiu que novos códigos fossem gerados de forma automática a partir de uma interface visual de parametrização, melhorando a produtividade do desenvolvimento de novos comportamentos.

Os testes, realizados sob formato de estudo de caso, mostraram que 68% dos códigos finais de um novo comportamento puderam ser gerados de forma automática. Em um contato inicial com a ferramenta, participantes atingiram uma performance média de 5 minutos e meio para finalizar a etapa de geração automática do comportamento.

A Solução contribui mais efetivamente com o SPlanner, os códigos gerados para o FFS correspondem à métodos sem implementação. A implementação desses métodos variam de acordo com a capacidade e a inteligência do agente e possuem um nível de especificidade que foge à proposta de uma ferramenta de alto nível de abstração, portanto devem ser desenvolvidos e adicionados manualmente após a etapa de geração.

Acredita-se que o participante 1 da validação (seção 9), apesar de ter experiência sólida no *Framework* FCPortugal *Setplays*, não visualizou ganhos efetivos com a solução proposta pois seu trabalho no ACSO não estava ligado à edição da ferramenta SPlanner. Isso pode ser evidenciado pela resposta do participante 2, que atualmente desenvolve no ACSO um trabalho de edição da ferramenta SPlanner, e observou um ganho mais efetivo com relação ao quesito diminuição da complexidade de desenvolvimento.

A Solução mostrou-se capaz otimizar o processo da criação de novos comportamentos para o *Framework* FCPortugal *Setplays* e seu módulo gráfico de jogadas ensaiadas SPlanner utilizando a geração automática de código, que busca aumentar a produtividade e a qualidade do desenvolvimento**.** A Solução inclui artefatos que tencionam diminuir a curva de aprendizado dos desenvolvedores que irão realizar essa tarefa.

10.1. Trabalhos futuros

A solução proposta nessa monografia pode ser vista como o início de um trabalho de inclusão de ferramentas de alto nível de abstração que dêem suporte ao ambiente do time BahiaRT. Foram identificados os seguintes trabalhos para dar continuidade ao projeto já realizado:

- Realizar testes com uma quantidade maior de pessoas.
- Realizar geração automática dos códigos voltados para a capacidade e inteligência artificial do agente, que foram incluídos manualmente na fase de validação da Solução realizada na seção 9.
- Verificar se existe modalidades de comportamento que não foram mapeados para geração na ferramenta.
- Analisar outros elementos do *Framework* FCPortugal *Setplays* e de seu módulo SPlanner que possam ser incluídos em uma ferramenta de edição ou criação de código similar a que foi desenvolvida neste trabalho.

REFERÊNCIAS BIBLIOGRÁFICAS

[1] ROBOCUP. O que se pesquisa na RoboCup?. 2018. Acessado em 17 de julho de 2018, de http://www.robocup.org.br/pesquisa.php.

[2] ROBOCUP. A Brief History of RoboCup.2018.Acessado em 17 de julho de 2018, de www.robocup.org/a_brief_history_of_robocup.

[3] ROBOCUP. Main Page. 2018. Acessado em 17 de julho de 2018, de www.wiki.robocup.org/Main_Page.

[4] ROBOCUP. Soccer SimulationLeague. 2018. Acessado em 17 de julho de 2018, de www.wiki.robocup.org/Soccer_Simulation_League.

[5] ACSO. BahiaRT. 2017. Acessado em 17 de abril de 2018, de www.acso.uneb.br/bahiart/.

[6] RAMOS, C.E.R.; Planejador Multiagentes Para Criação De Jogadas Ensaiadas Em Um Time De Futebol De Robôs Simulados.2017. Tese de Conclusão de Curso – Universidade do Estado da Bahia, Salvador, 2017.

[7] João Cravo, Fernando Almeida, Pedro Henriques Abreu, Luís Paulo Reis, Nuno Lau, Luís Mota, Strategy planner: Graphical definition of soccer set-plays, Data & Knowledge Engineering, Volume 94, Part A, 2014, Pages 110-131, ISSN 0169-023X.

[8] ROBOCUP. Histórico de edições internacionais da RoboCup. 2017. Acessado em 17 de abril de 2017, de www.robocup.org.br/historico_edicoes.php

[9] ROBOCUP. The Standard Problem. 2017. Acessado em 17 de abril de 2018, de www.robocup.org/objective

[10] RUSSEL, S. J.; NORVIG, P. Inteligencia Artificial. [S.l.]: Pearson Education, 2003.

[11] Cravo, J. G. B. Splanner - Uma aplicação gráfica de definição flexível de jogadas estudadas no RoboCup. 2011. Tese de Mestrado – Universidade do Porto, Porto, 2011.

[12] ROBOCUP. RoboCup Soccer - Simulation 3D. 2017. Acessado em 17 de abril

de 2018, de www.robocup.org/leagues/25.

[13] ALDEBARAN Robotics. 2017. Acessado em 17 de abril de 2017, 2017, de doc.aldebaran.com/2-1/home_nao.html.

[14] NODA, I.; MATSUBARA, H.; HIRAKI, K.; FRANK, I. Soccer server: A tool for research on multiagent systems. Applied Artificial Intelligence, Taylor & Francis, v. 12, n. 2-3, p. 233–250,1998.

[15] STONE, P. The robocup soccer server and CMUnited clients: Implemented infrastructure for MAS research. Itsuki Noda Electrotechnical Laboratory. 2000.

[16] MOTA, L., REIS, L.P.: Setplays: Achieving coordination by the appropriate use of arbitrary pre-defined flexible plans and inter-robot communication. In Winfield, A.F.T., Redi, J., eds.: First International Conference on Robot Communication and Coordination (ROBOCOMM 2007). Volume 318 of ACM International Conference Proceeding Series.,IEEE (2007)

[17] SACERDOTI, E. D.; What Language Understanding Research Suggests About Distributed Artificial Intelligence. In: Proc. Distributed Sensor Nets Workshop, Pittsburgh, Pennsylvania, 1978, Carnegie-Mellon University.

[18] Konolige, K. (1982). A first order formalization of knowledge and action for a multi-agent planning system. In Hayes, J. E., Michie, D., and Pao, Y.-H. (Eds.), Machine Intelligence 10. Ellis Horwood.

[19] Kurt Konolige and Nils J. Nilsson. 1980. Multiple-agent planning systems. In Proceedings of the First AAAI Conference on Artificial Intelligence (AAAI'80). AAAI Press 138-142.

[20] Weiss, G. (2000a). Multiagent systems - A Modern Approach to Distributed Artificial Intelligence.MIT Press.

[21] SICHMAN, J. S.; DEMAZEAU, Y. Exploiting social reasoning to deal with agency level inconsistency. In: ICMAS. [S.l.: s.n.], 1995. p. 352–359.

[22] Mota, L. ; Fabro, J. A. ; Reis, L. P. ; Lau, N. Collaborative Behavior in Soccer:

The Setplay Free Software Framework. In: The 18th annual RoboCup International Symposium, 2014, Joao Pessoa-PB-Brazil.

[23] L. P. Reis, R. Lopes, L. Mota and N. Lau, "Playmaker: Graphical definition of formations and setplays," 5th Iberian Conference on Information Systems and Technologies, Santiago de Compostela, 2010, pp. 1-6.

[24] L. Mota, N. Lau and L. P. Reis, "Co-ordination in RoboCup's 2D simulation league: Setplays as flexible, multi-robot plans," 2010 IEEE Conference on Robotics, Automation and Mechatronics, Singapore, 2010, pp. 362-367.

[25] Marques, F. T. (2010). Generic coordination methodologies applied to the robocup simulation leagues. Master'sthesis, Faculdade de Engenharia da Universidade do Porto, Porto.

[26] Reis, Cláudia & Laranjeira, Camila & Jr, Sergio & Simões, Marco & Frias, Diego & Souza, Josemar. (2015). Posicionamento Estratégico em Situações de Cobrança de Escanteio para um Time de Futebol de Robôs Autônomos. 58-67.

[27] DURFEE, E. H. Distributed problem solving and planning. In: Multi-agent systems and applications. [S.l.]: Springer, 2001. p. 118–149.

[28] RUSSEL, S. J.; NORVIG, P. Inteligencia Artificial 3.ed. [S.l.]: Pearson Education, 2013.

[29] Hidehisa Akiyama and Hiroki Shimora.Helios2010 team description. In RoboCup 2010: Robot Soccer World Cup XIV, volume 6556 of Lecture Notes in Computer Science. Springer, 2011.

[30] Football Manager 2018. 2018. Acessado em 17 de abril de 2018, de www.footballmanager.com/games/football-manager-2018.

[31] Habibi, Jafar&Chiniforooshan, Ehsan&Heydarnoori, Abbas &Mirzazadeh, Mehdi& Safari, Msm&Younesy, Hamid. (2002). Coaching a Soccer Simulation Team in RoboCupEnvironment. 117-126. 10.1007/3-540-36087-5_14.

[32] Luís Paulo Reis, Coordenação em Sistemas Multi-Agente: Aplicações na Gestão Universitária e Futebol Robótico (Coordination in Multi-Agent Systems: Applications in University Management andRobotic Soccer), PhD Thesis, FEUP, July of 2003

[33] Rui Manuel Figueiredo de Almeida, Análise e Previsão das Formações das Equipas no Domínio do Futebol Robótico

[34] Luís Henrique RamiloMota, Multi-robot Coordination using Flexible Setplays: Applications in RoboCup's Simulation and Middle-Size Leagues.

[35] Patrick Riley. 2005. Coaching: Learning and Using Environment and Agent Models for Advice. Ph.D. Dissertation.Carnegie Mellon Univ., Pittsburgh, PA, USA. AAI3166283.

[36] P. D. O'Brien and R. C. Nicol. 1998. FIPA — Towards a Standard for Software Agents. BT Technology Journal 16, 3 (July 1998), 51-59.DOI=http://dx.doi.org/10.1023/A:1009621729979

[37] RoboCup Soccer, HumanoidLeague, Lawsofthe Game - 2017/2018.Acessado em 17 de abril de 2018, de https://www.robocuphumanoid.org/wp-content/uploads/RCHL-2018-Rules-Proposal_changesMarked_final.pdf.

[38] FC Portugal 3D Simulation Team: TeamDescription Paper. Acessado em 23 de junho de 2018, de https://www.robocup2017.org/file/symposium/soccer_sim_3D/FCPortugal3D_TDP_2017.pdf

[39] Reis, Luís Paulo and Nuno Lau. FC Portugal Team Description: RoboCup 2000 Simulation League Champion.RoboCup (2000).

[39] Dias, R., Amaral, F., Azevedo, J.L., Cunha, B., Dias, P., Lau, N., Neves, A.J., Pedrosa, E., Pereira, A., Silva, J.D., &Trifan, A. (2016). CAMBADA ' 2016 : Team Description Paper. RoboCup (2016).

[40]Laue, T. et al, B-Human - Team Description for RoboCup 2017.Robocup 2017.

[41] Hofmann, M., Gurster, F. GOL - A Language to Define Tactics in Robot Soccer. The 10th Workshop on Humanoid Soccer Robotsat 15th IEEE-RAS International

Conference on Humanoid Robots Seoul, Korea, November 3rd 2015.

[42] M. Loetzsch, M. Risler and M. Jungel, "XABSL - A Pragmatic Approach to Behavior Engineering," 2006 IEEE/RSJ International Conference on Intelligent Robots and Systems, Beijing, 2006, pp. 5124-5129.doi: 10.1109/IROS.2006.282605

[43] Akiyama H., Dorer K., Lau N. (2015) On the Progress of Soccer Simulation Leagues. In: Bianchi R., Akin H., Ramamoorthy S., Sugiura K. (eds) RoboCup 2014: Robot World Cup XVIII. RoboCup 2014. Lecture Notes in Computer Science, vol 8992. Springer, Cham

[44] Chen, M., E. Foroughi, F. Heintz, S. Kapetanakis, K. Kostadis, J. Kummeneje, I. Noda, O. Obst, P. Riley, T. Steffens, and Y. W. X. Yin (2003). Users manual: Robocup soccer server manula for soccer serverversion 7.07 and later. Disponível em: http://sourceforge.net/projects/sserver/.

[45] Tomoichi Takahashi. Kasugabito III. In M. Veloso, E. Pagello, and H. Kitano,editors, RoboCup-99: Robot Soccer World Cup III, pages 592–595. SpringerVerlag, Berlin, 2000.

[46] Luis Paulo Reis and Nuno Lau. COACH UNILANG - a standard language for coaching a(robo)soccer team. In Andreas Birk, Silvia Coradeschi, and Satoshi Tadokoro, editors,RoboCup-2001: Robot Soccer World Cup V, volume 2377 of Lecture Notes in ArtificialIntelligence, pages 183–192. Springer-Verlag, 2002. 7.4

[47]RoboCup Technical Committee.RoboCup Standard Platform League (NAO) Rule Book, 2018 rules,Robocup 2018.

[48] Lau, N., Reis, L. P., Shafii, N., Ferreira, R., Abdolmaleki, A.,FC Portugal 3D Simulation Team: TeamDescription Paper. Robocup 2014.

[49] WELCOME TO ROBOCUP. 2018. Acessado em 17 de julho de 2018, de https://www.robocup.org.

[50] ROBOCUP 3D SIMULATION. 2018. Acessado em 17 de julho de 2018, de http://www.cs.utexas.edu/~AustinVilla/sim/3dsimulation/.

[51] Reis, Cláudia & Laranjeira, Camila & Jr, Sergio & Simões, Marco & Frias, Diego & Souza, Josemar. (2015). Posicionamento Estratégico em Situações de Cobrança de Escanteio para um Time de Futebol de Robôs Autônomos. 58-67.

[52] ROBOCUP. Objective. 2018. Acessado em 17 de julho de 2018, de www.wiki.robocup.org/objective.

[53] R. Dias, F. Amaral, J. L. Azevedo, B. Cunha, P. Dias, M. Fiore N. Lau, A. J.R. Neves, E. Pedrosa, A. Pereira, F. Pinto, D. Silva, J. Silva. CAMBADA'2018: Team Description Paper. 2018, Intelligent Robotics and Intelligent Systems Lab IEETA/DETI – University of Aveiro, Portugal.

[54] L. Almeida, J.L. Azevedo, P. Bartolomeu, E. Brito, M.B. Cunha,J.P. Figueiredo, P. Fonseca, C. Lima, R. Marau, N. Lau, P. Pedreiras, A. Pereira,A. Pinho, F. Santos, L. Seabra Lopes, J. Vieira. CAMBADA: Team DescriptionPaper. 2004, TransverseActivityonIntelligentRobotics IEETA/DET – Universidade de Aveiro3810-193 Aveiro, Portugal.

[55] FCPORTUGAL FRAMEWORK SETPLAY. 2013. Acessado em 17 de julho de 2018, de http://sourceforge.net/projects/fcportugalsetplays.

[56] THE ROBOCUP SOCCER SIMULATOR. 2016. Acessado em 17 de julho de 2018, de https://sourceforge.net/projects/sserver/.

[57] Standard Platform League. 2018. Acessado em 17 de julho de 2018, de http://wiki.robocup.org/Standard_Platform_League.

[58] Standard platform league overview. 2018. Acessado em 17 de julho de 2018, de http://spl.robocup.org/history/.

[59] FIPA, "FIPA ACL Message Structure Specification", FIPA December 6, 2002.

[60] XAMPP LINUX. 2018. Acessado em 17 de julho de 2018, de https://sourceforge.net/projects/xampp/files/XAMPP%20Linux/7.2.9/.

[61] Solingen, R. Basili, V.;Caldiera,G.; Rombach, H.D. GoalQuestion Metric (GQM) Approach. John Wiley & Sons.Inc.,2002.

[62] RUSSEL, S. J.; NORVIG, P. Inteligencia Artificial. 3. ed. Pearson Education, 2013.

[63] Network Protocol. 2017. Acessado em 16/12/2018, de https://gitlab.com/robocup-sim/SimSpark/wikis/Network-Protocol

[64] Agents. 2017. Acessado em 16/12/2018, de https://gitlab.com/robocup-sim/SimSpark/wikis/Agents

[65] Object Mananged Grounp - OMG. MOF Model to Text Transformation Language Version 1.0. 2008.

[66] Brambilla, M.; Cabot, J.; Wimmer, M. Model-Driven Software Engineering in Practice. Morgan & Claypool Publishers, 2012.

[67] United Modeling Language Infrastructure Specification, Version 2.5. Disponível em: < http://www.omg.org/spec/UML/2.0 >. Acessado em: 26 de novembro de 2017.

APÊNDICES

APÊNDICE A – QUESTIONÁRIO DE VALIDAÇÃO DO GERADOR DE COMPORTAMENTO

O apêndice A apresenta o questionário aplicado na validação da solução proposta, a todos os participantes selecionados. Este questionário tem o objetivo de coletar dados relacionados a avaliação da solução Gerador de Comportamento

Perfil do entrevistado:

Q1. Qual o seu envolvimento com o ACSO?
( ) estudante ( ) especialista

Q2. Está atualmente participando do time de futebol de robôs do ACSO?
( ) sim ( )Não

Q3. Durante quanto tempo participa/participou do time de futebol de robôs do ACSO?

Questões aplicadas a todos os participantes do estudo de caso:

Q1. Sobre a geração do código do comportamento, assinale a alternativa que melhor reflete o resultado do teste realizado.

a) O código não foi gerado
b) O código foi gerado parcialmente
c) O código foi gerado, mas o comportamento não pode ser visto no SPlanner
d) O código foi gerado e o comportamento pode ser visto no SPlanner

Q2. É possível fazer uma jogada ensaiada utilizando esse comportamento?

a) Não é possível utilizar o comportamento em uma jogada ensaiada
b) É possível realizar uma jogada ensaiada, mas visualmente o comportamento não é desenhado na tela do SPlanner
c) É possível realizada uma jogada ensaiada e visualizar corretamente o comportamento no SPlanner

Questões aplicadas ao perfil especialista:

Q3. Diminuiu a complexidade do desenvolvimento?

Q4. Facilitou o trabalho de criação de um novo comportamento?

Q5. Tornou o entendimento do que precisa ser feito mais fácil?

APÊNDICE B – TUTORIAL DE PREPARAÇÃO DO AMBIENTE DA SOLUÇÃO PROPOSTA

Esse apêndice demonstra um roteiro passo a passo para a preparação do ambiente da solução proposta. Esse tutorial inclui a instalação do XAMPP, a criação do ícone de execução do XAMPP e a modificação da localização da pasta que corresponde ao localhost do XAMPP. Para instalar o XAMPP, deve-se seguir os seguintes passos:

1) Baixar a versão mencionada do XAMPP, disponível em [60];
2) Conceder permissão para executar o arquivo de instalação baixado, através do comando no terminal: sudo chmod +x xampp-linux-x64-7.2.9-0-installer.run
3) Executa o instalador no terminal, utilizando: sudo ./ xampp-linux-x64-7.2.9-0-installer.run
4) Será aberta a janela de instalação do XAMPP, em todas elas deve-se proceder clicando em *"Next"*, exceto pela janela indicada na Figura 25, que é importante verificar se a opção *"XAMPP Core Files"*, indicada pelo índice 1 na Figura 25, está marcada, pois é por meio dessa opções que a página do Gerador de Comportamento será executada.
5) Ao terminar a instalação não execute o XAMPP.

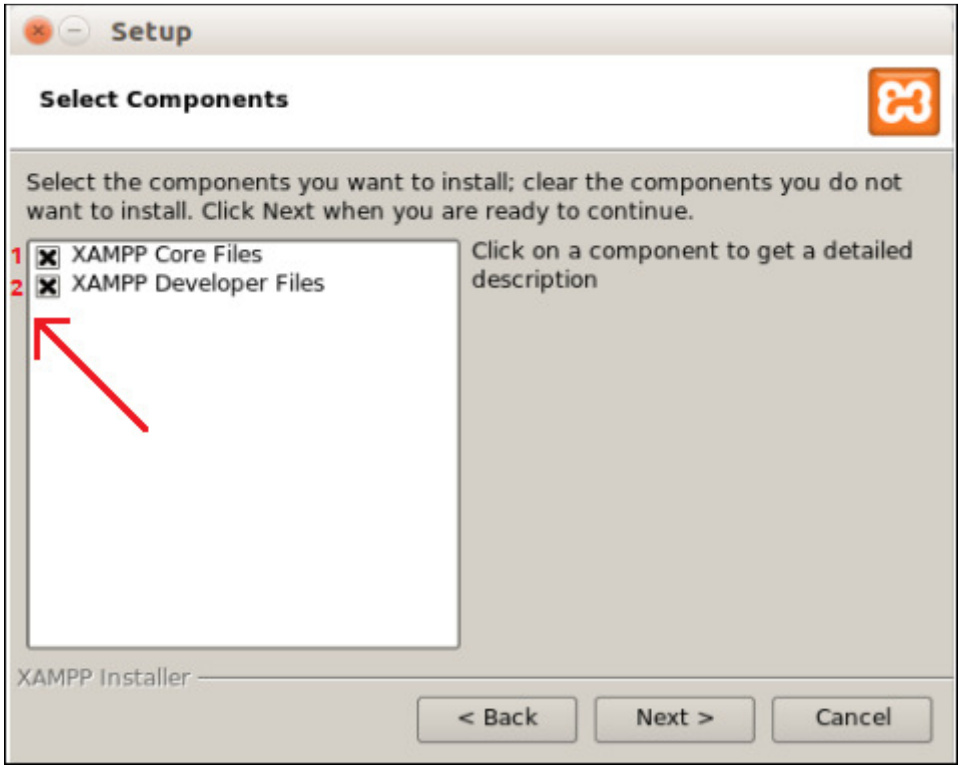
A próxima etapa será criar o atalho de execução do XAMPP, é opcional, mas facilita a inicialização do XAMPP:

1) Ainda no terminal, digite o comando: echo -e '[Desktop Entry]\n Version=1.0\n Name=xampp\n Exec=gksudo /opt/lampp/manager-linux-x64.run\n Icon=/opt/lampp/htdocs/favicon.ico\n Type=Application\n Categories=Application' | sudo tee /usr/share/applications/xampp.desktop
2) Certifique-se de que o gksudo está instalado:
   a. sudo apt-get update
   b. sudo apt-get install gksu
3) Em seguida basta buscar pela palavra xampp na tela de busca do Ubuntu, como pode ser visto na Figura 26, e usá-lo para iniciar e parar o servidor Apache sempre que precisar.

Seguindo o mesmo padrão do FFS e do SPlanner, a pasta do gerador de comportamentos deve ser armazenada dentro do diretório /home/xxx (xxx deve ser substituído pelo nome do usuário), denominado pasta pessoal, onde ficam as pastas Imagens, *Downloads* e Documentos. Sendo assim, é necessário alterar a localização da pasta que corresponde ao diretório *localhost* do XAMPP, onde ficará armazenada a pasta do Gerador de Comportamento, seguindo os seguintes passos:

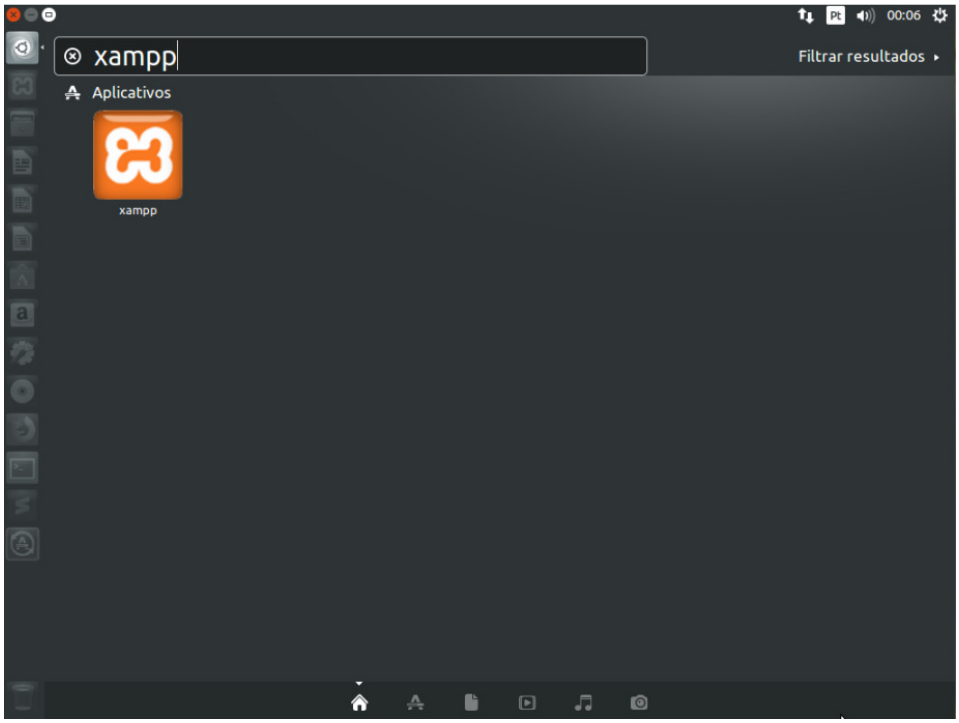
1) Alterar a configuração do XAMPP para reconhecer outra pasta como *htdocs*:
   a. sudo gedit /opt/lampp/etc/httpd.conf
   b. procure (Ctrl + f) pela palavra *htdocs*, nas duas ocorrências, substitua */opt/lampp/htdocs* pelo novo caminho: */home/usuário/htdocs*.
2) Copie a pasta htdocs de /opt/lampp/ para /home/xxx:
   a. sudo cp -Ra /opt/lampp/htdocs /home/xxx
3) Copiar a pasta do Gerador de Comportamento para dentro do diretório /home/xxx/htdocs.
4) Altere a permissão de acesso do seu usuário xxx à nova pasta *htdocs*:
   a. sudo chown -R xxx:xxx /home/xxx/htdocs
   b. sudo chmod -R 777 /home/xxx/htdocs
5) Altere a permissão para o Gerador de Comportamento poder acessar as pastas *bahiart* e *Splanner*:
   a. sudo chmod -R 777 /home/xxx/bahiart
   b. sudo chmod -R 777 /home/xxx/Splanner

Figura 25 – Tela de instalação do XAMPP.



Fonte: Adaptação do instalador do programa XAMPP.

Figura 26 – Tela de busca do Ubuntu 16.01

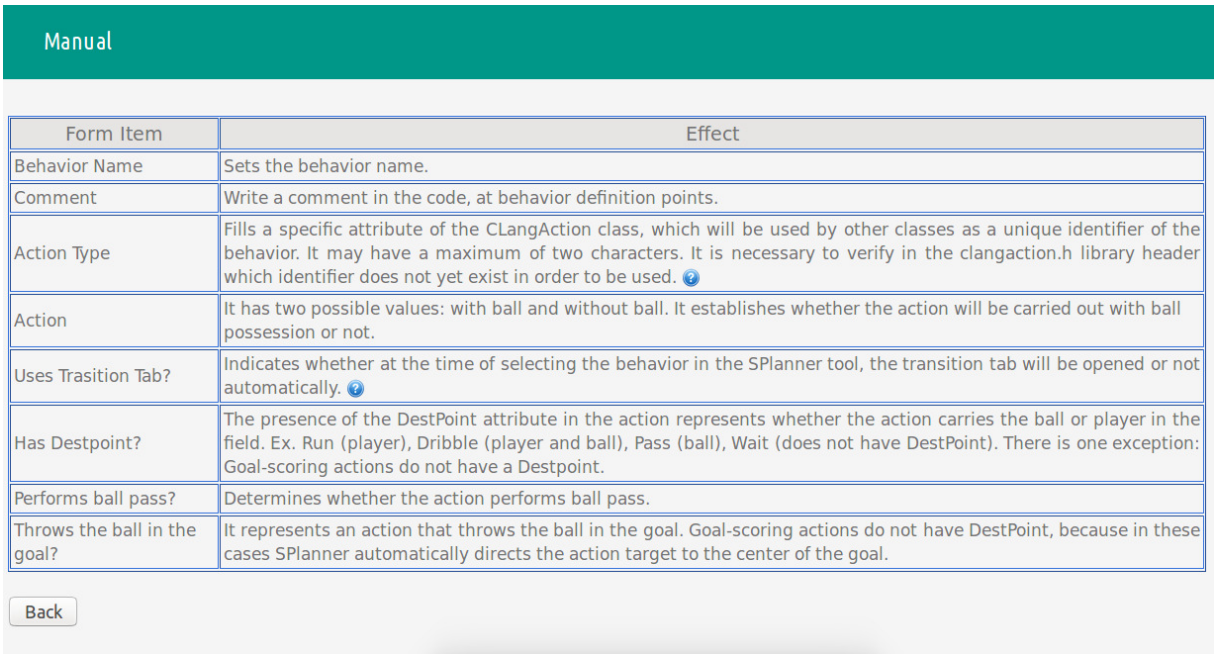


Fonte: Ubuntu 16.01.

APÊNDICE C – MANUAL DO GERADOR DE COMPORTAMENTOS

Esse apêndice apresenta a página configurada como manual de utilização do Gerador de Comportamento, conforme apresentado na seção 8.2. É possível visualizar na Figura 27 a página que é apresentada quando o usuário clica no botão Manual da página principal do gerador, que pode ser visto na Figura 15.

Figura 27 - Página de manual do Gerador de Comportamento.



Fonte: O autor.

APÊNDICE D - GRAMÁTICA DA LINGUAGEM DOS BLOCOS DE CONDIÇÃO ESPECÍFICA

Este apêndice apresenta a definição formal da linguagem apresentada nos blocos de condição específica, conforme estão definidos na seção 8.3.

Nas regras léxicas e sintáticas descritas abaixo os caracteres da notação BNF estão grifados em verde.

- Alternativas são separadas por barras verticais, ou seja, 'a | b' significa "a ou b".
- Colchetes indicam ocorrência opcional: '[a]' significa um a opcional, ou seja, "a | ε" (ε refere-se à cadeia vazia).
- Parênteses indicam ocorrência de uma das alternativas: '(a | b | c)' significa obrigatoriedade de escolha de a ou b ou c.

1. Regras Léxicas

opc1::= “play mode”

opc2::= “setplay step”

opc3::= “on special areas”|“ball stoped”

pmode::= “play on” |“kickoff” |“throw in” | “direct free kick” | “indirect free kick” | “corner kick” | ”goal kick” | “keeper catch”

step::= 0

2. Regras Sintáticas

Símbolos *não-terminais* são apresentados em itálico; símbolos **terminais** são apresentados em negrito e, às vezes, entre aspas por questões de clareza.

2.1 Regras de produção da gramática

Op::= **=**
| **!=**

Op_rel::= **e**
| **ou**

Exp::= [*exp_simp*[*Op_rel exp_simp*] [*Op_rel exp_simp*] [*Op_rel exp_simp*] [*Op_rel exp_simp*]]

Exp_simp::=(**opc1** *Op* **pmode** | **opc2** *Op* **step** | [**!**] **opc3**)

APÊNDICE E - AMBIENTE CONTROLADO DE TESTE DA SOLUÇÃO

Este apêndice apresenta o plano de testes realizado para validar a solução proposta contendo a combinação completa dos parâmetros solicitados na tela de interface com o desenvolvedor (Figura 15).

As tabelas 9, 10 e 11 mostram para cada comportamento (c1, c3, c3, c4, etc.) testado quais os parâmetros selecionados (com bola, sem bola, etc.). A penúltima linha de cada tabela mostra a saída esperada do sistema na realização dos casos de teste. A última linha de cada tabela mostra o resultado real observado (saída obtida) quando o teste foi realizado. Como pode ser visto, todos os testes tiveram resultado positivo (OK).

Tabela 9 - Grupo 1 de casos de teste

		Comportamento			
		C1	**C2**	**C3**	**C4**
Parâmetros	Com Bola	X	X	X	X
	Sem Bola				
	Com Transição	X		X	
	Sem Transição		X		X
	Com *Destpoint*	X	X		
	Sem *Destpoint*			X	X
	Com Passe	X	X		
	Sem Passe			X	X
	Com lance a gol			X	X
	Sem lance a gol	X	X		
	Saída esperada	Ação de passe com Transição	Ação de passe sem transição	Ação de lance a gol com transição	Ação de lance a gol sem transição
	Saída obtida	Ok	Ok	Ok	Ok

Fonte: O autor.

Tabela 10 - Grupo 2 de casos de teste

		Comportamento			
		C5	**C6**	**C7**	**C8**
Parâmetros	Com Bola	ɣ	ɣ		
	Sem Bola			ɣ	ɣ
	Com Transição	ɣ		ɣ	
	Sem Transição		ɣ		ɣ
	Com *Destpoint*	ɣ	ɣ	ɣ	ɣ
	Sem *Destpoint*				
	Com Passe				
	Sem Passe	ɣ	ɣ	ɣ	ɣ
	Com lance a gol				
	Sem lance a gol	ɣ	ɣ	ɣ	ɣ
	Saída esperada	Ação com movimentação com posse de bola, com transição	Ação com movimentação com posse de bola, sem transição	Ação com movimentação sem posse de bola, com transição	Ação com movimentação sem posse de bola, sem transição
	Saída obtida	Ok	Ok	Ok	Ok

Fonte: O autor.

Tabela 11 - Grupo 3 de casos de teste

		Comportamento			
		C9	**C10**	**C11**	**C12**
Parâmetros	Com Bola	ɣ	ɣ		
	Sem Bola			ɣ	ɣ
	Com Transição	ɣ		ɣ	
	Sem Transição		ɣ		ɣ
	Com *Destpoint*				
	Sem *Destpoint*	ɣ	ɣ	ɣ	ɣ
	Com Passe				
	Sem Passe	ɣ	ɣ	ɣ	ɣ
	Com lance a gol				
	Sem lance a gol	ɣ	ɣ	ɣ	ɣ
	Saída esperada	Ação sem movimentação, com posse de bola, com transição	Ação sem movimentação, com posse de bola, sem transição	ação sem movimentação e sem posse de bola, com transição	ação sem movimentação e sem posse de bola, sem transição
	Saída obtida	Ok	Ok	Ok	Ok

Fonte: O autor.