Abstract

In recent years, there have been an increasing number of studies in activity recognition, mainly due to the popularization of mobile devices which include a set of sensors that enable this research and bring the technology closer to people. Despite this, there is an area that can benefit from activity recognition and that has not yet been properly explored: mobile games. This study intends to analyze an activity recognition system for a mobile endless running game. In this work, it is shown that it is possible to obtain a system which classifies simple activities like standing, shift left or right, squat and jump with good accuracy. Beyond this, an activity recognition library is developed to facilitate the creation and use of this type of systems in mobile applications. As prove of concept, a game that integrates the previously studied system, using the developed library, is presented.

Keywords: Activity Recognition library, Mobile Exergames, Movement-based interaction, Immersion, Machine learning, Accelerometer, Gyroscope, Smartphone, Physical activity, short activities

Resumo

Nos últimos anos têm-se verificado um crescente número de estudos na área do reconhecimento de atividades. Isto deve-se, principalmente, à popularização dos dispositivos móveis, por incluírem um conjunto de sensores que possibilitam estes estudos e facilitam a aproximação da tecnologia às pessoas. Apesar disto, existe uma área que poderá beneficiar deste tipo de tecnologia e que não tem sido devidamente explorada: os jogos para dispositivos móveis. Com este estudo pretende-se analisar um possível sistema de reconhecimento de atividades a utilizar um jogo de *endless running* para dispositivos móveis. Com este trabalho é demonstrado que é possível obter um sistema para classificar atividades simples como estar parado, mover para o lado esquerdo e direito, agachar e saltar com boa exatidão. Para além disso, é desenvolvida e apresentada uma biblioteca para facilitar a criação e utilização deste tipo de sistemas em aplicações móveis. Como prova de conceito, é apresentado um jogo que implementa o sistema de reconhecimento de atividades previamente estudado através da biblioteca desenvolvida.

Palavras-chave: Biblioteca de reconhecimento de atividades, *Mobile Exergames*, interação baseada em movimentos, Imersão, *Machine learning*, Acelerómetro, Giroscópio, Smartphone, Atividade física, atividades curtas.

Agradecimentos

Gostaria de expressar o meu mais profundo agradecimento à minha orientadora, Ana Alves, pelo apoio prestado e disponibilidade durante todo o projeto.

Aos restantes professores do ISEC que partilharam o seu conhecimento e me inspiraram a aprofundar o meu.

Aos meus pais e irmã pelo apoio incondicional dado ao longo de todo o meu percurso académico.

A todos os participantes e voluntários que contribuíram para este estudo, pela disponibilidade, paciência e entusiasmo com que o fizeram.

Índice

Lista de Figuras

Lista de Tabelas

Definições e Acrónimos

API – *Application Programming Interface* – Conjunto de métodos, objetos e protocolos que os programadores podem utilizar para criar aplicações de software ou interagir com sistemas externos.

ARFF – *Attribute-Relation File Format* – Tipo de ficheiro de texto utilizado pela ferramenta Weka para guardar uma lista de instâncias que partilham o mesmo conjunto de características.

CSV – *Comma-Separated Values* – Tipo de ficheiro de texto que guarda valores numéricos e textuais separados por virgulas num formato tabular.

IDE – *Integrated Development Environment* – Plataforma que reúne um conjunto de ferramentas para agilizar o processo de desenvolvimento de software.

Característica – Trata-se de um valor que caracteriza uma amostra de dados recolhida. Pode ser um dos dados em bruto recolhidos ou uma transformação efetuada a partir destes. Num ficheiro CSV ou ARFF representa uma coluna de valores.

Instância de características – Trata-se de um conjunto de valores que representa as características de uma dada amostra ou janela de dados recolhida. Num ficheiro CSV ou ARFF representa uma linha de valores.

1 Introdução

Neste capítulo é apresentado o tema do trabalho elaborado, através de um enquadramento inicial, apresentação de objetivos e da estrutura do presente documento.

1.1 Enquadramento

À medida que a tecnologia evolui, os *smartphones* têm-se tornado cada vez mais comuns e poderosos. Os dispositivos mais recentes incorporam um leque de sensores que possibilita um conjunto vasto de oportunidades para aplicações de *data mining*. Consequentemente, a investigação relativa ao reconhecimento de atividades através da utilização deste tipo de dispositivos também tem aumentado.

Uma das potenciais, e menos exploradas, aplicações deste tipo de tecnologia são os jogos para dispositivos móveis. Existem hoje, no mercado, vários jogos com componente de atividade física, também conhecidos como *exergames* [1], tendo-se este género estabelecido como um tipo de jogo bastante popular e viável comercialmente. No entanto, estes jogos necessitam de equipamento específico (por vezes caro) e não são adequados para uso exterior (fora de casa). Os jogos para dispositivos móveis, por seu lado, têm uma grande potencialidade de levar a atividade física para qualquer lado mas, até à data, são poucos os que têm tentado explorar este segmento. Na maioria dos casos, estes jogos baseiam-se em geolocalização, fazendo os jogadores deslocarem-se até pontos específicos de um mapa, limitando-se à utilização de sensores como o GPS. Apesar de promoverem a atividade física, obrigando o jogador a mover-se, este tipo de jogos não implementa uma interação baseada em movimentos, muito menos um mapeamento realista entre os movimentos e as ações no jogo como alguns *exergames* disponíveis para as consolas.

O objetivo deste trabalho é, por isso, focar o estudo do reconhecimento de atividades nos jogos para dispositivos móveis. Pretende-se, através da utilização de sensores como o acelerómetro e giroscópio, possibilitar uma interação imersiva e realista, baseada em movimentos. De acordo com um estudo [2], elementos como o modo natural de controlo, mimetismo dos movimentos, postura e desafio físico, influenciam a imersão em interações baseadas em movimento. Assim, este tipo de interfaces proporciona uma forma de controlo mais natural do que interfaces tradicionais. A utilização de movimentos corporais é também descrita como intuitiva por parte dos jogadores e criticada quando os movimentos representados não se assemelham aos reais. Em jogos baseados em movimentos, a personagem virtual deve copiar os movimentos do jogador, levando a uma identificação mais forte com ela. Também é importante para um jogador ser capaz de entender facilmente os controlos de um jogo. Esta aprendizagem pode ser facilitada através da utilização de

um mapeamento realista entre as atividades do jogo e os movimentos naturais que o jogador já conhece.

Como prova de conceito, pretende-se criar um jogo do estilo *endless running* [3], para o sistema operativo *Android*, que integre um sistema de reconhecimento de atividades em vez do sistema de interação típico baseado em toque. O sistema a desenvolver terá de ser capaz de reconhecer em tempo real, através de apenas um *smartphone*, se o utilizador executou uma das atividades permitidas no jogo: estar parado, mover para o lado esquerdo, mover para o lado direito, agachar ou saltar.

Este género de jogos é um excelente candidato à utilização de reconhecimento de atividades como forma de interação para os seus movimentos devido ao conjunto simples de atividades que contempla, possibilitando um mapeamento realista entre a atividade da personagem no jogo e a atividade que o jogador terá que fazer na realidade para jogar. Desta forma, é possível criar um jogo que permita reconhecer a atividade do jogador e representar essa atividade no ecrã, criando uma experiência inovadora, mais realista e imersiva. Para além de uma nova forma de jogar e interagir com os dispositivos móveis, pretende-se também que este tipo de jogo constitua uma forma de motivar e promover a prática de atividade física na população mais jovem.

1.2 Objetivos

Os objetivos gerais deste trabalho são os seguintes:

- Apresentar um sistema de reconhecimento de atividades que possibilite reconhecer as atividades propostas em tempo real.
- Desenvolver uma biblioteca que permita facilitar a criação e utilização deste tipo de sistemas.
- Criar um jogo como prova de conceito que implemente o sistema de reconhecimento de atividades estudado através da biblioteca desenvolvida e verificar se este constitui uma forma inovadora e imersiva de interagir com o dispositivo móvel, promovendo a atividade física.

1.3 Estrutura

A restante estrutura deste documento organiza-se da seguinte forma:

No capítulo 2, é apresentado o estudo do estado da arte para este trabalho. Divide-se em três partes: uma dedicada aos *exergames*, outra aos estudos na área do reconhecimento de atividades e uma última às bibliotecas para o reconhecimento de atividades.

No Capítulo 3, é detalhado o estudo realizado para o sistema de reconhecimento de atividades a implementar. Ao abordar a criação do sistema de reconhecimento de atividades necessário neste trabalho, é descrito o processo de reconhecimento de atividades aplicado, definindo as atividades a classificar, como os dados são recolhidos, que características são extraídas e apresentados os algoritmos de seleção e classificação testados. São também apresentadas algumas decisões tomadas relativamente ao sistema. No final deste capítulo são apresentados os resultados do estudo.

No capítulo 4 é abordada a biblioteca criada com o intuito de facilitar o desenvolvimento de aplicações que implementem sistemas de reconhecimento de atividades como o estudado neste trabalho. São identificados os requisitos, as dependências e descrita a arquitetura da biblioteca.

O capítulo 5 apresenta uma prova de conceito. Neste capítulo, começa-se por apresentar o conceito inicial pretendido. De seguida, é abordado o processo de desenvolvimento do jogo, desde a criação dos objetos tridimensionais até à integração do sistema de reconhecimento de atividades previamente estudado. Para finalizar, são apresentados os resultados obtidos após a implementação.

Para finalizar, o capítulo 6 resume as conclusões deste trabalho. Inicialmente, é apresentada uma visão geral deste projeto. Em seguida, são discutidas algumas questões relacionadas com o estudo efetuado e os seus objetivos, analisando até que ponto foram alcançados. Posteriormente, são descritas as contribuições deste trabalho. Para terminar, são discutidas as suas principais limitações e dadas orientações para trabalho futuro.

2 Estado da arte

Neste capítulo são apresentados e analisados alguns conteúdos e trabalhos de investigação anteriores nas áreas de relevo para este trabalho. Assim, encontra-se subdividido em três subcapítulos: O primeiro é dedicado ao tema dos *exergames;* O segundo apresenta alguns dos trabalhos efetuados na área do reconhecimento de atividades através de sensores presentes em dispositivos móveis. No final do segundo subcapítulo, é ainda apresentada uma tabela comparativa (ver Tabela I) entre os estudos abordados e este trabalho; No terceiro e último subcapítulo, é revisto o atual conjunto de bibliotecas disponíveis para a implementação de uma aplicação que integre reconhecimento de atividades.

2.1 Exergames

Segundo Oh e Yang [1], a definição mais comum para *exergames* é "jogos de vídeo que requerem atividade física para que possam ser jogados". Este tipo de jogos tem como objetivo proporcionar uma forma de interagir que não constitua uma atividade sedentária, promovendo a atividade física. Para além disto podem constituir uma forma de criar uma experiencia de jogo mais envolvente e realista em que o jogador se sente parte do jogo. Os *exergames* têm vindo a ser, cada vez mais, alvo de maior estudo nos últimos anos devido a uma crescente preocupação com o grau de obesidade verificado nas populações, principalmente nos países desenvolvidos.

Em 2014, Dutz [4] analisou a interface de vários *exergames* para dispositivos móveis existentes. Segundo o seu artigo, este tipo de jogos pode ser agrupado de forma simplista em duas grandes categorias: jogos *indoor*, em que se joga normalmente em espaços interiores e os jogos móveis, jogados através de um dispositivo móvel e que por isso podem ser utilizados no exterior.

Relativamente aos jogos *indoor*, existem no mercado vários jogos deste tipo para as consolas Wii [5], Xbox [6] e Playstation [7]. Entre estes, é de destacar o *Wii Sports*, lançado em 2006 com a consola Wii e o comando *Wii Remote*. Este jogo requer que o utilizador faça certos movimentos com os braços e tronco através de um controlador específico (o *Wii Remote*) para que possa interagir com os minijogos de desporto incluídos, referentes a diferentes modalidades desportivas. Este jogo é, ainda hoje, o mais bem-sucedido comercialmente numa plataforma, com mais de 82 milhões de cópias vendidas [8], demonstrando bem a procura que existe por jogos que integram a atividade física na sua jogabilidade.

Uma característica interessante para este trabalho relativamente a estas plataformas está nos sensores utilizados pelos seus comandos de mão. Tanto a Wii como a PlayStation 3 incorporam

acelerómetros e giroscópios como sensores de movimento nos seus controladores de mão (*Wii Remote Plus* e *Playstation Move*) utilizados para interagir com os jogos, sendo que, inicialmente, o *Wii remote* apenas incorporava um acelerómetro. No entanto, mais tarde, foi criado o dispositivo *Wii motion plus* para expandir o *Wii remote*, tornando-o mais preciso, permitindo a captura de movimentos mais complexos. Uma das alterações foi a adição de giroscópios para determinar o movimento rotacional [9]. O seu sucessor *Wii Remote Plus* já incorpora estes sensores [10].

Relativamente aos jogos para dispositivos móveis, Dutz [4] identifica alguns jogos que requerem atividade física para serem jogados. Um dos primeiros e mais conhecidos é o *Geocaching* [11]. Este jogo consiste em encontrar caixas previamente colocadas em locais cujas coordenadas são divulgadas publicamente. A ideia é colocar os jogadores numa aventura de caça ao tesouro fazendo-os deslocarem-se fisicamente aos locais e procurar as caixas. Os primeiros jogadores empunhavam GPS de mão, no entanto, hoje em dia, com a popularização dos *smartphones* que integram dispositivos GPS e ligação à internet, o jogo tornou-se mais acessível e fácil de jogar, tendo a sua popularidade aumentado.

Jogos como o *Geocaching* são denominados de *location based* (baseados em localização), pois a sua jogabilidade requer que o jogador se desloque fisicamente para determinados locais de forma a progredir no jogo, sendo a sua posição determinada por um GPS. Atualmente existem vários jogos e aplicações baseados em localização. Um dos exemplos mais recentes e de maior notoriedade é o Pokemon Go [12], desenvolvido pela Niantic em parceria com a The Pokemon Company. Neste jogo é explorada a franquia *Pokémon*, sobejamente conhecida, permitindo localizar, capturar, lutar e treinar criaturas virtuais denominadas de *pokémons* que aparecem no ecrã com base na localização do dispositivo e se integram com o ambiente. Lançado em Julho de 2016 para dispositivos móveis *Android* e *IOS*, rapidamente se tornou um sucesso, popularizando as tecnologias baseadas em localização e de realidade aumentada. A Niantic já tinha anteriormente desenvolvido o jogo Ingress [13] que utilizava o mesmo tipo de tecnologias. De forma simplista, o objetivo era ativar ou desativar portais virtuais posicionados em locais físicos do mundo real.

Apesar de nenhum destes exemplos integrar o reconhecimento de atividades, têm sido conduzidas algumas experiências nesse sentido. Em 2013, Buddharaju e Pamidi [14] apresentaram uma plataforma para *exergaming* que requer que o utilizador se mova e salte fisicamente de forma a pontuar num jogo para dispositivos móveis. O sistema desenvolvido utiliza uma tábua/tapete denominada de *ExerPad*, colocada no chão, para seguir os movimentos horizontais do utilizador e os sensores do smartphone. O *ExerPad* contém quatro formas desenhadas, quando um utilizador se move horizontalmente para a posição de uma dessas formas, o smartphone deteta a sua posição através da utilização da câmara e de um algoritmo de deteção de formas. Para detetar o salto são utilizados os sensores de aceleração e rotação do smartphone. Como prova de conceito, é adaptado à plataforma proposta o famoso jogo *space invaders*. No referido jogo, o utilizador tem de se mover horizontalmente de forma a mover o canhão laser e tem de saltar para disparar contra os

aliens que vão caindo a partir do topo do ecrã. A plataforma proposta foi testada por 20 participantes, tendo o estudo concluído após um inquérito, que os *exergames* propostos ajudam os seus utilizadores a terem um bom treino físico enquanto se divertem.

Esta plataforma pretende ser uma forma nova de interagir com os jogos através de atividade física. É também genérica, querendo isto dizer que pode ser utilizada para vários tipos de jogos diferentes, não existindo, por isso, um mapeamento direto ou realista entre os movimentos do jogador e os movimentos de uma personagem do jogo. Os movimentos constituem assim meramente uma forma de *input* que substitui a interação tradicional efetuada através de botões ou toque. Apesar de esta abordagem ser interessante para o desenvolvimento de *exergames*, difere do objetivo deste estudo, que é o de criar um sistema que permita reproduzir os movimentos do utilizador aplicando-os a uma personagem da forma o mais realista e direta possível.

Noutro artigo, Karime [15] propõe um sistema que consiste em duas *SmartinSoles* [16], que podem ser colocadas dentro das sapatilhas do utilizador, um monitor de frequência cardíaca e um jogo desenvolvido para promover atividades de corrida e salto. O objetivo é substituir a interação tradicional nos dispositivos móveis por formas de promover atividade física. Convém salientar que as atividades a executar através deste sistema são unidimensionais, querendo isto dizer que o jogador exerce a atividade sem sair do mesmo sítio. O objetivo é mover os pés, levantando-os do chão ao mesmo tempo, no caso do salto ou alternadamente, no caso da corrida.

Esta proposta é bastante interessante relativamente ao que se propõe a fazer (deteção de salto e corrida), no entanto, parece limitada quanto ao tipo de movimentos. Outro aspeto é o facto de requerer equipamento extra, para além de um dispositivo móvel.

2.2 Reconhecimento de atividades através de sensores de dispositivos móveis

O reconhecimento de atividades humanas é uma área de investigação e desenvolvimento em Computação Ubíqua que permite inferir que atividades estão a ser realizadas por uma pessoa, através de dados obtidos a partir de um ou vários sensores. Nos últimos anos têm-se verificado um crescente número de estudos nesta área, principalmente devido à popularização de dispositivos móveis que incluem um conjunto de sensores que possibilitam estes estudos e facilitam a aproximação da tecnologia às pessoas, abrindo portas a um conjunto vasto de possibilidades relativamente a aplicações pervasivas.

Em 2014, Su, Tong e Ji [17] apresentaram um *survey* extensivo que sumarizava os recentes avanços té à data no estudo do reconhecimento de atividades através da utilização dos sensores de *smartphones*. No artigo são revistos pontos essenciais na diversa literatura existente, como os

sensores utilizados, atividades classificadas, técnicas de extração de características dos dados recolhidos, classificadores, desafios e aplicações da tecnologia.

Diferentes estudos focam-se em diferentes etapas e técnicas do processo de reconhecimento de atividades. Em 2012, Ayu [18] avaliou através de um estudo comparativo sete categorias diferentes de algoritmos de classificação de atividades. Este estudo deu continuidade a uma investigação anterior [19] efetuada pelos mesmos autores, em busca de um algoritmo adequado e fiável para o reconhecimento de atividades em tempo real através da utilização de um dispositivo móvel. As atividades testadas foram as mais comuns para este tipo de estudo: correr, saltar, sentar, andar e ficar parado de pé. A avaliação foi efetuada através da ferramenta Weka. O estudo concluiu que todos os algoritmos tiveram resultados satisfatórios, tendo, no geral, a taxa de exatidão (*accuracy*) ultrapassado os 90% nos testes de classificação, o que representa resultados muito animadores.

Preece e Goulermas [20] apresentaram em 2008 uma comparação entre catorze métodos de extração de características através de sinais vindos do acelerómetro. Os resultados do estudo mostram que as características baseadas em frequência obtiveram maior exatidão relativamente à transformada *wavelet* quando se classifica atividades dinâmicas.

Outros estudos têm uma abordagem mais geral, abrangendo todo o processo de reconhecimento de atividades, desde a recolha de dados até à sua classificação. Em 2010, Kwapisz [21] descreve e avalia um sistema para reconhecimento de atividades baseado em dispositivos móveis. O objetivo é permitir adquirir conhecimento útil sobre os hábitos de milhões de utilizadores de forma passiva, bastando apenas que estes levem um smartphone no bolso, durante as suas atividades diárias. Para implementar este sistema foram recolhidos dados de 29 utilizadores enquanto realizavam atividades diárias como andar, correr, subir escadas, sentado, ficar parado de pé e depois agregadas estas séries temporais de dados em exemplos que sumarizavam a atividade realizada durante intervalos de 10 segundos. Através deste estudo foi demonstrado que é possível reconhecer atividades de forma precisa tendo a grande maioria das atividades sido reconhecidas corretamente 90% das vezes.

Em 2014 Arif [22] apresenta um sistema para classificação de atividades (andar, correr, sentado, de pé, subir escadas e descer escadas) através do processamento de dados vindos do acelerómetro de um dispositivo móvel, onde afirma que, com a sua configuração, obteve uma exatidão (*accuracy*) superior a 98% na classificação das seis atividades. Neste estudo é utilizado o mesmo *dataset* que Kwapisz [21] estando, por isso, o smartphone posicionado no bolso da frente das calças do seu utilizador. Foram extraídas características temporais a partir dos dados do acelerómetro e utilizadas técnicas de *feature subset selection* para diminuir esse conjunto de características, selecionando as mais relevantes, e diminuir a complexidade do algoritmo. O algoritmo de classificação utilizado foi o KNN (*K-nearest neighbor*) [23].

Tanto Kwapisz [21] como Arif [22] utilizam como posição para o *smartphone* o bolso da frente do utilizador, no entanto existem muitos outros locais explorados em diversos estudos. Para este artigo, a posição mais relevante é a palma da mão, com o dispositivo em frente ao utilizador, de modo a que este possa observar o jogo enquanto se move. Ayu [18] no seu estudo comparativo de classificadores já tinha abordado duas posições, o bolso da camisola e a palma da mão. Em 2014 Bayat [24] apresenta o seu sistema de reconhecimento de atividades através dos dados do acelerómetro de um smartphone, onde também utiliza para além do bolso das calças, a palma da mão como posição para o *smartphone*.

Grande parte dos estudos efetuados na área do reconhecimento de atividades utiliza o acelerómetro como sensor principal e muitas vezes único para inferir as atividades que estão a ser efetuadas. No entanto, pouco foi explorado quanto às potencialidades do giroscópio. Em 2014, Shoaib [25] explora a forma como os diferentes sensores (acelerómetro e giroscópio) se comportam em diferentes situações no processo de reconhecimento de atividades. Foi levada a cabo uma experiência onde dez participantes realizaram sete atividades diferentes com cinco *smartphones* em diferentes posições à volta do corpo. Esta experiência demonstrou que cada um destes sensores pode ter um papel importante no reconhecimento de atividades e que quando utilizados em conjunto melhoram a exatidão geral do reconhecimento ou pelo menos mantêm-no igual à performance máxima individual dos sensores em quase todas as situações com muito poucas exceções.

Relativamente ao presente trabalho, existem algumas diferenças substanciais que é necessário realçar em comparação com outros trabalhos efetuados até agora:

- É necessário que as atividades sejam reconhecidas através de apenas um smartphone, tendo a sua posição de ser obrigatoriamente na mão do utilizador, uma vez que é necessário que este esteja a ver o ecrã do dispositivo para jogar.
- Este estudo inclui o reconhecimento de atividades curtas e não continuas (ex. desviar para o lado), contrariamente às atividades comummente abordadas em outros estudos como andar ou correr.
- Existem atividades, como é o caso do movimento lateral esquerdo e direito, que constituem um movimento idêntico cuja diferença está na sua direção. Apesar disto, devem ser detetados como atividades diferentes.
- São extraídas características a partir dos dados do acelerómetro e giroscópio.
- As atividades devem ser reconhecidas em tempo real.

2.2.1 Tabela comparativa dos estudos de reconhecimento de atividades analisados

Na Tabela I são apresentados os trabalhos previamente descritos comparando-os com o estudo que é efetuado neste trabalho.

Tabela I - Comparação de alguns estudos de reconhecimento de atividades anteriores.

Artigo	***Atividades***	***Sensores***	***Nº de smartphones***	***Posição dos smartphones***	***Tempo real***	***Exatidão (accuracy)***
Ayu *[18]*	Parado em pé, sentado, andar, correr, saltar	Acelerómetro	1	Bolso da camisola, Mão	Sim	95%, 100% (*10-fold cross-validation*)
Bayat *[24]*	Parado em pé, andar, correr, abdominais, aspirar, lavar os dentes, subir e descer escadas	Acelerómetro	1	Bolso, Mão	?	90.34%, 91.15%
Kwapisz *[21]*	Parado em pé, sentado, andar, correr, subir e descer escadas	Acelerómetro	1	Bolso da frente das calças	Não	91.7%
Arif *[22]*	Parado em pé, sentado, andar, correr, subir e descer escadas	Acelerómetro	1	Bolso da frente das calças	?	99.3% (*10-fold cross-validation*)
Shoaib *[25]*	Parado em pé, sentado, andar, correr, bicicleta, subir e descer escadas	Acelerómetro e giroscópio	5	Bolsos da frente das calças, cintura, parte superior do braço direito e pulso direito	Sim	N/A
Este	Parado em pé, movimento lateral, saltar, agachar	Acelerómetro e giroscópio	1	Mão	Sim	> 99% (*10-fold cross-validation*)

2.3 Bibliotecas para reconhecimento de atividades

O reconhecimento de atividades é um tipo específico de problema de *machine learning*, uma área vasta e em rápida expansão, especialmente devido ao fácil acesso e abundancia de dados, bem como à crescente necessidade de analisar e extrair valor destes.

Hoje em dia existem várias *frameworks* e bibliotecas que integram algoritmos de *machine learning* com diferentes propósitos. Um conjunto vasto está relacionado com a área de *Big Data* e *clustering*. No entanto, existe também um grande número de bibliotecas de propósito genérico. Dois dos exemplos mais populares são o scikit-learn [26], desenvolvido em Python e o Weka [27] em Java. Estas bibliotecas focam-se na disponibilização de um conjunto diverso de algoritmos comumente utilizados nos problemas de *machine learning*, requerendo algum conhecimento dessas técnicas para criar o sistema específico pretendido. São também difíceis de integrar em aplicações móveis, pois não foram pensadas para esse fim. Desta forma, não são o caminho mais simples para quem pretende integrar um sistema de reconhecimento de atividades numa aplicação móvel.

Para possibilitar uma fácil utilização do reconhecimento de atividades em dispositivos móveis *Android*, a Google introduziu em 2013 a *ActivityRecognitionAPI* [28] [29]. Esta API permite aos programadores integrar nas suas aplicações o reconhecimento de certas atividades sem necessitar de conhecimentos de *machine learning*. No mesmo ano, a Apple introduziu uma API similar denominada de *CMMotionActivity* [30] para *IOS*. Analogamente, a Microsoft lançou uma interface para o *Windows Phone* denominada de *Activity Monitor API* [31]. Estas interfaces permitem que uma aplicação possa detetar de forma muito simples se o utilizador do dispositivo está a andar, correr, de carro, bicicleta ou parado sem a necessidade de criar modelos complexos de *machine learning*. O modelo já está criado e treinado com um conjunto de atividades predeterminadas. O programador apenas regista o seu interesse em obter informação de qual atividade está a ser executada e é informado a cada nova deteção.

No entanto, apesar de estas interfaces permitirem detetar, de uma forma extraordinariamente simples, que atividade está a ser executada pelo utilizador do dispositivo móvel, são pouco flexíveis. Isto porque o sistema não permite ao programador fazer um sistema personalizado para detetar atividades específicas. Trata-se de um sistema já treinado com atividades predefinidas, o que limita bastante a utilização deste tipo de serviço. Desta forma, pretende-se que a biblioteca a desenvolver neste trabalho represente um ponto intermédio entre as bibliotecas genéricas de *machine learning* e estas interfaces de reconhecimento de atividades. Por um lado, pretende-se que se foque no processo de reconhecimento de atividades através de uma interface simples de utilizar. Por outro lado, que permita criar e treinar um modelo personalizado que vá de encontro às necessidades específicas do programador.

3 Estudo do sistema de reconhecimento de atividades a implementar

Neste capítulo é apresentado o estudo preliminar efetuado para determinar o sistema de reconhecimento de atividades que mais tarde se pretende implementar. Em primeiro lugar, é apresentada uma visão geral do processo típico de reconhecimento de atividades. Depois, são descritas as atividades que se pretende detetar, seguindo-se as descrições das várias etapas do processo de reconhecimento de atividades envolvido neste estudo, desde a recolha dos dados até à classificação de atividades. No final do capítulo são ainda apresentados e discutidos os resultados do sistema sugerido. O objetivo é que este sistema possa reconhecer com exatidão e em tempo real os cinco movimentos necessários para o jogo a implementar. É ainda de referir que a recolha de dados é efetuada a partir de dois sensores de um smartphone *Android*: o acelerómetro e o giroscópio. Para o estudo dos algoritmos de seleção e classificação a aplicar será utilizada a ferramenta de *machine learning* Weka.

3.1 Visão geral do processo de reconhecimento de atividades

O processo de reconhecimento de atividades através de dispositivos móveis engloba diferentes fases, como a Figura 1 ilustra. Pretende-se agora analisar e sumarizar cada uma dessas fases, posteriormente detalhadas ao longo do restante capítulo:

- **Recolha de dados**
  Nesta fase é efetuada a recolha de uma amostra ou janela de dados através dos sensores do dispositivo móvel. No caso especifico do reconhecimento de atividades, a forma como a recolha de dados é efetuada torna-se bastante importante. Isto porque, a escolha da janela de dados a recolher difere com o tipo de atividades que se pretende detetar, bem como com a frequência de classificação que é necessária. É também importante que durante o treino se obtenha um conjunto de dados com menor ruído possível a partir de um espetro variado de participantes para que se possa criar um modelo robusto e eficiente.
- **Extração de características**
  Nesta fase, os dados em bruto recolhidos anteriormente são processados com o propósito de extrair características relevantes que facilitem a posterior tarefa de classificação. Esta extração resulta numa instância de características, utilizada posteriormente pelo sistema.
  A fase de extração de características é uma das mais importantes na criação de modelos de *machine learning*, sendo normalmente referida como um dos aspetos mais importantes nas competições da plataforma Kaggle [32] [33]. A extração de características tem por base

conhecimento específico do domínio do problema, podendo, por isso, ter maior influência no resultado do que a escolha dos algoritmos de classificação a utilizar.

Segundo Tim Dettmers, num artigo publicado no *blog* de desenvolvimento da Nvidia [34], "A engenharia de características é a habilidade mais importante quando se pretende obter bons resultados para a maioria das tarefas de predição. No entanto, é difícil de aprender e dominar uma vez que diferentes conjuntos de dados e diferentes tipos de dados requerem abordagens diferentes. Existem apenas diretrizes rudimentares, o que torna a engenharia de características mais uma arte do que uma ciência".

- Seleção

  Nesta fase, as características previamente extraídas são reduzidas através da utilização de algoritmos de seleção de características. O objetivo é obter um subconjunto de características, selecionando as mais relevantes para tornar o processo de classificação mais simples. No caso de existir um conjunto alargado de características redundantes ou irrelevantes, os resultados da classificação podem até melhorar.

- Classificação

  Nesta fase, uma instância de características final é classificada. No caso de se estar a treinar o sistema, é necessário dar manualmente um rótulo identificador à instância. No caso de se estar a testar, é utilizado um algoritmo de classificação que tentará prever o rótulo da atividade com base no treino efetuado.

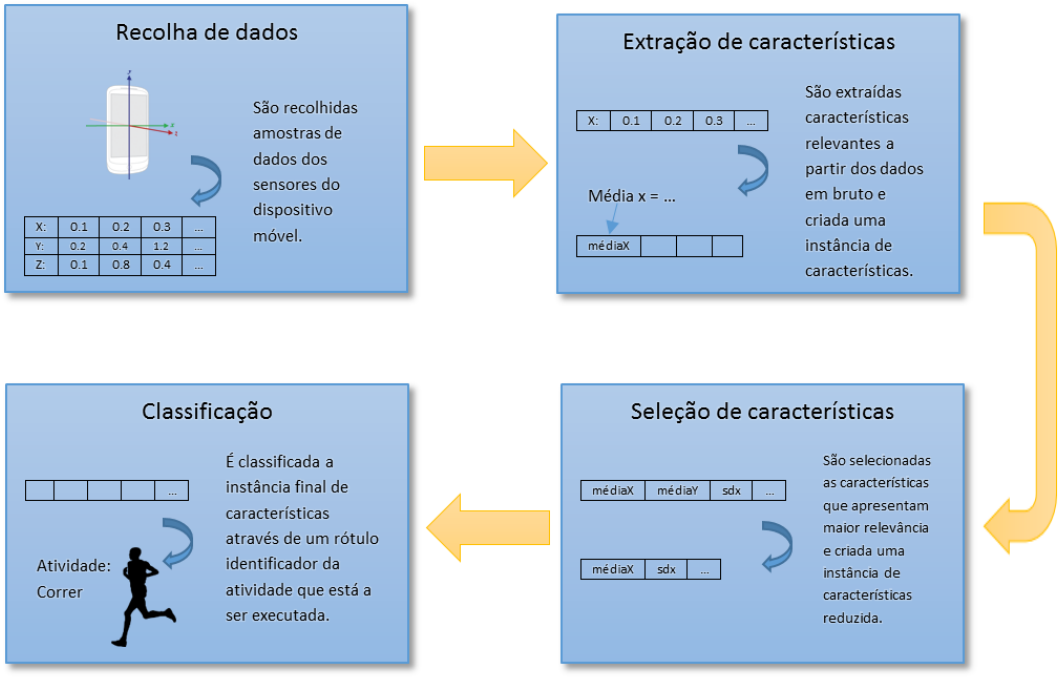


Figura 1 - Processo típico de reconhecimento de atividades através de dispositivos móveis.

3.2 Atividades

No jogo que se pretende implementar, a personagem controlada pelo jogador tem um percurso à sua frente que deverá percorrer, sendo que o seu objetivo é chegar o mais longe possível evitando obstáculos frontais enquanto corre. Para este tipo de jogo, existem quatro atividades principais que deverão ser reconhecidas:

- **Parado de pé**. Movimento por defeito no jogo e para o qual a escolha deve recair em caso de dúvida. O jogador encontra-se parado e a personagem do jogo continua a correr sem sofrer alterações ao seu movimento. É um movimento contínuo, que se prolonga de forma cíclica. Este tipo de atividade envolve pequenos movimentos que ocorrem por períodos alargados de tempo tornando a sua deteção mais fácil que movimentos curtos e não contínuos, como os descritos a seguir.
- **Movimento lateral**. Movimento efetuado quando o jogador se desvia lateralmente de um obstáculo. Trata-se de um movimento rápido e não contínuo. É constituído por duas variantes, cuja diferença se encontra na direção: movimento lateral para a esquerda e movimento lateral para a direita.
- **Agachar**. O jogador tem de se agachar para ultrapassar um obstáculo. Trata-se de um movimento rápido e não contínuo. Para além disto, pode ter alguma variabilidade na forma como diferentes utilizadores o executam.
- **Saltar**. O jogador tem de saltar na vertical para ultrapassar um obstáculo que esteja ao nível do chão. É o movimento de maior intensidade caracterizado pela impulsão de todo o corpo para cima. É também um movimento curto e não contínuo.

Para não introduzir incerteza e ambiguidade na classificação das atividades, optou-se por considerar o movimento lateral como sendo apenas uma atividade, independentemente de este movimento ser para a esquerda ou para a direita, devido à similaridade que existiria entre as duas atividades. Estando a diferença essencialmente nos valores do eixo do x do acelerómetro, optou-se por criar um classificador dedicado à análise dessa informação para determinar apenas se o movimento é para a esquerda ou para a direita. Assim, existirá um sistema para classificar 4 atividades (estar parado, movimento lateral, agachar e saltar) e outro para, quando o movimento lateral for detetado, determinar se este foi para a esquerda ou para a direita. Separar estas duas atividades, colocando-as num sistema à parte, pode reduzir a incerteza que poderia de outra forma ser introduzida no classificador do sistema principal.

3.3 Recolha de dados

Para permitir a recolha de dados foi criada uma aplicação, posteriormente instalada no telemóvel utilizado pelos participantes. A aplicação consiste em dois modos: treino e teste. No modo de treino, ilustrado na Figura 2 (a), é possível, através de uma interface simples, escolher a atividade que será executada, começar e parar a recolha de dados. No modo de teste, ilustrado na Figura 2 (b), foi implementada uma versão preliminar do sistema apresentado neste capítulo para obter *feedback* após o treino das atividades. Este modo revelou-se crucial para perceber se o treino estava a correr bem e quais as atividades que necessitavam de mais exemplos de treino. O funcionamento interno da aplicação pode ser observado através do diagrama da Figura 3.

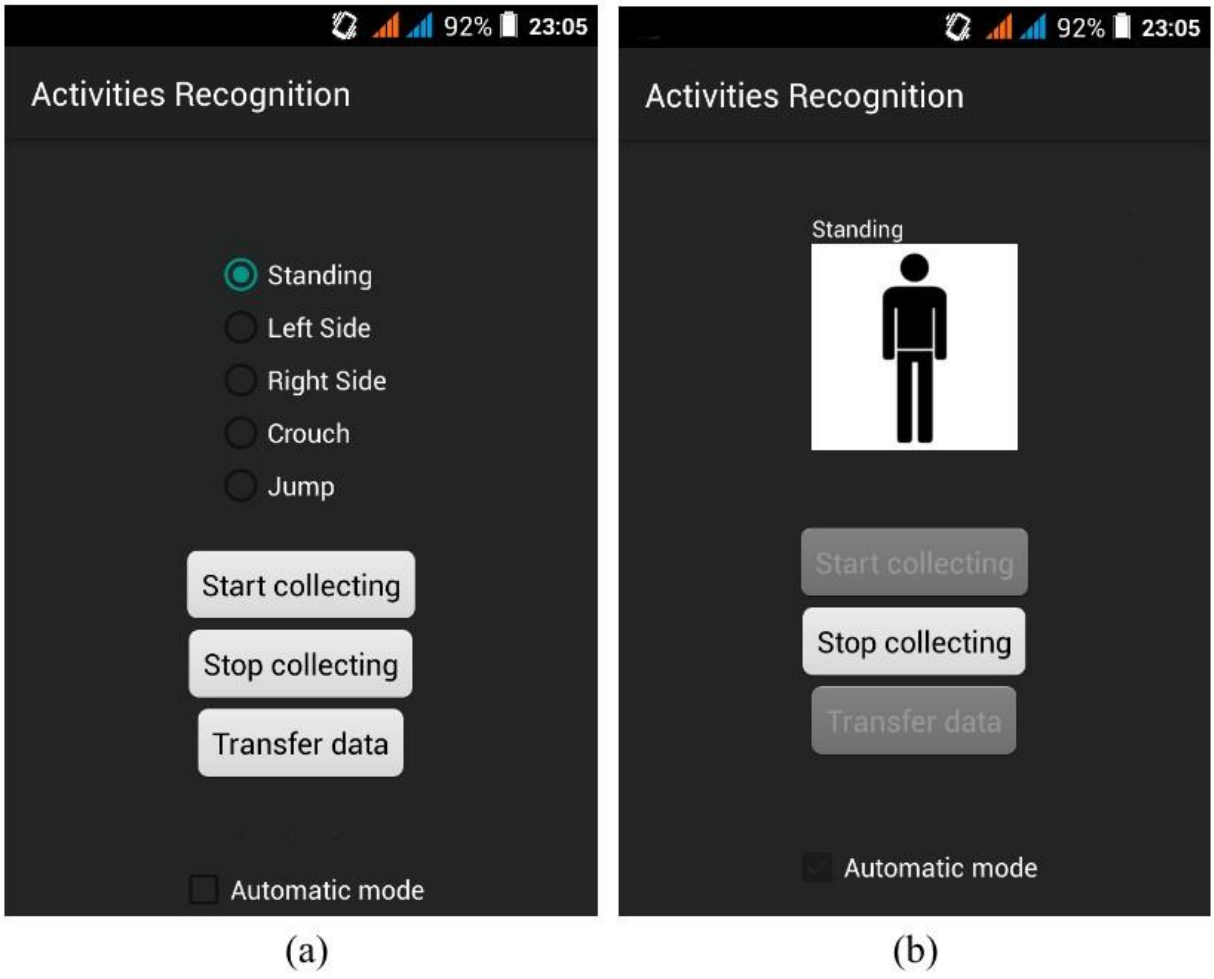


Figura 2 - *Screenshots* da interface de treino (a) e de teste (b) da aplicação de recolha de dados.

A aplicação recolhe conjuntos de 64 valores (janela de amostragem ou instância) a partir dos três eixos do acelerómetro e do giroscópio, estando ajustada para obter dados com uma taxa de amostragem de 0,02 segundos. Esta taxa deve-se à utilização da *flag* SENSOR_DELAY_GAME [35] do sistema operativo *Android*. Desta forma, cada instância, constituída pelos 64 valores amostrais, representa cerca de 1,28 segundos. Existe também uma restrição quanto ao número de

dados a recolher por instância, tendo este número de ser uma potência de dois, devido à utilização da transformada rápida de Fourier (FFT – *Fast Fourier Transform*) [36], na fase de extração de características, permitindo um processamento mais rápido deste algoritmo. Dentro dos valores possíveis, este torna-se assim o mais adequado, dadas as durações médias das atividades requeridas neste estudo. Para o treino de atividades contínuas e no modo de teste, a aplicação deverá também aplicar uma sobreposição de 50% nas janelas de valores recolhidas. Isto quer dizer que serão recolhidos dados novos a cada 32 conjuntos de valores e adicionados aos últimos 32 valores já recolhidos da janela anterior, formando assim um conjunto de 64 valores em que os primeiros 32 são referentes à amostra anterior e os últimos 32 à amostra atual. Na aplicação, existe ainda a opção de parar a recolha automaticamente. Através desta opção, a recolha é automaticamente interrompida após terem sido recolhidos 64 valores amostrais para uma atividade, sendo necessário voltar a carregar manualmente na opção de recolher (*Start Collecting*) para obter mais 64 valores. Assim, quando esta funcionalidade está ativa, não é aplicada sobreposição aos dados recolhidos, uma vez que se trata de uma recolha única. O objetivo desta funcionalidade é obter dados da forma mais "limpa" possível para as atividades curtas e não contínuas, como é o caso do movimento lateral, agachamento e salto. Assim, apenas são recolhidas instâncias de dados relevantes e representativas do movimento pretendido. Após a recolha de algumas instancias de dados, o utilizador poderá observar o quão bem a aplicação está a prever os seus movimentos através do modo automático (modo de teste). Ao mudar para este modo, o classificador do sistema é imediatamente treinado com as instâncias de treino recolhidas até ao momento pelo utilizador.

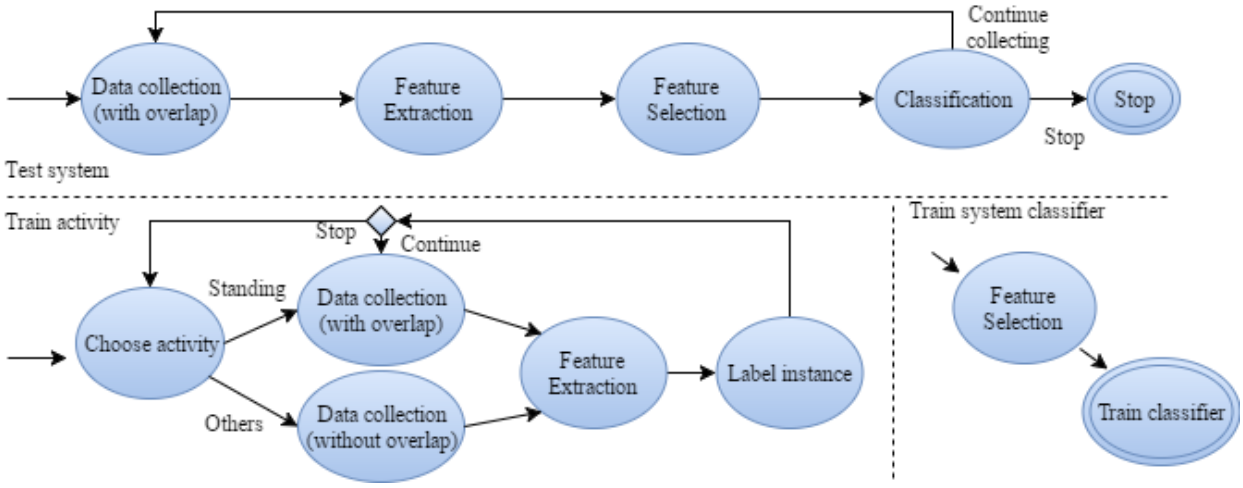


Figura 3 - Diagrama representativo do funcionamento interno da aplicação de recolha de dados.

A fase de recolha de dados teve a participação de 6 voluntários que executaram as diferentes atividades descritas neste estudo. No total, 4176 instâncias de dados foram recolhidas, como é possível observar na Tabela II. A recolha foi supervisionada para garantir que as atividades eram executadas corretamente. A fase de recolha de dados é de extrema importância para garantir bons resultados de classificação por parte do sistema. Assim, foi-lhe dada bastante atenção, requerendo

algum esforço por parte quer do monitor quer por quem estava a executar a atividade. Enquanto que, para o movimento lateral, agachamento e salto as atividades têm de ser executadas de forma o mais correta possível, para a atividade de ficar parado em pé, os participantes foram encorajados a executar alguns pequenos movimentos espontâneos como tirar a mão do bolso, ajustar a posição do dispositivo na mão, rodar o torso, fletir ligeiramente os joelhos ou mover ligeiramente os pés. O objetivo é tornar esta atividade mais abrangente, incluindo vários pequenos movimentos não classificados, para que em caso de dúvida, esta seja a atividade escolhida. Para além disto, foi integrado um treino antifraude com o intuito de tornar o sistema mais robusto contra tentativas de o enganar e encorajar o jogador a fazer as atividades corretamente. Para isso, enquanto treinavam a atividade de ficar parado de pé, foi pedido aos participantes que tentassem enganar o jogo, movendo o braço para tentar simular as outras atividades sem mover o seu corpo na realidade. Com isto, pretende-se que exista uma maior probabilidade de os movimentos serem interpretados corretamente pelo sistema se forem feitos corretamente pelo utilizador do que tentando enganá-lo evitando a execução da atividade.

Tabela II - Número de instâncias recolhidas por participante e atividade.

ID do participante	***Parado de pé***	***Movimento lateral esquerdo***	***Movimento lateral direito***	***Agachar***	***Saltar***	***Total***
1	553	233	233	108	95	1212
2	160	91	92	154	37	534
3	159	49	49	40	46	343
4	434	100	101	89	74	798
5	171	125	131	223	169	819
6	160	119	109	43	39	470
Total	1637	717	705	657	460	4176

A execução das atividades durante o treino contemplou ainda 3 tipos de posturas para cada atividade, como ilustra a Figura 4. Na primeira posição o utilizador está hirto com o dispositivo para baixo (a). Na segunda, encontra-se com a mesma postura corporal, mas agora com a posição do dispositivo mais subida, em frente à face (b). Na terceira e última posição, os movimentos são executados a partir de uma base mais sólida em que as pernas estão ligeiramente afastadas, à largura dos ombros, as pernas ligeiramente fletidas e a posição do dispositivo subida, em frente à face da pessoa (c). Esta terceira posição representa uma postura mais ágil e desportiva, promovendo a execução correta dos movimentos. Desta forma, deve ser a postura a incentivar aos jogadores. No entanto, apesar de ser considerada a mais indicada, é essencial ter em conta outras

posturas comuns por forma a flexibilizar a execução das atividades e melhorar a capacidade de classificação do sistema.

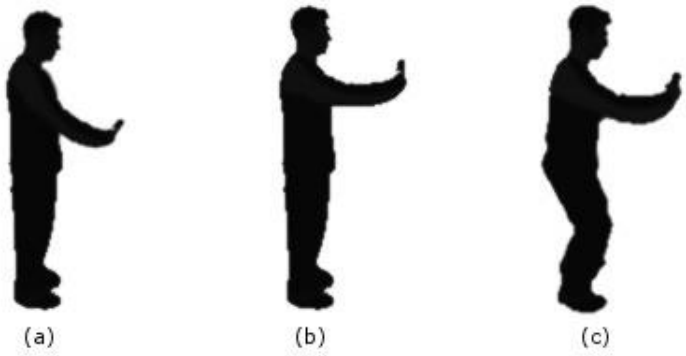


Figura 4 - As três posturas utilizadas no treino das atividades.

3.4 Extração de características

Nesta fase são tratados os dados recolhidos a partir dos três eixos do acelerómetro e giroscópio do dispositivo móvel com vista a retirar informação útil que simplifique e melhore a posterior tarefa de classificação. Como mencionado anteriormente, existem dois sistemas de reconhecimento de atividades, o sistema principal, composto por 4 atividades (parado de pé, movimento lateral, agachar e saltar) e o sistema secundário, composto por 2 atividades (movimento para a esquerda e para a direita). Para o sistema principal, são extraídas as seguintes características a partir dos dados recolhidos:

Características de domínio temporal

- Média
- Desvio padrão
- Valor mínimo
- Valor máximo
- Coeficiente de correlação entre os pares dos eixos x, y e z.
- Área de sinal-magnitude (SMA)

Características no domínio das frequências

- Magnitudes dos coeficientes do FFT
- Energia
- Entropia

As características de domínio temporal são calculadas a partir dos dados vindos diretamente do acelerómetro e giroscópio. Para além dos valores dos eixos x, y, z destes dois sensores, são calculados os valores das magnitudes entre os eixos, utilizando a seguinte fórmula: $\sqrt{x^2 + y^2 + z^2}$. Assim, serão recolhidas para cada um dos eixos dos sensores, bem como para as suas magnitudes, a média, desvio padrão, valor mínimo e máximo. É também calculado o coeficiente de correlação de Pearson entre o eixo (x, y), (x, z) e (y, z). Este cálculo é obtido através da divisão da covariância entre os dois eixos pelo produto do desvio padrão dos mesmos dois eixos $corr(x, y) = \frac{cov(x,y)}{\sigma x * \sigma y}$ [37]. A área de sinal-magnitude é calculada através da soma das magnitudes.

As características referentes ao domínio das frequências são obtidas através das magnitudes dos coeficientes da FFT. Este algoritmo é utilizado para calcular a transformada discreta de Fourier (DFT) de uma sequência de valores, convertendo um sinal representado no seu domínio temporal para a sua representação no domínio da frequência. No entanto, para que este algoritmo seja eficiente é necessário que o tamanho da sequência de valores que lhe é passado seja uma potência de dois. O resultado obtido através da utilização do método FFT consiste em duas sequências de valores, uma representando a parte real e a outra a parte imaginária do sinal. O resultado final é obtido através do cálculo das magnitudes entre estes dois conjuntos de valores da seguinte forma: $\sqrt{real^2 + imaginário^2}$. Através deste processo, são assim obtidas as magnitudes dos coeficientes do FFT para cada eixo do acelerómetro e giroscópio bem como para as suas magnitudes de cada um destes sensores.

A energia é a soma das magnitudes dos coeficientes resultantes do FFT ao quadrado. Esta soma é depois dividida pelo tamanho da janela utilizada para normalizar o resultado. A entropia é calculada através da fórmula $\mathrm{H(x)} = -\sum_{i=1}^{N} p(xi) \log_{10} p(xi)$ como demonstrado em [38]. Esta característica poderá ser útil na diferenciação de atividades com energias similares [39].

Assim, são obtidas, no total, 568 características para este sistema. Esta quantidade é propositadamente elevada, principalmente devido à inclusão da informação do FFT para os diversos eixos e suas magnitudes, com o intuito de testar posteriormente quanta desta informação é considerada relevante pelo algoritmo de seleção de características que será utilizado no subcapítulo seguinte.

Relativamente ao sistema de reconhecimento da direção do movimento lateral, são utilizadas as seguintes características apenas para o eixo do x do acelerómetro:

- Valores em bruto do eixo do x
- Diferença entre posição do valor máximo e mínimo
- Média do 1º quarto da amostra
- Média do 2º quarto da amostra

- Média do 3º quarto da amostra
- Média do 4º quarto da amostra
- Valor mínimo
- Valor máximo

Para este sistema decidiu-se utilizar toda a informação já obtida relativa ao eixo do x, excluindo informação de frequência, por esta não se mostrar relevante na diferenciação da direção. Foram inicialmente testadas várias características previamente utilizadas como a média, desvio padrão, valor máximo e mínimo, no entanto estes não demonstraram ser relevantes na distinção entre instâncias classificadas como movimento esquerdo e direito. Decidiu-se por isso adicionar outras características que melhor pudessem auxiliar os valores recolhidos em bruto. Através da análise do sinal vindo do eixo x do acelerómetro (ver Figura 5), verificou-se que a posição do valor máximo relativamente à posição do valor mínimo poderia ajudar na distinção da direção do movimento. Na maioria das amostras observadas, o movimento direito caracterizava-se pela presença de um valor máximo precedido de um valor mínimo, assim como o movimento esquerdo era caracterizado pelo comportamento contrário. Subtraindo estas duas posições, podemos obter um valor que caracteriza a posição destes dois valores relativamente um ao outro. Para além desta característica decidiu-se utilizar uma abordagem que incluísse valores médios ao longo do sinal. Visto que o valor da média de uma amostra não era relevante, dado o sinal ser caracterizado por duas variações que se anulam no valor global da média, optou-se por subdividir a amostra em quatro partes (quartos) e calcular as suas respetivas médias. Desta forma obtém-se quatro médias distintas para diferentes fases do sinal, aumentando a capacidade de obter características relevantes na distinção das variações do mesmo. Apesar de individualmente os valores máximo e mínimo não permitirem a distinção dos movimentos, em conjunto com as demais características, estes demonstraram ser uteis, dada a sua relação com as características introduzidas. Desta forma, optou-se também por incluir estas duas características. Ao todo este sistema inclui, por isso, um conjunto de 71 características.

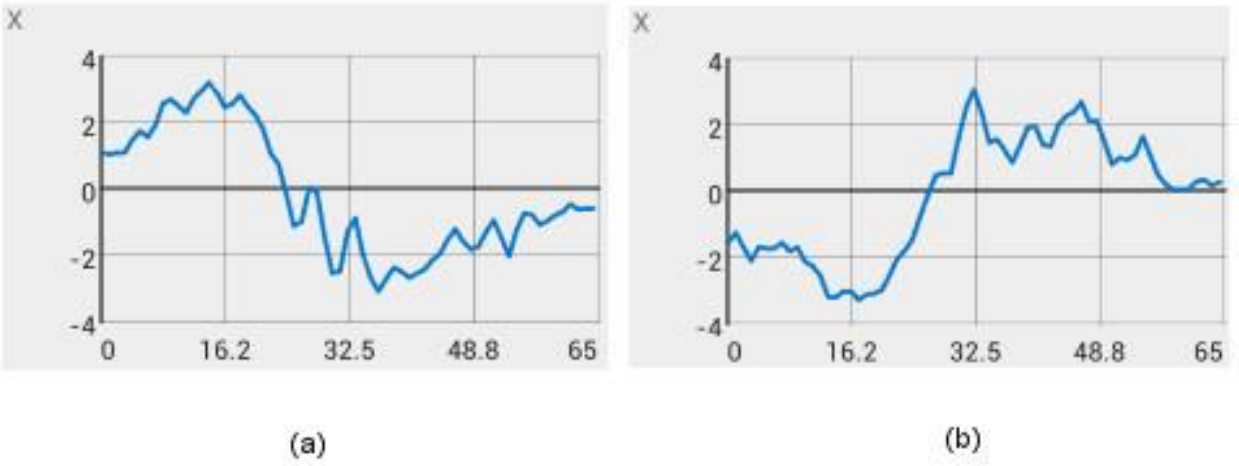


Figura 5 - Sinal típico produzido pelos 64 valores do eixo x do acelerómetro quando é executado o movimento lateral direito (a) e esquerdo (b).

3.5 Seleção de características

Após a recolha e extração de características, é necessário que este conjunto de valores seja diminuído de forma a otimizar o futuro processo de classificação, antes de ser utilizado para treinar o modelo. Esta fase de seleção de características torna-se importante devido ao grande número de características utilizadas. Demasiadas características podem dificultar o trabalho do classificador, tornando-o mais lento e até diminuindo a sua exatidão, no caso de existirem demasiadas características irrelevantes que só contribuem para confundir a classificação.

Para diminuir o conjunto de características utilizadas, foi utilizado um método de seleção baseado em correlação (CFS – *Correlation-based Feature Selection*) [40]. Este método seleciona características altamente correlacionadas com a classe a identificar, mas não correlacionadas entre si. Assim, este método pode indicar rapidamente características irrelevantes, redundantes ou que representem ruído separando assim as características relevantes, desde que sua relevância não seja fortemente dependente de outras características. Em conjunto com o método CFS, podem ser utilizados diferentes métodos de pesquisa. Neste estudo, foram inicialmente testados vários métodos oferecidos pelo Weka, como se pode observar na Tabela III. Para prosseguir o estudo, foi escolhido o método *linear forward selection* (LFS) [41] e um algoritmo genético simples [42]. A escolha deve-se essencialmente ao número de atributos que geraram, tendo o LFS sido o método que gerou o menor conjunto de atributos e o algoritmo genético o que gerou o maior conjunto. Tanto o linear LFS como vários outros métodos de pesquisa presentes no Weka são capazes de gerar conjuntos reduzidos de características. No entanto, reduzir em demasia o conjunto de características também pode introduzir uma diminuição na exatidão posterior do classificador a

utilizar. Por isso, optou-se por testar também o algoritmo de pesquisa genético que produz a menor redução entre todos os métodos. Assim, poderá ser interessante observar como o número de características selecionadas pode afetar os resultados de classificação e obter uma ideia da relevância das características previamente extraídas.

Tabela III - Métodos de procura inicialmente testados no Weka.

Método de procura	***Número de características selecionadas***	
	Sistema principal	**Sistema secundário**
Best first	74	5
Linear forward selection	67	5
Greedy stepwise	74	6
Tabu search	74	5
Genetic search	217	27

3.6 Classificação

Para a fase de classificação, foram testados diferentes tipos de algoritmos disponibilizados pelo Weka, correspondentes a diferentes categorias de classificação. Todos os classificadores foram treinados e testados utilizando o método de validação *10-fold cross validation* disponibilizado no Weka.

Relativamente aos algoritmos baseados em árvores, foi escolhido o *Random Forest* [43]. Este algoritmo consiste num conjunto de *n* árvores de decisão construídas considerando *k* características aleatoriamente. Por defeito, $n = 100$ e $k = \lfloor \log_2(número\ de\ características) + 1 \rfloor$. A classificação resultante é obtida por uma votação de todas as árvores que constituem a floresta, ganhando a classe mais votada.

Relativamente aos classificadores *lazy*, foi escolhido o *K-nearest neighbours* (KNN) [23], cuja implementação no Weka tem o nome de *IBk*. Este algoritmo classifica uma dada instância com base na classificação das *k* instâncias que lhe são mais similares, previamente processadas. O *IBk* consegue selecionar um valor apropriado para o *k*, baseado em *cross-validation* e ainda calcular o peso das distâncias entre os *k* vizinhos mais próximos. Neste estudo optou-se também por utilizar uma estrutura *k-d tree* para o algoritmo de pesquisa do *IBk* em vez do algoritmo *brute force* utilizado por defeito.

Para os classificadores do tipo *function* foi escolhido o *sequential minimal optimization* (SMO) [44], um algoritmo para treino de *support vector machines* (SVM). Este algoritmo resolve problemas de programação quadrática (QP), separando-os em problemas o mais pequenos possível, sendo

responsável por introduzir uma simplicidade conceptual, facilidade de implementação, rapidez e melhores propriedades de escalonamento na resolução de problemas de SVM difíceis em comparação com os algoritmos *standard* de treino de SVM previamente utilizados.

Para complementar o estudo foram ainda escolhidos mais dois classificadores, o *Naive Bayes* [45] pertencente ao grupo de classificadores probabilísticos Bayesianos e o PART [46] pertencente a um grupo de classificadores baseados em regras. Os resultados para cada um destes classificadores são apresentados no próximo subcapítulo.

3.7 Resultados e discussão

A Tabela IV revela a performance dos diversos classificadores anteriormente apresentados, quer individualmente, quer em conjunto com os métodos de seleção de características, para o sistemas de classificação principal e secundário. Na tabela V é apresentada a matriz de confusão para o melhor classificador do sistema principal. A Tabela VI e a Tabela VII apresentam, respetivamente, o segundo e terceiro melhor classificador (com inclusão de seleção de características) pertencentes a diferentes categorias. Analogamente, na Tabela VIII são apresentados os resultados detalhados para os três melhores classificadores. Como é possível observar, para o sistema principal, todos os diferentes classificadores obtêm resultados muito satisfatórias. Ainda assim, o *Random Forest* acaba por se apresentar como o melhor classificador para o sistema principal. Para o sistema secundário, tanto o PART como o *Random Forest* com CFS+LFS para seleção de características, alcançam uma exatidão de 100%.

Para além dos méritos dos algoritmos de classificação e seleção de características, estes resultados também validam de forma muito positiva o conjunto de características extraídas a partir dos dados obtidos através do acelerómetro e giroscópio. Como pode ser observado na tabela 4 apenas o *Naïve Bayes* obtém o seu melhor resultado em conjunto com o método de procura LFS para seleção de características. O *IBk* e PART obtêm o seu melhor resultado em conjunto com o algoritmo de procura genético. Já o SMO e o *Random Forest* obtêm o seu melhor resultado sem qualquer seleção de características. Apesar de a seleção de características providenciar melhores resultados para quase todos os classificadores, os seus melhores resultados foram obtidos maioritariamente com o método de procura genético, que produz uma redução de características muito menor do que o LFS. Isto poderá significar que grande parte das características extraídas é relevante. Contrariamente, no sistema secundário, uma redução mais acentuada como a que o LFS produz revelou ter melhores resultados em quase todos os classificadores. Isto deve-se à inclusão de valores em bruto do eixo do x do acelerómetro, pois grande parte dessas características foi descartada. Já outras características como a diferença entre a posição do valor máximo e mínimo ou as médias dos quartos da amostra foram consideradas bastante relevantes.

Os bons resultados obtidos também validam o treino cuidadoso e minucioso realizado. Ao tentar ao máximo remover o ruido e incluir apenas instâncias o mais limpas possível para cada movimento, contribuiu-se para um sistema mais capaz de identificar as atividades pretendidas. Outro fator a ter em consideração é a junção do movimento lateral esquerdo e direito numa só atividade no sistema principal. Esta decisão poderá ter ajudado a remover alguma ambiguidade que poderia de outra forma ter sido introduzida no sistema principal através da inclusão de duas atividades demasiado similares no treino do seu classificador. Lidar com a distinção destas duas atividades num sistema à parte, com um conjunto de características específico e um classificador dedicado, torna todas as atividades mais distintivas entre si e por isso, mais fáceis de classificar.

Tabela IV - Resultados de classificação para o sistema principal e secundário.

Categoria de classificação	***Classificador***	***Seletor de características***	***Exatidão (Accuracy)***	
			Sistema principal	**Sistema secundário**
Bayes	Naïve Bayes	-	68.63%	99.65%
		CFS + LFS	87.72%	99.72%
		CFS + Genetic	71.72%	99.51%
Functions	SMO	-	98.04%	99.86%
		CFS + LFS	94.30%	99.93%
		CFS + Genetic	97.22%	99.86%
Lazy	IBk	-	95.04%	99.93%
		CFS + LFS	96.00%	99.93%
		CFS + Genetic	96.19%	99.86%
Regras	PART	-	95.95%	100%
		CFS + LFS	96.14%	100%
		CFS + Genetic	96.58%	100%
Árvores	Random Forest	-	99.16%	99.79%
		CFS + LFS	98.40%	100%
		CFS + Genetic	99.11%	99.72%

Para além de bons resultados de classificação, é importante observar as matrizes de confusão geradas para perceber que par de atividades está a ser confundida. Neste estudo, algumas classificações erradas podem ser consideradas "inofensivas". Considera-se um erro "inofensivo"

ou pouco grave qualquer classificação em que o erro esteja na previsão da atividade "estar parado" em vez da correta. Como dito anteriormente, em caso de dúvida a atividade que deve ser identificada é "estar parado". Por exemplo se num jogo se fizer o movimento lateral e o classificador classificar a atividade como "parado", não é tão perigoso como estar parado e a classificação ser "movimento lateral". No primeiro caso, muito provavelmente o movimento não foi bem executado ou não teve intensidade suficiente e o jogador deve repeti-lo. Já no segundo caso, trata-se de um erro grave que é mais difícil de aceitar por parte do jogador. É claro que se um jogador estiver constantemente a fazer uma atividade e o jogo não responder bem prevendo a atividade de ficar parado, isso também poderá ser frustrante para o jogador. Ainda assim, é de esperar que exista alguma confusão entre a atividade de ficar parado e todas as outras pois trata-se de uma atividade mais abrangente e menos distintiva que também inclui movimentos muito similares aos das outras atividades como parte do treino antifraude levado a cabo na fase de recolha de dados. Isto pode ser observado nas matrizes de confusão presentes nas tabelas V, VI e VII.

Tabela V - Matriz de confusão para o classificador *Random Forest* sem seleção de caracteristicas do sistema principal.

Parado de pé	***Movimento lateral***	***Agachar***	***Saltar***	
1620	12	1	4	***Parado de pé***
8	1414	0	0	***Movimento lateral***
8	0	648	1	***Agachar***
0	1	0	459	***Saltar***

Tabela VI - Matriz de confusão para o classificador SMO sem seleção de características do sistema principal.

Parado de pé	***Movimento lateral***	***Agachar***	***Saltar***	
1596	35	5	1	***Parado de pé***
30	1391	1	0	***Movimento lateral***
8	1	648	1	***Agachar***
1	0	0	459	***Saltar***

Tabela VII - Matriz de confusão para o classificador PART com CFS e algoritmo genético do sistema principal.

Parado de pé	***Movimento lateral***	***Agachar***	***Saltar***	
1620	42	10	12	***Parado de pé***
40	1382	0	0	***Movimento lateral***
19	3	630	5	***Agachar***

8	0	4	448	***Saltar***

Tabela VIII - Resultados detalhados para os 3 melhores classificadores pertencentes a diferentes categorias do sistema principal.

Atividade	***Random Forest***			***SMO***			***PART (CFS+Genetic)***		
	Precision	Recall	F-measure	Precision	Recall	F-measure	Precision	Recall	F-measure
Parado de pé	0,990	0,990	0,990	0,976	0,975	0,976	0,959	0,961	0,960
Movimento lateral	0,991	0,994	0,993	0,975	0,978	0,976	0,968	0,972	0,970
Agachar	0,998	0,986	0,992	0,991	0,986	0,989	0,978	0,959	0,968
Saltar	0,989	0,998	0,994	0,998	0,998	0,998	0,963	0,974	0,969
Média	0,992	0,992	0,992	0,980	0,980	0,966	0,998	0,966	0,966

Para complementar este estudo e melhor escolher o classificador a utilizar posteriormente, optou-se ainda por testar a performance de cada classificador em termos de tempo de classificação para o sistema principal. Isto torna-se importante tendo em consideração que o jogo necessita de prever as atividades do utilizador em tempo real. Apesar de o tempo de classificação não ser o único elemento que irá influenciar a performance do sistema, é um elemento importante a ter em consideração para decidir qual o classificador mais indicado. O teste foi efetuado utilizando os dados de treino num computador portátil com um processador Intel Core 2 Duo T5750. O teste foi repetido 10 vezes para cada classificador sendo calculado o tempo médio de criação e teste dos classificadores. Como pode ser observado na Tabela IX, todos os classificadores obtêm tempos bastante satisfatórios, tendo o PART sido o que se destacou em termos de tempos de classificação, logo seguido pelo *Random Forest*. É também possível concluir que a seleção de atributos ajuda a reduzir os tempos de classificação consideravelmente. Considerando tanto os resultados de classificação como os tempos ópidos por cada classificador, o *Random Forest* parece ser a melhor escolha para o sistema principal. Para o sistema secundário, tanto o *Random Forest* + LFS como o PART podem ser utilizados pois são extremamente exatos e rápidos.

Tabela IX - Tempos de criação e classificação para cada classificador do sistema principal em milissegundos.

Classificador	*Seletor de características*	*Média de tempo de criação*	**Média de tempo de classificação**	*Media de tempo de classificação por instância*
Naïve Bayes	-	1852	9732	2.33
	CFS + LFS	18903	731	0.18
	CFS + Genetic	128425	3556	0.85
SMO	-	101354	794	0.19
	CFS + LFS	20891	97	0.02
	CFS + Genetic	141499	334	0.08
IBk	-	22624	2712	0.65
	CFS + LFS	18337	318	0.08
	CFS + Genetic	121959	1232	0.30
PART	-	36441	196	0.05
	CFS + LFS	20616	73	0.02
	CFS + Genetic	111257	68	0.02
Random Forest	-	13891	718	0.17
	CFS + LFS	21491	429	0.10
	CFS + Genetic	85616	509	0.12

4 Biblioteca de reconhecimento de atividades

Após o estudo do sistema de reconhecimento de atividades a implementar, decidiu-se criar uma biblioteca *open source* que permitisse a reutilização do código em projetos independentes. Assim, em vez de se implementar o sistema especificamente para o jogo a desenvolver, optou-se por uma abordagem mais modular, criando uma API que pretende facilitar a criação e utilização deste tipo de sistemas de uma forma mais genérica. Assim, pretende-se também contribuir para que outros programadores possam utilizar este tipo de tecnologia e tirar proveito deste estudo. Em qualquer outra biblioteca de *machine learning* genérica, para criar um sistema de reconhecimento de atividades como o pretendido neste trabalho, seria necessário ao programador criar manualmente estruturas para recolher dados, sobrepor janelas de dados, fazer o seu pré processamento e criar a instância final de características que depois se pretenderia classificar. Caso o uso de subsistemas fosse necessário tudo isto teria de ser feito também para cada um deles. Com a biblioteca a desenvolver, pretende-se que todas estas estruturas e processamento sejam criados automaticamente com o mínimo esforço por parte do programador através de uma interface simples e fluída. Pretende-se também que a utilização do sistema, uma vez criado, seja o mais simples e direta possível, possibilitando treinar e testar as atividades pretendidas com recurso a poucas linhas de código. Tendo isto em consideração, convém esclarecer que a biblioteca desenvolvida não pretende substituir uma ferramenta genérica de *machine learning* como o Weka. Pretende sim ser-lhe um complemento focado na criação de sistemas de reconhecimento de atividades e na extração de características dos dados recolhidos através de sensores. Apesar disto, poderá ainda assim ser perfeitamente utilizável em outros contextos não explorados neste trabalho. Assim sendo, os seus principais objetivos e funcionalidades são:

- Facilitar a criação de um sistema de reconhecimento de atividades complexo, previamente estudado (no Weka ou outra ferramenta) que pode incluir subsistemas;
- Facilitar a extração de características dos dados através de um conjunto de extratores já implementados;
- Facilitar a criação/personalização de extratores de características que rapidamente são integrados no processo de criação do sistema desejado;
- Apesar de o foco neste momento estar nos dados numéricos, a biblioteca deverá permitir que se possa adicionar e processar tanto dados numéricos como alfanuméricos;
- Leitura e persistência de instâncias de características em ficheiros ARFF e CSV;
- Permitir a utilização de algoritmos de classificação e seleção de características disponíveis no Weka.

4.1 Identificação de requisitos

Tendo em consideração os objetivos e funcionalidades anteriormente enunciados, bem como o processo típico de reconhecimento de atividades e as necessidades específicas deste projeto abordadas no capítulo anterior, procedeu-se à recolha dos requisitos através de uma descrição de alto nível para a biblioteca a desenvolver. Esta descrição contempla fases distintas da utilização da biblioteca. Em primeiro lugar, a criação do sistema e subsistemas adicionais através de construtores apropriados. Depois a utilização do sistema, contemplando as suas diferenças de utilização no modo de treino e no modo de teste. Para além disto, foram também identificados requisitos não funcionais importantes para atingir os objetivos da biblioteca.

4.1.1 Construtor de um sistema

O construtor do sistema é a estrutura responsável pela criação coerente de um sistema de reconhecimento de atividades. Tem os seguintes requisitos:

1. Recetor de dados. Este tipo de objeto é responsável por receber e guardar os dados em bruto vindos dos sensores. Contempla os seguintes parâmetros:
    a. Nome identificador do recetor. Por exemplo: “acelerómetro”.
    b. Nomes identificadores das dimensões do recetor. Por exemplo: “x”, “y”, “z”.
    c. Comprimento da janela de dados.
    d. Quantidade de sobreposição entre novas janelas de dados.
2. Extrator de características. Este tipo de objeto é responsável por processar os dados e extrair os valores que formarão a instância final de características do sistema. A ordem pela qual os extratores são adicionados ao sistema é a ordem pela qual serão executados. Contempla os seguintes parâmetros:
    a. Nome do extrator e a respetiva classe.
    b. Objeto(s) com os dados que se pretende serem processados pelo extrator de características. Um extrator pode receber dados de dimensões e/ou resultados de outros extratores de características anteriores a ele.
    c. Opção de adicionar o extrator de características sem que o seu resultado seja adicionado à instância de características finais.

    Restrições:

    d. Um extrator de características só pode implementar um construtor.
    e. Os objetos passados para o extrator de características devem ser coerentes com os esperados pelo seu construtor.
3. Seletor de características. Este tipo de objeto é responsável por diminuir o número de características do sistema a utilizar na classificação, selecionando para isso as mais relevantes. Esta funcionalidade tem como objetivo tornar clara a etapa de seleção de

caraterísticas no construtor do sistema, no entanto deverá ser possível adicionar um classificador do Weka que já possua seleção de atributos integrada. Nesse caso esta funcionalidade não deve ser utilizada. Contempla o seguinte parâmetro:

   a. Objeto adaptador que permite integração com algoritmos de seleção de características do Weka.

4. Classificador. Este tipo de objeto é responsável por fazer a classificação dos dados devolvendo como resultado uma das atividades definidas no sistema.
   a. Objeto adaptador que permite integração com algoritmos de classificação do Weka.
5. Categorias ou classes das atividades a classificar pelo sistema. São identificadores das atividades que o sistema deve determinar como sendo a atividade que está a ser executada. Cada classe deve ser única para um dado sistema. Exemplo: “standing”, “side”, “squat”, “jump”.
6. Subsistema. O construtor do sistema pode receber um construtor de um subsistema, ficando responsável pela criação deste. O nome identificador do construtor de um subsistema deverá ser igual ao de uma das atividades existentes no sistema que recebe esse subsistema.
7. Delimitador de categorias. É possível definir como o sistema deve delimitar nomes de subactividades. No caso de uma classificação devolver um resultado que engloba um subsistema, as categorias e subcategorias são concatenadas através de um delimitador como por exemplo “_”. Assim, se um sistema é classificado com “side” e o seu subsistema com “left” a classificação final é “side_left”. Este delimitador pode também ser utilizado pelo sistema sempre que este necessite de delimitar algum nome. Por exemplo, quando é extraído mais do que um valor para determinado extrator de características ou em nomes de ficheiros para subsistemas.
8. Reutilização de instâncias. É possível definir se se pretende reutilizar a mesma instância ou ir guardando em memória as instâncias de características que vão sendo criadas/utilizadas pelo sistema.
9. Criar sistema. O sistema descrito é verificado e criado bem como os seus subsistemas.

4.1.2 Construtor de um subsistema

A construção de um sistema e de um subsistema deverá ser diferenciada. Isto porque apesar de semelhantes, o construtor de um subsistema deverá ter algumas restrições:

1. Não deverá permitir a criação do subsistema delegando essa tarefa para o sistema que recebe o construtor.
2. Não deve permitir adicionar um recetor de dados, uma vez que os recetores dos subsistemas serão os mesmos do sistema principal.
3. A opção de definir o delimitador de categorias apenas deve estar disponível no sistema principal.

4.1.3 Utilização do sistema

Após criação de um sistema este pode ser utilizado essencialmente para dois fins: treinar atividades e testar atividades. Inclui por isso os seguintes requisitos:

1. Adicionar dados. Permite adicionar dados vindos dos sensores ao sistema. Contempla os seguintes parâmetros:
    a. Identificar o recetor de dados. Pode ser através do seu índice ou nome.
    b. Adicionar os dados respetivos a cada dimensão. O número de dados introduzidos deve ser igual ao número de dimensões do recetor indicado.

    Restrições:

    c. Um recetor assim que atinja a sua capacidade máxima (janela de amostragem) não pode receber mais dados até que o sistema o reinicie. Esse reinício é feito após a extração de características no modo de treino do sistema ou após uma classificação no modo de teste.
2. Obter informação de sistema cheio. Permite determinar se todos os recetores do sistema atingiram a sua capacidade máxima (dados preenchidos para uma instância).
3. Mudar valor da sobreposição da janela de amostragem de dados para um recetor.
4. Mudar modo atual do sistema. Os modos disponíveis são:
    a. Modo de treino. Este modo permite treinar o sistema com os dados vindos dos sensores.
    b. Modo de teste. Este modo permite classificar dados vindos dos sensores após o sistema ter sido treinado.
5. Carregar instâncias para o sistema a partir de ficheiro. Permite ler instâncias guardadas num ficheiro ARFF ou CSV. Contempla as seguintes possibilidades de parâmetros:
    a. Nome do ficheiro. Deve conter o seu caminho e a sua extensão. Caso exista um subsistema e na mesma diretoria esteja um ficheiro cujo nome esteja delimitado corretamente carrega automaticamente esse ficheiro para o subsistema.
    b. Nome do ficheiro e nome do sistema/subsistema. Idêntico ao anterior, mas apenas carrega as instâncias para o sistema/subsistema especificado.
    c. Nome da diretoria e tipo de ficheiro. Em alternativa aos parâmetros anteriores, é possível definir um nome da pasta onde estão os ficheiros do sistema e subsistemas. Se estes corresponderem aos nomes dados ao sistema e subsistemas serão automaticamente carregados.
    d. Anexar. Cada opção anterior permite definir se se pretende anexar as instancias a carregar às existentes no sistema.
6. Guardar instâncias do sistema para um ou mais ficheiros. Permite guardar as instâncias criadas pelo sistema num ficheiro ARFF ou CSV. Contempla as seguintes possibilidades de parâmetros:

   a. Nome do ficheiro. Deve conter o seu caminho e a sua extensão. As instâncias do sistema principal são guardadas no ficheiro especificado. Caso existam subsistemas são automaticamente criados ficheiros com o nome delimitado de acordo com delimitador do sistema e os nomes dos subsistemas.
   b. Nome do ficheiro e nome do sistema/subsistema. Idêntico ao anterior mas apenas guarda as instâncias do sistema/subsistema especificado.
   c. Nome da diretoria e tipo de ficheiro. Em alternativa aos parâmetros anteriores, é possível definir o nome da pasta onde se pretende guardar as instâncias. O sistema encarrega-se de criar os ficheiros com os respetivos nomes.
   d. Anexar. Cada opção anterior permite definir se se pretende anexar as instancias a guardar às existentes num ficheiro.
7. Guardar instância atual do sistema para ficheiro. Permite guardar uma instância de cada vez. Este modo de escrita de instâncias requer que o utilizador abra um "escritor" de instâncias e depois o feche assim que não seja necessário escrever mais.

Apenas em modo de treino:

8. Extrair características. Extrai as características que formam uma instância através da execução dos extratores anteriormente adicionados ao sistema, responsáveis pela transformação dos dados recolhidos.
   a. Categoria ou classe a extrair. Elemento textual que representa a atividade que se pretende associar aos dados que se acabou de recolher após extração de características. Caso se refira a uma atividade com subactividades, deve-se indicar cada categoria separada por vírgulas ou apenas utilizando o delimitador definido no construtor do sistema para efetuar a separação.

   Restrições:

   b. Só pode ser efetuado após todos os recetores de dados do sistema terem atingido a sua capacidade máxima para uma instância.
   c. O nome da classe a extrair tem de corresponder a uma das categorias existentes no sistema.
9. Treinar classificador. Treina o classificador com as instâncias de características presentes no sistema.

Apenas em modo de teste:

10. Classificação. A classificação produz um elemento textual que representa a classe ou atividade a que os dados recolhidos melhor se associam. Inicialmente faz a extração de características, gerando a instância de valores que depois é classificada através do algoritmo de classificação escolhido para o sistema. A instância classificada recebe como

resultado dessa classificação o nome da categoria a que pertence. Existem duas possibilidades para a classificação:

a. Classificação simples. Apenas devolve a atividade mais provável, resultante da classificação.
b. Classificação com probabilidades. Para além de devolver a atividade mais provável, o sistema cria as probabilidades para as diferentes classes envolvidas.

Restrições:

c. Só pode ser efetuado após todos os recetores de dados do sistema terem atingido a sua capacidade máxima para uma instância
d. O classificador do sistema tem de estar treinado.

4.1.4 Requisitos não funcionais

Para além do comportamento específico é necessário determinar de que forma deve operar a biblioteca. Isto é especialmente importante no que toca à tomada de decisões relativamente à arquitetura a seguir. Assim, existem essencialmente três elementos importantes a ter em consideração:

1. Facilidade de utilização. A API a desenvolver deve ser simples e intuitiva mesmo para utilizadores que não estão familiarizados com o processo de reconhecimento de atividades ou com a aplicação de *machine learning*. Isto significa que após apresentação dos métodos e um exemplo prático de utilização, um programador deve ser capaz de criar treinar e testar um determinado sistema de reconhecimento de atividades que deseje.

2. Rapidez. Uma vez que uma possível utilização da biblioteca poderá ser em jogos em tempo real, esta tem de ser capaz de processar e classificar os dados recolhidos em paralelo com o processamento do jogo sem diminuir a performance deste. Isto significa não diminuir o número de *frames* por segundo do jogo em causa quando se aplicar o sistema de reconhecimento de atividades desejado neste projeto.

3. Java e *open source*. Uma vez que existe uma dependência nos algoritmos do Weka e o jogo a desenvolver tem como objetivo o sistema operativo *Android*, a linguagem de programação a escolher é o Java. Pretende-se também que a biblioteca seja disponibilizada como um projeto *open source*.

4.2 Dependências

Neste capítulo são apresentadas as dependências externas utilizadas pela biblioteca a desenvolver.

4.2.1 Weka

O **W**aikato **E**nvironment for **K**nowledge **A**nalysis (Weka) é um *Workbench* ou *suite* de ferramentas de *machine learning* escritas em Java. Esta escolha prende-se com o facto de todo o estudo anterior ter sido efetuado com recurso aos algoritmos presentes nesta ferramenta, sendo conhecidos os seus resultados e performance. No entanto, é de notar que esta ferramenta é de difícil integração noutros projetos devido ao facto de a sua ênfase ser colocada no *toolset* desenvolvido e não na API Java. Um exemplo claro é a não disponibilização de uma API Java limpa e utilizável que não inclua os elementos respetivos à interface gráfica da ferramenta. Isto acarreta vários problemas principalmente quando se tenta utilizar a biblioteca no sistema operativo *Android*, onde vários elementos não são compatíveis. Para colmatar este problema foi necessário compilar uma versão própria do Weka onde foram removidas todas as classes referentes a elementos da interface da ferramenta. Assim, apesar de ser uma ferramenta muito interessante para testar rapidamente ideias, estudar e aprender conceitos de *machine learning*, dificulta a sua utilização puramente como biblioteca. Na biblioteca desenvolvida, utilizou-se o Weka para seleção e classificação de características bem como leitura e escrita de ficheiros ARFF.

4.2.2 Commons Math

O Commons Math [47] é uma biblioteca leve cujo foco é disponibilizar métodos para os problemas mais comuns nos domínios de matemática e estatística, não disponíveis na linguagem de programação Java. É essencialmente utilizado pelas classes de extração de características presentes na biblioteca, como por exemplo no cálculo do desvio padrão e correlação de Pearson.

4.2.3 Commons CSV

O Commons CSV [48] é uma biblioteca para ler e escrever vários formatos de ficheiros CSV. Começou com o objetivo de criar uma interface simples para ler e escrever ficheiros do tipo CSV segundo uma licença Apache (ASL). É utilizado pela biblioteca para fornecer uma alternativa ao formato ARFF utilizado pelo Weka, sem depender deste.

4.3 Arquitetura

Neste capítulo são abordadas e descritas as decisões mais importantes tomadas relativamente à biblioteca desenvolvida. Para complementar o conteúdo descrito e obter uma visão mais detalhada da estrutura e organização das classes foram disponibilizados em anexo diferentes diagramas. No anexo B, é possível consultar um grafo alto nível representativo da hierarquia de classes da biblioteca. No anexo C, são apresentados vários diagramas de domínio que detalham o relacionamento entre classes por pacote. No anexo D, é possível observar alguns excertos de código exemplificativos da utilização da biblioteca.

A arquitetura de *software* tem como objetivo fundamental debruçar-se sobre as decisões estruturais que devem ser tomadas e cuja mudança posterior à sua implementação se torna cara. Os requisitos não funcionais anteriormente identificados são os que mais influenciam estas decisões. Assim sendo, os objetivos para a arquitetura da biblioteca são:

- Criar uma interface simples de utilizar;
- Criar código que não comprometa a performance;
- Criar estruturas fáceis de manter e estender.

4.3.1 Construtores de sistema, subsistema e extrator de características

Uma das partes mais importantes na implementação de um sistema de reconhecimento de atividades é a sua construção, antes sequer de qualquer utilização. Como anteriormente referido, através de uma biblioteca de *machine learning* genérica, um programador teria de criar manualmente um conjunto de estruturas para lidar com os dados, sobrepor janelas se necessário, fazer o pré processamento e criar a instância final de características para só então a poder classificar. Caso o uso de subsistemas fosse necessário tudo isto teria de ser feito também para cada um deles. Assim, no que toca à construção de um sistema, o que se pretende é que todas estas estruturas sejam criadas automaticamente com o mínimo esforço por parte do programador através de uma interface simples e fluida. Para além disto, pretende-se controlar a forma como os objetos são criados e interligados sem que o utilizador necessite de ser exposto a detalhes internos. Apenas necessita de configurar o sistema pretendido e ele será criado de acordo com os parâmetros indicados.

Para conseguir isto, decidiu-se implementar um construtor para o sistema através do padrão *Builder* [49]. Este padrão permite criar um objeto de forma faseada através de uma interface fluida tornando o processo de criação simples e legível. Este esquema é ideal para o tipo de sistema que se pretende por este possibilitar um processo de construção bem definido, com fases concretas que devem ser definidas antes da criação de um objeto. Assim sendo, as suas principais vantagens relativamente a um construtor normal de uma classe são:

- É possível controlar a forma como os objetos de um sistema são criados, encapsulando o processo de criação.
- Permite uma criação faseada do sistema.
- Não permite criar um objeto que esteja num estado inconsistente.

Para além da construção de um sistema existe também a construção de um subsistema. Um subsistema não é mais do que um sistema que está contido noutro. Apesar de não necessitar de uma classe específica que o represente, a sua construção deve ser diferenciada da construção de um sistema, tal como foi indicado no levantamento de requisitos. Atendendo ao princípio de não

repetição de código (***D**on't **R**epeat **Y**ourself*), é necessário criar um *builder* abstrato com os métodos comuns aos *builders* concretos que o irão estender. No entanto, isto cria um problema relativamente à interface fluida que se pretende para o padrão *Builder*. Uma interface deste tipo requer que o objeto que a implementa seja devolvido em cada um dos seus métodos para possibilitar um encadeamento de chamadas. Assim sendo, que objeto deve o *builder* abstrato devolver para os seus métodos? Não poderá devolver a classe abstrata pois isso excluiria os métodos específicos de cada um dos *builders* concretos. Para solucionar este problema optou-se por uma abordagem em que são utilizados genéricos em conjunto com um método reimplementado nos *builders* concretos que devolve o objeto correto, como descrito nos seguintes artigos online [50] [51]. Através desta abordagem é possível não especificar o objeto a devolver pelo *builder* abstrato e delegar essa especificação para os *builders* concretos. Consegue-se uma solução que remove possíveis duplicações de código e permite que que as funcionalidades de sugestão e conclusão automática de código do IDE utilizado funcionem normalmente.

Um dos métodos incluído na construção de um sistema é a criação de um extrator de características. Este tipo de objeto contempla um número variável de elementos que podem ser passados e por isso, optou-se também por efetuar a sua construção através de um *builder*. Desta forma pretende-se integrar a construção de um extrator de características na construção do sistema mantendo o encadeamento e por isso a interface fluida. Para que isto seja possível, devido às questões anteriormente abordadas, criou-se um *builder* extrator de características abstrato e outros dois concretos, um para o construtor de sistema e outro para o construtor de subsistema. Estes dois apenas se diferenciam pela devolução do construtor indicado quando a criação do extrator de características é finalizada e se pretende voltar para a criação do sistema ou subsistema em que se estava.

Tudo isto permite que seja possível um encadeamento de chamadas aos métodos do *builder* permitindo sob o ponto de vista do utilizador, uma interface única e fluida como demonstra o código de exemplo apresentado no anexo D. É também possível verificar as interações anteriormente descritas através do diagrama de sequência representado na Figura 6.

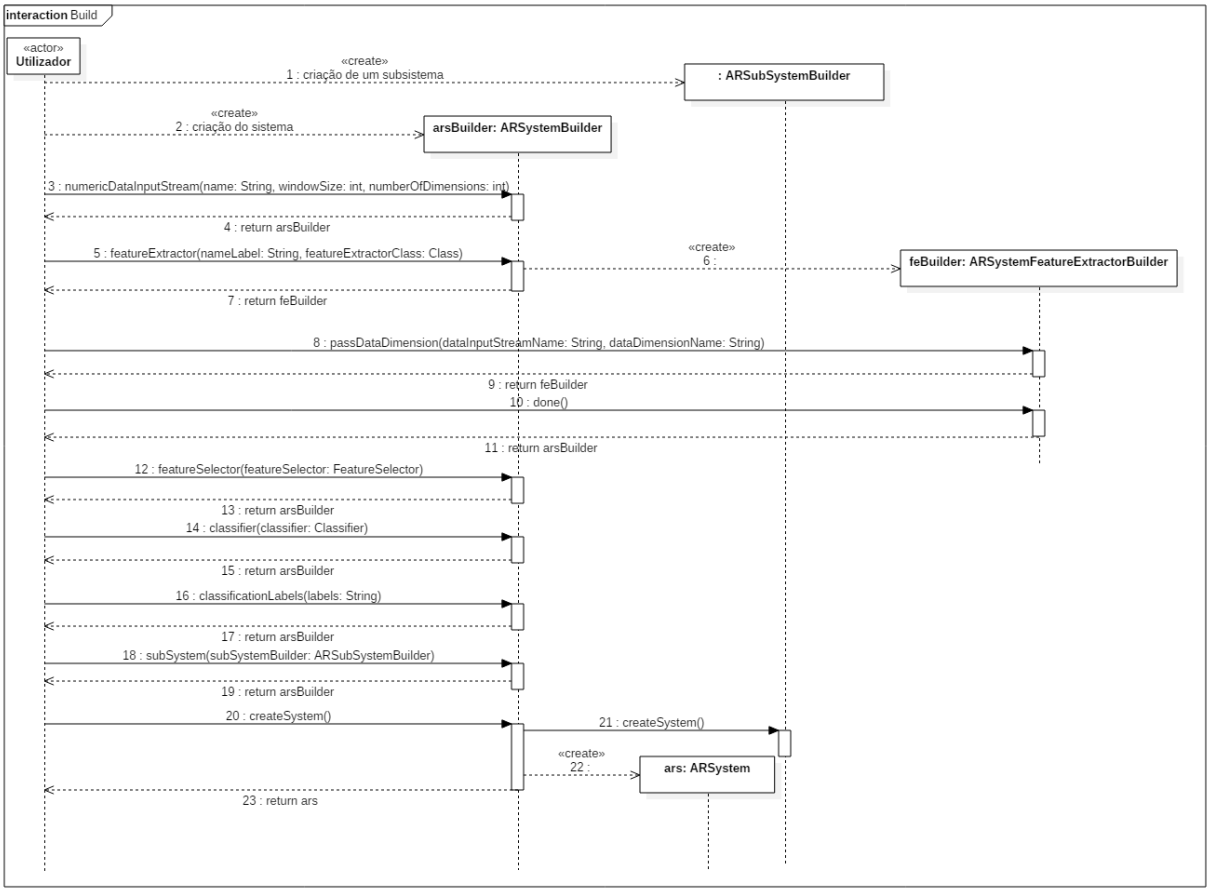


Figura 6 - Diagrama de sequência exemplificativo para a construção de um sistema.

Após definição do sistema pretendido, é chamado o método de criação do sistema. É neste método que está delegada a tarefa de verificar e criar o sistema descrito bem como os seus subsistemas. Tanto recetores de dados como extratores são criados e interligados quando necessário. Uma instância final de características é criada após os extratores serem percorridos e testados para determinar o número total de caraterísticas que esta vai conter. Existe uma ênfase na criação de elementos estáticos como a instância final de características, sem que o programador tenha que saber qual vai ser o seu tamanho final. Isto porque esse tipo de informação é determinado automaticamente na altura de criação do sistema.

Dado que um dos principais requisitos é o processamento em tempo real devido às especificidades da prova de conceito a desenvolver, é necessário que exista um esforço para que durante a utilização do sistema os vários objetos e estruturas sejam reutilizados ao máximo. Desta forma, a criação de objetos é na sua maioria efetuada durante a construção do sistema.

4.3.2 API do sistema de reconhecimento de atividades

Após criação do sistema, existem essencialmente duas formas de o utilizar. Uma é treiná-lo com dados recolhidos. Outra é testá-lo, classificando os dados. Com casos de uso tão simples e tendo como objetivo simplificar ao máximo a interface a ser utilizada, decidiu-se ter uma interface única que não requer qualquer exposição a elementos internos utilizados pela biblioteca através da implementação do padrão fachada [49]. Entre outras, as vantagens de utilizar este padrão são:

- Esconde as complexidades e detalhes de implementação do sistema providenciando uma interface mais simples ao utilizador.
- Torna a biblioteca mais fácil de utilizar e entender disponibilizando métodos convenientes para as tarefas comuns a executar no sistema.
- Junta um conjunto de interfaces numa única API bem desenhada.

O treino utiliza as instâncias rotuladas do sistema para treinar o seu classificador. Estas instâncias podem ser criadas através da introdução de janelas de dados no sistema. Para isso é necessário adicionar dados recolhidos até preencher os recetores do sistema e extrair as características que formam uma instância, indicando qual a atividade que se quer associar à instância. Isto é feito até se ter o número pretendido de instâncias recolhidas e rotuladas. Após esta recolha pode-se pedir ao sistema para treinar o classificador. Em alternativa é possível carregar um conjunto de instâncias de características já recolhidas e treinar o sistema com estas. Convém salientar que para que o método de extração de características e de treino possam ser chamados é necessário que o sistema se encontre no modo de treino. É também necessário que a opção de reutilização da mesma instância por parte do sistema esteja desligada se se quiser treinar o sistema com instâncias recolhidas para que estas fiquem guardadas no sistema. No diagrama de sequência da Figura 7 é possível verificar a ação de recolha e treino descrita anteriormente.

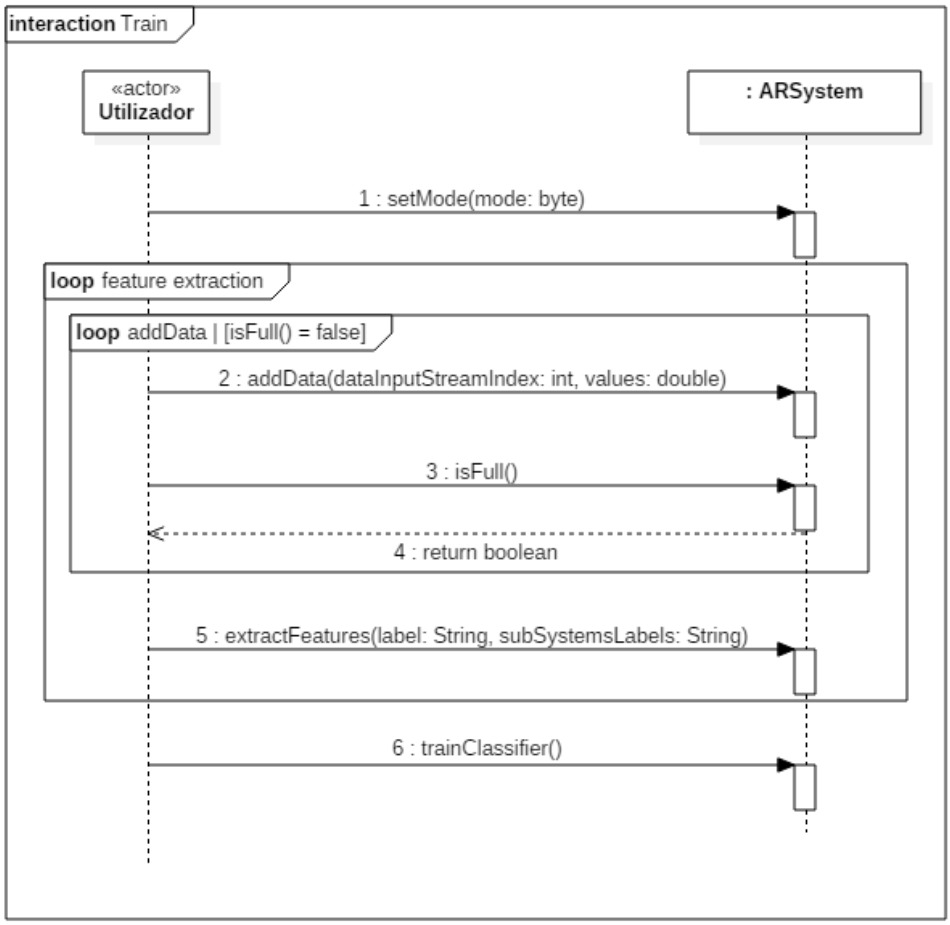


Figura 7 - Diagrama de sequência para o treino do Sistema.

O teste do sistema pode ser efetuado a cada nova janela de dados recolhida. Após todos os recetores de dados estarem completos, é possível chamar o método para classificar instâncias que automaticamente faz a extração de características e consequente classificação com base no treino efetuado. É de salientar que o método de classificação apenas está disponível no modo de teste e após treino do classificador do sistema. No diagrama de sequência da Figura 8 é possível verificar a ação de teste descrita.

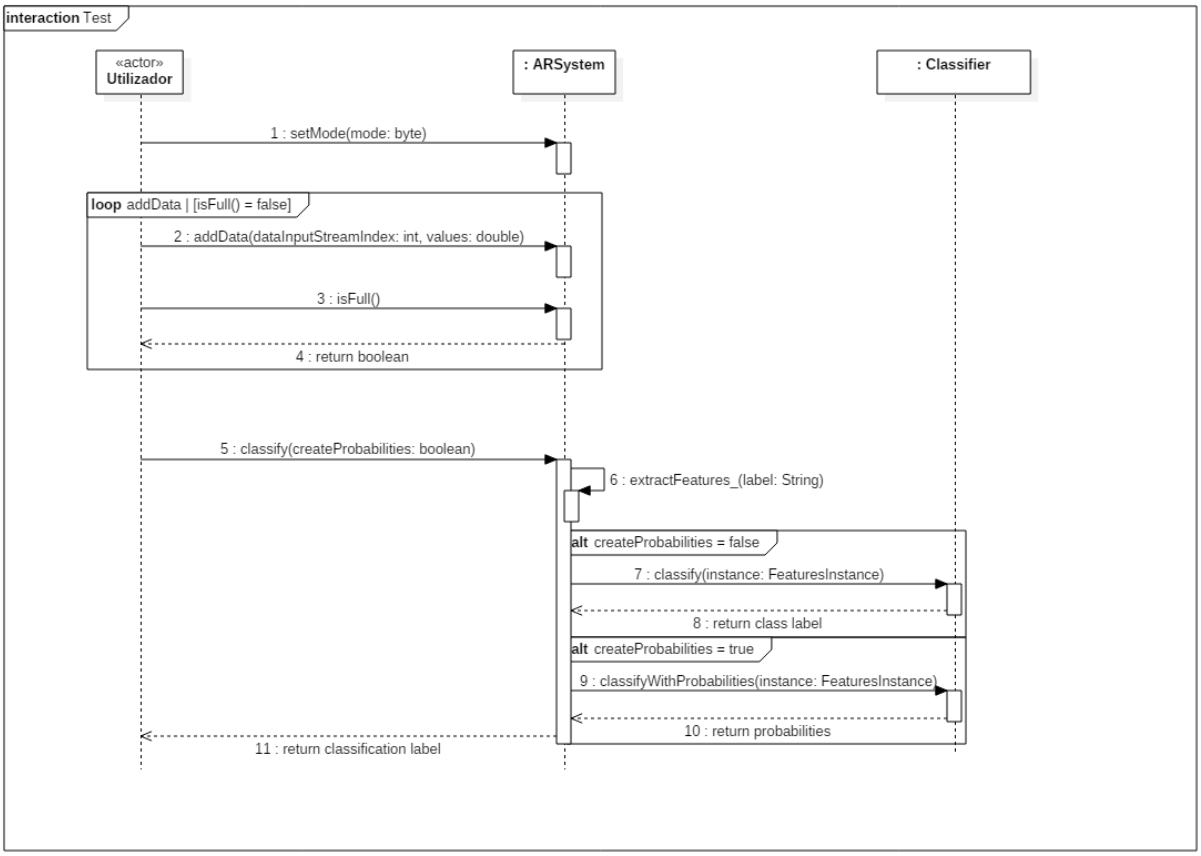


Figura 8 - Diagrama de sequência para o teste do sistema.

É necessário ter em consideração a opção de reutilizar a mesma instância. Esta opção possibilita que o sistema reutilize uma instância em vez de estar sempre a criar uma nova a cada conjunto de dados recolhidos e consequente extração de características. O seu objetivo é minimizar a construção de objetos no caso em que se está a utilizar o sistema para classificar dados e não se pretende guardar esses dados em memória. Por exemplo no jogo a desenvolver, esta vai recebendo os dados dos sensores, classificando-os a cada instância recolhida. Cada classificação gera um consequente movimento na personagem. Assim sendo, a classificação está a ser feita a partir dos dados recolhidos em tempo real, sendo desnecessário estar constantemente a criar e guardar novas instâncias. Para além do mais, mesmo que se queira guardar a instância classificada, é possível guardá-la num ficheiro e reutilizar a mesma instância no sistema. Para isso basta utilizar a função *saveInstance()*. Este método guarda a instancia atualmente criada num ficheiro previamente aberto pela função *openSingleInstanceWriter()*. O programador é responsável por fechar o ficheiro utilizando a função *closeSingleInstanceWriter()* assim que não necessitar de escrever mais instancias.

Para além desta forma de guardar instâncias únicas, a API do sistema contempla ainda a possibilidade de guardar o seu conjunto total de instâncias, assim como, ler um ficheiro e guardar

as suas instâncias no sistema. Em ambos os casos a biblioteca encarrega-se de abrir o ficheiro indicado, ler/escrever as suas instâncias e fechar o ficheiro automaticamente, sem que o programador necessite de se preocupar com isso. No conjunto de requisitos inicialmente recolhidos, as chamadas de escrita e leitura de ficheiros da API contemplam um conjunto variado de possibilidades. Apesar disto pretende-se que estas sejam bastante simples e diretas. Assim sendo, optou-se por criar apenas dois métodos na API, uma para escrita (*saveFeatureInstances*) e outro para leitura (*loadFeatureInstances*). Cada um destes métodos vai, no entanto, contemplar o conjunto de possibilidades descrito através de *method overloading*. Esta funcionalidade presente na linguagem Java permite que sejam desenvolvidos vários métodos com o mesmo nome, mas com parâmetros e implementações diferentes. Sob o ponto de visto do utilizador, o método é o mesmo, apenas contempla configurações diferentes.

4.3.3 Estruturas de recolha de dados e extração de características

O sistema conta com estruturas internas para lidar com os dados recolhidos e possibilitar o conjunto de funcionalidades requerido. Entre outras estruturas, o sistema mantém uma lista de recetores e suas respetivas dimensões, bem como uma lista de extratores, resultados extraídos e de instâncias de características. Todas estas estruturas estão interligadas. Os extratores recebem dimensões e resultados de outros extratores sob a interface *ImmutableData*. Os resultados extraídos de cada extrator referenciam a instância que está a ser utilizada pelo sistema adicionando-lhe o seu valor quando isso é requerido. Todas estas associações entre os objetos são efetuadas na altura de criação do sistema conforme as configurações especificadas no *builder*. Na Figura 9 é possível observar um esboço da sequência de operações efetuadas por diferentes estruturas de dados, desde a introdução de dados no sistema até à criação da instancia final de características.

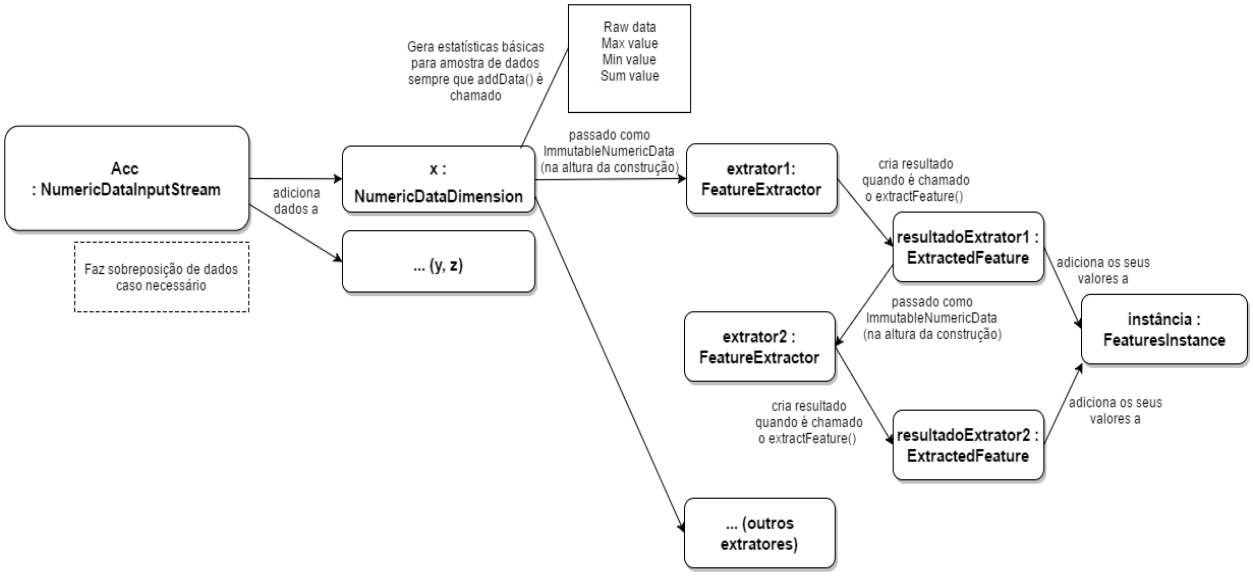


Figura 9 – Sequência de operações e relação entre as estruturas de recolha de dados e extração de características.

Os recetores de dados (DataInputStream) existentes no sistema funcionam como uma metáfora para os sensores de dados que se pretende utilizar na biblioteca. Desta forma, contemplam uma ou mais dimensões onde os dados em bruto são efetivamente guardados. Por exemplo, um recetor que recebe dados vindos de um acelerómetro, irá receber dados de três eixos diferentes: x, y e z. Cada um destes eixos será representado por uma dimensão diferente no recetor.

Ao longo da biblioteca são utilizados *arrays* para guardar os dados em vez das classes da *framework Collections* do Java. Esta opção deve-se essencialmente a dois fatores:

- Os conjuntos de dados a guardar têm tamanhos estáticos após a criação do sistema.
- As classes de coleções no Java só lidam com objetos. Caso se pretenda guardar valores de tipos primitivos neste tipo de estrutura, o Java automaticamente utiliza técnicas denominadas como *boxing* para converter estes dados em objetos ou *unboxing* para converter os objetos para tipos primitivos, o que acarreta custos.

Um recetor de dados apenas lida com objetos do tipo DataDimension, sendo responsável por adicionar os dados que recebe às suas dimensões que implementam um tipo concreto como NumericDataDimension ou StringDataDimension, criados através do padrão *Factory* [49]. A interface DataDimension contem métodos para receber dados tanto numéricos como do tipo *String*, delegando para as classes que a implementem a responsabilidade de implementar o tipo de dados que faz sentido receberem. Esta abordagem foi seguida para permitir que a biblioteca possa lidar com diferentes tipos de dados, não se restringindo a dados numéricos. Uma opção alternativa e que poderia contribuir para uma melhor manutenção do código seria utilizar dimensões que guardassem dados com recurso às capacidades genéricas do Java. No entanto, essa abordagem traria algumas inconveniências devido às limitações da linguagem. Como anteriormente referido, pretende-se evitar o *auto-boxing / unboxing* que ocorreria. Uma vez que existe a necessidade de estar constantemente a processar dados, estas conversões automáticas iriam reduzir a performance.

Outra característica das dimensões de dados é a implementação de um pré processamento de estatísticas básicas comummente utilizadas na extração de características. O objetivo é evitar estar constantemente a percorrer os dados recolhidos e a efetuar cálculos repetidos. Por exemplo, grande parte dos extratores implementados utilizam o valor mínimo ou máximo da amostra. Este tipo de informação é facilmente obtido sem grandes custos de processamento durante a recolha a cada introdução de novo valor. No entanto na extração de características, para obter esta informação a partir da amostra é necessário que esta seja percorrida. Desta forma, cada dimensão terá para além dos dados em bruto, alguns valores pré calculados como o valor máximo, mínimo e soma da amostra. Estes cálculos são efetuados sempre que um novo valor é adicionado a uma dimensão, diluindo assim o seu processamento e evitando ter de se percorrer o conjunto de dados recolhido desnecessariamente quando forem efetuadas extrações de características.

A extração de características é uma das principais funcionalidades da biblioteca. Para além da implementação de uma serie de extratores cuja oferta se tenciona aumentar no futuro, pretende-se também que o programador possa criar facilmente o seu próprio extrator sem a necessidade de expor detalhes internos do sistema.

Um extrator de características permite extrair dados relevantes a partir de um determinado conjunto de dados. Como anteriormente especificado nos requisitos, existem dois tipos de fontes de dados que podem ser enviados para um extrator: dimensões de recetores de dados e resultados de extratores anteriores.

Para implementar um extrator de características personalizado, basta estender a classe de um dos extratores abstratos disponíveis (numérico ou do tipo *String*). Depois é necessário implementar um dos seus possíveis construtores, que definem os dados que o extrator irá receber. Por último é necessário implementar a função de extração que se pretende, adicionando os valores resultantes do processamento efetuado a um objeto que representa um conjunto de características extraídas. Este objeto está automaticamente encarregue de adicionar os resultados à instância final de características do sistema. No anexo D, é possível observar um excerto de código que exemplifica a descrição anterior.

Para possibilitar a passagem dos tipos de dados requeridos para um extrator sem complicar a sua interface decidiu-se criar uma interface comum a implementar pelos objetos. Esta interface tem também como objetivo impedir o acesso a métodos de modificação dos valores internos dos objetos tornando-os imutáveis. Assim tanto as dimensões (DataDimension) como os resultados das extrações (ExtractedFeature) são classes que implementam a interface ImmutableData, mais especificamente um dos tipos ImmutableNumericData ou ImmutableStringData. Este tipo de interface apenas permite acesso aos seus valores. Apesar de os dados de entrada do extrator terem o mesmo tipo, é possível definir se um extrator espera apenas um conjunto de dados ou vários. Para isso, é necessário escolher se se pretende implementar um construtor que recebe um objeto ImmutableData ou um *array* de objetos deste tipo. Isto tem como objetivo restringir a utilização do extrator evitando utilização indevida. Por exemplo um extrator que implementa o cálculo do desvio padrão de uma amostra espera receber um objeto, não vários. Um extrator que implementa o cálculo de correlação de Pearson tem de receber um *array* com pelo menos dois objetos. Caso no momento da construção do extrator de características o número de elementos não estiver correto é gerada uma exceção.

4.3.4 Integração de algoritmos do Weka

Como anteriormente descrito, pretende-se permitir a utilização de algoritmos de classificação e seleção de características disponíveis no Weka. Apesar de para este trabalho isso não ser requerido, no futuro, poderá existir a necessidade da biblioteca implementar os seus próprios algoritmos. Para além disso o código externo deve ser claramente separado do código desta biblioteca. Desta forma,

desenvolveram-se interfaces que abstraem um classificador bem como um seletor de características e implementou-se um classificador adaptador para um classificador do Weka. A utilização do padrão adaptador [49] tem em mente os seguintes objetivos:

- Permitir a integração de elementos externos que oferecem a mesma funcionalidade através de interfaces diferentes.
- Preservar a independência das classes.
- Promover a expansibilidade.

A utilização deste padrão permite assim desacoplar elementos externos salvaguardando o sistema de possíveis alterações aos elementos. Qualquer alteração efetuada apenas terá impacto na classe adaptadora. Assim caso se pretenda implementar um determinado classificador apenas é necessário implementar a interface *Classifier*. O *WekaClassifier* é apenas uma dessas implementações, funcionando como um adaptador para uma configuração de classificação do Weka. Outra implementação é o *FeatureSelectedClassifier* que contempla um classificador e um seletor de características e implementa um classificador que reduz o número de características antes da classificação conforme a sua relevância. As estruturas de seleção de atributos organizam-se de forma análoga às de classificação. A classe *WekaFeatureSelector* representa um adaptador da interface de seleção de características providenciada pelo Weka. É possível observar este relacionamento através da Figura 10.

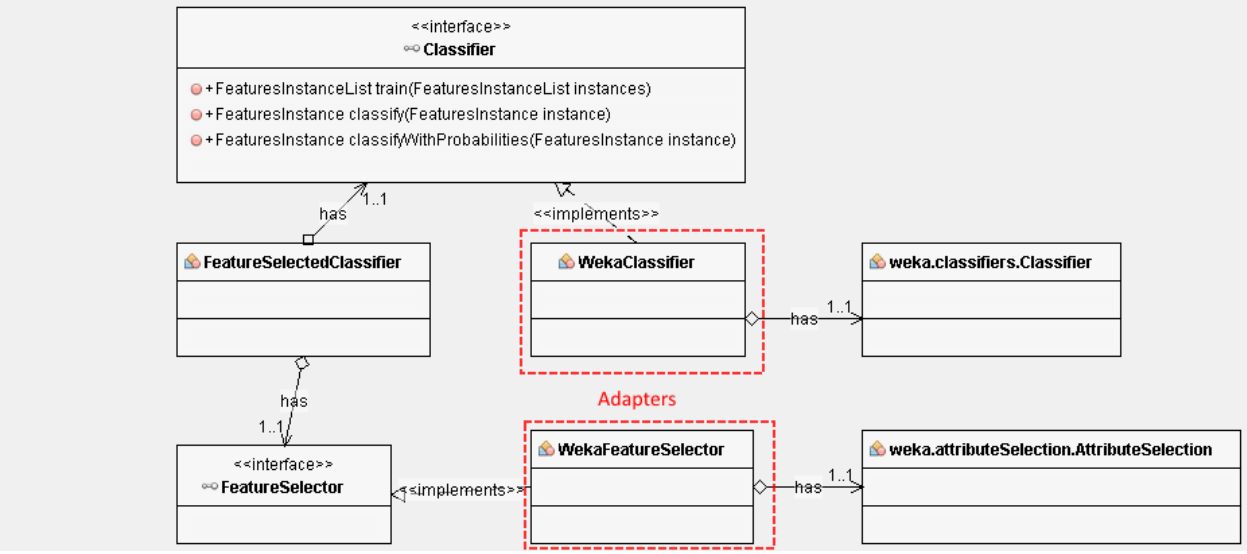


Figura 10 - Representação do padrão adaptador através das estruturas de classificação e seleção de características.

Para permitir que a instância de características produzida pela biblioteca possa ser processada pelos algoritmos do Weka é também necessário que sejam utilizadas as suas estruturas de instâncias. Desta forma, foram criados métodos uteis para conversão das estruturas da biblioteca para as utilizadas pelo Weka. Estes métodos são utilizados pelas classes adaptadores dos algoritmos do

Weka, estando disponíveis de forma estática através de uma classe auxiliar presente no pacote de *helpers* da biblioteca.

4.3.5 Estruturas de leitura e escrita de ficheiros de instâncias

Como anteriormente descrito, a API do sistema disponibiliza métodos para leitura e escrita de instâncias em ficheiros, sendo a sua utilização direta e simples. Para possibilitar estas funcionalidades existe um conjunto de estruturas internas que devem ser analisadas.

Internamente, as classes de leitura e escrita de ficheiros de instâncias organizam-se de forma a promover a extensibilidade e flexibilidade do código. Existe uma interface para leitura de ficheiros de instâncias e outra para escrita, bem como implementações concretas para cada tipo de ficheiro. Apesar disto, para promover o desacoplamento optou-se por criar uma classe para leitura e outra para escrita que funcionam como adaptadores, agregando os tipos concretos de leitura / escrita. No caso de ser necessário modificar algum leitor / escritor de um tipo concreto apenas a classe adaptadora será afetada.

Relativamente à escrita de instâncias existe ainda uma característica que faz com que um tipo concreto tenha de ser implementado contemplando certos passos. Isto deve-se à necessidade de configurar o objeto de escrita de forma diferente conforme se deseje anexar instâncias a um ficheiro existente ou não. Assim sendo, decidiu-se utilizar o padrão *Template* [49], introduzindo uma classe abstrata que implementa a construção de um escritor e obriga uma classe concreta a implementar três métodos que representam diferentes passos utilizados na construção, como pode ser observado na Figura 11.

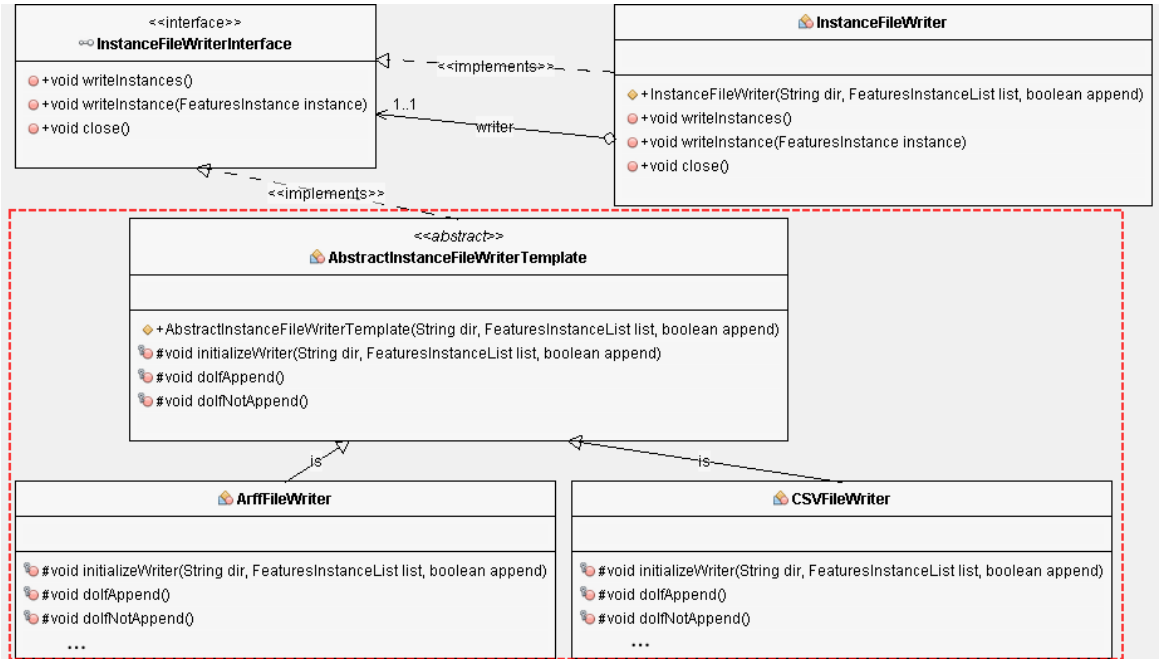


Figura 11 - Representação do padrão *template aplicado à escrita de ficheiros de instâncias.*

5 Desenvolvimento da prova de conceito

Neste capítulo é apresentado um jogo de *endless running* que constitui a prova de conceito deste trabalho. É abordada a ideia pretendia, o desenvolvimento que se seguiu e os resultados após a sua implementação.

5.1 Descrição do conceito inicial

Como anteriormente referido, pretende-se implementar um jogo de *endless running* a três dimensões, para o sistema operativo *Android*, em que o sistema de interação habitual, baseado em toque, é substituído por um sistema de interação baseado no movimento corporal do jogador. A execução desses movimentos por parte da personagem deve representar no mundo virtual o que está a ser feito pelo jogador no mundo real.

Este tipo de jogos caracteriza-se por um caminho e uma personagem que o percorre em corrida constante. Ao longo do percurso surgem obstáculos dos quais a personagem se deve desviar com a ajuda do jogador. O objetivo é manter a personagem a correr, evitando a sua colisão com esses obstáculos. Na implementação pretendida, existe uma pista com três faixas que representa o percurso onde a personagem pode correr. O jogador poderá executar 5 atividades que devem ser representadas pela personagem virtual do seguinte modo:

- Estar parado. No caso de uma pessoa estar parada, a personagem mantêm-se a correr, não executando nenhuma atividade de desvio dos obstáculos.
- Mover para o lado esquerdo. Quando o jogador se move para a sua esquerda a personagem deve fazer o mesmo, deslocando-se para a faixa imediatamente à sua esquerda.
- Mover para o lado direito. Tal como o movimento anterior, quando o jogador se move para a sua direita a personagem deve fazer exatamente o mesmo, deslocando-se para a faixa imediatamente à sua direita.
- Agachar. Quando o jogo deteta um agachamento, a personagem deve fazer o mesmo movimento, mantendo-se na mesma faixa.
- Saltar. Quando o jogo deteta um salto, a personagem deve executar o mesmo movimento mantendo-se na mesma faixa.

Inicialmente a personagem virtual inicia a corrida na faixa central, como demonstra o esboço inicial ilustrado na Figura 12 (a). O jogador pode deslocar-se para uma das outras faixas laterais

sempre que lhe convier e for possível. No caso de estar na faixa da esquerda, já não é possível mover-se mais para a esquerda. O mesmo acontece para a direita, sendo o movimento nesse sentido ignorado. Ao longo do percurso, existem três tipos de obstáculos que obrigam o jogador a executar as diferentes atividades disponíveis no jogo, como ilustra a Figura 12 (b). Seguem-se as suas descrições:

- Obstáculo simples. Este tipo de obstáculo é posicionado no chão de uma das três faixas da pista. Cobre apenas a faixa onde se encontra e impede o jogador de progredir nessa faixa. Obriga por isso a que o jogador se desvie para uma das outras faixas ou salte o obstáculo.
- Obstáculo longo no chão. Este tipo de obstáculo é similar ao anterior no entanto cobre todas as faixas da pista. O objetivo é introduzir um obstáculo no percurso que obrigue o jogador a executar a atividade de salto.
- Obstáculo longo elevado. Este tipo de obstáculo é similar ao anterior no entanto em vez de se encontrar no chão, encontra-se elevado por forma a obrigar que o jogador execute um agachamento.

Caso o jogador não se desvie de um obstáculo atempadamente, é provocada uma queda na personagem virtual o que faz terminar imediatamente a corrida. Sempre que uma corrida termina devem ser mostrados o último e o melhor resultado conseguido no dispositivo. Esse resultado trata-se, apenas, do tempo que a personagem conseguiu ficar a correr sem cair.

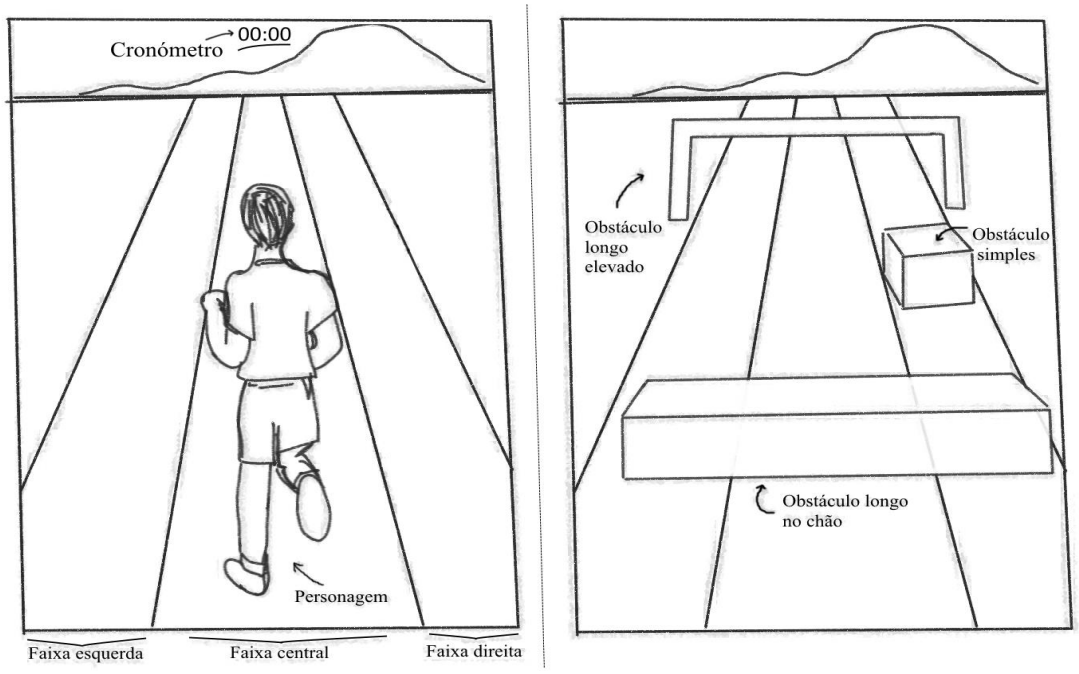


Figura 12 - Esboço inicial da prova de conceito.

5.2 Ferramentas utilizadas

Neste subcapítulo, pretende-se introduzir as ferramentas e tecnologias utilizadas na criação da prova de conceito. Assim sendo, segue-se a lista das principais ferramentas e tecnologias utilizadas:

- MakeHuman [52]. Trata-se de um projeto *open source* para criar modelos tridimensionais e realistas de pessoas, de forma simples. A ferramenta disponibiliza um modelo humano base, andrógino, cuja tipologia pode ser alterada por forma a criar um conjunto variado de personagens quer masculinos, quer femininos. Para além disto, o modelo disponibilizado pelo MakeHuman tem já associado um esqueleto completamente configurado e pronto a animar. Esta ferramenta é, por isso, utilizada para criar o modelo inicial da personagem do jogo.

- Blender [53]. O Blender é um *software* profissional e *open source* de computação gráfica 3D. Permite um conjunto vasto de funcionalidades, entre as quais: modelagem, animação, texturização, composição, *rendering*, edição de vídeo e criação de aplicações interativas em 3D. Para o desenvolvimento da prova de conceito, utiliza-se esta ferramenta essencialmente pelas suas características de modelagem, texturização e animação, bem como pelas suas capacidades de importação e exportação de tipos de ficheiros diferentes para modelos 3D. Permite também uma boa interligação com o MakeHuman, através de *plugins* específicos.

- Base de dados de ficheiros de captura de movimento da Universidade Carnegie Mellon [54]. Esta base de dados contém um conjunto variado de movimentos humanos capturados e disponibilizados em diferentes formatos. Alguns dos ficheiros disponíveis foram importando no Blender e utilizados para criar as animações da personagem do jogo.

- LibGDX [55]. O libGDX é uma *framework open source* de desenvolvimento de jogos em Java. É multiplataforma, permitindo o desenvolvimento tanto de jogos para computador como para dispositivos móveis e Web, através de uma API unificada.

- Libgdx-fbxconv-gui [56]. Trata-se de uma interface gráfica multiplataforma desenvolvida em Java que utiliza a ferramenta fbx-conv [57] para converter alguns formatos de ficheiros de modelos tridimensionais em outros mais simples de processar em *runtime*. Serve também como uma utilidade de *preview* dos modelos, antes de serem importados no libGDX.

- Android Studio. Trata-se do ambiente de desenvolvimento integrado oficial, disponibilizado pela Google, para desenvolvimento nativo na plataforma *Android*. Todo o código do jogo é criado com recurso a este IDE.

5.3 Processo de criação do jogo

Após a criação do esboço inicial para o jogo, segue-se o processo de criação propriamente dito onde primeiro são criados os objetos do mundo tridimensional e depois a implementação da lógica do jogo. Para finalizar, o sistema de reconhecimento de atividades é integrado através da biblioteca previamente desenvolvida neste trabalho.

5.3.1 Criação dos objetos do mundo tridimensional

Após a criação do esboço inicial da prova de conceito, começou-se por criar a personagem do jogo. Inicialmente, utilizou-se a ferramenta MakeHuman para gerar um modelo tridimensional humano com esqueleto pronto a ser animado. De seguida a personagem foi carregada no Blender para ser finalizada e animada. A Figura 13 ilustra a utilização destas ferramentas. Para que as animações fossem o mais realistas possível, optou-se por utilizar ficheiros de captura de movimentos disponíveis *online*, na base de dados da Universidade Carnegie Mellon. Esses ficheiros foram depois importados no Blender e reajustados para atingir o resultado desejado. Algumas animações foram ainda criadas de raiz por não se encontrar ficheiros que as representassem devidamente, como é o caso das quedas da personagem.

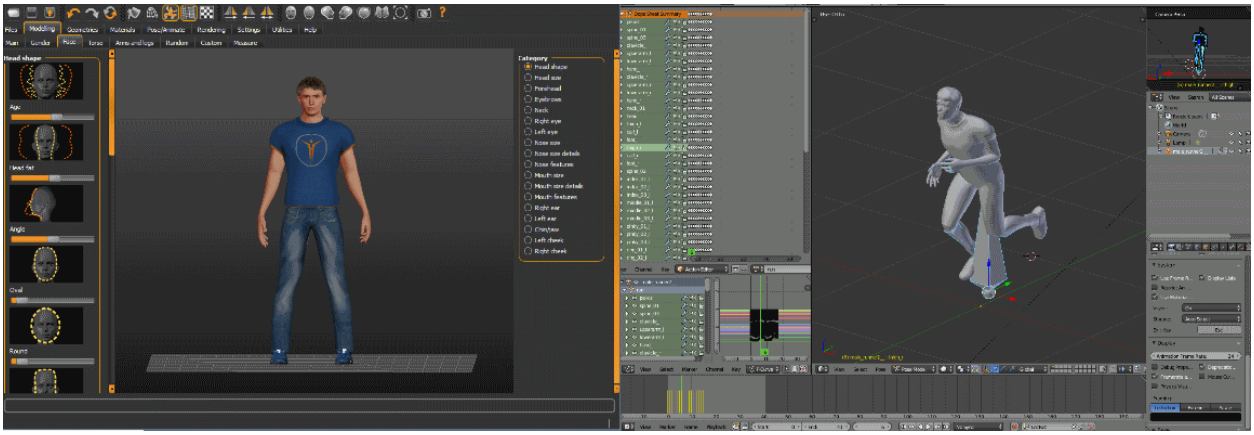


Figura 13 - Ferramentas utilizadas para criar a personagem virtual. À esquerda a ferramenta MakeHuman e à direita a ferramenta Blender.

Depois de criada a personagem virtual, foram também criados os restantes objetos do jogo, como os obstáculos e a pista. Todos estes objetos foram criados manualmente no Blender. Após criação de todos os objetos do jogo, estes foram exportados e convertidos para um formato suportado pelo libgdx através da ferramenta libgdx-fbxconv-gui. Esta ferramenta foi também utilizada para obter uma pré-visualização dos modelos antes de serem importados no libGDX.

5.3.2 Implementação do jogo

Após conversão dos objetos, estes foram carregados utilizando a classe *AssetManager* disponibilizada pelo libGDX. É ainda criada uma classe *World* que representa o mundo tridimensional e é responsável por executar a lógica de cada um dos objetos do jogo, bem como fazer o seu *rendering*. O diagrama de classes referente à lógica de jogo pode ser observado no anexo E. A gestão dos ecrãs presentes na aplicação é feita através de uma máquina de estados implementada internamente pelo libGDX. Para utilizar esta funcionalidade é apenas necessário que a classe principal do jogo estenda a classe *Game* e cada um dos ecrãs ou estados da aplicação implemente a classe *Screen*. Apenas existem quatro estados: ecrã de carregamento, menu inicial, ecrã de jogo e resultados, como ilustrado na Figura 14.

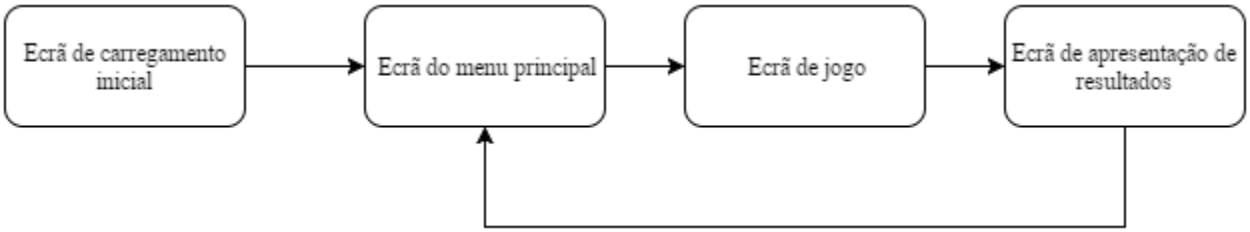


Figura 14 - Máquina de estados representativa dos ecrãs do jogo.

Relativamente à implementação da lógica presente no ecrã de jogo, optou-se por uma abordagem o mais simples possível, dada a sua natureza meramente demonstrativa. Assim, optou-se por colocar a personagem numa posição estática e em constante execução da animação de corrida. A posição da personagem apenas é alterada no eixo do x quando esta se move para um dos lados. Já o objeto que representa a pista desloca-se apenas no eixo do z em direção à personagem como se de um tapete rolante se tratasse. Os obstáculos são colocados aleatoriamente sobre a pista e deslocam-se de acordo com esta. As colisões entre a personagem e os obstáculos são também calculadas de forma simples, sem recurso a motores de física complexos. É apenas calculada a distância entre a posição no eixo do z da personagem e dos obstáculos.

Para poder testar o jogo mais facilmente através do computador utilizado para o seu desenvolvimento, optou-se também por criar um sistema de interação através de *inputs* vindos de um teclado. Este sistema é implementado na classe da personagem através da utilização de métodos utilitários disponibilizados pelo libGDX que determinam se uma tecla foi premida. Em caso afirmativo, dependendo da tecla, é executada uma das atividades disponíveis.

5.3.3 Integração do sistema de reconhecimento de atividades

Implementado o jogo base, abstraído de qualquer sistema de interação, segue-se a integração do sistema de reconhecimento de atividades. Pretende-se que seja implementado um sistema que

reconheça as atividades do jogador como modo de interação de forma semelhante à interação produzida pelo sistema de teclado utilizado para testes. Apesar de este último ter sido inserido na classe da personagem por questões de simplicidade, para o sistema de reconhecimento de atividades optou-se por criar um módulo à parte que é executado em paralelo com a lógica de jogo, como ilustrado na Figura 15.

Este módulo é representado por duas classes que utilizam a biblioteca previamente desenvolvida. A classe *ActivityReconition* é responsável por criar o sistema de reconhecimento de atividades, sendo utilizada no ecrã de carregamento inicial do jogo. A classe *DataCollector* é responsável por recolher amostras de dados vindas dos sensores do *smartphone* e enviá-las para o sistema criado e disponibilizado de forma estática pela classe *ActivityReconition*. Assim que uma janela de dados é completamente recolhida, os dados são processados e gerada uma classificação. Após obtenção do resultado da classificação, é chamado o método que representa a animação da atividade reconhecida presente na classe *Runner* (referente à personagem do jogo). No caso de a atividade ser "estar parado", nada é executado e a personagem continua a correr normalmente.

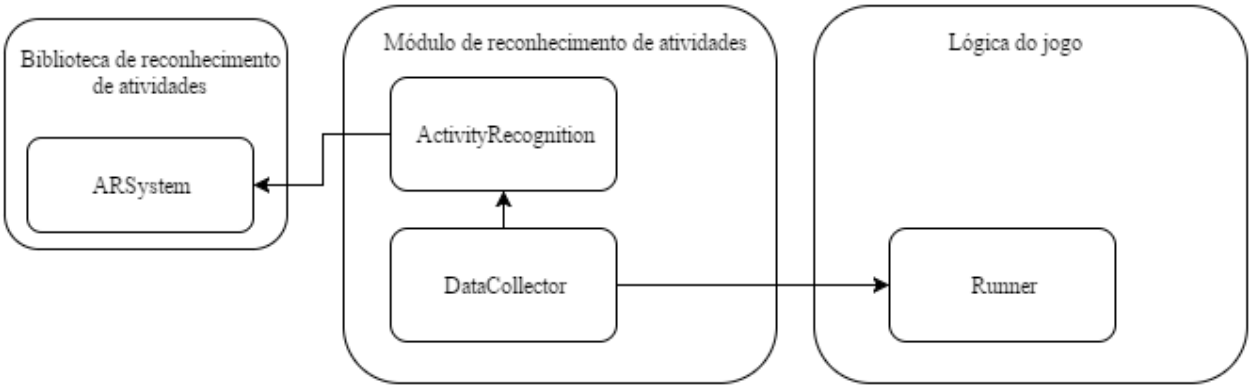


Figura 15 – Diagrama representativo da integração do módulo de reconhecimento de atividades.

5.4 Resultados e reajustes

Após implementação de uma versão inicial do jogo, este foi testado no dispositivo móvel utilizado para treino, bem como noutros dispositivos móveis disponibilizados para teste por alguns voluntários. Os resultados dos vários ecrãs da aplicação podem ser observados através da Figura 16. A Figura 17 ilustra as diversas atividades implementadas quando o jogador se move.

Figura 16 - Os 4 ecrãs do jogo após implementação. Da esquerda para a direita: ecrã de carregamento inicial, ecrã do menu principal, ecrã de jogo e ecrã de apresentação de resultados.

Apesar de uma implementação simples, após alguns testes, o resultado revelou-se bastante convincente, quer a nível gráfico, quer a nível de jogabilidade. Para além dos testes efetuados internamente, foi pedido a algumas pessoas que participaram na fase inicial de treino da aplicação para testar e dar a sua opinião sobre o jogo. Através da observação dos testes de alguns participantes e das suas opiniões, foi possível levantar um conjunto de alterações a fazer para melhorar o jogo.

Relativamente à lógica do jogo, a primeira questão observada foi o grau de dificuldade que este apresentava. Isto devia-se a um ritmo demasiadamente acelerado, existindo uma aproximação demasiado rápida dos obstáculos. Este problema foi corrigido através da diminuição da velocidade dessa aproximação. Outra questão era a distância mínima entre os obstáculos. Por vezes ocorriam situações onde estes eram colocados muito próximos uns dos outros tornando praticamente impossível evitar a colisão. Isto também foi corrigido através da alteração do algoritmo de colocação aleatória dos obstáculos na pista, tendo este passado a incluir uma distância mínima entre os obstáculos.

Relativamente ao sistema de reconhecimento de atividades, houve na generalidade uma receção bastante positiva, com a grande maioria das atividades efetuadas a serem reconhecidas e executadas devidamente. Apesar disto, foi observado que o sistema por vezes falhava no reconhecimento da atividade de mover para o lado em alguns participantes. Como a lógica do jogo ao receber ordem para executar uma atividade não responde a mais nenhuma até que a anterior termine, criou-se uma versão em que eram mostradas as atividades que estavam a ser reconhecidas. Após análise, observou-se que alguns participantes tinham um movimento para o lado que se caracterizava por ter primeiro uma inclinação ligeira para o lado inverso para onde se estavam a deslocar. Por exemplo, ao moverem-se para o lado direto, faziam primeiro uma inclinação do corpo

para a esquerda e só depois se efetivava a movimentação para a direita. Algumas vezes o classificador do sistema detetava essa movimentação inversa e logo a seguir a outra, no entanto, esta última já não se refletia na personagem. Apesar disto, nem sempre esta sequência de classificações era a correta, uma vez que a atividade de movimentação para o lado é dada a variações simétricas como explicado no estudo inicial do sistema de reconhecimento de atividades. Assim, para tentar corrigir estas variações e minimizar erros optou-se por implementar um mecanismo de correção do movimento lateral baseado nas probabilidades das classificações. Ao reconhecer um movimento lateral, por exemplo para a esquerda, é guardada a probabilidade de classificação, sendo de seguida o movimento executado pela personagem. Se na classificação seguinte é reconhecido o movimento para a direita, a sua probabilidade é comparada com a do movimento anterior. No caso de ser superior e a personagem ainda estiver a executar o movimento anterior, é dito à personagem para corrigir o seu movimento, movendo-se então para a direita.

Após a implementação das sugestões anteriormente apresentadas, o jogo foi novamente testado. No geral, a interação com base em reconhecimento de atividades foi descrita como uma experiência bastante positiva. Os movimentos foram descritos como realistas e a resposta do sistema como rápida. Outro aspeto notório foi o aumento da imersão provocada pela interação com o jogo. Em vários testes verificou-se que, quando se fazia um agachamento, algumas pessoas tinham tendência para ficar numa posição baixa até que o obstáculo elevado passasse. Isto acontecia mesmo sabendo que o movimento correto é fazer um agachamento completo voltando para a posição base rapidamente. No entanto as pessoas mantinham-se em baixo como se um objeto real ainda estivesse a passar por cima delas. Outra característica que revelou eficácia foi o sistema antifraude implementado no treino. Nos testes, as pessoas tentaram por diversas vezes enganar o sistema de reconhecimento de atividades e, embora algumas vezes tenham conseguido, não obtiveram sucesso de uma forma consistente. Isto resulta assim num claro incentivo à execução correta das atividades físicas pretendidas, uma vez que o jogador sente que tem mais hipóteses de obter bons resultados caso execute as atividades corretamente do que tentando ludibriar o sistema.

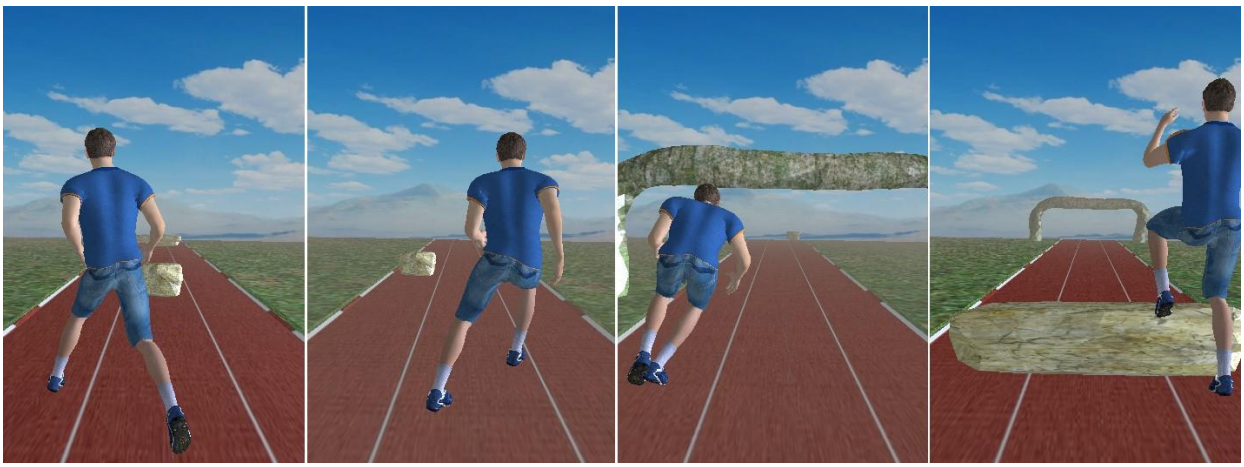


Figura 17 - Resultado da execução das 4 atividades físicas para desvio dos obstáculos: mover para o lado esquerdo, mover para o lado direito, agachar e saltar.

6 Conclusões

Neste capítulo é apresentada a conclusão do trabalho elaborado. Inicialmente é feita uma revisão geral do tema. Em seguida, são analisados os objetivos propostos e discutido até que ponto foram atingidos. Posteriormente, são apresentadas as contribuições deste trabalho. Para terminar, são também apresentadas as suas limitações.

6.1 Revisão geral do tema

Combinar jogos de vídeo para dispositivos móveis com movimento corporal é uma forma de motivar e promover a prática de atividade física na população, principalmente a mais jovem. Para além disso, constitui uma forma interessante e diferente de interagir com os dispositivos, permitindo uma experiencia mais imersiva. Apesar de hoje em dia os *smartphones* incorporarem um conjunto interessante de sensores, poucos estudos têm sido conduzidos para tentar explorar as suas potencialidades no que diz respeito à implementação de jogos com recurso a sistemas de reconhecimento de atividades.

Neste trabalho foi abordado um estudo para determinar um sistema de reconhecimento de atividades aplicável a um jogo de *endless running* para *smartphones Android*. Este sistema tem como objetivo perceber se o jogador está a executar uma das 5 atividades permitidas pelo jogo: estar parado, mover para o lado esquerdo, mover para o lado direito, agachar ou saltar. Dadas as suas características, este tipo de jogo torna-se ideal como prova de conceito. Tanto as suas atividades simples e naturais, facilmente reconhecidas pelo jogador, como o posicionamento da camara no mundo tridimensional, ajudam a criar uma experiência imersiva. Desta forma, pretende-se, com este tema, perceber a viabilidade de criar uma aplicação desta natureza e contribuir para uma nova forma de interagir com os dispositivos.

6.2 Revisão dos objetivos e trabalho efetuado

Listam-se agora os objetivos inicialmente delineados e analisa-se o trabalho desenvolvido para alcançar cada um deles.

O primeiro objetivo proposto para este trabalho é referente ao estudo de um sistema de reconhecimento de atividades capaz de prever em tempo real, através de apenas um *smartphone*, um conjunto de 5 atividades anteriormente apresentadas. No estudo efetuado, foi dada bastante enfase à fase de recolha de dados, tendo sido conduzido um treino meticuloso para treinar as

atividades rápidas e não continuas introduzidas. Cada participante teve de executar estas atividades com recurso à funcionalidade de paragem automática fornecida pela aplicação criada para a recolha. Este modo permite recolher uma janela de 64 valores vindos do acelerómetro e giroscópio do *smartphone*, representando um movimento de 1.28 segundos, sem sobreposição, efetuado após o utilizador ter carregado no botão "start". A recolha é automaticamente interrompida após esse período de tempo. Este treino permitiu, assim, recolher um conjunto de amostras limpas para cada atividade curta realizada. Foi também introduzido no treino da atividade de ficar parado um conjunto de recolhas que constituem uma forma de tentar tornar o sistema mais robusto contra tentativas de o iludir. Este sistema antifraude tem como finalidade incentivar o jogador a executar as atividades físicas corretamente, tendo sido bem sucedido nesse quesito, pelo menos nos testes efetuados. Outra característica importante desde estudo esteve na presença de duas atividades muito similares, o movimento lateral para a esquerda e o movimento lateral para a direita. Para evitar a introdução de confusão adicional no sistema, devido à similaridade do movimento presente nessas duas atividades, optou-se por distribuir as atividades por dois sistemas de classificação distintos. O sistema principal ficou encarregue de classificar as quatro atividades principais (parado, movimento lateral, agachar, saltar) e o sistema secundário ficou encarregue de distinguir a direção (esquerda ou direita) do movimento lateral, quando este é classificado pelo primeiro sistema. Foi depois utilizado um conjunto diferenciado de métodos de extração de características em cada um dos sistemas e testados vários algoritmos de seleção e classificação. Os resultados obtidos foram muito positivos, com o melhor sistema a atingir uma exatidão que superou os 99% através de *cross-validation*.

O segundo objetivo deste trabalho era criar uma biblioteca que permitisse facilitar a criação e utilização de sistemas de reconhecimento de atividades como o anteriormente estudado. Nesta fase, foi dado relevo aos requisitos requeridos pelo sistema específico do estudo, no entanto tentou-se ter uma abordagem genérico e o mais abrangente possível para contemplar o processo típico de reconhecimento de atividades através de dispositivos móveis. A versão atual da biblioteca permite criar, treinar e testar através de uma interface simples o sistema pretendido. Este objetivo foi assim conseguido apesar de a biblioteca representar um trabalho que ainda pode ser estendido, como veremos no subcapítulo 6.5.

O terceiro objetivo deste trabalho é referente à criação de um jogo de *endless running* proposto como prova de conceito. Com este jogo pretendeu-se validar todo o conceito introduzindo, demonstrando que é possível criar um jogo com uma interação inovadora, mais imersiva e que promova o exercício físico ao mesmo tempo. Após a implementação do jogo e de ter sido testado por alguns participantes, pode concluir-se que este objetivo foi devidamente atingido.

6.3 Principais contribuições

O trabalho desenvolvido tem como principais contribuições:

- Um sistema de reconhecimento de atividades robusto estudado e validado para prever em tempo real cinco atividades típicas de um jogo, sendo elas maioritariamente curtas e não contínuas. O estudo deste sistema inclui ainda várias sugestões de como recolher este tipo de atividades e como treinar um sistema difícil de ludibriar.
- Uma biblioteca *open source* para facilitar a criação e utilização de sistema de reconhecimento de atividades através dos sensores de dispositivos móveis.
- Uma prova de conceito que demonstra claramente as possibilidades deste estudo e aplica formas de corrigir a previsão do movimento em casos de variabilidade e incerteza como o movimento lateral estudado neste trabalho.

Para além destas contribuições, é de referir que este trabalho deu ainda origem a um artigo cientifico, recentemente submetido à revista *Pervasive and Mobile Computing*, da qual está à espera de aprovação.

6.4 Limitações

Pretende-se agora identificar e apresentar algumas das limitações presentes no trabalho desenvolvido.

Relativamente ao estudo efetuado, o sistema de reconhecimento de atividades apresentado inclui apenas 5 atividades simples. Estas atividades são as ideais para o tipo de jogo pretendido mas podem não servir para outros. Assim o sistema estudado deve ser adaptado consoante as necessidades específicas. Uma possibilidade é a utilização da biblioteca desenvolvida que possibilita fazer essa adaptação facilmente. Para além disto, o treino do sistema esteve limitado à participação de 6 voluntários, todos eles destros. Outra questão importante foi a utilização do mesmo dispositivo móvel por parte de todos os participantes no treino. Desta forma não foram incluídas no treino do sistema variabilidades que poderiam ser introduzidas caso diferentes dispositivos fossem utilizados.

Relativamente à biblioteca desenvolvida, esta focou-se nas necessidades específicas do trabalho proposto. Apesar de se ter feito um esforço para criar uma abordagem genérica e abrangente para criação de um sistema de reconhecimento de atividades, a atenção foi maioritariamente dada às funcionalidades requeridas pelo sistema estudado.

6.5 Direções para trabalho futuro

Pretende-se agora apresentar algumas das questões que poderão ser abordadas futuramente, no seguimento deste trabalho.

Como anteriormente referido nas limitações deste trabalho, o número de participantes e dispositivos testados é bastante limitado. Será por isso necessário testar a capacidade do sistema de reconhecimento de atividades com um conjunto mais vasto de participantes com diferentes fisionomias e estilos de locomoção. Para além de um conjunto diverso de pessoas, convém analisar se existem variações significativas relativamente a diferentes modelos de dispositivos, dada a grande fragmentação existente hoje em dia no mercado de *smartphones*. Outro aspeto a verificar é se a utilização da mão esquerda, no caso de pessoas esquerdinas, introduz incerteza no sistema e como abordar esta incerteza. Por exemplo, verificar se é melhor criar dois sistemas, um para mão esquerda e outro para mão direita, ou se basta uma fase de recolha e treino com um conjunto de pessoas que inclua esquerdinos.

Outra questão a estudar poderá ser a integração de mais tipos de atividades, diferentes das que foram aqui estudadas, tentando perceber até que ponto é possível introduzir formas de interação mais complexas neste tipo de sistemas. Também é necessário analisar o sistema antifraude apresentado e perceber se a sua eficiência se mantém ao introduzir essas novas atividades.

Relativamente à biblioteca desenvolvida, há ainda uma grande margem para a sua extensão. Entre o conjunto vasto de funcionalidades que poderão ser adicionadas, destaca-se, por exemplo: a serialização completa do sistema, permitindo gravar um sistema completamente treinado num computador e posteriormente carrega-lo numa aplicação, pronto a ser utilizado; adição de métodos para avaliação do sistema, como por exemplo *cross-validation*; adição de algoritmos próprios de classificação e seleção, bem como mais algoritmos de extração, possibilitando que, no futuro, a biblioteca não tenha de depender de bibliotecas de classificação externas. Também seria interessante criar uma interface gráfica para criar facilmente os sistemas de reconhecimento de atividades.

Referências

[1] Y. Oh and S. P. Yang, "Defining Exergames & Exergaming," in *Meaningful Play 2010 Conference Paper*, Michigan State University, East Lansing, Michigan, 2010.

[2] M. Pasch, N. Bianchi-Berthouze, B. van Dijk and A. Nijholt, "Movement-based sports video games: Investigating motivation and gaming experience," *Entertainment Computing*, vol. 1, no. 2, pp. 49 - 61, 2009.

[3] R. J. V. d. M. a. T. F. V. d. Medeiros, "Procedural Level Balancing in Runner Games," in *2014 Brazilian Symposium on Computer Games and Digital Entertainment*, 2014.

[4] T. Dutz, S. Hardy, M. Knöll, S. Göbel and R. Steinmetz, "User Interfaces of Mobile Exergames," in *Human-Computer Interaction. Applications and Services*, vol. 8512, M. Kurosu, Ed., Springer International Publishing, 2014, pp. 244-255.

[5] "Wii.com," Nintendo Co., Ltd., [Online]. Available: http://wii.com. [Accessed 30 11 2016].

[6] "Xbox.com," Microsoft Corporation, [Online]. Available: http://www.xbox.com. [Accessed 30 11 2016].

[7] "PlayStation® Official Site," Sony Corporation, [Online]. Available: http://www.playstation.com. [Accessed 30 11 2016].

[8] "IR Information: Sales Data - Top Selling Software Sales Units - Wii Software," Nintendo Co., Ltd., [Online]. Available: https://www.nintendo.co.jp/ir/en/sales/software/wii.html. [Accessed 30 11 2016].

[9] F. Caron, “Of gyroscopes and gaming: the tech behind the Wii MotionPlus,” arstechnica, 26 08 2008. [Online]. Available: http://arstechnica.com/gaming/2008/08/wii-motion-sensor/. [Accessed 09 12 2015].

[10] W. Greenwald, “Nintendo Wii Remote Plus Review,” PCMag UK, [Online]. Available: http://uk.pcmag.com/controllers-accessories-products/6251/review/nintendo-wii-remote-plus. [Accessed 09 12 2015].

[11] "Geocaching," [Online]. Available: https://www.geocaching.com. [Accessed 30 11 2016].

[12] "Pokémon GO | Pokémon Video Games," [Online]. Available: http://www.pokemon.com/us/pokemon-video-games/pokemon-go/. [Accessed 30 11 2016].

[13] "Ingress," [Online]. Available: https://www.ingress.com.

[14] P. Buddharaju and N. Pamidi, "Mobile Exergames - Burn Calories While Playing Games on a Smartphone," in *Computer Vision and Pattern Recognition Workshops (CVPRW), 2013 IEEE Conference on*, 2013.

[15] A. Karime, B. Hafidh, W. Gueaieb and A. E. Saddik, "A modular mobile exergaming system with an adaptive behavior," in *Medical Measurements and Applications (MeMeA), 2015 IEEE International Symposium on*, 2015.

[16] B. Hafidh, H. Al Osman and A. El Saddik, "SmartInsole: A foot-based activity and gait measurement device," *Multimedia and Expo Workshops*, pp. 1-4, 2013.

[17] X. Su, H. Tong and P. Ji, "Activity recognition with smartphone sensors," *Tsinghua Science and Technology*, vol. 19, no. 3, pp. 235-249, 2014.

[18] M. A. Ayu, S. A. Ismail, A. F. A. Matin and T. Mantoro, "A Comparison Study of Classifier Algorithms for Mobile-phone's Accelerometer Based Activity Recognition," *Procedia Engineering* , vol. 41, pp. 224-229, 2012.

[19] M. Ayu, T. Mantoro, A. Matin and S. Basamh, "Recognizing user activity based on accelerometer data from a mobile phone," in *Computers Informatics (ISCI), 2011 IEEE Symposium on*, 2011.

[20] S. Preece, J. Goulermas, L. Kenney and D. Howard, "A Comparison of Feature Extraction Methods for the Classification of Dynamic Activities From Accelerometer Data," *Biomedical Engineering, IEEE Transactions on*, vol. 56, no. 3, pp. 871-879, March 2009.

[21] J. R. Kwapisz, G. M. Weiss and S. A. Moore, "Activity Recognition Using Cell Phone Accelerometers," *SIGKDD Explor. Newsl.*, vol. 12, no. 2, pp. 74-82, 2011.

[22] M. Arif, M. Bilal, A. Kattan and S. Ahamed, "Better Physical Activity Classification using Smartphone Acceleration Sensor," *Journal of Medical Systems*, vol. 38, no. 9, 2014.

[23] D. W. Aha, D. Kibler and M. K. Albert, "Instance-Based Learning Algorithms," *Mach. Learn.*, vol. 6, no. 1, pp. 37-66, 1991.

[24] A. Bayat, M. Pomplun and D. A. Tran, "A Study on Human Activity Recognition Using Accelerometer Data from Smartphones," *Procedia Computer Science* , vol. 34, pp. 450-457, 2014.

[25] M. Shoaib, S. Bosch, O. D. Incel, H. Scholten and P. J. Havinga, "Fusion of smartphone motion sensors for physical activity recognition," *Sensors*, vol. 14, no. 6, pp. 10146-10176, 2014.

[26] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot and E. Duchesnay, "Scikit-learn: Machine Learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825-2830, 2011.

[27] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann and I. H. Witten, "The WEKA Data Mining Software: An Update," *SIGKDD Explor. Newsl.*, vol. 11, no. 1, pp. 10-18, 2009.

[28] "ActivityRecognitionApi | Google APIs for Android | Google Developers," Google Inc., [Online]. Available: https://developers.google.com/android/reference/com/google/android/gms/location/ActivityRecognitionApi. [Accessed 30 11 2016].

[29] M. Kumar, "Google's Activity Recognition API is awesome but where are the apps?," [Online]. Available: http://geoawesomeness.com/googles-activity-recognition-api-is-awesome-but-where-are-the-apps/. [Accessed 30 11 2016].

[30] "CMMotionActivity - Core Motion | Apple Developer Documentation," Apple Inc., [Online]. Available: https://developer.apple.com/reference/coremotion/cmmotionactivity. [Accessed 30 11 2016].

[31] "Activity Monitor API," Microsoft Corporation, [Online]. Available: https://msdn.microsoft.com/en-us/library/dn932084.aspx. [Accessed 30 11 2016].

[32] "Kaggle: Your Home for Data Science," [Online]. Available: https://www.kaggle.com. [Accessed 30 11 2016].

[33] D. K. Wind, "The Official Blog of Kaggle.com: Learning from the best," 08 01 2014. [Online]. Available: http://blog.kaggle.com/2014/08/01/learning-from-the-best/.

[34] T. Dettmers, "Deep Learning in a Nutshell: Core Concepts," 3 11 2015. [Online]. Available: https://devblogs.nvidia.com/parallelforall/deep-learning-nutshell-core-concepts/. [Accessed 30 11 2016].

[35] "Sensors Overview | Android Developers," [Online]. Available: http://developer.android.com/guide/topics/sensors/sensors_overview.html. [Accessed 30 11 2016].

[36] J. W. T. James W. Cooley, "An Algorithm for the Machine Calculation of Complex Fourier Series," *Mathematics of Computation*, vol. 19, no. 90, pp. 297-301, 1965.

[37] N. Ravi, N. Dandekar, P. Mysore and M. L. Littman, "Activity Recognition from Accelerometer Data," in *Proceedings of the 17th Conference on Innovative Applications of Artificial Intelligence - Volume 3*, 2005.

[38] T. P. K. Ekstein, "Entropy And Entropy-based Features In Signal," in *Proceedings of PhD Workshop Systems & Control*, Balatonfured, 2004.

[39] L. Bao and S. Intille, "Activity Recognition from User-Annotated Acceleration Data," in *Pervasive Computing*, vol. 3001, A. Ferscha and F. Mattern, Eds., Springer Berlin Heidelberg, 2004, pp. 1-17.

[40] M. A. Hall, "Correlation-based Feature Subset Selection for Machine Learning," Hamilton, New Zealand, 1998.

[41] M. Gutlein, E. Frank, M. Hall and A. Karwath, "Large-scale attribute selection using wrappers," in *Computational Intelligence and Data Mining, 2009. CIDM '09. IEEE Symposium on*, 2009.

[42] D. E. Goldberg, Genetic Algorithms in Search, Optimization and Machine Learning, 1st ed., Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1989.

[43] L. Breiman, "Random Forests," *Mach. Learn.*, vol. 45, no. 1, pp. 5-32, 2001.

[44] J. C. Platt, "Advances in Kernel Methods," B. Schölkopf, C. J. C. Burges and A. J. Smola, Eds., Cambridge, MA, USA, MIT Press, 1999, pp. 185-208.

[45] G. H. John and P. Langley, "Estimating Continuous Distributions in Bayesian Classifiers," in *Proceedings of the Eleventh Conference on Uncertainty in Artificial Intelligence*, San Francisco, CA, USA, 1995.

[46] E. Frank and I. H. Witten, "Generating Accurate Rule Sets Without Global Optimization," in *Proceedings of the Fifteenth International Conference on Machine Learning*, San Francisco, CA, USA, 1998.

[47] "Commons Math: The Apache Commons Mathematics Library," [Online]. Available: https://commons.apache.org/proper/commons-math/. [Accessed 30 11 2016].

[48] "Commons CSV," [Online]. Available: https://commons.apache.org/proper/commons-csv/. [Accessed 30 11 2016].

[49] E. a. F. E. a. S. K. a. B. B. Freeman, Head First Design Patterns, O'Reilly Media, Incorporated, 2004.

[50] P. Villega, "Builder Pattern and Inheritance in Java," 22 2 2014. [Online]. Available: https://medium.com/@GumtreeDevTeam/builder-pattern-and-inheritance-in-java-25ccd2d70c9d#.gds5mmtaj. [Accessed 30 11 2016].

[51] "Using inheritance with fluent interfaces: get this," 9 6 2010. [Online]. Available: http://egalluzzo.blogspot.pt/2010/06/using-inheritance-with-fluent.html. [Accessed 30 11 2016].

[52] "MakeHuman | Open source tool for making 3d characters," [Online]. Available: http://www.makehuman.org/index.php. [Accessed 30 11 2016].

[53] "blender.org - Home of the Blender project - Free and Open 3D Creation Software," [Online]. Available: https://www.blender.org/.

[54] "Carnegie Mellon University - CMU Graphics Lab - motion capture library," [Online]. Available: http://mocap.cs.cmu.edu/. [Accessed 30 11 2016].

[55] "libGDX," [Online]. Available: http://libgdx.badlogicgames.com/. [Accessed 30 11 2016].

[56] "GitHub - ASneakyFox/libgdx-fbxconv-gui: Gui Wrapper for fbx-conv and 3D model previewer.," [Online]. Available: https://github.com/ASneakyFox/libgdx-fbxconv-gui. [Accessed 30 11 2016].

[57] "GitHub - libgdx/fbx-conv: Command line utility using the FBX SDK to convert FBX/Collada/Obj files to a custom text/binary format for static, keyframed and skinned meshes.," [Online]. Available: https://github.com/libgdx/fbx-conv. [Accessed 30 11 2016].

Anexo A – Proposta de estágio

PROPOSTA DE ESTÁGIO

MESTRADO em INFORMÁTICA E SISTEMAS
especialização em Desenvolvimento de Software

Ano Letivo de 2015/2016

TEMA

Reconhecimento de Atividades em Jogos para Dispositivos Móveis

SUMÁRIO

O reconhecimento de atividades é uma área de investigação e desenvolvimento em Computação Ubíqua que permite inferir que atividades estão a ser realizadas por uma pessoa, em tempo real, através de um ou vários sensores. Nos últimos anos têm-se verificado um crescente número de estudos nesta área, principalmente devido à popularização de dispositivos móveis que incluem um conjunto de sensores que possibilitam estes estudos e facilitam a aproximação da tecnologia às pessoas. Apesar disto, existe uma área que poderá beneficiar deste tipo de tecnologia e que não tem sido devidamente explorada: os jogos. Este trabalho pretende, assim, estudar e desenvolver formas de integrar o reconhecimento de atividades em jogos para dispositivos móveis, criando uma experiência de interação inovadora e imersiva, com recurso a apenas um *smartphone*, combatendo desta forma os hábitos sedentários que alguns destes utilizadores adquirem.

Palavras-chave: Computação Ubíqua, Reconhecimento de Atividades, Tempo Real, Realidade Virtual, Jogos.

- ## ÂMBITO

  À medida que a tecnologia avança e os dispositivos móveis se tornam cada vez mais poderosos e versáteis, a oportunidade para estudar estes dispositivos como forma de reconhecimento de atividades tem crescido imenso nos últimos anos. Sendo uma área relativamente recente, o estudo do reconhecimento de atividades tem sido impulsionado e tem suscitado cada vez mais interesse, principalmente devido à disseminação destes dispositivos móveis que permitem facilmente adquirir dados de um conjunto variado de sensores, como é o caso do acelerómetro.

  Apesar de esta ser uma área que tem sido alvo de intensivo estudo, até agora, pouco se investigou quanto à possibilidade da sua aplicação em jogos para dispositivos móveis. Tendo em conta que este tipo de aplicações representa uma enorme fatia do mercado torna-se interessante explorar essa possibilidade. Neste tipo de jogos, recorremos normalmente ao

ecrã tátil para interagir com o ambiente virtual. Em alguns casos, podemos usar o acelerómetro para controlar um objeto, movendo o telemóvel em diferentes direções. No entanto, este tipo de interação não é suficientemente imersiva. Estudar a hipótese de criar jogos em que uma personagem virtual responde aos movimentos do jogador abriria portas a um novo mundo de jogos interativos nos dispositivos móveis. Atualmente, existem várias tecnologias inovadoras na área dos jogos e da realidade virtual, como é o caso do *Wii remote*, *Kinect*, etc, no entanto estas tecnologias não são apropriadas para dispositivos móveis e são algo dispendiosas.

Com este tema, pretende-se introduzir tecnologias de reconhecimento de atividades em jogos móveis, com o objetivo de criar uma experiência de interação inovadora, imersiva e que promova a atividade física, com recurso a apenas um *smartphone*. Pretende-se assim, explorar formas de adquirir e processar dados vindos de um dispositivo móvel e através deles reconhecer comandos básicos como mover para o lado, saltar ou baixar e, em tempo-real, fazer uma personagem virtual do jogo responder a esses movimentos, replicando-os. Trata-se de produzir um jogo em que os movimentos do jogador resultam em ações no ambiente virtual que visualiza, fazendo-o sentir-se parte do jogo.

- **OBJECTIVOS**

O presente projeto pretende atingir os seguintes objetivos genéricos:

- Estudar métodos de extração de características dos dados vindos de sensores do dispositivo móvel, bem como os diferentes algoritmos de classificação existentes e bibliotecas que os implementem.
- Desenvolver código para criação de um modelo de classificação de atividades que seja genérico o suficiente, por forma a ser utilizado em outros jogos ou aplicações, sem necessidade de se estar a treinar novamente o modelo para processar o reconhecimento de atividades em cada novo projeto. Este modelo poderá ser utilizado como um módulo por outras aplicações (e.g. jogos) que desejem incorporar reconhecimento de atividades como forma de enriquecer a sua interação com o utilizador. Trata-se de criar um módulo que irá servir de intermediário entre uma aplicação e os recursos de reconhecimento de atividades, encapsulando estes. Pretende-se ainda, que este módulo seja disponibilizado como uma biblioteca para que outras pessoas possam utilizar e até contribuir.
- Desenvolvimento de um jogo para dispositivos móveis que implemente a tecnologia pretendida, como prova de conceito.

Estes objetivos intermédios vão permitir atingir o objetivo principal deste projeto, o estudo da utilização do reconhecimento de atividades como forma de criar uma interação inovadora e imersiva, que promova a prática do exercício físico em jogos móveis.

- **PROGRAMA DE TRABALHOS**

O Projeto consistirá nas seguintes atividades e respetivas tarefas:

- T1 – Estudo e caracterização das tecnologias envolvidas
- T2 – Desenvolvimento de uma biblioteca que vai permitir ao jogo implementar o reconhecimento de atividades
- T3 – Testes à biblioteca, com recolha efetiva dos dados e classificação
- T4 – Levantamento e análise de requisitos para o jogo a desenvolver. *Design* da arquitetura
- T5 – Desenvolvimento do jogo
- T6 – Testes ao jogo (Arquitetura/Funcionais/Usabilidade)
- T7 – Documentação (manuais, artigo e dissertação)

• CALENDARIZAÇÃO DAS TAREFAS

As Tarefas acima descritas, incluindo os testes de validação de cada módulo, serão executadas de acordo com a seguinte calendarização:

Tarefas		N		N+1		N+2		N+3		N+4		N+5		N+6		N+7		N+8	
T1																			
T2																			
T3																			
T4																			
T5																			
T6																			
T7																			
Metas	INI				M1						M2	M3					M4		M5

O plano de escalonamento dos trabalhos é apresentado em seguida:

INI		Início dos trabalhos
M1	(INI + 8 Semanas)	Tarefa T1 terminada
M2	(INI + 20 Semanas)	Tarefa T2 e T3 terminada
M3	(INI + 22 Semanas)	Tarefa T4 terminada
M4	(INI + 34 Semanas)	Tarefa T5 e T6 terminada
M5	(INI + 36 Semanas)	Tarefa T7 terminada

• RESULTADOS

Os resultados do estágio serão consubstanciados num conjunto de documentos a elaborar pelo estagiário de acordo com o seguinte plano:

M1:

Artigo com um conjunto de estudos realizados com tecnologias relacionadas com o objetivo do Projeto.

M2:

Relatório técnico de análise à biblioteca criada e testes efetuados

M3:

Relatório técnico com a análise de requisitos e especificação do jogo a desenvolver

M4:

Relatório técnico com a descrição das etapas de desenvolvimento do jogo e testes efetuados

M5:

Relatório final de estágio

- **LOCAL DE TRABALHO**

DEIS

- **METODOLOGIA**

Organização de um Dossier de Projeto e reuniões semanais.

- **ORIENTAÇÃO**

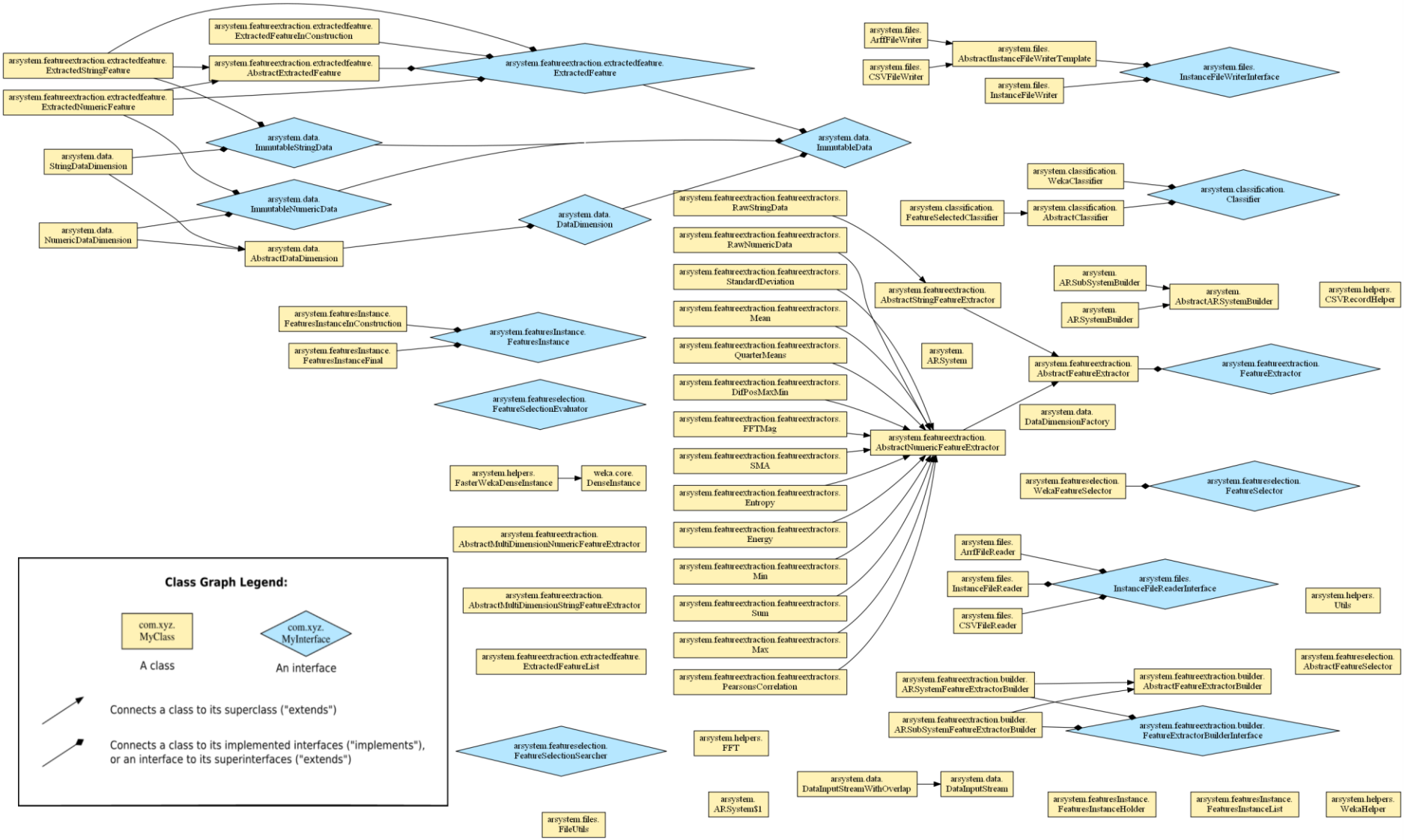
Ana Cristina Oliveira Alves (aalves@isec.pt)
Professora Adjunta

Orientando: Alexandre Almeida
Aluno do Mestrado de Informática e de Sistemas

- **CARACTERIZAÇÃO**
  - Data de início: Novembro de 2015

Data de fim: Julho de 2016

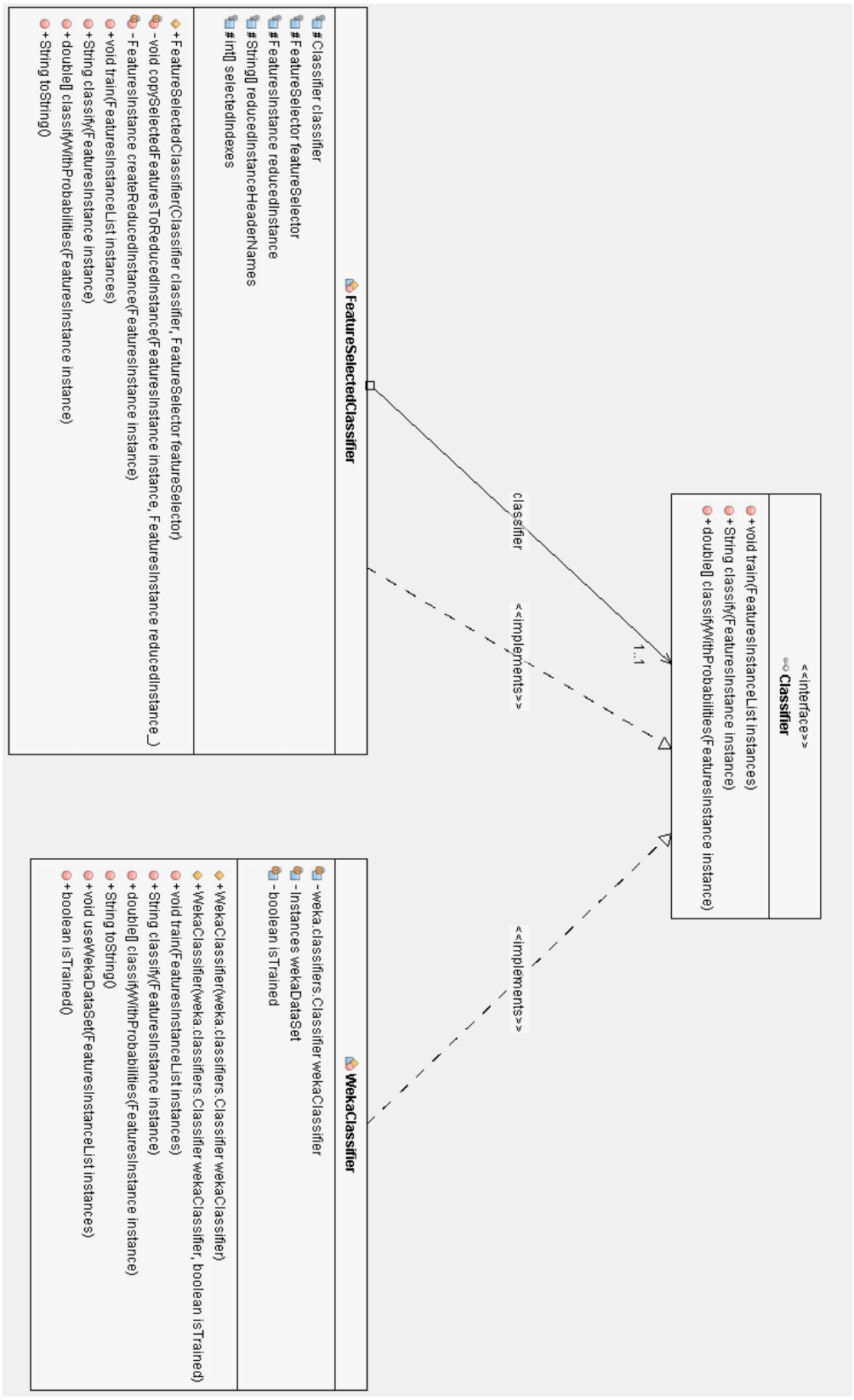
Anexo B – Grafo representativo da hierarquia de classes da biblioteca desenvolvida


Class Graph Legend:
com.xyz.MyClass
A class
com.xyz.MyInterface
An interface
Connects a class to its superclass ("extends")
Connects a class to its implemented interfaces ("implements"), or an interface to its superinterfaces ("extends")
arsystem.featureextraction.extractedfeature.ExtractedStringFeature
arsystem.featureextraction.extractedfeature.ExtractedNumericFeature
arsystem.featureextraction.extractedfeature.ExtractedFeatureInConstruction
arsystem.featureextraction.extractedfeature.AbstractExtractedFeature
arsystem.featureextraction.extractedfeature.ExtractedFeature
arsystem.data.StringDataDimension
arsystem.data.NumericDataDimension
arsystem.data.ImmutableStringData
arsystem.data.ImmutableNumericData
arsystem.data.AbstractDataDimension
arsystem.data.DataDimension
arsystem.data.ImmutableData
arsystem.featuresInstance.FeaturesInstanceInConstruction
arsystem.featuresInstance.FeaturesInstanceFinal
arsystem.featuresInstance.FeaturesInstance
arsystem.featureselection.FeatureSelectionEvaluator
arsystem.helpers.FasterWekaDenseInstance
weka.core.DenseInstance
arsystem.featureextraction.AbstractMultiDimensionNumericFeatureExtractor
arsystem.featureextraction.AbstractMultiDimensionStringFeatureExtractor
arsystem.featureextraction.extractedfeature.ExtractedFeatureList
arsystem.featureselection.FeatureSelectionSearcher
arsystem.files.FileUtils
arsystem.helpers.FFT
arsystem.AR.System$1
arsystem.data.DataInputStreamWithOverlap
arsystem.data.DataInputStream
arsystem.featureextraction.featureextractors.RawStringData
arsystem.featureextraction.featureextractors.RawNumericData
arsystem.featureextraction.featureextractors.StandardDeviation
arsystem.featureextraction.featureextractors.Mean
arsystem.featureextraction.featureextractors.QuarterMeans
arsystem.featureextraction.featureextractors.DiffPosMaxMin
arsystem.featureextraction.featureextractors.FFTMag
arsystem.featureextraction.featureextractors.SMA
arsystem.featureextraction.featureextractors.Entropy
arsystem.featureextraction.featureextractors.Energy
arsystem.featureextraction.featureextractors.Min
arsystem.featureextraction.featureextractors.Sum
arsystem.featureextraction.featureextractors.Max
arsystem.featureextraction.featureextractors.PearsonsCorrelation
arsystem.featureextraction.AbstractStringFeatureExtractor
arsystem.featureextraction.AbstractNumericFeatureExtractor
arsystem.AR.System
arsystem.featureextraction.AbstractFeatureExtractor
arsystem.featureextraction.FeatureExtractor
arsystem.data.DataDimensionFactory
arsystem.AR.SubSystemBuilder
arsystem.AR.SystemBuilder
arsystem.AbstractAR.SystemBuilder
arsystem.helpers.CSVRecordHelper
arsystem.classification.WekaClassifier
arsystem.classification.FeatureSelectedClassifier
arsystem.classification.AbstractClassifier
arsystem.classification.Classifier
arsystem.featureselection.WekaFeatureSelector
arsystem.featureselection.FeatureSelector
arsystem.files.ArffFileWriter
arsystem.files.CSVFileWriter
arsystem.files.AbstractInstanceFileWriterTemplate
arsystem.files.InstanceFileWriter
arsystem.files.InstanceFileWriterInterface
arsystem.files.ArffFileReader
arsystem.files.InstanceFileReader
arsystem.files.CSVFileReader
arsystem.files.InstanceFileReaderInterface
arsystem.helpers.Utils
arsystem.featureselection.AbstractFeatureSelector
arsystem.featureextraction.builder.AR.SystemFeatureExtractorBuilder
arsystem.featureextraction.builder.AR.SubSystemFeatureExtractorBuilder
arsystem.featureextraction.builder.AbstractFeatureExtractorBuilder
arsystem.featureextraction.builder.FeatureExtractorBuilderInterface
arsystem.featuresInstance.FeaturesInstanceHolder
arsystem.featuresInstance.FeaturesInstanceList
arsystem.helpers.WekaHelper

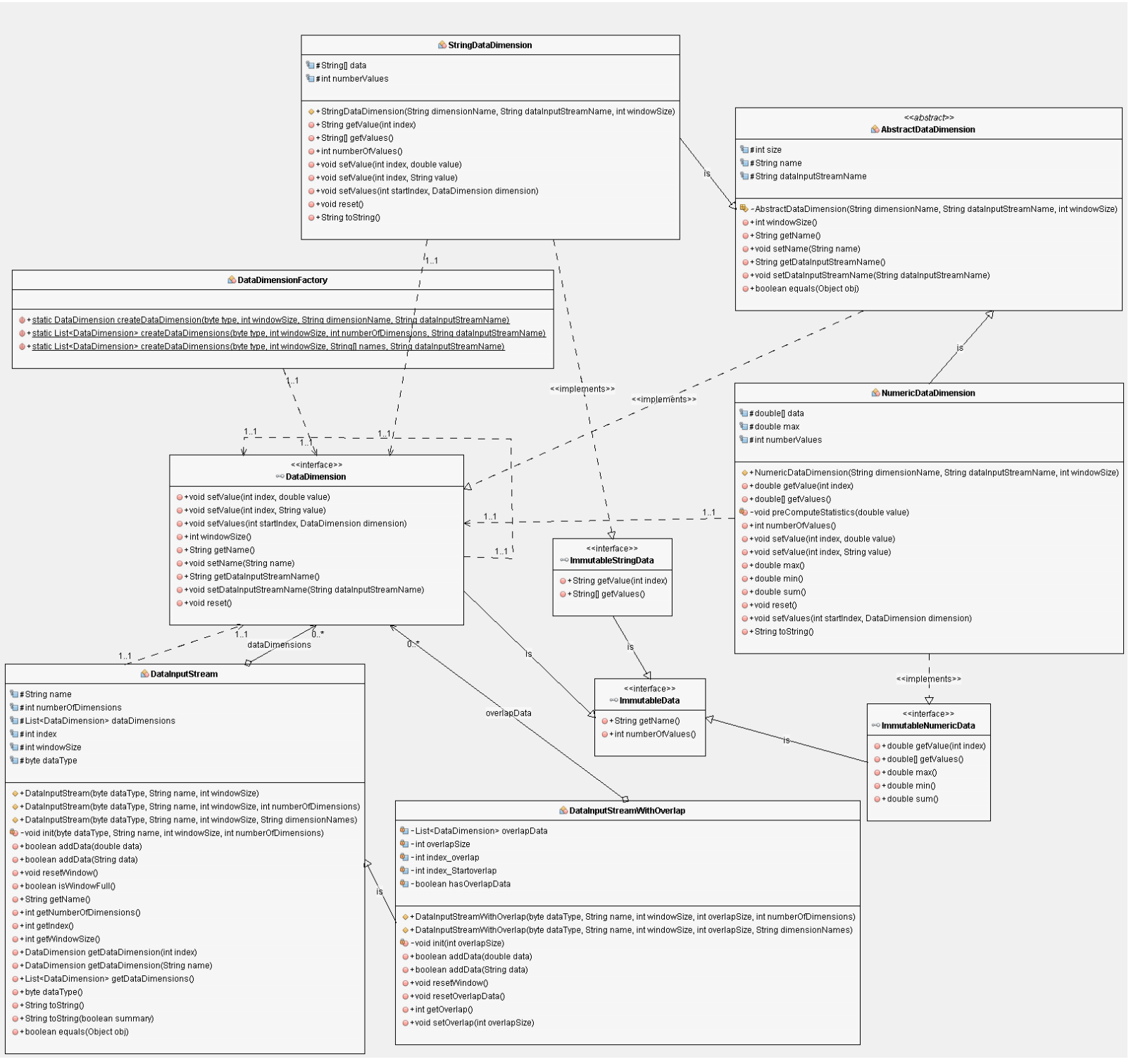
Anexo C – Diagramas de classes da biblioteca por pacote

Pacote arsystem

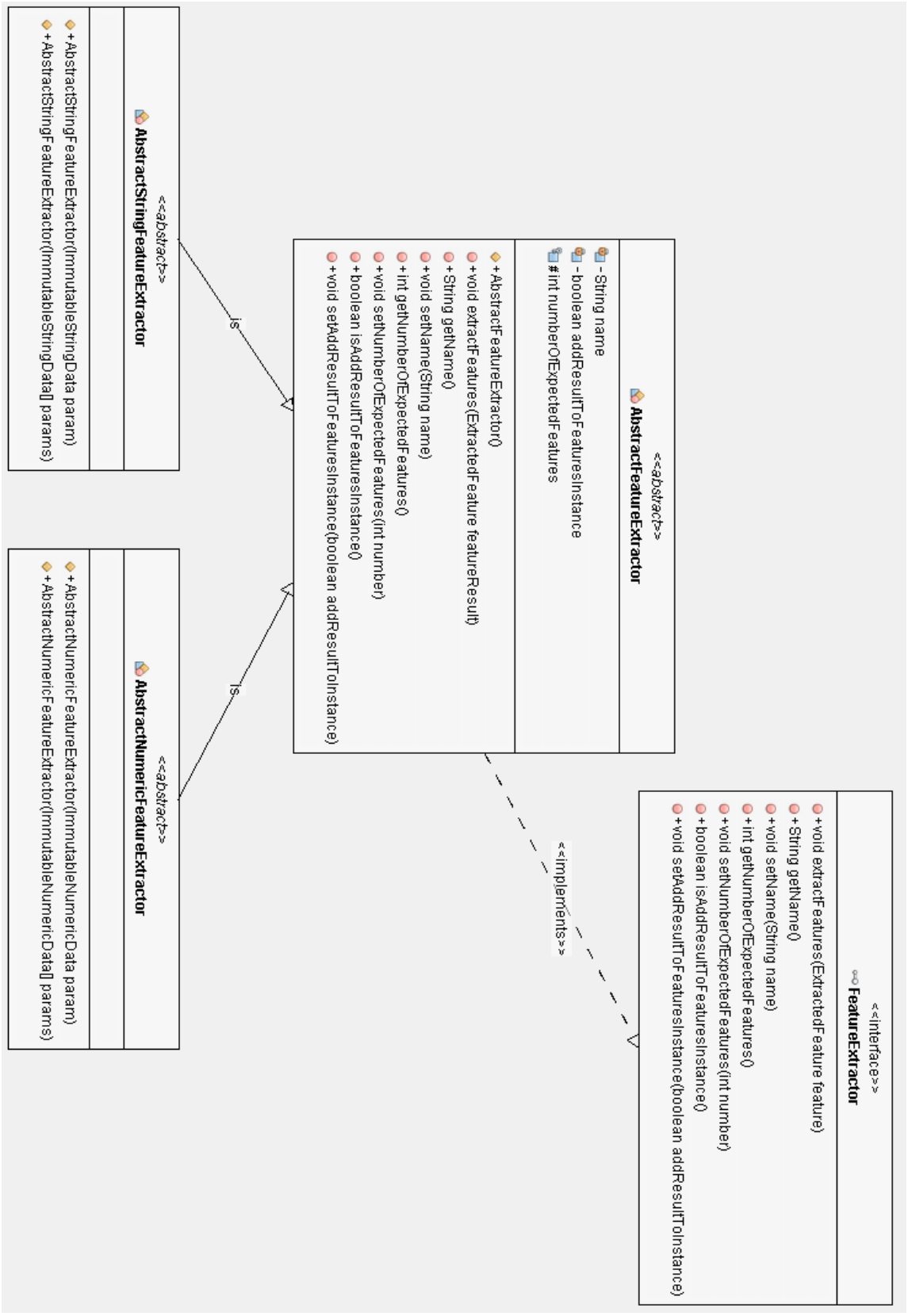
Pacote arsystem.classification



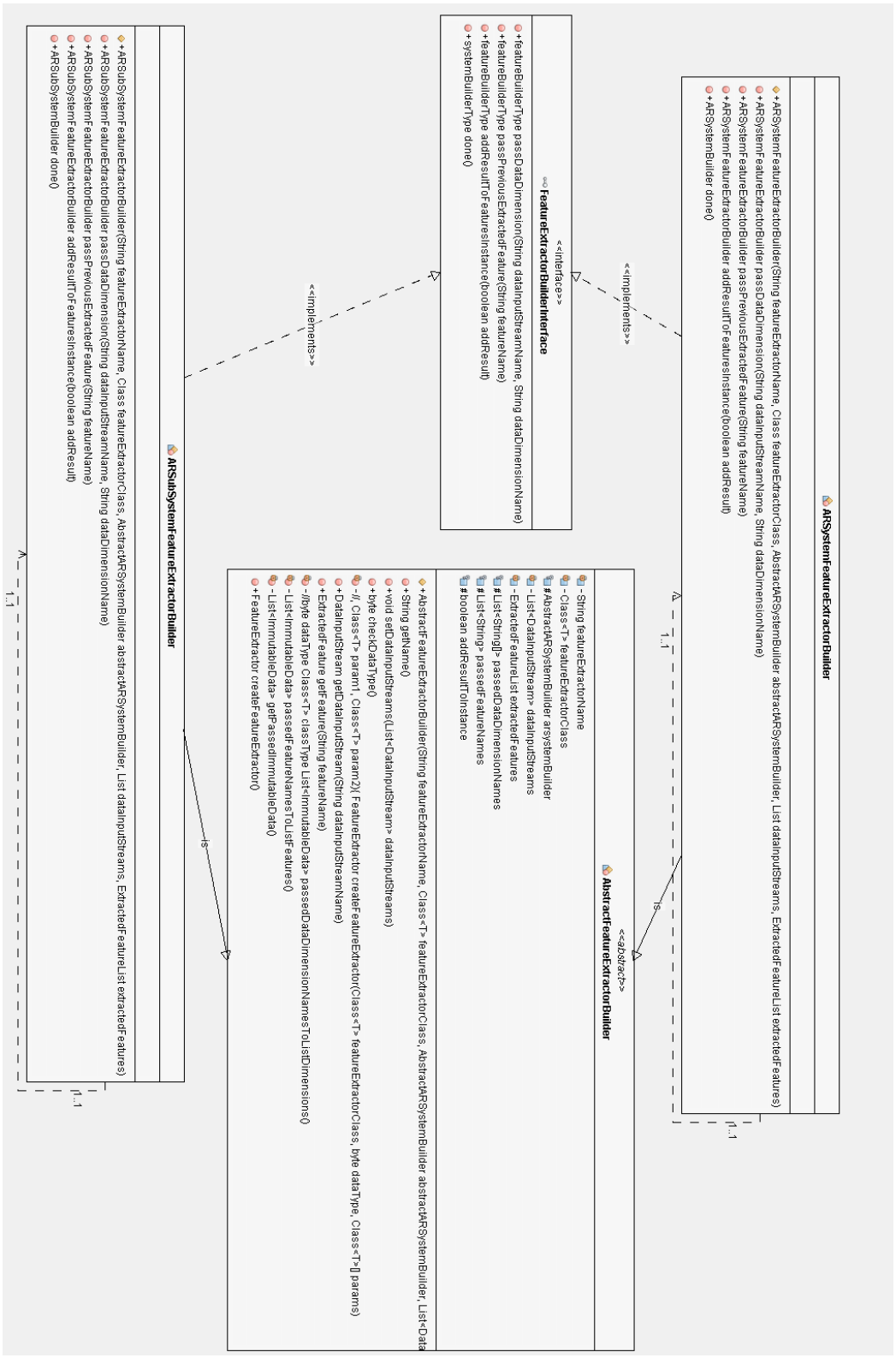
Pacote arsystem.data



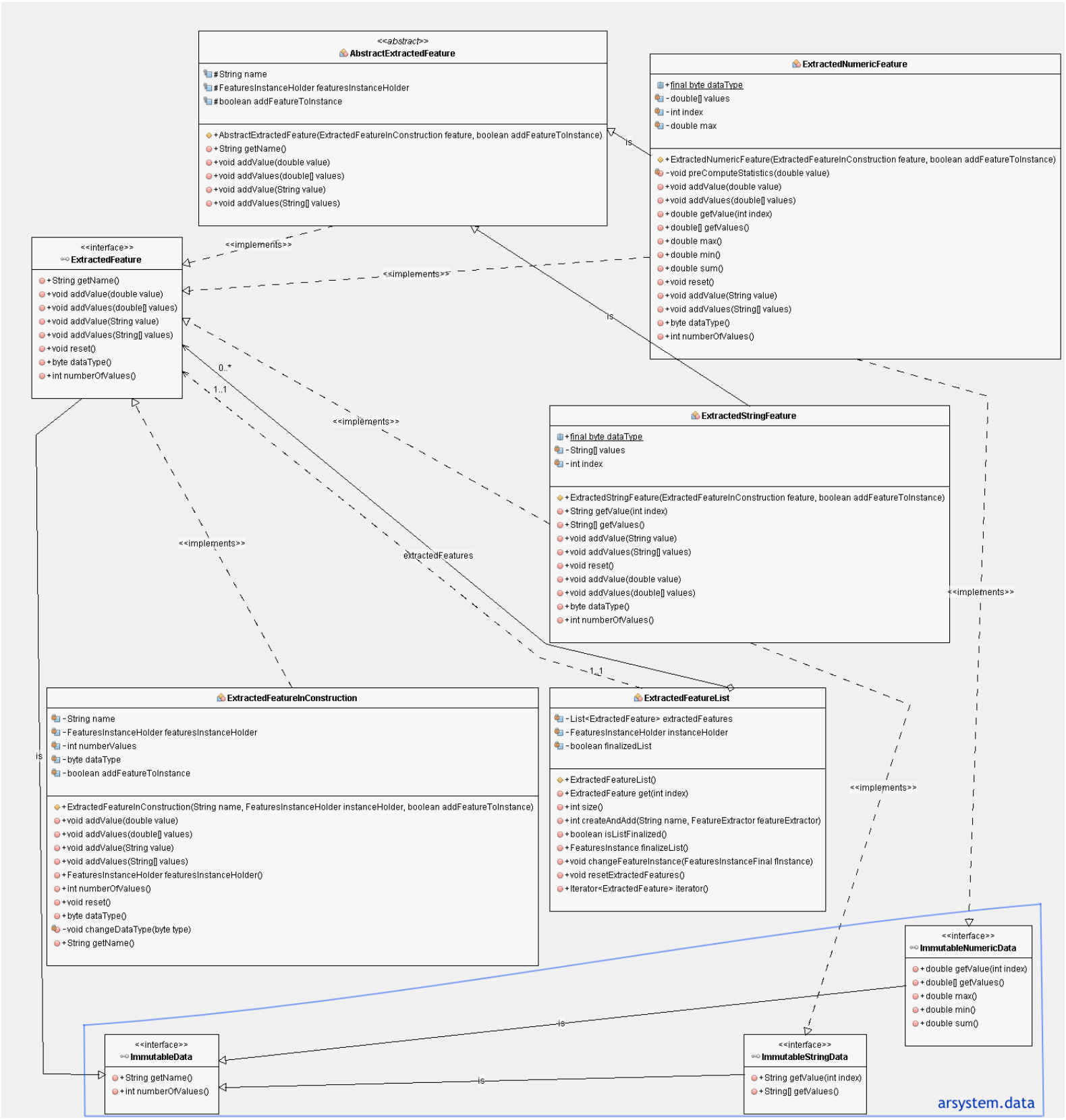
Pacote arsystem.featureextraction



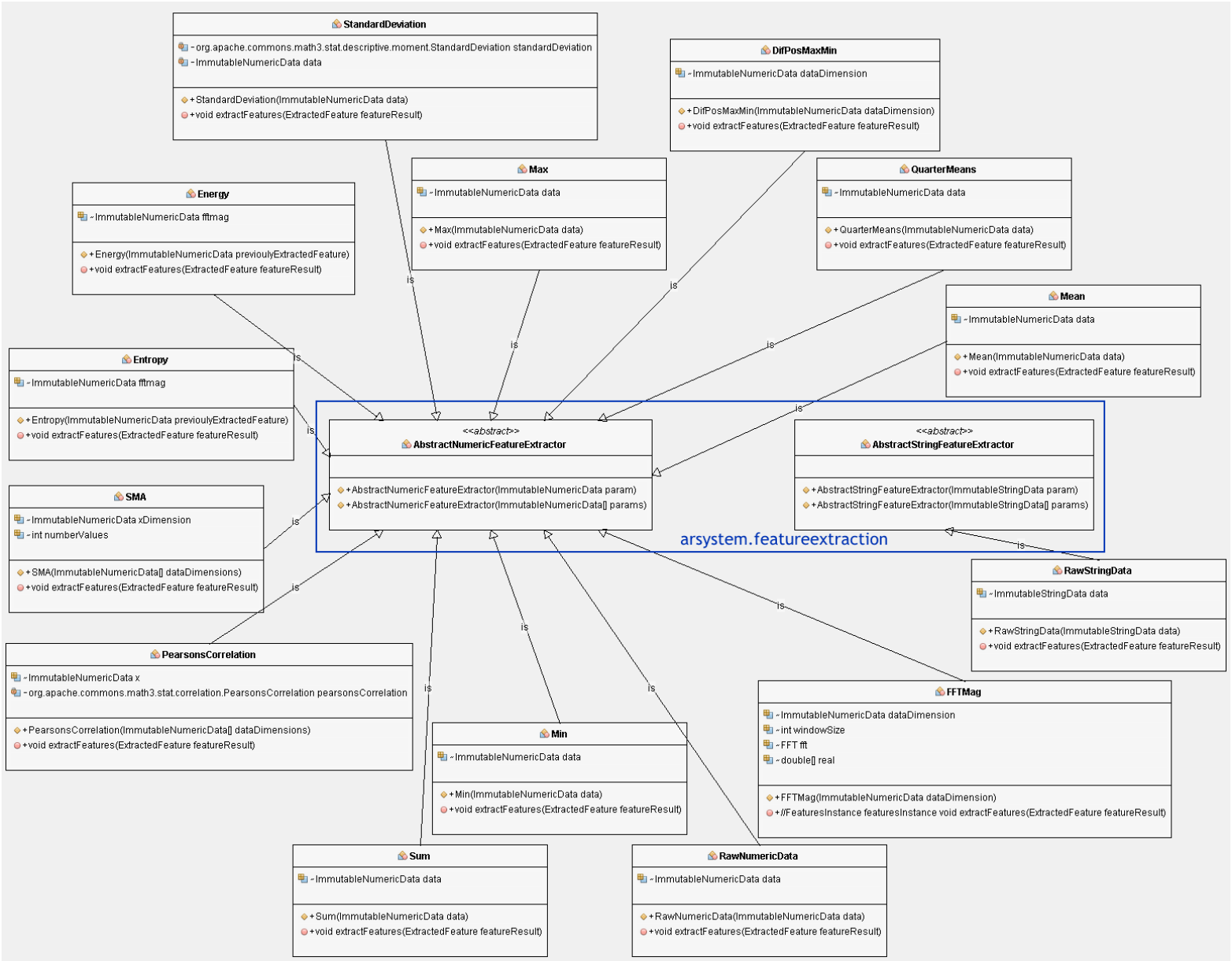
Pacote arsystem.featureextraction.builder



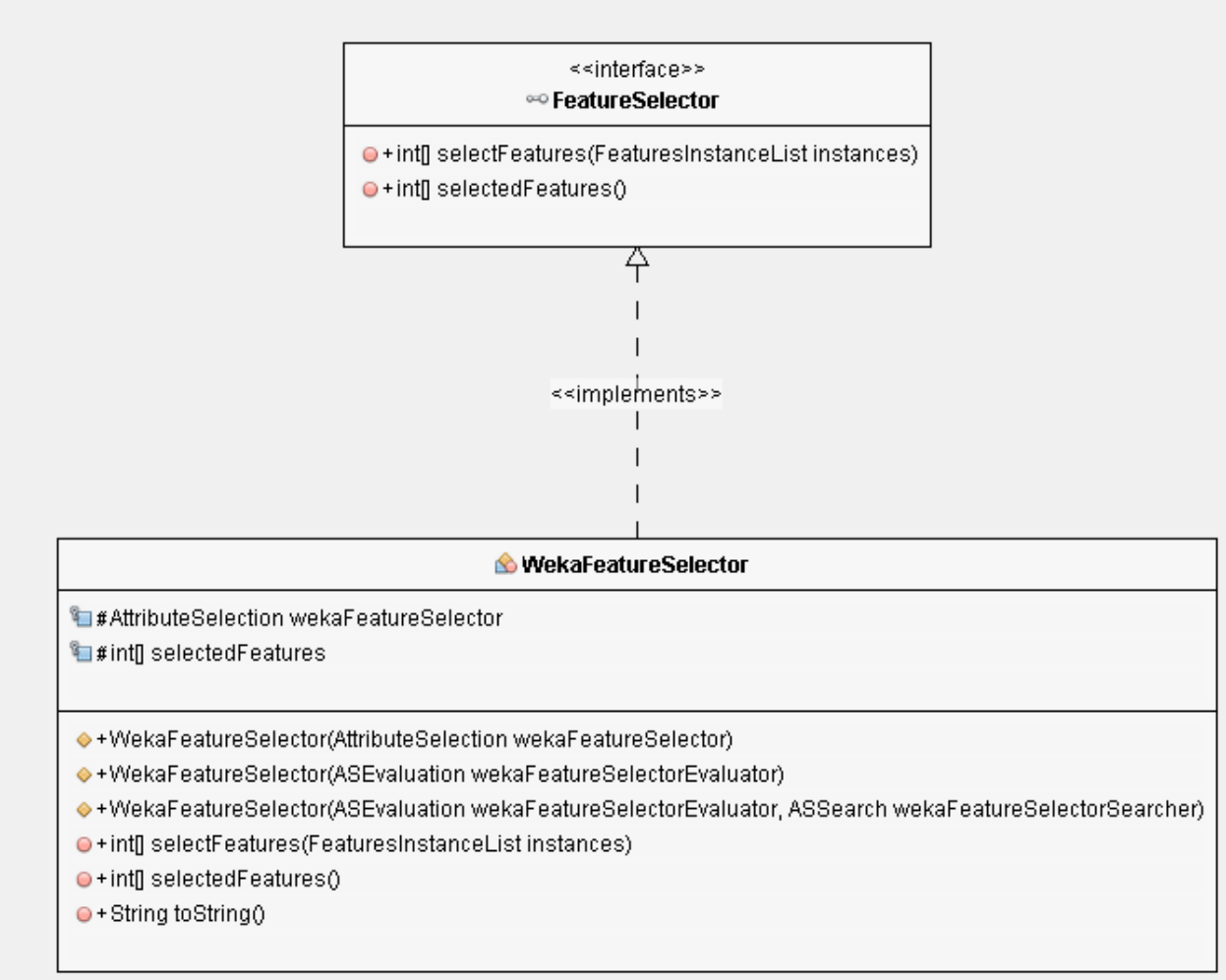
Pacote arsystem.featureextraction.extractedfeatures



Pacote arsystem.featureextraction.featureextractors

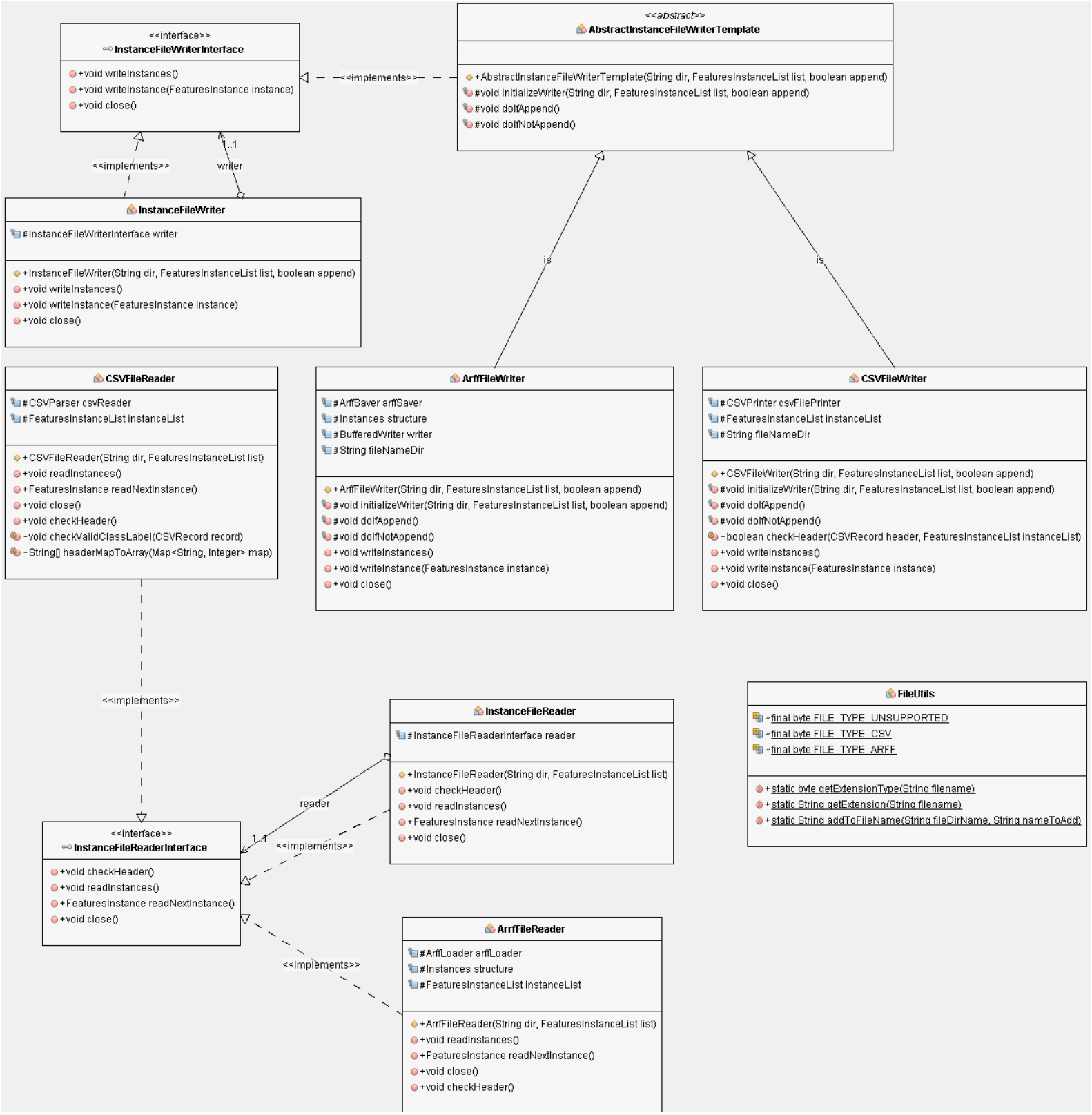


Pacote arsystem.featureextraction.featureselection

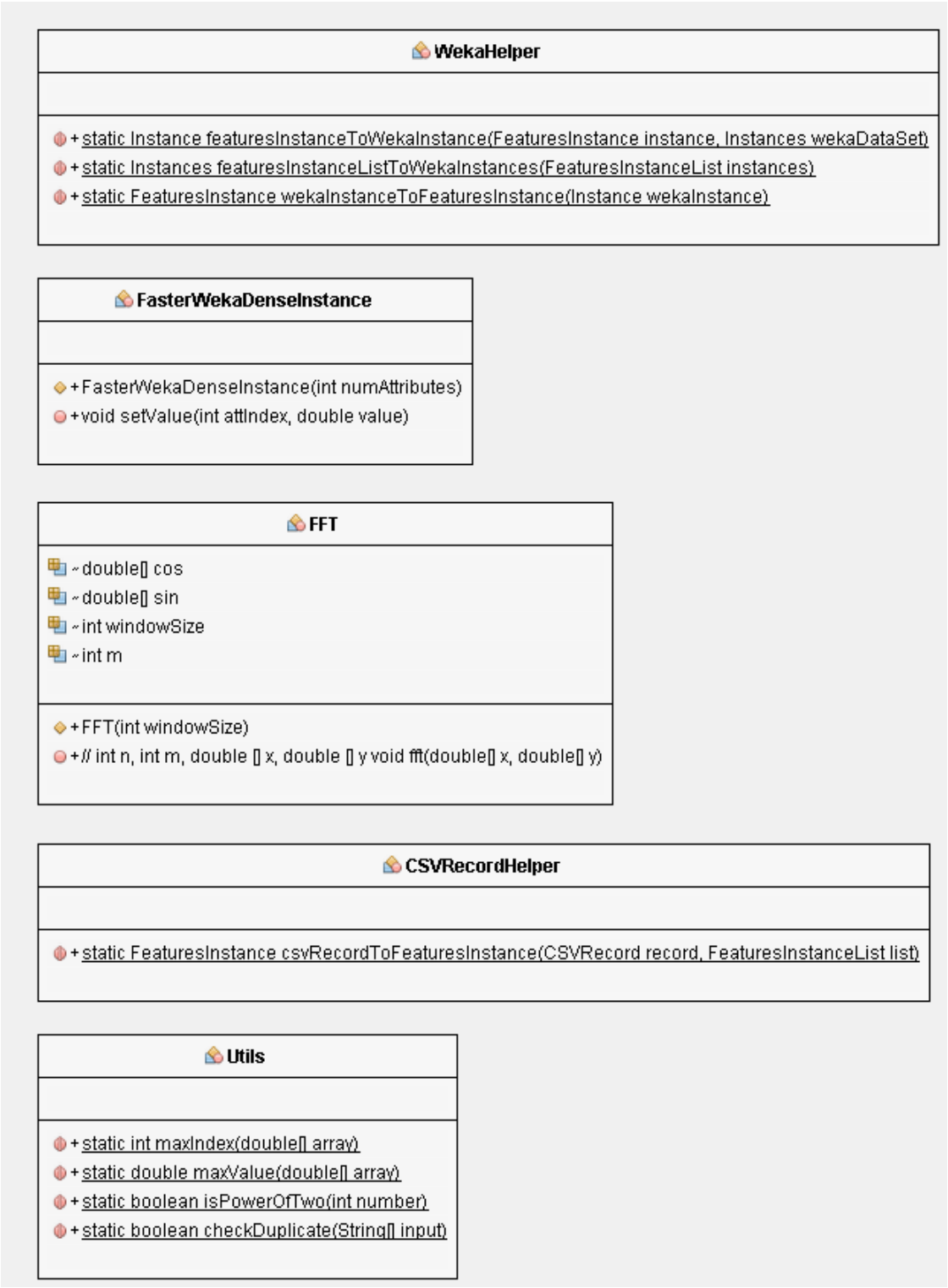


Pacote arsystem.featureextraction.featuresinstance

Pacote arsystem.featureextraction.files



Pacote arsystem.featureextraction.helpers

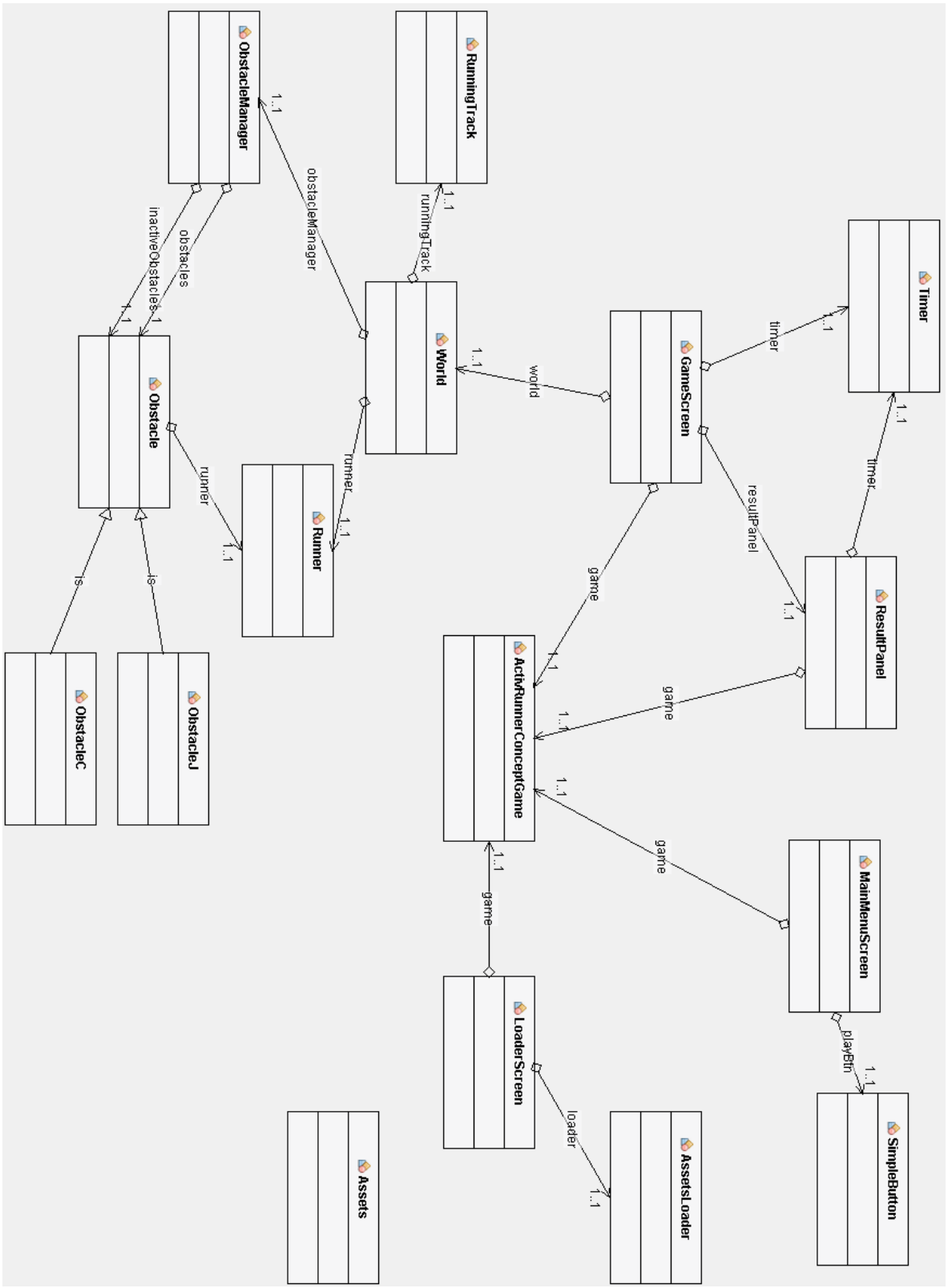


Anexo D – Código exemplificativo da utilização da biblioteca

```
// ---------------------
//    create system
// ---------------------

final int windowSize = 64;
final int overlapSize = 32;

//SUBSYSTEM
ARSubSystemBuilder subSystemBuilder = new ARSubSystemBuilder("SIDE");
subSystemBuilder
.featureExtractor("acc_x", RawNumericData.class).passDataDimension("acc", "x").done()
.featureExtractor("difPosMaxMin_acc_x", DifPosMaxMin.class).passDataDimension("acc", "x").done()
.featureExtractor("acc_x_quarterMean", QuarterMeans.class).passDataDimension("acc", "x").done()
.featureExtractor("max_acc_x", Max.class).passDataDimension("acc", "x").done()
.featureExtractor("min_acc_x", Min.class).passDataDimension("acc", "x").done()
.featureSelector( new WekaFeatureSelector( new CfsSubsetEval() ) )
.classificationLabels("LEFT", "RIGHT")
.classifier( new WekaClassifier( new RandomForest() ) );

//MAIN SYSTEM
ARSystemBuilder systemBuilder = new ARSystemBuilder("ARSystem");
ARSystem ars = systemBuilder
.numericDataInputStream( "acc", windowSize, overlapSize, "mag", "x", "y", "z" )
.numericDataInputStream( "gyro", windowSize, overlapSize, "mag", "x", "y", "z" )
//feature extractors
.featureExtractor("acc_x_fft", FFTMag.class).passDataDimension("acc", "x").done()
.featureExtractor("entropy_acc_x", Entropy.class).
        passPreviousExtractedFeature("acc"+"_x_fft").done()
// etc ...
.classifier( new WekaClassifier( new RandomForest() ) )
.classificationLabels("STANDING", "SIDE", "SQUAT", "JUMP")
.subSystem(subSystemBuilder)
.subLabelDelimiter("_")
.reuseSameInstance(true)
.createSystem();

System.out.println( ars.toString() );

// ---------------------
//    train system
// ---------------------
ars.setMode(ARSystem.MODE_TRAINING);

while(true){ //add your own logic instead
        while(!ars.isFull()){
                ars.addData(0, 1.0, 0.8, 0.1, 0.2); //add data from acc
                ars.addData(1, 1.0, 0.8, 0.1, 0.2); //add data from gyro

        }
        ars.extractFeatures( "standing" ); //←
}
ars.trainClassifier();//←

// ---------------------
//    test system
// ---------------------
ars.setMode(ARSystem.MODE_TESTING);

while(true){ //add your own logic instead
        while(!ars.isFull()){
                ars.addData(0, 1.0, 0.8, 0.1, 0.2); //add data from acc
                ars.addData(1, 1.0, 0.8, 0.1, 0.2); //add data from gyro

        }
        String label = ars.classify(); //←
        System.out.println( " ---> Classification: " + label );
}
```

```
//---------------------------------
// Custom Feature Extractor Example
//---------------------------------
public class SomeFeatureExtractor extends AbstractNumericFeatureExtractor{

        ImmutableNumericData data;

        public SomeFeatureExtractor(ImmutableNumericData data) {
                super(data);
                // store data input values
                this.data = data;
        }

        @Override
        public void extractFeatures(ExtractedFeature featureResult) {
                // do something with the data...
                double [] values = data.getValues();
                // add result to a feature result object
                featureResult.addValues(values);

        }

}
```

Anexo E – Diagrama de classes referente à lógica da prova de conceito


Timer
RunningTrack
ObstacleManager
GameScreen
World
Obstacle
Runner
ResultPanel
ActivRunnerConceptGame
MainMenuScreen
SimpleButton
LoaderScreen
AssetsLoader
Assets
ObstacleC
ObstacleJ
timer
runningTrack
obstacleManager
obstacles
inactiveObstacles
world
resultPanel
game
runner
playBtn
loader
is
1..1