**Faculdade de Engenharia da Universidade do Porto**



**Mestrado Integrado em Engenharia Informática e Computação**

2º ano

**Laboratório de Programação Orientada por Objetos - EIC0111**

Ano Letivo 2014/2015

“**Air Hockey**”

**Estudantes**

Diogo Belarmino Coelho Marques

up201305642@fe.up.pt

Pedro Miguel Pereira de Melo

up201305618@fe.up.pt

Índice

[1. Introdução 3](#_Toc421444563)

[2. Manual (aplicação *desktop*) 4](#_Toc421444564)

[2.1 Menu inicial 4](#_Toc421444565)

[2.2 Menu *“PREFERENCES”* 6](#_Toc421444566)

[2.3 Menu *“SINGLEPLAYER”* 7](#_Toc421444567)

[2.4 Menu *“MULTIPLAYER”* 8](#_Toc421444568)

[2.5 Modo singleplayer 9](#_Toc421444569)

[2.6 Modo multiplayer 12](#_Toc421444570)

[3. Manual (aplicação *Android*) 15](#_Toc421444571)

[3.1 Menu inicial 15](#_Toc421444572)

[4. Concepção e implementação 19](#_Toc421444573)

[4.1 Bibliotecas, tecnologias e ferramentas utilizadas 19](#_Toc421444574)

[4.2 Padrões de desenho 20](#_Toc421444575)

[4.3 Testes unitários 22](#_Toc421444576)

[4.4 Mecanismos de comunicação 23](#_Toc421444577)

[5. Diagrama de casos de utilização 24](#_Toc421444578)

[6. Diagrama de pacotes *(packages)* 25](#_Toc421444579)

[7. Diagrama de classes 27](#_Toc421444580)

[7.1 *Package* lpoo.proj2 27](#_Toc421444581)

[7.2 *Package* lpoo.proj2.audio 28](#_Toc421444582)

[7.3 *Package* lpoo.proj2.gui 29](#_Toc421444583)

[7.4 *Package* lpoo.proj2.logic 30](#_Toc421444584)

[7.5 *Package* lpoo.proj2.net 31](#_Toc421444585)

[8. Conclusão 32](#_Toc421444586)

# Introdução

# Manual (aplicação *desktop*)

## 2.1 Menu inicial



O menu inicial do Air Hockey surge quando se inicia a aplicação. Através dele o pode iniciar uma nova partida no modo singleplayer, modificar os controlos da barra ou entrar no menu do modo multiplayer. Existem ao todo quatro botões distintos neste menu:

* **Botão *“SINGLEPLAYER”*** – inicia uma nova partida no modo *singleplayer*.



* **Botão *“MULTIPLAYER”*** – inicia uma nova partida no modo *multiplayer.*



Figura 1 – menu inicial do Air Hockey

* **Botão “PREFERENCES”** - possuí dois estados, *Mouse* (por defeito) e *Keyboard* que indicam qual o dispositivo a utilizar para controlar a barra durante o jogo no modo singleplayer.



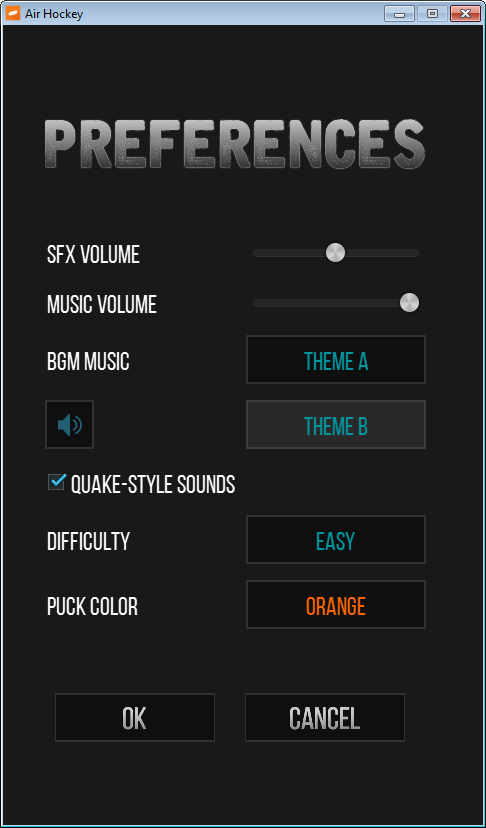
* **Botão “CREDITS”**



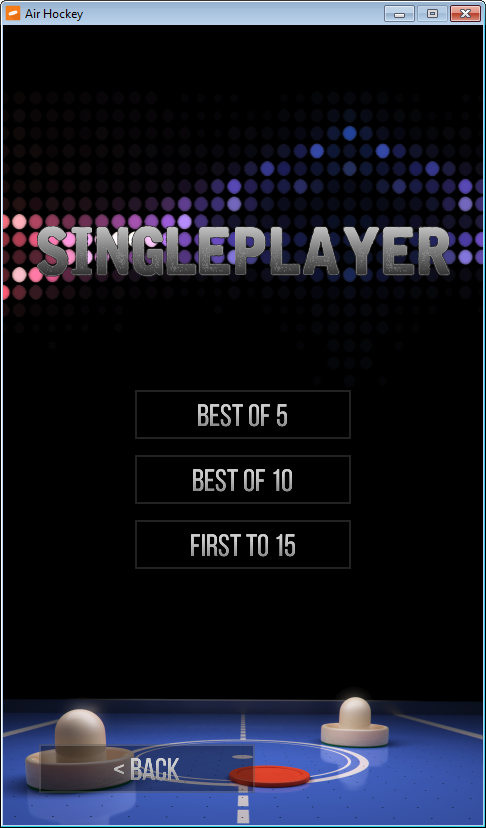
* **Botão “EXIT”** – encerra a aplicação quando pressionado.



## Menu *“PREFERENCES”*

* **SFX VOLUME**
* **MUSIC VOLUME**
* **BGM MUSIC**
* **QUAKE-STYLE SOUNDS**
* **DIFFICULTY**
* **PUCK COLOR**
* **Botão “OK”**
* **Botão “Cancel”**

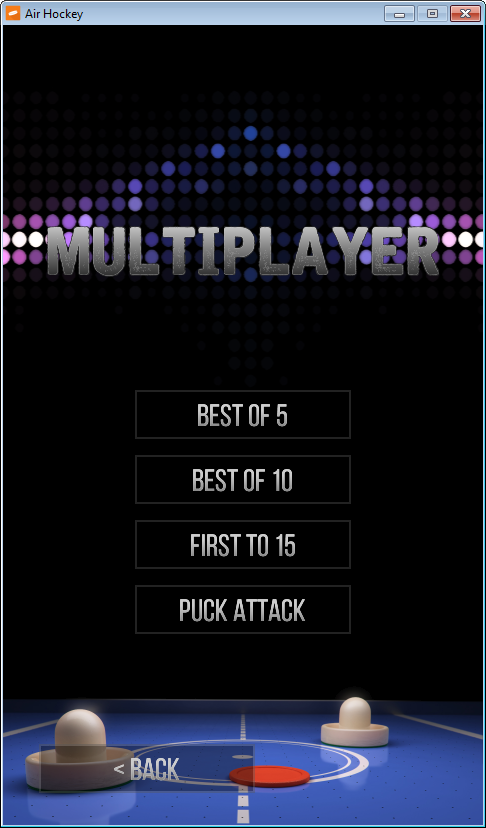
## Menu *“SINGLEPLAYER”*

O menu do modo *singleplayer* permite iniciar uma nova partida no modo *singleplayer* (um único jogador); **do tipo local**, em que ambos os jogadores se enfrentam no mesmo computador ou **do tipo em série**, no qual ambos os jogadores se enfrentam em computadores diferentes.

* **Botão “*BEST OF 5*”** - inicia uma nova partida do tipo “melhor de cinco”
* **Botão “*BEST OF 10*”** - inicia uma nova partida do tipo “melhor de dez”
* **Botão “*FIRST TO 15*”** - inicia uma nova partida do tipo “primeiro a chegar aos quinze pontos”
* **Botão “< BACK”** – permite sair do menu *singleplayer*, regressando ao menu inicial.



## Menu *“MULTIPLAYER”*

**O menu do modo *multiplayer* permite iniciar uma nova partida no modo *multiplayer* (dois jogadores); **do tipo local**, em que ambos os jogadores se enfrentam no mesmo computador ou **do tipo em série**, no qual ambos os jogadores se enfrentam em computadores diferentes.

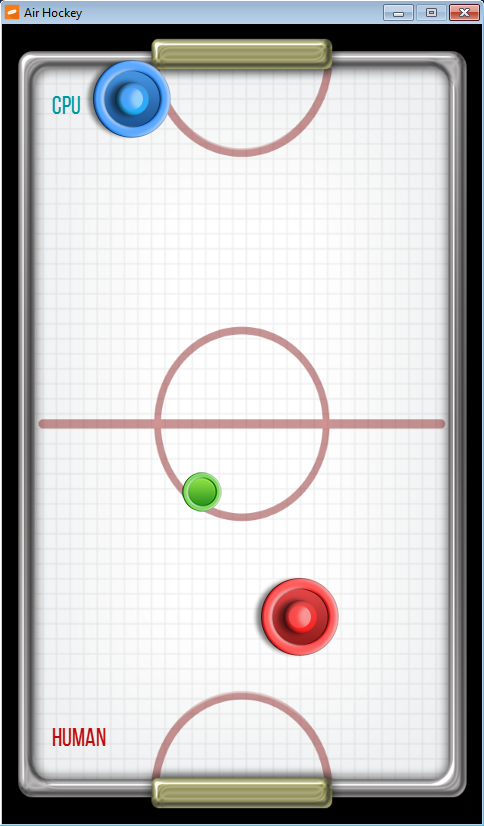
* **Botão “*BEST OF 5*”** - inicia uma nova partida do tipo “melhor de cinco”
* **Botão “*BEST OF 10*”** - inicia uma nova partida do tipo “melhor de dez”
* **Botão “*FIRST TO 15*”** - inicia uma nova partida do tipo “primeiro a chegar aos quinze pontos”
* **Botão “*PUCK ATTACK*”** - inicia uma nova partida do tipo “primeiro a chegar aos quinze pontos”



* **Botão “< BACK”** – permite sair do menu *multiplayer*, regressando ao menu inicial.



## Modo singleplayer

No modo singleplayer, o jogador terá de ultrapassar três níveis distintos. Para tal necessita de controlar uma barra e uma bola, de forma a destruir todos os blocos presentes em cada um dos níveis. O utilizador possui um total de três vidas, com as quais terá de ultrapassar todos os níveis do Arkanix. O jogador perde uma vida sempre que não conseguir desviar a bola com a barra, deixando-a tocar no fundo da área de jogo.

O jogador possui ainda uma pontuação, que vai aumentando consoante o número de blocos que destruir.

Na figura seguinte pode ver-se o último nível do Arkanix:

A componente principal deste modo é a **área de jogo** (figura à esquerda), estando toda a ação da partida aqui confinada. No topo da área de jogo encontram-se os **blocos** a eliminar pelo jogador.

Figura 4 – blocos do jogo

Na parte inferior da área de jogo encontra-se a **barra** e a **bola**, inicialmente presa. O utilizador deve libertar a bola na posição que lhe parecer estrategicamente mais conveniente, necessitando apenas de pressionar a tecla *Enter* ou o botão esquerdo do rato consoante esteja a utilizar o teclado ou o rato, respetivamente.

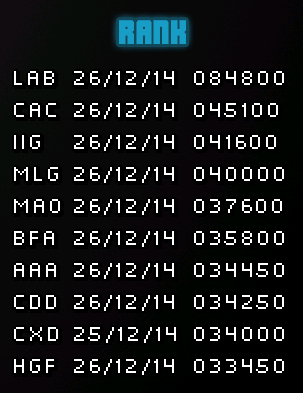


Figura 5 – barra do jogador, com respetiva bola

No centro da área de jogo pode ainda ver-se uma **etiqueta** que indica o número do respetivo nível a ser jogado.

****

Figura 6 – informação do nível a ser jogado



No lado direito da área de jogo, o jogador pode consultar **a tabela de melhores pontuações** (figura à direita), onde se mostram as melhores classificações no modo singleplayer até ao momento da partida. Ao lado de cada pontuação é possível ver-se o nome do jogador que a obteve bem como a data desse feito.

Acima da tabela de pontuações o jogador pode consultar a sua **pontuação** em todos os instantes da partida (figura à direita). A pontuação de um jogador incrementa sempre que a bola atinge um dos blocos na área de jogo. Aos blocos de diferentes cores está atribuída uma diferente pontuação (ver anexo).

Acima da área de jogo, no canto superior esquerdo pode ver-se as restantes **vidas do jogador** (figura à direita). O jogador começa sempre uma partida com três vidas, únicas para os três níveis. O jogador perde uma vida sempre que a bola tocar no fundo da Área de Jogo sem ser defletida pela barra. Quando isto acontece o número de vidas é também atualizado no ecrã.

Se a pontuação do jogador estiver entre as melhores, mesmo que não tenha conseguido superar todos os níveis, terá direito a inserir uma nova pontuação da tabela de melhores pontuações, sendo mostrado o ecrã seguinte.



O jogador deverá então utilizar o teclado para inserir o seu nome. A pontuação será automaticamente inserida na tabela de pontuações, assim que carregar na tecla *Enter*.

Quando uma partida no modo singleplayer termina o utilizador é redirecionado novamente para o menu iniciar. Deve-se notar que o estado do Botão Controlo assume sempre o valor de defeito (*Mouse*) após cada partida (independentemente do modo jogado).

O jogador pode abandonar o modo singleplayer pressionando a tecla ESC do teclado ou o botão direito do rato caso, se estiver a utilizar, respetivamente, o teclado ou o rato para controlar a barra.

## 2.6 Modo multiplayer

No **modo** **multiplayer**, é possível defrontarem-se dois jogadores.

Assim sendo, existem em simultâneo duas áreas de jogo iguais (*splitscreen*), independentes uma da outra, correspondendo cada uma a um jogador diferente. A mecânica de jogo no modo multiplayer é igual à do modo singleplayer, sendo cada barra controlada do mesmo modo que no modo singleplayer.

Uma novidade deste modo é a presença de um **temporizador** (em segundos) que vai decrementando no decorrer da partida. O tempo de uma partida no modo *multiplayer* é 90 segundos.

Na figura seguinte pode ver-se uma partida do Arkanix no modo multiplayer:

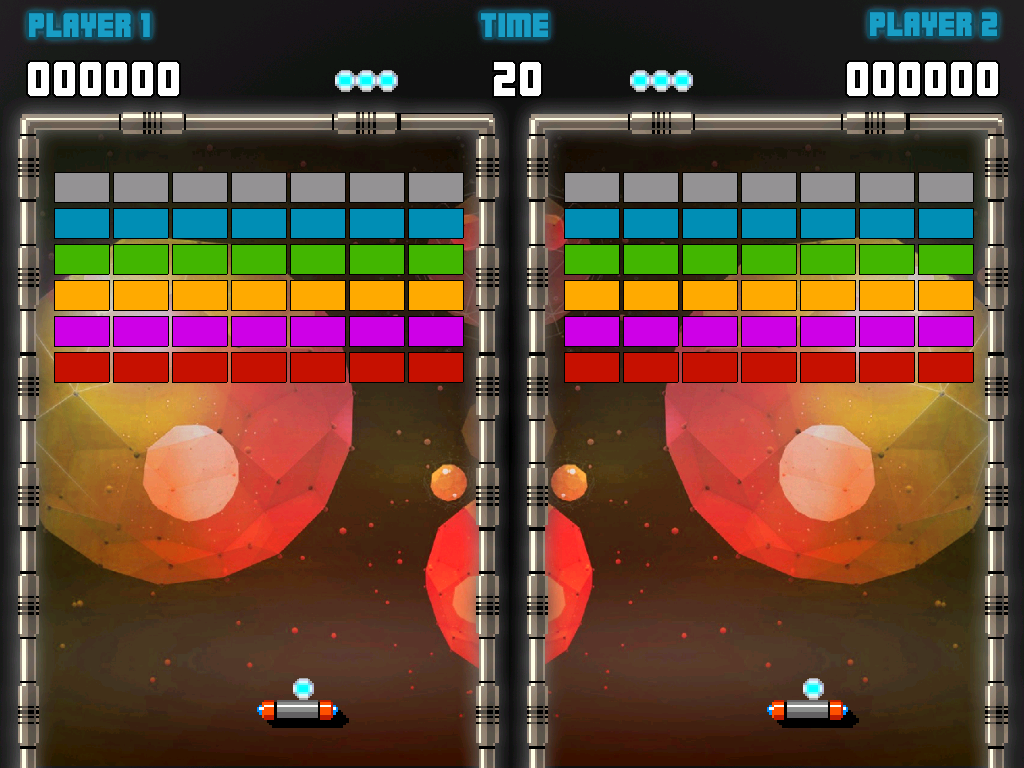


Figura 12 – partida típica no modo multiplayer

Acima de cada uma das áreas de jogo pode ver-se as pontuações de cada jogador bem como o respetivo número de vidas (em semelhança ao modo singleplayer).



Figura 13 – número de vidas e pontuação de um jogador

É possível ver entre as áreas de jogo o temporizador, constantemente atualizado e que delimita o tempo máximo até ao fim da partida.



Figura 14 – temporizador do modo multiplayer

É sagrado vencedor o jogador que no menor tempo conseguir uma maior pontuação (destruindo assim um maior número de blocos).

Sempre que se chega ao fim de uma partida surge o ecrã do vencedor.

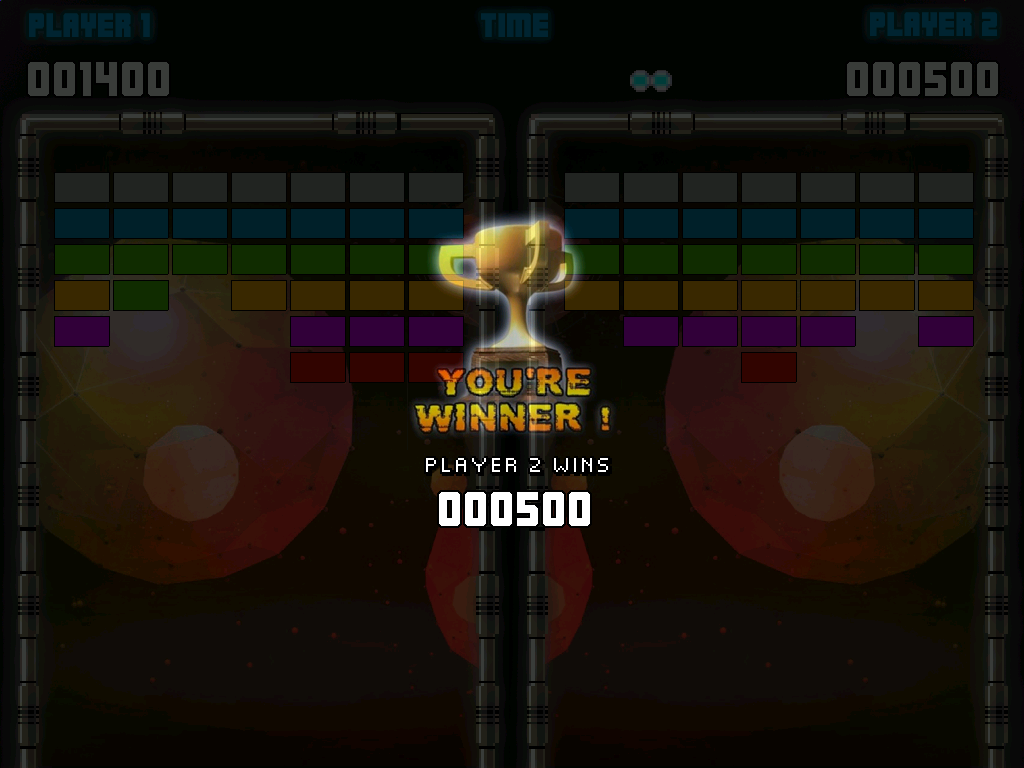


Figura 15 – ecrã do jogador vencedor

Aqui pode ver-se o jogador vencedor e a pontuação obtida por este no decorrer da partida.

Se o temporizador chegar a zero sem que haja vencedor, a partida termina e é mostrado o ecrã de empate (figura em baixo). Ainda em caso de empate, é mostrada a pontuação obtida por ambos os jogadores.

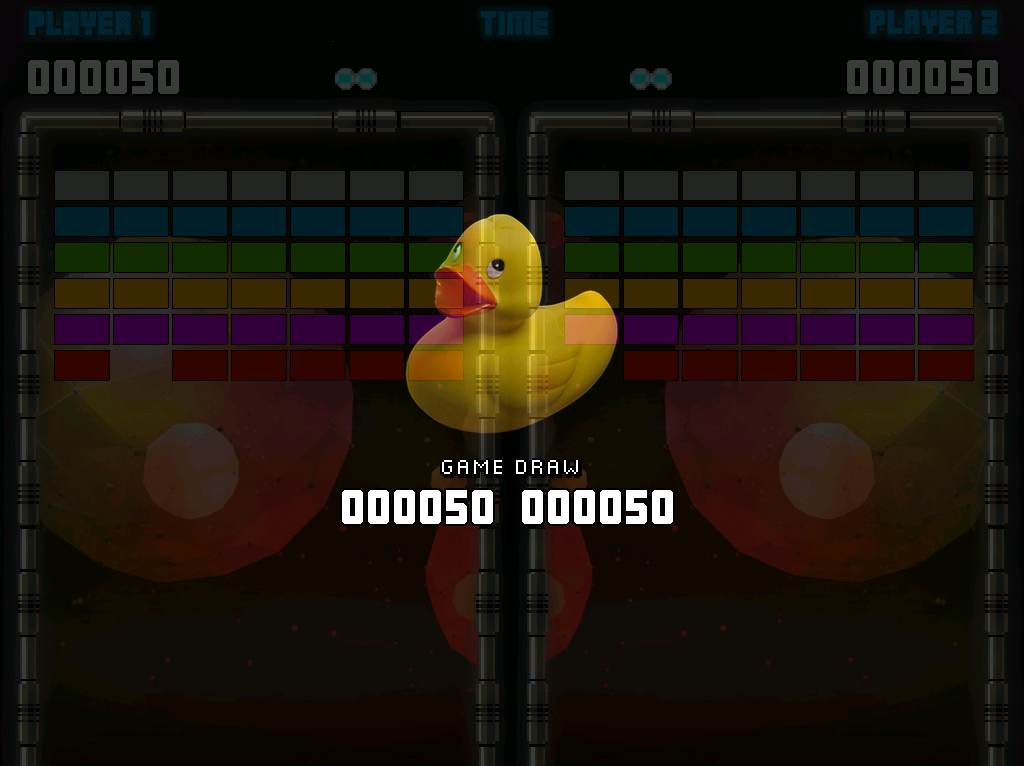


Figura 16 – ecrã mostrado numa situação de empate

Um jogador pode ainda desistir, utilizando os mesmos atalhos definidos na secção anterior. Nesse caso, o adversário sagra-se vencedor e é mostrado um ecrã a avisar do sucedido.

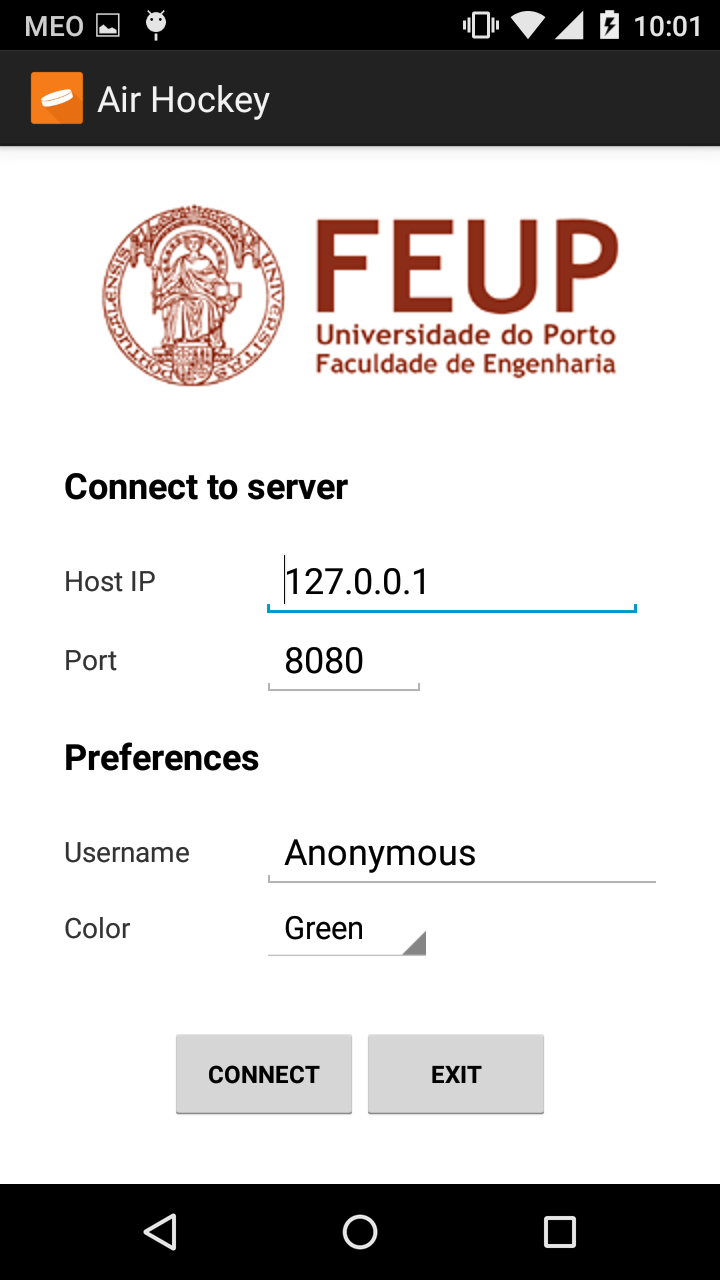


Figura 17 – ecrã mostrado em caso de desistência de um jogador

No modo multiplayer não existe tabela de pontuações pelo que no fim de cada partida os jogadores serão redirecionados para o menu inicial, após 5 segundos, sem que as melhores pontuações obtidas por eles sejam guardadas.

# Manual (aplicação *Android*)

## 3.1 Menu inicial

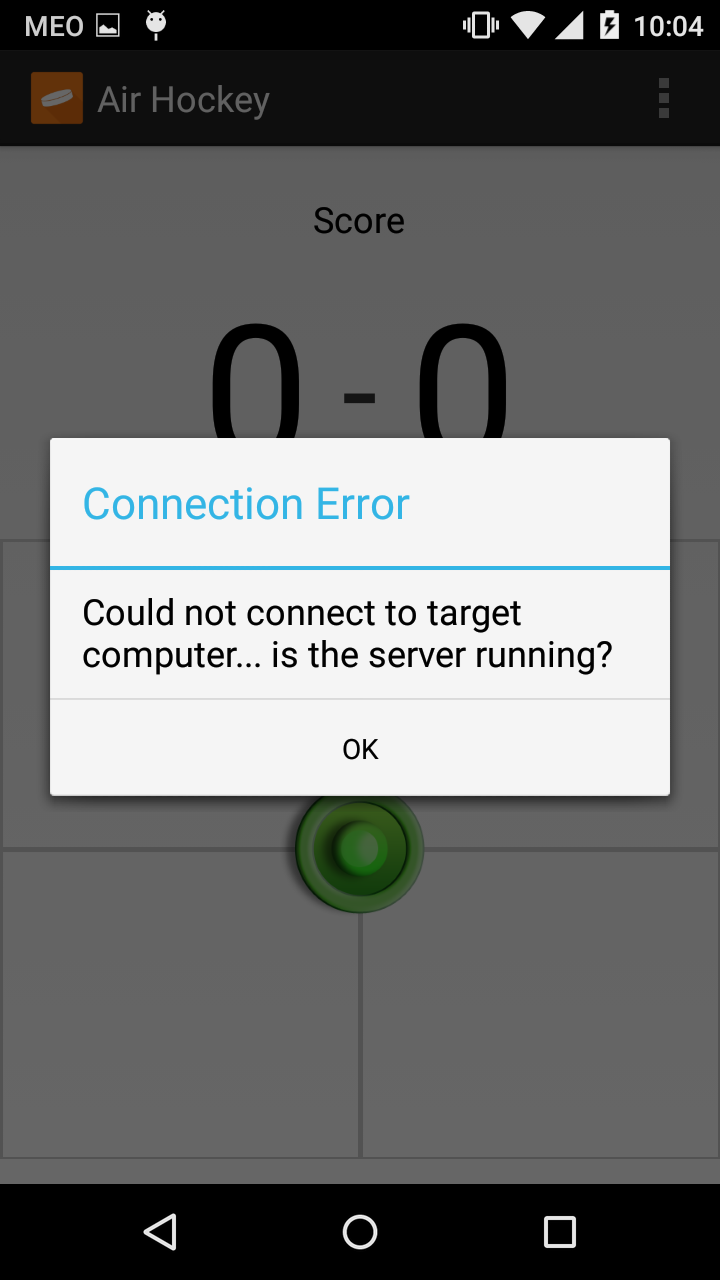
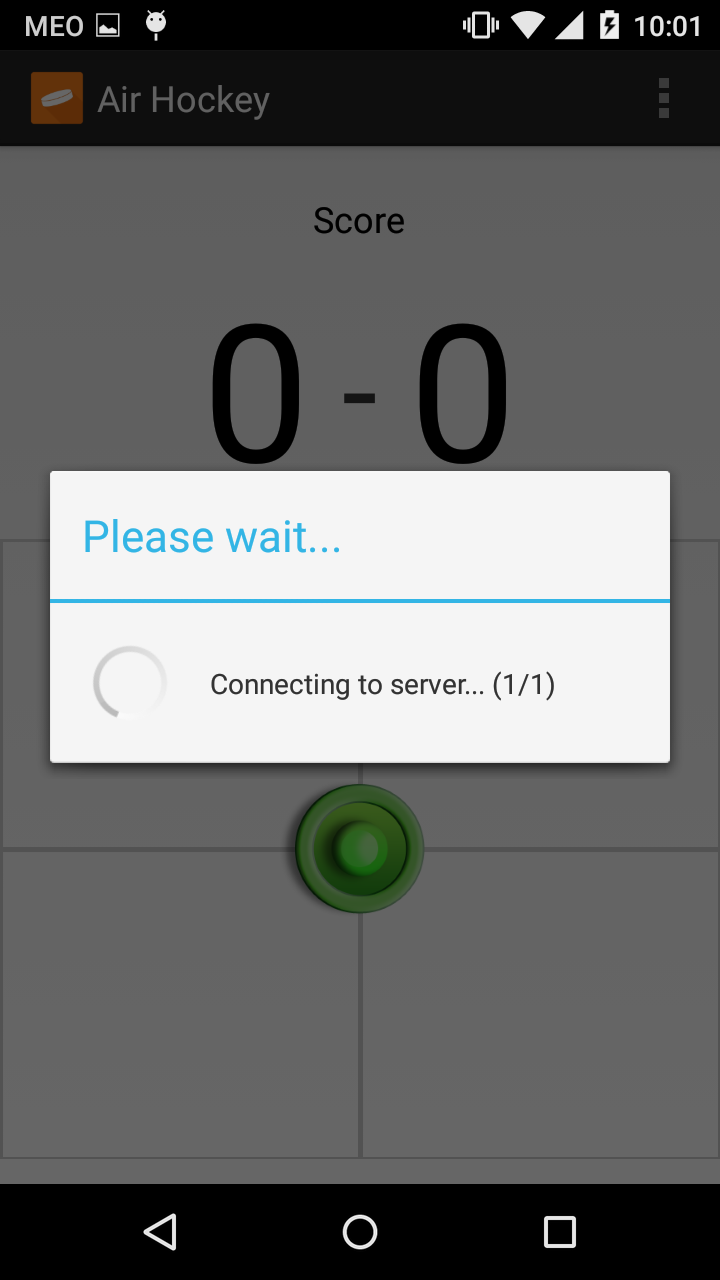


Quando o utilizador inicia a aplicação *Android* do cliente do *Air Hockey bla bla bla...*

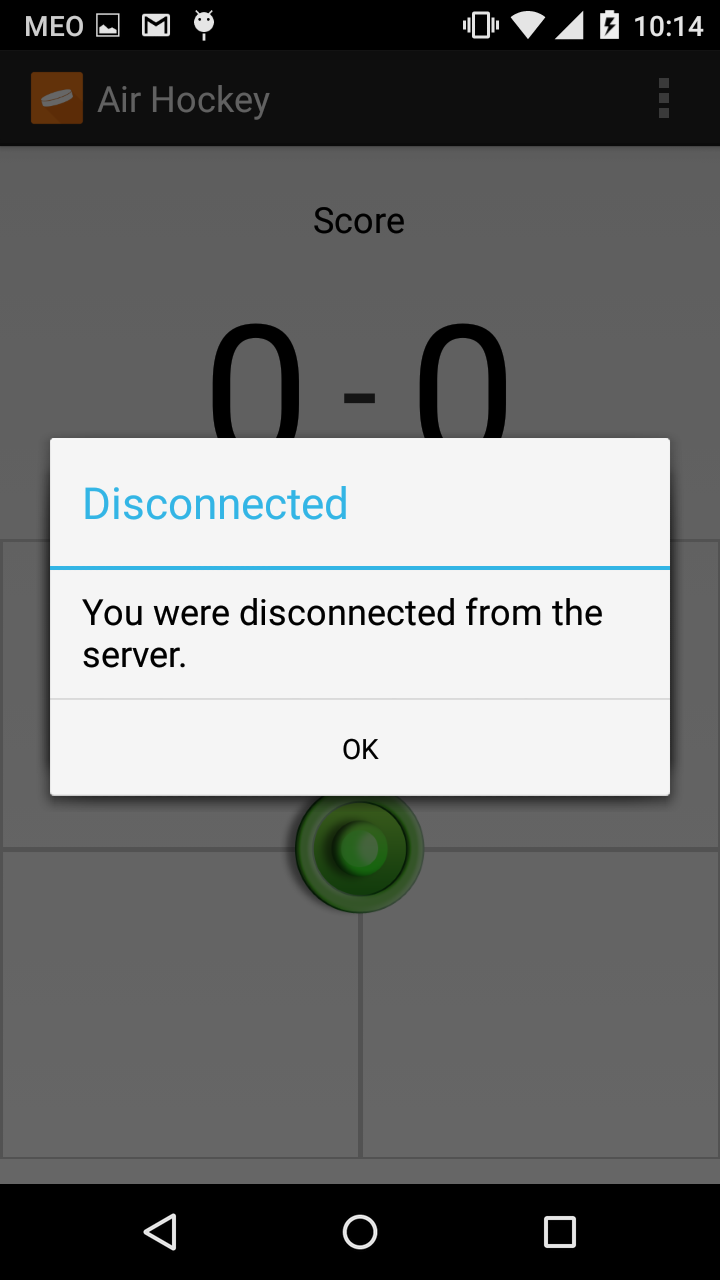
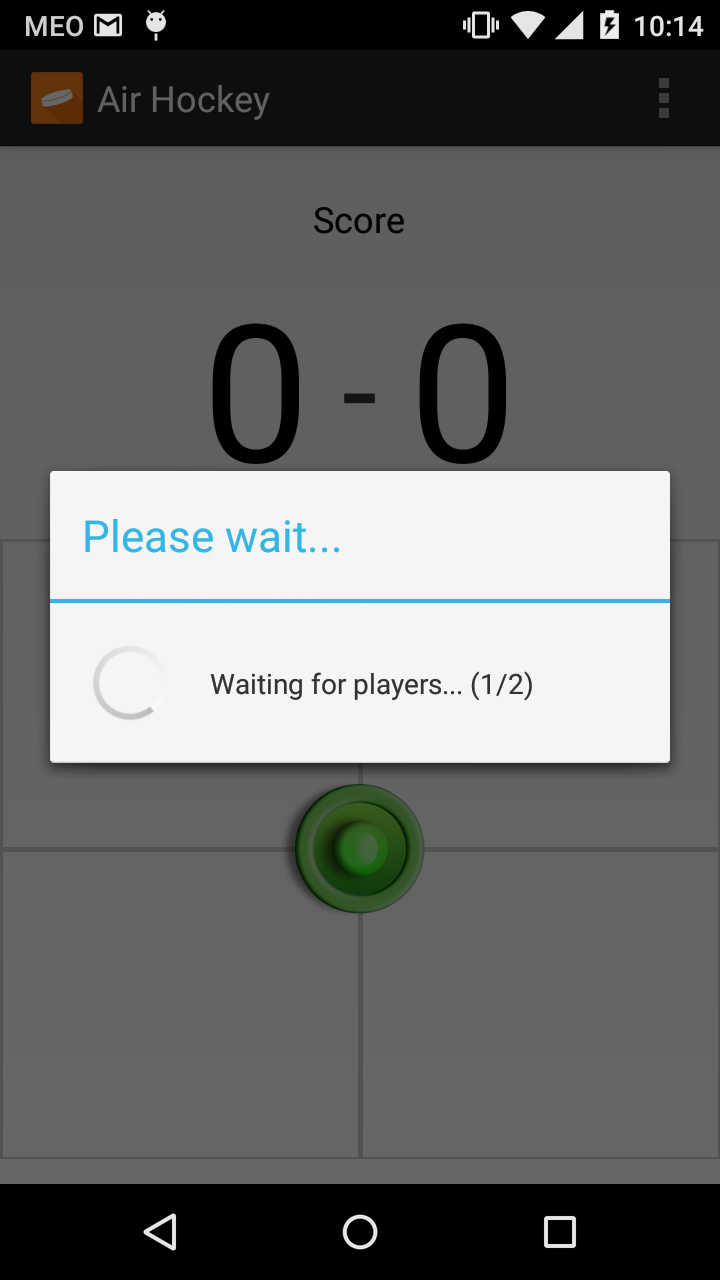
* Host IP / Endereço IP – endereço local / *internet* onde o servidor se encontra a correr
* Port / Porta – porta TCP do servidor
* Username / Nome – nome ou alcunha do utilizador
* Color / Cor – cor do *paddle* do utilizador

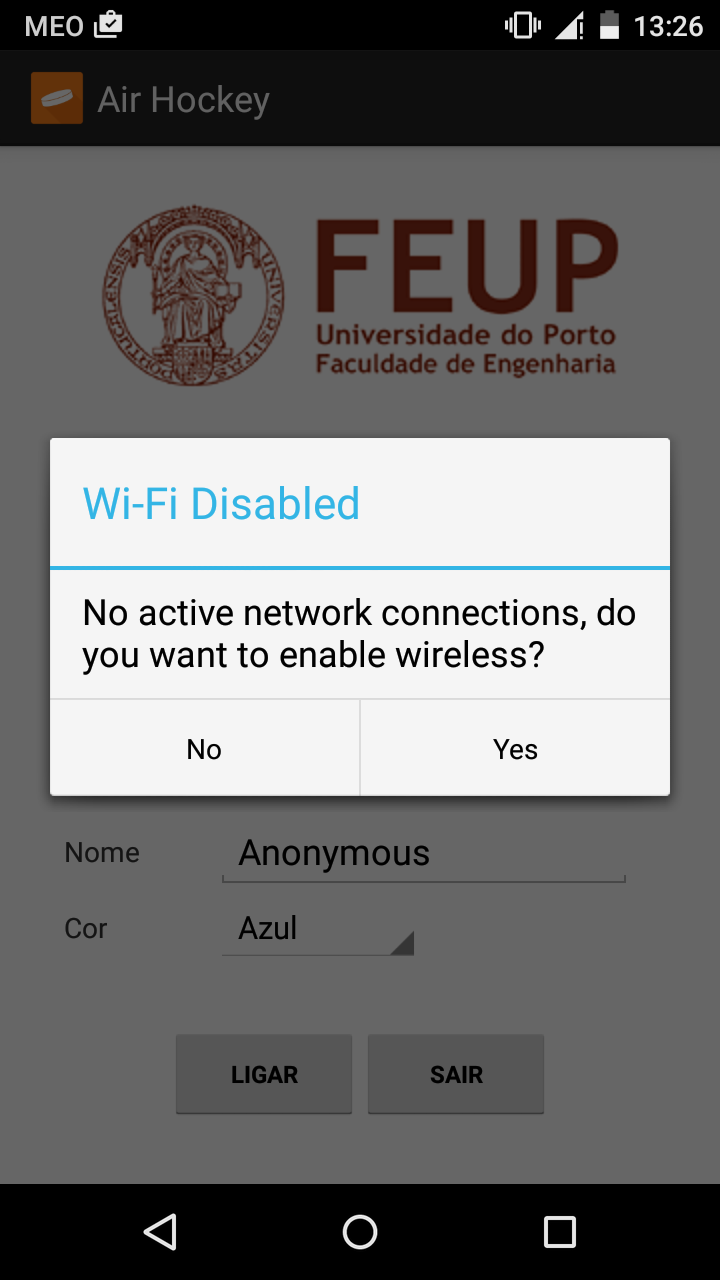
É possível escolher uma das seis cores disponíveis para o *paddle* do utilizador: vermelho (*red*), azul *(blue*), amarelo *(yellow)*, verde *(green)*, laranja *(orange)* e violeta *(purple)*.

Ao carregar no botão “Ligar” é criada uma nova *activity* na aplicação do cliente que representa o campo do jogador.



Após uma tentativa de ligação falhada é apresentada uma janela de diálogo para informar o utilizador de tal acontecimento. Ao pressionar o botão “OK”, o utilizador é encaminhado de volta ao menu inicial da aplicação. Como os dados que introduziu na *activity* principal não são apagados durante a transição para a *activity* do Game, o utilizador pode voltar a conectar-se ao servidor carregando simplesmente no botão “Ligar”.





**(se o utilizador não tiver nenhuma ligação de rede ativa)**

# Concepção e implementação

## 4.1 Bibliotecas, tecnologias e ferramentas

**Bibliotecas utilizadas no desenvolvimento do projeto:**

* **LibGDX** (http://libgdx.badlogicgames.com) – biblioteca Java multiplataforma para desenvolvimento de aplicações multimédia e jogos. Permite o desenvolvimento e teste dos projetos em computador e que através de alterações mínimas no código o projeto possa ser também exportado e publicado noutras plataformas e tecnologias modernas, tais como *Android*, *iOS*, *HTML5*.
* **Kryonet** (https://github.com/EsotericSoftware/kryonet) – biblioteca Java que implementa uma API simples e eficaz para comunicações em rede cliente-servidor através dos protocolos TCP/UDP e métodos n*on-blocking I/O* nativos. A biblioteca *Kryonet* utiliza a biblioteca de serialização *Kryo* para transferir objectos de forma automatizada e eficiente através de uma rede de computadores. Esta biblioteca tem ainda a vantagem de correr em várias plataformas (*Windows, Linux, Mac e Android*).

**IDEs utilizadas no desenvolvimento do projeto:**

* **Eclipse Luna** (https://www.eclipse.org/luna) – IDE principal utilizada no desenvolvimento da aplicação *desktop* e na finalização da aplicação Android
* **Android Development Tools Plugin** (http://developer.android.com/tools/sdk/eclipse-adt.html) **–** extensão para o *Eclipse* que permite o desenvolvimento de aplicações *Android*
* **Android Studio** (https://developer.android.com/sdk/index.html) – IDE oficial utilizada no desenvolvimento de aplicações Android

**Ferramentas utilizadas no desenvolvimento do projeto:**

* **Hiero** (https://github.com/libgdx/libgdx/wiki/Hiero) **–** utilizada na criação das *bitmap fonts* para os menus, apresenta funcionalidades mais avançadas relativamente à ferramenta *BMFont*, tendo sido utilizada sobretudo para aplicar efeitos de degradê e contorno nos tipos de letra utilizados. Integrada na distribução LibGDX.
* **TexturePacker** (https://github.com/libgdx/libgdx/wiki/Texture-packer) – utilizada no “empacotamento” das texturas das componentes da *interface gráfica* num único ficheiro e criação do respetivo dicionário de texturas. Integrada na distribuição LibGDX.
* **BMFont** (http://www.angelcode.com/products/bmfont) **–** para a criação de *bitmap fonts*, ferramenta mais antiga e com menos funcionalidades do que a *Hiero* anteriormente referida, foi utilizada na criação dos tipos de letra de cor
* **Paint.NET** (http://www.getpaint.net/index.html) **–** utilizada na criação das imagens do jogo, da *interface* gráfica da aplicação *desktop* e na adaptação dos *drawables* da aplicação *Android* aos vários tamanhos de ecrã e *pixel densities* (LDPI, MDPI, HDPI, XHDPI)

## 4.2 Padrões de desenho

|  |  |
| --- | --- |
| Facade | *“Provide a unified interface to a set of interfaces in a subsystem. Facade defines a higher-level interface that makes the subsystem easier to use.”*  onde: classe *AudioManager*  porquê: encapsulamento de um sistema complexo (o gestor de áudio da aplicação e os seus subsistemas de gestão de vários ficheiros de áudio, controlo de volume, controlo de reprodução) numa classe que através da implementação de métodos mais simples (*setSFXVolume, setMusicVolume, playSong, playSound*) facilita a interacção entre o programador/utilizador e o sistema em questão |
| Observer | *“Define a one-to-many dependency between objects so that when one object changes state, all its dependents are notified and updated automatically.”*  **onde:** classes *ServerListener*, *ClientListener* implementadas pela biblioteca *Kryonet* para a comunicação em rede entre servidor e cliente, *ClickListener* e *InputProcessor* implementadas pela biblioteca LibGDX para processamento de eventos de *input* do rato e teclado, *MenuListener* como classe derivada de *ClickListener*  **porquê: ...** |
| Singleton | *“Ensure a class has only one instance, and provide a global point of access to it.”*  **onde:** classe*AudioManager*  **porquê:** não faz sentido existir mais do que uma instância do gestor de áudio na aplicação – além de ser uma classe relativamente pesada (vários assets que são carregados em memória quando esta é instanciada, várias instâncias implicaria a existência de conteúdos duplicados em memória, pouco eficiente...), deve ainda permitir a qualquer classe da camada da interface gráfica ou da camada de lógica guardar uma referência a essa instância e aceder globalmente aos seus métodos |
| State | *“Allow an object to alter its behavior when its internal state changes. The object will appear to change its class*.”  **onde:** superclasse *GameState* e todas as suas classes derivadas (*GameRunningState, GamePausedState, GameOverState, PlayerScoredState, WaitingState, ConfirmExitState, DisconnectedState*)  **porquê:** possibilidade de comutar entre vários estados e ecrãs de jogo, representados por instâncias de classes que definem o que é apresentado no ecrã ao utilizador ou como a aplicação reage ao *input* do utilizador |

|  |  |
| --- | --- |
| Template Method | “*Define the skeleton of an algorithm in an operation, deferring some steps to client subclasses. Template Method lets subclasses redefine certain steps of an algorithm without changing the algorithm's structure*.”  onde: superclasse *GameRules* e todas as suas classes derivadas (*RulesBest5, RulesBest10, RulesFirst15 e RulesAttack)*, que implementam os métodos em questão  porquê: subclasses partilham parte dos métodos e atributos, no entanto existem diferenças em três nos métodos implementados *checkOver(), checkLast() e checkTie()* que justificam o aparecimento das mesmas. A superclasse não pode ser instanciada (apresenta funcionalidades genéricas e métodos abstratos, não existem “regras” genéricas para um jogo concreto) |
| Visitor | ***“****Represent an operation to be performed on the elements of an object structure. Visitor lets you define a new operation without changing the classes of the elements on which it operates.”*  **onde:** *interface CollisionDetector*, implementada pela classe abstracta *DynamicEntity*  **porquê:** vários objetos de classes relacionadas (bola, baliza, parede e paddle) que podem interagir entre si, mas cujo comportamento (um único método) depende do conjunto de pares que interagem e da ordem do seu emparelhamento |

## 4.3 Testes unitários

Como se trata de um jogo marioritariamente gráfico, assente na simulação física, não foram implementados testes unitários automáticos em *JUnit*. No entanto, foram realizados testes manuais ao funcionamento do programa e à robustez do código implementado, tanto na aplicação *desktop* como na aplicação do cliente para dispositivos *Android*. Segue-se uma lista dos testes que foram realizados.

**Na aplicação *desktop*:**

* foram realizadas várias tentativas de alterar as preferências da aplicação, verificando se eram aplicadas imediatamente quando o utilizador carregava em “OK” e se eram anuladas quando o utilizador decidia cancelar as alterações feitas
* foram realizadas várias partidas completas no modo *singleplayer*, nos diferentes modos de jogo, tendo sido avaliados os movimentos e reações do adversário controlado por computador
* foram realizadas várias partidas completas no modo *multiplayer*, nos diferentes modos de jogo, com dois jogadores ligados, tendo sido avaliada a estabilidade da ligação e o efeito da latência na recepção dos *packets*

**Na aplicação Android:**

* foram realizadas várias tentativas de ligação a combinações de endereços IP inválidos, e endereços válidos mas com portas não atribuídas para testar a robustez da biblioteca *Kryonet* e dos filtros de entrada de texto (*EditText)* implementados na aplicação *Android*
* foram testados casos em que o servidor é desligado enquanto decorre uma partida *multiplayer* com dois clientes ligados e enquanto um cliente ligado espera pela vinda do segundo
* foram testados casos em que o dispositivo não estava ligado a nenhuma rede móvel/Wi-Fi e o utilizador tentava conectar-se a um servidor
* em qualquer um dos casos anteriores esperava-se que a aplicação apresentasse uma mensagem ao utilizador a relatar o sucedido e fechasse a ligação com o servidor, voltando à *activity* principal
* foram testados casos em que durante o decorrer de uma partida um dos jogadores desiste, carregando no botão “*Retroceder”* ou escolhendo a opção “*Disconnect”* no menu da aplicação

A aplicação Android foi testada nos seguintes dispositivos, com diferente *hardware*, resoluções de ecrã e *pixel densities*:

* **Motorola Moto G** (XHDPI, resolução de ecrã 720x1280, densidade 326 dpi)
* **Huawei Ideos X5** (HDPI, resolução de ecrã 480x800, densidade 242 dpi)

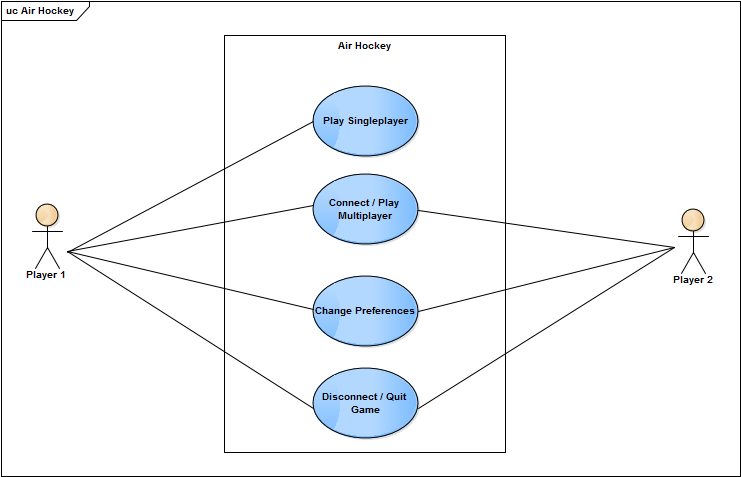
## 4.4 Mecanismos de comunicação

A comunicação em rede implementada consiste na troca de objetos entre cliente e servidor. Estes objetos pertencem a classes relativamente minimalistas, tendo sido implementadas especificamente para este propósito: transferir a menor quantidade de informação entre os dispositivos ligados, e deixar o processamento da informação a cargo do servidor, para garantir um desempenho óptimo na maior gama de redes móveis/Wi-Fi e dispositivos *Android* possível.

Segue-se uma lista das classes utilizadas por ambos os programas na comunicação em rede:

* **GameOver** – utilizado pelo servidor para informar todos os clientes que a partida chegou ao fim e o nome do jogador que se sagrou o vencedor da mesma;
* **PlayerLogin** – utilizado pelo cliente para informar o servidor que este se tentou ligar ao mesmo; é enviado ao servidor o nome/alcunha do jogador e a cor do *paddle* escolhida
* **PlayerConnected** – utilizado pelo servidor para informar os restantes clientes que um novo cliente se juntou ao servidor
* **PlayerDisconnected** – utilizado pelo servidor para informar os restantes clientes que um cliente se desligou do servidor, abandonando a partida
* **ServerFull** – utilizado pelo servidor para informar o cliente que se tentou conectar de que o servidor atingiu a sua capacidade máxima (dois jogadores); este objecto não possui parâmetros
* **UpdatePaddle** – utilizado pelo cliente para informar o servidor da nova posição do *paddle* correspondente a esse cliente; são enviadas as coordenadas *x* e *y* da nova posição para o servidor sempre que o utilizador interage com a vista de controlo *do paddle* na aplicação *Android*
* **UpdateScore** – utilizado pelo servidor para informar todos os clientes de uma situação de golo e atualizar a respetiva pontuação; são enviadas as pontuações de ambos os jogadores, *p1Score* e *p2Score* sempre que o servidor recebe este pedido

# Diagrama de casos de utilização

****

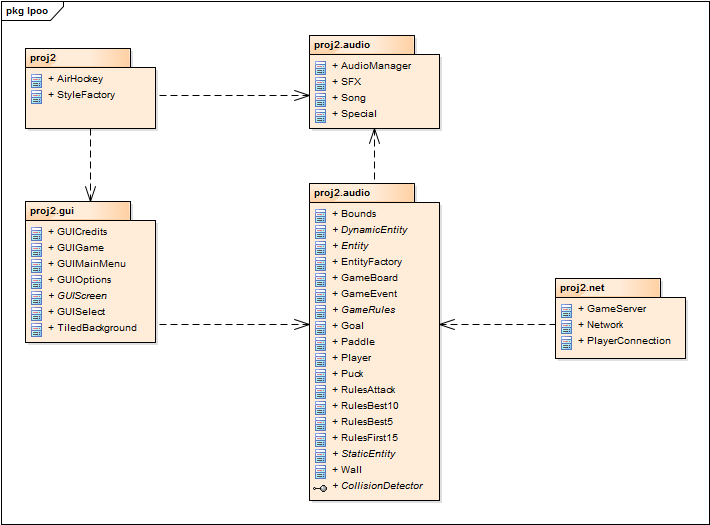
***Player* 1**

* iniciar uma nova partida no modo *singleplayer*
* iniciar uma nova partida no modo *multiplayer* (criar um servidor e/ou juntar-se a uma partida já existente)
* alterar a configuração global (volume dos sons, volume da música, música de fundo, cor da bola)
* alterar a configuração do modo *singleplayer* (dificuldade do adversário)
* alterar o perfil do jogador (alcunha e cor do *paddle*)
* abandonar uma partida no modo *multiplayer*, desconectando-se do servidor
* encerrar a aplicação

***Player* 2**

* iniciar uma nova partida no modo *multiplayer* (criar um servidor e/ou juntar-se a uma partida já existente)
* alterar o perfil do jogador (alcunha e cor do *paddle*)
* abandonar uma partida no modo *multiplayer*, desconectando-se do servidor

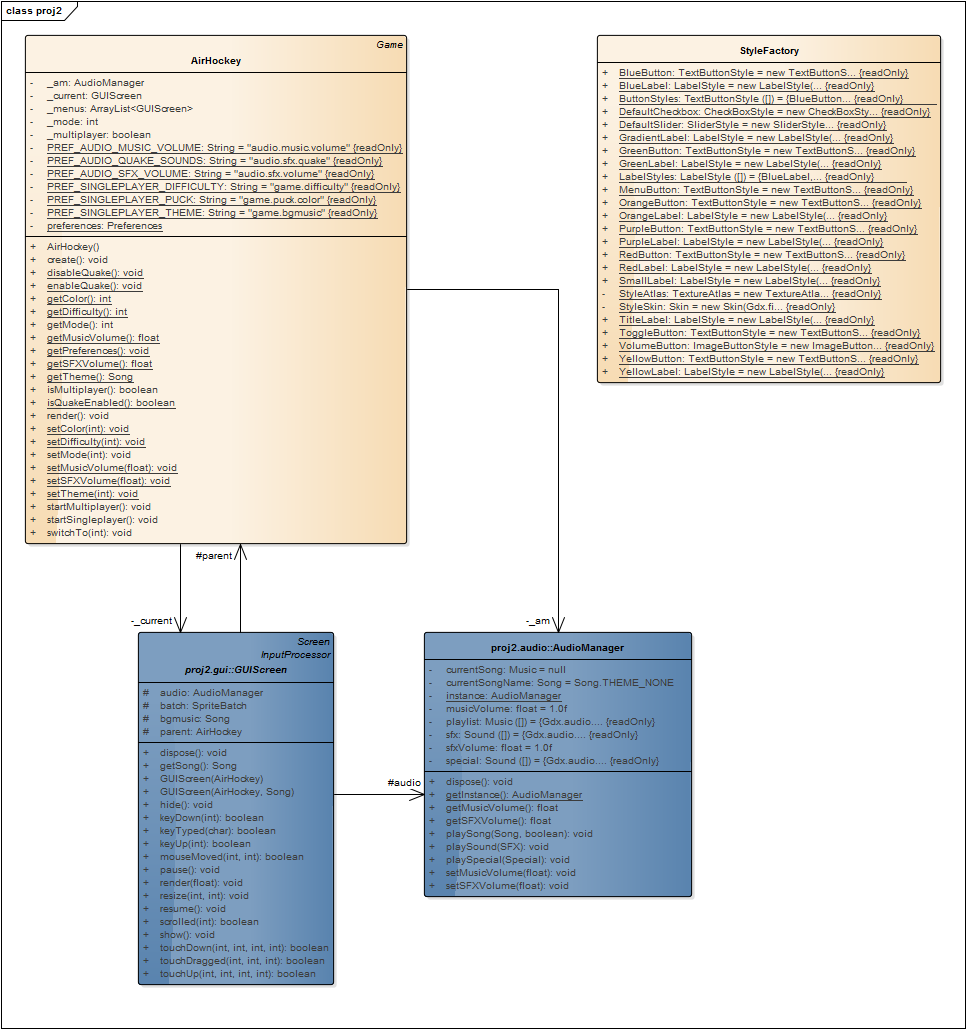
# Diagrama de pacotes *(packages)*



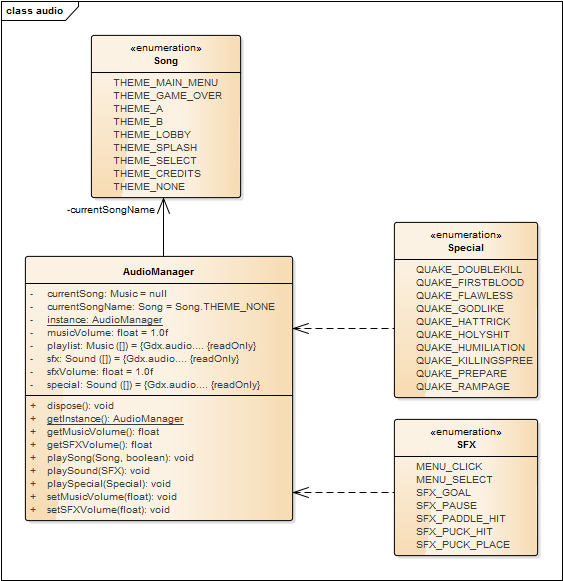
|  |  |
| --- | --- |
| *Package* | Descrição |
| *proj2* | * camada principal (topo hierárquico) da aplicação, constituída por apenas duas classes, *AirHockey* e *StyleFactory* * a classe *AirHockey* contém o gestor de ecrãs, responsável pela transição entre ecrãs e menus da aplicação, bem como métodos de serialização para carregamento e armazenamento das preferências do utilizador * permite à camada da lógica obter informações do modo a ser jogado (*singleplayer* ou *multiplayer*) e da dificuldade do adversário no modo *singleplayer* * a classe *StyleFactory* armazena e carrega para a memória vários *assets* e estilos visuais de forma a serem utilizados posteriormente pelas classes da camada gráfica com tempo de acesso mínimo |
| proj2.*audio* | * contém toda a camada de áudio da aplicação, funcionando como gestor e controlador de som integrado * durante o arranque da aplicação armazena em memória todos os *assets* de áudio da aplicação, de forma a serem utilizados posteriormente com tempo de acesso mínimo * constituído por uma classe (AudioManager), contendo a lógica de controlo e três enumeráveis (*SFX*, *Song* e *Special*) que identificam o ficheiro de som a ser reproduzido *e* representam, respetivamente, efeitos sonoros, música de fundo e voz * existe ainda a possibilidade de aumentar/diminuir o volume da música de fundo e dos efeitos sonoros de forma independente e de desativar a voz, sendo esta substituída por outros efeitos sonoros |
| proj2.*gui* | * contém toda a camada da *interface* gráfica da aplicação, isto é, tudo o que possa ser apresentado no ecrã ao utilizador, bem como métodos e *listeners* que permitem a interação do utilizador com o jogo através do rato e do teclado * constituído por várias classes, cada uma representando menus ou ecrãs distintos (menu inicial, ecrã do jogo, menu de opções, menus de escolha, etc...), à excepção de *TiledBackground* * *TiledBackground* é uma componente gráfica dos menus (responsável pela criação de imagens de fundo com *scrolling* infinito), tendo sido também incluída na camada gráfica da aplicação |
| proj2.*logic* | * contém toda a camada lógica do jogo (simulação de física, barreiras, balizas) * estabelece também uma ponte de comunicação entre a *interface* gráfica e a camada de rede, ao permitir que eventos do servidor sejam também notificados ao utilizador através da *interface* * ***COMPLETAR...........*** |
| proj2.*net* | * camada de comunicação por rede que permite a criação de partidas *multiplayer* entre dois jogadores (clientes), que podem interagir com o servidor através de um telemóvel *smartphone* com sistema operativo *Android* (arquitetura cliente-servidor) * constituída por apenas duas classes, *GameServer* e *Network*, sendo esta constituída por várias *inner classes* que representam mensagens utilizadas comunicação entre os clientes e o servidor e vice-versa * a gestão dos clientes é feita pela classe *Game*Server |

# Diagrama de classes

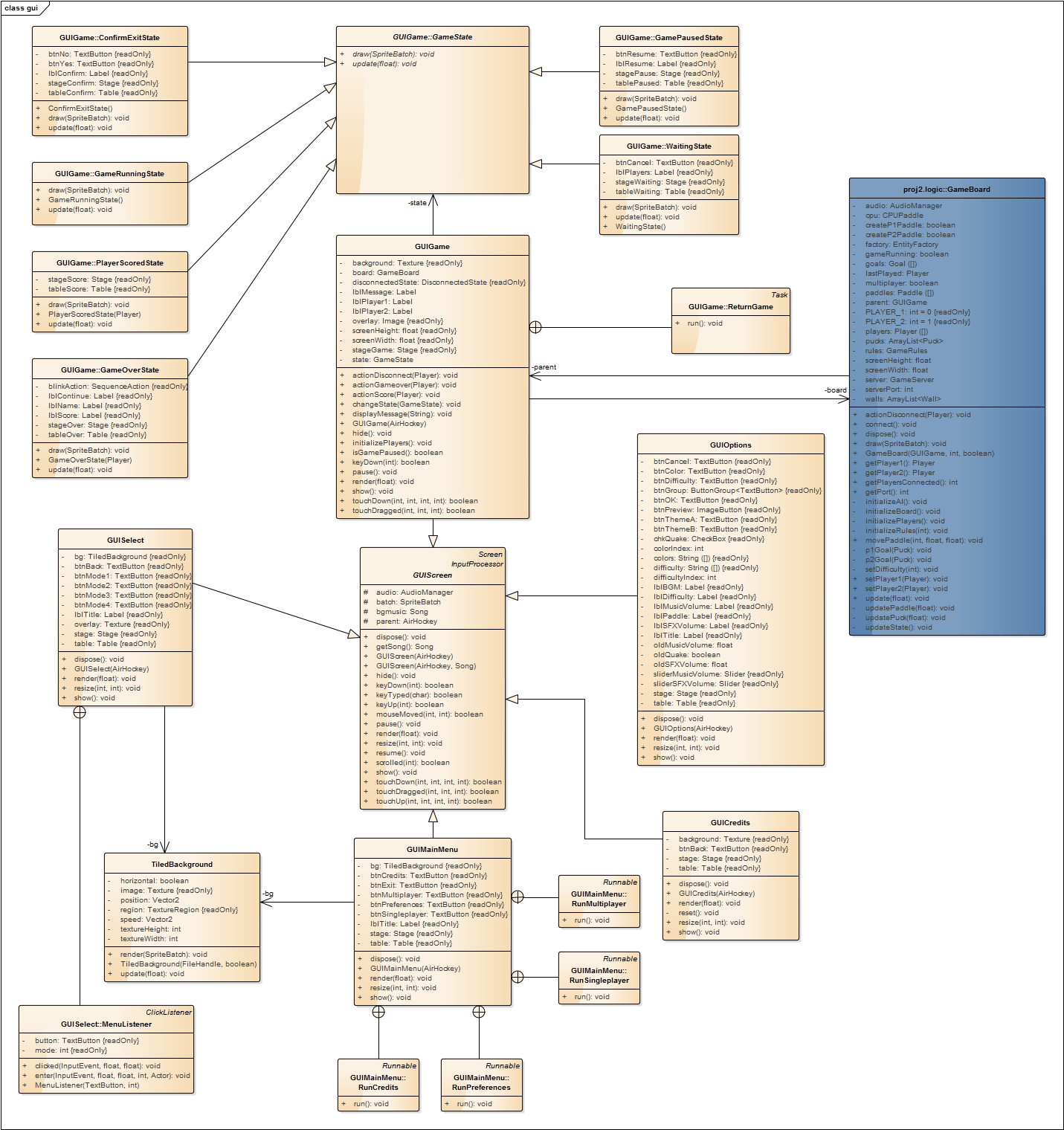
## 7.1 *Package* lpoo.proj2



## 7.2 *Package* lpoo.proj2.audio



## 7.3 *Package* lpoo.proj2.gui



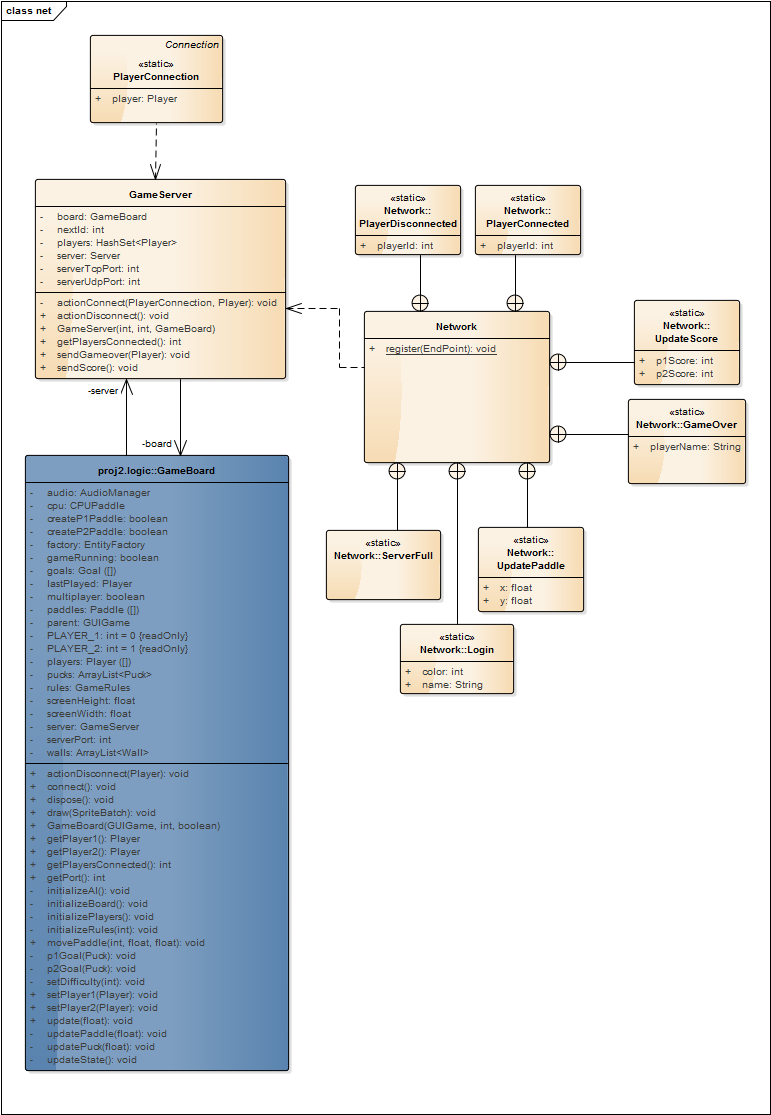
## 7.4 *Package* lpoo.proj2.logic

A porta série implementada transmite e recebe dados a uma taxa de bits fixa de 2400 bauds durante todo o seu tempo de funcionamento. A transmissão é feita por varrimento (*polling*), isto é, é verificado o estado do *Transmitter Holding Register*, e caso este esteja vazio (sem dados para enviar nem receber) e tenha ocorrido uma interrupção do teclado, envia o respetivo *scancode* para o computador de destino.

Quando o utilizador entra no modo *multiplayer* em série, o jogo fica em estado de espera (é apresentada a mensagem “*WAITING FOR PLAYERS...”* até receber um pacote específico do computador ao qual está ligado. O computador do adversário, por sua vez, também envia o mesmo pacote enquanto não receber uma resposta. Os pacotes são enviados a um período constante, a cada 0.5 segundos. Asosm, os jogadores não precisam de entrar simultâneamente nomodo *multiplayer*em série e um dos jogadores pode entrar no jogo a qualquer momento.

A receção dos dados enviados é feita no destino através de interrupções geradas pela *UART*. O *handler* recebe a interrupção, verifica o tipo de interrupção (foram ativadas apenas interrupções do tipo *Reciever Data Available* e *Line Status Register*), e processa o *scancode* recebido. É chamada a função *paddle\_keyboard\_update()* para atualizar a posição da barra do adversário e as teclas especiais que pressionou (*Enter*, barra de espaços, *Esc*).

## 7.5 *Package* lpoo.proj2.net



# Conclusão

Embora tenham sido explicados os vários excertos de código em linguagem C fornecidos para serem utilizados nos laboratórios e no projeto final; estes excertos, na sua maioria "uma orientação para o que deveria ser feito" e por vezes bastante genéricos, foram abordados de um ponto de vista pouco prático.

Em consequência, deparamo-nos com a dificuldade de passar da "teoria à prática", não por não se conhecer a matéria, mas porque mesmo acompanhados dos diapositivos das aulas teóricas, dos apontamentos aí tirados e dos guiões das aulas laboratoriais não se sabia por onde começar a escrita do código (por nunca ter havido um verdadeiro contacto com a parte prática).

Assim sendo, sugerimos que o professor construa com os alunos pequenos excertos de código na aula e que os execute, podendo até mostrar propositadamente formas incorretas de escrever o código e explorar as consequências que daí advêm. A oportunidade de contactar com esta componente mais prática seria uma grande ajuda na realização dos trabalhos práticos.