UNIVERSIDADE DO ALGARVE

SISTEMA DE INFORMAÇÃO PARA A REPRESENTAÇÃO E DETEÇÃO DE MODELOS DE JOGO NO FUTEBOL

ANTÓNIO MIGUEL GONÇALVES BELGUINHA

Dissertação

Mestrado em Engenharia Elétrica e Eletrónica

Trabalho efetuado sob a orientação de:
Professor Doutor Pedro Cardoso & Professor Doutor João Rodrigues

2014

SISTEMA DE INFORMAÇÃO PARA A REPRESENTAÇÃO E DETEÇÃO DE MODELOS DE JOGO NO FUTEBOL

Declaração de autoria de trabalho

Declaro ser o autor deste trabalho, que é original e inédito. Autores e trabalhos consultados estão devidamente citados no texto e constam da listagem de referências incluída.

©2014, ANTÓNIO MIGUEL GONÇALVES BELGUINHA

A Universidade do Algarve tem o direito, perpétuo e sem limites geográficos, de arquivar e publicitar este trabalho através de exemplares impressos reproduzidos em papel ou de forma digital, ou por qualquer outro meio conhecido ou que venha a ser inventado, de o divulgar através de repositórios científicos e de admitir a sua cópia e distribuição com objetivos educacionais ou de investigação, não comerciais, desde que seja dado crédito ao autor e editor.

Resumo

Excelência é a palavra-chave usada para todos os envolvidos no futebol. Tal como os jogadores precisam de mostrar as suas habilidades físicas e táticas em campo, os treinadores e equipas técnicas precisam de ferramentas que lhes permitam garantir o desempenho dos jogadores ao mais alto nível. Conceder a todas as partes um sistema de informação multifuncional, com o objetivo de minimizar os efeitos adversos dos pontos mais sensíveis e críticos do futebol é, então, fundamental. É nesse sentido que esta dissertação foca uma série de ferramentas que visam auxiliar os treinadores e equipas técnicas na deteção de modelos de jogo, na deteção trocas de bola, na visualização de mapas de passes, de perdas de bola e de calor, facilitando assim o trabalho das equipas técnicas no que se refere à análise tanto à sua equipa como às equipas adversária. Estas ferramentas estão integradas num sistema de informação multiplataforma, recorrendo às mais recentes tecnologias de desenvolvimento de aplicações web e de bases de dados, trabalhando com informação vinda de um sistema automático de *tracking*. As ferramentas apresentadas estão incluídas num Sistema Integrado de Gestão para o Futebol (SIGF) que está a ser desenvolvido no projeto QREN I&DT - FootData.

Palavras-chave: Linguagem Programação Visual, Deteção de Modelo de Jogo, Deteção de Trocas de Bola, Python, Mapas de Calor, Futebol.

Abstract

Excellence is the keyword to everyone involved in soccer. While players need to show their physical and tactical skills on the field, coaches and technical staff need tools that allow them to guarantee the performance of the players at the highest level. Granting to all the parts a multifunctional information system, with the goal of minimizing the adverse effects of the most sensitive and critical points of football is then crucial. In that sense, this dissertation focuses on a series of tools that aim to assist coaches and technical staff in the detection of game models, the detection of ball exchanges, the visualization of passes, loss balls and heat maps, thus making the work of technical teams easier in regard to the analysis of both his team and the opposing teams. These tools are integrated in an cross platform information system, using the latest web applications and databases development technologies, working with information coming from an automatic *tracking* system. All these tools are included in an integrated system aimed for Football Resource Planning (FRP) which is being developed in the QREN I&DT - FootData project.

Keywords: Visual Programing Language, Game Model Detection, Ball Exchanges Detection, Python, Heat Maps, Soccer.

Dedico esta dissertação aos meus pais e à minha irmã, pelo que me ensinaram e por todo o apoio que me deram. À minha restante família e amigos. E ao mais recente membro da família, a minha sobrinha Beatriz.

Agradecimentos

Gostaria de agradecer em primeiro lugar ao meu orientador, Professor Doutor Pedro Cardoso, e ao meu co-orientador, Professor Doutor João Rodrigues, por toda a ajuda que me deram ao longo do meu percurso académico, pela disponibilidade que demonstraram e pelos conhecimentos que me facultaram para a elaboração desta dissertação.

À minha mãe e ao meu pai por me terem dado a oportunidade de estudar na Universidade do Algarve e pelo apoio dado desde sempre.

À Cristiana, a minha namorada, por me apoiar e aturar.

Devo ainda um agradecimento aos meus colegas de projeto Carlos Gomes, Tiago Sousa e Pedro Rodrigues pelo contributo que tiveram nas ferramentas aqui apresentadas.

Quero ainda agradecer aos meus amigos e colegas de curso: Kevin Santos, Bruno Fonseca e Bruno Correia pelo apoio e pelos vários momentos vividos que nunca mais serão esquecidos.

Por fim, mas não menos importante, um agradecimento a todos os meus colegas de curso que sempre me acompanharam e apoiaram neste meu percurso académico. E aos colegas do laboratório de projeto pelos bons momentos passados no decorrer de todo o projeto.

Conteúdo

Lista de Tabelas

Lista de Figuras

Lista de Abreviaturas

API	*Aplication Programming Interface*
BSON	*Binary JSON*
CSS	*Cascading Style Sheets*
DOM	*Document Object Model*
ED	*Esquema Dinâmico*
EE	*Esquema Estático*
HTML	*Hypertext Markup Language*
HTML5	*Hypertext Markup Language version 5*
JS	*JavaScript*
JSON	*JavaScript Object Notation*
LPV	*Linguagem de Programação Visual*
SIGF	*Sistema Integrado de Gestão para o Futebol*
SVG	*Scalable Vector Graphics*
XML	*eXtensible Markup Language*

1

Introdução

O futebol é um desporto onde todas as entidades envolvidas (jogadores, equipas técnicas e agentes) estão numa constante busca por melhores resultados. Não só os jogadores têm de mostrar as suas qualidades no campo, mas também o treinador e a restante equipa técnica necessitam de ter as suas próprias ferramentas de forma a desempenharem o seu trabalho nas melhores condições possíveis. Por outras palavras, as atuais exigências de rendimento no futebol tornam imperativo o recurso a novas tecnologias de observação e análise, tanto na formação como no alto rendimento (Castelo, 2014).

Tendo este facto em conta e associando-o à constante evolução que se tem verificado nos Sistemas de Informação (SI), é crucial disponibilizar a ambas as partes (jogadores e equipas técnicas) uma solução assente num SI multifuncional com o objetivo

de otimizar recursos e resultados que possam provir dos pontos mais críticos e sensíveis do futebol. Por exemplo, a análise do que decorre durante um jogo de futebol gera uma enorme quantidade de informação e, consequentemente, é importante haver uma forma de a filtrar e fornecer ao treinador em cada instante e em função da atividade que esteja a decorrer (treino ou competição), a informação mais importante. Outro ponto crítico, é o facto de que as equipas de hoje, na sua maioria, têm nos seus quadros jogadores e *staff* de diversas nacionalidades, o que pode ser um obstáculo à comunicação e ao correto entendimento entre todas as partes, pelo que, um sistema tecnológico, com todos os mais variados aspetos e componentes, pode ajudar a diminuir essa barreira.

1.1 Enquadramento

O "FootData: Sistema Integrado de Gestão de Informação para o Futebol" é um projeto QREN I&DT, n.º 23119, que tem como promotor a empresa INESTING - Marketing Tecnológico, SA e co-promotor a Universidade do Algarve. O referenciado projeto tem como objetivo a construção de uma (nova) plataforma *web* para o futebol baseada nas tecnologias de informação e sistemas de comunicação, contemplando duas componentes fundamentais do mundo do futebol: i) Uma rede social, onde são integradas todas as caraterísticas típicas das redes sociais, com informação especializada para os fãs e entre os fãs de todos os clubes, estádios e jogadores, complementada com uma ligação especializada entre os treinadores num fórum exclusivo, bem como uma plataforma dedicada aos empresários do futebol, onde estes podem expor, de uma forma simples mas bastante apelativa, textos, fotos e vídeos dos seus jogadores. A rede social basear-se-á num espírito "*wiki*" onde os utilizadores (jogadores, agentes e treinadores) serão o motor de crescimento e atualização da base de dados. Em paralelo, a plataforma contempla ii) Uma componente profissional, que engloba um sistema de aquisição e gestão de informação transversais a um departamento de futebol, incluindo uma pla-

taforma automatizada para a recolha de informação das equipas, tanto na vertente de competição como na de preparação (treinos). Esta plataforma assenta num protótipo que terá por base o processamento das imagens adquiridas em tempo real (jogo ou treino), ou em diferido a partir da análise de vídeos. Este análise procura fornecer algo mais do que simples compilações estatísticas, na medida em que permitirá a análise da estrutura de um jogo, como por exemplo a computação da "largura e comprimento" de uma equipa, permitindo assim verificar se as ações da equipa, no que diz respeito à ocupação de espaços, estão a ser feitas de modo racional e otimizado. Todavia, o principal objetivo é recolher automaticamente informação relativa ao plano tático e alertar a equipa técnica sobre a ocorrência de determinados eventos no instante em que eles ocorrem.

Ficou consagrado no projeto FootData que este deve ser desenvolvido para ambiente *web*, em que o acesso é feito a partir de um qualquer computador ou dispositivo móvel (*smartphone* ou *tablet*) a partir de um *browser* e que as ferramentas desta plataforma devem ser projetadas e construídas usando tecnologias *open source*.

1.2 Objetivos

Esta dissertação tem como objetivos: i) O projeto e desenvolvimento de uma aplicação capaz de detetar os vários desenhos que compõem o modelo de jogo; ii) O projeto e desenvolvimento de um algoritmo capaz de detetar passes e perdas de bola; iii) A geração de mapas de passes, de perdas de bola e de calor para integração com a interface web.

Em termos mais específicos, para cada um do objetivos a projetar e implementar, podemos referir:

Deteção de Modelos de Jogo - A aplicação consiste na implementação de uma Linguagem de Programação Visual (LPV) capaz de passar para Python desenhos do modelo de jogo criados no interface Editor de Campo (Rodrigues, 2014). O código fonte

gerado é depois usado em conjunto com os dados provenientes do sistema automático de deteção, segmentação e *tracking* de jogadores e bola (daqui a diante genericamente designado por *tracking*). Este consiste na aquisição da posição dos jogadores e da bola para cada *frame* de uma jogo/treino de futebol (Sousa, 2012), para assim verificar se a situação previamente desenhada está a acontecer no jogo ou não. A aplicação deve manter um registo de quando o desenho se verificou e de quando este não se verificou, sendo que para esta última situação terá que ser guardado o que falhou. A aplicação deve ter um desempenho suficiente para processar os desenhos em tempo real de modo a que em qualquer altura se possa de forma instantânea testar novos modelos de jogo.

Algoritmo de Deteção de Trocas de Bola - Este algoritmo deve ser capaz de detetar com o mínimo erro possível as trocas de bola efetuadas por ambas as equipas durante o jogo. As trocas de bola serão depois usadas para obter os passes e as perdas de bola efetuadas pelas equipas. Mais uma vez, os dados de entrada provêm do sistema automático de *tracking*. Também neste caso, a implementação do algoritmo deve ter desempenho suficiente para conseguir detetar os passes e as perdas de bola em tempo real.

Geração de Mapas de Passes, de Perdas de Bola e de Calor - Com este objetivo pretende-se gerar um mapa (imagem), dos passes, das perdas de bola e dos mapas de calor tendo como fundo um campo de futebol. Os passes devem ser representados como setas de cor diferente para cada equipa, nas extremidades devem ter o número do jogador que efetuou o passe e do jogador que recebeu e devem-se encontrar posicionadas no campo de modo a estarem situadas na posição onde foi efetuado o passe e onde foi recebido. As perdas de bola devem ser representadas como bolas e colocadas na localização onde a bola foi perdida, esta deve ter uma cor distinta para cada equipa. Nos mapas de calor, deve ser possível selecionar uma equipa, um jogador ou vários jogadores, em que estes devem ter cores diferentes para facilmente serem distinguidos no mapa de calor. Todos estes mapas devem ser representados num formato de

imagem vetorial, para facilmente se adaptarem a várias resoluções em ambiente web.

1.3 Perspetiva global

No presente capítulo foi introduzido o tema e explicitados os objetivos, as principais contribuições e o enquadramento desta dissertação. No Capítulo 2 é apresentado o estado da arte, onde são dados a conhecer alguns dos produtos existentes no mercado, os quais se encontram relacionados de alguma forma com esta dissertação. Serão também apresentados e explicados os principais conceitos e tecnologias abordadas. No Capítulo 3 é apresentado o desenvolvimento de cada uma das funcionalidades individualmente e de forma detalhada, onde ainda são incluídas algumas comparações com produtos já existentes para o mesmo efeito. No Capítulo 4 são apresentados alguns testes e resultados às funcionalidades implementadas e por fim, no Capítulo 5 as conclusões, considerações finais e as metas para trabalhos futuros.

2

Estado da arte e conceitos gerais

Neste capítulo serão apresentados alguns produtos existentes no mercado em utilização relacionado com o trabalho desenvolvido. Além disso, serão apresentados alguns conceitos gerais referentes às linguagens de programação visual, tecnologias usadas nomeadamente o Python, MongoDB e JSON e por fim uma comparação entre o SVG e HTML5 Canvas.

2.1 Produtos Existentes no Mercado

Atualmente, existem no mercado diversos produtos que conjugam algumas das funcionalidades que procuramos implementar nas ferramentas que apresentamos nesta

dissertação. Procurámos incidir principalmente sobre os produtos mais usados no mundo do futebol.

Existem algum produtos semelhantes ao que pretendemos implementar com a Deteção de Passes e Perdas de bola, dos quais se destacam: o *Prozone MatchViewer* (MatchViewer, 2014), que fornece aos utilizadores uma ferramenta de análise pós-jogo, em que é possível aceder a uma análise tática e técnica sobre a equipa e os jogadores, estando incluído nessas funcionalidades os passes dos jogadores e perdas de bola. No entanto, não foi possível verificar se a obtenção dos passes era feita automaticamente ou manualmente. Já o projeto *ASPOGAMO* (ASPOGAMO, 2014), contém um sistema de *tracking* e também uma faz alguma análise de jogo, na qual está inserida um algoritmo automático para deteção de passes (Beetz et al., 2009). Em (Yu et al., 2003) encontramos um trabalho relacionado com o desenvolvimento de um sistema de *tracking* de jogadores e bola, juntamente com a deteção automática de evento tal como o golo, posse de bola, passes, entre outros.

Relativamente aos produtos semelhantes em que seja possível detetar os Desenhos do Modelo de Jogo, tanto quanto foi possível verificar de momento não existe nenhum no mercado, existindo apenas alguns em que é possível desenhar o modelo de jogo. Desses produtos destacam-se: o K-PlayMaker (K-PlayMaker, 2014), o TacticalPad (TacticalPad, 2014), CoachFX (CoachFX, 2014), o Easy Animation (EasyAnimation, 2014) o eSpor Software Soccer eAssist (eSpor, 2014) e o Tactics Manager Software (Tactics Manager Software, 2014). Relativamente ao K-PlayMaker este é um dos produtos da aplicação Kizanaro focado no desenho de jogadas de futebol. O K-PlayMaker permite a inserção de diversos elementos como pinos, barreiras, entre outros, para além de jogadores e bola nessas jogadas. Oferece ainda a possibilidade de inserção de variantes de um campo de futebol (completo, metade atacante, metade defensiva, entre outras) e permite ainda a gravação e reprodução de jogadas. O TacticalPad é uma plataforma que permite estabelecer posicionamentos e fazer simulações de jogadas de futebol em 2D ou em 3D. O CoachFX contém um vasto leque de produtos para desenho de joga-

das e sessões de treino. Outro dos produtos existentes é o Easy Animation que permite fazer o desenho de jogadas, disponibilizando vários elementos para desenho de sessões de treino, sendo complementadas com uma componente de animação. O eSpor Software Soccer eAssist permite ao treinador de futebol a criação e organização de planos de treino. Fornece várias ferramentas para o estabelecimento de movimentações e possui ainda uma componente de animação. Por último, o Tactics Manager Software é uma aplicação onde é possível fazer a definição de esquemas de futebol estáticos, de movimentos e de zonas em 2D e 3D. Volta-se a referir que tanto quanto foi possível verificar, nenhum dos sistemas conjuga o *tracking* com os esquemas desenhados no sentido de detetar eventuais situações.

2.2 Linguagem de Programação Visual

Em computação, uma linguagem de programação visual (LPV) é uma linguagem de programação que permite aos utilizadores criar programas através de elementos gráficos em vez de elementos textuais. Muitas LPVs são baseadas em "setas e caixas", onde as caixas ou outros elementos semelhantes, são tratados como entidades, ligadas por setas, linhas ou arcos os quais representam as relações entre as entidades. Normalmente as LPVs são usadas em situações onde o utilizador tem um conhecimento de programação limitado.

Por exemplo o Ktechlab (Ktechlab, 2014) é uma LPV para eletrónica e microcontroladores no qual são usados fluxogramas para programar os microcontroladores de forma gráfica. Outro exemplo é o Scratch (Scratch, 2014), (Resnick et al., 2009) que usa uma interface intuitiva composta por blocos com formas especificas, em que só é possível encaixa-los de maneira que faça sentido sintaticamente, permitindo programar várias ações facilmente sem ter quaisquer conhecimentos sobre programação. Devido a estas características, o Scratch está a ser usado com sucesso para ensinar novos programadores e também para fins educacionais, tais como escolas, sendo projetado espe-

cialmente para a faixa etária dos 8 aos 16, mas é usado por pessoas de todas as idades. Já o Blockly (Blockly, 2014) é um projeto Google de uma LPV, desenvolvida para funcionar em ambiente web. O Blockly é muito semelhante ao Scratch, em que apenas se tem que arrastar e largar os blocos de programação de modo a interagir com os objetos visíveis no ecrã do computador. Para programadores mais avançados também existem LPVs, tal como o Simulink® (Simulink, 2014), (Dabney & Harman, 2001). O Simulink é uma ferramenta de modelagem, simulação e análise de sistemas integrada no Matlab®, que funciona com base em diagramas de blocos permitindo o design ao nível do sistema, simulação, e do teste e verificação contínua de sistemas embebidos. O Scicos (Scicos, 2014), (Nikoukhah & Steer, 1996), um modelador e simulador gráfico de sistemas, permite aos utilizadores criar através do uso de diagrama de blocos, modelos e simular o dinamismo de sistemas dinâmicos híbridos, podendo depois ser compilados para código executável. O Scicos é usado para o processamento de sinal, sistemas de controlo e para estudar sistemas físicos e biológicos. Outros exemplos de Linguagens de Programação Visual podem ser encontrados em (Dobesova, 2011), (Lucanin & Fabek, 2011), (Marchiori et al., 2011) e (Tekli et al., 2013).

2.3 Python, JSON, MongoDB

A linguagem de programação escolhida para o desenvolvimento de toda a parte de análise foi o Python (Python, 2014), isto devido à sua vasta lista de bibliotecas (e.g., *numpy*, *SciPy*), que permitem desenvolver código multiplataforma de modo eficiente, facilitando assim o desenvolvimento de aplicações. Além disso, existem diversas *frameworks* web desenvolvidas em Python, o que facilita a futura integração dos processos implementados. Entre essas *frameworks* estão incluídas *frameworks* web *open source*, tais como Django (Alchin et al., 2009) e Flask (Flask, 2014) que como já foi referido, podem ser usadas para integrar a parte de análise de jogo na aplicação web.

O JSON (JavaScript Object Notation) (JSON, 2014), segundo a página oficial, é uma

formatação leve para troca de dados e carateriza-se por ser independente da linguagem. É também bastante intuitiva para os humanos e fácil de interpretar e gerar pelo computador. Além disso, como veremos a seguir é o formato base usado no MongoDB (MongoDB, 2014) para armazenar dados. Com o JSON conseguimos, portanto, alcançar uma estrutura flexível e versátil para os dados em diversos contextos. Um objeto JSON consiste num conjunto de pares desordenados de nome/valor, contidos dentro de { } (chavetas). Cada nome é seguido de ":" (dois pontos), e cada par de nome/valor é separado por "," (vírgula). O campo valor, pode ser uma *string*, *numero*, *array*, *boleano*, *null* ou outro objeto JSON. A Fig. 2.1 exemplifica um objeto JSON.

```
1 {"nome" : "Antonio", "apelido" : "Belguinha"}
```

Figura 2.1: Exemplo de um objeto JSON.

O MongoDB é uma base de dados NoSQL orientada a documentos, de alta performance, com alta fiabilidade e facilmente escalável. Uma base de dados MongoDB consiste num conjunto de coleções onde são inseridos documentos. Estes documentos são objetos BSON (formato binário do documento JSON) permitindo ainda que os documentos tenham um esquema dinâmico, i.e., que os documentos na mesma coleção não tenham que ter o mesmo número de campos ou a mesma estrutura (conjunto de nomes).

Uma vez que o MongoDB usa o BSON para estruturar os documentos, e sendo o JSON muito semelhante ao dicionário Python, a comunicação Python - MongoDB fica assim facilitada, evitando-se deste modo processos complexos para criação e descodificação do JSON. Além disso, tal como o próprio nome indica (JSON - "Notação de Objeto JavaScript") o JSON é integrado diretamente em JavaScript, uma das outras linguagens extensivamente usada no projeto FootData. Estas características fizeram com que o JSON fosse considerado como método mais adequado, e consequentemente usado para troca de informação entre as várias ferramentas aqui apresentadas, permitindo que estes sejam diretamente gravados na base de dados ou facilmente manipu-

lados nas linguagens de programação Python e JavaScript.

2.4 SVG vs HTML5 Canvas

Até há bem pouco tempo só era possível representar gráficos interativos na web recorrendo a certas tecnologias como o *Adobe Flash* (Flash, 2014), *Scalable Vector Graphics* (SVG) ou o *Microsoft Silverlight* (Silverlight, 2014). Contudo, com a introdução do HTML5 conseguiram-se essas e outras funcionalidades sem ser necessário o recurso a qualquer tipo de *plugins*, não só na componente gráfica como também na componente de áudio e vídeo. Atualmente as duas principais tecnologias no que diz respeito a gráficos na web são o HTML5 Canvas e o SVG. De acordo com Jakus et al. (2010), o SVG permite representar gráficos de alta resolução em qualquer nível de ampliação devido à sua natureza vetorial. Pelo contrário, no HTML5 Canvas os seus elementos mais básicos são os píxeis e portanto não existe uma diferenciação objetiva entre os píxeis existentes no gráfico. Isto implica ter de redesenhar o gráfico na sua totalidade sempre que se pretende efetuar alguma alteração. Por outras palavras, o conceito do Canvas está associado a gráficos bitmap e o SVG está associado a gráficos vetoriais (Hawkes, 2011).

No caso do Canvas, os gráficos bitmap são guardados como píxeis no formato 1-para-1, o que pode levar à pixelização ou distorção da imagem caso se faça um redimensionamento. Por sua vez, os gráficos vetoriais são guardados sob a forma de coordenadas que descrevem a forma que pretendemos desenhar, sendo que, caso houvesse um redimensionamento, as coordenadas seriam redimensionadas mas manteriam as relações entre elas. Posteriormente eram convertidas em píxeis para serem mostrados e independente do redimensionamento, o gráfico manteria a qualidade (Hawkes, 2011).

Em termos de tecnologia, a principal diferença na construção das imagens reside no facto do SVG ser baseado em XML (eXtensible Markup Language) de alto nível, onde

HTML5 Canvas	**SVG**
Baseado em JavaScript	Baseado em XML
Não existem elementos concretos, apenas um conjunto de píxeis	Os elementos possuem uma natureza vetorial
Sem API para animações. Dependemos de temporizadores e eventos para a atualização do Canvas	Suporta animações através de scripts, CSS e SMIL (*Synchronized Multimedia Integration Language*)
Melhor performance quando é grande o número de objetos desenhados	Melhor performance quando os objetos desenhados ocupam uma grande área
A API não suporta a acessibilidade	Permite o acesso direto através do DOM

Tabela 2.1: Tabela comparativa entre HTML5 Canvas e SVG.

se desenha através da criação de elementos XML com atributos para definir a imagem, enquanto que o Canvas oferece uma API de desenho de baixo nível que é acedida diretamente através de JavaScript (Cecco, 2011). A Tab. 2.1 sintetiza as principais caraterísticas destas duas tecnologias.

Existem, atualmente, vários artigos que visam comparações mais ou menos aprofundadas entre estas duas tecnologias. Sucan (2010) apresenta um estudo sobre algumas das caraterísticas mencionadas e indica vários casos de uso relativamente a estas duas tecnologias. Sugere-se ainda a consulta de (Rowell, 2011) onde são indicados os passos a seguir de forma a que um iniciante em Canvas entenda não só as bases, mas também particularidades mais complexas como a visualização de dados, o desenvolvimento de jogos e a modelação 3D. Kaipiainen & Paksula (2010) abordam o tema da integração entre Canvas e SVG.

Um ponto importante a focar, é que o Canvas é uma tecnologia baseada no píxel e portanto as formas que desenharmos vão somente ser entendidas como píxeis coloridos e não como objetos interativos. Com certas *frameworks* podemos ultrapassar esse problema visto que elas já têm definido uma estrutura de modelos de objetos. Hoje em dia, existem *frameworks* robustas e bibliotecas úteis para que qualquer programador web consiga tirar o maior partido possível do Canvas. Algumas delas são: EaselJS (EaselJS, 2014), Fabric.js (Fabric, 2014), KineticJS (KineticJS, 2014) e Paper.js (Paperjs,

2014).

Para o SVG também existem algumas bibliotecas que podem ser usadas para facilmente criar SVGs e animações nos mesmos, tais como Raphaël (Raphaël, 2014), Snap.js (SnapSVG, 2014) e Svg.js (SVGJS, 2014).

3

Análise de Jogo

Neste capítulo iremos abordar o projeto e implementação da Deteção de Modelos de Jogo, do Algoritmo de Deteção de Passes e Perdas de Bola no contexto do projeto FootData. Serão descritos os detalhes da cada uma das ferramentas individualmente, fazendo-se referências a diversas tecnologias usadas para a sua implementação. Serão ainda discutidos os resultados obtidos, comparando dentro do possível os produtos existentes no mercado com os propostos por nós. Neste sentido, convém desde já salientar que não foi possível testar em concreto outras aplicações no mercado devido à inexistência de versões de teste abertas ao público em geral, e uma vez que a sua aquisição também não era solução. Posto isto, procurou-se fazer comparações tendo como base, a documentação e os vídeos de demonstração disponibilizados pelas entidades

envolvidas na criação dessas aplicações.

3.1 Deteção de Modelos de Jogo - Esquemas Estáticos

O interface usado no FootData para desenhar os modelos de jogo, é chamado de Editor de Campo. O Editor de Campo é uma ferramenta web para programação visual, criada usando HTML 5, CSS e JavaScript que permite aos treinadores desenhar as suas táticas para serem posteriormente processadas no centro de dados do FootData. As funcionalidades implementadas no Editor de Campo, incluem ferramentas para inserir jogadores no campo, e configura-los (e.g., alterar cor, número ou posição ocupada no campo), inserir a bola e atribui-la a um jogador, definir áreas e atribuir distâncias entre os vários elementos em campo (e.g., jogadores, bola, áreas) sendo também possível marcar distâncias entre o ponto intermédio de dois jogadores e outro elemento no campo. As áreas são do tipo polígono ou elipse e podem ser usadas para saber se um jogador ou bola está dentro de uma determinada zona do campo. Também é permitido definir áreas dentro de áreas, para situações, como por exemplo, em que os jogadores tenham de estar contidos dentro de uma área, em que essa mesma área se possa deslocar dentro de outra área maior. A Fig. 3.1 mostra um exemplo de uma situação desenhada no Editor de Campo, em que foram inseridos quatro jogadores (círculos vermelhos), uma bola, três distâncias e duas áreas. Este esquema tem como objetivo verificar se sempre que a bola está dentro da área verde, os jogadores estão dentro da área a vermelho com uma distância entre si de 12 metros. Apesar de não ser visível na imagem, a distância entre os jogadores admite um erro de 3 metros, valor este configurável na interface do Editor de Campo. Quando o utilizador acaba o esquema desejado, tipicamente o treinador ou os membros da equipa técnica, este tem que ser passado para o *datacenter* do FootData para que este seja inserido na base de dados, de modo a permitir ser posteriormente carregado no interface e reeditado. Além disso, uma vez no *datacenter*, o esquema é transformado em código para poste-

rior processamento juntamente com a informação adquirida pelo *tracking*.

Nas secções seguintes iremos ver como é feita a serialização do esquema de modo a ser introduzido na base de dados, assim como a sua conversão para Python de modo a ser mais tarde usado em conjunto com o *tracking* na deteção do modelo de jogo.

3.1.1 Serialização dos Esquemas Desenhados

Na comunicação entre o Editor de Campo (aplicação web) e o *datacenter* do FootData, foi pensado um processo de serialização tendo em consideração que o Editor de Campo está implementado em JavaScript e as bibliotecas do FootData usadas para verificação dos esquemas estão implementadas em Python. Foi então decidido que a solução seria converter o esquema desenhado para um documento JSON (JavaScript Object Notation). Ao usar o JSON permiti-nos tirar partido de: (1) a facilidade que o JavaScript tem no tratamento dos documentos JSON, (2) o facto de esses documentos serem muito parecido com os dicionários de Python, evitando assim processos de

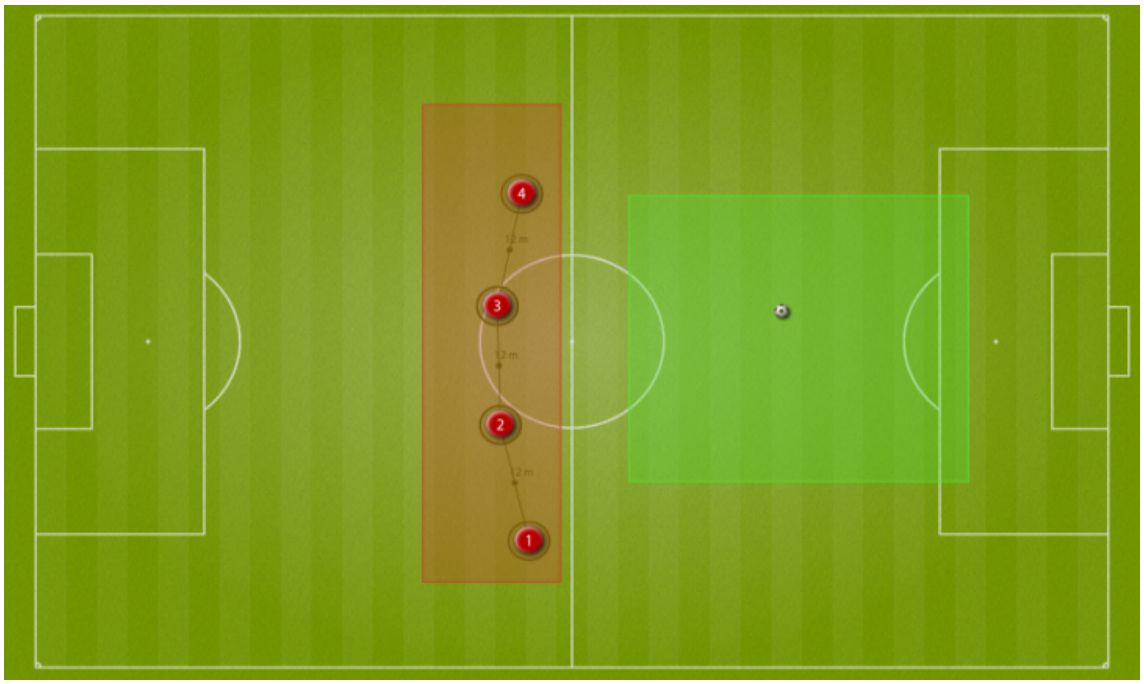

Figura 3.1: Exemplo de desenho efetuado no Editor de Campo, em que os quatro jogadores devem estar na zona vermelha quando a bola estiver na zona verde, respeitando as distâncias entre si.

conversão complicados, e (3) o facto de a base de dados usada para guardar os documentos ser o MongoDB (base de dados orientada a documentos com formato JSON).

A Fig. 3.2 mostra o documento JSON gerado pelo Editor de Campo a partir do desenho previamente efetuado no mesmo editor (Fig. 3.1). O documento JSON apresentado tem uma estrutura predefinida, onde está incluído o nome do esquema, o tipo de processo de jogo, as dimensões do campo, a escala do campo, os jogadores, a bola, as áreas, as linhas, as distâncias e por fim uma lista de condições. Em mais detalhe, o atributo *name* refere-se ao nome que o utilizador/treinador atribui ao esquema desenhado, e o *type* é usado para saber em que momento do jogo se insere o esquema (Processo Defensivo, Processo Ofensivo, Transição Ofensiva ou Transição Defensiva). Os atributos *field_dimension* e *field_scale* são referentes à dimensão e escala do campo em que o esquema é desenhado. Estes atributos são necessários pois o campo usado no Editor de Campo tem dimensões normalizadas e dependendo da resolução do ecrã esta imagem pode ser alterada para melhor satisfazer as necessidades do utilizador. Com estes atributos e sabendo as dimensões do campo onde ocorre o jogo, é possível ajustar o esquema desenhado para um determinado estádio. Na Fig. 3.3 estão ilustradas as dimensões do campo em metros e correspondentes dimensões em píxeis. Estas dimensões em píxeis são depois usadas para ajustar os elementos utilizados no desenho do modelo de jogo ao campo em que se está a realizar o jogo, pois este pode ter dimensões diferentes das do campo modelo.

O atributo *players* é composto por um *array*, em que cada elemento corresponde a um jogador inserido no Editor de Campo. Cada jogador é definido através de um subdocumento JSON com a seguinte estrutura: `{"id": 1, "number": 1,"team": "A", "points": [514, 559], "radius": 3}`, a qual contém um `id` único para cada elemento inserido no Editor de Campo, sendo assim possível aceder a qualquer objeto unicamente pelo seu `id`. Contém também informação sobre o número e equipa do jogador, `number` e `team` respetivamente, sendo o número do jogador um inteiro e a equipa o caráter 'A' ou 'B'. Finalmente assim como as coordenadas (`points`), em pí-

```
{
"name": "schema 1",
"description": "Ball Pressure",
"type": "Defensive Process",
"field_dimension": [105, 68],
"field_scale": [0.1, 0.1],
"players": [
  {"id":1,"number":1,"team":"A","points":[514, 559],"radius":3},
  {"id":2,"number":2,"team":"A","points":[486, 438],"radius":3},
  {"id":3,"number":3,"team":"A","points":[483, 314],"radius":3},
  {"id":4,"number":4,"team":"A","points":[507, 197],"radius":3}
],
"ball": {"id": 5,"points": [762, 320], "owner": "None"},
"zones": [
  {"id":7,"name":"Zone 7","points":[[945, 498],[945, 198],[611, 198],[611,
     498]]},
  {"id":6,"name":"Zone 6","points":[[546, 603],[546, 103],[409, 103],[409,
     603]]}
 ],
"lines": [
  {"id": 10,"objects": [3, 4]},{"id": 9,"objects": [2, 3]},
  {"id": 8,"objects": [1, 2]}
  ],
"distances": [
  {"objects": [3, 4], "distance": 12, "margin": 3},
  {"objects": [2, 3], "distance": 12, "margin": 3},
  {"objects": [1, 2], "distance": 12, "margin": 3}
  ],
"conditions": [
  [
    {"verify_location": [1, "IN", 6]},
    {"verify_location": [2, "IN", 6]},
    {"verify_location": [3, "IN", 6]},
    {"verify_location": [4, "IN", 6]},
    {"verify_location": [5, "IN", 7]},
    {"verify_distances": true}
  ],
  [
    {"alert": "True", msg": "Pressing the ball"}
  ],
  [
    {"alert": "False","msg": ""}
  ]
]
}
```

Figura 3.2: Exemplo de um documento JSON gerado pela ferramenta Editor de Campo.

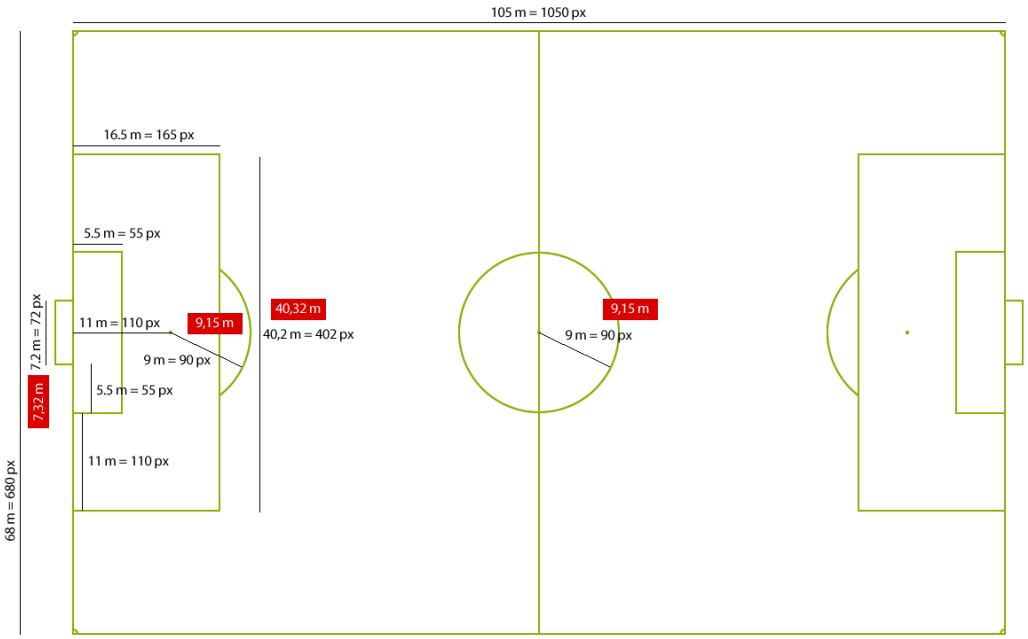

Figura 3.3: Dimensões do campo modelo usadas no Editor de Campo.

xeis, do elemento no campo e um valor em metros sobre a qual a posição do jogador pode variar, referido no documento como `radius`.

Quando é inserida uma distância entre dois elementos, é adicionado ao documento JSON um atributo *distances* (distâncias) juntamente com o atributo *lines* (linhas), sendo ambos compostos por um *array* em que cada elemento contém um documento JSON. Para o caso das linhas o documento JSON é composto por um `id` único, como já referido anteriormente, e um atributo `objects` (objectos) o qual é composto por um *array* contendo o `id` onde a linha começa e o `id` onde a linha acaba. Por exemplo a seguinte estrutura `{"id": 8, "objects": [1, 2]}` indica que existe uma linha entre o elemento `id` 1 e o elemento `id` 2. O documento JSON que constitui as distâncias contém igualmente o atributo `objects`, juntamente com o atributo `distances` (distâncias) e `margin` (margem), sendo a distância referente à distância em que os objetos se devem encontrar em metros e a margem o valor em metros no qual essa distância pode va-

riar. A título de exemplo, o documento JSON para as distâncias tem a e estrutura `{"objects": [3, 4], "distance": 12, "margin": 3}`, a qual indica que o elemento com `id` 3 e `id` 4 têm que se encontrar a uma distância de 12 metros, podendo essa distância variar entre 9 e 15 metros devido a margem de 3 metros. A distinção entre *lines* e *distances*, apesar de serem de estrutura muito semelhante, é necessária pois é possível fazer linhas sem que estas estejam diretamente ligadas às distâncias.

O atributo *zones* é constituído por um *array*, em que cada elemento corresponde a uma área sendo essa área, descrita através do uso de um subdocumento JSON. O subdocumento contém um `id`, um `name` (nome), no qual pode ser atribuído um nome à zona e um atributo `points` (pontos) que é composto por um *array* em que cada elemento do *array* é composto por um outro *array* contendo um par de coordenadas `[x, y]` que representam os vértices de um polígono. Essas coordenadas correspondem à posição dos vértices no campo modelo do Editor de Campo. Por exemplo, o documento JSON que define as zonas pode ter a seguinte estrutura: `{"id": 6, "name": "Zone 6", "points": [[546, 603], [546, 103], [409, 103], [409, 603]]}`. Uma vez que as áreas podem ser polígonos, elipses ou de desenho livre, seria normal ter uma representação para cada uma, mas de modo a evitar ter que criar essas representações, todos são tratados como polígonos, sendo que para a elipse são calculados vários pontos da elipse de modo a que a sua representação como polígono seja o mais aproximado possível à forma original.

Tendo toda esta informação reunida, só falta adicionar um último atributo ao documento JSON de modo a que o programa saiba o que detetar, atributo esse chamado *conditions* (condições). Como o próprio nome indica, neste campo é possível criar condições, mais especificamente condições do tipo *if-then-else*. De modo a ser possível fazer estas condições foi definida uma estrutura com um *array* contendo 3 sub-arrays: o primeiro sub-array corresponde à condição *if*, o segundo corresponde ao que fazer caso a condição seja verdadeira (*then*) e o terceiro corresponde ao que fazer caso a condição seja falsa (*else*). A Fig.3.4 contém um exemplo ilustrativo da estrutura do atri-

```
"conditions": [
  [
    {"verify_location": [1, "IN", 6]},
  ],
  [
    {"alert": "True", msg": "Pressing the ball"}
  ],
  [
    {"alert": "False","msg": ""}
  ]
]
```

Figura 3.4: Exemplo da estrutura do atributo *conditions*.

buto *conditions*. Neste caso, pretende-se que o sistema gere uma mensagem de alerta (*"Pressing the ball"*) sempre que o objeto com `id` 1 esteja dentro do objeto com `id` 2.

É também possível criar combinações de maior complexidade. Por exemplo: (1) é possível encadear condições *if-then-else*, ao adicionar dentro da condição *then* (segunda entrada do *array*) ou *else* (terceira entrada do *array*) outra condição do tipo *if-then-else*, i.e., introduzir mais um *array* de três elementos que respeite a estrutura apresentada anteriormente. Além disso, também é possível introduzir (2) conjunções e (3) disjunções. As conjunções são definidas ao introduzir uma única condição no documento JSON, por exemplo: `[{"verify_location": [1, "IN", 6]}, {"verify_location": [2, "IN", 6]}]`, em que dentro do sub-array *if* são introduzidos dois documentos JSON separados por virgula, resultando numa proposição verdadeira caso o resultado de cada documento JSON seja verdadeiro. As disjunções são definidas ao inserir várias condições no mesmo documento JSON, por exemplo: `[{"verify_location": [1, "IN", 6], "verify_distances": true]}]`, na qual resulta numa proposição verdadeira caso uma das condições seja verdadeira. Como é de esperar, também é possível combinar as conjunções e disjunções, por exemplo: `[{condition1, condition2}, {condition3, condition4}]`, o qual resultaria na seguinte expressão logica: `(condition1` $\vee$ `condition2)` $\wedge$ `(condition3` $\vee$ `condition4)`. Para além da estrutura usada para definir a condição *if-then-else*, também as condições têm a sua sintaxe própria. Por exemplo a condição *verify_location*, cuja função é verificar se um elemento

(bola, jogador, etc) se encontra dentro de uma determinada área, tem que ser passado como valor um *array* de três elementos em que (1) recebe o `id` do elemento a verificar, (2) a condição "IN" ou "OUT" para detetar se está dentro ou fora e (3) o `id` da zona. A condição *verify_distances*, usada para verificar as distâncias entre elementos, por sua vez só precisa que seja passado um valor booleano de *True* ou *False*.

3.1.2 Programação para Análise

Tendo o documento JSON totalmente construído, é agora necessário passa-lo para a linguagem de programação Python. Inicialmente os elementos *players*, *distances*, *lines* e *zones* são instanciados a partir de métodos previamente implementados na biblioteca Python implementada para o FootData. Os objetos criados são serializados através do uso de um módulo pertencente à biblioteca padrão Python chamando *pickle*, tornando assim possível o armazenamento de objetos, para que quando é necessário o seu uso, estes sejam apenas carregados evitando assim instanciá-los todas as vezes que o desenho tenha que ser verificado. Este processo é também justificado pelo facto de os dados do Editor de Campo serem gravados na base de dados MongoDB e tendo os ficheiros com os objetos previamente instanciados irá acelerar o processo de inicialização. O passo seguinte é converter o atributo *conditions* para código Python, para que este possa ser depois usado para verificar o desenho. A conversão é feita ao aceder ao *array conditions*, para depois criar as condições *if-then-else* de acordo com o que foi inserido, tendo em atenção para o caso de haver condições encadeadas e/ou conjunções e disjunções com a estrutura anteriormente referida. Os nomes usados no *array conditions* são correspondidos com os respetivos métodos previamente implementados, de igual modo são também correspondidos os atributos das condições com os atributos previamente instanciados. Como exemplo temos que para a condição `"verify_location": [5, "IN", 7]` no documento JSON (Fig. 3.2), corresponde o método `verify_location( frame, ['ball', 'IN', zones7])`. O argumento *frame*, corresponde a um *frame* proveniente do sistema de *tracking*, e o argumento `['ball',`

```
from lib import gpm
def run():
  result = False
  if gpm.verify_location(frame,['ball','IN',zones7]):
    if gpm.verify_location(frame, [players2,'IN',zones6]):
      if gpm.verify_location(frame, [players3,'IN',zones6]):
        if gpm.verify_location(frame, [players4,'IN',zones6]):
          if gpm.verify_location(frame, [players1,'IN',zones6]):
            if gpm.verify_distances(frame, distances) :
              gpm.sendmsg('Pressing the ball')
              gpm.sendalert('True')
              result = True
  if not result:
    gpm.sendmsg('')
    gpm.sendalert('False')

  return result, gpm
```

Figura 3.5: Código Python gerado a partir do documento JSON.

`'IN', zones7]` provem da correspondência dos `ids` com os objetos previamente instanciados, podendo-se verificar na Fig. 3.2 que o `id` 5 corresponde à bola (*ball*) e o `id` 7 à zona 7. À medida que as condições são processadas, o ficheiro Python com as condições *if-then-else* vai sendo gerado através do uso de uma biblioteca de geração de código Python criada para o propósito.

A Fig.3.5 mostra o código Python gerado pelo processo descrito anteriormente, correspondente ao documento JSON da Fig.3.2. Com o ficheiro Python gerado, falta apenas a informação acerca dos jogadores e da bola para testar o desenho em causa. Como já anteriormente mencionado, esta informação é adquirida através de um sistema de *tracking* (Rodrigues et al., 2014) que devolve para cada *frame*, a posição dos jogadores e da bola em metros tendo como referência o canto superior esquerdo do campo. Para tal o sistema de *tracking* deteta os jogadores/bola dentro de cada *frame*, aplicando posteriormente uma homografia (Sebe & Lew, 2003) de modo a colocá-los na posição correspondente dentro do campo modelo. As *frames* devolvidas pelo sistema de *tracking* têm a seguinte estrutura: `{"frame_id": t, "teamA": {"player_1": (x1, y1), "player_2": (x2, y2), ...}, "teamB": {...}, "ball": (x,y)}`.

Voltando ao código da Fig. 3.5 o módulo `gpm` aí importado contém um conjunto de

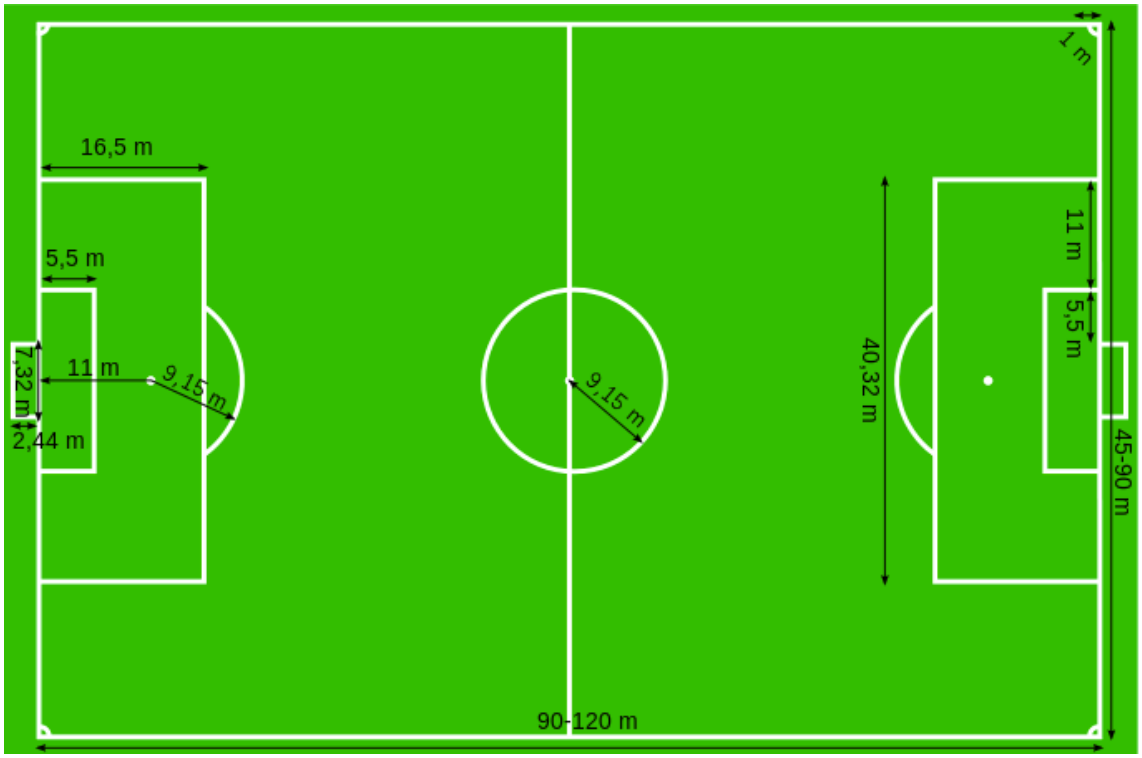

Figura 3.6: Campo de futebol com as dimensões padrão.

métodos pré-implementados, que são usados para a deteção das situações desenhadas e também para guardar a informação recolhida por cada método que é executado. Tendo os desenhos convertidos em ficheiros com código Python, foi necessário encontrar uma maneira de os executar sem que fosse necessário importar à mão cada ficheiro gerado, o que seria extremamente dispendioso, visto serem esperados centenas de desenhos. Para esse fim foi criado um módulo Python que permite importar cada ficheiro usando apenas o seu nome de ficheiro e a *package* em que está inserido, facilitando assim a importação de um grande número de desenhos. Este módulo também é usado para correr os desenhos, bastando apenas passar o *frame* como argumento. A Fig. 3.7 mostra como os desenhos são importados e executados através do módulo *ModelRun* criado para o efeito.

Internamente este módulo irá importar o ficheiro de Python correspondente ao desenho e irá também importar os objetos anteriormente instanciados e serializados. O ficheiro com os objetos é desserializado e os seus objetos carregados no módulo atra-

```
1
2   desenho = ModelRun("pkg.foo.bar.desenho1")
3   desenho.run(frame)
```

Figura 3.7: Exemplo de como os ficheiros com o código Python são importados e corridos para verificação.

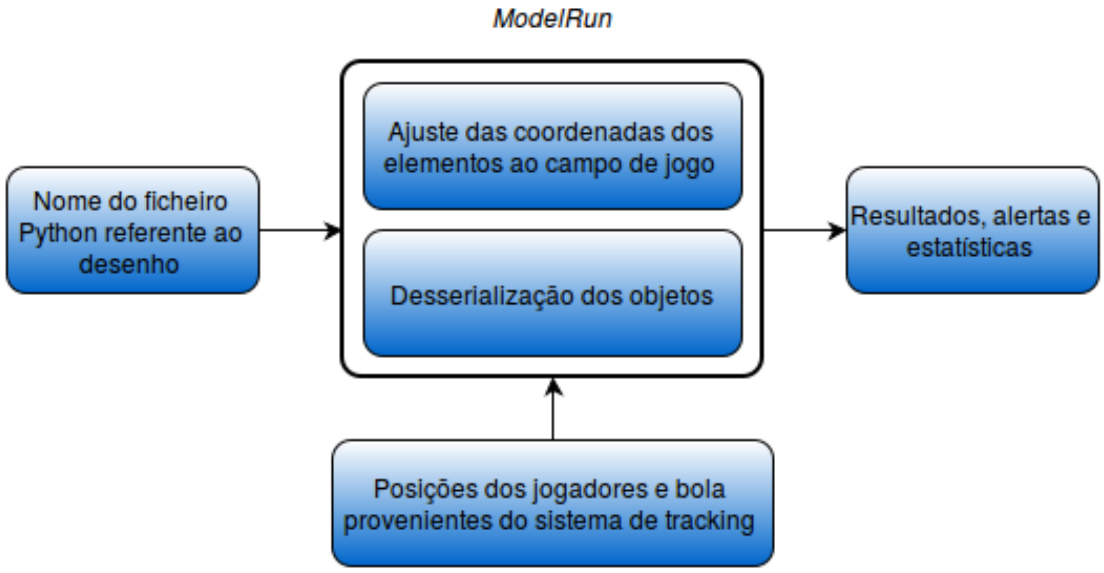

Figura 3.8: Arquitetura usada para correr os desenhos.

vés do método *setattr* pertencente à linguagem Python. Como referido anteriormente, é também possível que as dimensões do campo variem, visto ser possível terem uma largura de 90 a 120 metros e uma altura de 45 a 90 metros (Fig. 3.6). Caso as dimensões do campo em que está a decorrer o jogo sejam diferentes das dimensões usadas no Editor de Campo, tem que se proceder a um ajuste dos elementos do desenho para que estes fiquem ajustados às novas dimensões do campo. Este ajuste é feito através do uso de uma regra de três simples, visto termos as dimensões do novo campo, as coordenadas dos elementos no desenho e as dimensões usadas no campo de Editor de Campo, bastando apenas calcular as coordenadas dos elementos para o novo campo.

Com as coordenadas dos elementos para o novo campo calculadas, podemos passar para a análise do desenho, que é feita para cada *frame* ou para um conjunto de n *frames*. No final da análise, um dicionário Python é devolvido, contendo os resultados

para cada *frame*. A Fig. 3.8 resume a arquitetura acima explicada.

3.1.3 Resultado da Deteção

Como já foi referido, para cada *frame* em que é verificado o desenho do modelo de jogo, é gerada informação correspondente à verificação, ou não, das condições desenhadas. Essa informação precisa de ser estruturada e armazenada para posteriormente serem criadas estatísticas relativas ao desenho verificado na partida/treino . A estrutura usada para as armazenar, foi uma vez mais o JSON. A Fig. 3.9 exemplifica o resultado obtido através da verificação do desenho do modelo de jogo apresentado na Fig. 3.1 para a *frame* 947 de uma partida de futebol.

```
{'t': 947,
 'verify_location':[
 {'True':None,'first_arg':'ball','second_arg':Zones7,'condition':'IN'},
 {'True':None,'first_arg':'Number:2,Team:'A,'second_arg': 'Zones6', '
    condition': 'IN'},
 {'True':None,'first_arg':'Number:3,Team:A','second_arg': 'Zones6', '
    condition': 'IN'},
 {'True':None,'first_arg':'Number:4,Team:A','second_arg': 'Zones6', '
    condition': 'IN'},
 {'True':None,'first_arg':'Number:1,Team:A','second_arg': 'Zones6', '
    condition': 'IN'}
 ],
 'verify_distances':[
    {'players':('Number:3,Team:A','Number:4, Team:A'), 'True':None, '
       distance_def':12,'distance':9.83},
    {'players':('Number:2,Team:A','Number:3, Team:A'), 'True':None,'
       distance_def':12,'distance':11.44},
    {'players':('Number:1,Team:A','Number:2, Team:A'), 'True': None,'
       distance_def': 12, 'distance': 14.14}
  ]
 }
```

Figura 3.9: Resultados obtidos através da verificação do desenho da Fig. 3.1 para uma *frame* de uma partida de futebol.

Na Fig. 3.9, a chave *verify_location* contém informação referente a todas as vezes em que o método que verifica se um determinado objeto está dentro de um espaço foi executado, armazenando essa informação na forma de subdocumento JSON. Em cada um desses subdocumentos existem as chaves `first_arg` e `second_arg` que correspon-

dem ao elemento que foi verificado e qual a zona onda ele deve estar, respetivamente. A chave `condition` refere-se à condição que foi verificada, sendo que caso seja 'IN' o `first_arg` teria que se encontrar dentro do `second_arg` e caso seja 'OUT' teria que se encontrar fora. Caso se tenha verificado a condição pretendida é adicionada uma chave com o nome *True* ou uma chave com valor *False* caso contrário (contendo como valor *None*). Este último campo é adicionado desta forma pelo facto de em Python se mais eficiente verificar se uma determinada chave existe dentro de um dicionário, do que ter por exemplo uma chave chamada *Result* em que se iria verificar se o seu conteúdo era *True* ou *False*. Como é visível no exemplo anteriormente referido, no primeiro elemento da lista temos que foi verificado que a bola (*ball*) foi verificada dentro ('IN') da zona 7 (Zones7), (sendo o resultado verdadeiro devido à existência de uma chave com o nome *True*).

Relativamente à chave *verify_distances*, nesta é armazenada informação referente a todas as vezes em que este método foi executado para verificar as distâncias entre os objetos definidos no desenho. Esta contém uma lista, com um subdocumento JSON para cada distância verificada, o qual tem a seguinte estrutura: a chave (1) `players` indica quais os dois objetos para os quais estão a ser verificadas as distâncias, podendo ser jogadores, zonas, bola e distâncias intermédias. A chave (2) `distance_def` diz-nos a distância que foi definida no desenho e a chave (3) `distance` a distância a que os dois objetos se encontravam naquele *frame*. A estrutura tem ainda a chave *True* ou *False*, caso os dois objetos em causa estejam a respeitar a distância ou não (a explicação de usar estas chaves desta forma tem a ver com a eficiência da verificação dos testes em Python, como já foi referido anteriormente).

Todo este processo é repetido para cada uma das *frames* sendo os resultados armazenados na base de dados. No final são criadas estatísticas detalhadas sobre o que e quando falhou para cada um dos modelos de jogo desenhados. Essa informação é depois passada ao treinador de modo a que este possa saber o que está a correr bem ou mal num determinado jogo, de acordo com os seus desenhos do modelo de jogo.

Na Secção 4.1 serão apresentados os resultados de alguns testes efetuados executando este método.

3.2 Deteção de Modelos de Jogo - Esquemas Dinâmicos

Entende-se por esquemas dinâmicos, os esquemas que contêm situações em que há sequência por parte dos objetos inseridos no Editor de Campo, i.e., esquemas com situações que devem acontecer numa determinada ordem, sendo estes esquemas por norma mais complexos do que os descritos na Secção 3.1.

3.2.1 Serialização dos Esquemas Desenhados

De modo geral, os Esquemas Dinâmicos (ED) são tratados como um conjunto de N Esquemas Estáticos (EE), permitindo desta forma a reutilização da maior parte das ferramentas de criação e deteção usadas para os EE. Neste modo, o documento JSON criado tem uma estrutura semelhante a dos EE, em que apenas foi adicionado um novo campo chamado *steps*, onde são introduzidos os conjuntos de EE, que compõem a situação pretendida. Na Fig. 3.10 temos um exemplo de um ED em que o jogador 8 quando está na zona a laranja tem que passar a bola para o jogador 1 que se deverá encontrar dentro da zona a verde, que por sua vez tem que passar ao jogador 6 que deve estar na zona a azul.

De modo a passar a situação desenhada na Fig. 3.10 para o documento JSON, o desenho é dividido em três momentos: (1) o momento em que o jogador 8 e a bola se encontram dentro da zona a laranja, (2) o momento em que o jogador 1 e a bola se encontram dentro da zona a verde e (3) o momento em que o jogador 6 e a bola se encontram dentro da zona azul. Esta divisão em momentos é feita automaticamente no interface do Editor de Campo. Como referido, foi adicionado o campo *steps* ao documento JSON que descreve o esquema desenhado. O campo *steps* irá conter os 3 momentos que foram referenciados, sendo estes enumerados para que se saiba a

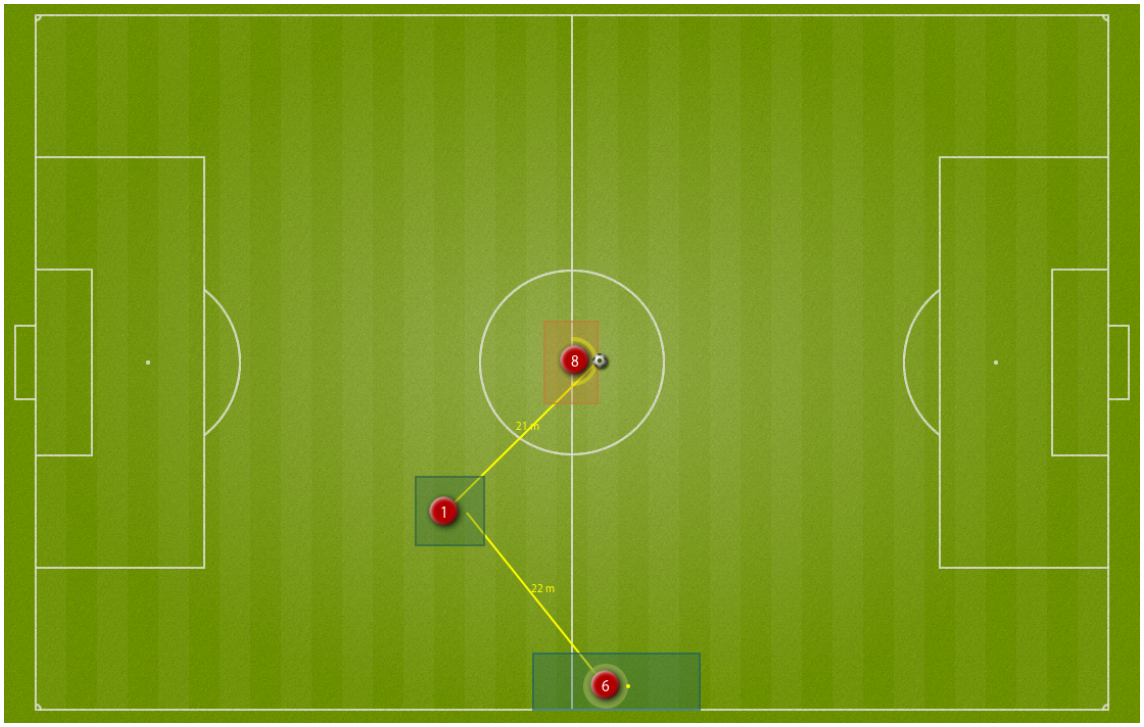

Figura 3.10: Exemplo de um desenho do Modelo de Jogo, utilizando o Esquema Dinâmico, em que pretende-se detetar se a bola passa por os jogadores 8, 1 e 6 nas zonas definidas para os mesmos.

ordem em que têm que ser detetados. O documento JSON gerado para o exemplo da Fig. 3.10, pode ser visto na Fig. 3.11.

Cada momento do campo *steps*, irá conter todos campos que são necessários para que o momento seja detetado como EE, exceto os campos *name*, *type*, *field_dimension* e *field_scale*, dado que esses valores são comuns em todos os momentos. Como é visível no exemplo da Fig. 3.11, cada momento apenas tem o jogador e a zona que é necessária para a deteção do mesmo, contendo também o campo *conditions* onde é definido o que deve ser detetado em cada momento e qual o resultado a dar caso este aconteça e caso não aconteça.

3.2.2 Programação para Análise

Tendo o documento JSON gerado, é necessário passa-lo para código Python. No módulo criado para o efeito, primeiro é verificado se existe no documento o campo *steps*.

```
{ "name" : "esq_din1",
"type" : "Processo Ofensivo",
"field_dimension" : [ 105, 68 ],
"field_scale" : [ 0.1, 0.1 ],
"steps" : {
  "1" : {"players" : [
      { "id" : 2, "number" : 8, "team" : "A", "points" : [ 559, 350 ], "
          radius" : 3 }],
    "zones" : [
      { "id" : 1, "name" : "Zona 1", "points" : [ [ 582, 392 ], [ 582, 310
          ], [ 528, 310 ], [ 528, 392 ] ] } ],
    "ball" : { "id" : 7, "points" : [ 584, 350 ], "owner" : 2 },
    "conditions" : [
      [   { "verify_location" : [ 2, "IN", 1 ] },
          { "verify_location" : [ 7, "IN", 1 ] }],
      [   { "msg" : "Primeira Fase Aconteceu" }],
      [   { "msg" : "" } ]
    ]
  },
  "2" : {"players" : [
      { "id" : 3, "number" : 1, "team" : "A", "points" : [ 431, 498 ], "
          radius" : 3 }],
    "zones" : [
      { "id" : 4, "name" : "Zona 4", "points" : [ [ 471, 531 ], [ 471, 462
          ], [ 402, 462 ], [ 402, 531 ] ] } ],
    "ball" : { "id" : 7, "points" : [ 584, 350 ], "owner" : 3 },
    "conditions" : [
      [   { "verify_location" : [ 3, "IN", 4 ] },
          { "verify_location" : [ 7, "IN", 4 ] }],
      [   { "msg" : "Segunda Fase Aconteceu" }],
      [   { "msg" : "Segunda Fase Nao Aconteceu" } ]
    ]
  },
  "3" : {"players" : [
      { "id" : 6, "number" : 6, "team" : "A", "points" : [ 589, 668 ], "
          radius" : 3 } ],
    "zones" : [
      { "id" : 5, "name" : "Zona 5", "points" : [ [ 682, 692 ], [ 682, 635
          ], [ 517, 635 ], [ 517, 692 ] ] } ],
    "ball" : { "id" : 7, "points" : [ 584, 350 ], "owner" : 6 },
    "conditions" : [
      [   { "verify_location" : [ 6, "IN", 5 ] },
          { "verify_location" : [ 7, "IN", 5 ] }],
      [   { "msg" : "Desenho Aconteceu" }],
      [   { "msg" : "Terceira Fase Nao Aconteceu" } ]
    ]
  }
}
}
```

Figura 3.11: Exemplo de um documento JSON gerado pela ferramenta Editor de Campo para um ED.

Caso exista, então este documento é tratado como um ED, caso contrário trata-se de um EE. Sendo um ED, cada momento do campo *steps* é processado como se fosse um EE, (ver Secção 3.1.2), em que todos os atributos exceto as condições são instanciados, serializados e gravados para um ficheiro, e o atributo *conditions* é usado para a criação do ficheiro Python contendo as condições *if-then-else*. No final iremos ter um ficheiro Python e um ficheiro dos atributos instanciados, para cada momento existente no campo *step*. De modo a que não haja confusão com o nome dos ficheiros e para que seja mais fácil distinguir os vários momentos, foi adicionado no final da cada nome do ficheiro um sufixo *_step[numero]*, em que o *numero* corresponde ao momento a que aquele ficheiro corresponde. Para o exemplo da Fig. 3.11, como contém três momentos, irão ser criados três ficheiros Python com os nomes *esq_din1_step1.py*, *esq_din1_step2.py* e *esq_din1_step3.py* juntamente com os respetivos ficheiros de atributos.

Para detetar os desenhos do modelo de jogo é usado o mesmo módulo referido nos EE, tendo apenas que ser inserido mais um argumento *dynamic = True*, para informar ao módulo que se trata de um ED. Na Fig. 3.12 está exemplificada a alteração referida.

```
1
2   desenho = ModelRun("pkg.foo.bar.desenho1", dynamic = True)
3   desenho.run(frame)
```

Figura 3.12: Exemplo de como os ficheiros com o código Python são importados e corridos para verificação nos Esquemas Dinâmicos.

O nome do desenho a detetar, mesmo os ficheiros sendo gravados com o sufixo *_step[numero]*, mantêm-se o mesmo que foi dado ao desenho, ficando o módulo encarregue de procurar os ficheiros Python corretos para poder proceder à deteção. Para carregar os ficheiros, o módulo vai procurar os ficheiros cujo nome corresponda ao nome do módulo inserido mais o sufixo *_step[numero]*. Cada ficheiro encontrado é carregado no módulo como um *step*, em que é feito o mesmo processo que nos EE, desserialização dos atributos instanciados e caso necessário, adaptação às medidas do campo de futebol. Depois de carregados todos os *steps*, é necessário detetar se as situ-

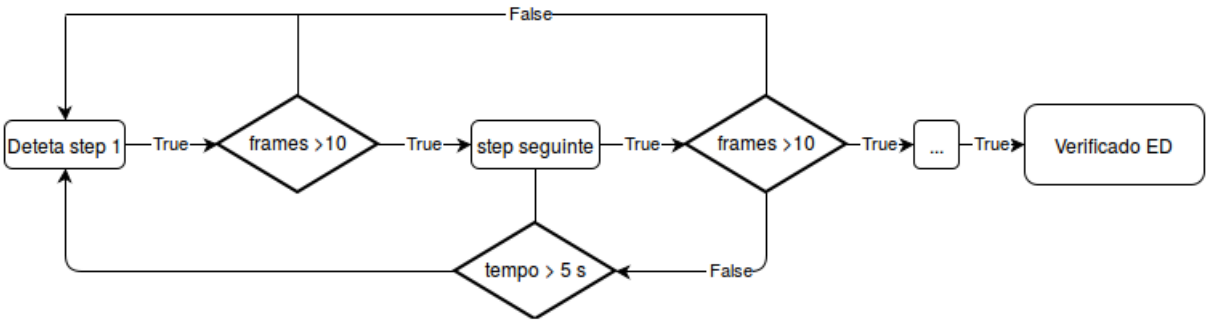

Figura 3.13: Arquitetura usada para a deteção dos ED.

ações ocorrem no campo ou não. Para esse efeito, inicialmente é detetado o primeiro *step*. Caso ele aconteça, inicia-se um contador para contar durante quantos *frames* a situação acontece. Quando a situação deixa de acontecer o contador deixa de ser incrementado e é verificado se o número de *frames* em que as condições daquele *step* aconteceram é suficiente para ser considerado como ocorrido (empiricamente, para os testes realizados considera-se o valor de 10 *frames*). Uma vez que o primeiro *step* é detetado, passamos à deteção do próximo *step* onde existe um intervalo máximo definido de 5 segundos (valor também definido empiricamente para os testes efetuados) em que o *step* tem de ser detetado. Caso este não seja detetado nesse intervalo, voltamos à deteção do primeiro *step*. Os parâmetros de deteção tal como o número mínimo de *frames* para ser considerado como detetado e o intervalo máximo em que o *step* seguinte tem que acontecer são configuráveis de modo a que se adequem melhor à situação desenhada. No caso de o *step* a ser detetado não for o primeiro, ao mesmo tempo que é verificado se o *step* N acontece é também verificado se o *step* N-1 acontece, isto para os casos em que se valida um momento, passa-se a detetar o próximo momento mas passado poucos *frames* o momento anterior volta a acontecer. Este processo evita pequenos erros que possam acontecer durante a deteção dos *steps*. A Fig. 3.13 resume a arquitetura usada para a deteção dos ED.

3.2.3 Resultado da Deteção

O resultado da deteção é semelhante ao da secção 3.1.3 (para os Esquemas Estáticos). A diferença reside no facto de que iremos ter para cada *step* um documento JSON (como o da Fig. 3.9), contendo toda a informação sobre o resultado dos métodos usados, podendo-se assim saber os motivos que levaram os *steps* a não ocorrem, i.e., o que ou quem falhou ao que foi definido pelo treinador no esquema.

Na Secção 4.2 serão apresentados os resultados de alguns testes efetuados executando este método.

3.3 Deteção de Trocas de Bola

A deteção de trocas de bola tem como objetivo a criação de uma lista de quem tocou a bola durante o jogo, independentemente da equipa. Os dados da lista serão usado para a deteção de passes e perdas de bola.

3.3.1 Projeto e Desenvolvimento da Deteção de Trocas de Bola

No processo de pesquisa para a criação do algoritmo para deteção de trocas de bola, foi encontrado em (Beetz et al., 2009) e em (Yu et al., 2003) exemplos de deteção que se baseiam no cálculo de um conjunto de pontos pivot através das variações de velocidade e direção que são depois cruzados com um conjunto de pontos onde os jogadores tocaram na bola, dando assim os locais onde possivelmente houveram passes ou outras ações. O método utilizado para a deteção de trocas de bola aqui apresentado foi feito com base nestas informações e partindo do principio básico que cada toque na bola irá alterar a sua trajetória e velocidade. O processo, de um modo geral, é efetuado na seguinte forma:

1. É verificado quando a bola entra e sai do raio de ação do jogador (definido empiricamente com um valor de 1,8 metros);

2. É calculada a velocidade e ângulo de entrada e de saída da bola no raio de ação do jogador;

3. Através dos valores calculados para a velocidade e ângulo é decidido se o jogador teve efetivamente a bola ou não.

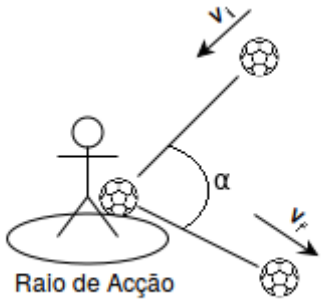

Figura 3.14: Exemplo ilustrativo do processo de deteção de trocas de bola.

A Fig. 3.14 ilustra o processo enumerado anteriormente, em que v_i e v_f correspondem à velocidade inicial e final da bola respetivamente e α à variação do ângulo de entrada e saída da bola do raio de ação.

Visto a deteção de trocas de bola ter que permitir processar dados em tempo real, a arquitetura do algoritmo foi desenvolvida a pensar nesse propósito. Para serem detetados os passes, o módulo tem que receber *frames* com as localizações dos jogadores e da bola provenientes do sistema automático de *tracking* (Rodrigues et al., 2014). Supõem-se que o algoritmo de *tracking* é capaz de processa as usuais 25 *frames* por segundo, sendo essas *frames* inseridas num *buffer* de dimensão fixa. Quando é atingido o limite do tamanho do buffer, essas *frames* são passadas para o algoritmo de deteção de passes. No algoritmo, inicialmente é calculado o jogador mais perto da bola para todos os *frames* do *buffer*, no qual é devolvido uma lista com a distância de cada jogador à bola. Essa lista irá ser percorrida até que haja um *frame* em que a distância de um jogador à bola seja inferior ao raio de ação, i.e., até que a bola entre no raio de ação do jogador. Caso a bola entre no raio de ação, esse *frame* é guardado como o instante em que o jogador recebeu a bola. Passamos agora à fase em que temos de detetar se a

distância do mesmo jogador à bola é superior ao raio de ação, i.e., quando a bola sai do raio de ação do jogador. Existe a possibilidade de chegar ao último *frame* do *buffer* e só se encontrar o momento em que a bola entra no raio de ação. Nesse caso espera-se que mais *frames* cheguem ao *buffer* e é novamente procurado o momento em que a bola sai do raio de ação.

Quando temos um *frame* de entrada e outro de saída da bola do raio de ação, é verificado se nos *frames* seguintes a bola volta a entrar no raio de ação do mesmo jogador, isto pois há situações de deslocação com bola em que o jogador coloca a bola um pouco à sua frente para poder correr, como esboçado na Fig. 3.15, o que se iria refletir no algoritmo como entradas e saídas da bola no raio de ação.

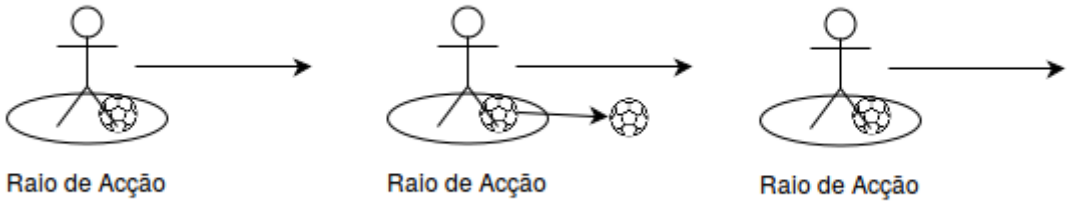

Figura 3.15: Exemplo ilustrativo de deslocação com bola, em que a bola entra e sai do raio de ação do jogador.

Tendo um *frame* de entrada e um de saída do raio de ação do jogador, procede-se ao cálculo das velocidades e dos ângulos de entrada e saída da bola. O cálculo das velocidades é efetuado para todos os *frames* onde a bola esteve dentro do raio de ação do jogador, sendo calculada a velocidade com um intervalo de 1 *frame*. A fórmula usada para o calculo da velocidade é a seguinte:

$$v = \sqrt{(delta_x)^2 + (delta_y)^2} \times framerate \tag{3.1}$$

em que $delta_x$ e $delta_y$ correspondem à distância percorrida do *frame* n ao $n+1$ em x e em y, respetivamente e o *framerate* ao número de *frames* por segundo do vídeo da par-

tida de futebol. No cálculo do ângulo de entrada são usados o *frame* inicial ($frame\_i_n$) e o seguinte ($frame\_i_{n+1}$) para o cálculo do ângulo de saída é usado o *frame* de saída ($frame\_s_n$) e seguinte ($frame\_s_{n+1}$). Para o cálculo do ângulo é usada a seguinte fórmula:

$$\alpha = atan2(delta_y, delta_x) \tag{3.2}$$

em que $delta_x$ e $delta_y$ são calculados da mesma forma do que para a velocidade e *atan2* é uma função implementada na maior parte das bibliotecas de cálculo computacional, corresponde ao arco tangente de duas variáveis, devolvendo um valor de α que varia no intervalo de $[-\pi, \pi]$. Com os valores da velocidade e dos ângulos calculados, passamos para a fase onde é decidido se um jogador tocou na bola ou não através da combinação dos mesmos. Como não foi possível encontrar uma forma em que a combinação do resultado da velocidade e do ângulo dessem os resultados de forma consistente e com menor percentagem de erro, foram criados vários aspetos decisivos que abrangem grande parte das situações de jogo de modo a garantir o maior número de trocas de bola detetadas com a menor percentagem de erro. Assim a velocidade e o ângulo são analisados separadamente e a cada um é atribuída uma percentagem de certeza que posteriormente é usada para decidir se houve troca de bola ou não. Para a análise das velocidades temos vários aspetos decisivos:

- Caso a diferença entre a velocidade de saída e a velocidade mínima seja maior ou igual que a velocidade média de um passe (10 m/s) (Kellis & Katis, 2007) (Paulo Sousa & Garganta, 2003), então é dado um valor de 90% de certeza para a velocidade. Esta situação pretende abranger os casos em que o jogador recebe a bola, permanece com as bola nos pés (velocidade baixa) e depois efetua um passe (velocidade elevada). Não poderia ser a diferença entre a velocidade de entrada e a de saída, pois caso estas fossem iguais (mesmo que elevadas) ou muito semelhantes, o valor da diferença seria muito baixo e não chegaríamos a qualquer conclusão. Neste caso usando a velocidade mínima, para quando o

jogador tem a bola nos pés, e a sua velocidade de saída, conseguimos com um maior nível de certeza saber que um jogador teve a bola nos pés e depois efetuou um passe, sendo o resultado desta diferença mais conclusiva.

- Caso a diferença entre a velocidade de saída e a velocidade de entrada ($delta_v$) esteja compreendida entre a velocidade mínima definida para um passe (5 m/s) e a velocidade máxima definida para um passe (15 m/s), é calculado um valor de percentagem de certeza de acordo com a seguinte fórmula:

$$p_c = \frac{delta_v \times 80}{velocidade\_maxima\_passe} \tag{3.3}$$

  podendo o valor variar entre aproximadamente *26,66%* (para $delta_v = 5m/s$) e *80%* (para $delta_v = 15m/s$). Neste caso pretende-se abranger as situações em que o jogador recebe a bola com uma velocidade relativamente baixa, e efetua um passe com uma velocidade superior à recebida, um exemplo desta situação é um passe ao primeiro toque, no qual a bola não chega a parar no pé do jogador.

- Caso a velocidade de saída seja menor que a velocidade mínima definida para um passe, então é verificado se durante o intervalo que o jogador teve a bola a velocidade máxima da bola é superior à velocidade mínima definida para um passe e se o número de *frames* em que o jogador teve a bola é superior a 60% do *framerate*. Nesta situação pretende-se detetar quando um jogador fica sem a bola sem ter efetuado um passe, por exemplo ter sofrido um desarme por parte do adversário. A contagem do número de *frames* serve para eliminar as situações em que a bola passa por dentro do raio de ação do jogador, sem este ter tido qualquer interação com a bola. Nesta situação o número de *frames* em que a bola está dentro do raio de ação do jogador é muito pequeno, logo ao dizermos que tem que ser maior que 60% do *framerate*, estamos a eliminar esse caso.

Na análise do ângulo apenas é considerada uma situação:

- Caso o módulo da diferença do ângulo de saída e do ângulo de entrada seja superior ou igual a 36 graus, então o ângulo é considerado como fator decisivo para o jogador ter tido a posse de bola, caso contrário o "fator" ângulo é desprezado. É necessário usar um valor mínimo em graus para a decisão, pois o *tracking* pode ter pequenos erros de precisão na localização da bola. Deste modo, tendo em conta que o ângulo devolvido pela função *atan2* pode variar entre -180^{o} e 180^{o}, considerou-se um valor mínimo de 20% de 180^{o}, chegando-se assim aos 36 graus anteriormente referidos.

Finalmente, apenas temos que juntar os resultados obtidos para a velocidade e para o ângulo. Antes do cruzamento dos dados é verificado se o número de *frames* em que a bola esteve dentro do raio de ação do jogador é superior a 8% do *framerate*, equivalendo a 2 *frames* para um *framerate* de 25 *frames* por segundo. Assim conseguimos eliminar os casos em que a bola apenas passa por dentro do raio do jogador, sem este nunca ter tido contacto com ela, pois para existir uma entrada da bola no raio de ação, contacto com a bola e saída do raio de ação, a bola teria que percorrer 3,6 metros (2 x raio de ação) em $\frac{2}{25}$ segundos (considerando um *framerate* de 25 *frames* por segundo), a qual teria que viajar a uma velocidade de 45 m/s, velocidade esta quase impossível de alcançar num remate de futebol, logo muito menos provável para um passe. Sendo assim, caso o número de *frames* em que a bola está no raio de ação do jogador seja superior a 8% do *framerate*, são consideradas como trocas de bola as seguintes situações:

- Caso a percentagem de certeza para a velocidade seja superior a 80%;
- Caso a diferença entre o *frame* de saída e o *frame* de entrada seja superior a 2 vezes o *framerate*;
- Caso o ângulo e a velocidade tenham sido considerados como fatores decisivos, não entrando a percentagem de certeza, i.e., caso os valores calculados para o ângulo e a velocidade se enquadrem numa das condições anteriormente referidas para ambos.

- Caso o resultado da verificação do ângulo seja superior a 70 graus.

No segundo caso apenas são considerados os *frames*, pois caso a bola tenha permanecido durante mais de 2 segundos dentro do raio de ação do jogador, então é extremamente provável que a bola tenha estado na sua posse, não sendo assim necessário recorrer à velocidade e ao ângulo para decisão. No ultimo caso é usado o valor de 70 graus, pois sendo este o único fator utilizado para deteção da troca de bola, teria que ser um valor que não deixa-se duvidas quanto à existência de troca de bola. Este valor foi alcançado após alguns testes.

Para guardar as trocas de bola de modo a que a sua ordem não seja comprometida, estas são guardadas numa lista, em que cada elemento da lista é composto por um *tuplo* que contém o número de jogador, equipa, *frame* de entrada e saída da bola do raio de ação e uma percentagem de certeza. Na Fig. 3.16 está representado um exemplo da lista com a estrutura anteriormente descrita.

```
1 [(('9', 'A'), 1, 8, 90), (('1', 'A'), 24, 37, 90), (('6', 'A'), 49, 53, 90)
    , (('1', 'A'), 64, 81, 0), (('5', 'B'), 91, 93, 33.83), (('6', 'B'),
    107, 137, 90), (('7', 'A'), 153, 156, 0)]
```

Figura 3.16: Exemplo da estrutura usada para armazenar as trocas de bola.

Na Secção 4.3 serão apresentados alguns testes ao algoritmo.

3.3.2 Passes

Compreende-se como passe a ação de enviar a bola para um companheiro, sendo assim, todas as trocas de bola efetuadas entre membros da mesma equipa são considerados como passes. Para esse efeito apenas é necessário percorrer a lista com as trocas de bola e verificar as trocas de bola que existem para a mesma equipa, ficando assim com os instantes em que os passes foram efetuados e recebidos e qual o jogador que passou e recebeu.

Para que os passes pudessem ser analisados de forma visual, foi definido no projeto que estes teriam que aparecer num campo de futebol como a forma de setas, de

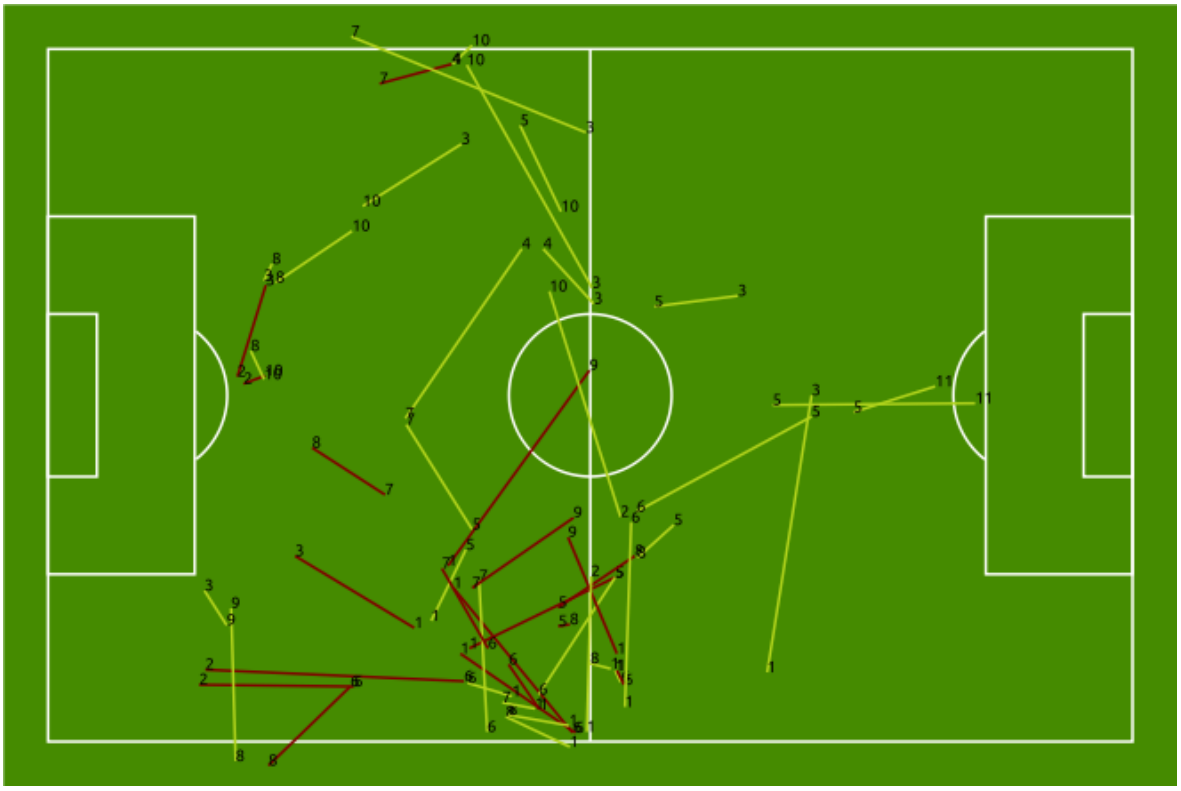

Figura 3.17: Mapa de passes gerado em SVG contendo os passes efetuados por ambas as equipas. A vermelho os passes efetuados pela equipa *A*, e a verde claro os passes efetuados pela equipa *B*.

cor diferente para cada equipa ao qual foi chamado de *mapa de passes*. Como este seria inserido em ambiente web, escolheu-se usar o SVG para a sua criação, devido à possibilidade de criar imagens vetoriais, adaptando-se assim às várias resoluções de ecrã que os utilizadores possam usar e à sua composição ser basicamente XML, não tendo assim grande impacto na performance da aplicação web. De modo a agilizar a criação do SVG, foi criada uma biblioteca em Python na qual é criado o SVG de forma dinâmica, podendo este ser inserido diretamente na página web ou gravado para um ficheiro.

Na criação do *mapa de passes* deve ser possível escolher todos os passes efetuados no jogo por ambas as equipas (Fig. 3.17), escolher apenas os passes de uma equipa (Fig. 3.18) ou então os passes efetuados por um determinado jogador (Fig. 3.19).

Como já anteriormente foi referido, para encontrar os passes efetuados é necessário

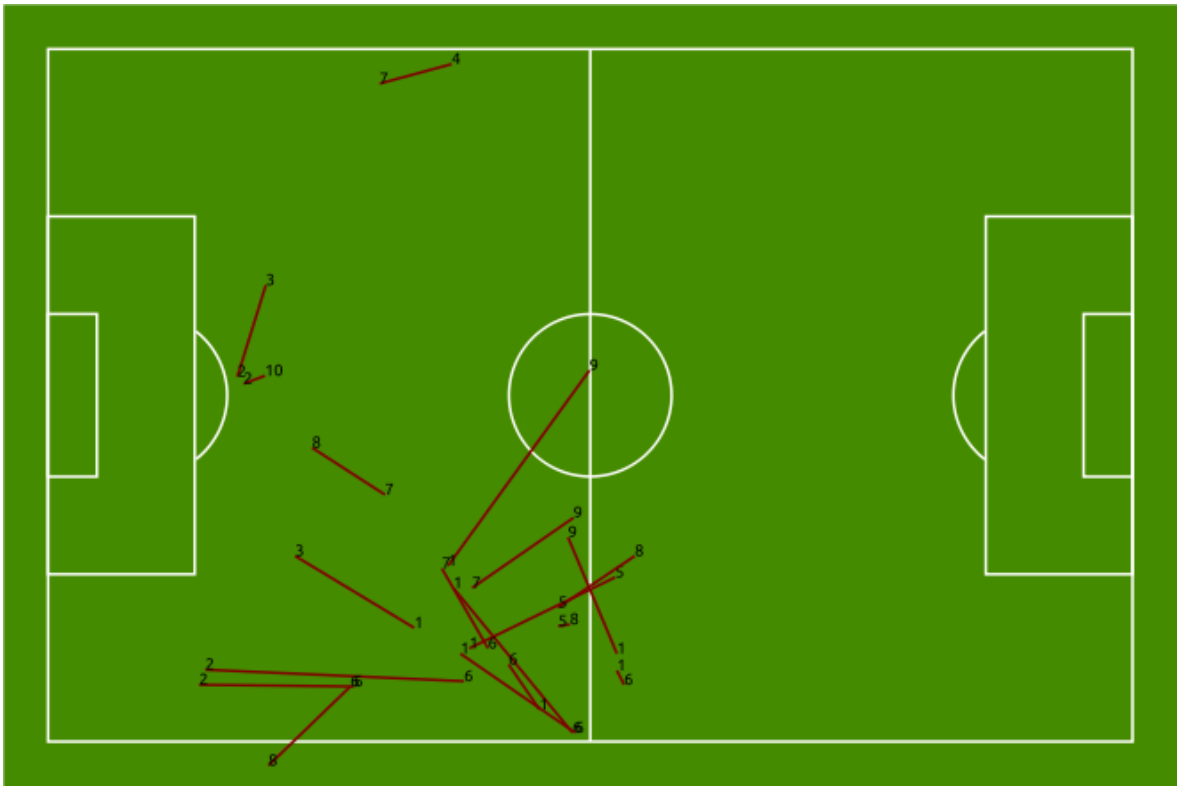

Figura 3.18: Mapa de passes gerado em SVG contendo os passes efetuados por apenas uma equipa.

percorrer a lista de trocas de bola e procurar as trocas de bola existentes entre elementos da mesma equipa. Usando a Fig. 3.16, diríamos que o jogador *9* efetuou um passe no *frame 8* (momento em que a bola saiu do seu raio de ação) e o jogador *1* recebeu a bola no *frame 24* (momento em que a bola entrou no seu raio de ação). Depois de encontrados os passes, é necessário colocar a posição exata do passe e da receção no *mapa de passes*, sendo preciso fazer uma correspondência entre o número do *frame* e a posição do jogador naquele instante. Essa correspondência resulta numa nova lista (lista de passes), contendo a posição em píxeis onde se encontrava o jogador quando efetuou o passe e a posição quando recebeu, assim como os números e equipas dos mesmos, resultando na seguinte estrutura:

$$[((x_p, y_p), (x_r, y_r)), (jogador_p, equipa_p), (jogador_r, equipa_r)] \tag{3.4}$$

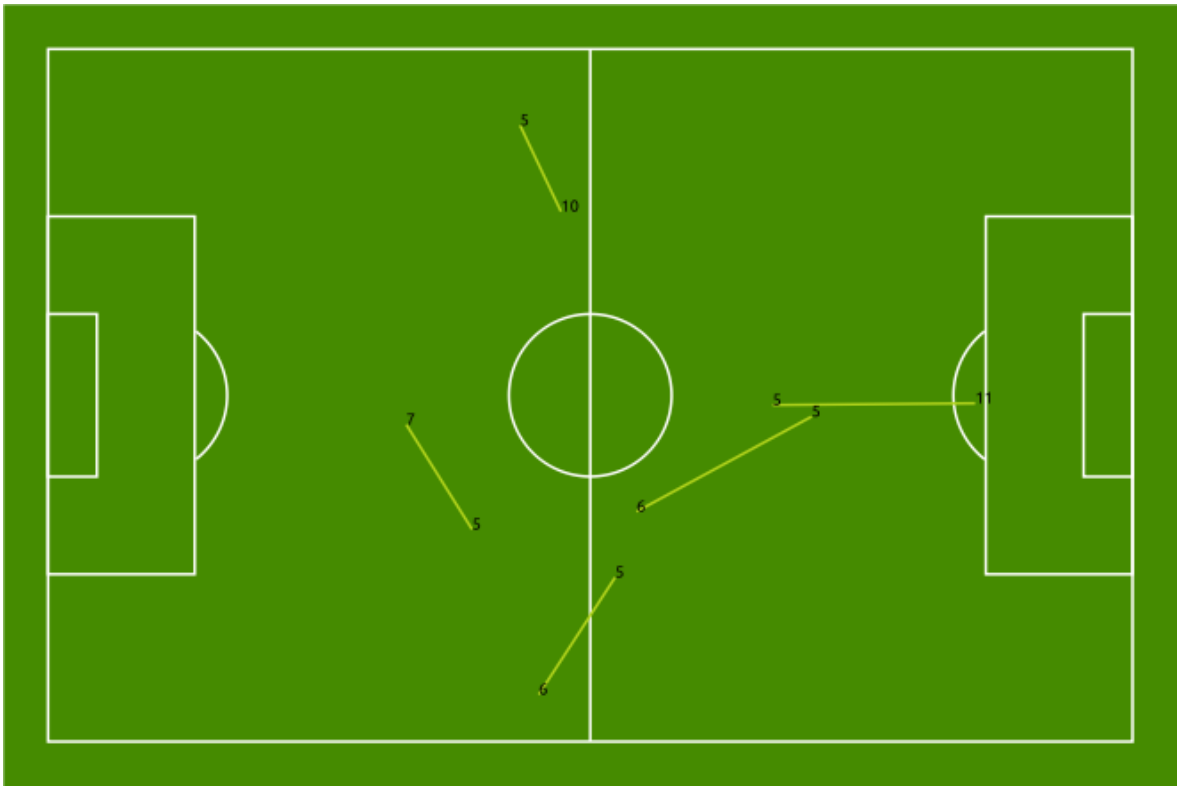

Figura 3.19: Mapa de passes gerado em SVG contendo os passes efetuados por um jogador (número 5 da equipa B).

em que (x_p, y_p) e (x_r, y_r) correspondem às coordenadas onde é feito e onde é recebido o passe respetivamente e $(jogador_p, equipa_p)$ e $(jogador_r, equipa_r)$ correspondem ao número e equipa do jogador que faz e recebe o passe. Depois de criada a lista, esta é usada no módulo SVG anteriormente referido, para criar as setas ilustrativas dos passes. Na Fig. 3.20 podemos ver o exemplo da lista gerada para todos os passes efetuados pelo jogador 5 da equipa B, ilustrada na Fig. 3.19.

```
1 [[((638, 595), (552, 727)), ('5', 'B'), ('6', 'B')], [((531, 86), (577,
      183)), ('5', 'B'), ('10', 'B')], [((815, 401), (1043, 399)), ('5', 'B'),
       ('11', 'B')], [((859, 414), (662, 521)), ('5', 'B'), ('6', 'B')],
      [((477, 541), (403, 423)), ('5', 'B'), ('7', 'B')]]
```

Figura 3.20: Lista de passes gerada para o jogador 5 da equipa B.

3.3.3 Perdas de Bola

Para a deteção das perdas de bola, apenas se tem que verificar quando a troca de bola é feita de uma equipa para a outra, querendo assim dizer que o jogador que tinha a bola, perdeu-a para o adversário.

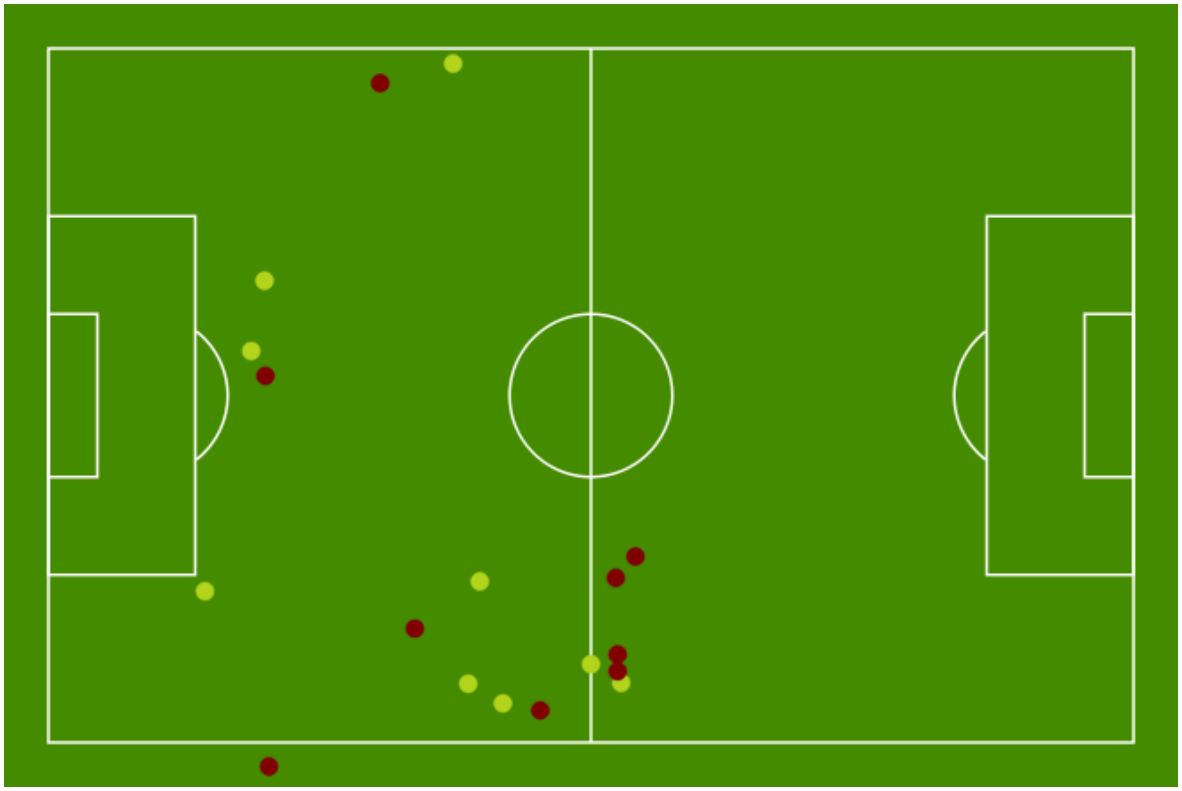

Figura 3.21: Mapa de perdas de bola gerado em SVG contendo as perdas de bola efetuadas por ambas as equipas.

De modo análogo ao *mapa de passes*, também é usada a biblioteca SVG para a geração do *mapa de perdas de bola* sendo também possível selecionar as perdas de bola por equipa e por jogador. Para a construção do mapa é utilizada a mesma estrutura que em 3.20, mas contendo apenas as situações em que o valor de $equipa_p \neq equipa_r$. As perdas de bola estão identificadas no mapa por circunferências, correspondendo a cor vermelha a perda por parte da equipa A e a cor verde a perda por parte da equipa B. Na Fig. 3.21 está ilustrado o *mapa de perdas de bola* gerado para as duas equipas. Para o *mapa de perdas de bola* da equipa (Fig. 3.22) e do jogador (Fig. 3.23), foram colocados também os passes para melhor se compreender como o jogador perdeu a bola.

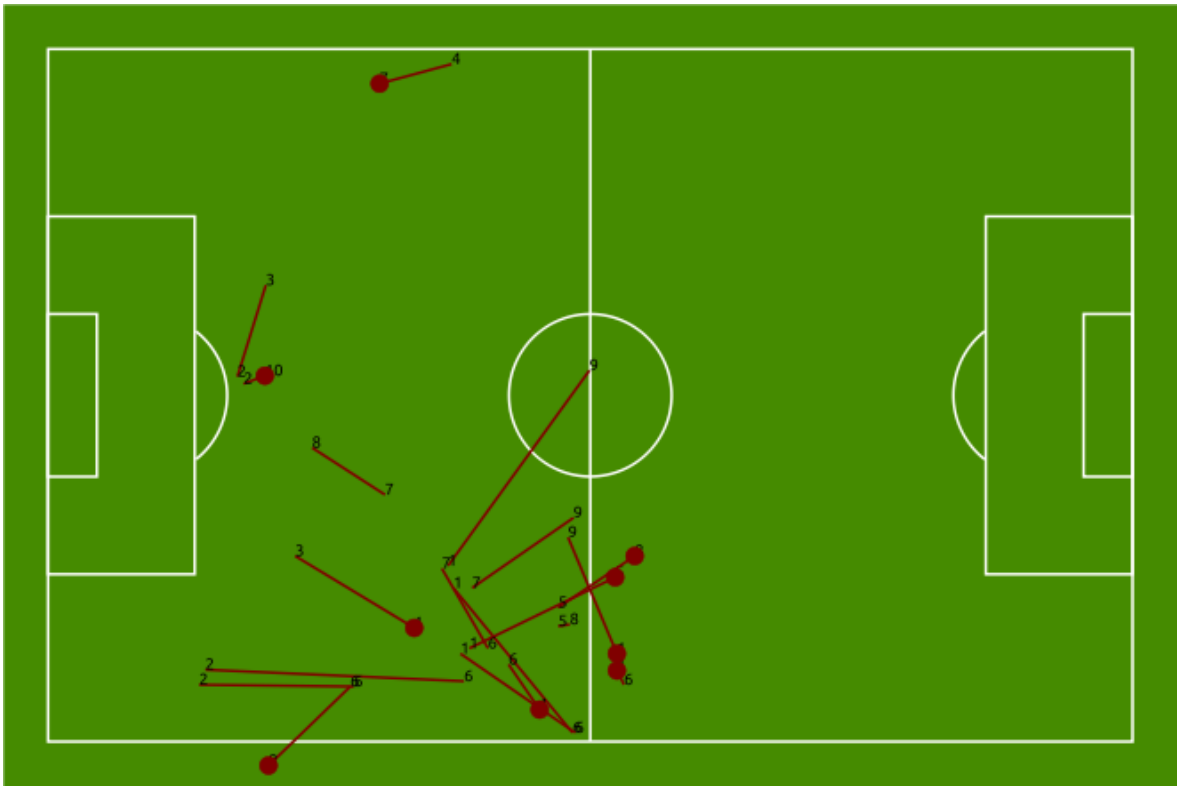

Figura 3.22: Mapa de perdas de bola gerado em SVG contendo as perdas de bola e passes efetuados para a equipa A.

Ao analisar a Fig. 3.22, é visível que as perdas de bola (ilustradas com uma circunferência a vermelho) estão colocadas nas extremidades de passes, podendo-se entender que o jogador perdeu a bola ao efetuar um passe, ou então foi-lha roubada enquanto a tinha nos pés. Estas situações podem ser distinguidas ao analisar a distância de onde ocorreu o passe e de onde foi feita a perda de bola, podendo-se assim distinguir ainda dentro das perdas de bola, perdas de bola no passe e em deslocação de bola/drible.

3.4 Mapa de Calor

Os Mapas de Calor são representações gráficas de dados onde os valores individuais de uma matriz são representados como cores. No desporto os mapas de calor podem ter diferentes usos, tal como a deteção de áreas mais usadas por parte de uma equipa,

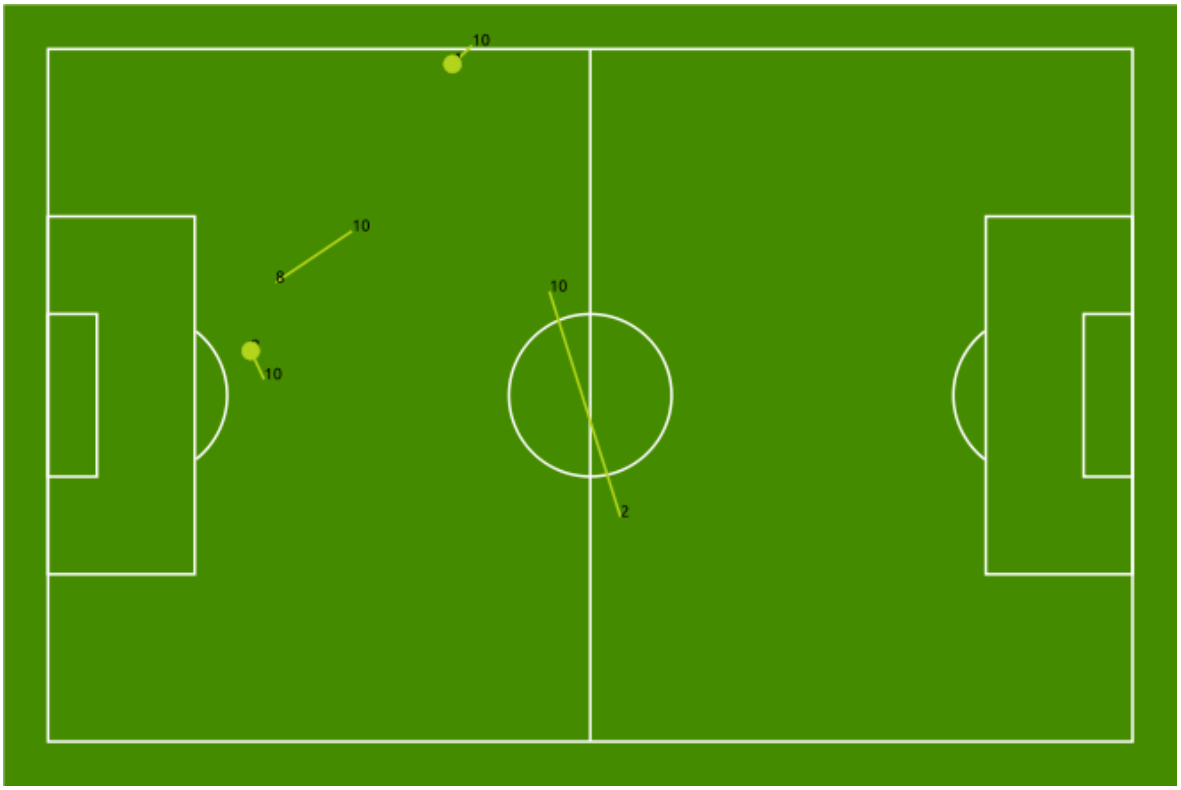

Figura 3.23: Mapa de perdas de bola gerado em SVG contendo as perdas de bola efetuadas pelo jogador 9 da equipa B.

jogador ou conjunto de jogadores. No caso particular do futebol, os mapas de calor podem ser usados para detetar se um jogador tem um lado ou área preferido ou se a defesa está a ocupar bem os espaços no campo.

No nosso caso para a criação do mapa de calor, começamos por definir uma matriz, HM, que irá conter os valores a serem representados. HM é definida com a mesma dimensão que a resolução usada para representar o mapa de calor, i.e., caso se pretende um mapa de calor com uma resolução de 100x60, então HM terá 100 por 60 quadrados. A matriz é preenchida com dados provenientes do sistema de *tracking*, podendo assim ser criados mapas de calor para a bola, ambas as equipas, um determinado conjunto de jogadores ou um só jogador. O processo de criação do mapa de calor, começa por definir uma matriz nula com a dimensão pretendida. No passo seguinte, como referido, são usados os dados do sistema de *tracking* para preencher as posições correspondentes ao elementos do mapa de calor. O preenchimento é feito para cada *frame*,

$$HM_1 = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} \quad HM_2 = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 2 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Figura 3.24: Exemplo em que na matriz HM1 o elemento central não é afetado pelo fator de dispersão e em HM2 é afetado por um fator de dispersão igual a 1.

e a correspondência da posição (x, y) do elemento (bola ou jogador) no campo, com a posição (x', y') da matriz HM, é feita através da aplicação de duas regras de três simples, tendo em conta as dimensões do campo de jogo, a resolução do mapa de calor, e a posição dos elementos no campo. Tendo a posição (x', y') calculada, o elemento $HM_{x'y'}$ é incrementado uma unidade. É ainda possível definir um fator de dispersão, em que caso este seja igual a x, todas a posições com distância de x elementos à volta do $HM_{x'y'}$ são incrementadas uma unidade, inclusive o próprio (Fig. 3.24).

Uma vez que todas as *frames* sejam processadas, a matriz HM é normalizada, obtendo-se a matriz $\overline{HM}$, i.e., cada elemento da matriz é dividido pelo maior valor existente na matriz. Finalmente, com os valores da matriz HM a variarem entre 0 e 1, utiliza-se as funções da biblioteca implementada para criar o SVG do mapa de calor usando os valores da matriz $\overline{HM}$ para definir a opacidade da cor. Deste modo as zonas de maior ação ficam mais opacas, e as de menor com uma tonalidade mais transparente.

Na Fig. 3.25 é mostrado o mapa de calor da equipa A, criado através da biblioteca SVG, com um fator de dispersão igual a 1 onde é visível a zona onde a equipa esteve mais concentrada durante um intervalo de tempo de 2 minutos. Na Secção 4.4 serão apresentados um conjunto mais alargado de exemplos.

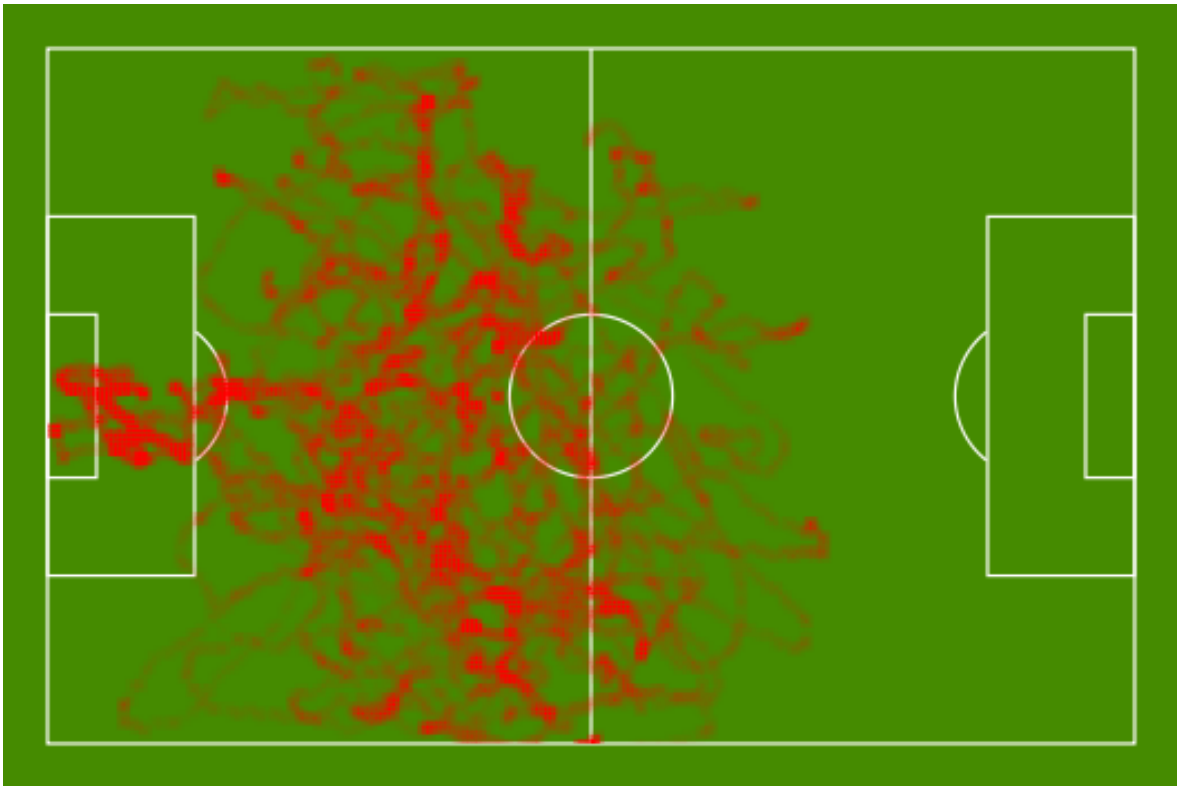

Figura 3.25: Mapa de Calor da equipa A, com um fator de dispersão igual a 1.

4

Testes e Resultados

Neste capítulo apresentamos e discutimos alguns dos resultados obtidos para as várias aplicações aqui apresentadas. Todos os testes aqui efetuados usam 2 minutos de um jogo de futebol em que são dadas as coordenadas dos jogadores e da bola em pixéis. As equipas serão chamadas de Equipa A e Equipa B, tendo ambas 11 jogadores numerados de 1 a 11. Os tempos de execução apresentados foram obtidos num computador com um processador Intel Core i5-3470, 16 GB de memória RAM e sistema operativo Ubuntu 14.04.

4.1 Modelos de Jogo - Esquemas Estáticos

Nesta Secção vamos começar por proceder à análise do exemplo do EE ilustrado na Fig. 3.1, em que quando a bola se encontra na zona verde, os quatro jogadores (bolas a vermelho) devem-se encontrar dentro da zona a vermelho e ter uma distância entre si de 12 metros admitindo um erro de 3 metros. A Fig. 4.1 é uma ilustração do que está a ser processado pelo detetor de modelos de jogo, em que é possível ver as zonas e os jogadores (bolas a rosa) que foram inseridos no Editor de Campo e os jogadores (bolas vermelhas e azuis) e a bola nas posições provenientes do sistema de *tracking*. Esta figura mostra uma situação em que o desenho do modelo de jogo é verificado no campo, ou seja todas as condições definidas no interface Editor de Campo através da Linguagem de Programação Visual e consequentemente no ficheiro Python estão a acontecer. A Fig. 4.2 mostra aproximadamente o mesmo instante da Fig. 4.1 mas na imagem real do jogo, em que foram desenhadas as zonas (manualmente) de modo a haver uma base de comparação.

Comparando as duas imagens é visível que os jogadores e a bola estão a ser detetados corretamente dentro das áreas.

Sendo que para cada *frame* processada é gerado um documento JSON como o da Fig. 3.9 apenas irão ser mostradas algumas estatísticas obtidas através da análise desses documentos. Assim, pela análise dos 1440 documentos (2 minutos a 12 *frames* por segundo) gerados retirou-se a seguinte informação:

- O desenho do modelo de jogo aconteceu em 142 *frames*.
- A bola esteve na zona a verde durante 248 *frames*.
- O jogador 1 esteve 236 *frames* dentro da zona a vermelho.
- O jogador 2 esteve 248 *frames* dentro da zona a vermelho.
- Os jogadores 3 e 4 estiveram 243 *frames* dentro da zona a vermelho.
- Todos os jogadores estiveram dentro da zona 236 *frames*.

- As distâncias foram cumpridas em 142 *frames*.
- A distância entre o jogador 1 e 2 foi cumprida em 214 *frames*.
- A distância entre o jogador 2 e 3 foi cumprida em 234 *frames*.
- A distância entre o jogador 3 e 4 foi cumprida em 151 *frames*.
- A distância entre o jogador 3 e 4 foi menor que a distância definida de 12 metros, em 83 *frames*.

Ao analisar estes dados, podemos concluir que o jogador número 4 teve alguma culpa no fato de o desenho não se ter verificado mais vezes durante o jogo, uma vez que a bola esteve dentro da zona durante 248 *frames*, o desenho do modelo de jogo só foi verificado durante 142 *frames* e a distância entre o jogador 3 e 4 foi a que se verificou menos vezes em comparação com as outras distâncias. Caso esta situação fosse recorrente durante o jogo, o treinador poderia chamar a atenção ao jogador 4, dizendo que ele teria que aumentar a distância ao jogador 3 naquela situação. Isto baseado no facto de que na maior parte do tempo em que o desenho não se verificou, foi devido à distância entre ele e o jogador 3 ser menor que os 12 (9 metros contando com a margem de 3 metros) metros definidos.

Na Fig. 4.3 está ilustrada uma situação em que quando o jogador 9 da equipa B (azul) e o jogador 2 da equipa A (vermelha) estiverem dentro da zona, estes devem-se encontrar a uma distância de 3 metros (com uma margem de erro de 3 metros). O documento JSON gerado pelo Editor de Campo (Fig. 4.4) é então usado para criar o código Python (Fig. 4.5) para poder ser feita a sua deteção no jogo.

Mais uma vez, apenas irão ser mostradas algumas estatísticas obtidas através da análise dos documentos JSON obtidos na deteção da situação desenhada no jogo de futebol. Assim, pela análise dos 1440 documentos (2 minutos a 12 *frames* por segundo) retirou-se a seguinte informação:

- O desenho do modelo de jogo aconteceu em 379 *frames*.

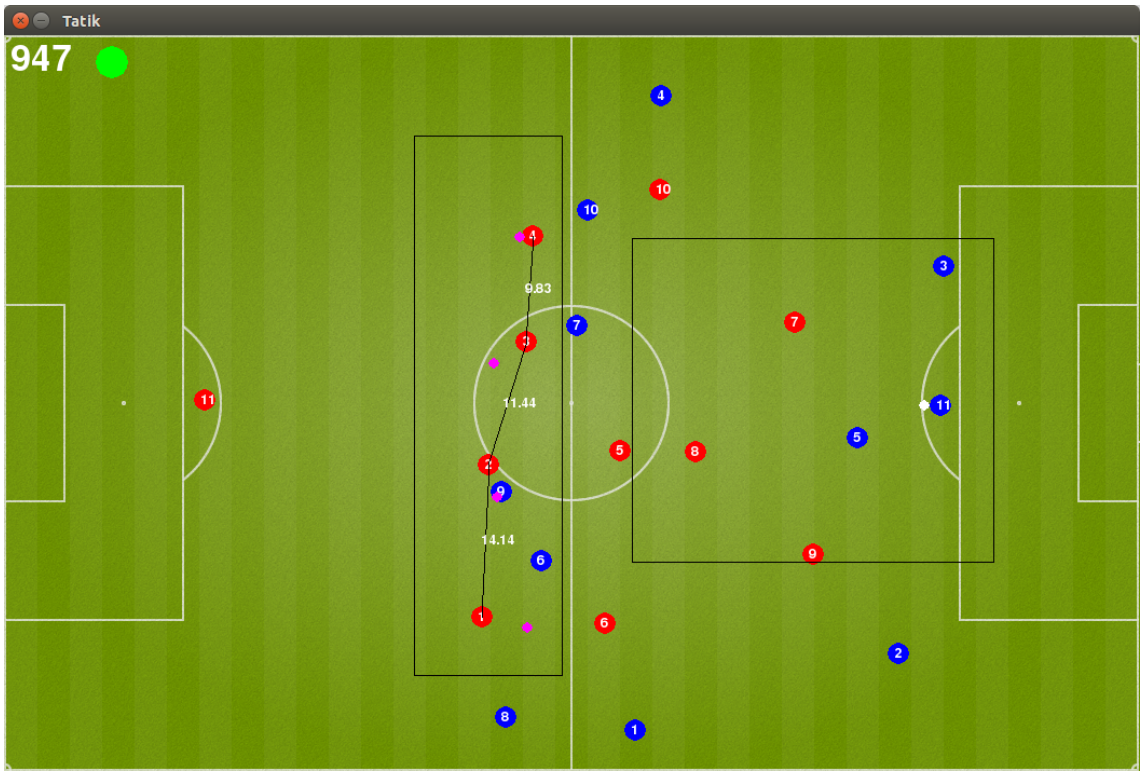

Figura 4.1: Processamento do Esquema Estático da Fig. 3.1, contendo um instante em que o desenho do modelo de jogo é validado. A vermelho os jogadores da equipa A e a azul os da equipa B.

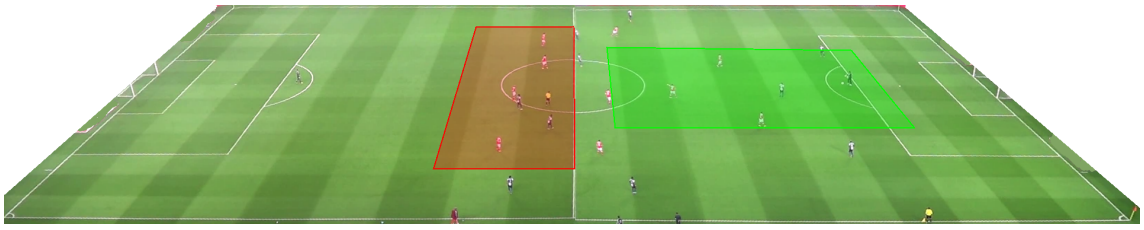

Figura 4.2: Imagem real do jogo, com instante aproximado da verificação referente ao Esquema Estático da Fig. 4.1.

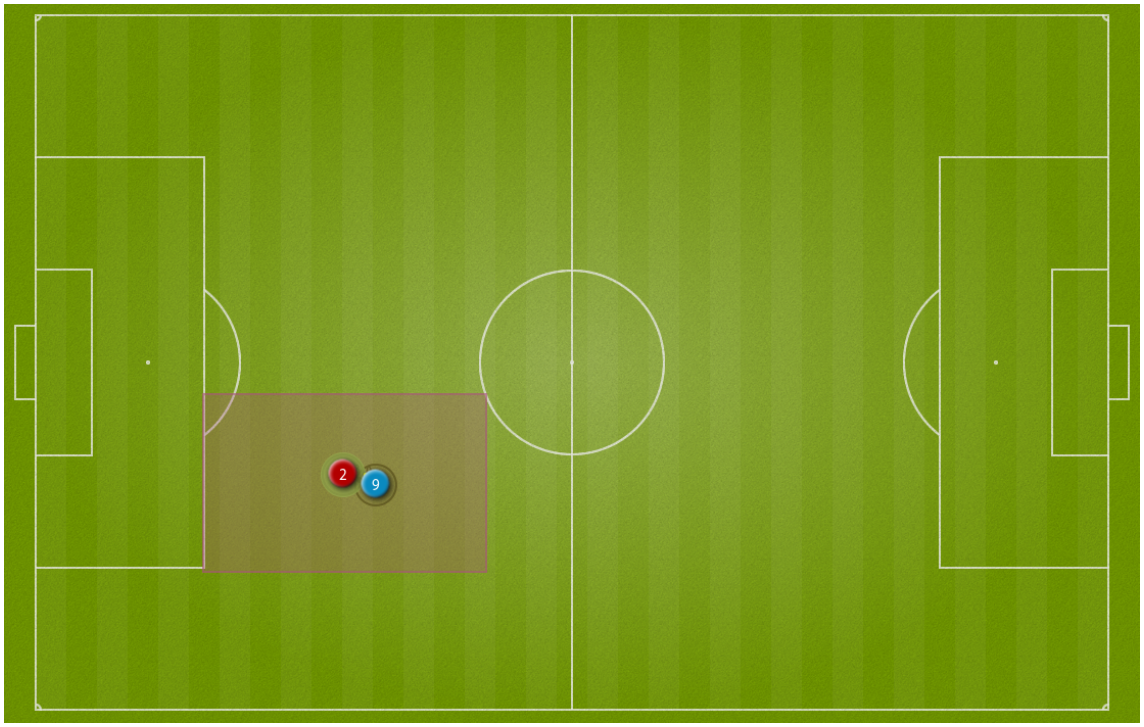

Figura 4.3: Esquema Estático contendo uma situação em que quando os dois jogadores se encontram dentro da zona, estes devem-se encontrar a uma distância de 3 metros.

- O jogador 9 esteve 959 *frames* dentro da zona.
- O jogador 2 esteve 625 *frames* dentro da zona.
- Os jogadores estiveram dentro da zona ao longo de 625 *frames*.
- A distância entre o jogador 9 e 2 foi maior que a distância definida de 6 metros (3 metros mais 3 de margem), em 246 *frames*.

Com este nível de detalhe sobre o que aconteceu durante o jogo, o treinador e a equipa técnica podem alertar os jogadores sobre o que devem corrigir de forma a melhorar o seu desempenho no jogo.

Quanto ao desempenho da aplicação, para o primeiro exemplo foram gastos aproximadamente de 0.07 segundos e no segundo aproximadamente 0.05 segundos, tornando assim possível a sua utilização na deteção de esquemas estáticos com muito maior complexidade em tempo real.

```
{ "name" : "marcacao2ao9",
  "type" : "Processo Defensivo",
  "field_dimension" : [  105,  68 ],
  "field_scale" : [  0.1,  0.1 ],
  "players" : [
      {"id" : 2, "number" : 2, "team" : "A", "points" : [332, 461], "radius
         " : 3 },
      {"id" : 3, "number" : 9, "team" : "B", "points" : [364, 471], "radius
         " : 3 }
  ],
  "zones" : [
      {"id" : 1, "name" : "Zona 1", "points" : [ [473, 557 ], [473, 381 ],
         [194, 381 ], [194, 557 ] ] }
  ],
  "lines" : [  {  "id" : 4,  "objects" : [  2,  3 ] } ],
  "distances" : [{"objects" : [2, 3], "distance" : 3, "margin" : 3 }],
  "conditions" : [
      [
           {"verify_location" : [3, "IN", 1 ] },
           {"verify_location" : [2, "IN", 1 ] },
           {"verify_distances" : "true" }
      ],
      [
           {"alert" : "True", "msg" : "O modelo ocorreu!" }
      ],
      [
           {"alert" : "True", "msg" : "O modelo nao ocorreu!" }
      ]
   ]
 }
```

Figura 4.4: Exemplo de um documento JSON gerado pela ferramenta Editor de Campo.

```
from lib import gpm
def run():
  result = False
  if gpm.verify_location(frame,[players3,'IN',zones1]) :
     if gpm.verify_location(frame,[players2,'IN',zones1]) :
        if gpm.verify_distances(frame, distances) :
           gpm.sendmsg('O modelo ocorreu!')
           gpm.sendalert('True')
           result = True
  if not result:
     gpm.sendmsg('O modelo nao ocorreu!')
     gpm.sendalert('True')
     result = False
  return result, gpm
```

Figura 4.5: Código Python gerado a partir do documento JSON da Fig. 4.4.

4.2 Modelos de Jogo - Esquemas Dinâmicos

Relativamente aos ED, vamos proceder à análise do exemplo ilustrado na Fig. 3.10, em que o jogador 8 quando está na zona a laranja tem que passar a bola para o jogador 1 que se deverá encontrar dentro da zona a verde, que por sua vez tem que passar ao jogador 6 que deve estar na zona a azul. Como referido na Secção 3.2, este esquema será dividido em 3 esquemas estáticos que serão detetados individualmente, respeitando a ordem do desenho.

Primeiro irá ser detetado quando o jogador 8 e a bola se encontram dentro da zona a laranja. Na Fig. 4.6 é mostrado um momento em que essa situação está a acontecer. Quando a situação deixa de acontecer, é verificado se o intervalo de *frames* em que a situação esteve a acontecer é superior a 10 *frames*. A Fig. 4.7 mostra aproximadamente o mesmo instante da Fig. 4.6 mas na imagem real do jogo, em que foi desenhada a zona (manualmente) de modo a haver uma base de comparação. Como o número de *frames* foi superior a 10, então passamos à deteção do segundo momento, estando ilustrado na Fig. 4.8 o instante da sua deteção na ferramenta de análise e na Fig. 4.9 o respetivo instante na imagem real do jogo. Uma vez que está situação ocorreu mais do que 10 *frames*, passa-se à detecção do próximo e último momento, em que mais uma vez na Fig. 4.10 é mostrado o instante da sua deteção na ferramenta de análise e na Fig. 4.11 o instante na imagem real do jogo. Este último foi validado 9 vezes, confirmando assim que o esquema dinâmico aconteceu durante o jogo.

Comparando as imagens das deteções dos vários momentos com as imagens reais, é percetível que nos 3 momentos do esquema os jogadores e a bola encontravam-se dentro da respetiva zona, validando assim a deteção dos mesmos por parte da ferramenta de análise.

No exemplo da Fig. 4.12 é ilustrada uma situação em que o jogador 11 quando está na zona a amarelo deve passar a bola ao jogador 5 que se deve encontrar na zona a azul, que de seguida deve passar ao jogador 6 que se deverá encontrar na zona a rosa,

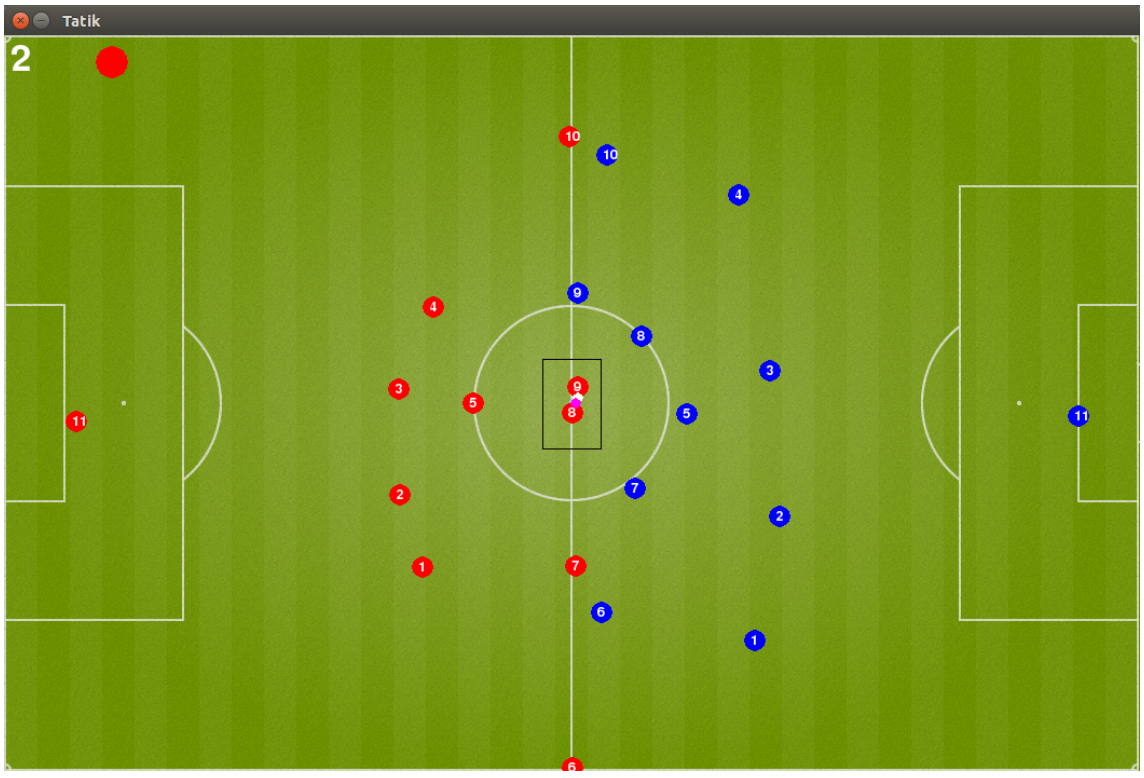

Figura 4.6: Processamento do Esquema Dinâmico da Fig. 3.10, contendo um instante em que o momento 1 do desenho do modelo de jogo é validado.

Figura 4.7: Imagem real do jogo, com instante aproximado da verificação referente ao momento 1 do Esquema Dinâmico da Fig. 4.6.

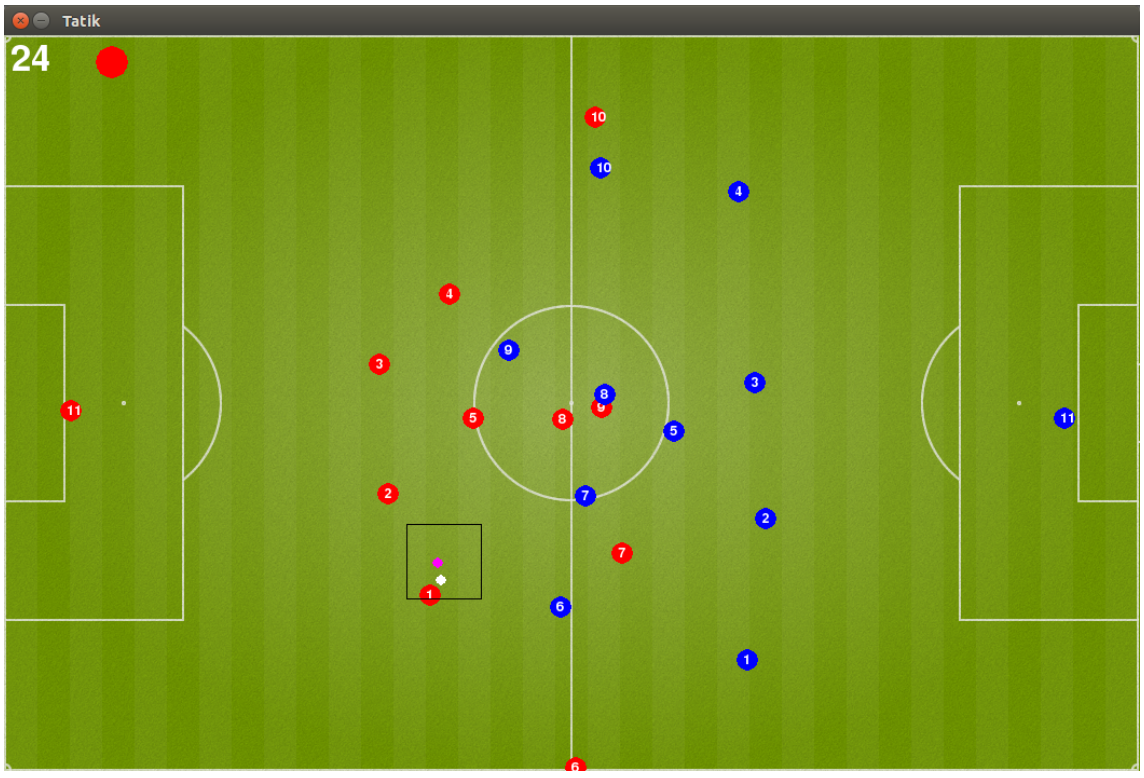

Figura 4.8: Processamento do Esquema Dinâmico da Fig. 3.10, contendo um instante em que o momento 2 do desenho do modelo de jogo é validado.

Figura 4.9: Imagem real do jogo, com instante aproximado da verificação referente ao momento 2 do Esquema Dinâmico da Fig. 4.8.

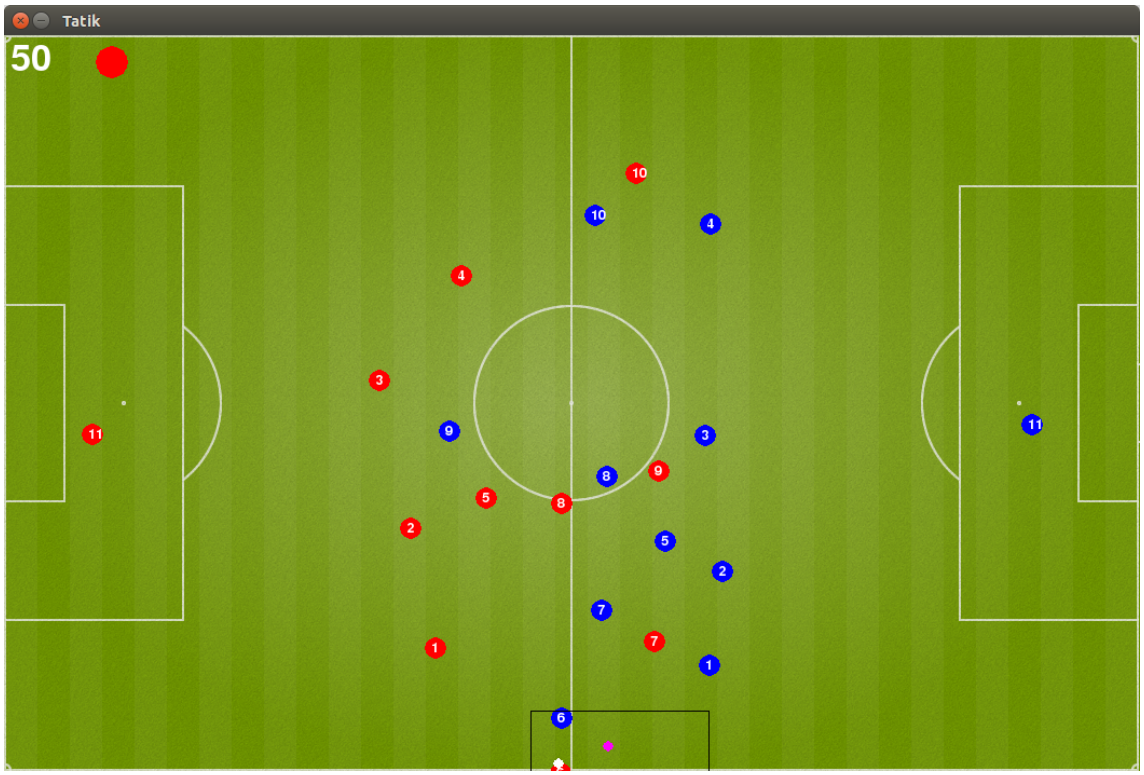

Figura 4.10: Processamento do Esquema Dinâmico da Fig. 3.10, contendo um instante em que o momento 3 do desenho do modelo de jogo é validado.

Figura 4.11: Imagem real do jogo, com instante aproximado da verificação referente ao momento 3 do Esquema Dinâmico da Fig. 4.10.

que por sua vez passa ao jogador 1 na zona a vermelho e por fim é feito um passe para a zona a verde. Em simultâneo com o último passe, o jogador 6 deve-se deslocar da zona a rosa para a zona a verde. Para a deteção do ED, o esquema foi dividido em 5 EE, que serão detetados individualmente respeitando a ordem do desenho.

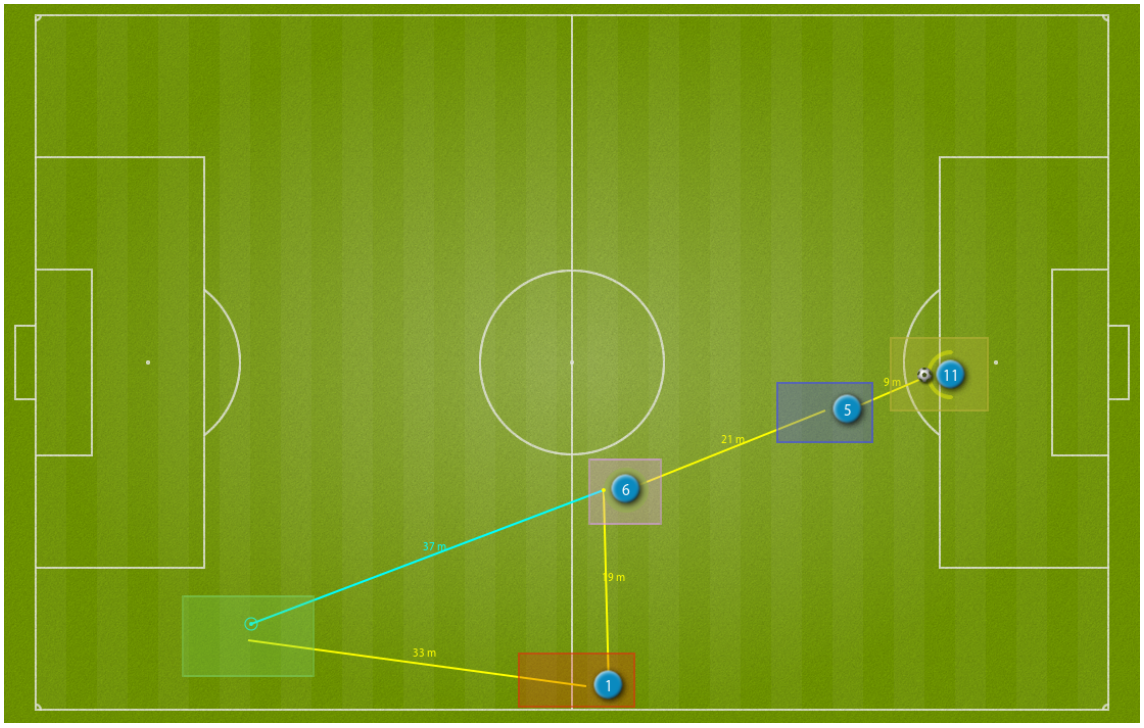

Figura 4.12: Esquema Dinâmico em que pretende-se detetar se a bola passa por os jogadores 11, 5, 6 e 11 nas zonas definidas para os mesmos e por fim uma desmarcação para a zona a verde.

Primeiro irá ser detetado quando o jogador 11 e a bola se encontram dentro da zona a amarelo (Fig. 4.13 a)), de seguida o momento em que o jogador 5 e abola estão dentro da zona a azul (Fig. 4.13 b)). Uma vez validados estes dois momentos, passamos à deteção dos momento em que o jogador 5 e a bola se encontram na zona a rosa (Fig. 4.14 a)) e de seguida passa ao jogador 1 que deve se encontrar dentro da zona a vermelho (Fig. 4.14 b)). Para o último momento, temos que o jogador 1 irá ter que fazer o passe (ilustrado pela linha amarela) para a zona a verde e o jogador 6 terá que se deslocar (ilustrado pela linha azul) da zona rosa para a zona a verde (Fig. 4.12). Este momento resume-se mais uma vez por detetar se o jogador 6 e a bola se

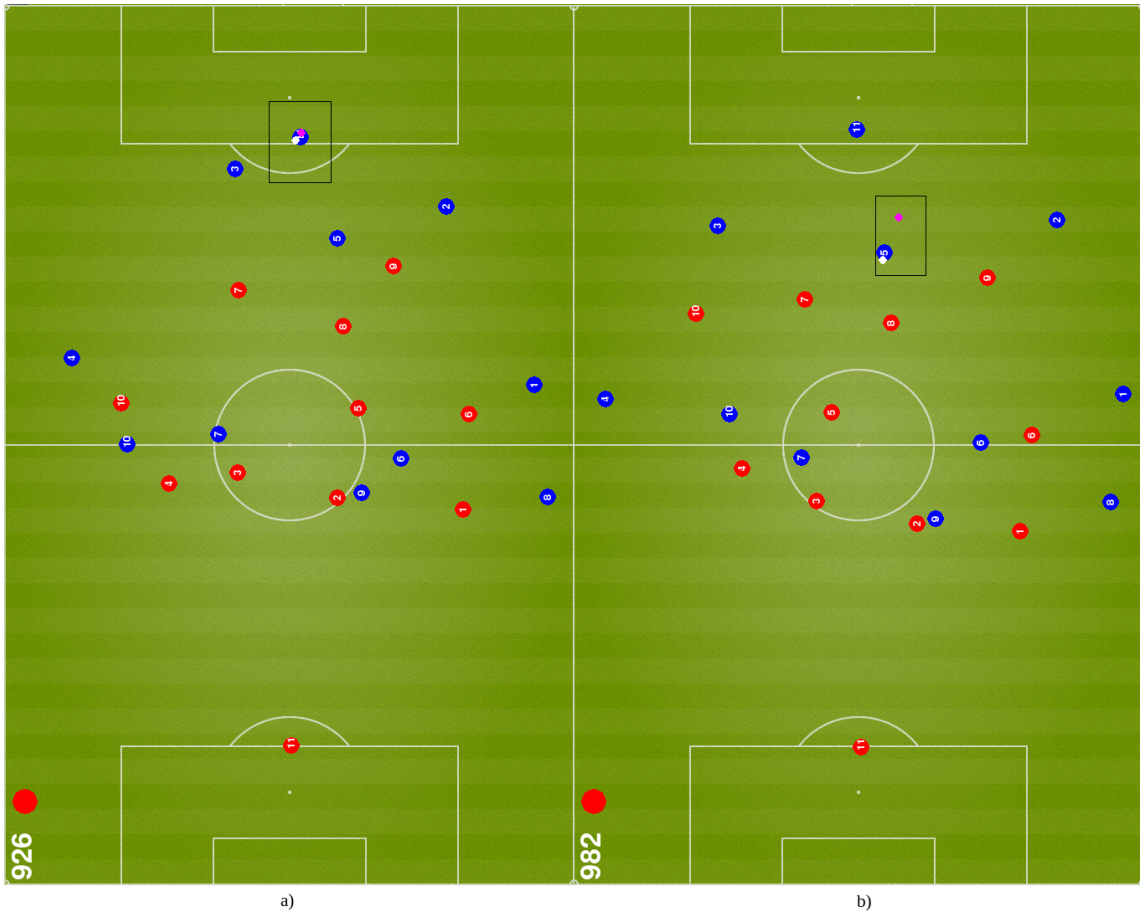

Figura 4.13: Processamento do Esquema Dinâmico da Fig. 4.12, contendo um instante em que o momento do desenho do modelo de jogo é validado: a) momento 1 e b) momento 2.

encontram na zona a verde, ilustrado na Fig. 4.15.

Quanto ao desempenho da aplicação, esta demorou cerca 0.05 segundos para processar o primeiro exemplo e aproximadamente 0.06 segundos para o segundo exemplo, tornando assim possível a sua utilização na deteção de esquemas estáticos em tempo real (mesmo que bastante mais complexos).

4.3 Deteção de Trocas de Bola

Para testar o algoritmo de trocas de bola, irão ser mostrados os resultados obtidos através da variação do valor de raio de ação do jogador, valor em metros, para assim serem comparados os resultados obtidos. No excerto do jogo existe um total de 52

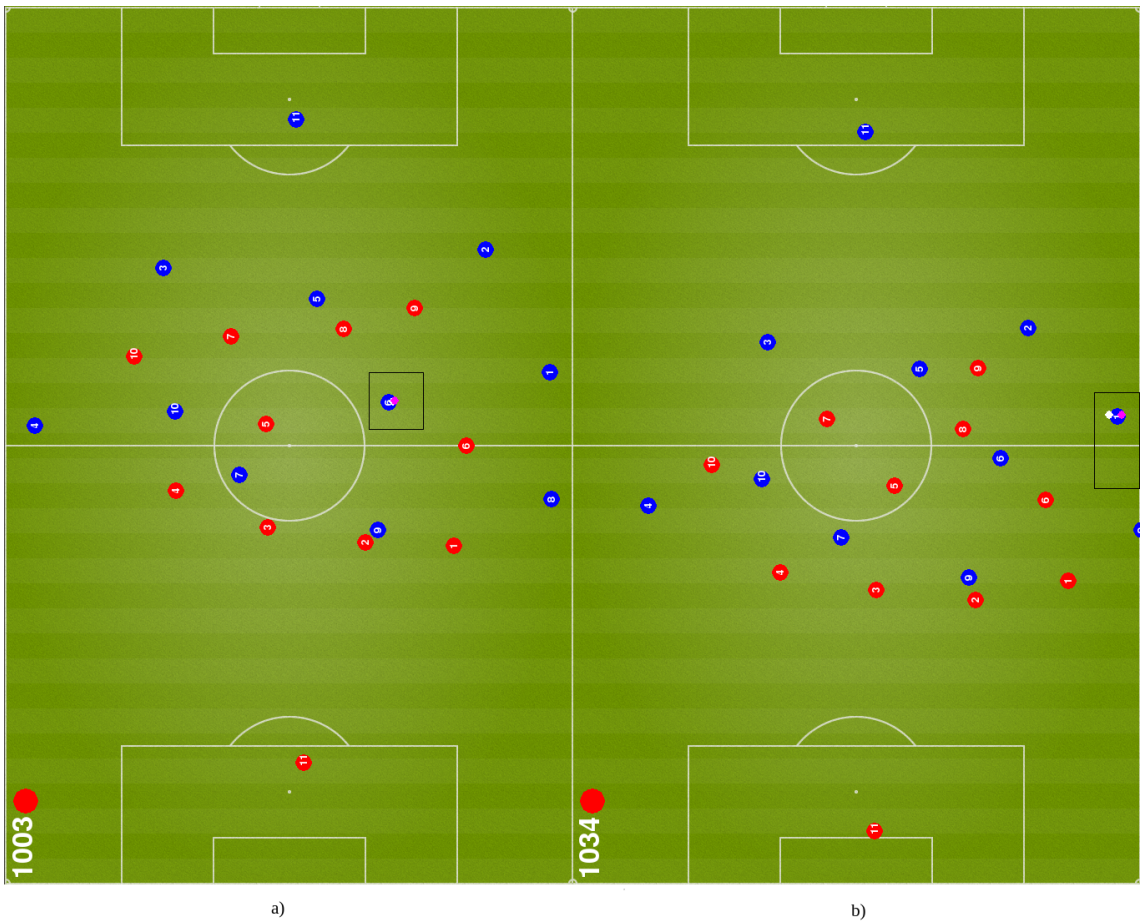

Figura 4.14: Processamento do Esquema Dinâmico da Fig. 4.12, contendo um instante em que o momento do desenho do modelo de jogo é validado: a) momento 3 e b) momento 4.

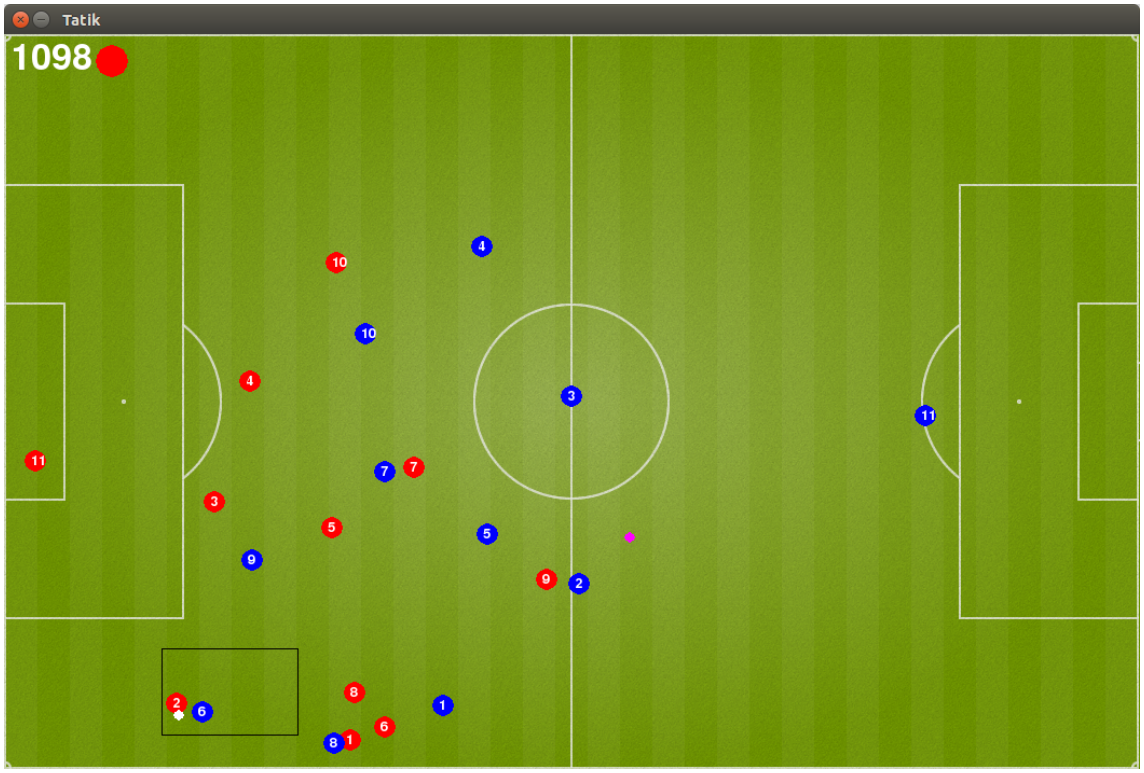

Figura 4.15: Processamento do Esquema Dinâmico da Fig. 4.12, contendo um instante em que o momento 5 do desenho do modelo de jogo é validado.

trocas de bola (*groundtruth*).

A Tab. 4.1 resume os resultados obtidos ao variar o valor do raio de ação no algoritmo de deteção de trocas de bola. É visível que para raios de ação mais pequenos o algoritmo não tem o desempenho desejado, falhando na deteção de um número significativo de trocas de bola. À medida que o valor do raio de ação é aumentado a taxa de acertos aumenta e o número de trocas de bola não detetadas (Falsos Negativos) diminui. Para a gama de raios de ação de 1.4 a 1.7 metros há uma crescente deteção de trocas de bola e uma redução dos falsos positivos e negativos já para valores superiores a 1.7 metros o aumento da trocas de bola é refletido num aumento de falsos positivos. Assim sendo o valor de 1.7 metros parece ser o valor mais adequado de modo a obter o maior número de passes acertados com o menor erro (este resultado requer uma validação através da execução de um conjunto mais extensível de testes). Em termos de performance do algoritmo, visto ter sido obtido uma média de 3 segun-

Raio de Ação (m)	Trocas Detetadas	Falso Positivo	Falso Negativo	Precisão
1.4	45	6	13	75.0%
1.5	49	7	10	80.8%
1.6	51	8	9	82.7%
1.7	53	6	5	90.4%
1.8	54	7	5	90.4%
1.9	54	7	5	90.4%
2.0	57	10	5	90.4%

Tabela 4.1: Tabela comparativa dos resultados obtidos na deteção de trocas de bola, ao variar o valor do raio de ação.

dos na deteção das trocas de bola para um exemplo de 2 minutos, faz com que a sua utilização em tempo real seja possível.

4.4 Mapas de Calor

Nesta Secção iremos apresentar alguns exemplos dos mapas de calor que podem ser criados. Na Fig. 4.16 encontra-se ilustrado o mapa de calor da equipa A com uma resolução de 250 por 150, no qual o valor de dispersão varia entre 0 e 2. É visível que no mapa de calor sem dispersão, não é tão percetível os locais onde houve mais ação, enquanto que nos mapas de calor com dispersão esses locais já são mais percetíveis.

Na Fig. 4.17 é ilustrada a capacidade de obter mapas de calor com diferentes resoluções, podendo ser alterada a granularidade conforme desejado. É também possível obter o mapa de calor de um só jogador (Fig. 4.18), ou ainda um mapa de calor com um elemento da equipa A e outro elemento da equipa B (Fig. 4.19). Com este conjunto de opções é possível criar uma grande variedade de mapas de calor de forma a analisar as equipas, jogadores ou grupo de jogadores quanto à sua ocupação de espaços no campo.

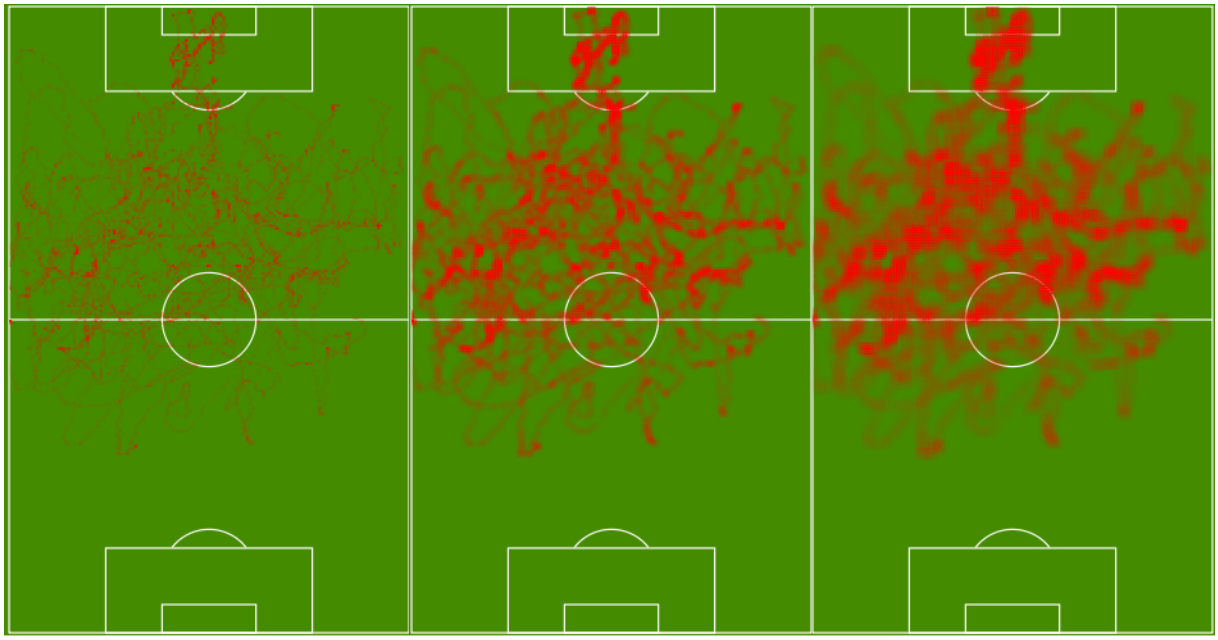

Figura 4.16: Mapa de Calor da equipa A: a) fator de dispersão igual a 0, b) fator de dispersão igual a 1 e c) fator de dispersão igual a 2.

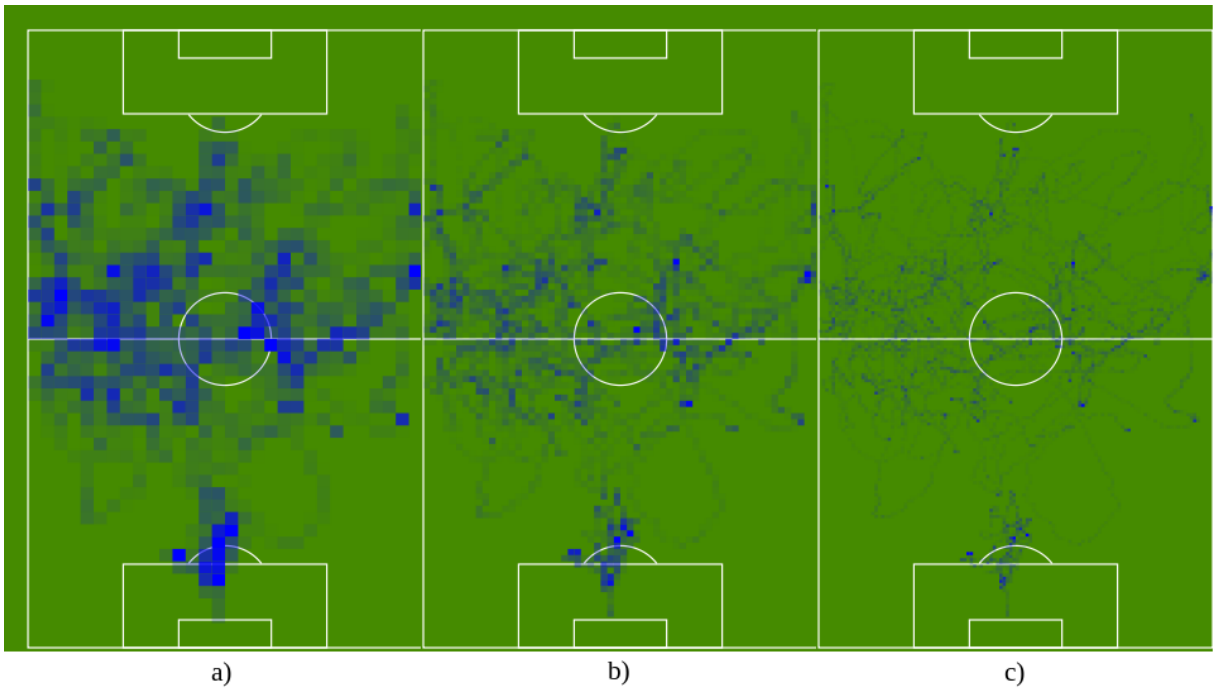

Figura 4.17: Mapa de Calor da equipa A (dispersão nula): a) resolução de 50x30, b) resolução de 100x60 e c) resolução de 200x120.

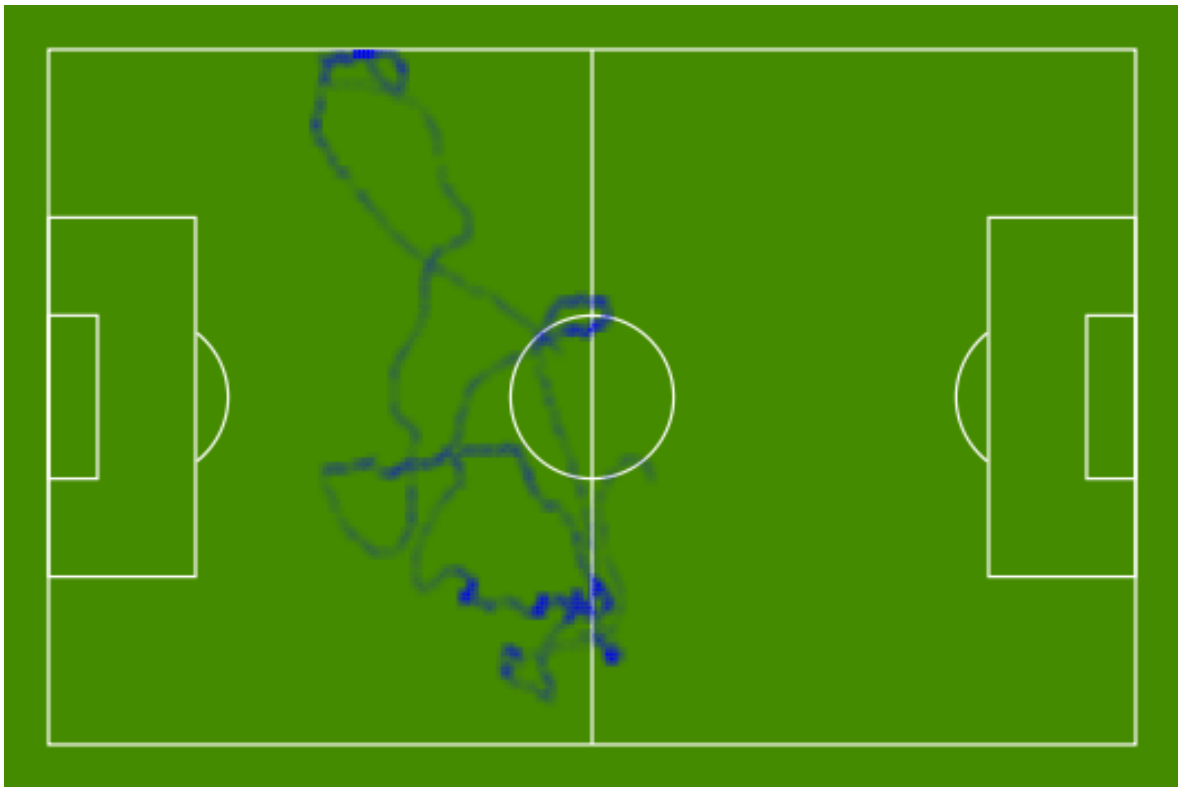

Figura 4.18: Mapa de Calor do jogador 7 da equipa B, com um fator de dispersão igual a 1.

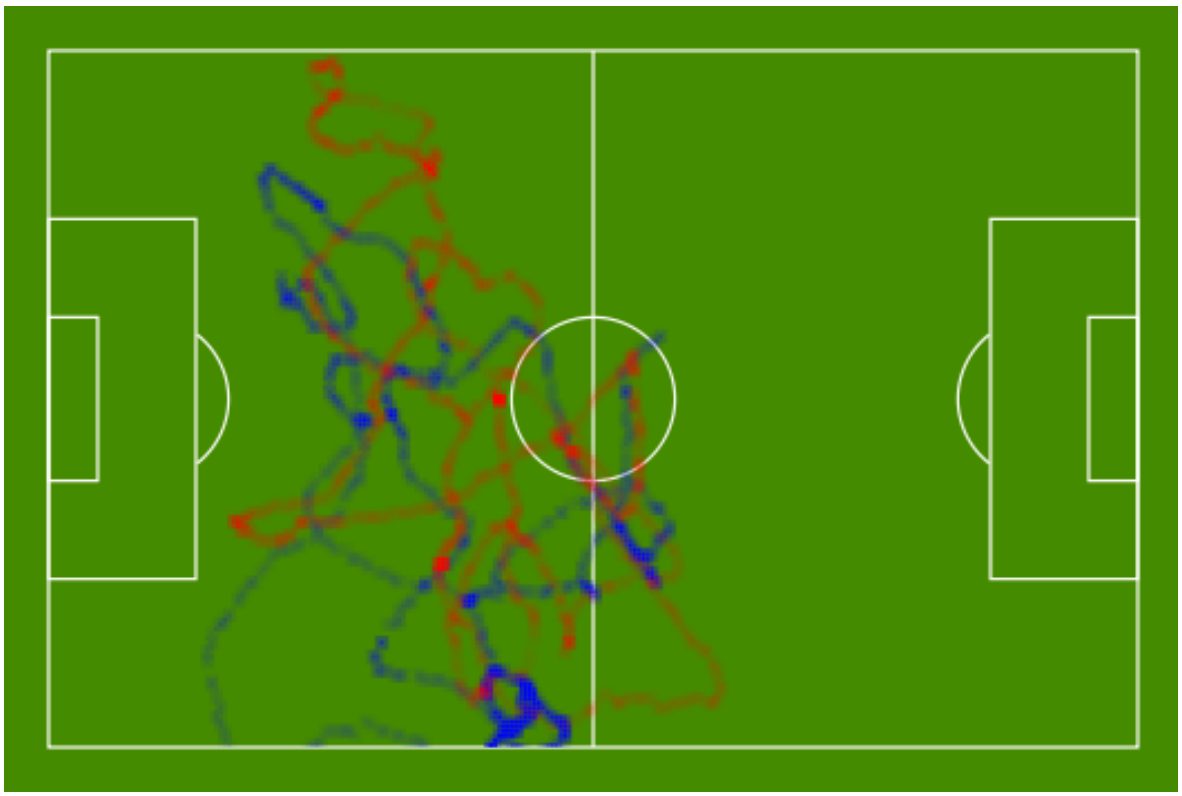

Figura 4.19: Mapa de Calor do jogador 8 da equipa B (Azul) e do jogador 5 da equipa A (Vermelho), com um fator de dispersão igual a 1.

5

Conclusões e trabalho futuro

No âmbito do desenvolvido podemos destacar as algumas das principais contribuições. Em primeiro a implementação de uma (i) Linguagem de Programação Visual (LPV) para a deteção dos modelos de jogo. Esta LPV permite aos utilizadores, sem conhecimentos de programação, fazer esquemas numa aplicação web que posteriormente são convertidos em código que possibilita uma análise automática a um jogo de futebol, fornecidos os dados do sistema de *tracking*. A ferramenta analisa as situações pretendidas para cada *frame* ou conjunto de *frames*, dando oportunidade ao treinador de analisar detalhadamente um grande número de aspetos do jogo, que de outro modo iriam requerer muito tempo e trabalho manual. Esta ferramenta está integrada com outras que possibilitam o envio de alertas durante o jogo, permitindo assim

ao treinador fazer ajustes à sua equipa em tempo real. Outras das principais contribuições foram o desenvolvimento de um (ii) algoritmo capaz de detetar as trocas de bola existentes num jogo e (iii) a implementação de uma biblioteca em Python para a criação de mapas de calor, de passes e de perdas de bola. O algoritmo de deteção de trocas de bola, juntamente com os mapas de passes e perdas de bola, permitem aos treinadores e equipa técnica saber quantos passes foram efetuados com sucesso assim como as perdas de bola. Esta informação pode ser filtrada por equipa e jogador, facilitando a análise de dados por parte dos treinadores e equipas técnicas. Por outro lado, a análise dos mapas de calor permite saber os locais onde os jogadores e bola se concentram mais, sendo possível filtrar esta informação por jogador, equipa e intervalo de tempo.

É também importante referir que estas ferramentas servem não só para analisar a própria equipa como para estudar a equipa adversária. Isto dá às equipas a oportunidade de irem melhor preparados para os jogos, ou no caso, de ser usado durante o jogo, de ser notificado no banco podendo assim fazer ajustes à equipa no momento.

Sendo que os principais objetivos a que nos propusemos foram alcançados, há contudo uma série de testes e exaustivos que necessitam de ser feitos até à obtenção de um produto final. Assim, como trabalho futuro, pretendemos completar a integração do sistema no SIGF do FootData-Pro e realizar um conjunto de testes exaustivos de validação das aplicações. Acresce a implementação de aplicações e funcionalidades que estão constantemente a surgir com base nos pareceres do consultor do projeto.

5.1 Lista de publicações

No que diz respeito à disseminação do trabalho desenvolvido, foram publicados quatro artigos em conferências:

- Belguinha, A., Cardoso, P.J.S. Rodrigues, P., Paciência, D., Rodrigues, J.M.F. (2014) A Visual Programming Language for Soccer, In Proc. for 9th Int. Conf. on Soft-

ware Paradigm Trends, Vienna, Austria, 21-31 Ago, pp. 121-127.

- Rodrigues, J., Cardoso, P.J.S., Vilas, T., Silva, B., Rodrigues, P., Belguinha, A., and Gomes, C. (2014). A computer vision based web application for tracking soccer players. In Universal Access in Human-Computer Interaction. Design and Development Methods for Universal Access, pages 450–462. Springer.

- Belguinha, A., Rodrigues, J.M.F., Cardoso, P. (2013) Processing sports acquired information from a tracking system, In Proc. 19th Portuguese Conf. on Pattern Recognition, Lisboa, Portugal, 1 Nov., 2 pp

- Rodrigues, P., Belguinha, A., Gomes, C., Cardoso, P., Vilas, T., Mestre, R., Rodrigues, J. M. F. (2013). Open Source Technologies Involved in Constructing a Web-Based Football Information System. In Á. Rocha, A. M. Correia, T. Wilson, & K. A. Stroetmann (Eds.), Advances in Information Systems and Technologies SE - 66, vol. 206, pp. 715–723.

Referências

Alchin, M., Kaplan-Moss, J., & Vilches, G. (2009). *Pro Django*. Springer.

ASPOGAMO (2014). http://www9.cs.tum.edu/projects/aspogamo/. Acedido em Agosto de 2014.

Beetz, M., von Hoyningen-Huene, N., Kirchlechner, B., Gedikli, S., Siles, F., Durus, M., & Lames, M. (2009). ASpoGAMo: Automated Sports Game Analysis Models. *International Journal of Computer Science in Sport*, 8(1).

Blockly (2014). https://code.google.com/p/blockly/. Acedido em Agosto de 2014.

Castelo, J. (2014). Observação de jogo e tecnologias digitais de análise. http://www.jorgecastelo.com/Academy/. Acedido em Agosto de 2014.

Cecco, R. (2011). *Supercharged JavaScript Graphics*. O'Reilly Media.

CoachFX (2014). www.coachfx.com. Acedido em Agosto de 2014.

Dabney, J. B. & Harman, T. L. (2001). *Mastering Simulink 4*. Prentice Hall PTR.

Dobesova, Z. (2011). Visual programming language in geographic information systems. In *Proc. 2nd Int. Conf. on Applied Informatics and Computing Theory*, AICT'11, pp. 276–280, Stevens Point, Wisconsin, USA. World Scientific and Engineering Academy and Society (WSEAS).

EaselJS (2014). EaselJS. www.createjs.com/#!/EaselJS. Acedido em Agosto de 2014.

EasyAnimation (2014). Easy Animation. www.de.easy-animation.com. Acedido em Agosto de 2014.

eSpor (2014). eSpor Software Soccer eAssist. www.espor.com. Acedido em Agosto de 2014.

Fabric (2014). Fabric.js. www.fabricjs.com. Acedido em Agosto de 2014.

Flash, A. (2014). http://www.w3.org/TR/WCAG20-TECHS/flash.html. Acedido em Agosto de 2014.

Flask (2014). http://flask.pocoo.org/. Acedido em Agosto de 2014.

Hawkes, R. (2011). *Foundation HTML5 Canvas: For Games and Entertainment*. friendsofED, 1st edition.

Jakus, G., Jekovec, M., Tomažič, S., & Sodnik, J. (2010). New technologies for web development. *Elektrotehniški vestnik*, 77(5):273–280.

JSON (2014). http://www.json.org/. Acedido em Agosto de 2014.

K-PlayMaker (2014). K-PlayMaker. www.kplaymaker.com. Acedido em Agosto de 2014.

Kaipiainen, S. & Paksula, M. (2010). From SVG to Canvas and back. In *8th Int. Conf. on Scalable Vector Graphics*, pp. 111–120.

Kellis, E. & Katis, A. (2007). Biomechanical characteristics and determinants of instep soccer kick. *Journal of Sports Science and Medicine*, 6:154–165.

KineticJS (2014). KineticJS. www.kineticjs.com. Acedido em Agosto de 2014.

Ktechlab (2014). https://github.com/ktechlab/ktechlab/wiki. Acedido em Agosto de 2014.

Lucanin, D. & Fabek, I. (2011). A visual programming language for drawing and executing flowcharts. In *MIPRO, 2011 Proc. of the 34th Int. Convention*, pp. 1679–1684. IEEE.

Marchiori, E. J., Del Blanco, Á., Torrente, J., Martinez-Ortiz, I., & Fernández-Manjón, B. (2011). A visual language for the creation of narrative educational games. *Journal of Visual Languages & Computing*, 22(6):443–452.

MatchViewer, P. (2014). http://www.prozonesports.com/product/matchviewer/. Acedido em Agosto de 2014.

MongoDB (2014). http://www.mongodb.com/. Acedido em Agosto de 2014.

Nikoukhah, R. & Steer, S. (1996). Scicos-a dynamic system builder and simulator. In *Proc. of the 1996 IEEE Int. Symposium on Computer-Aided Control System Design*, pp. 430–435. IEEE.

Paperjs (2014). Paper.js. www.paperjs.org. Acedido em Agosto de 2014.

Paulo Sousa, J. G. & Garganta, R. (2003). Estatuto posicional, força explosiva dos membros inferiores e velocidade imprimida à bola no remate em futebol. *Revista Portuguesa de Ciências do Desporto*, vol. 3(nº 3):27–35.

Python (2014). https://www.python.org/. Acedido em Agosto de 2014.

Raphaël (2014). http://raphaeljs.com/. Acedido em Agosto de 2014.

Resnick, M., Maloney, J., Monroy-Hernández, A., Rusk, N., Eastmond, E., Brennan, K., Millner, A., Rosenbaum, E., Silver, J., Silverman, B., et al. (2009). Scratch: programming for all. *Communications of the ACM*, 52(11):60–67.

Rodrigues, J., Cardoso, P. J., Vilas, T., Silva, B., Rodrigues, P., Belguinha, A., & Gomes, C. (2014). A computer vision based web application for tracking soccer players. In *Universal Access in Human-Computer Interaction. Design and Development Methods for Universal Access*, pp. 450–462. Springer.

Rodrigues, P. (2014). Ferramentas web para análise da performance de uma equipa de futebol. Dissertação de Mestrado, Instituto Superior de Engenharia da Universidade do Algarve.

Rowell, E. (2011). *HTML5 Canvas Cookbook*. Packt Publishing.

Scicos (2014). http://www.scicos.org/. Acedido em Agosto de 2014.

Scratch (2014). http://scratch.mit.edu/. Acedido em Agosto de 2014.

Sebe, N. & Lew, M. S. (2003). *Robust computer vision: Theory and applications*. Kluwer Academic Dordrecht.

Silverlight, M. (2014). http://www.microsoft.com/silverlight/. Acedido em Agosto de 2014.

Simulink (2014). http://www.mathworks.com/products/simulink/. Acedido em Agosto de 2014.

SnapSVG (2014). http://snapsvg.io/. Acedido em Agosto de 2014.

Sousa, T. (2012). Rastreamento de jogadores de futebol tendo em vista a análise de modelos de jogo. Dissertação de Mestrado, Instituto Superior de Engenharia da Universidade do Algarve.

Sucan, M. (2010). SVG or Canvas? Choosing between the two. https://dev.opera.com/articles/svg-or-canvas-choose/.

SVGJS (2014). http://www.svgjs.com/. Acedido em Agosto de 2014.

TacticalPad (2014). TacticalPad. www.tacticalpad.com. Acedido em Agosto de 2014.

Tactics Manager Software (2014). www.soccertutor.com/tacticsmanager/. Acedido em Agosto de 2014.

Tekli, G., Chbeir, R., & Fayolle, J. (2013). A visual programming language for xml manipulation. *Journal of Visual Languages & Computing*, 24(2):110–135.

Yu, X., Xu, C., Leong, H. W., Tian, Q., Tang, Q., & Wan, K. W. (2003). Trajectory-based ball detection and tracking with applications to semantic analysis of broadcast soccer video. In *Proc. of the 11th ACM Int. Conf. on Multimedia*, MULTIMEDIA '03, pp. 11–20, New York, NY, USA. ACM.