UNIVERSIDADE DO ALGARVE

FERRAMENTAS WEB PARA ANÁLISE DA PERFORMANCE DE UMA EQUIPA DE FUTEBOL

PEDRO GIL DE ALMEIDA RODRIGUES

Dissertação

Mestrado em Engenharia Elétrica e Eletrónica

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

2014

FERRAMENTAS WEB PARA ANÁLISE DA PERFORMANCE DE UMA EQUIPA DE 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.

Resumo

Existem duas palavras de ordem no futebol de alta competição: exigência e resultados. Perante tal cenário e com o grande avanço das novas tecnologias, é importante pensar no que se poderia alcançar e que novas metas se poderiam definir no mundo do futebol, utilizando ferramentas tecnológicas, como solução para as mais variadas atividades neste desporto, nomeadamente, na fase de preparação, no decorrer, ou na fase posterior aos jogos. Esta dissertação foca uma série de aplicações *web* que visam diminuir o impacto de alguns problemas na gestão de uma equipa de futebol, auxiliando as equipas técnicas na representação de jogadas, esquemas táticos e modelos de jogo, na visualização e edição de vídeos com possibilidade de definição e filtragem de eventos, na criação semiautomática de palestras e apresentações aos jogadores e, por fim, na correção de dados (jogadores e bola) provenientes de um sistema de *tracking* que funciona sobre vídeos e em tempo real num estádio de futebol. Todas estas ferramentas estão incluídas num Sistema Integrado de Gestão para o Futebol (SIGF) que está a ser desenvolvido no projeto QREN I&DT - FootData.

Palavras-chave: Ferramentas Web, Interfaces Web, Aplicações Web, Tecnologias Web, HTML5, JavaScript, Futebol.

Abstract

There are two crutial words in football at the highest level: exigency and results. Before such demands and with the great development in the new technologies, it is important to think of what could be achieved and which goals could be settled in the football world by using technological tools as a solution for many activities in this sport, such as, in the pre-match analysis, during a match, and in the post-match analysis. This thesis presents a series of web applications with the objective of decreasing the negative impact of some problems that can arise from the football team management, aiding the technical staff in the movement representation, tactical schemes and game models, in the video visualization and edition with the possibility of defining and filtering events, in the creation of semiautomatic presentations for team reunions and, finally, in the correction of data (players and ball) provided by a tracking system that can be applied in videos and in real time in a football stadium. 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: Web Tools, Web Interfaces, Web Applications, Web Technologies, HTML5, JavaScript, Football.

Aos meus pais,

pelo o que me ensinaram e por todo o apoio que me deram.

À minha restante família e amigos,

pela força e motivação nos momentos menos bons.

Agradecimentos

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

Devo ainda um agradecimento aos meus colegas de projeto Carlos Gomes, Tiago Sousa e António Belguinha por um ano de trabalho bastante gratificante a nível pessoal e profissional.

Deixo um obrigado a todas as pessoas da empresa INESTING, pelo apoio no *design* das aplicações constantes neste trabalho e ainda ao consultor do projeto Domingos Paciência pela informação e recursos prestados.

Por fim, mas não menos importante, um agradecimento a todos os meus colegas de curso e do laboratório de projeto pelos bons momentos passados no decorrer do meu percurso académico.

Conteúdo

Lista de Tabelas

Lista de Figuras

1

Introdução

O futebol é um desporto onde todas as entidades envolvidas (e.g., 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, 2011).

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 (jo-

gadores e treinadores) 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 com a maior brevidade possível. Outro ponto crítico, é o facto de que as equipas de hoje, na sua maioria, têm jogadores e *staff* de diversas nacionalidades nos seus quadros, o que pode ser um obstáculo à comunicação e ao correto entendimento entre todas as partes, pelo que, este sistema tecnológico, com todos os seus 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 plataforma 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 principais objetivos: i) O projeto e o desenvolvimento de quatro ferramentas *web*, de acordo com os requisitos estabelecidos no projeto FootData; ii) O estudo e a escolha das melhores e mais recentes tecnologias para as implementar e iii) A integração das referidas ferramentas no SI do projeto FootData.

Em termos mais específicos, para cada uma das ferramentas a projetar e implementar, podemos referir:

a) Editor de Campo - fornece ao treinador e restante equipa técnica uma plataforma interativa onde é possível desenhar não só movimentos para um jogo-treino como também jogadas mais complexas. Estes desenhos permitem a definição de mo-

delos de jogo que por sua vez serão comparados em tempo real com os dados obtidos a partir de um sistema automático de *tracking* dos jogadores em campo ou em vídeo. Esta mesma ferramenta pode também ser utilizada antes e durante o jogo para o treinador exemplificar aos seus jogadores, jogadas que ele pretende que estes executem, substituindo o tradicional quadro (*tactical board*). De uma forma mais detalhada, os objetivos do Editor de Campo são os seguintes:

- Permitir criar jogadas ou movimentos numa plataforma de desenho interativa focada no futebol;
- Permitir criar animações a partir dos desenhos efetuados;
- Transferir e codificar essas animações para a base de dados do projeto FootData;
- Permitir apresentar jogadas predefinidas aos jogadores, mostrando possíveis movimentações que estes terão de fazer em campo.

b) Editor de Vídeo - é uma ferramenta onde é possível fazer a edição e manipulação de vídeos de futebol, marcação de eventos (e.g., os passes efetuados pelo jogador "X", numa zona "Y" do campo, no decorrer do processo ofensivo) e ainda permitir inserir texto e outras formas nas *frames* do vídeo como sejam setas, círculos, entre outras. Mais detalhadamente, os objetivos são os seguintes:

- Permitir o carregamento e a visualização de jogos;
- Permitir inserir diferentes barras de ferramentas, por exemplo barra de navegação (*play*, *stop*, *pause*, *forward*, *rewind*, entre outros);
- Permitir a visualização em ecrã completo;
- Permitir marcar manualmente e reproduzir eventos no vídeo;
- Permitir partir manualmente as ações do jogo em categorias personalizadas;

- Gerar videoclipes personalizados, onde é permitida numa determinada *frame* a manipulação das posições dos jogadores, inserir texto, setas e outros elementos.

c) Editor de *Tracking* - permite dar ao utilizador a possibilidade de corrigir, caso seja necessário, o sistema de *tracking* automático (deteção e classificação automática de jogadores e bola em campo, *on-site* ou em vídeo) (Sousa, 2012), i.e., permite fazer correções a possíveis falhas na classificação de jogadores. De uma forma mais detalhada, os objetivos são:

- Criar uma plataforma de correção ao sistema de *tracking* do projeto FootData;
- Visualizar em tempo real as filmagens que estão a ser feitas *on-site* e processadas pelo sistema de *tracking*;
- Permitir fazer correções aos números dos jogadores e/ou às equipas nos dados provenientes do *tracking* (*on-site* e vídeo);
- Permitir marcar alertas quando o sistema está a funcionar em tempo real no campo.

d) Editor de Apresentações - permite criar apresentações semiautomáticas para as palestras da equipa técnica aos jogadores. Mais detalhadamente, os objetivos são os seguintes:

- Permitir aceder aos documentos do Editor de Campo e de Vídeo;
- Editar imagens, onde é permitida a manipulação das posições dos jogadores, inserir texto, setas e outros elementos;
- Mostrar informação vinda de uma base de dados (texto ou gráficos);
- Interligar a aplicação com o sistema de projeção eBeam (Imagina, 2013).

A contribuição principal desta dissertação é o projeto e o desenvolvimento de um conjunto de ferramentas *web* para ajudar na análise da *performance* de uma equipa de

futebol, sendo a mais-valia principal o Editor de Campo, onde as jogadas ou os modelos de jogo idealizados pelo treinador, podem ser descritos graficamente em ambiente *web*, de forma a serem automaticamente convertidos para código, para posterior análise em tempo real do desempenho técnico-tático dos jogadores ou equipa em campo.

1.3 Perspectiva 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 abordados dois tipos de aplicação: nativa e *web*, e 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 ferramentas *web* individualmente e de forma detalhada, complementado com comparações com alguns produtos já existentes para o mesmo efeito. No mesmo capítulo, apresentamos ainda a implementação de algumas das ferramentas já em versão de produção com o *design* final. Por fim, são apresentadas, no Capítulo 4, as conclusões, considerações finais e as metas para trabalhos futuros.

2

Estado da arte e conceitos gerais

Neste capítulo será feita uma breve análise a dois tipos de aplicações: nativa e *web*. Com base nestes dois conceitos, serão descritos de forma sucinta alguns dos principais produtos do mundo do futebol. Estes produtos, baseados em tecnologias de informação, apresentam funcionalidades semelhantes àquelas que se implementaram nesta dissertação.

Com vista à compreensão das ferramentas *web* que foram desenvolvidas, abordar-se-ão aspetos como a arquitetura em *tiers* e *layers* e as principais tecnologias utilizadas nas aplicações *web*, explicando-se ainda os seus objetivos e exemplos de *design patterns* e *frameworks*.

Por fim, dar-se-ão a conhecer alguns tópicos sobre aplicações *web* com uma forte

componente gráfica e vídeo.

2.1 Aplicações nativas e aplicações *web*

Antes de começarmos a identificar os produtos existentes no mercado é importante definir o que é uma aplicação nativa e o que é uma aplicação *web*, bem como quais são os fatores que requerem uma ponderação prévia com o objetivo de se determinar qual é a melhor solução para um dado problema.

Nesse sentido, vários parâmetros podem ser considerados, tais como: objetivos e conteúdos da aplicação, funcionalidades que se pretendem implementar, público-alvo, plataformas/sistemas-alvo, entre outros. De facto, hoje em dia, são constantes os debates sobre qual a melhor solução quando se pretende projetar e implementar uma aplicação (Charland and Leroux, 2011), pois tanto as aplicações nativas como as aplicações *web* possuem os seus pontos fortes e os seus pontos fracos.

Define-se uma aplicação nativa, como uma aplicação que é criada para ser executada nativamente no dispositivo do utilizador, sendo portanto, projetada de acordo com a plataforma ou sistema operativo onde vai ser executada (e.g., *Microsoft Windows*, *Linux*, *OS X*, *iOS*, *Android*, entre outros). Como é de prever, para correr este tipo de aplicação é necessário haver a sua prévia instalação no dispositivo do utilizador. Por outro lado, a aplicação *web* define-se como uma aplicação geralmente projetada para ser compatível com todos os *browsers*. Ao contrário da aplicação nativa, a aplicação *web* não necessita de qualquer instalação, bastando apenas a utilização de um *browser* num dispositivo com ligação à *Internet* (Al-Fedaghi, 2011).

Vejamos alguns dos pontos fortes e fracos de ambas:

a) Experiência do utilizador e desempenho - este é um dos pontos mais importantes quando se pretende desenvolver um produto para o mercado. Parâmetros como *performance*, desempenho e estabilidade são geralmente piores nas aplicações *web*, dado que as aplicações nativas são pensadas, desenhadas e desenvolvidas com

base nos requisitos de cada dispositivo ou sistema-alvo. Além do mais, nas aplicações *web*, os programadores devem ter em consideração o modo como a aplicação é visualizada nos diversos *browsers* enquanto que as aplicações nativas ficam com uma interface mais rica e intuitiva para os utilizadores, visto que seguem as convenções e normas das plataformas de destino (Anacleto, 2012). Silva (2012) refere, contudo que, os custos de desenvolvimento multiplataforma são elevados, obrigando ainda a uma revisão integral das interfaces. Charland and Leroux (2011) apresentam uma discussão sobre vários aspetos inerentes à experiência do utilizador nestes dois tipos de aplicação, no que diz respeito ao contexto e implementação.

b) Distribuição - os dois tipos de aplicação diferem bastante no que diz respeito a este ponto. As aplicações *web* demonstram grande flexibilidade nesta área já que podem ser distribuídas pelo *browser* de forma eletrónica. O ato de empacotar, inscrever nos CDs ou DVDs, imprimir manuais e o transporte sai mais caro. Eletrões enviados pela *Internet* saiem mais barato (Fowler and Stanwick, 2004). Como já foi referido anteriormente, também não é por norma necessária a instalação de qualquer *software* no caso das aplicações *web*, sendo esta outra vantagem deste tipo de aplicações.

c) Segurança - este é um ponto que gera alguma preocupação sobretudo para os programadores, administradores e utilizadores de aplicações *web*. Não só é necessário existir uma proteção eficaz na parte acessível ao utilizador como também é importante garantir a segurança da informação no lado inacessível ao utilizador comum, nomeadamente na parte dos servidores e bases de dados. Visto que a maioria das atuais aplicações nativas não possui uma centralização dos dados como as aplicações *web*, e muitas nem requerem que exista uma ligação à rede para funcionarem, a segurança pode ser vista como um ponto favorável para as aplicações nativas (Fowler and Stanwick, 2004; Cross, 2007).

d) Manutenção e atualização - após o lançamento de uma aplicação nativa, a sua atualização e manutenção pode ser dispendiosa. Por exemplo, para uma empresa com centenas de empregados que adquire uma aplicação nativa e a tenha de atualizar no

Tabela 2.1: Comparação entre aplicações nativas e *web*.

Parâmetro	**Aplicação Nativa**	**Aplicação *Web***
Experiência do utilizador	✓	-
Distribuição	-	✓
Segurança	✓	-
Manutenção e atualização	-	✓

computador de cada funcionário provoca mais constrangimentos do que uma aplicação *web* que é atualizada de forma centralizada pelos seus desenvolvedores/administradores, sem ser necessário que os utilizadores intervenham para o efeito (Fowler and Stanwick, 2004).

A Tab. 2.1 resume algumas das conclusões resultantes da comparação entre aplicações nativas e *web*, destacando-se qual a melhor consoante os parâmetros abordados. Assim, através da análise da tabela e do exposto, pode-se concluir que o desenvolvimento de uma aplicação *web* será vantajosa em termos de distribuição, manutenção e atualização e vai ao encontro dos requisitos do projeto FootData.

De seguida vamos analisar as diferentes aplicações que estão relacionadas com o tema desta dissertação, classificando-as também quanto ao seu tipo.

2.2 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 alguns produtos semelhantes ao que se pretende fazer no Editor de Campo, dos quais se destacam: o *K-PlayMaker* (K-PlayMaker, 2013), que 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 a possibilidade de inserção de variantes de um campo de futebol (completo, metade atacante, metade defensiva, entre outras), vem sob o formato de aplicação *web* e permite ainda a gravação e reprodução de jogadas. O *TacticalPad* (TacticalPad, 2013), é uma plataforma que permite estabelecer posicionamentos e fazer simulações de jogadas de futebol em 2D ou em 3D. O *TacticalPad* vem sob o formato de aplicação nativa para *Microsoft Windows*, *iOS* e *Android*. O *CoachFX* (CoachFX, 2013) contém um vasto leque de produtos para desenho de jogadas e sessões de treino. Vem no formato de aplicação nativa para *Microsoft Windows* e *Android*. Outro dos produtos existentes é o *Easy Animation* (EasyAnimation, 2013), que permite fazer o desenho de jogadas, disponibilizando vários elementos para desenho de sessões de treino, sendo complementadas com uma componente de animação. Vem no formato de aplicação nativa para *Microsoft Windows*. O *eSpor Software Soccer eAssist* (eSpor, 2013), 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. Vem no formato de aplicação nativa para *Microsoft Windows*. Por último, o *Tactics Manager Software* (Tactics Manager Software, 2013) é uma aplicação nativa para *Microsoft Windows* onde é possivel fazer a definição de esquemas de futebol estáticos, de movimentos e de zonas em 2D e 3D.

Relativamente a produtos semelhantes ao Editor de Vídeo, há a salientar: o *K-Studio* (K-Studio, 2013), que é um dos produtos da aplicação *Kizanaro*, no formato de aplicação nativa para *Microsoft Windows*, virado para a análise de vídeos e que permite categorizar determinadas ações e eventos. O *Prozone Playback* (Playback, 2013), no formato de aplicação nativa para *Microsoft Windows*, permite a análise de vídeos em tempo real no contexto das *performances* coletivas ou individuais, auxiliando o treinador e atletas na identificação de incidentes específicos. O *Sportstec SportsCode* (SportsCode, 2013) é uma aplicação nativa exclusiva para *OS X*, focada na edição e manipulação de vídeos com a possibilidade de definir eventos.

Relativamente aos produtos semelhantes ao Editor de *Tracking*, tanto quanto é do nosso conhecimento e do consultor do projeto, o treinador Domingos Paciência, apenas foi possível identificar a aplicação *Prozone MatchInsight* (MatchInsight, 2013). Esta aplicação é uma ferramenta cujo objetivo principal é a análise de *performances* a partir de imagens provenientes de uma câmara. Permite aos treinadores e restante equipa técnica codificar aspetos específicos relativos à *performance* e produzir informação técnica e tática detalhada em tempo real no panorama coletivo ou individual.

Por último, não existem no mercado produtos semelhantes ao que se pretende implementar para o Editor de Apresentações. Atualmente, a maioria dos treinadores, quando pretende efetuar palestras aos seus jogadores, utiliza o *Microsoft Powerpoint* (PowerPoint, 2013), que é um programa conhecido pelos utilizadores do pacote *Office* da *Microsoft*, para criação de apresentações. É importante reforçar que, no contexto do futebol, não existe nenhuma ferramenta específica e/ou integrada nas aplicações referidas nesta secção para criar as apresentações.

Sendo que a aplicação a ser desenvolvida será *web*, devido às condicionantes impostas pelo consórcio do projeto, apoiadas pela análise feita na Secção 2.1, será de seguida feita uma breve abordagem à arquitetura em *tiers* e *layers* e, posteriormente, serão analisadas as tecnologias de base para a criação de uma aplicação *web*.

2.3 Arquitetura das aplicações *web* baseada em *tiers* e *layers*

Por norma, o desenvolvimento de uma aplicação envolve, numa fase inicial, um estudo da arquitetura, isto é, a ponderação acerca de um molde que mantenha o projeto dentro de certos limites, tendo em vista aspetos como a robustez, a escalabilidade e a flexibilidade (Thakur, 2008). É de extrema importância ter uma vista de alto nível do nosso projeto em termos físicos e em termos lógicos (Bell, 2005; Thakur, 2008). Por conseguinte, surgem os conceitos de *tier* e *layer*. Numa arquitetura baseada em *tiers*, existe

uma separação física do código, onde por exemplo, podemos ter o código distribuído e a ser executado em diversas unidades (e.g., servidor A, servidor B). Nesse sentido, uma arquitetura com três *tiers* assentará em três nós de processamento principais. Por outro lado, a separação em *layers* significa que existe uma divisão lógica do código, ou seja, podemos distribuir os ficheiros de código por diversas pastas para facilitar a sua manutenção, contudo, isso não implica que estejam a ser executados em vários nós de processamento (Thakur, 2008). De um modo geral, podemos ter uma aplicação cuja arquitetura está definida em vários *tiers* (*n-tiers*) e *layers*, consoante a necessidade e complexidade do projeto.

A estrutura típica de uma aplicação *web*, em termos lógicos (ver Figura 1, constante em (Microsoft Developer Network, 2014), Capítulo 21, página 2), baseia-se numa arquitetura de três *layers*, nomeadamente, de apresentação (*presentation layer*), de negócio (*business layer*) e de dados (*data layer*). O *layer* de apresentação engloba todos os componentes que dizem respeito à interface do utilizador e com o qual o mesmo interage. É por vezes referido como o *front-end*. O *layer* de negócio encerra toda a lógica da aplicação e estabelece a forma como a informação é processada, fazendo uma ponte entre os *layers* de apresentação e de dados. É muitas vezes denominado por *middleware*. Por fim, o *layer* de dados gere o acesso a bases de dados ou sistema de ficheiros e é também referido como *back-end*.

Tendo este conceito de divisão física e lógica em mente, torna-se pois importante, ter em conta os recursos a utilizar tanto no *front-end* como no *back-end* (MacCaw, 2011). Na Secção 2.4 ir-se-á identificar e analisar algumas das tecnologias usadas no *layer* de apresentação. A Secção 2.5 conjuga diversos aspetos do *layer* de negócio. Ainda respeitante às aplicações *web*, serão abordados, na Secção 2.6, *design patterns* e algumas *frameworks* a ter em conta.

2.4 *Front-end* das aplicações *web*

Como Jakus et al. (2010) referem, o conceito de *World Wide Web* é inseparável da *Hyper Text Markup Language* (HTML) - a linguagem para descrever páginas *web*. Através dos diversos elementos de marcação (*markup tags*) que o HTML disponibiliza, é possível representar a semântica de uma página *web* numa estrutura em árvore.

A semântica em HTML significa que o *markup* fornece informação sobre o significado do conteúdo em vez de somente definir o aspeto visual (Makzan, 2011). De entre esses elementos destacam-se cabeçalhos, parágrafos, tabelas, entre outros. Geralmente, tem-se como uma boa prática, a separação da estrutura e estilos. Enquanto que a estrutura é definida no HTML, os estilos, ou seja, a apresentação visual é definida através de *Cascading Style Sheets* (CSS) (Meloni, 2011; Duckett, 2009). McFarland (2009) refere que um estilo (*style*) não é mais do que uma instrução a descrever a formatação de um certo elemento do HTML, e que uma folha de estilos (*style sheet*) é um conjunto dessas instruções. McFarland (2009) alerta ainda para o facto de que o HTML e o CSS não são a mesma coisa, pois o CSS é uma linguagem diferente que trabalha em conjunto com o *browser* para melhorar a forma como o HTML é mostrado.

Apesar de se poder introduzir os estilos diretamente no código HTML, torna-se mais fácil a manutenção do código e ainda se elimina possíveis redundâncias. Para além da estrutura e dos estilos, existe ainda uma componente que introduz a dinâmica e a interatividade nas páginas *web*: o JavaScript (Duckett, 2009; Keith and Sambells, 2010). O JavaScript foi a primeira linguagem *web* de *scripts*, a ser suportada e interpretada pela maioria dos *browsers*, permitindo trabalhar conjuntamente com o *Document Object Model* (DOM) (Meloni, 2011). O DOM visa a representação de uma página *web* numa hierarquia com nós, dando o controlo sobre o conteúdo e a estrutura do documento, podendo-se remover, adicionar, substituir ou modificar esses nós (Zakas, 2012). Meloni (2011) refere o DOM como uma hierarquia utilizada pelo JavaScript com objetos pais e filhos, em que cada objeto tem propriedades que são variáveis

que descrevem a página *web*, e métodos que são funções que permitem trabalhar com as diferentes partes da página *web*. Para mais detalhes acerca do DOM, sugere-se (Flanagan, 2006), que apresenta uma descrição exaustiva sobre o DOM em diversos contextos.

No passado, uma página *web* era principalmente criada sobretudo para efeitos de leitura (MacCaw, 2011), apresentando algumas limitações nos conteúdos que disponibilizava num ambiente estático e pouco amigável para o utilizador. Felizmente, o grande potencial da *web* tem vindo a ser explorado, pelo que novas especificações e funcionalidades do trio HTML+CSS+JavaScript têm sido lançadas para melhorar a experiência do utilizador (e facilitar a vida do programador).

HTML5 é a nova especificação do HTML, cujo desenvolvimento está sob a alçada do *World Wide Web Consortium* - W3C (www.w3.org). O HTML5 vem melhorar não só a semântica como também a relação entre a estrutura definida na marcação, a interpretação de caraterísticas definida pelas diretivas de estilo e o conteúdo da página *web* definido pelo texto em si. Além do mais, introduziu novos *open standards* nativos para a implementação de conteúdo multimédia como áudio e vídeo, armazenamento local, API para geolocalização, WebSockets e muito mais (Casario et al., 2011; Taivalsaari and Mikkonen, 2011; Pilgrim, 2010).

Contudo, no HTML5 nem tudo é "perfeito", principalmente quando se fala em suporte *cross-browser*. Schmitt and Simpson (2011) alertam para o facto de que apesar dos antecessores do HTML5 serem suportados na sua totalidade pelos *browsers*, existem funcionalidades introduzidas pelo HTML5 que apresentam limitações no que se refere a suporte *cross-browser*. Existem, no entanto, algumas ferramentas que podem vir a ser úteis quando se pretende aferir acerca do suporte dos *browsers* sobre determinado elemento do HTML5, nomeadamente o HTML5test (HTML5test, 2013) e o Can I Use (CanIUse, 2013). Outro sítio bastante útil é o HTML5 Rocks (HTML5rocks, 2013) com artigos interessantes e explicações sobre diversos temas e as mais recentes atualizações no domínio do HTML5.

De forma similar, também se verificou uma evolução no CSS com o surgimento do CSS3, que permite ao desenvolvedor criar *designs* e estilos de forma nativa sem os ter de construir externamente recorrendo a aplicações específicas, como o *Adobe Photoshop*, e de seguida proceder à sua exportação. Por exemplo, o simples facto de colocar sombra no texto de um cabeçalho permite uma grande poupança de tempo (Anthes, 2012).

Mais especificamente no caso das aplicações *web* com uma forte componente interativa, o JavaScript é provavelmente a tecnologia que desempenha o papel fundamental, pois como já foi mencionado, é através dele que se consegue a dita interação entre o utilizador e a interface da aplicação. O JavaScript acabou por se tornar na linguagem por *scripts* padrão no lado do *browser*, apesar da má reputação que deteve durante quase uma década desde que foi criado em 1995 (Kessin, 2011). Bibeault and Katz (2010) apontam como principal factor para a recuperação do prestígio por parte do JavaScript, o renovado interesse em aplicações *web* altamente interativas de nova geração, denominadas *Rich Internet Applications* (RIA) (Lennon, 2010; Taivalsaari and Mikkonen, 2011) ou DOM-*scripted Applications*. Um exemplo deste facto é dado por MacCaw (2011) que refere que uma série de aplicações novas têm aparecido recentemente dando uma experiência ao utilizador que era habitual em aplicações nativas e que era desconhecida em ambiente *web*. Moore et al. (2007) explora de forma aprofundada o tema das RIA, incidindo não só nas fases iniciais de desenvolvimento deste tipo de aplicações, como também em funcionalidades mais específicas, das quais se destacam a usabilidade e validação de formulários, a interação com o utilizador, os efeitos e as animações.

2.5 *Middleware* e *back-end* das aplicações *web*

Como já foi referido na secção 2.3, o *middleware* visa interligar os *layers* de apresentação e de dados, sendo este último também conhecido como *back-end*. No *middleware* é

portanto crucial, sobretudo em aplicações de larga escala e com um grande número de utilizadores, existirem mecanismos robustos em aspetos como *routing*, segurança e acesso às bases de dados constantes no *back-end* (Thakur, 2008).

Há semelhança do que acontece para o *front-end* (ver Secção 2.4), também no *middleware* se constata o uso de diversas linguagens para a sua codificação. Alguns exemplos são: PHP, ASP, Python, Java, C#, entre outras (Thakur, 2008; Bakken, 2003). Todavia, para além do conceito de JavaScript no lado do cliente (*Cliente-Side* JavaScript - CSJS), nos últimos tempos tem-se verificado cada vez mais referências a JavaScript no lado do servidor (*Server-Side* JavaScript - SSJS).

Como Kiessling (2011) referiu, as primeiras "encarnações" do JavaScript estavam associadas a *browsers*, no entanto, tendo em conta a possibilidade de se utilizar JavaScript no *middleware* e *back-end*, constata-se a vantagem de se poder usar a mesma linguagem para desenvolver uma aplicação *web* tanto no lado do cliente como no lado do servidor (Waterson, 2009). A extensão do JavaScript para o lado do servidor é possível via *engines* de JavaScript embebidas. As duas mais conhecidas são o SpiderMonkey e o Rhino (Waterson, 2009), atualmente sob a alçada da Mozilla Foundation (Mozilla, 2013). O SpiderMonkey é o nome de código para a primeira *engine* de JavaScript *open source*, implementada em C. Por sua vez, o Rhino é uma implementação em Java de JavaScript (Waterson, 2009).

Apesar das tecnologias já existentes, uma em particular tem dado provas do seu valor e tem atraído cada vez mais adeptos: o NodeJS (NodeJS, 2013). O NodeJS permite correr código JavaScript no *back-end*, utilizando para o efeito, a *engine* V8 da Google (Google, 2013) que é usada no Chrome, para facilitar a criação de aplicações *web* escaláveis. Além do mais, existe uma grande diversidade de módulos que se podem utilizar no NodeJS para os mais variados fins (Herron, 2011). Pode-se, portanto dizer que o NodeJS é uma biblioteca e um *runtime environment* (Kiessling, 2011). O'Dell (2011) refere como se deu o crescimento do NodeJS e alguns exemplos de aplicações. Kiessling (2011) ensina os primeiros passos para um principiante ter uma ideia de como

o NodeJS funciona, explicando a criação de um servidor HTTP básico e de conceitos pertinentes sobre essa matéria.

2.6 *Design patterns* e *frameworks*

Como foi já referido, a grande evolução verificada não só no JavaScript, como também nos *browsers* é, segundo MacCaw (2011), um dos principais motivos para a exequibilidade de aplicações *web* em JavaScript e para a sua crescente popularidade. MacCaw (2011) aponta ainda aplicações como o Gmail ou o Google Maps como revolucionários na forma como pensamos e projetamos aplicações *web*. Não existe uma única forma correta de estruturar uma aplicação *web*, todavia existem alguns princípios ou, por outras palavras, "boas práticas" que devem ser seguidas por forma a se alcançar uma arquitetura o mais próximo possível da ideal.

Nesse sentido surgem os *design patterns*, que não são mais que soluções reutilizáveis para problemas comuns que ocorrem no *design* de *software*. Osmani (2012) aponta três vantagens cruciais no recurso a *design patterns*, nomeadamente: a) O facto de serem soluções com provas dadas, visto que refletem a experiência e percepção dos programadores que ajudaram à sua definição e melhoramento; b) São facilmente reutilizáveis, pois podem-se adaptar às nossas próprias necessidades e c) São geralmente expressivos, dado que quando olhamos para um determinado *pattern* aplicado numa certa situação, constatamos a existência de uma estrutura e um vocabulário predefinidos para a sua solução, que pode auxiliar a projeção de soluções em situações mais complexas de uma forma elegante.

Todavia, Osmani (2012) reforça que os *design patterns* não são uma solução exata, pois eles não resolvem todos os problemas de *design* nem substituem os programadores, apenas auxiliam providenciando um esquema para a solução. Alguns exemplos de *design patterns* são: *Constructor Pattern*, *Singleton Pattern*, *Module Pattern*, *Observer Pattern*, *Model-View-Controller (MVC) Pattern*, entre outros (Osmani, 2012).

Como já foi mencionado, utilizar JavaScript nem sempre é uma tarefa linear devido às complicações que surjem quando pretendemos, por exemplo, um suporte *cross-browser*. O principal motivo para esta dificuldade tem a ver com o facto de cada *browser* ter a sua implementação de JavaScript, podendo tornar-se um pesadelo tentar garantir que uma aplicação seja compatível em todos eles (Lennon, 2010). É aqui que entram as *frameworks* de JavaScript. Elas disponibilizam um conjunto de funções tornando mais fácil a produção de código JavaScript compatível com todos os *browsers*, visto que as *frameworks* são alvo de diversos testes em diferentes versões dos mesmos. Assim consegue-se uma maior garantia de que a nossa aplicação *web* irá funcionar de forma semelhante nos vários *browsers*.

Lennon (2010) e Lindley (2009) apresentam algumas vantagens relevantes na utilização de *frameworks*: a) Facilitam a escrita de código para manipulação de elementos do DOM, com funções básicas que devolvem referências a certos elementos e também funções para alteração dos mesmos; b) Permitem o tratamento de eventos, que é outro ponto difícil de alcançar para um suporte *cross-browser* e c) Padronizam processos, ajudando na flexibilidade dos projetos.

Um exemplo de uma *framework* de JavaScript bastante utilizada é o jQuery (jQuery, 2013). O jQuery não é mais que uma biblioteca de JavaScript *open source* que simplifica as interações entre o documento HTML, ou mais precisamente o DOM e o JavaScript (Lindley, 2009). Para além de ser bastante intuitivo de se usar, o jQuery resolve um ponto bastante importante referido anteriormente que são as incompatibilidades *cross-browser*. Além do mais, o jQuery permite (Chaffer and Swedberg, 2011): a) Facilidade no acesso e alteração dos diversos elementos de um documento, pois coloca ao dispor dos programadores um robusto mecanismo de seletores, tornando simples o seu acesso para inspeção ou manipulação incluindo ainda a possibilidade de inserção de novos elementos no documento HTML; b) Modificar a aparência de uma página *web*. Embora o CSS influencie a aparência do documento, nem todos os *browsers* suportam os mesmos *standards*. Com o jQuery consegue-se alterar o estilo das classes ou

de elementos individuais. O jQuery dispõe ainda de uma variedade de efeitos animados para se conseguir criar ambientes interativos e c) Oferece melhor controlo dos eventos, permitindo responder de forma consistente a diversos eventos como cliques, arrastamentos, entre outros.

Um outro ponto que requer atenção é a interface do utilizador (*User Interface* - UI). No que diz respeito à implementação de interfaces apelativos e funcionais, há a considerar o jQuery UI (jQuery UI, 2013). Sarrion (2012) define o jQuery UI como um conjunto de *plugins* que assentam na biblioteca jQuery (vista anteriormente), para a criação de *user interfaces*, fornecendo novos efeitos, *widgets* e temas. Existem ainda outras *frameworks* cujo objetivo é meramente providenciar uma forma fácil de se criar interfaces *web* que se ajustam ou adaptam ao dispositivo e ao *browser* que está a ser utilizado, isto é, interfaces responsivas. De entre as mais conhecidas, há a salientar o UIkit (UIkit, 2013) e o Bootstrap (Spurlock, 2013).

Por outro lado, no que diz respeito à criação de aplicações *web*, existem *frameworks* para os mais variados fins. Todavia, há a destacar: Ember.js, Backbone.js e AngularJS (Osmani, 2012; MacCaw, 2011; Green and Seshadri, 2013).

2.7 Tecnologias para a implementação de uma aplicação *web* com uma forte componente gráfica e vídeo

Até há bem pouco tempo só era possível representar gráficos interativos na *web* recorrendo a certas tecnologias como o *Adobe Flash*, *Scalable Vector Graphics* (SVG) ou o *Microsoft Silverlight*. 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 (como se irá ver na secção 3.3). 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 amplia-

ção devido à sua natureza vetorial. Pelo contrário, no 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 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. 2.2 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 and Paksula (2010) abordam o tema da integração entre Canvas e SVG.

Tabela 2.2: Tabela comparativa entre HTML5 Canvas e SVG.

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

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, 2013), Fabric.js (Fabric, 2013), KineticJS (KineticJS, 2013) e Paper.js (Paperjs, 2013).

Quando se fala de vídeo na *web*, existem duas fases que marcaram este ramo da multimédia: o "antes" e o "depois" do surgimento do HTML5. Anteriormente ao HTML5, e de forma similar ao que aconteceu na componente gráfica interativa, a apresentação de conteúdo áudio e vídeo na *web* era somente possível com o recurso a *plugins* externos, como o *Adobe Flash*. Assim, os programadores só poderiam incluir vídeo numa página através dos elementos `<object>` e `<embed>`. Hoje em dia, o HTML5 disponibiliza o elemento `<video>` que permite a inserção de vídeos numa página sem o recurso a quaisquer *plugins*. Apesar do elemento `<video>` ser suportado pelos *browsers* mais recentes, temos de ter em conta que os *browsers* mais antigos poderão não

suportá-lo.

O Internet Explorer 8, por exemplo, é um dos que não suportam o vídeo nativo da especificação HTML5 e, portanto seria útil implementar nestes casos um mecanismo de *fallback*, por exemplo para o *Adobe Flash* (Lawson and Sharp, 2011; Pfeiffer, 2010; Casario et al., 2011). Desta forma conseguimos minimizar as perdas para os seus utilizadores. É verdade que alguma funcionalidade poderá ser perdida se o Editor de Vídeo for projetado e implementado em torno do HTML5, contudo esses utilizadores terão acesso à visualização de vídeos. Existe outro pormenor a ter em conta para além do suporte do elemento `<video>`, que é o suporte dos formatos de vídeo. Aquando da escrita desta dissertação, existiam três formatos suportados: MP4, OGG e WebM. Todavia, há que ter em consideração que cada formato engloba um *codec* de vídeo e um *codec* de áudio. A Tab. 2.3 mostra a conjuntura atual no que diz respeito ao suporte dos principais *codecs* por *browser* e plataforma.

Tabela 2.3: Suporte dos *codecs* de vídeo e áudio por *browser*/plataforma.
Fonte: HTML5test (2013).

***Browser*/Plataforma**	***Codecs* de Vídeo**	***Codecs* de Áudio**
Internet Explorer 11	H.264	AAC, MP3
Firefox 26	OGG Theora, WebM VP8	OGG Vorbis, MP3
Chrome 31	H.264, OGG Theora, WebM VP8	AAC, OGG Vorbis, MP3
Opera 18	OGG Theora, WebM VP8	OGG Vorbis
Safari 7	H.264	AAC, MP3
Android 4.4	H.264, WebM VP8	AAC, OGG Vorbis, MP3
iOS 7	H.264	AAC, MP3
Windows Phone 8	H.264	AAC, MP3

No nosso ponto de vista, é necessário usar uma combinação de dois formatos diferentes de forma a abranger a grande maioria dos *browsers*. Pfeiffer (2010) indica que se se pretende criar um *website* de raíz com suporte de vídeo, a melhor escolha seria utilizar MP4 (ficheiro: MPEG-4; *codec* de vídeo: H.264; *codec* de áudio: AAC) em conju-

gação com WebM (ficheiro: WebM; *codec* de vídeo: VP8; *codec* de áudio: OGG Vorbis), já que o domínio do WebM está em constante melhoramento. Pfeiffer (2010) apresenta ainda mais detalhes sobre os *codecs* de áudio e vídeo e também explica como converter os vídeos para os formatos específicos (MP4/M4A, OGG/OGV e WebM).

De forma semelhante ao que se verifica atualmente no domínio do HTML5 Canvas, também para o HTML5 Vídeo existem algumas *frameworks* de JavaScript, nomeadamente: Video.js (Videojs, 2013), HTML5media (HTML5media, 2013), MediaElement.js (MediaElement, 2013), Popcorn.js (Popcorn, 2013) e jPlayer (jPlayer, 2013). Powers (2011) e Pfeiffer (2010) referenciaram veemente a conjugação de HTML5 Vídeo e Canvas para a criação de aplicações multimédia mais ricas.

2.8 Breve discussão

Pelo exposto anteriormente, e de acordo com as imposições estabelecidas no projeto FootData (Rodrigues et al., 2013), decidiu-se optar pela elaboração de quatro aplicações *web* nomeadamente, os Editores de Campo, Vídeo, *Tracking* e Apresentações, inseridas num contexto de *cloud-computing* (Sosinsky, 2011; Miller, 2008; Buyya et al., 2011), cujos *front-ends* serão implementados utilizando HTML5, CSS3 e JavaScript. No que diz respeito ao *middleware* e *back-end*, todo este módulo será estruturado utilizando o NodeJS.

Em ambos, iremos recorrer a diversas das *frameworks* de JavaScript mencionadas neste capítulo (e.g., jQuery, KineticJS, entre outras). As tecnologias a utilizar para a construção das bases de dados necessárias para todo o SI não fazem parte do contexto desta dissertação.

Existem algumas ferramentas não mencionadas no presente capítulo, mas que merecem destaque. Um exemplo é o Yeoman (Yeoman, 2013). Esta ferramenta engloba uma série de *frameworks* (e.g., AngularJS), fornecendo um ambiente de desenvolvimento e produção para a projeção de uma aplicação *web*, automatizando ainda algu-

mas tarefas relacionadas com a inicialização, construção e teste da aplicação. Green and Seshadri (2013) exemplificam a criação de uma aplicação *web* utilizando o Yeoman, abordando com detalhe o processo de instalação do Yeoman, desenvolvimento, teste e produção da uma aplicação.

3

Projeto e desenvolvimento das ferramentas *web*

Neste capítulo iremos abordar o projeto e implementação dos Editores de Campo, de Vídeo, de *Tracking* e de Apresentações 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 os produtos existentes no mercado com os propostos por nós. Todavia, convém 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

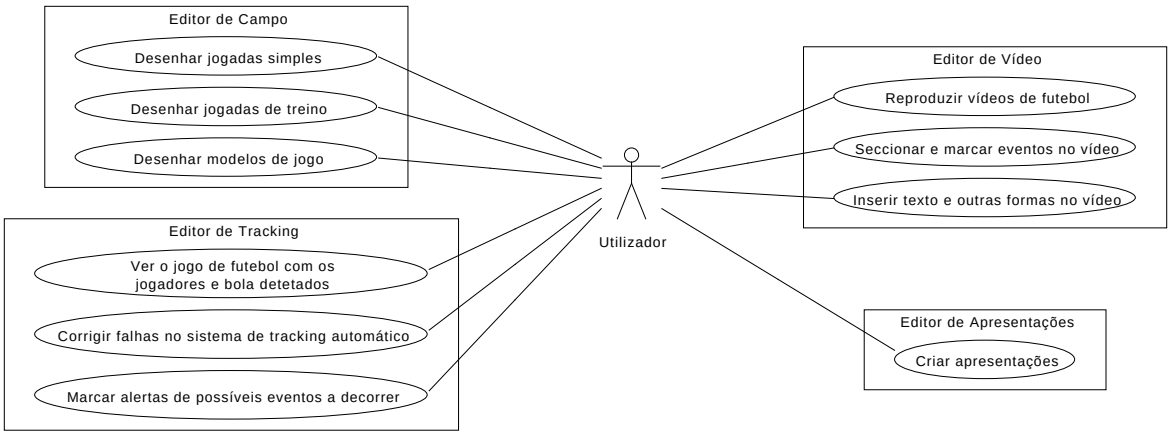


Figura 3.1: Diagrama de casos de utilização das ferramentas a implementar.

solução, procurou-se fazer comparações tendo como base, a documentação e os vídeos de demonstração disponibilizados pelas entidades envolvidas na criação dessas aplicações.

3.1 Introdução

O desenvolvimento das ferramentas (e de todo o projeto) fugiu ao desenvolvimento e planificação "tradicional" (Pfleeger and Atlee, 2009). O consórcio do projeto sabia exatamente quais os objetivos finais a atingir com cada ferramenta (ver Fig. 3.1 e o descrito na Secção 1.2), no entanto existiam muitas dúvidas sobre qual a melhor forma de os alcançar, bem como ao nível da apresentação das interfaces e da usabilidade. O consultor, treinador Domingos Paciência, foi de extrema utilidade nesta fase de planificação e projeto, dando informações extremamente valiosas, contudo, rapidamente se chegou à conclusão de que não seria possível fazer o desenho ("engenharia") completo de cada uma das ferramentas, e depois a sua implementação. Por outras palavras, teríamos de adotar inicialmente uma estratégia de prototipagem.

Prototipagem de *software* é um processo iterativo de geração de modelos, isto é, uma atividade de desenvolvimento de uma versão inicial do sistema baseada no entendimento dos requisitos ainda pouco definidos. Um protótipo simula a aparência

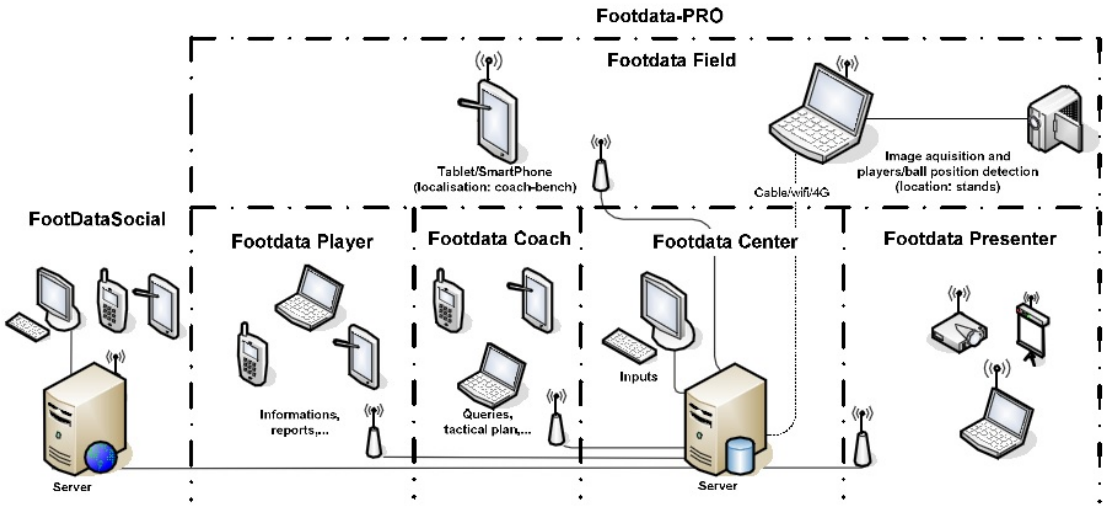


Figura 3.2: Estrutura da plataforma do projeto FootData.

e funcionalidade do *software* permitindo que os clientes, analistas, desenvolvedores e gerentes percebam os requisitos do sistema podendo interagir, avaliar, alterar e aprovar as caraterísticas mais marcantes na interface e funcionalidades (Bell, 2005). Neste caso optou-se por fazer uma prototipagem evolucionária, tendo ocorrido várias interações entre a equipa de investigação e desenvolvimento e os treinadores (futuros utilizadores) tendo como base o desenvolvimento de cada uma das ferramentas individualmente mas não esquecendo que algumas podem comunicar e interagir entre elas.

Como já foi referido no Capítulo 2, pretendemos que as ferramentas sejam desenvolvidas para serem usadas em qualquer *browser* e estejam enquadradas no projeto FootData (v.d. Secção 1.1), onde as interfaces sejam estruturadas com HTML5, estilizadas com CSS e as funcionalidades controladas por JavaScript. A Fig. 3.2 complementa a informação fornecida na Secção 1.1, e esquematiza a estrutura geral do produto, indicando os seus diversos módulos.

Convém relembrar que esta plataforma possui duas componentes fundamentais: uma rede social e uma componente profissional. De acordo com a Fig. 3.2, a rede social está contida no módulo *FootDataSocial*, e por outro lado, a componente profis-

sional está associada ao módulo *FootData-Pro*, sendo que, este último, contém vários submódulos, cujas designações foram atribuídas consoante as caraterísticas do ramo de gestão da equipa em que se inserem, pelo que, cada submódulo alberga diversas ferramentas e interfaces.

O submódulo *FootData Player* contém várias interfaces especialmente desenvolvidas para os jogadores com a mais variada informação (e.g., relatórios da equipa técnica e convocatórias). O *FootData Coach* abrange todas as ferramentas e interfaces exclusivas à equipa técnica desde a organização e gestão dos recursos até à planificação e análise de jogos e treinos. De entre as ferramentas mais importantes neste submódulo, destacam-se o Editor de Campo e o Editor Vídeo. O *FootData Presenter* inclui todas as interfaces e utilitários para efeitos de apresentações e palestras à equipa. Como é de prever, o Editor de Apresentações faz parte deste submódulo. No *FootData Field*, encontram-se todas as ferramentas inerentes à aquisição e processamento de imagens captadas no campo. Esse processamento será realizado através do sistema de *tracking* de campo especificado no projeto FootData (Sousa, 2012), onde estarão a ser detetadas e registadas as posições da bola e dos jogadores. No entanto, caso surja a necessidade de correção desses dados, existe uma ferramenta denominada Editor de *Tracking* para o efeito. Por fim, o submódulo *FootData Center* pode ser encarado como o núcleo, que interliga todos os outros submódulos do *FootData-PRO* e também o *FootDataSocial*, onde estão contidos os servidores e bases de dados para tratamento e armazenamento da informação.

Em termos funcionais (v.d. Fig. 3.3), os utilizadores acedem à *cloud* FootData, por meio de interfaces *web*, pelo que tudo o que envolve os servidores e bases de dados é-lhes transparente. Na fase de prototipagem, o *back-end* desta aplicação tem sido desenvolvido utilizando o NodeJS, na medida em que se criou um servidor HTTP onde ocorre o processamento da informação enviada pelas ferramentas *web*. No que diz respeito à comunicação entre o servidor referido e as ferramentas *web*, é feita por WebSockets, mais propriamente, através do Socket.IO (SocketIO, 2013), que é uma biblioteca

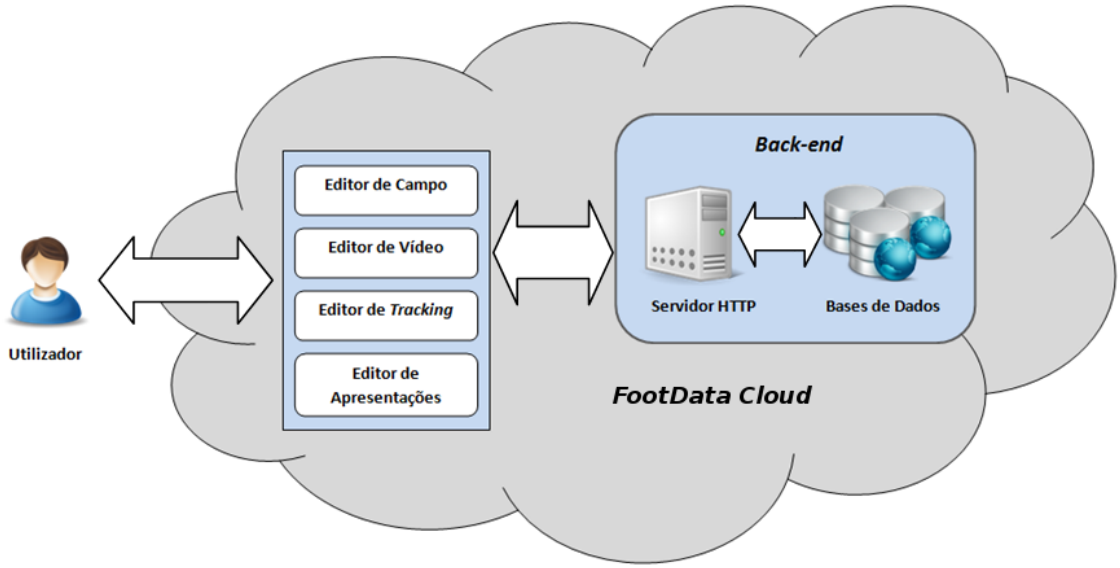


Figura 3.3: Interação entre o utilizador e a plataforma FootData, no que diz respeito às ferramentas *web* em discussão.

em JavaScript especialmente desenvolvida para aplicações *web*, constituída por duas componentes: uma biblioteca para o lado do cliente (*browser*) e uma biblioteca para o lado do servidor (implementado sobre o NodeJS).

Voltamos a salientar que o estudo e análise das bases de dados a utilizar, não faz parte do âmbito desta dissertação, pelo que, passamos de seguida a descrever cada uma das ferramentas e o respetivo processo de desenvolvimento.

3.2 Editor de Campo

Como foi descrito nos objetivos da dissertação (Secção 1.2), com o Editor de Campo pretende-se oferecer à equipa técnica uma plataforma para: a) Desenho de jogadas de futebol (*tactical pad*); b) Desenho de esquemas de treinos e c) Criação de esquemas mais complexos denominados modelos de jogo. Esses modelos podem ser estáticos, ou seja, considera-se apenas o posicionamento do jogador e/ou equipa e da bola num dado instante do jogo, ou dinâmicos, em que se considera o posicionamento dos mesmos durante um intervalo de tempo.

Os dados dos modelos de jogo serão utilizados para cruzamento de informação com a análise automática feita pelo sistema de *tracking* de jogadores e bola em tempo real (ou vídeo) e consequentemente, alertar o treinador acerca de ações feitas pelos jogadores ou equipa que estejam fora do modelo idealizado. De notar que pode acontecer a situação inversa, isto é, o treinador pode querer ser avisado quando o modelo de jogo que representou acontece, pelo que temos ambas as situações previstas.

Tendo em conta que as funcionalidades a) e b) podem ser adaptadas a partir da funcionalidade c), decidiu-se inicar a prototipagem por c), i.e., pela criação de uma plataforma que permita a definição de modelos de jogo. Na secção seguinte, mostraremos os detalhes do protótipo, implementado a partir dos *designs* iniciais para a ferramenta fornecidos pelo parceiro de projeto INESTING, que a moldou em torno da informação providenciada pelo treinador consultor.

A Fig. 3.4 representa o *design* inicial para o Editor de Campo. O ambiente de trabalho nesta aplicação vai ser um campo de futebol normal. No futuro pretende-se implementar outros ambientes como por exemplo, 1/2 campo, 1/3 campo, campo de treino, entre outros (no que diz respeito às funcionalidades a) e b)). Para além do utilizador ser capaz de inserir elementos no campo, poderá ainda, como já foi mencionado, interagir com eles: arrastando, atribuindo/alterando as suas propriedades (e.g., nome e tamanho), etc.

É importante nesta fase reforçarmos a ideia do que pretendemos representar no Editor de Campo, isto é, a implementação do modelo de jogo (funcionalidade c)). Tendo em conta as indicações dos consultores do projeto FootData, criámos a seguinte lista de elementos obrigatórios a implementar: campo, jogadores, bola, zonas, movimentos e medidas. O objetivo desta ferramenta é desenhar todos os modelos de jogo possíveis e jogadas que qualquer treinador possa idealizar, mais, que a representação possa facilmente ser convertida automaticamente em código, que por sua vez vai interagir com o *tracking* de jogadores e bola para a tomada de decisões (deteção de ações efetuadas pelos jogadores). Pretende-se ainda que a ferramenta seja de fácil utiliza-

Figura 3.4: *Design* inicial do Editor de Campo.

ção e intuitiva, pois a maioria dos treinadores não tem conhecimentos profundos em informática.

Reforçando a ideia previamente apresentada, as jogadas podem ser caraterizadas por, por exemplo, uma situação em que se procura verificar a distância entre jogadores ou setores da equipa em zonas específicas do campo e em função da bola, ou por uma situação definida por uma sequência de instantes que podem incluir, também a título de exemplo, distâncias entre jogadores, setores, o seu posicionamento relativamente à bola, passes e remates, isto durante um intervalo de tempo definido automaticamente ou pelo treinador.

Em resumo, os objetivos das variantes desta ferramenta podem ser definidos como:

Editor de Táticas, será provavelmente a variante mais simples. Procuramos providenciar ao treinador uma plataforma onde possa incluir jogadores de acordo com esquemas táticos predefinidos, dos quais são exemplo os vulgarmente designados por: $4 \times 4 \times 2$, $4 \times 3 \times 3$, entre outros. Tal plataforma pode vir a ser útil quando o treinador pretenda, de forma rápida e sem grande esforço, mostrar um certo esquema ou movimentação à equipa ou a um jogador, por exemplo antes de entrar em campo para substituir um colega. Por conseguinte, se por um lado elementos como jogadores e bola são à primeira vista os principais intervenientes, algumas funcionalidades como o estabelecimento de distâncias e zonas não parecem ter grande relevância neste caso. Nenhuma componente de animação será implementada nesta variante, contudo o utilizador poderá interagir com os elementos.

Editor de Jogadas de Treino, pretendemos que o treinador seja capaz de criar jogadas de treino, tendo para o efeito à sua disposição elementos como: jogadores, bolas, cones, estacas, balizas, barreiras, zonas e outros elementos gráficos pertinentes para o efeito. Pretendemos ainda que exista uma possibilidade de escolha do campo. Alguns exemplos de campos em discussão são: campo com linhas específicas existentes nos campos de treino, só meio campo, só um terço do campo, só áreas, campo sem linhas, entre outras hipóteses. Esta plataforma terá ainda de possuir uma componente

de animação de forma ao treinador poder explicar um exercício em particular por via da movimentação dos diversos elementos.

Editor de Modelos de Jogo, pretendemos implementar uma forma de desenho para posterior codificação (leia-se criar "código automático"; fora do âmbito desta tese) de modelos de jogo para se efetuar a análise e comparação com os dados do sistema de *tracking* (Sousa, 2012). O treinador deverá poder não só desenhar modelos de jogo para a sua equipa, como também estabelecer padrões relativamente à equipa adversária de forma a poder analisá-los mais tarde. Neste caso, elementos como jogadores, bola, zonas e estabelecimento de distâncias entre elementos serão fundamentais. Ao contrário do Editor de Jogadas de Treino, aqui não é necessário escolher o tipo de campo, i.e., o treinador terá somente o campo de futebol completo à sua disposição. Aqui também terá de existir a componente de animação, bem como a possibilidade de definição de corredores e setores. O treinador poderá ainda indicar se pretende ser avisado caso o modelo de jogo que criou ocorra de facto durante um jogo (tal ocorrência é detetada através da comparação do modelo aqui desenhado com a informação que se obtém através do sistema de *tracking*, como já foi mencionado) e que tipo de mensagem pretende receber no alerta em questão. Ainda nesta ferramenta, para além da representação dos elementos referidos, pretendemos implementar um outro modo de visualização mais técnico, devido à precisão que se exige no desenho de modelos de jogo. Por outras palavras, para que se consiga inserir jogadores e interagir com eles (clicar, arrastar, etc.), é necessário que eles possuam uma dimensão considerável para o efeito, daí eles serem representados no protótipo inicial como círculos com um diâmetro de 26 píxeis. Dadas as atuais dimensões do campo virtual (700×448 píxeis) e as dimensões de um campo de futebol real (105×68 metros), o elemento gráfico que corresponde ao jogador terá cerca de 3,9 metros. Constatamos pois, que um jogador ocupa uma área irreal no campo, tornando impossível, por exemplo, representar de forma adequada um grande conjunto de jogadores num espaço de reduzidas dimensões. Daí surgiu a necessidade de dar ao treinador uma visão mais exata dos desenhos

Tabela 3.1: Resumo das caraterísticas das variantes do Editor de Campo.

Parâmetros	**Editor de Jogadas de Treino**	**Editor de Modelos de Jogo**	**Editor de Táticas**
Elementos	Jogadores, bolas, cones, estacas, balizas, barreiras, zonas, etc	Jogadores, bola, zonas, distâncias	Jogadores e bola
Animação	Sim	Sim	Não
Interação com os elementos	Sim	Sim	Sim
Caraterísticas	Definição de jogadas de treino em diversos campos com uma variedade de elementos à disposição	Criação de modelos de jogo para comparação com os dados do sistema de *tracking*. Dois modos de visualização: genérico (mais funcional) e técnico (mais exato)	Acesso e uso em qualquer altura para representar jogadores, tendo como base esquemas táticos

feitos no Editor de Modelos de Jogo, em que representamos todos os elementos com as dimensões o mais real possível. Neste modo de visualização, os jogadores vistos de cima seriam representados por um único ponto/píxel (ou por 4 píxeis conetados), os restantes elementos serão representados por linhas, criando assim uma representação "*wireframe*" de todo o esquema (v.d. Secção 3.7.1, Fig. 3.45). Conjugando estes dois modos de visualização conseguiríamos alcançar a exatidão desejada sem comprometer a usabilidade e a interatividade da aplicação.

A Tab. 3.1 resume os principais aspetos de cada uma das variantes do Editor de Campo. Na secção seguinte vamos explicar detalhadamente o projeto e o desenvolvimento do Editor de Campo.

3.2.1 Projeto e desenvolvimento do Editor de Campo

Durante o processo de avaliação e escolha da melhor tecnologia para servir de base para o Editor de Campo surgiu um dilema na medida em que se ponderou a utilização de apenas uma tecnologia (HTML5 Canvas ou SVG) (v.d. Secção 2.7) ou a conjugação de ambas aproveitando dessa forma os pontos fortes das duas. É importante salientar quais as principais caraterísticas que o Editor de Campo deverá possuir: a) Dinâmica, visto que pretendemos ter uma componente de animação para os movimentos; b) Amigável para o utilizador, pois é importante que seja intuitiva para um treinador de futebol; c) Funcional, fornecendo várias ferramentas para um treinador desenhar praticamente qualquer tipo de jogada; d) Interativa, o utilizador deverá poder interagir com os objetos por meio de cliques, duplo cliques, arrastes, entre outros e e) Robusta, pois a animação de uma jogada que envolva vinte e dois jogadores deverá ter a mesma *performance* de uma jogada que envolva dois jogadores.

Dito isto, chegámos à conclusão que utilizar o HTML5 Canvas poderia ser mais vantajoso pelo facto de ter melhor *performance* que o SVG, quando existe uma grande quantidade de objetos desenhados e por se basear em JavaScript. Ainda assim, mantém-se a porta aberta para uma possível e futura integração com o SVG sempre que necessário. Sucan (2010) refere que, nestes casos, devemos ter sempre em conta ambas as tecnologias, sem se descartar uma em relação à outra.

Dada a complexidade pretendida para o Editor de Campo e dadas as limitações do Canvas anteriormente analisadas (v.d. Secção 2.7), é necessário termos uma boa *framework* de JavaScript para este, de forma a contornar o muito baixo nível da sua API nativa. Noutras palavras, se o nosso objetivo fosse desenhar somente formas simples não haveria qualquer transtorno em não usar uma *framework* de JavaScript para o Canvas, contudo, neste caso em que nós pretendemos formas mais complexas e animadas é crucial o recurso a uma *framework*. Na análise efetuada foram tidos em conta alguns aspetos como a documentação e API de cada *framework*, o suporte a eventos *desktop* e *mobile* e a variedade de objetos para utilizar. Outros dois pontos fundamentais foram o

facto da *framework* permitir a criação de animações e a dimensão da sua comunidade.

Perante as *frameworks* apresentadas na Secção 2.7, optou-se por utilizar o KineticJS (KineticJS, 2013), porque utiliza uma estrutura com *stage*, *layers* e *shapes*, que nos pode vir a ser bastante útil, permitindo ainda agrupar as diferentes *shapes* e aplicar ações específicas a esses grupos. Além do mais, o KineticJS, aquando da escrita desta dissertação, era um dos poucos que suportava eventos para os dispositivos móveis (e.g., *touchstart*, *touchmove*, *touchend*, *tap* e *dbltap*). Esta *framework* suporta ainda SVG *paths* que é um ponto que se poderá revelar vantajoso aquando da integração com os restantes módulos do projeto FootData. Outro aspeto muito importante e que teve bastante peso na nossa escolha é o facto do KineticJS suportar um grande número de *shapes* em simultâneo sem perder *performance* (HTML5CanvasTutorials, 2013).

Além do mais, uma outra funcionalidade que nós tínhamos como importante era poder gravar e carregar os esquemas criados nesta ferramenta, e de facto, o KineticJS permite guardar as formas inseridas em documentos no formato JSON (JSON, 2013). Segundo a página oficial, o *JavaScript Object Notation* - JSON 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. Com o JSON conseguimos, portanto, alcançar uma estrutura flexível e versátil para os dados em diversos contextos. A Figura 3.5 exemplifica um documento em JSON.

Escolhida a *framework* de JavaScript para se trabalhar no Canvas, o próximo passo é a implementação da realidade do futebol em código. Como foi referido anteriormente, o KineticJS dispõe de uma estrutura hierárquica com os elementos: *stage*, *layer*, *group* e *shape* (v.d. Fig. 3.6). Como se pode verificar, o *stage* é o elemento pai que irá conter todos os outros elementos. O *stage* pode conter vários *layers* e dentro desses *layers* podemos adicionar *shapes* ou grupos de *shapes*. Por sua vez, os grupos de *shapes* podem conter outros grupos de *shapes* ou *shapes* individuais. Contudo, convém referir que todos estes elementos não passam de nós virtuais, de forma similar ao que acontece nos nós do DOM em páginas HTML. O KineticJS cria assim, nós virtuais e possui

```
{
   "alunos": [
      {
         "id": 36569,
         "primeiroNome": "Pedro",
         "apelido": "Rodrigues",
         "dissertacao": {
            "ano": 2014,
            "titulo": "Ferramentas web para analise da performance de uma
               equipa de futebol"
         }
      }
   ]
}
```

Figura 3.5: Exemplo de um documento no formato JSON.

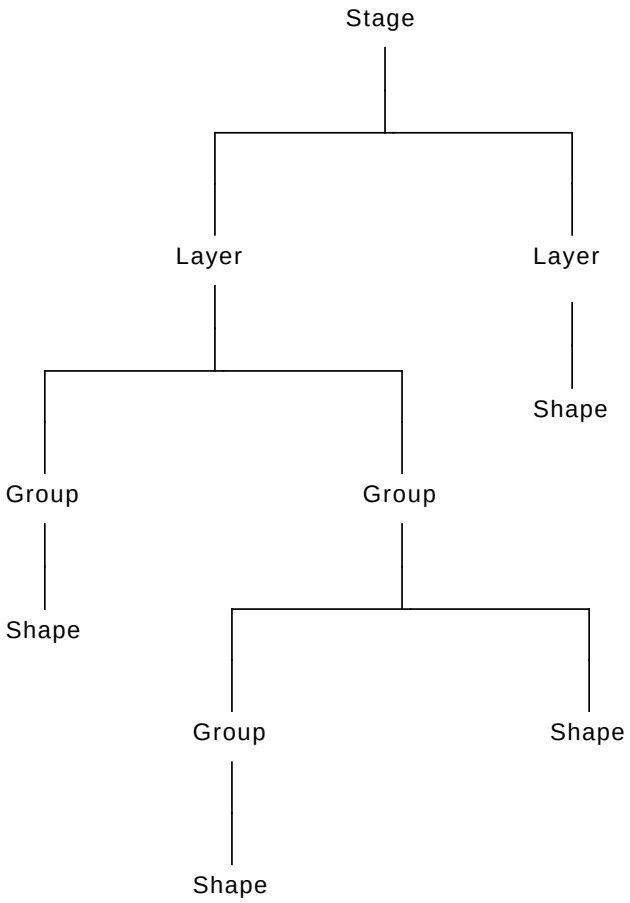


Figura 3.6: Exemplo de uma possível estrutura no KineticJS.
Adaptado de (KineticJS, 2013).

Tabela 3.2: Parâmetros de inicialização da classe *Field*.

Parâmetros de Inicialização	**Definição**
`width`	Largura
`height`	Altura
`container`	id do elemento de HTML que ficará associada ao *stage*

shapes muito simples, pelo que o problema com que nos deparamos é: com base no que o KineticJS nos fornece, conseguir criar uma plataforma com elementos do mundo do futebol.

O código em JavaScript foi feito o mais possível seguindo uma estrutura de programação semelhante à da programação orientada a objetos, de forma a torná-lo mais escalável e facilitar futuras manutenções. Visto que o KineticJS (como seria de esperar) não dispõe de elementos específicos do futebol como jogadores, zonas e movimentos, criámos as nossas próprias classes em que conjugámos várias classes mais simples e básicas do KineticJS (e.g., `Kinetic.Circle`, `Kinetic.Rect`, `Kinetic.Group`, etc) de forma a produzir os complexos elementos do futebol. Para facilitar a compreensão dos elementos de futebol que criámos, na Fig. 3.7 encontra-se esquematizado o diagrama de classes relativas ao Editor de Campo.

De seguida providenciamos uma breve explicação sobre os objetivos de cada classe e disponibilizamos ainda os diagramas das suas estruturas e exemplos da sua aplicação no campo:

a) Classe *Field* - com esta classe pretendemos criar uma base em que todos os outros elementos do futebol, que irão ser criados posteriormente, assentem. Os parâmetros de inicialização constam na Tab. 3.2. Quando se instancia esta classe, é criado um *stage* e três *layers*: *background_layer*, *objects_layer* e *tooltips_layer*. O primeiro irá servir para tratar as imagens de fundo. O *objects_layer* será o *layer* onde ficarão todos os elementos do futebol que serão desenhados, enquanto que no *tooltips_layer* estará contido

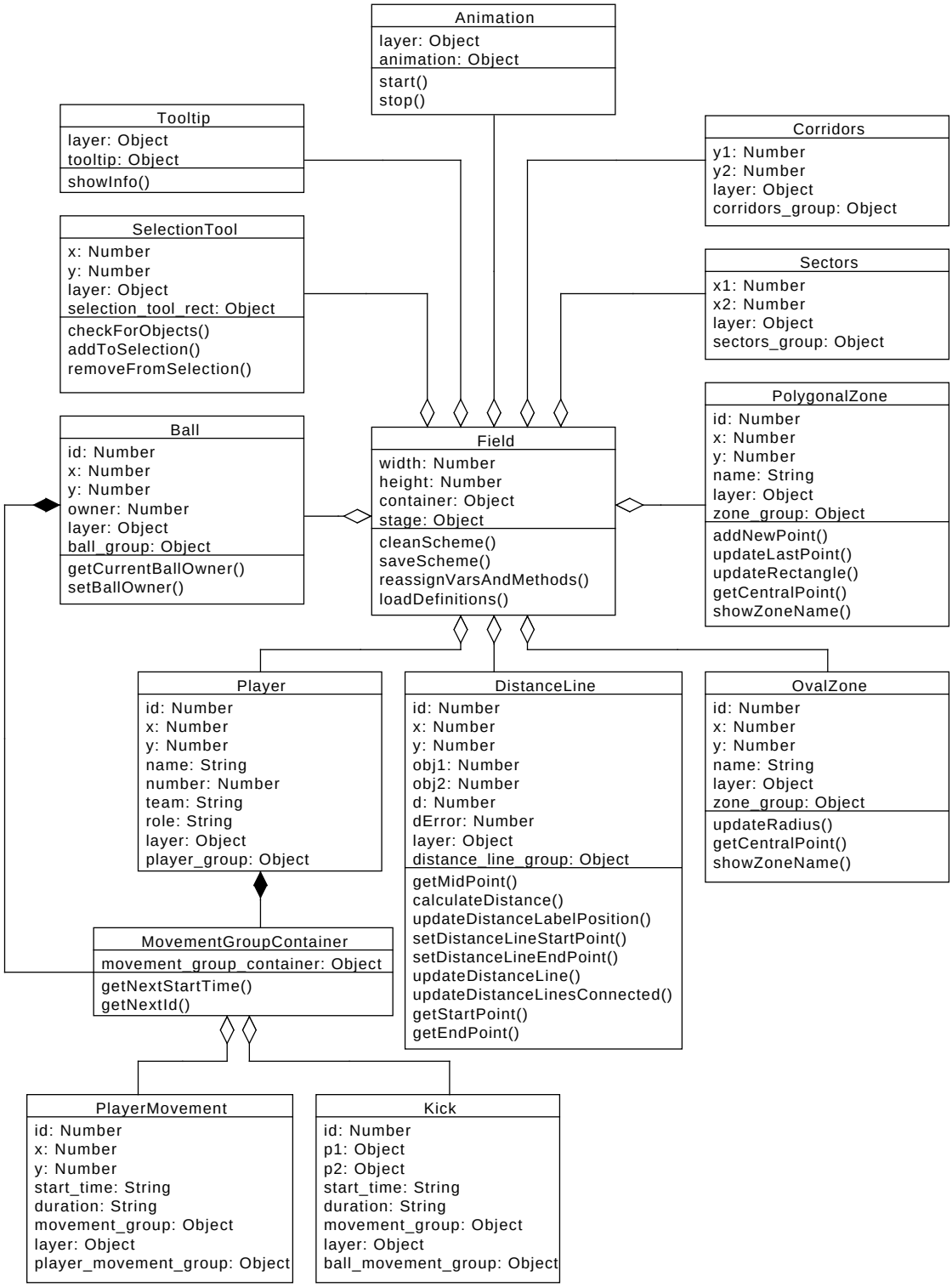


Figura 3.7: Diagrama de classes relativamente ao Editor de Campo.

um *tooltip* de informação que será explicado posteriormente. Ainda na classe *Field*, é estabelecido um grupo que servirá para guardar os elementos que foram selecionados

Tabela 3.3: Métodos da classe *Field*.

Métodos da classe *Field*	**Definição**
`cleanScheme`	Limpa todas os elementos do campo
`saveScheme`	Grava o desenho em campo e todas as definições para JSON e para imagem
`reassignVarsAndMethods`	Método que irá ser chamado quando ocorre o carregamento de algum esquema, com o objetivo de se reatribuir os métodos aos objetos presentes nesse esquema
`loadDefinitions`	Carrega as definições do esquema, isto é, de toda a informação relativa ao esquema, ao campo e ao sistema de alertas

com a ferramenta de seleção, e serão também instanciadas as classes *Animation* e *Tooltip*. Por fim, define-se ainda alguns métodos importantes para a manipulação e edição do campo (ver Tab. 3.3).

b) Classe *Tooltip* - esta classe, como o nome indica, será uma caixa com informação acerca de algum elemento. Quando é instanciada, é estabelecido um `Kinetic.Label` que inclui as classes `Kinetic.Tag` e `Kinetic.Text` onde são definidas as propriedades tanto da caixa que irá conter a informação, como do próprio texto. Por fim, é também definido um método denominado `showInfo` que controla o comportamento do *Tooltip*, tornado-o ou não vísivel e ainda alterando o texto consoante o elemento. Na Fig. 3.8 é exemplificado o aspeto do *Tooltip* para um jogador e para uma zona.

c) Classe *Player* - esta classe tem como principal objetivo representar um jogador de futebol. Os parâmetros de inicialização para a sua criação estão representados na Tab. 3.4. Quando esta classe é instanciada, é estabelecido um `Kinetic.Group` onde irão ser inseridas outras classes constantes no KineticJS: dois `Kinetic.Circle` em que um representa o jogador e o outro representa a sua margem de posicionamento, um `Kinetic.Text` para o número do jogador que é colocado no círculo de representação

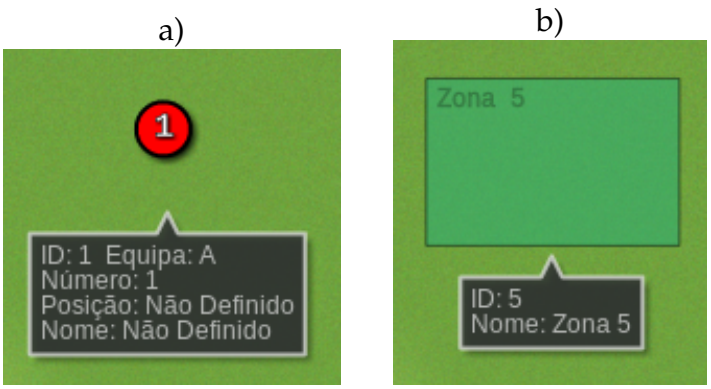


Figura 3.8: Exemplos de aplicação do *Tooltip*: a) Num jogador e b) Numa zona.

Tabela 3.4: Parâmetros de inicialização da classe *Player*.

Parâmetros de Inicialização	**Definição**
`id`	Número único para a representação do jogador em termos computacionais
`x`	Coordenada *x* da posição do jogador
`y`	Coordenada *y* da posição do jogador
`name`	Nome a atribuir ao jogador
`number`	Número a atribuir ao jogador
`team`	Equipa do jogador
`role`	Posição do jogador em termos táticos (defesa, avançado, etc)
`layer`	*Layer* onde o jogador será inserido

do jogador, e por fim é instanciado o *MovementGroupContainer* que irá conter todos os movimentos que possam posteriormente ser definidos para esse jogador.

De forma a entender melhor como foi estruturada esta classe, a Fig. 3.9 a) representa os seus principais componentes, enquanto que a Fig. 3.9 b) mostra o grafismo quando o jogador é inserido no campo.

Aqui, para além da representação do jogador, adicionámos mais atributos de forma a ir de encontro aos requisitos do futebol, dos quais são exemplo os atributos equipa (*team*), nome (*name*) e posição (*role*).

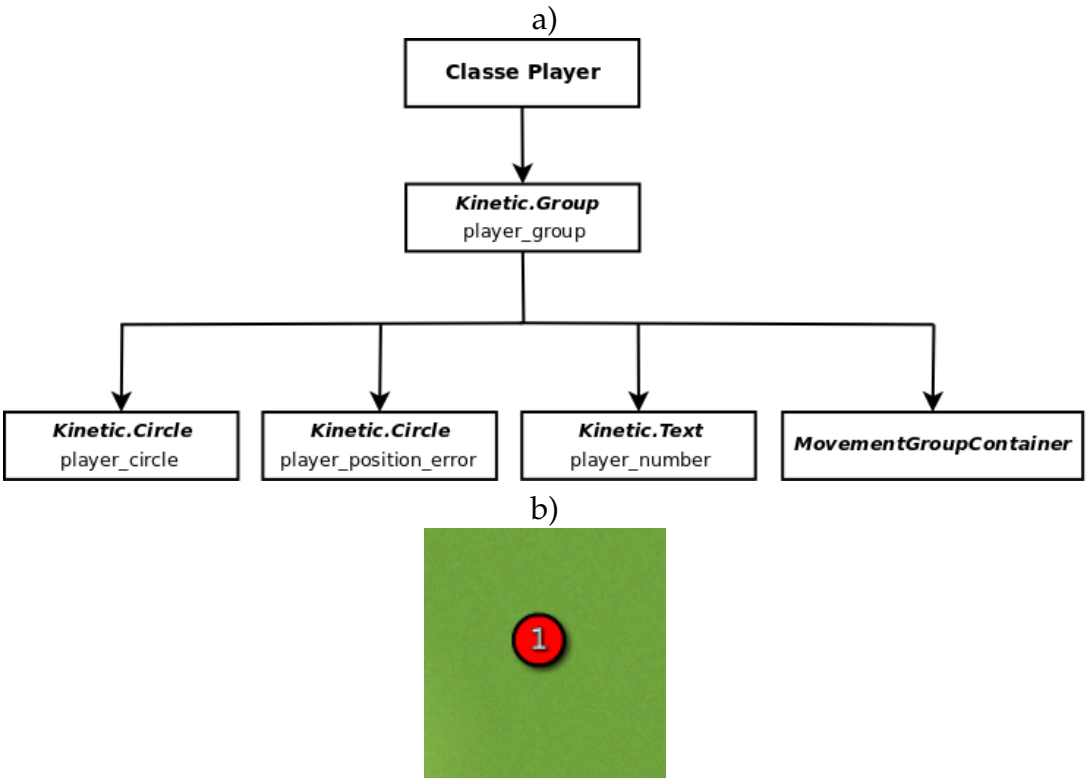


Figura 3.9: Classe *Player*: a) Estrutura do elemento com o KineticJS e b) Representação do elemento.

Tabela 3.5: Parâmetros de inicialização da classe *Ball*.

Parâmetros de Inicialização	**Definição**
`id`	Número único para a representação da bola em termos computacionais
`x`	Coordenada x da posição da bola
`y`	Coordenada y da posição da bola
`owner`	id do jogador portador da bola, caso exista
`layer`	*Layer* onde a bola será inserida

d) Classe *Ball* - como o nome sugere, esta classe destina-se à representação da bola de futebol (ver Fig. 3.10 b)). A Tab. 3.5 ilustra quais são os seus parâmetros de inicialização. Quando se instancia esta classe, há semelhança do que acontece na classe anterior, também aqui é criado um `Kinetic.Group` que irá conter várias instâncias de

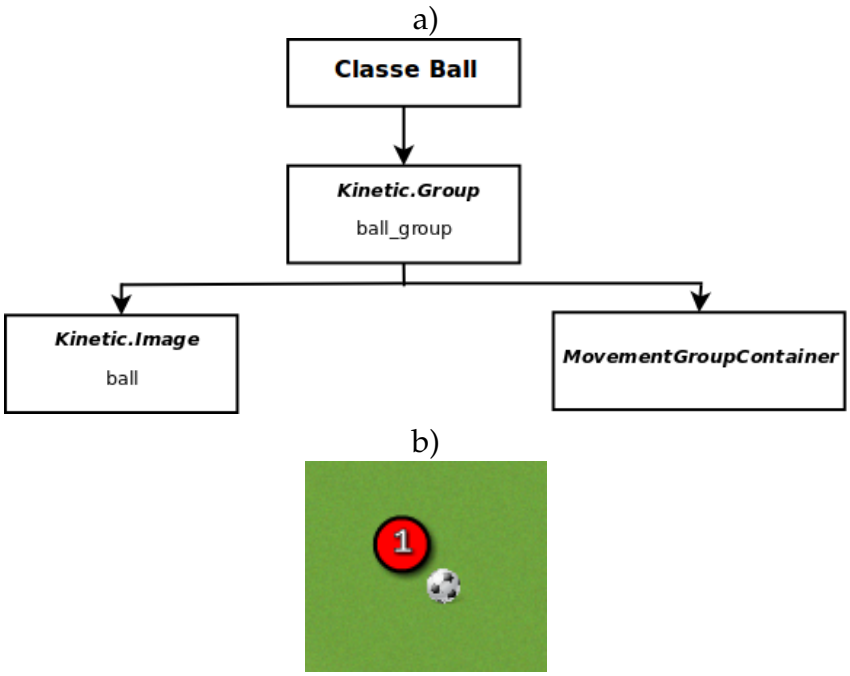


Figura 3.10: Classe *Ball*: a) Estrutura do elemento com o KineticJS e b) Representação de um jogador com posse de bola.

classes específicas do KineticJS (ver Fig. 3.10 a)). São elas: o `Kinetic.Image` onde se referencia o ícone no formato PNG (*Portable Network Graphics*) representativo da bola e o *MovementGroupContainer* que, de modo semelhante ao referido anteriormente, irá conter todos os movimentos para a bola.

Foram ainda criados dois métodos para esta classe visando o controlo da posse de bola (ver Tab. 3.6).

e) Classes *PolygonalZone* e *OvalZone* - a classe *PolygonalZone* permite a criação de zonas poligonais que irão ser fundamentais para condicionar as posições de elementos como jogadores, bola e inclusivamente outras zonas. Como se pode verificar pela Fig. 3.11, esta classe é constituída por um `Kinetic.Group`, onde estarão contidas instâncias de classes do KineticJS, tais como `Kinetic.Polygon` e `Kinetic.Text`. Quanto aos métodos para esta classe, eles encontram-se resumidos na Tab. 3.7.

De forma semelhante às zonas poligonais, as zonas ovais terão também o mesmo propósito de estebelecer limites no posicionamento de outros elementos no campo. No que diz respeito à estrutura (ver Fig. 3.12), irá ser praticamente igual à da zona poligo-

Tabela 3.6: Métodos da classe *Ball*.

Métodos da classe *Ball*	**Definição**
`getCurrentBallOwner`	Devole o id do jogador com a posse de bola, ou caso não exista nenhum portador de bola, devolve "Não Definido"
`setBallOwner`	Após se verificar que a bola foi arrastada para uma determinada posição, este método é invocado para verificar se houve colisão da bola com um jogador. Em caso afirmativo, esta situação é encarada como uma atribuição de posse de bola ao jogador em questão

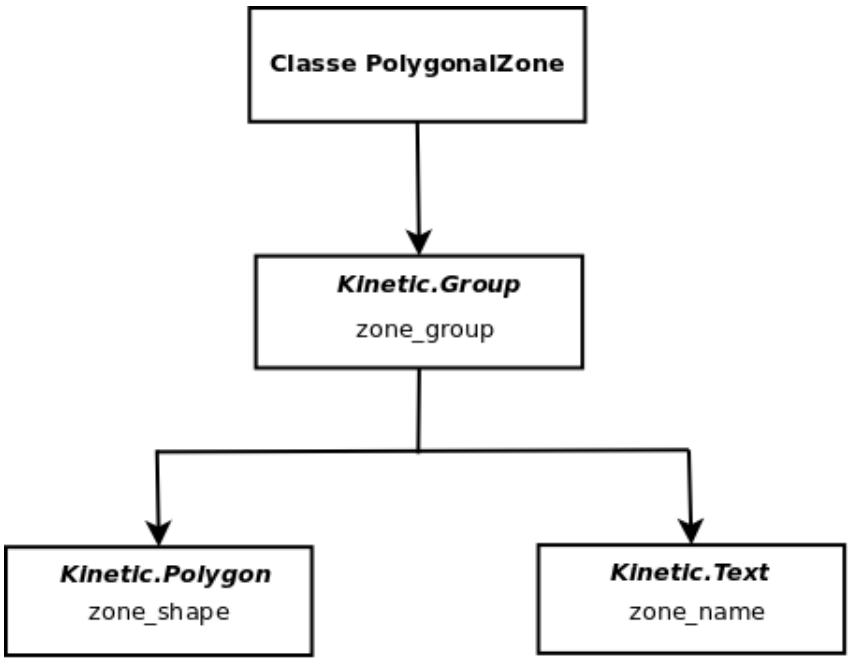


Figura 3.11: Classe *PolygonalZone*: estrutura do elemento com o KineticJS.

nal com a exceção de possuir uma instância da classe `Kinetic.Ellipse` em detrimento da classe `Kinetic.Polygon`. Os métodos elaborados para as zonas ovais estão resumidos na Tab. 3.8.

A Fig. 3.13 mostra três exemplos de zonas: duas poligonais em que uma é retangular (zona azul à esquerda) e a outra é uma zona personalizável desenhada ponto a

Tabela 3.7: Métodos da classe *PolygonalZone*.

Métodos da classe *PolygonalZone*	**Definição**
`addNewPoint`	Adiciona um ponto novo à zona
`updateLastPoint`	Atualiza a zona com base nos novos pontos, caso seja um polígono não retangular
`updateRectangle`	Atualiza a zona com base nos novos pontos, caso seja um retângulo
`getCentralPoint`	Devolve o ponto central da zona (caso seja retangular)
`showZoneName`	Mostra o nome da zona dentro da mesma (caso seja retangular), quando esta atinge uma dimensão mínima

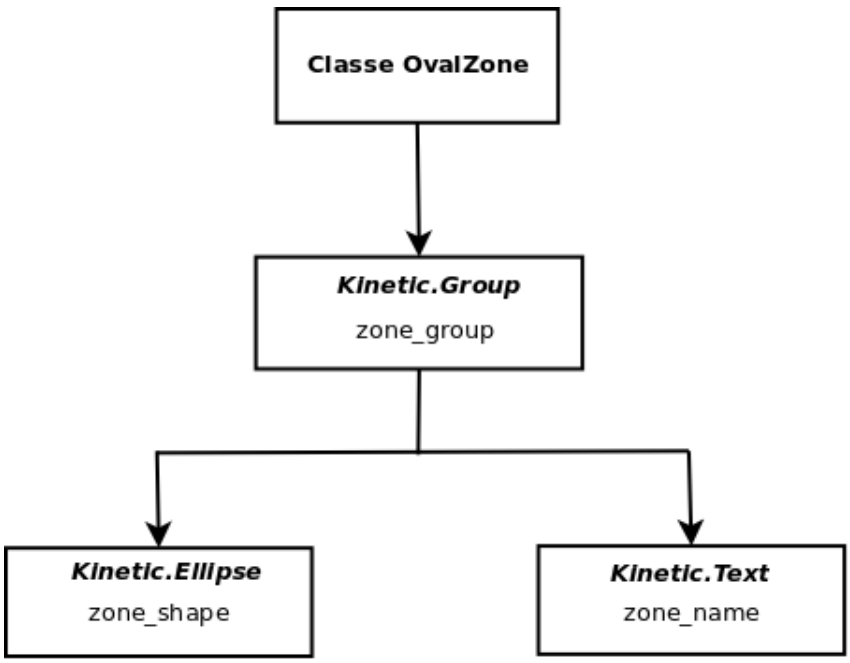


Figura 3.12: Classe *OvalZone*: estrutura do elemento com o KineticJS.

ponto (zona azul à direita) e por fim uma zona oval (zona verde à esquerda).

f) Classe *Corridors* e *Sectors* - estas classes surgiram da necessidade dos treinadores de futebol especificarem as distâncias dos setores e corredores para um dado modelo de jogo, isto é, uma forma de divisão segundo o comprimento e a largura, em diversas regiões cruciais no esquema tático. Decidimos abordar estas duas classes

Tabela 3.8: Métodos da classe *OvalZone*.

Métodos da classe *OvalZone*	**Explicação**
`updateRadius`	É utilizado quando se pretende alterar os raios horizontais e verticais da zona
`getCentralPoint`	Devolve o ponto central da zona
`showZoneName`	Mostra o nome da zona dentro da mesma, quando esta excede determinados valores para os raios horizontal e vertical

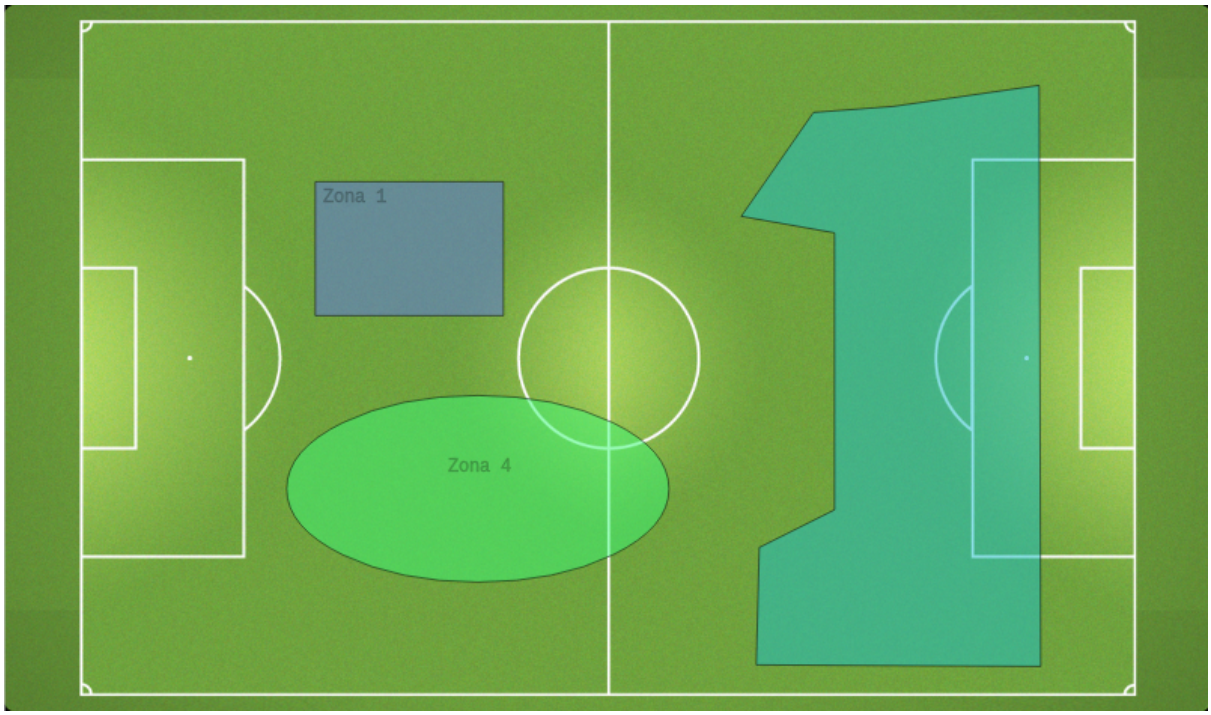


Figura 3.13: Exemplo de duas zonas poligonais (azul) e uma zona oval (verde).

em grupo visto que ambas desempenham o mesmo papel com a diferença de que o estabelecimento de corredores é feito por meio de divisões na horizontal e o de setores ser feito por divisões na vertical. Antes da explicação acerca da estrutura, neste caso, é mais conveniente exemplificar em primeiro lugar o funcionamento através da Fig. 3.14.

Na aplicação é possível arrastar os segmentos de reta (a tracejado vermelho) pelos respetivos cursores (quadrados cinzentos, numa das extremidades de cada segmento

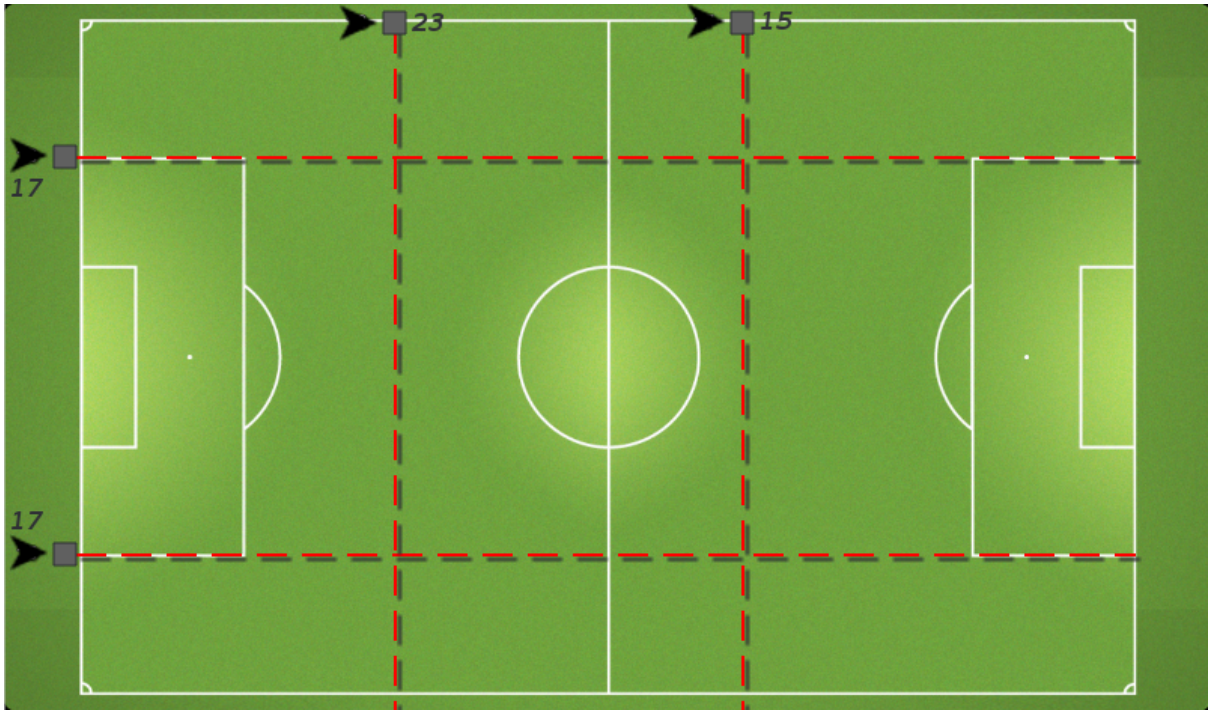


Figura 3.14: Funcionamento dos corredores e setores.

de reta), por forma a estabelecer a dimensão que se pretende para os corredores e setores. A referência para a distância nos setores é a linha do meio campo, enquanto que para a distância nos corredores, as referências são as respetivas linhas laterais. Os dois segmentos de reta horizontais fazem parte da classe *Corridors* e os dois segmentos de reta verticais fazem parte da classe *Sectors*. Como se pode ver pelas Figs. 3.15 e 3.16, ambas as classes são compostas por um `Kinetic.Group`, onde estão contidos dois outros `Kinetic.Groups`: um que agrupa todas as *shapes* de um dos segmentos de reta e o outro com as *shapes* do outro segmento de reta. Essas *shapes* são um `Kinetic.Line` para a representação do segmento de reta, um `Kinetic.Rect` para a representação do cursor cinzento e um `Kinetic.Text` para o número que se encontra junto de cada segmento de reta e que indica a distância à respetiva referência.

De notar que as duas estruturas são praticamente iguais com a exceção da disposição dos segmentos de reta. A classe *Corridors* alberga os segmentos de reta horizontais (*top* e *bottom*, no diagrama da Fig. 3.15) e a classe *Sectors* os segmentos de reta verticais (*left* e *right*, no diagrama da Fig. 3.16).

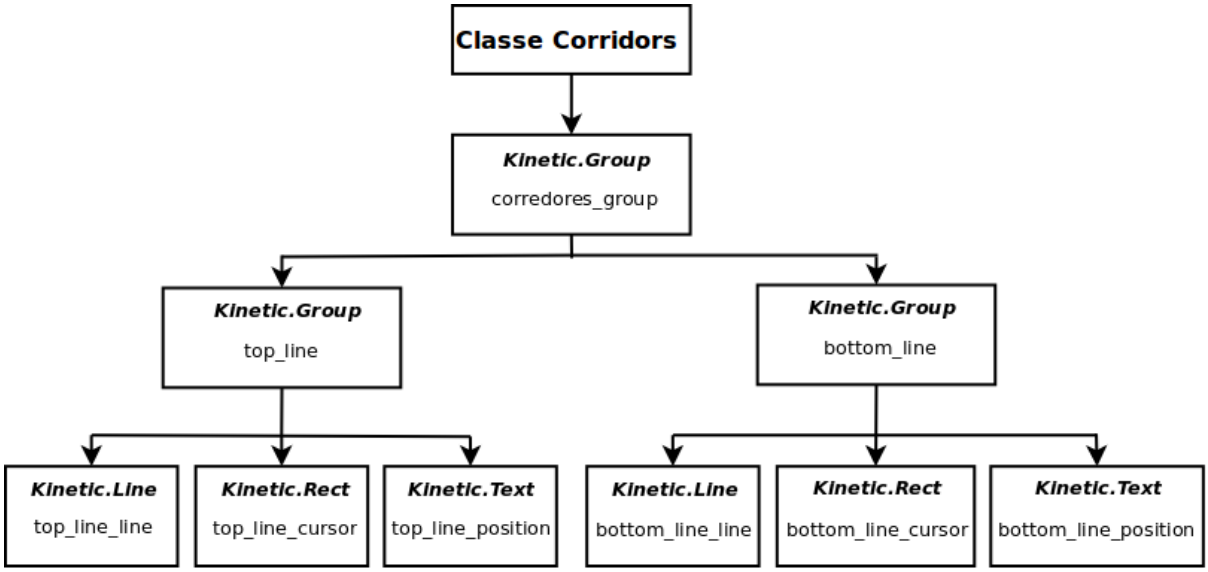


Figura 3.15: Classe *Corridors*: estrutura do elemento com o KineticJS.

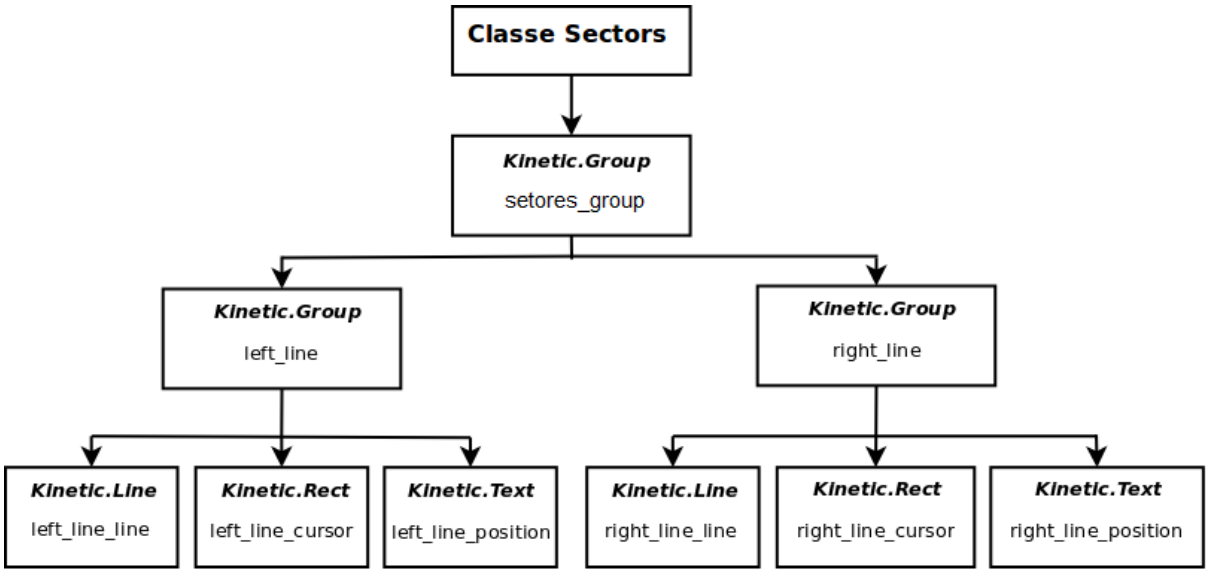


Figura 3.16: Classe *Sectors*: estrutura do elemento com o KineticJS.

g) Classe *SelectionTool* - a criação desta classe prende-se com o facto de facilitar o manuseamento de vários elementos ao mesmo tempo. Quando é instanciada, esta ferramenta gera um retângulo denominado zona de seleção (Fig. 3.17 a)), cuja dimensão é atualizada com o deslocamento do cursor, e à semelhança do que acontece em várias aplicações genéricas (Fig. 3.17 b)), todos os elementos nela contidos ficam selecionados sendo possivel efetuar ações, em simultâneo, sobre todos eles (Fig. 3.17 c)). Em termos de estrutura, esta classe é meramente constituída por uma instância da classe

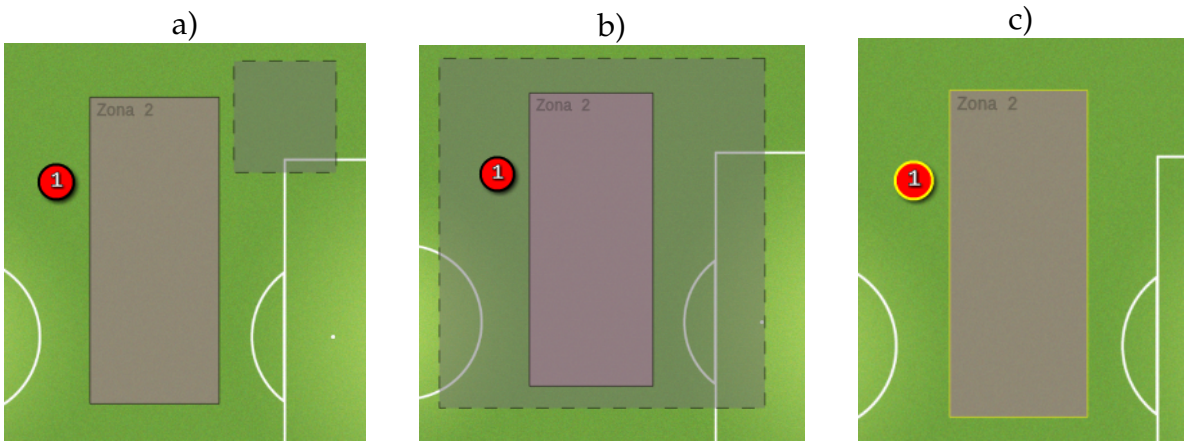


Figura 3.17: Funcionamento da classe *SelectionTool*: a) Zona de seleção à direita com o rebordo a tracejado; b) Exemplo da zona de forma a abranger os elementos que pretendemos selecionar e c) Os elementos ficam selecionados (o rebordo fica amarelo).

Tabela 3.9: Métodos da classe *SelectionTool*.

Métodos da classe *SelectionTool*	**Definição**
`checkForObjects`	Obtém os elementos que se encontram dentro da zona de seleção
`addToSelection`	Adiciona um elemento ao grupo de elementos selecionados
`removeFromSelection`	Remove um elemento do grupo de elementos selecionados

`Kinetic.Rect`.

A Tab. 3.9 apresenta os métodos implementados. Internamente, a seleção de elementos funciona da seguinte forma: imagine-se que temos um jogador e uma zona que pretendemos selecionar (como os representados na Fig. 3.17 a)). Antes de serem selecionados estes elementos encontram-se no *layer* dos objetos (Fig. 3.18 a)), todavia, após a seleção eles são movidos para dentro de um `Kinetic.Group` denominado *selected_shapes_group* gerado aquando da instanciação da classe *Field* (Fig. 3.18 b)). Sempre que se executar ações a um elemento selecionado (e.g., arrastar), faz-se uma verificação ao conteúdo deste grupo, pelo que, caso existam mais elementos dentro deste grupo, essa mesma ação é aplicada a todos esses elementos.

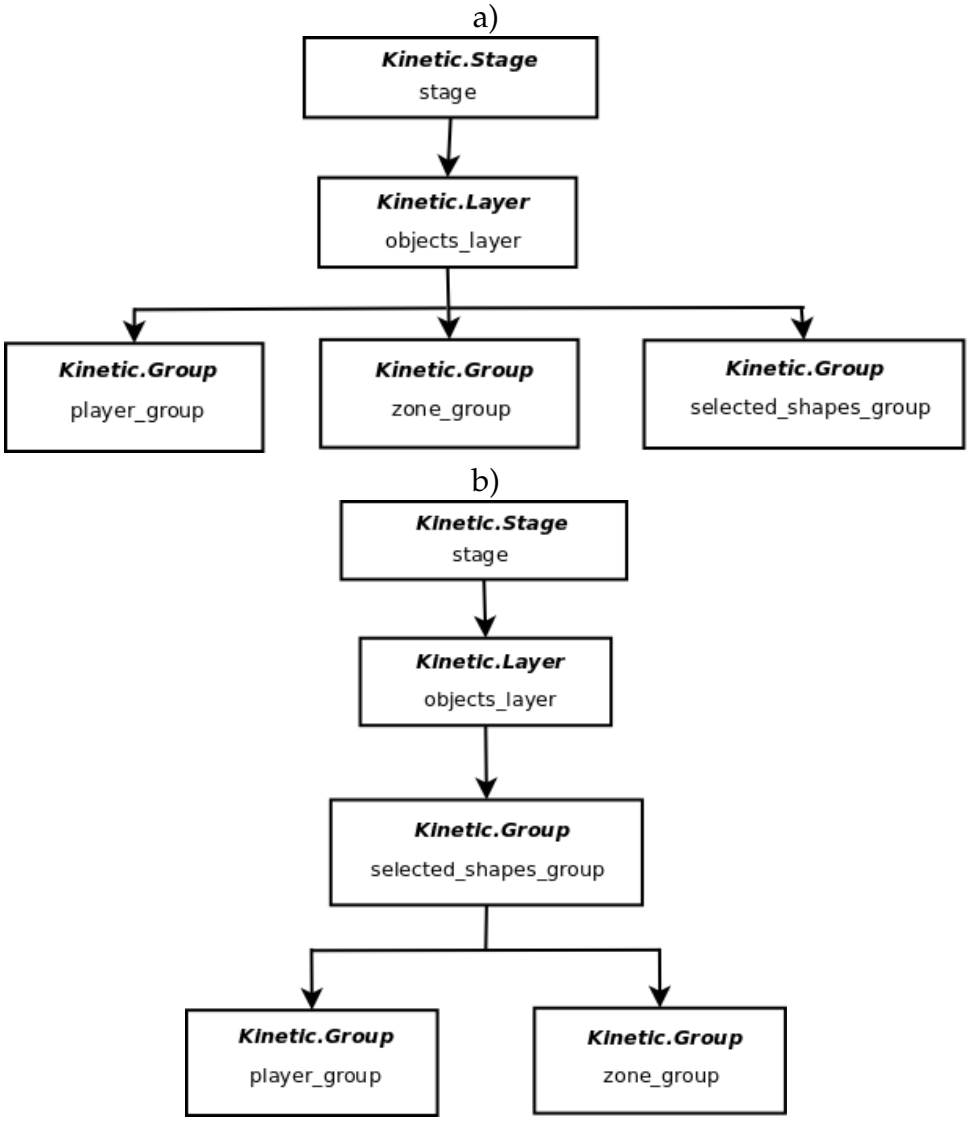


Figura 3.18: Estrutura do processo de seleção de elementos: a) Antes da seleção e b) Após a seleção.

h) Classe *DistanceLine* - esta classe foi pensada para oferecer ao utilizador do Editor de Campo uma ferramenta que permita a medição de distâncias e não é mais do que um segmento de reta entre dois pontos distintos com um texto a indicar o valor dessa distância. Quando a *DistanceLine* é instanciada, em termos visuais, é gerado um segmento de reta em que uma das extremidades segue o cursor. Quando ocorre a deslocação do cursor, tanto o segmento de reta como o texto relativo ao valor da distância são atualizados. Nesta fase, pode-se calcular a distância entre jogadores, ou pontos aleatórios do campo ou entre zonas. Contudo também é possível o cálculo de distân-

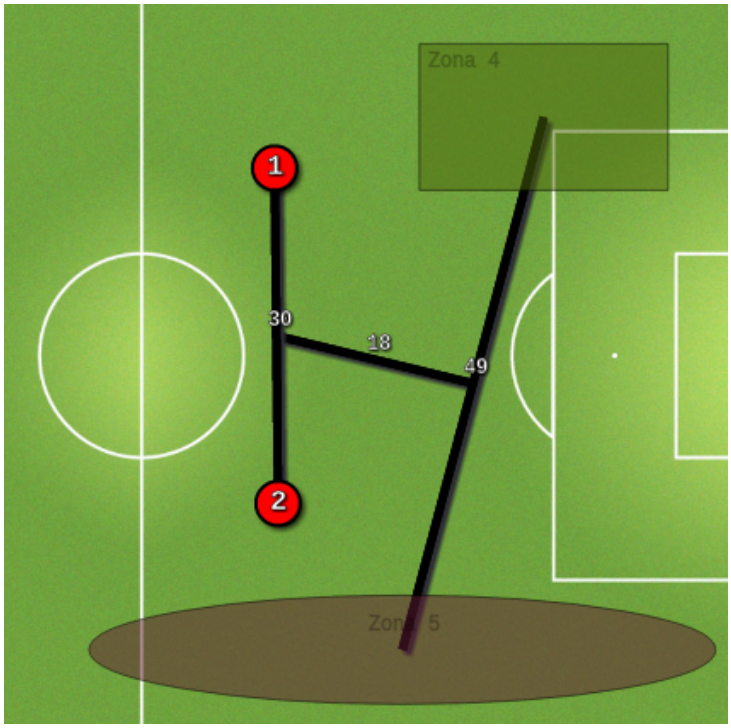


Figura 3.19: Exemplos de aplicação da classe *DistanceLine*.

cias entre elementos distintos (e.g., jogadores e zonas, ponto aleatório e um jogador, entre outros), como se pode verificar na Fig. 3.19. Devido à necessidade de criação de modelos de jogo mais complexos é possível também calcular distâncias entre pontos médios de duas outras *DistanceLine*. De forma a facilitar a utilização e melhorar a precisão, criámos uma série de pontos-alvo para a *DistanceLine*. De forma a entender esta situação, tenhamos em conta o procedimento de desenho: quando se utiliza a *DistanceLine*, tem de se efetuar um primeiro clique para estabelecer o ponto inicial e outro para o ponto final. No entanto, quer seja numa situação ou noutra, sempre que o cursor passa por um elemento específico (jogador, zona retangular, zona oval ou ainda uma *DistanceLine*) a extremidade é automaticamente colocada no centro desse elemento (no caso da *DistanceLine* é o ponto médio, como mencionado anteriormente).

Em termos de programação quando é detetada uma colisão entre a extremidade da *DistanceLine* e um elemento-alvo, altera-se as coordenadas da extremidade para o valor das coordenadas do centro do elemento-alvo com o qual ocorreu a colisão.

A estrutura desta classe encontra-se esquematizada na Fig. 3.20. Como se pode

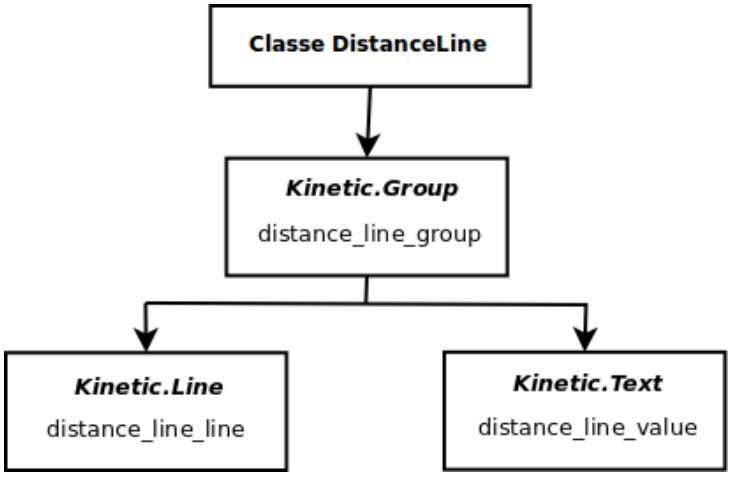


Figura 3.20: Classe *DistanceLine*: estrutura do elemento com o KineticJS.

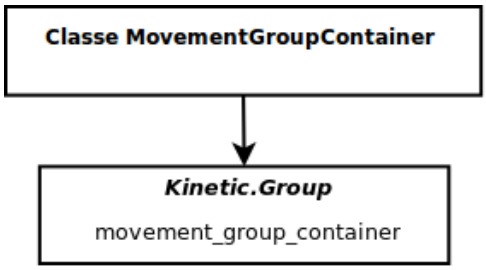


Figura 3.21: Classe *MovementGroupContainer*: estrutura do elemento com o KineticJS.

verificar, quando se instancia esta classe é gerado um `Kinetic.Group` que irá conter uma instância da classe `Kinetic.Line` para a representação do segmento de reta e ainda uma instância da classe `Kinetic.Text` que terá o objetivo de representação do valor da distância. Os métodos desenvolvidos, encontram-se resumidos e explicados na Tab. 3.10.

i) Classe *MovementGroupContainer* - como se pôde verificar em algumas das classes abordadas, como por exemplo, as classes *Player* e *Ball*, foi feita referência à classe *MovementGroupContainer*, isto porque o seu propósito é o agrupamento de movimentos dos elementos (e.g., deslocações e passes), para as classes em que está contida. A estrutura desta classe está representada na Fig. 3.21. Os métodos criados estão resumidos na Tab. 3.11.

j) Classe *Animation* - esta classe serve para estruturar a animação no Editor de Campo. Quando é instanciada, é chamado o construtor do `Kinetic.Animation` que

Tabela 3.10: Métodos da classe *DistanceLine*.

Métodos da classe *DistanceLine*	**Definição**
`getMidPoint`	Obtém o ponto médio do segmento de reta
`calculateDistance`	Calcula a distância entre os pontos inicial e final do segmento de reta
`updateDistanceLabelPosition`	Posiciona o valor da distância perto do ponto médio do segmento de reta
`setDistanceLineStartPoint`	Estabelece o ponto inicial do segmento de reta
`setDistanceLineEndPoint`	Estabelece o ponto final do segmento de reta
`updateDistanceLine`	Atualiza os pontos do segmento de reta
`updateDistanceLinesConnected`	Atualiza outras *DistanceLine* conetadas a esta
`getStartPoint`	Obtém o ponto inicial do segmento de reta
`getEndPoint`	Obtém o ponto final do segmento de reta

irá atualizar o Canvas, dando assim a ilusão de animação. Quando é invocado o método `start` a animação é iniciada, pelo que todos os elementos criados são percorridos e caso tenham movimentos associados, eles deslocar-se-ão de acordo com esse movimento durante um certo intervalo de tempo estabelecido aquando da sua criação. De seguida serão abordadas as classes representativas dos movimentos.

k) Classe *PlayerMovement* - como o nome indica, esta classe foi implementada para a representação da deslocação um jogador. É composto por um `Kinetic.Group` onde estarão contidas uma instância da classe `Kinetic.Shape` para a representação do percurso de deslocação do jogador e duas instâncias da classe `Kinetic.Text` para a representação do tempo inicial e para a duração do movimento. Os parâmetros tempo inicial e duração têm especial importância na parte da animação, pois será com base neles que se conseguirá intercalar vários movimentos de elementos diferentes,

Tabela 3.11: Métodos da classe *MovementGroupContainer*.

Métodos da classe *MovementGroupContainer*	**Definição**
`getNextStartTime`	Obtém o tempo inicial para o movimento que vai ser definido
`getNextId`	Obtém o id para o movimento que vai ser definido

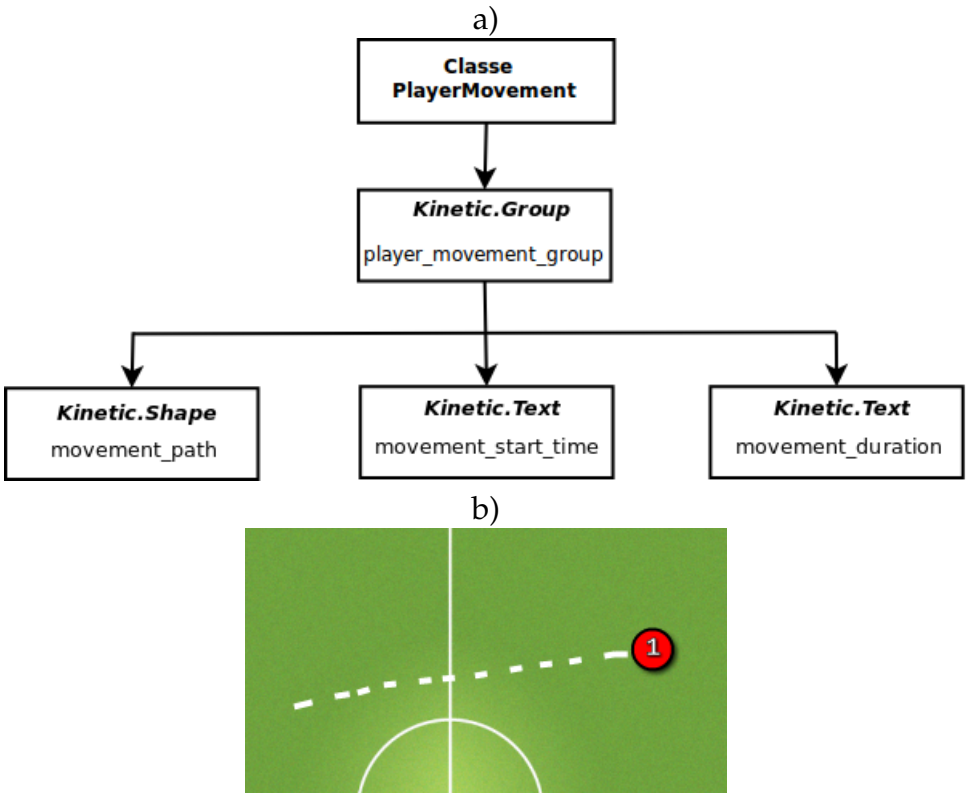


Figura 3.22: Classe *PlayerMovement*: a) Estrutura do elemento com o KineticJS e b) Exemplo de um jogador com um movimento de deslocação definido.

segundo uma base temporal comum a todos. A Fig. 3.22 a) mostra a constituição desta classe, enquanto que na mesma figura em b) encontra-se um exemplo de um jogador com um movimento de deslocação definido.

l) Classe *Kick* - esta classe visa a representação do movimento remate ou passe. Estruturalmente é igual à classe *Player Movement* (v.d. Fig. 3.22a)), diferindo somente nos atributos da instância da classe `Kinetic.Shape`, na medida em que a deslocação é

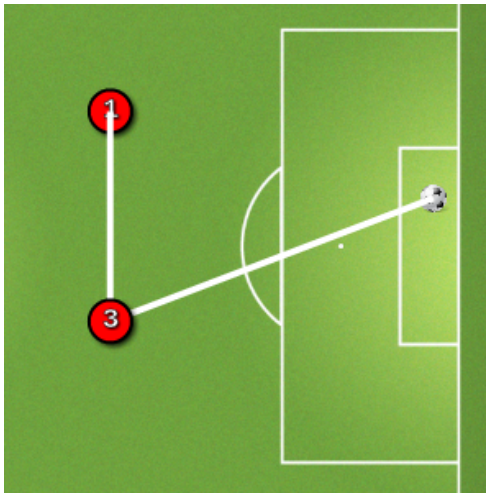


Figura 3.23: Exemplo de aplicação da classe *Kick*.

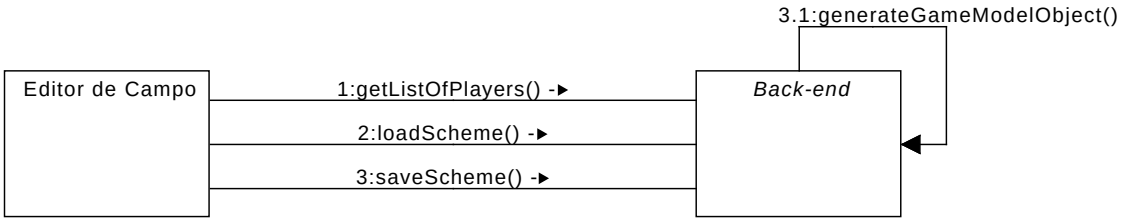


Figura 3.24: Diagrama de comunicação entre o Editor de Campo e o *back-end*.

representada a tracejado e um remate ou passe é representado com um traço contínuo. A Fig. 3.23 exemplifica um passe do jogador 1 para o jogador 3 e de um remate, deste último, para a baliza.

Estando as classes estruturadas, é importante estabelecer a ligação com o *back-end*, e alcançar uma forma plausível para guardar os modelos de jogo criados.

3.2.2 Comunicação com o *back-end*

Como referido, é importante explicar como é feita a comunicação com o *back-end* e quais os dados enviados entre ambos. A Fig. 3.24 esquematiza a comunicação entre a ferramenta *web* em questão e o *back-end*. Analisando a figura, constata-se que numa primeira fase, ocorre o envio dos dados dos jogadores que estão armazenados

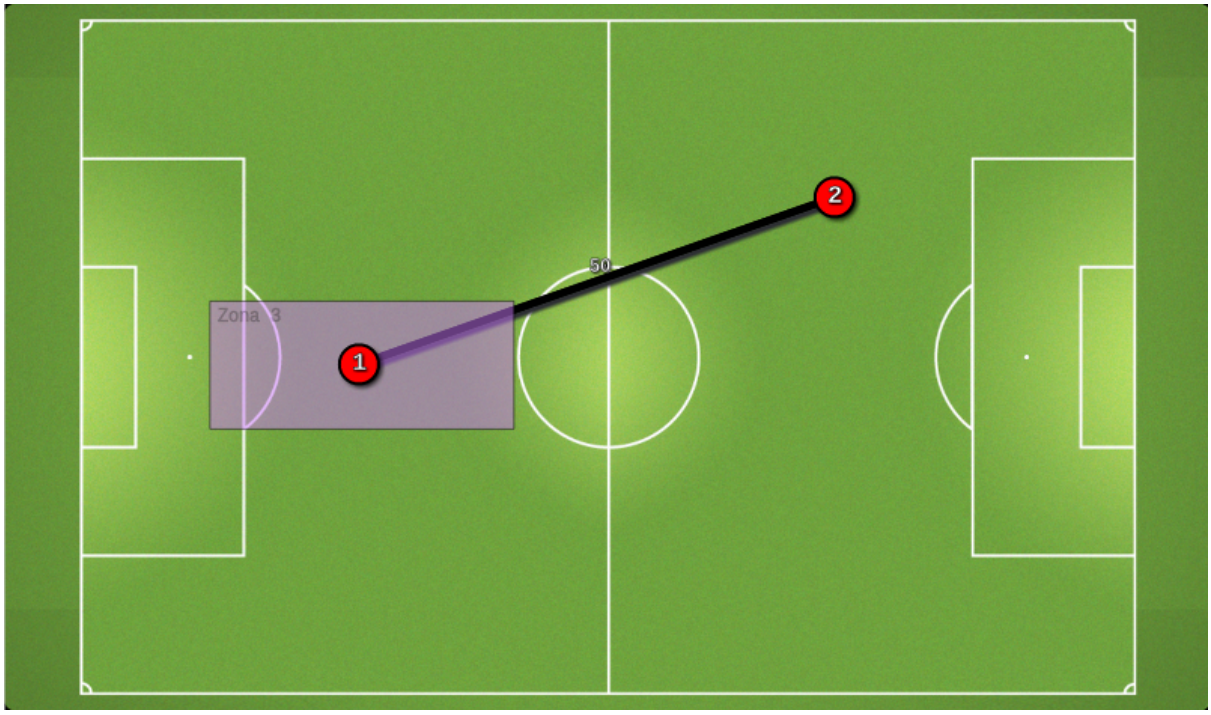


Figura 3.25: Exemplo de um esquema criado no Editor de Campo.

na base de dados para a ferramenta *web*, quando esta o solicita (`getListOfPlayers`). Isto deve-se ao facto do treinador poder querer desenhar modelos de jogo com jogadores específicos da sua equipa, em detrimento de uma jogada com jogadores em que não é necessária a sua identificação. É portanto crucial ter estas duas situações previstas. Por outro lado, o utilizador da ferramenta, para além da criação dos esquemas de jogo, poderá ainda gravá-los (`saveScheme`) ou carregá-los da base de dados (`loadScheme`). Acerca da gravação de esquemas, o KineticJS, como já foi referido permite exportar as formas criadas para JSON. Quando um esquema é gravado, o primeiro passo é convertê-lo em JSON.

A Fig. 3.25 mostra um esquema criado no Editor de Campo, o qual será convertido para um documento JSON. O método `toJSON` do KineticJS grava algumas propriedades pouco relevantes para o nosso esquema de futebol mas, fundamentais para que se possa mais tarde fazer o carregamento/importação desse esquema novamente, ou por outras palavras, fazer a sua reconstrução.

Para além do JSON gerado usando a API do KineticJS, é ainda criado e enviado

um outro documento JSON contendo informação, não sobre as caraterísticas dos elementos presentes no esquema, mas sobre propriedades do esquema de futebol em si, nomeadamente: o nome e a categoria personalizável em que se insere (e.g., Processo Ofensivo (PO), Processo Defensivo (PD), Transição Ofensiva (TO)). Há ainda a destacar outras propriedades como o tamanho real do campo, a razão entre píxeis e metros e também sobre o sistema de alertas, isto é, caso o treinador pretenda ser avisado se aquele modelo de jogo ocorrer ou não e ainda qual a mensagem que pretende receber.

Estes dois documentos JSON são então enviados, por WebSockets para o nosso servidor onde são processados e armazenados (para que mais tarde se consiga fazer o carregamento do esquema na ferramenta *web*, caso seja necessário), e é gerado aquilo a que chamamos de objeto *GameModel* (`generateGameModelObject`). O objeto *GameModel* é basicamente uma linguagem criada para que se consiga fazer uma comparação com os dados obtidos do sistema de *tracking*, aferindo então acerca do correto cumprimento dos modelos de jogo. Esse objeto, no formato JSON (v.d. Fig. 3.26), é armazenado na base de dados e, posteriormente, enviado para o algoritmo de comparação com os dados do sistema de *tracking*. A Fig. 3.27 esquematiza todo o processo até à obtenção do objeto *GameModel*.

Com uma breve análise ao documento JSON da Fig. 3.26, constata-se que existe um conjunto de atributos principais, sendo eles: *type*, *field_dimension*, *field_scale*, *players*, *ball*, *zones*, *lines* e *conditions*. Grande parte já foi explicada anteriormente, contudo existem alguns que necessitam de uma explicação mais aprofundada. O atributo *players* contém um *array* onde estão definidos outros sub-documentos. Cada um desses sub-documentos corresponde a um jogador inserido no esquema, pelo que contém informações sobre o jogador respetivo como seja o id, número, equipa, posição e o raio de ação. Visto que, para os modelos de jogo restringimos o número de bolas de futebol num esquema para uma, o atributo *ball* irá possuir só um sub-documento em vez de um *array* de sub-documentos. Todavia, no exemplo, o sub-documento relativo à bola está vazio, dado que não se colocou uma bola no esquema que se utilizou. No atributo

```
{
  "type":"PD",
  "field_dimension":[105,68],
  "field_scale":[0.15,0.15178571428571427],
  "players":[
    {"id":1,"number":1,"team":"A","points":[235.859375,239.375],"radius":3}
      ,
    {"id":2,"number":2,"team":"A","points":[551.859375,128.375],"radius":3}
  ],
  "ball":{},
  "zones":[
    {"id":3,"points":[[136.859375,197.375],[136.859375,282.375],[338.859375
      ,282.375],[338.859375,197.375]]}
  ],
  "lines":[
    {"id":4,"objects":[1,2]}
  ],
  "distances":[
    {"objects":[1,2],"distance":50,"margin":3}
  ],
  "conditions":[
    [
      {"verify_location":[1,"IN",3]},
      {"verify_distances":"true"}
    ],
    [
      {"alert":"True","msg":"A jogada especificada ocorreu!"}
    ],
    [
      {"alert":"False","msg":""}
    ]
  ]
}
```


Figura 3.26: Objeto *GameModel*.

zones, de forma semelhante ao que acontece para o atributo *players*, utiliza-se um *array* de sub-documentos em que, neste caso, cada sub-documento contém informação relativa a uma zona, quer seja poligonal, quer seja oval. Uma vez mais se utiliza o mesmo padrão para os atributos associados às *lines* e às *distances*.

Antes de prosseguir, convém referir que durante o desenho no Editor de Modelos de Jogo os segmentos de reta estavam sempre associados às distâncias (*DistanceLine*), contudo na criação do objeto *GameModel* fizemos uma separação lógica, na medida em que colocámos num lado a informação relativa ao objeto (atributo *lines*) e no outro a condição de futebol associada aos segmentos de reta que são as distâncias (atributo

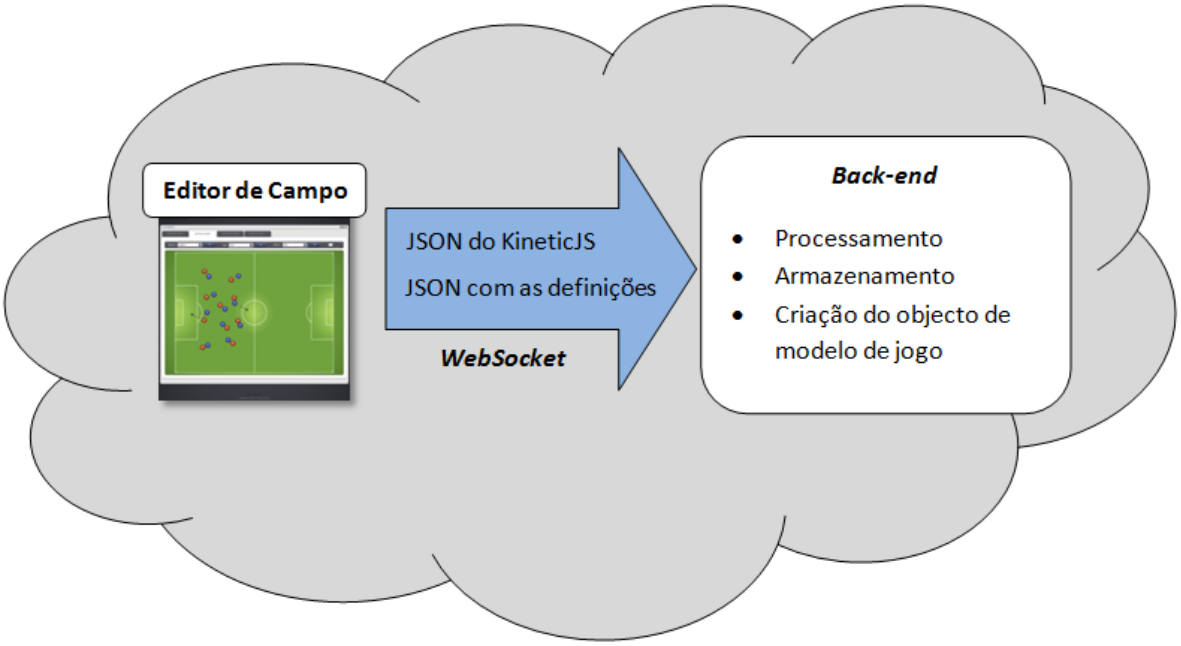


Figura 3.27: Comunicação entre o Editor de Campo e o *back-end* até à obtenção do objeto *GameModel*.

distances). Assim, a primeira possui informação relativamente a todas os segmentos de reta, nomeadamente, o seu `id` e os elementos que interligam (elementos em cada extremidade). Como se pode verificar, os elementos com `id` 1 e 2, respetivamente, estão interligados pelo segmento de reta com `id` 4 (`{"id":4,"objects":[1,2]}`). Neste esquema, são os jogadores com os números 1 e 2, respetivamente. Por sua vez, o atributo *distances*, indica que se pretende ter em conta a análise da distância entre os elementos de `id` 1 e 2, cuja distância entre si é 21 metros e que pode variar +/- 3 metros.

Por fim, resta-nos analisar o atributo *conditions*, que é constituído por três *arrays* fundamentais, satisfazendo uma estrutura *if-then-else*. O primeiro refere-se às condições a analisar, neste caso: verificar se o elemento com `id` 2 se encontra dentro da zona com `id` 3 e se as distâncias mencionadas anteriormente estão a ser cumpridas. O segundo *array* indica que o treinador pretende ser notificado com a mensagem: "A jogada especificada ocorreu!" sempre que este esquema se verificar. O terceiro e último *array* indica o que fazer caso o esquema não se verifique. Neste caso, o utilizador

pretendeu não ser notificado caso o esquema não ocorra.

Esta é a estrutura atual que temos vindo a desenvolver de acordo com os esquemas que temos trabalhado. De notar que, esta estrutura já foi ampliada várias vezes devido à necessidade de representação dos diferentes esquemas de futebol, pelo que poderá sofrer mais ajustes no futuro.

3.3 Editor de Vídeo

A análise de vídeos de futebol é, hoje em dia, um processo bastante recorrente nas equipas de topo. É de extrema importância fazer-se uma análise o mais completa possível, não só relativamente à própria equipa, como também em relação ao adversário. O Editor de Vídeo visa providenciar à equipa técnica uma ferramenta *web* virada para a edição e manipulação de vídeos.

Tendo como ponto de partida o *design* inicial do Editor de Vídeo (Fig. 3.28), elaborado pelo parceiro de projeto INESTING, começou-se a estruturação desta ferramenta. O consultor do projeto para o futebol, salientou algumas caraterísticas importantes a serem implementadas, nomeadamente: a) Maior controlo sobre o vídeo (com possibilidade de executar funções como *forward*, *backward*, *restart*, entre outros); b) Permitir desenhar no vídeo, utilizando formas simples como, por exemplo, setas, segmentos de reta, entre outras e c) Marcação e filtragem de eventos relativos a certos momentos desse vídeo.

Partindo desta base, procurou-se alcançar um primeiro protótipo desta ferramenta *web*.

3.3.1 Projeto e desenvolvimento do Editor de Vídeo

Tendo sido feita uma breve abordagem ao HTML5 Vídeo na Secção 2.7, é importante relembrar que utilizar HTML5 Vídeo e Canvas em simultâneo permite o alcance de novas funcionalidades e efeitos, mudando a experiência do utilizador na visualização

Figura 3.28: *Design* inicial do Editor de Vídeo.

de vídeos na *web* (Powers, 2011; Fulton and Fulton, 2011; Meyer, 2011). Este facto a juntar ao que nós já sabíamos ser possível alcançar com o Canvas (no Editor de Campo), permitiu-nos encarar a realidade HTML5 Vídeo + Canvas como uma solução legítima para alguns dos desafios que surgiram na implementação desta ferramenta de vídeo.

Começando pelo reprodutor de vídeo em si, é verdade que os *browsers* recentes têm o seu próprio reprodutor de HTML5 Vídeo com as funções mais básicas como,

por exemplo, *play*, *pause* e ajuste do som. No entanto, em situações em que se pretende mais funcionalidades, é comum desenvolvermos os próprios controlos para o vídeo. Além do mais, se desenvolvermos o nosso próprio reprodutor de vídeo podemos ainda estilizá-lo da forma que melhor entendermos. Funções como *restart*, *forward* e *backward* de 1 e 5 segundos não são comuns nos reprodutores de vídeo usuais, contudo, para análise de vídeos de futebol, são ferramentas com utilidade sobretudo se for necessário conferir um certa jogada várias vezes, por exemplo.

Após estarmos na posse de um reprodutor de vídeo mais complexo, centrámos as atenções na possibilidade do utilizador desenhar sobre *frames* do vídeo. De facto, este é um exemplo típico de que uma solução HTML5 Vídeo + Canvas permite um enriquecimento na forma como é disponibilizada média na *web*. Na prática o utilizador está a desenhar num Canvas de fundo transparente, sobreposto no vídeo, dando a ideia de que está a desenhar diretamente no reprodutor de vídeo.

Para esta componente foram úteis algumas das ferramentas que se implementaram no Editor de Campo, na medida em que se consegue fazer uma adaptação fácil para o Editor de Vídeo. Objetos como círculos, setas, desenho livre entre outros são facilmente personalizáveis e implementados nesta ferramenta.

No que diz respeito à marcação de eventos, como foi referido, os consultores indicaram que seria fundamental o utilizador poder capturar certos momentos (*frames*) e também secções de vídeo, denominados eventos, em que fosse possível agrupá-los e categorizá-los. Chegámos a um consenso em que a divisão dos eventos fosse feita segundo:

Zona - na medida em que o utilizador pode especificar a zona do campo em que ocorreu aquele evento. A Fig. 3.28 mostra uma possível divisão do campo em zonas (4×4);

Equipa - podendo-se associar o evento a uma ou às duas equipas;

Jogador - há semelhança do que acontece com as equipas, também se pode fazer uma

associação dos eventos aos jogadores;

Categoria e sub-categoria - pois existe a necessidade de se agrupar os eventos segundo categorias específicas do futebol personalizadas pelo utilizador (por exemplo, categoria "Processo Ofensivo", sub-categoria "Passe de Rutura").

A associação destes parâmetros na marcação de eventos é crucial para que, aquando da análise, se consiga fazer uma filtragem mais concisa. Imagine-se que temos um vídeo em que marcámos a ocorrência de cem eventos. Se não tivéssemos associado esses eventos aos parâmetros mencionados, seria um processo demoroso procurar por um evento em específico. Assim, consegue-se fazer uma filtragem de eventos de acordo com os parâmetros estabelecidos, facilitando a análise de vídeos.

Ainda acerca dos eventos, foi-nos sugerida a implementação de caixas de texto no vídeo, de forma a que o utilizador pudesse ter uma referência textual acerca dos eventos que ia marcando. Um dos módulos oferecidos pelo Popcorn.js (Popcorn, 2013) é o Footnote, que permite mostrar texto no vídeo, ao longo da reprodução quando é atingido um certo instante. Basicamente, é possível especificar o tempo inicial em que o texto deve ser mostrado e durante quanto tempo pretendemos que o texto esteja visível.

Por fim, há que referir que estamos ainda a estudar qual a melhor forma de gravar no vídeo não só os desenhos efetuados, como também o texto introduzido por meio da definição de eventos.

3.3.2 Comunicação com o *back-end*

Relativamente à comunicação entre o Editor de Vídeo e o *back-end* (ver Fig. 3.29), pretendemos fazer um estudo mais aprofundado acerca dos dados a processar e a armazenar nas bases de dados. No momento de escrita desta dissertação, temos previsto que no Editor de Vídeo, o utilizador comece por solicitar o carregamento de um determinado vídeo que se encontra armazenado na base de dados (`loadVideo`). Caso existam

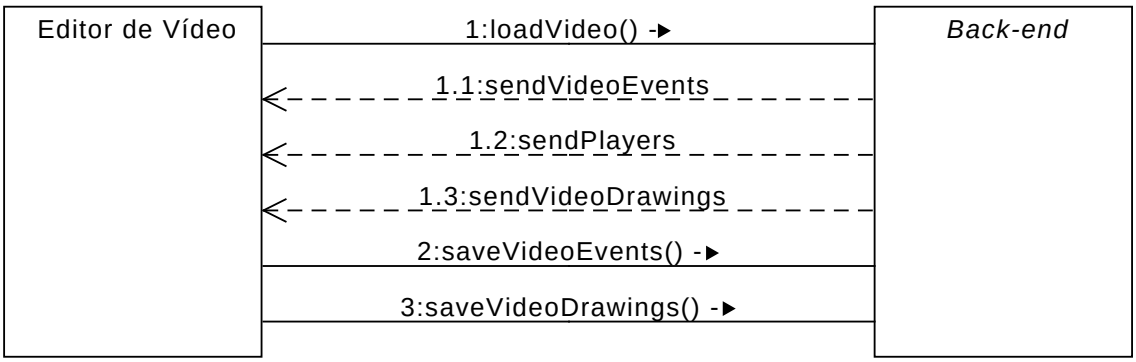


Figura 3.29: Diagrama de comunicação entre o Editor de Vídeo e o *back-end*.

desenhos ou eventos associados a esse vídeo, eles também serão enviados pelo *back-end* (`sendVideoDrawings` e `sendVideoEvents`, respetivamente). Nesse sentido, a lista de jogadores envolvidos nesse jogo também terá de ser enviada para a ferramenta *web* (`sendPlayers`).

Dado que o utilizador pode, por exemplo, inserir novos eventos no vídeo, ou representar uma certa movimentação no vídeo através do desenho de uma forma (seta, por exemplo), temos de providenciar um mecanismo de guardar esses dados, tanto ao nível da marcação de eventos (`saveVideoEvents`) como do desenho de formas no vídeo (`saveVideoDrawings`).

No momento da escrita desta dissertação, o que está definido é que os vídeos originais ficarão num servidor de conteúdos estáticos e referenciados na base de dados, assim como todos os instantes de início e de fim de eventos e respetiva classificação. Os "desenhos" sobre o vídeo também serão guardados e carregados sempre que o respetivo vídeo for solicitado.

Por conseguinte, iremos procurar abordar esta comunicação de forma semelhante à elaborada para o Editor de Campo, na medida em que se envia a informação no formato JSON via WebSockets para o servidor, sendo de seguida processada e armazenada para futura utilização.

3.4 Editor de *Tracking*

Com o Editor de *Tracking* pretende-se dar ao utilizador uma plataforma de apoio ao sistema de *tracking* do projeto FootData (Sousa, 2012). De uma forma genérica, o sistema de *tracking*, desenvolvido em C++, deteta a posição dos jogadores e da bola a partir das imagens que são recebidas de uma única câmara fixa montada nas bancadas do estádio ou a partir de imagens proveninentes de vídeos adquiridos e/ou transmissões televisivas. Em qualquer dos casos, o sistema de *tracking* pode falhar na identificação dos jogadores e bola quando, por exemplo, os jogadores se obstruem na imagem adquirida pela câmara, ou quando o jogo decorre sob condições meteorológicas adversas (e.g., nevoeiro). Além disso, quando aplicado em vídeos de transmissões televisivas, podem novamente ocorrer identificações incorretas, quando por exemplo ocorrem *zooms*. Esta ferramenta *web*, visa a resolução deste tipo de problemas, oferecendo uma plataforma para ajustamento e correção de eventuais identificações incorretas por parte do sistema de *tracking*. Todo este processo de correção de *tracking* encontra-se esquematizado na Fig. 3.30.

De notar que, ainda não existe um *design* em concreto para esta ferramenta, pelo que, para efeitos de melhor entendimento dos propósitos desta aplicação, sugere-se a consulta da Fig. 3.31, que apresenta uma implementação inicial do Editor de *Tracking*, contemplando algumas das funcionalidades já descritas.

Dadas as diferentes fontes de imagens para o sistema de *tracking*, esta ferramenta irá ter dois modos de funcionamento distintos: a) Edição de *Tracking* de Campo e b) Edição de *Tracking* de Vídeo. Existem algumas diferenças nestes dois modos, como se irá analisar de seguida.

3.4.1 Projeto e desenvolvimento do Editor de *Tracking*

No caso da **Edição de *Tracking* de Campo**, o sistema de *tracking* processa as imagens recebidas da câmara enquanto o jogo de futebol decorre, enviando-as com os joga-

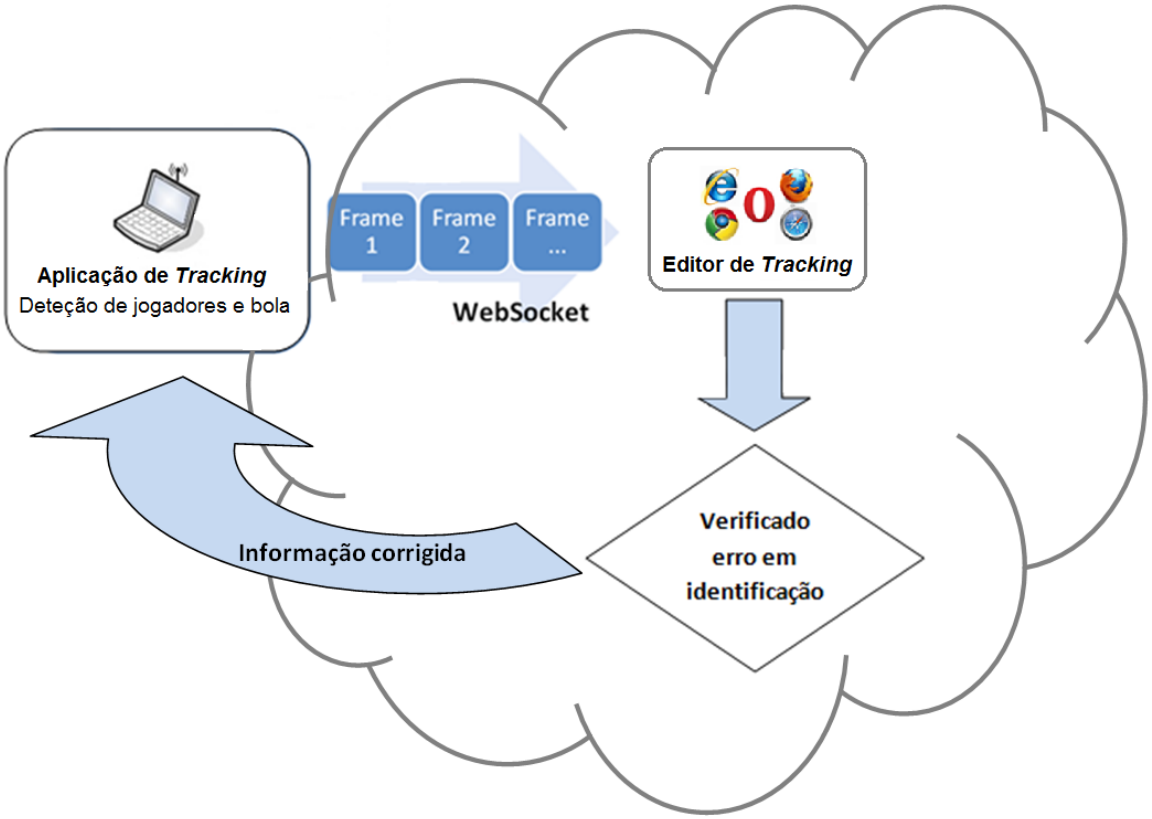


Figura 3.30: Processo de correção do *tracking*.

dores e a bola identificados, via WebSockets, para o cliente (neste caso o *browser*). A Fig. 3.32 mostra o exemplo de uma dessas *frames* onde se constata a deteção dos jogadores e da bola num jogo entre Belenenses e Sporting. A implementação do sistema de *tracking* não faz parte dos objetivos desta dissertação, para obter detalhes relativos a tal implementação, sugere-se a consulta de (Sousa, 2012). No entanto, para a compreensão da ferramenta, é necessária uma breve introdução, assim, como se pode ver na Fig. 3.32, os jogadores detetados possuem um retângulo em seu redor: jogadores do Belenenses possuem um retângulo azul, os do Sporting têm um retângulo verde e, por fim, a bola com um círculo preto. Neste caso, qualquer outro retângulo que não seja verde ou azul indica que o objeto identificado não tem qualquer influência no jogo ou então pode indicar conflito numa identificação. Perto de cada jogador encontra-se uma letra que indica a equipa ("B" indica Belenenses e "S" indica Sporting) e um nú-

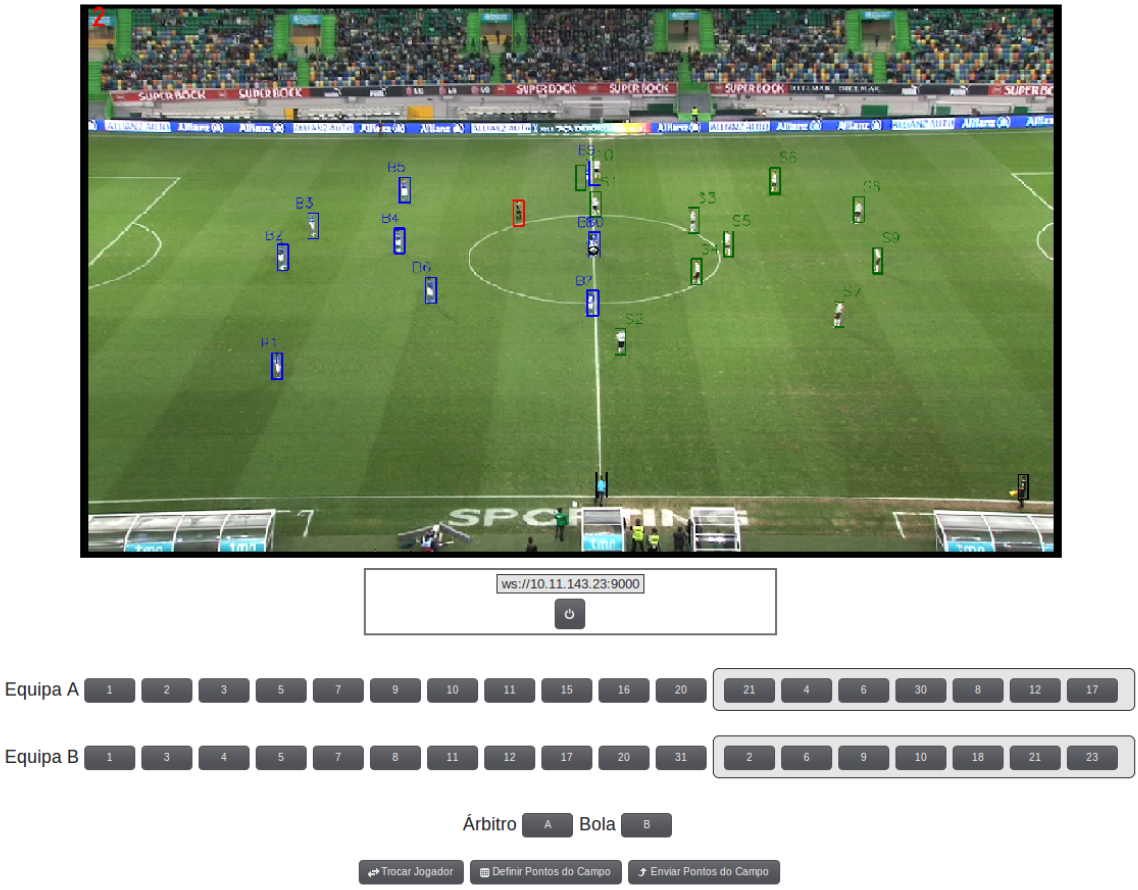


Figura 3.31: Ambiente do Editor de *Tracking*.

mero identificativo para cada jogador. De notar que, neste momento, este número em questão não corresponde ao número do jogador dentro da equipa, sendo meramente para fins de indicação e correção.

Um utilizador que se encontre a trabalhar nesta ferramenta (Fig. 3.31), cada vez que detete algum erro nas identificações, basta carregar nos botões correspondentes aos números dos jogadores trocados para proceder à correção da identificação. Internamente, o que acontece é que a informação relativa aos jogadores a serem trocados é enviada novamente por WebSockets para a aplicação de *Tracking*, que por sua vez irá proceder às alterações necessárias por forma a que nas *frames* seguintes a enviar, a identificação esteja feita de acordo com o corrigido pelo utilizador da ferramenta *web*.

Em termos de tecnologias, utilizou-se o HTML5 Canvas para mostrar as imagens

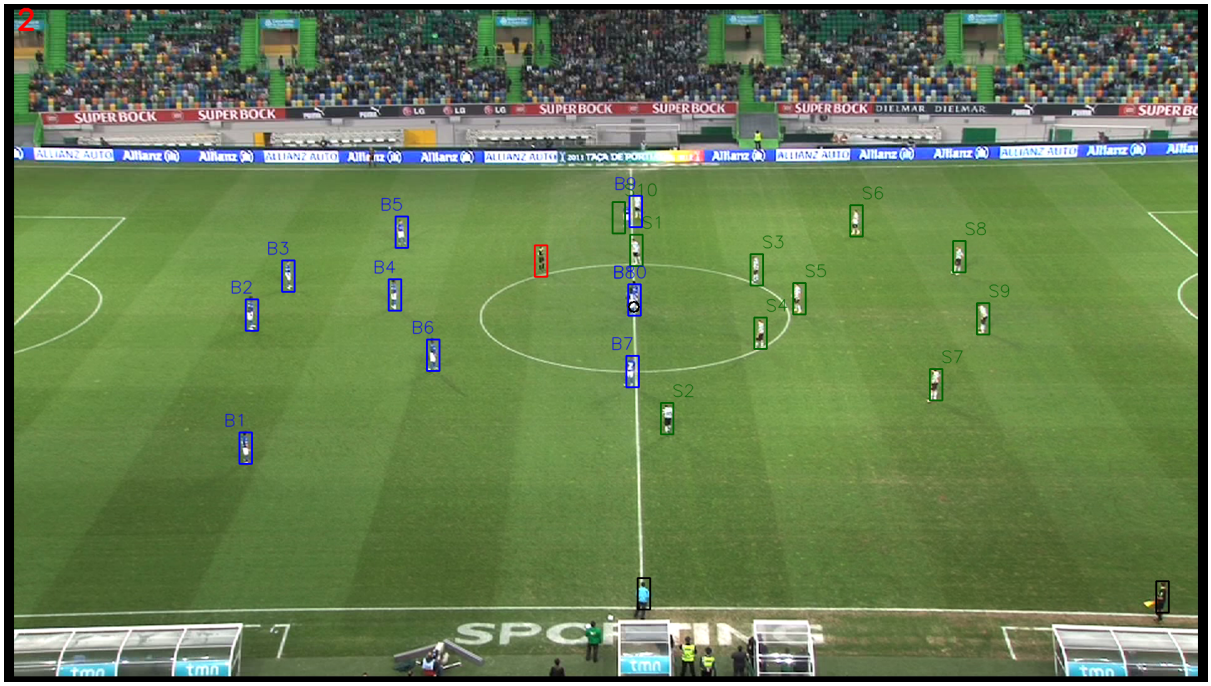


Figura 3.32: Exemplo de uma *frame* enviada pelo sistema de *tracking* para o Editor de *Tracking*.

enviadas pela aplicação de *tracking* por WebSockets, à medida que se recebem as novas *frames*. A velocidade com que é atualizado o Canvas, provoca no utilizador a sensação de que está a visualizar um vídeo. Visto que o sistema de *tracking* foi desenvolvido em C++, decidimos também utilizar uma implementação em C++ de WebSockets denominada WebSocket++ (WebSocket, 2013). Neste caso, o sistema de *tracking* desempenha o papel de servidor na comunicação por *socket* enquanto que o *browser* representa o cliente.

No caso da **Edição de *Tracking* de Vídeo**, a principal diferença relativamente ao anterior reside no facto de que o utilizador pode aplicar o sistema de *tracking* em vídeos de jogos de futebol que já decorreram. Esses vídeos podem ser de gravações da equipa técnica ou de transmissões televisivas, apresentando por norma pior qualidade do que os vídeos abordados anteriormente. Embora seja verdade que este último tipo de vídeo possua várias secções pouco interessantes para o *tracking*, como *zooms*, repetições, anúncios publicitários e outros planos de filmagem externos ao jogo de futebol em si, os consultores do projeto na área do futebol alertaram-nos para o facto de existir

uma grande quantidade de vídeos de futebol com interesse à disposição das equipas técnicas nesse formato (televisivo). Neste tipo de edição, o utilizador poderá escolher um vídeo de um conjunto de vídeos de jogos de futebol, dispondo igualmente das ferramentas para correção do *tracking* abordadas na Edição de *Tracking* de Campo.

De seguida abordar-se-ão alguns aspetos relativos à comunicação entre as entidades envolvidas no processo de *tracking*, isto é, o Editor de *Tracking*, a aplicação de *tracking* e o *back-end*.

3.4.2 Comunicação com o *back-end* e com a aplicação de *tracking*

A comunicação entre o Editor de *Tracking*, o *back-end* e a aplicação de *tracking* para os modos de edição de *tracking* de campo e vídeo encontram-se representados nas Figs. 3.33 e 3.34, respetivamente.

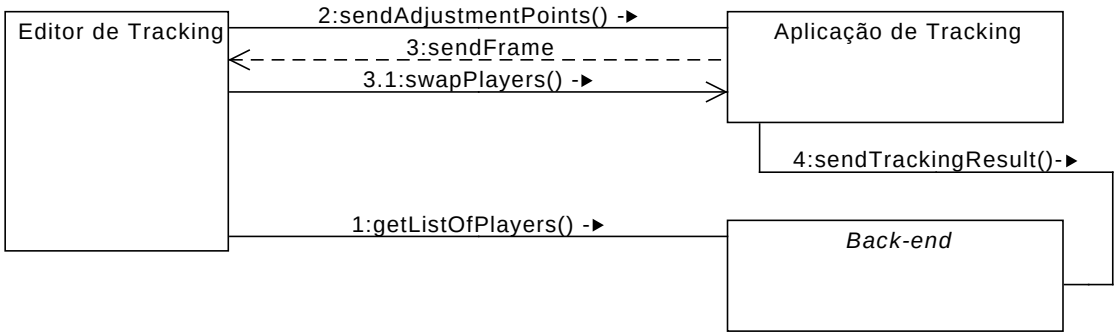


Figura 3.33: Diagrama de comunicação entre o Editor de *Tracking*, o *back-end* e a aplicação de *tracking*, na edição de *tracking* de campo.

Para que se consiga efetuar correções em possíveis identificações dos jogadores é necessário conhecer-se as convocatórias das duas equipas para o jogo em análise, de forma a se apresentar na interface os jogadores (nome e número). A informação acerca das convocatórias é enviada pelo *back-end* quando solicitado pelo Editor de *Tracking* (`getListOfPlayers`). Na edição de *tracking* de campo, começa-se por definir no Editor de *Tracking* uma "janela" retangular, cujo objetivo é enquadrar somente a área impor-

tante para o *tracking* (área do campo de futebol, procurando excluir as bancadas). Essa "janela" é definida por quatro pontos, que irão ser enviados para a aplicação de *tracking* (`sendAdjustmentPoints`).

Iniciado o processo de *tracking*, a aplicação de *tracking* irá estar a detetar os jogadores e a bola e a enviar *frame* a *frame* para o Editor de *Tracking* (`sendFrame`). Quando o utilizador deteta uma identificação incorreta e efetua a sua correção essa informação é enviada para a aplicação de *tracking* (`swapPlayers`).

Concluído este processo, a aplicação de *tracking* envia para o *back-end* toda a informação resultante da deteção realizada (`sendTrackingResult`).

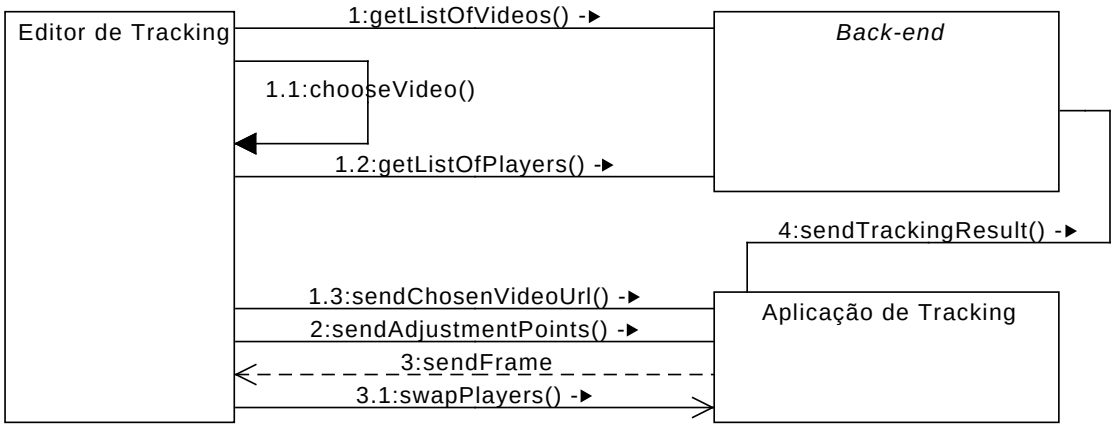


Figura 3.34: Diagrama de comunicação entre o Editor de *Tracking*, o *back-end* e a aplicação de *tracking*, na edição de *tracking* de vídeo.

Quanto à edição de *tracking* de vídeo, é necessário receber do *back-end* a informação que diz respeito aos vídeos disponíveis para se realizar o processo de *tracking*. Essa informação é enviada após solicitação do Editor de *Tracking* (`getListOfVideos`). O utilizador terá a oportunidade de escolher um vídeo de entre os que se encontram referenciados no *back-end* (`chooseVideo`), e de seguida solicita-se ao *back-end* a convocatória das equipas envolvidas no jogo constante nesse vídeo (`getListOfPlayers`). Escolhido o vídeo e estando na posse das convocatórias das duas equipas para esse jogo, envia-se para a aplicação de *tracking* o endereço do vídeo no servidor (`sendChosenVideoUrl`).

O restante processo de estabelecimento e correção do *tracking* é exatamente igual ao mencionado na edição de *tracking* de campo.

Tanto na edição de *tracking* de campo como de vídeo, o resultado final não é mais que um documento no formato JSON com a identificação dos jogadores e bola em cada *frame*. É esta informação que irá ser comparada com o objeto *GameModel* referido na secção 3.2, numa plataforma específica para o efeito.

3.5 Editor de Apresentações

Nesta última ferramenta (das que fazem parte do âmbito desta dissertação), procurámos oferecer ao treinador uma plataforma versátil que facilitasse o seu trabalho na realização de apresentações e preleções aos jogadores. Basicamente, pretendemos que esta ferramenta permita um acesso fácil e rápido a uma série de recursos (vídeos, imagens, ficheiros, entre outros) que o treinador pretenda mostrar como forma de reforçar e alcançar um melhor entendimento das suas ideias no seio do plantel. Atualmente, ainda não dispomos de um *design* para esta ferramenta, pelo que a sua implementação ainda não foi feita.

No entanto, em todas as discussões do consórcio, em termos de projeto da aplicação foi sempre referido que esta ferramenta seria construída a partir das ferramentas já desenvolvidas para o Editor de Campo e Vídeo, acrescentando um Editor de Texto. Foi ainda definido no consórcio que o Editor de Apresentações poderia ser utilizado em conjunto com o sistema de projeção eBeam (Imagina, 2013). Este sistema, que simula um quadro digital interativo, já foi montado, testado e integrado na plataforma FootData.

3.6 Testes e resultados

Nesta secção apresentamos e discutimos alguns dos resultados obtidos, através de um conjunto de imagens relativas aos protótipos das ferramentas que se tem vindo a desenvolver e também com comparações e análises a outros produtos já existentes no mercado. Todavia, convém salientar que não foi possível testar em concreto outras aplicações no mercado semelhantes aos Editores de Vídeo e de *Tracking* devido à inexistência de versões de teste abertas ao público em geral. Uma vez que a sua aquisição também não era solução, procurou-se fazer comparações tendo como base a documentação e os vídeos de demonstração disponibilizados pelas entidades envolvidas na criação dessas aplicações.

Devemos ainda referir que todas as ferramentas desenvolvidas encontram-se neste momento, a serem convertidas para o *design* final da aplicação (ver Secção 3.7) e, só nesse momento é que irão ser exaustivamente testadas pelos utilizadores finais. Assim, os resultados apresentados e os comparativos serão feitos com a versão "alfa" dos nossos aplicativos.

3.6.1 Editor de Campo

A Fig. 3.35 a) apresenta o ambiente de desenho do Editor de Campo. Como se pode constatar, dispomos à esquerda de um campo de futebol genérico onde o utilizador poderá desenhar as suas jogadas, e à direita um menu onde se pode aceder às diversas funcionalidades do Editor de Campo, das quais se destacam: a) Gravação e carregamento de esquemas; b) Inserção de elementos; c) Definição dos alertas, corredores e setores e d) Controlo das animações.

De forma a comparar-se o Editor de Campo com outras aplicações já existentes no mercado, avaliou-se o módulo *K-PlayMaker* (K-PlayMaker, 2013) da aplicação Kizanaro (Fig. 3.35 b)), onde se incidiu sobre vários aspetos como: o ambiente de desenho, os elementos disponíveis para desenho de um esquema, as propriedades do esquema,

a)

b)

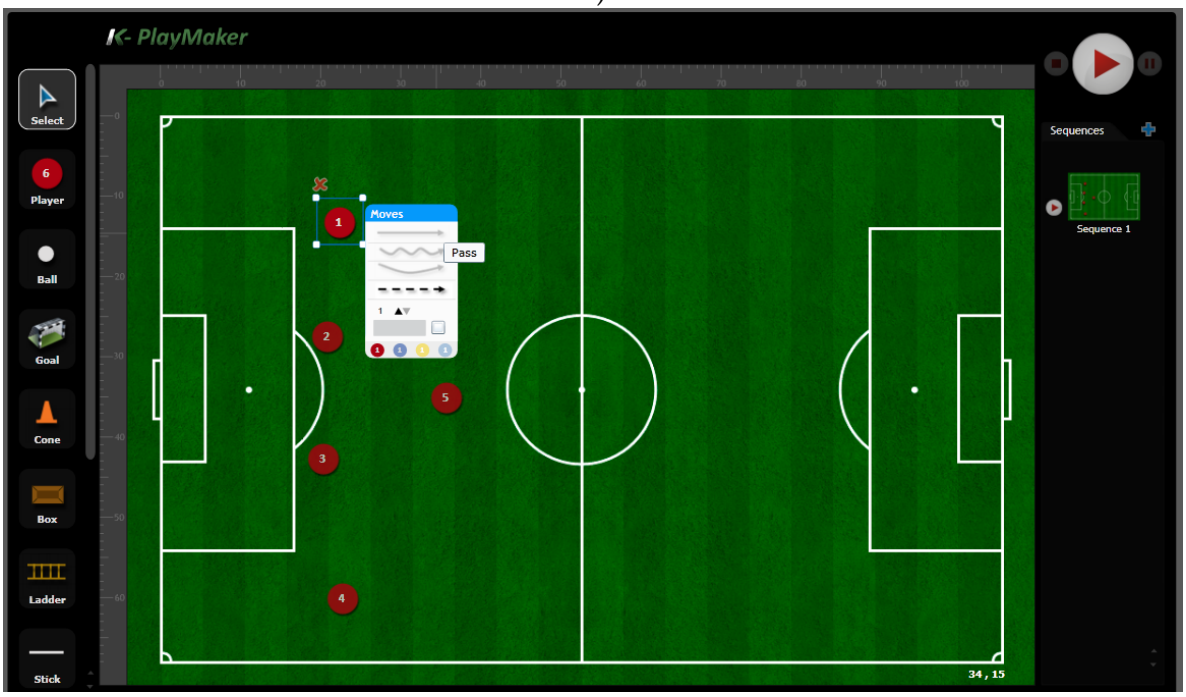
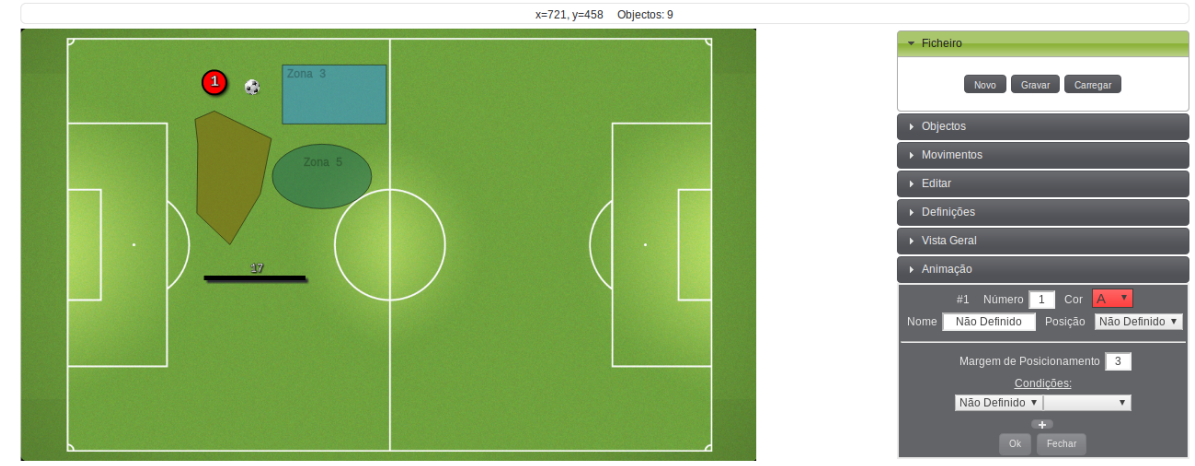


Figura 3.35: Ambientes de desenho: a) No Editor de Campo e b) No *K-PlayMaker*.

e por fim, a componente de animação. Na Fig. 3.36 a), pode-se verificar quais os elementos possíveis de se utilizar no Editor de Campo e na Fig. 3.36 b), mostramos não só a definição de corredores e setores, como também as opções disponíveis para cada esquema. É ainda destacada na Fig. 3.37 a), a representação de uma simples animação: dois passes entre três jogadores e por fim um remate à baliza. Na mesma figura em baixo (Fig. 3.37 b)), exemplificamos a definição de um modelo de jogo mais complexo,

a)



b)

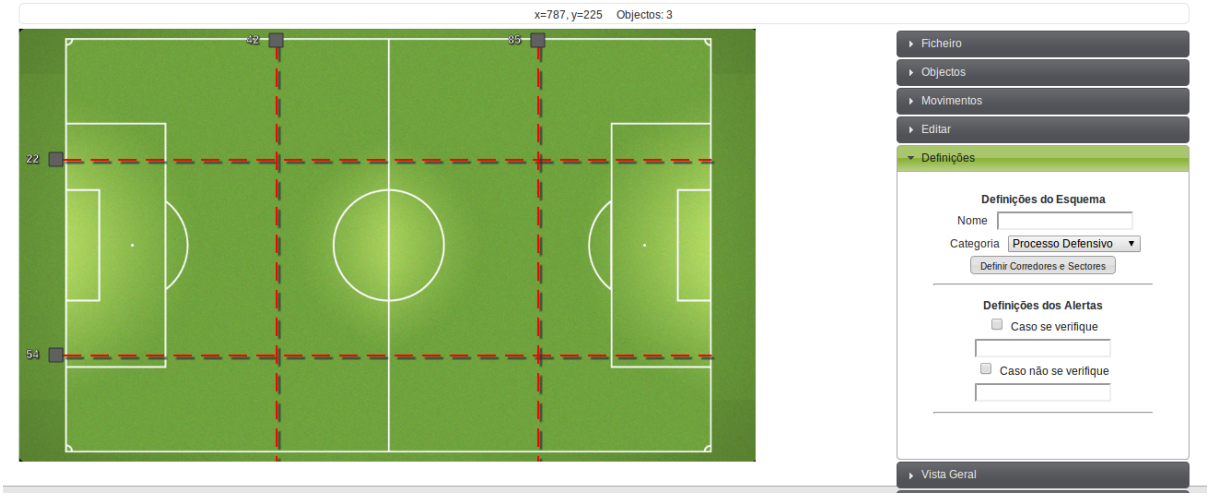


Figura 3.36: Editor de Campo: a) Elementos disponíveis e b) Definição de corredores e setores.

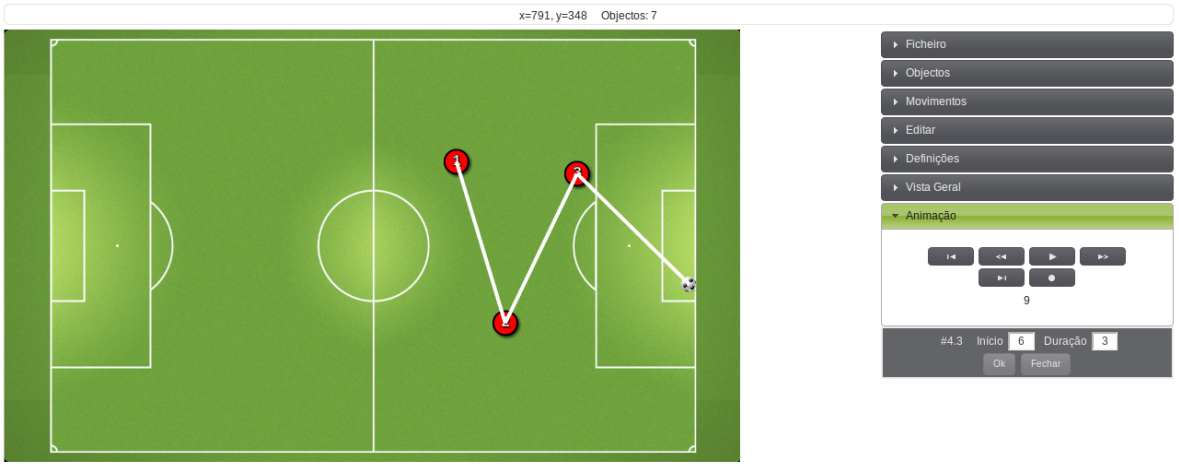
abrangendo não só uma movimentação, como distâncias entre elementos e zonas.

Além das figuras relativas ao Editor de Campo, são apresentados exemplos da aplicação *K-PlayMaker*, no que diz respeito às mesmas funcionalidades. Assim, na Fig. 3.38 a), são dados a conhecer os elementos utilizados nesta aplicação e na Fig. 3.38 b) é exemplificada uma animação no *K-PlayMaker*.

Com base no estudo efetuado tiraram-se as seguintes conclusões:

1. Quando se pretendeu aceder ao *K-PlayMaker* uma limitação encontrada foi o facto de ter sido desenvolvido com o Microsoft Silverlight, pelo que foi necessário a instalação de um *plugin* para conseguir executar a aplicação. No caso do

a)



b)

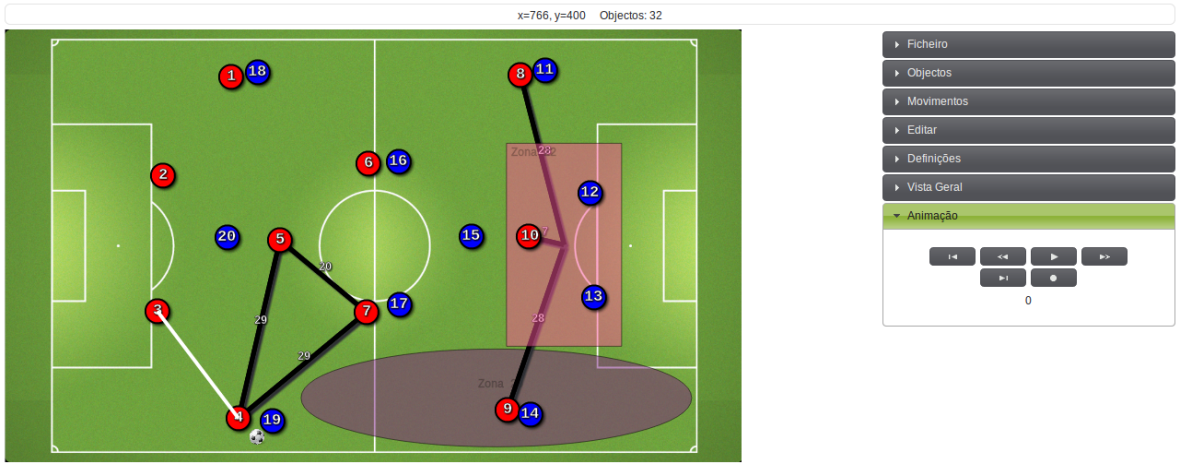


Figura 3.37: Editor de Campo: a) Definição de animações simples e b) Definição de modelos de jogo mais complexos.

Editor de Campo, visto que foi estruturado com o HTML5 Canvas, pode ser acedido através dos principais *browsers* em diferentes sistemas operativos sem ser necessária a instalação de quaisquer *plugins*.

2. O ambiente de desenho do *K-PlayMaker* (Fig. 3.35 b)), embora seja algo diferente do Editor de Campo (Fig. 3.35 a)) no que diz respeito à disposição dos diferentes componentes à volta do campo, em termos de funcionalidade e interatividade é bastante semelhante, pois para se inserir um elemento no campo basta clicar no botão respetivo e de seguida clicar no campo no local onde se pretende que o elemento em questão fique posicionado, podendo-se também interagir com ele

a)

b)

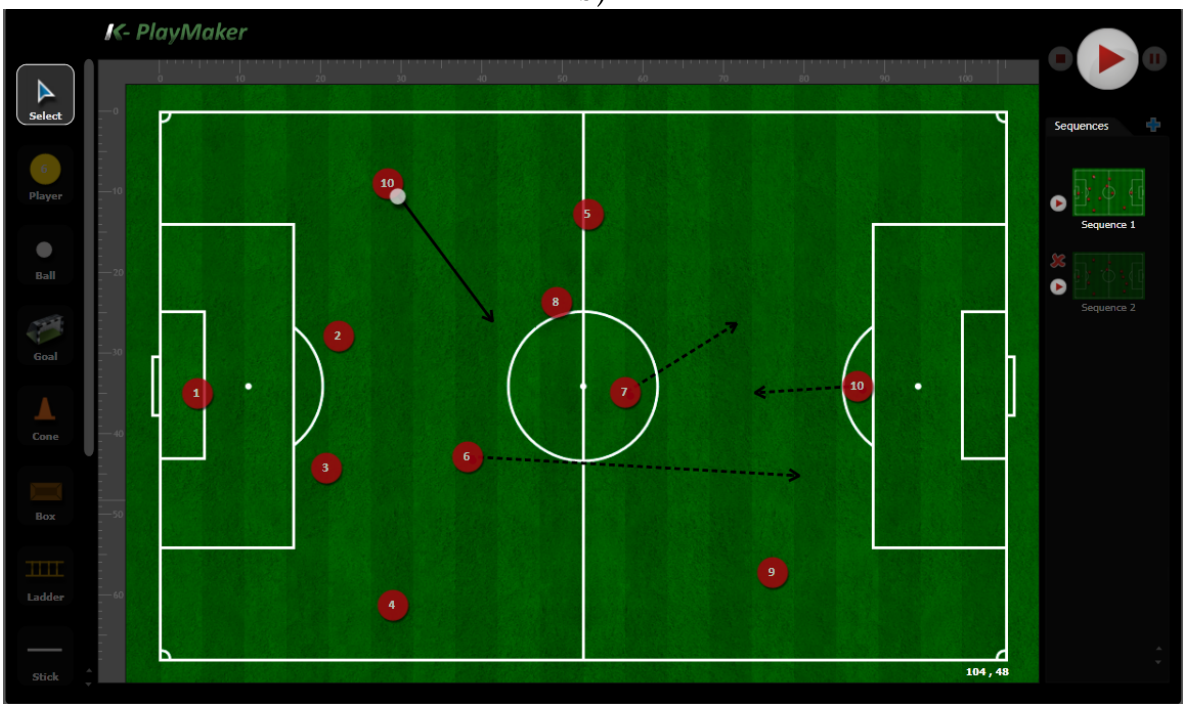


Figura 3.38: *K-PlayMaker*: a) Elementos disponíveis e b) Definição de animações.

através de cliques e arrastamentos.

3. O *K-PlayMaker* dispõe nesta fase de mais elementos que o Editor de Campo (Figs. 3.36 e 3.38), contudo temos prevista a implementação dos elementos em

falta num futuro próximo.

4. No Editor de Campo é possível estabelecer zonas retangulares, quaisquer zonas poligonais personalizáveis e ainda zonas ovais, enquanto que no *K-PlayMaker* só é possível a criação de zonas retangulares.

5. A ferramenta de criação de distâncias do *K-PlayMaker* permite saber a distância entre qualquer ponto, contudo não é tão precisa como a do Editor de Campo, visto que, se se pretender medir a distância entre dois jogadores, caso o utilizador não posicione o cursor exactamente no centro dos jogadores para a definição da distância, ela nunca irá ser exata. No Editor de Campo, é detetada a colisão do segmento de reta da distância com os elementos, encaminhando automaticamente a extremidade desse segmento de reta para o centro desse elemento, permitindo uma definição da distância com maior exatidão.

6. No que diz respeito aos movimentos, o *K-PlayMaker* dispõe de mais duas variantes de movimentos para além dos que existem no Editor de Campo, nomeadamente os passes com trajetória curvilínea e a deslocação em slalom, contudo também esperamos codificar esses movimentos brevemente.

7. Há semelhança do que acontece no *K-PlayMaker*, também no Editor de Campo se consegue alterar parâmetros como a cor, número e nome de um jogador. Todavia pretendemos dar ao utilizador mais formas de representação do elemento jogador (por exemplo, um ícone de um jogador visto de cima em vez de um círculo).

8. O processo de atribuição de posse de bola a um jogador funciona de forma semelhante ao Editor de Campo, na medida em que, após se colocar o jogador em campo, basta somente clicar e arrastar a bola até ao jogador de forma a que ele fique reconhecido como o portador de bola.

9. A descrição do esquema no *K-PlayMaker* (Fig. 3.38) revela-se algo limitada em

relação ao Editor de Campo (Fig. 3.36 b)), pois neste último damos a hipótese de categorização do esquema segundo categorias personalizáveis pelo treinador de forma a melhorar a organização e facilitar futuras pesquisas e filtragens.

10. O *K-PlayMaker* possui um sistema de animação (Fig. 3.38) por sequências bastante prático de forma a se dividir um movimento complexo em movimentos mais simples. No Editor de Campo a componente de animação (ver Fig. 3.37 a)) não permite ainda tal funcionalidade, contudo consegue-se estabelecer o tempo inicial para o começo da animação e ainda a duração de cada movimento, sendo que tal aspeto não é possível no *K-PlayMaker*.

11. Enquanto que no *K-PlayMaker* nós estabelecemos os pontos inicial e final do movimento, no Editor de Campo o movimento é gerado através do arrastar do elemento.

A Tab. 3.12 sintetiza o estudo efetuado acerca destas duas aplicações, destacando os parâmetros mais importantes para a sua comparação.

3.6.2 Editor de Vídeo

As Figs. 3.39 a) e b) apresentam o ambiente do Editor de Vídeo. Com uma breve análise às figuras mencionadas, verifica-se que esta ferramenta é composta por duas partes principais: i) O reprodutor de vídeos, com diversas funções para edição e manipulação dos vídeos e ii) A parte onde estão contidos os eventos, na qual se pode filtrá-los segundo os parâmetros já analisados na Secção 3.3.

Tendo em vista a validação do conceito do Editor de Vídeo implementado, pretendíamos averiguar como funcionava uma das aplicações mais usadas na área da análise de vídeos, denominada *Sportstec SportsCode* (SportsCode, 2013), v.d. Fig. 3.40. A Fig. 3.41 a) mostra a criação de um evento no Editor de Vídeo, com base numa secção temporal do mesmo e em que aspetos podemos categorizar esse evento. A mesma figura em baixo (Fig. 3.41 b)), representa novamente a criação de um evento, mas desta

Tabela 3.12: Resumo do estudo comparativo entre o Editor de Campo e o *K-PlayMaker*.

Parâmetro	**Editor de Campo**	***K-PlayMaker***
Tipo de aplicação	*Web*	*Web*
Dependência de *plugins*	Não	Sim, o Microsoft Silverlight
Número de elementos	7	10
Zonas disponíveis	Qualquer zona poligonal personalizável, retangular e oval	Retangular
Estabelecimento de distâncias	Sim, bastante preciso	Sim, pouco preciso
Movimentos disponíveis	Deslocações com e sem bola, passes retilíneos e remates	Deslocações com e sem bola, deslocações em slalom, passes retílineos e curvilíneos e remates
Descrição dos esquemas	Mais detalhada	Menos detalhada
Componente de animação	Definida através do arrastamento dos elementos	Definida através da especificação de movimentos segundo uma série de sequências

vez, associado a uma *frame* do vídeo. Ainda para o Editor de Vídeo, na Fig. 3.42 exemplificamos a funcionalidade de desenho no vídeo, onde se constata a representação de algumas formas. Por fim, a Fig. 3.43 mostra como se marcam eventos e se desenha no vídeo, na aplicação *Sportstec SportsCode*.

Contudo, devido ao facto de nos ser impossível testar a referida aplicação, pelos motivos mencionados no início do presente capítulo, tivémos de basear a nossa análise somente em vídeos existentes na *web* sobre a aplicação em causa. Assim, elaborou-se o seguinte estudo comparativo:

1. O *Sportstec SportsCode*, visto ser uma aplicação nativa, exige uma prévia instalação no computador do utilizador de forma a ser utilizada. O Editor de Vídeo visto ser uma aplicação *web*, não necessita de instalação, sendo apenas necessário

a)

b)

Figura 3.39: Ambiente do Editor de Vídeo: a) Reprodutor de vídeo e b) Zona de filtragem de eventos.

um *browser* com acesso à rede.

2. Constatou-se que o *Sportstec SportsCode* apresenta algumas limitações nos sistemas operativos sobre os quais é executado, pois é uma aplicação nativa exclusiva para *OS X*.

3. Ambas permitem carregar e reproduzir vídeos e também possuem as funções gerais que um reprodutor de vídeo comum apresenta (e.g., *play* e *pause*), com a inclusão de outras tantas (e.g., *forward* e *backward*) para melhorar a experiência

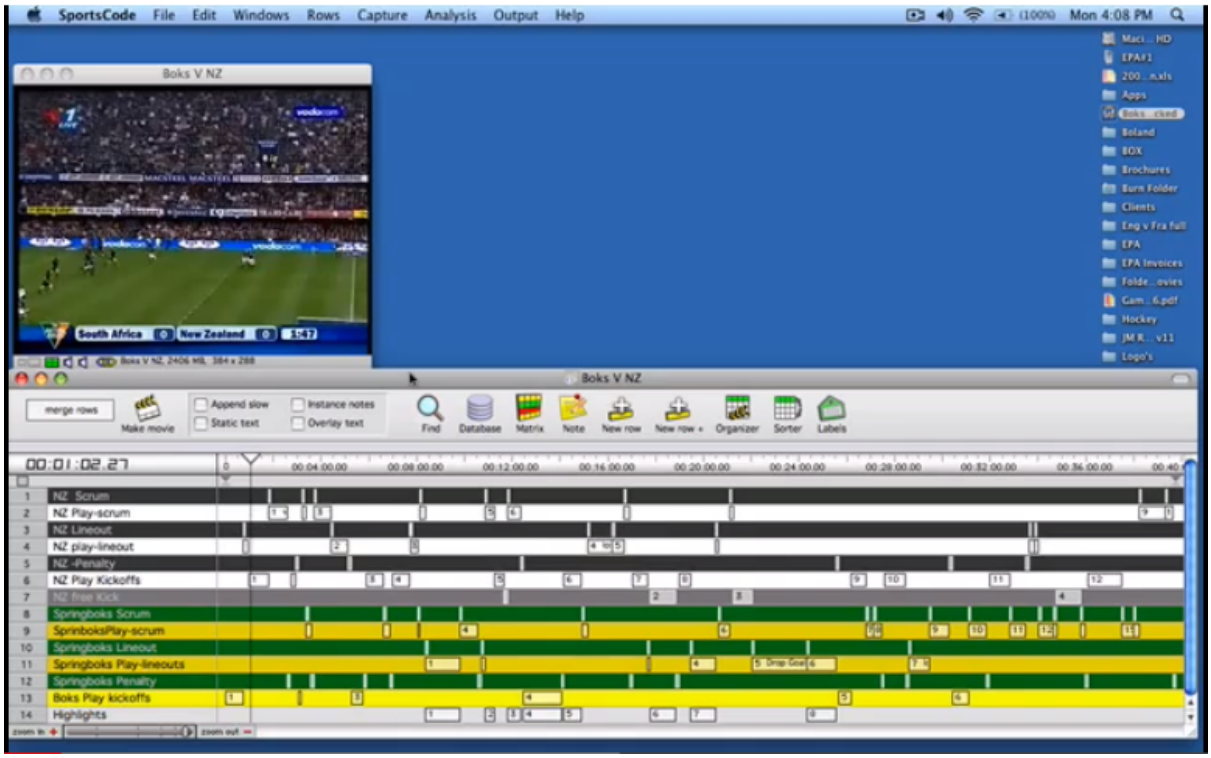


Figura 3.40: Ambiente de edição do *Sportstec SportsCode*.

do utilizador.

4. Um inconveniente que existe no Editor de Vídeo está relacionado com os formatos, como foi discutido anteriormente. Pode haver necessidade de se ter dois formatos diferentes de vídeo para o mesmo vídeo, visto que não existe um formato único que seja atualmente suportado em todos os *browsers*. De forma contrária, o *Sportstec SportsCode* não se encontra limitado nesta componente, permitindo abrir os formatos mais comuns de vídeo.

5. O *Sportstec SportsCode* possui um ambiente para edição de vídeos bastante complexo, podendo ser aplicado não só para o futebol como para outros desportos. O Editor de Vídeo também partilha dessa mesma caraterística, todavia a plataforma na qual se insere (projeto FootData) foi exclusivamente pensada para o futebol.

6. Por um lado, o *Sportstec SportsCode* permite a criação e categorização de eventos

a)

b)

Figura 3.41: Editor de Vídeo: a) Marcação de um evento relativamente a uma secção do vídeo e b) Marcação de um evento relativamente a uma *frame* do vídeo.

Figura 3.42: Desenho de algumas formas no Editor de Vídeo.

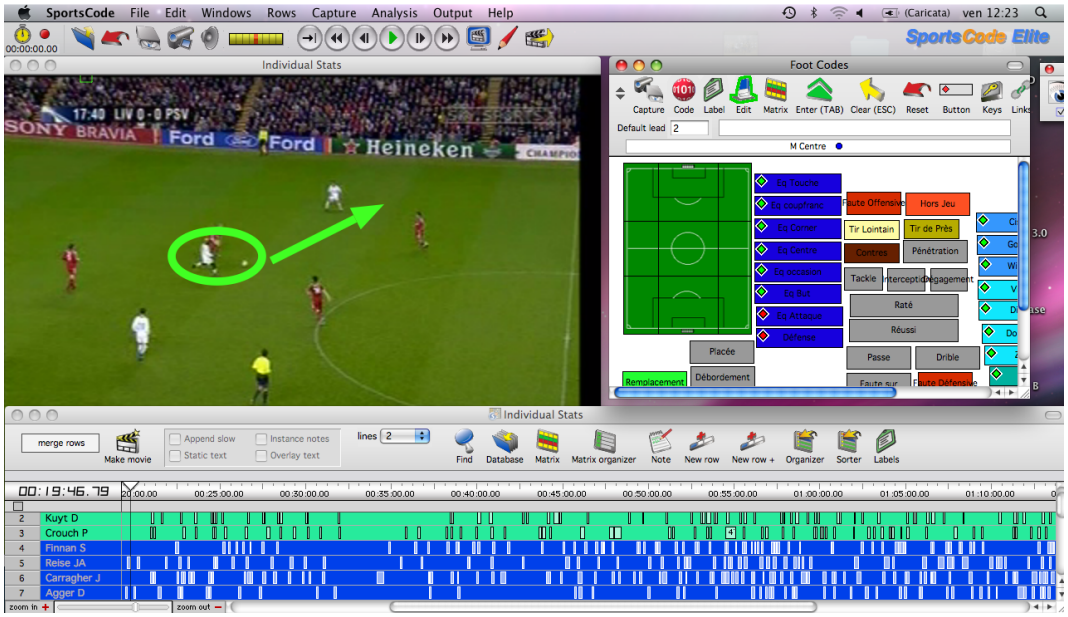


Figura 3.43: Marcação de eventos e desenho de algumas formas no *Sportstec SportsCode*.

num vídeo de forma a se conseguir fazer uma separação inteligente de diferentes momentos-chave, para posterior visualização, filtragem e análise (Fig. 3.43). Por outro lado, com o Editor de Vídeo é possível não só marcar certas secções de um vídeo (Fig. 3.41), como também capturar só uma *frame*. Ainda no Editor de Vídeo, também se pode fazer a categorização das secções e das *frames*, e uma vez mais, há semelhança do que aconteceu no Editor de Campo, também as categorias serão personalizáveis pelo utilizador.

7. Após a criação dos eventos, ambas as ferramentas permitem a sua filtragem, melhorando a organização e facilitando o acesso a um evento em particular.

8. Tanto o Editor de Vídeo, como o *Sportstec SportsCode*, permitem ao utilizador desenhar formas (e.g., setas, segmentos de reta, entre outros) em cima do vídeo (Figs. 3.42 e 3.43, respetivamente).

A Tab. 3.13 resume o estudo efetuado sobre estas duas aplicações, incidindo principalmente nos parâmetros que considerámos fundamentais para a sua comparação.

3.6.3 Editor de *Tracking* e Apresentações

Relativamente ao Editor de *Tracking* nenhuma comparação foi possível fazer, pois os *softwares* existentes são na sua maioria extremamente caros ou proprietários de apenas alguns clubes ou empresas, ou simplesmente não são comparáveis. No entanto, para efeitos de ilustração, na Fig. 3.31 apresentamos o ambiente de correção de *tracking*, onde se constata a existência de uma série de botões para que se consiga proceder à correção de qualquer tipo de identifição automática incorreta que possa eventualmente surgir.

Por último, falta referir o comparativo do Editor de Apresentações. Este não foi realizado porque a ferramenta ainda não foi desenvolvida, de qualquer forma não

Tabela 3.13: Resumo do estudo comparativo entre o Editor de Vídeo e o *Sportstec SportsCode*.

Parâmetro	**Editor de Vídeo**	***Sportstec SportsCode***
Tipo de aplicação	*Web*	Nativa
Dependência do sistema operativo	Não	Sim, *OS X*
Formatos de vídeo suportados	Está dependente dos formatos suportados pelo *browser*: MP4, OGG e WebM	A maioria dos formatos de vídeo
Versatilidade para outros desportos	Aplicável principalmente para o futebol	Bastante versátil
Marcação de eventos	Sim	Sim
Categorização de eventos	Sim	Sim
Filtragem de eventos	Sim	Sim
Desenho de formas	Sim	Sim

fazia sentido comparar este aplicativo com o *software* que os treinadores usam tradicionalmente, o *Microsoft Powerpoint*, uma vez que o que propomos, será uma ferramenta integrada que bebe diretamente de todo o sistema criando apresentações semiautomáticas, enquanto que o *Powerpoint* é uma ferramenta genérica para apresentações.

3.6.4 Breve discussão

Com o estudo efetuado, pudemos tirar várias conclusões e mais facilmente identificar as lacunas presentes no nosso trabalho e outros aspetos que devem ser melhorados. Novos *designs* foram produzidos pelo parceiro INESTING, a usabilidade foi repensada nalgumas ferramentas e outras funcionalidades foram projetadas de forma a colmatar alguns défices verificados. Nesse sentido, na secção seguinte ir-se-á abordar alguns tópicos acerca da produção final das ferramentas.

3.7 Produção final das ferramentas

Após o nosso estudo e teste das tecnologias necessárias de forma a levar a cabo a implementação das ferramentas (editores), bem como da validação do conceito (protótipos, versões "alfa"), apoiada pela comparação com outros produtos no mercado (v.d. Secção 3.6), foi-nos proposto pela empresa INESTING um novo *design* para cada uma das ferramentas, bem como um conjunto de funcionalidades (e usabilidade) mais adequadas aos consumidores finais, treinadores e sua equipa técnica. Nesta secção vamos apresentar esses *designs* bem como fazer algumas descrições necessárias para a sua compreensão.

3.7.1 *Design*

O *design* da interface de uma aplicação e o primeiro contato que o utilizador tem com ela são aspetos muito importantes, podendo fazer a diferença entre o sucesso e o insucesso da aplicação. Como já foi mencionado, o *design* da versão "alfa" (protótipo) era algo que tinhamos a noção que tinha de ser melhorado e repensado nalgumas situações.

O parceiro de projeto INESTING, responsável pelo *design* de todos os módulos do projeto FootData, produziu vários estudos, estabelecendo cores, fontes e os ícones a utilizar em cada uma destas ferramentas. As Figs. 3.44, 3.45 e 3.46 ilustram, a título de exemplo, três das interfaces a implementar para a versão final.

No Secção 3.2 foi referida a necessidade de se fazer uma divisão no Editor de Campo em três componentes: Editor de Jogadas de Treino, Editor de Modelos de Jogo e Editor de Tácticas. A Fig. 3.44 apresenta o ambiente de trabalho no Editor de Modelos de Jogo, onde se constata a presença dos elementos já analisados e de mais alguns que concluímos, em conjunto com os consultores, serem importantes para uma definição mais precisa de um modelo de jogo. Na Fig. 3.45, é mostrado na mesma figura o estudo do modo de precisão para os utilizadores (equipa técnica) poderem represen-

Figura 3.44: *Design* final do Editor de Modelos de Jogo (ambiente de trabalho).

tar os elementos com precisão (v.d. Secção 3.2, a explicação do Editor de Modelos de Jogo). No que diz respeito ao Editor de Vídeo, representado na Fig. 3.46, verifica-se que também será necessária a implementação do *design* e de algumas funcionalidades importantes, mas que, não difere em grande parte daquilo já ponderado até agora na respetiva versão "alfa".

3.7.2 Implementação

A fase de implementação da versão final de uma aplicação *web* é sempre uma altura crítica em que se tem de ter diversos factos em conta, pois após a conclusão da codif-

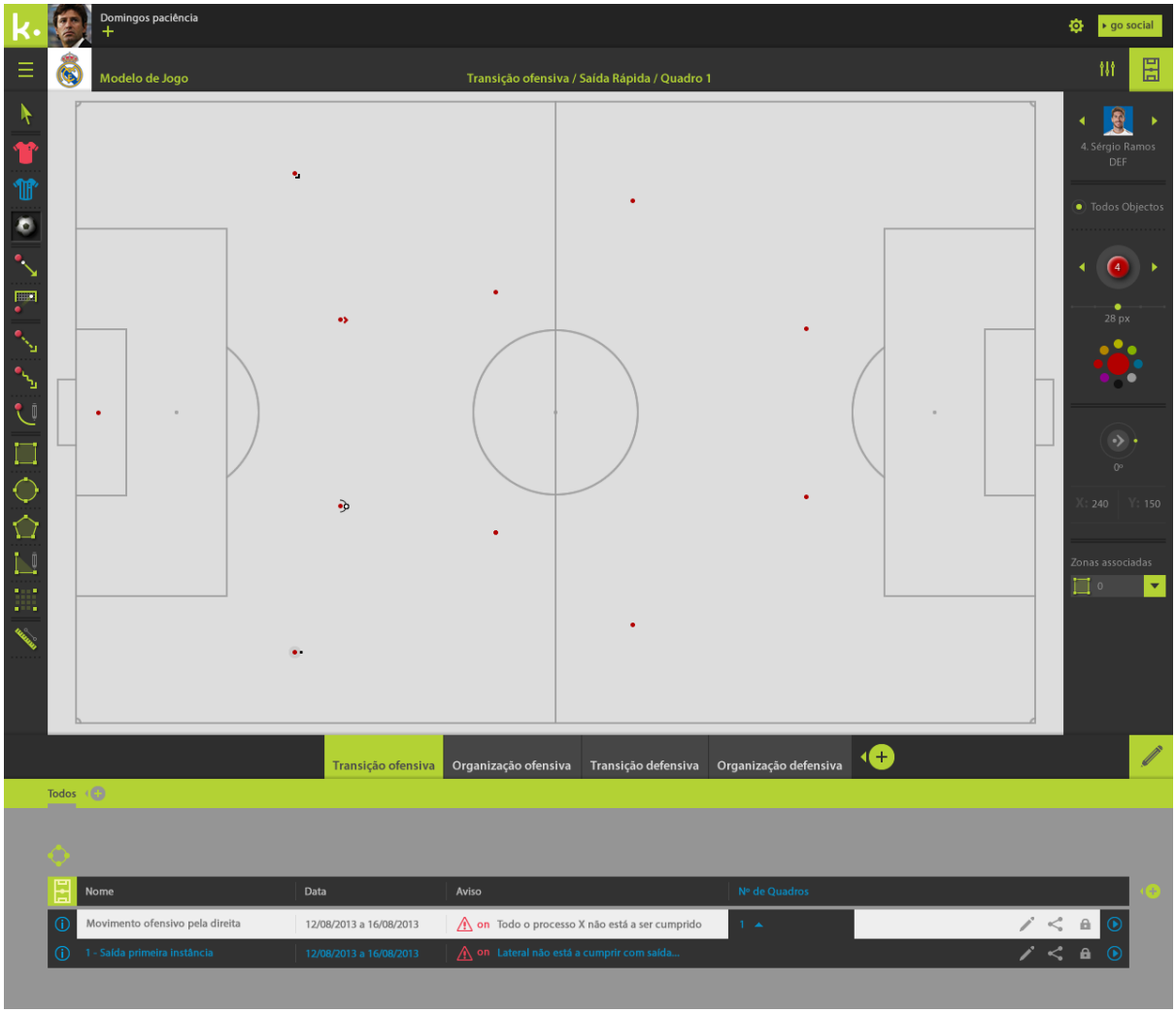


Figura 3.45: *Design* final do Editor de Modelos de Jogo (modo de precisão, estilo "*wireframe*").

cação da aplicação, é preciso testá-la, interligar os diversos módulos entre si e a sua manutenção tem de ser feita de forma fácil. Pelo que a forma como organizamos os ficheiros de código JavaScript, ficheiros de estilos, entre outros, tem de estar pensada para estes casos. O Yeoman permite-nos criar rapidamente um ambiente de desenvolvimento versátil para uma aplicação *web*, englobando ainda ferramentas importantes para teste da aplicação e compactação de ficheiros, por exemplo.

Se por um lado o código JavaScript precisa apenas de ser adaptado, a interface e a sua estilização tem de ser escrita de raíz de forma a ir ao encontro do *design* final. Nesse sentido, utilizou-se o Jade (Jade, 2014) que é uma *engine* para criação de *templates*

Figura 3.46: *Design* final do Editor de Vídeo.

cujo objetivo é a produção de ficheiros do estilo XML (HTML, RSS, etc) e que possui diversas vantagens, das quais se destacam o facto de ser fácil a sua aprendizagem e de

permitir herança de *templates*. Visto que algumas secções são comuns na maior parte das interfaces, a herança de *templates* é um aspeto muito positivo, tendo em conta o grande número de interfaces que toda a aplicação possuirá.

4

Conclusões e trabalho futuro

Esta dissertação apresenta o estudo e o desenvolvimento de quatro aplicações *web*, nomeadamente: Editor de Campo, Vídeo, *Tracking*, e Apresentações. Estas ferramentas foram desenvolvidas no âmbito de um Sistema Integrado de Gestão para o Futebol (SIGF), descrito no projeto QREN I&DT - FootData. Como principais contribuições, há a destacar: a) O estudo, projeto, desenvolvimento e implementação de quatro ferramentas *web* de acordo com os requisitos do projeto FootData; b) O estudo, análise e discussão das principais tecnologias atuais para o desenvolvimento das referidas ferramentas e c) O projeto e a implementação de uma ferramenta (Editor de Campo), onde as jogadas ou os modelos de jogo idealizados pelo treinador, podem ser descritos graficamente em ambiente *web*, sendo convertidos de forma automática para código,

que permitirá uma posterior análise em tempo real ao desempenho técnico-tático dos jogadores ou equipa em campo.

As novas tecnologias ajudam a encontrar soluções para os mais variados fins, e num desporto extremamente competitivo como o futebol, é legítimo ponderar o que poderia ser retificado ou melhorado, recorrendo a aplicações específicas para o futebol que assentem nas novas tecnologias.

Dois dos tipos de aplicação mais comuns são as aplicações nativas e as aplicações *web*. Por um lado, o conceito de aplicações nativas acarreta várias vantagens sobre as aplicações *web*, como por exemplo: experiência do utilizador, desempenho e segurança. Por outro lado, em aspetos como distribuição, manutenção e atualização, as aplicações *web* surgem como uma alternativa mais viável.

De forma a se conseguir melhorar esses pontos críticos da *web*, várias especificações têm surgido nos últimos anos relativamente às linguagens para criação de interfaces *web*, nomeadamente, HTML5, CSS3 e JavaScript. O HTML5 Canvas e o HTML5 Vídeo são exemplos dos novos elementos introduzidos pela mais recente especificação do HTML, elevando a fasquia na área da multimédia em ambiente *web*.

Todavia existem limites, pelo que pode ser necessário recorrer a *frameworks* quer seja para disponibilizar um certo elemento de forma uniforme (em termos visuais) em todos os *browsers*, ou para se tirar o maior proveito possível dos novos componentes introduzidos pelo HTML5.

Para se conseguir uma aplicação *web* robusta e funcional é crucial haver um planeamento criterioso tanto ao nível do *front-end* como no *back-end*. Há semelhança do que aconteceu para o *front-end*, também se constatou uma grande evolução das tecnologias utilizadas no *back-end* das aplicações, permitindo uma construção mais rápida de aplicações escaláveis. Uma dessas tecnologias é o NodeJS, que permite executar código JavaScript no lado do servidor. Tendo sido criado em 2009, tem vindo a revolucionar o ramo das aplicações baseadas em *real-time*.

O facto de uma aplicação *web* estar inserida num contexto de *cloud computing*,

remete-nos para a integração num só serviço de três componentes fundamentais, são elas: a computação, o armazenamento e o *software*.

É importante ter em conta como estruturamos a nossa aplicação não só fisicamente como também logicamente, pelo que se deve ter em conta os diversos *design patterns*, que embora não sejam soluções exatas para um determinado problema, fornecem um esquema para a sua solução, facilitando o trabalho dos programadores no que diz respeito ao *design* de *software*.

O recurso à prototipagem de *software* foi de extrema importância, na medida em que nos auxiliou bastante no entendimento dos requisitos, que na altura eram pouco claros pois envolviam conceitos de futebol algo complexos, e inclusive tem permitido um refinamento das funcionalidades até agora implementadas. Uma outra vantagem está associada ao facto de ter permitido um desenvolvimento em conjunto das diversas partes envolvidas no projeto, nomeadamente, programadores, *designers*, gestores e utilizadores (treinadores).

A plataforma FootData está estruturada numa *cloud*, onde é disponibilizado um serviço composto por vários módulos focados no mundo do futebol. De entre os módulos, há a salientar o *FootData Field*, *Coach* e *Presenter*, que encaixam em algumas das áreas mais críticas da gestão de uma equipa de futebol, das quais se destacam a preparação, planificação e análise de jogos. Com as ferramentas descritas nesta dissertação, procurou-se auxiliar a equipa técnica nessas mesmas áreas, providenciando meios eficazes para o desenho de jogadas e modelos de jogo (Editor de Campo), edição e análise de vídeos (Editor de Vídeo) e preparação de palestras e apresentações à equipa (Editor de Apresentações). Há ainda a salientar a possibilidade de correção dos dados provenientes de um sistema de *tracking* automático de jogadores e bola (Editor de *Tracking*).

Os principais objetivos a que nos propusémos foram alcançados. Há, contudo, um longo caminho a percorrer até à obtenção de um produto final para o consumidor.

Como trabalho futuro, pretendemos implementar algumas funcionalidades que

ainda não estão completamente finalizadas e outras que possam surgir com base no parecer dos consultores do projeto. O *design* de cada ferramenta foi e tem sido alvo de um estudo minucioso de forma a ser funcional e intuitivo sem comprometer a usabilidade, e brevemente todas as ferramentas estarão de acordo com os ditos *designs*.

4.1 Lista de publicações

No que diz respeito à disseminação do trabalho desenvolvido, encontra-se em preparação um artigo em revista, foram publicados dois artigos em conferências e encontra-se mais um aceite:

- Rodrigues, J.M.F., Cardoso, P., Vilas, T., Bruno, S., Rodrigues, P., Belguinha, A., Gomes, C. (2014) A computer vision based web application for tracking soccer players, *Accepted for 16th Int. Conf. on Human-Computer Interaction, Crete, Greece, 22-27 Jun.*

- Rodrigues, P., Cardoso, P. and Rodrigues, J.M.F. (2013) A Field, Tracking and Video Editor Tool for a Football Resource Planner, *In Proc. 8th Iberian Conf. on Information Systems and Technologies, Lisbon, Portugal, 19-22 June*, vol. 1, n. 2, pp. 734-739.

- 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 Proc World Conf. on Information Systems and Technologies (WorldCIST'13). Advances in Information Systems and Technologies, AISC 206, Á. Rocha et al. (Eds.), Vilamoura, Portugal, 27-30 March*, pp. 715–723.

Bibliografia

Al-Fedaghi, S. (2011). Developing web applications. *Int. J. of Software Engineering and Its Applications*, 5(2):57–68.

Anacleto, J. (2012). Desenvolvimento de uma aplicação web para dispositivos móveis - monitorização e controlo de uma rede de digital signage. Dissertação de Mestrado, Escola de Engenharia da Universidade do Minho.

Anthes, G. (2012). HTML5 leads a web revolution. *Communications of the ACM*, 55(7):16–17.

Bakken, D. (2003). *Middleware*. The Encyclopedia of Distributed Computing. Kluwer Academic Press.

Bell, D. (2005). *Software Engineering For Students: A Programming Approach*. Longman Group United Kingdom, 4th edition.

Bibeault, B. and Katz, Y. (2010). *jQuery in Action*. Manning Publications, 2nd edition.

Buyya, R., Broberg, J., and Goscinski, A. (2011). *Cloud Computing: Principles and Paradigms*. Wiley, 1st edition.

CanIUse (2013). Can I Use. www.caniuse.com. Acedido em 23 de Setembro de 2013.

Casario, M., Elst, P., Brown, C., Wormser, N., and Hanquez, C. (2011). *HTML5 Solutions - Essential Techniques for HTML5 Developers*. Apress.

Castelo, J. (2011). Observação de jogo e tecnologias digitais de análise. http://www.jorgecastelo.com/Academy/. Acedido em 23 de Setembro de 2013.

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

Chaffer, J. and Swedberg, K. (2011). *Learning jQuery*. Packt Publishing, 3rd edition.

Charland, A. and Leroux, B. (2011). Mobile application development: Web vs. native. *Communications of the ACM*, 54(5):49–53.

CoachFX (2013). CoachFX. www.coachfx.com. Acedido em 23 de Setembro de 2013.

Cross, M. (2007). *Developer's Guide to Web Application Security*. Syngress, 1st edition.

Duckett, J. (2009). *Beginning HTML, XHTML, CSS, and JavaScript*. Wrox, 1st edition.

EaselJS (2013). EaselJS. www.createjs.com/#!/EaselJS. Acedido em 24 de Setembro de 2013.

EasyAnimation (2013). Easy Animation. www.de.easy-animation.com. Acedido em 23 de Setembro de 2013.

eSpor (2013). eSpor Software Soccer eAssist. www.espor.com. Acedido em 23 de Setembro de 2013.

Fabric (2013). Fabric.js. www.fabricjs.com. Acedido em 24 de Setembro de 2013.

Flanagan, D. (2006). *JavaScript: The Definitive Guide*. O'Reilly Media, 5th edition.

Fowler, S. and Stanwick, V. (2004). *Web Application Design Handbook*. Morgan Keufmann.

Fulton, S. and Fulton, J. (2011). *HTML5 Canvas*. O'Reilly Media, 1st edition.

Google (2013). Google. www.google.com. Acedido em 23 de Setembro de 2013.

Green, B. and Seshadri, S. (2013). *AngularJS*. O'Reilly Media, 1st edition.

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

Herron, D. (2011). *Node Web Development*. Packt Publishing.

HTML5CanvasTutorials (2013). HTML5 Canvas Tutorials. www.html5canvas-tutorials.com/labs/html5-canvas-kineticjs-drag-and-drop-stress-test-with-1000-shapes/. Acedido em 24 de Setembro de 2013.

HTML5media (2013). HTML5 media. www.html5media.info. Acedido em 26 de Setembro de 2013.

HTML5rocks (2013). HTML5 Rocks. www.html5rocks.com. Acedido em 23 de Setembro de 2013.

HTML5test (2013). HTML5 Test. www.html5test.com. Acedido em 23 de Setembro de 2013.

Imagina (2013). Quadro interativo eBeam completo. http://www.imagina.pt/-produtos/hardware/quadro-interativo-ebeam-completo/. Acedido em 23 de Setembro de 2013.

Jade (2014). Jade. http://www.jade-lang.com/. Acedido em 6 de Janeiro de 2014.

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

jPlayer (2013). jPlayer. www.jplayer.org. Acedido em 26 de Setembro de 2013.

jQuery (2013). jQuery. www.jquery.com. Acedido em 23 de Setembro de 2013.

jQuery UI (2013). jQuery UI. www.jqueryui.com. Acedido em 12 de Dezembro de 2013.

JSON (2013). JSON. www.json.org. Acedido em 24 de Setembro de 2013.

K-PlayMaker (2013). K-PlayMaker. www.kplaymaker.com. Acedido em 23 de Setembro de 2013.

K-Studio (2013). K-Studio. www.kizanaro.com. Acedido em 23 de Setembro de 2013.

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

Keith, J. and Sambells, J. (2010). *DOM Scripting: Web Design with JavaScript and the Document Object Model*. friendsofED, 2nd edition.

Kessin, Z. (2011). *Programming HTML5 Applications*. O'Reilly Media.

Kiessling, M. (2011). *The Node Beginner Book*. Leanpub.

KineticJS (2013). KineticJS. www.kineticjs.com. Acedido em 24 de Setembro de 2013.

Lawson, B. and Sharp, R. (2011). *Introducing HTML5*. New Riders, 2nd edition.

Lennon, J. (2010). Compare JavaScript frameworks: An overview of the frameworks that greatly enhance JavaScript development. http://www.ibm.com/developerworks/library/wa-jsframeworks/.

Lindley, C. (2009). *jQuery Cookbook*. O'Reilly Media.

MacCaw, A. (2011). *JavaScript Web Applications*. O'Reilly Media.

Makzan (2011). *HTML5 Games Development by Example: Beginner's Guide*. Packt Publishing.

MatchInsight (2013). Prozone MatchInsight. www.prozonesports.com. Acedido em 23 de Setembro de 2013.

McFarland, D. S. (2009). *CSS: The Missing Manual*. O'Reilly Media, 2nd edition.

MediaElement (2013). MediaElement.js. www.mediaelementjs.com. Acedido em 26 de Setembro de 2013.

Meloni, J. (2011). *Sams Teach Yourself HTML, CSS, and JavaScript All in One*. Sams Publishing, 1st edition.

Meyer, J. (2011). *HTML5 and JavaScript Projects*. Apress.

Microsoft Developer Network (2014). Microsoft Developer Network. http://msdn.microsoft.com/. Acedido em 5 de Fevereiro de 2014.

Miller, M. (2008). *Cloud Computing: Web-Based Applications That Change the Way You Work and Collaborate Online*. Que Publishing.

Moore, D., Budd, R., and Benson, E. (2007). *Professional Rich Internet Applications: AJAX and Beyond*. Wrox, 1st edition.

Mozilla (2013). Mozilla Foundation. www.mozilla.org/foundation/. Acedido em 23 de Setembro de 2013.

NodeJS (2013). NodeJS. www.nodejs.org. Acedido em 23 de Setembro de 2013.

O'Dell, J. (2011). Why Everyone Is Talking About Node. http://mashable.com/-2011/03/10/node-js/.

Osmani, A. (2012). *Learning JavaScript Design Patterns*. O'Reilly Media.

Paperjs (2013). Paper.js. www.paperjs.org. Acedido em 24 de Setembro de 2013.

Pfeiffer, S. (2010). *The Definitive Guide to HTML5 Video*. Apress.

Pfleeger, S. L. and Atlee, J. M. (2009). *Software Engineering: Theory and Practice*. Prentice Hall, 4th edition.

Pilgrim, M. (2010). *HTML5: Up and Running*. O'Reilly Media, 1st edition.

Playback (2013). Prozone Playback. www.prozonesports.com. Acedido em 23 de Setembro de 2013.

Popcorn (2013). Popcorn.js. www.popcornjs.org. Acedido em 26 de Setembro de 2013.

PowerPoint (2013). Microsoft PowerPoint. http://office.microsoft.com/. Acedido em 23 de Setembro de 2013.

Powers, S. (2011). *HTML5 Media*. O'Reilly Media.

Rodrigues, P., Belguinha, A., Gomes, C., Cardoso, P., Vilas, T., Mestre, R., and Rodrigues, J. (2013). Open source technologies involved in constructing a web-based football information system. In *Advances in Information Systems and Technologies*, pages 715–723.

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

Sarrion, E. (2012). *jQuery UI*. O'Reilly Media.

Schmitt, C. and Simpson, K. (2011). *HTML5 Cookbook*. O'Reilly Media.

Silva, J. (2012). WebScout - Desenvolvimento de um interface gráfica para um software de Scouting. Dissertação de Mestrado, Faculdade de Engenharia da Universidade do Porto.

SocketIO (2013). Socket.IO. www.socket.io. Acedido em 25 de Setembro de 2013.

Sosinsky, B. (2011). *Cloud Computing Bible*. Wiley, 1st edition.

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.

SportsCode (2013). Sportstec SportsCode. www.sportstec.com. Acedido em 23 de Setembro de 2013.

Spurlock, J. (2013). *Bootstrap*. O'Reilly Media.

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

TacticalPad (2013). TacticalPad. www.tacticalpad.com. Acedido em 23 de Setembro de 2013.

Tactics Manager Software (2013). Tactics Manager Software. www.soccertutor.com/tacticsmanager/. Acedido em 23 de Setembro de 2013.

Taivalsaari, A. and Mikkonen, T. (2011). The web as an application platform: The saga continues. In *Proc. 37th EUROMICRO Conf. on Software Engineering and Advanced Applications, Oulu, Finland*, pages 170–174.

Thakur, V. (2008). *ASP.NET 3.5 Application Architecture and Design*. Packt Publishing.

UIkit (2013). UIkit. www.getuikit.com. Acedido em 12 de Dezembro de 2013.

Videojs (2013). Video.js. www.videojs.com. Acedido em 26 de Setembro de 2013.

Waterson, D. (2009). Back to the Server: Server-Side JavaScript On The Rise. https://developer.mozilla.org/en-US/docs/Web/JavaScript/Server-Side_JavaScript/Walkthrough.

WebSocket (2013). WebSocket++. http://www.zaphoyd.com/websocketpp. Acedido em 25 de Setembro de 2013.

Yeoman (2013). Yeoman. http://www.yeoman.io. Acedido em 20 de Dezembro de 2013.

Zakas, N. C. (2012). *Professional JavaScript for Web Developers*. Wrox, 3rd edition.