

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
2.	Manual (aplicação <i>desktop</i>).....	4
2.1	Menu inicial	4
2.2	Menu “ <i>PREFERENCES</i> ”	5
2.3	Menu “ <i>SINGLEPLAYER</i> ”	7
2.4	Menu “ <i>MULTIPLAYER</i> ”	8
2.5	Ecrã de jogo	10
3.	Manual (aplicação <i>Android</i>).....	13
4.	Concepção e implementação.....	17
4.1	Bibliotecas, tecnologias e ferramentas.....	17
4.2	Padrões de desenho	18
4.3	Testes unitários.....	20
4.4	Mecanismos de comunicação.....	21
5.	Diagrama de casos de utilização.....	22
6.	Diagrama de pacotes (<i>packages</i>).....	23
7.	Diagrama de classes.....	25
7.1	<i>Package</i> lpoo.proj2.....	25
7.2	<i>Package</i> lpoo.proj2.audio.....	26
7.3	<i>Package</i> lpoo.proj2.gui	27
7.4	<i>Package</i> lpoo.proj2.logic.....	28
7.5	<i>Package</i> lpoo.proj2.net.....	29
8.	Conclusão.....	30

1. Introdução

Neste relatório será explicado o funcionamento do jogo *Air Hockey* desenvolvido no âmbito da cadeira de Laboratório de Programação Orientado por Objectos (LPOO), bem como a estruturação que fora necessária ao seu desenvolvimento.

O *Air Hockey* é uma versão virtual do clássico jogo de hóquei de mesa no qual dois jogadores se defrontam, tentando inserir um *puck* (disco) na baliza adversária, controlando apenas um *paddle* (barra) que usam para defletir o *puck* bem como para defender a própria baliza. É de notar que a superfície de jogo é bastante polida, pelo que o *puck* pode atingir velocidades alucinantes, complicando-se o ataque e a defesa.

O *Air Hockey* é constituído por duas aplicações, uma *desktop* e uma *android*.

- **aplicação *desktop*:** a primeira aplicação, que trabalha independentemente da segunda, contém a componente gráfica principal, sendo responsável por mostrar a área de jogo e gerir os eventos que vão ocorrendo ao longo de uma partida. É responsável ainda por carregar as definições e definir as regras de uma dada partida, bem como de criar as condições necessárias para o funcionamento do modo *multiplayer*. Suporta ainda um modo de jogo *singleplayer*, em que um jogador pode confrontar o computador, para as situações em que não estiver por perto um adversário humano.
- **aplicação *Android*:** A segunda aplicação trabalha em conjunto com a primeira e é responsável por permitir a conexão de dois jogadores a um servidor por forma a poderem confrontar-se num jogo *multiplayer*, permitindo a cada utilizar controlar o respetivo *paddle*, no próprio visor do seu *smartphone*, bem como configurar o estilo do seu *paddle* e a possibilidade de este escolher um *username*.

2. Manual (aplicação *desktop*)

2.1 Menu inicial



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

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



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



✓ Botão “**PREFERENCES**” - abre o menu *Preferences*.



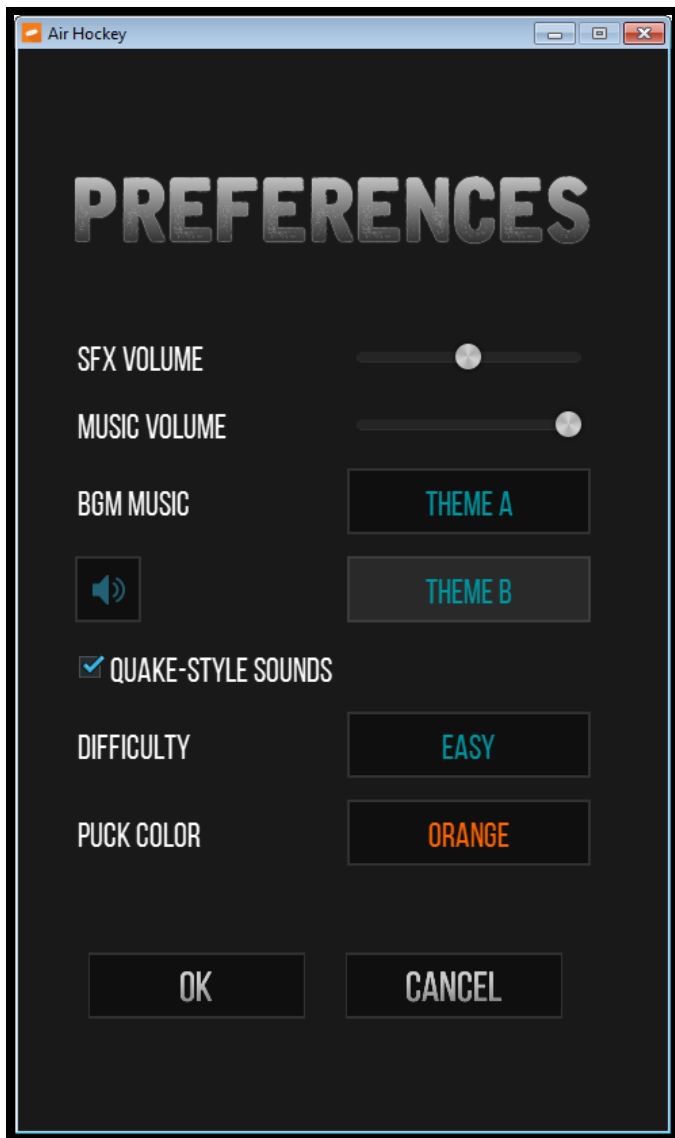
✓ Botão “**CREDITS**” - permite visualizar os créditos do jogo.



✓ Botão “**EXIT**” - encerra a aplicação quando pressionado.



2.2 Menu “PREFERENCES”



No menu *Preferences* o utilizador pode alterar as definições da partida e do jogo, como o volume da música de fundo e efeitos sonoros, a própria música de fundo, a cor do *puck* e a dificuldade do adversário controlado por computador no modo *singleplayer*. As definições são guardadas num ficheiro, sendo este carregado sempre que a aplicação é iniciada.

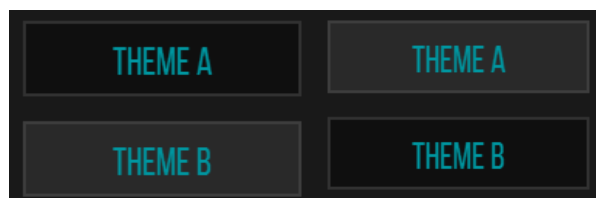
✓ Barra “**SFX VOLUME**” - permite alterar o volume dos efeitos sonoros e das vozes escutados durante as partidas, arrastando o cursor para os lados.



✓ Barra “**MUSIC VOLUME**” - permite alterar o volume da música de fundo escutado durante a navegação nos menus principais e durante o jogo, arrastando o cursor para os lados.



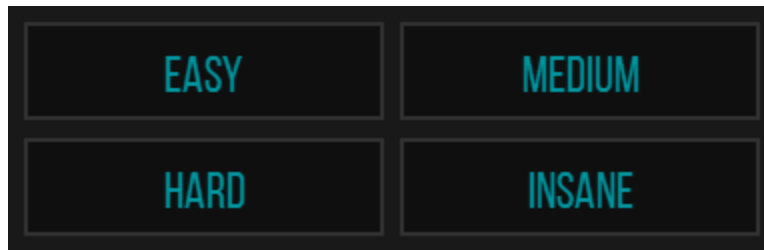
Botão “**BGM MUSIC**” - permite escolher o tema da música de fundo. O respetivo botão do tema escolhido apresentará uma cor cinza.



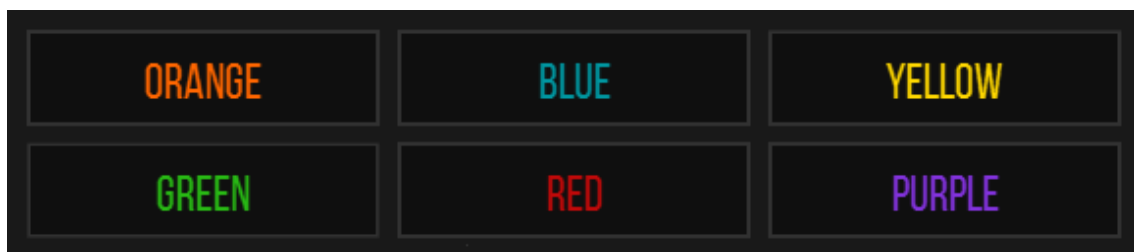
Checkbox “**QUAKE-STYLE SOUNDS**” - ativa/desativa as vozes e efeitos sonoros baseados no jogo *Quake*, que podem ser escutados quer no início das partidas, quer em situações de golo.



- ✓ **Botão “DIFFICULTY”** - permite seleccionar a dificuldade do adversário controlado pelo computador no modo *singleplayer*. Estão disponíveis quatro níveis de dificuldade distintos que influenciam o tempo de reação do adversário controlado artificialmente.



- ✓ **Botão “PUCK COLOR”** - permite seleccionar o cor do *puck*. Estão disponíveis seis cores distintas, sendo a cor do texto no botão a mesma cor que será aplicada ao *puck*.



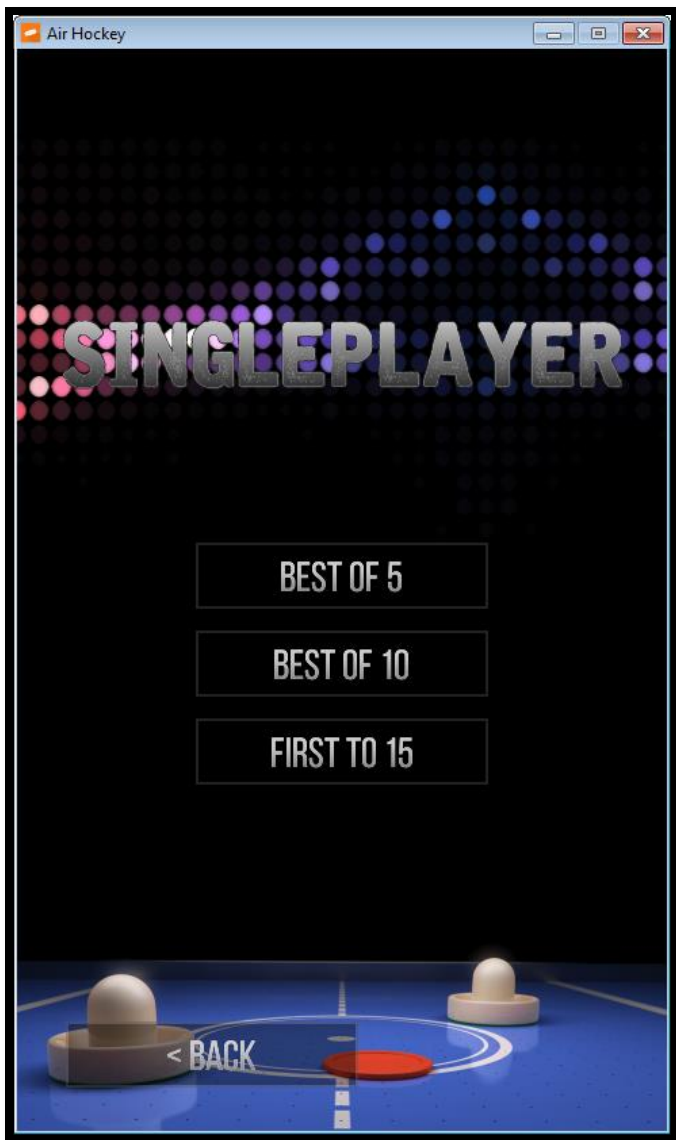
- ✓ **Botão “OK”** - guarda as preferências seleccionadas para futuras partidas e regressa ao menu inicial.



- ✓ **Botão “CANCEL”** - regressa ao menu inicial sem guardar as preferências seleccionadas, ficando em vigor as preferências previamente guardadas.



2.3 Menu “SINGLEPLAYER”



O menu do modo *singleplayer* permite iniciar uma nova partida no modo *singleplayer* (um único jogador). Existem três tipos de jogo que podem ser escolhidos em *singleplayer*. Em todos eles, o utilizador terá de enfrentar um adversário controlado pelo computador, que apresentará uma maior/menor velocidade de resposta às jogadas do utilizador consoante a dificuldade escolhida.

✓ Botão “**BEST OF 5**” - inicia uma nova partida do tipo “melhor de cinco”. Vence o “jogador” que possuir uma maior pontuação ao fim de cinco partidas.



✓ Botão “**BEST OF 10**” - inicia uma nova partida do tipo “melhor de dez”. Vence o “jogador” que possuir uma maior pontuação ao fim de dez partidas.



✓ Botão “**FIRST TO 15**” - inicia uma nova partida do tipo “primeiro a chegar aos quinze pontos”. Vence o “jogador” que conseguir obter em primeiro lugar quinze pontos.

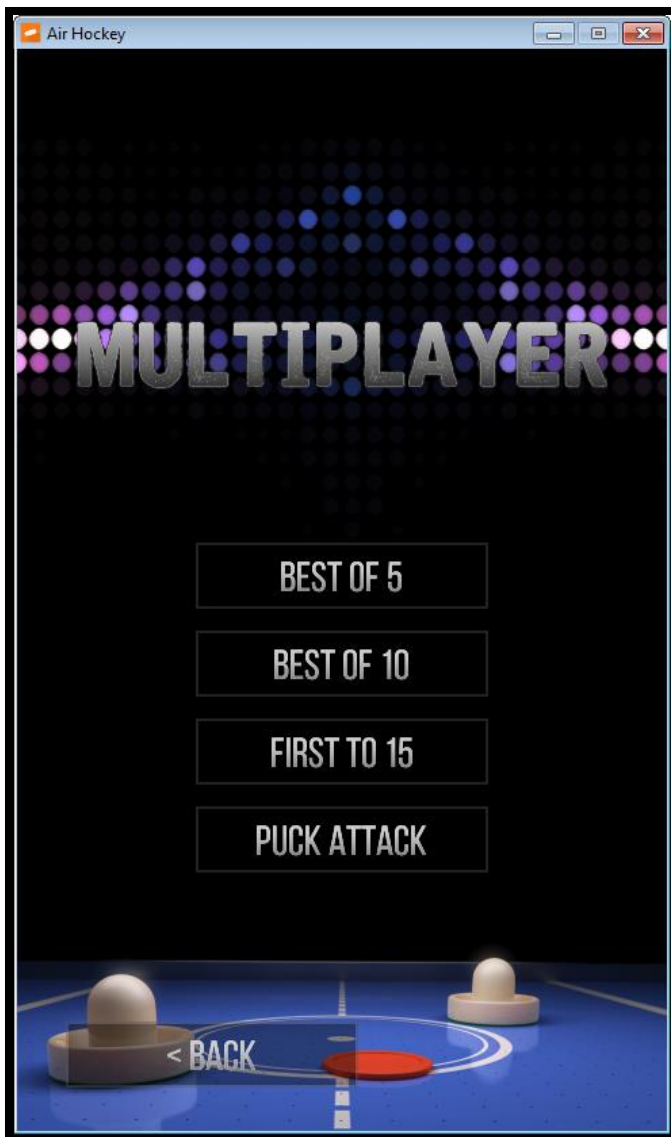


✓ Botão “< BACK” - permite sair do menu *singleplayer*, regressando ao menu inicial.



Após seleccionado o modo de jogo será aberto o ecrã de jogo.

2.4 Menu “MULTIPLAYER”



O menu do modo *multiplayer* permite iniciar uma nova partida no modo *multiplayer*. Existem quatro tipos de jogo que podem ser escolhidos para uma partida no modo *multiplayer*. Em todos eles, dois jogadores humanos defrontar-se-ão.

Botão “**BEST OF 5**” - inicia uma nova partida do tipo “melhor de cinco”. Igual ao modo *singleplayer*.



Botão “**BEST OF 10**” - inicia uma nova partida do tipo “melhor de dez”. Igual ao modo *singleplayer*.



Botão “**FIRST TO 15**” - inicia uma nova partida do tipo “primeiro a chegar aos quinze pontos”. Igual ao modo *singleplayer*.



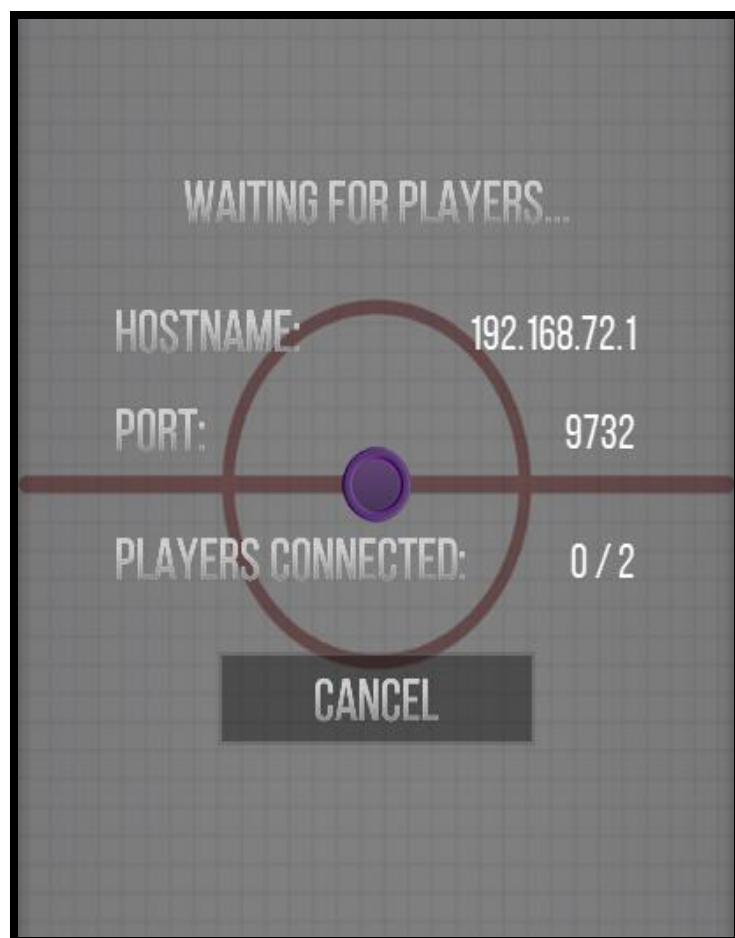
Botão “**PUCK ATTACK**” - inicia uma nova partida do tipo “Puck Attack”



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



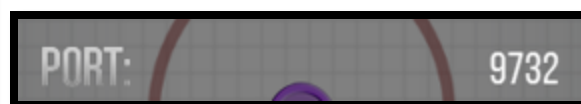
Após seleccionado o tipo de partida, é necessário que ambos os jogadores se conectem. É com este fim que aparece o seguinte ecrã. Aqui é possível encontrar a seguinte informação:



“HOSTNAME” - contém o endereço IP da máquina que servirá de anfitrião ao jogo.



“PORT” - contém a porta da máquina anfitriã a que os jogadores se devem conectar.



“PLAYERS CONNECTED” - contém o número de utilizadores já conectados naquele momento.



Botão “CANCEL” - cancela a procura de jogadores, fechando o servidor e regressando ao menu inicial.



Se ambos os jogadores se conectarem devidamente, será feita a transição para o ecrã de jogo.

2.5 Ecrã de jogo



Neste ecrã é possível visualizar a partida. O jogador (ou jogadores, no caso de uma partida *multiplayer*) terá de marcar golos na baliza do adversário. Para tal necessita de controlar um *paddle* para direccionar o disco em direcção à baliza do adversário, bem como para defender a sua própria baliza.

No ecrã de jogo é possível encontrar os seguintes elementos:

Paddle - é o único componente controlado exclusivamente pelo jogador em toda a partida e é com ele que deve proteger a baliza bem como rematar o *puck* à baliza adversária. Os *paddles* não podem entrar no campo do adversário.



No lado esquerdo de cada campo é possível encontrar uma etiqueta que indica de que lado joga o utilizador e o computador:



No caso de um jogo *multiplayer* existirão duas etiquetas para identificar o campo de cada um dos jogadores:



Puck - é um pequeno disco que está constantemente a ser defletido de um campo ao outro pelos jogadores. Sempre que o *puck* entra numa baliza começa uma nova ronda, sendo atribuída um ponto ao jogador marcador. O *puck* perde energia quando colide com as paredes da área de jogo, podendo no entanto adquirir grandes velocidades ao ser defletido pelos *paddles* dos jogadores.



- ✓ **Baliza** - localizada ao centro das paredes horizontais do campo, são os locais para onde se pretende enviar/desviar o *puck*, marcando golo.



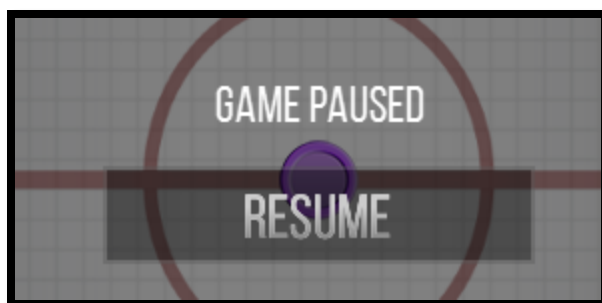
Sempre que se verifica uma situação de golo é mostrada uma mensagem com o nome do marcador e com a pontuação atualizada do jogo.



Quando um jogo termina é mostrado no ecrã uma mensagem semelhante, em que o nome do vencedor é mostrado, bem como a pontuação final da partida.



É possível colocar uma partida em pausa sempre que necessário, sendo mostrado ao utilizador a seguinte janela. Para regressar à partida basta pressionar o botão “*RESUME*”.



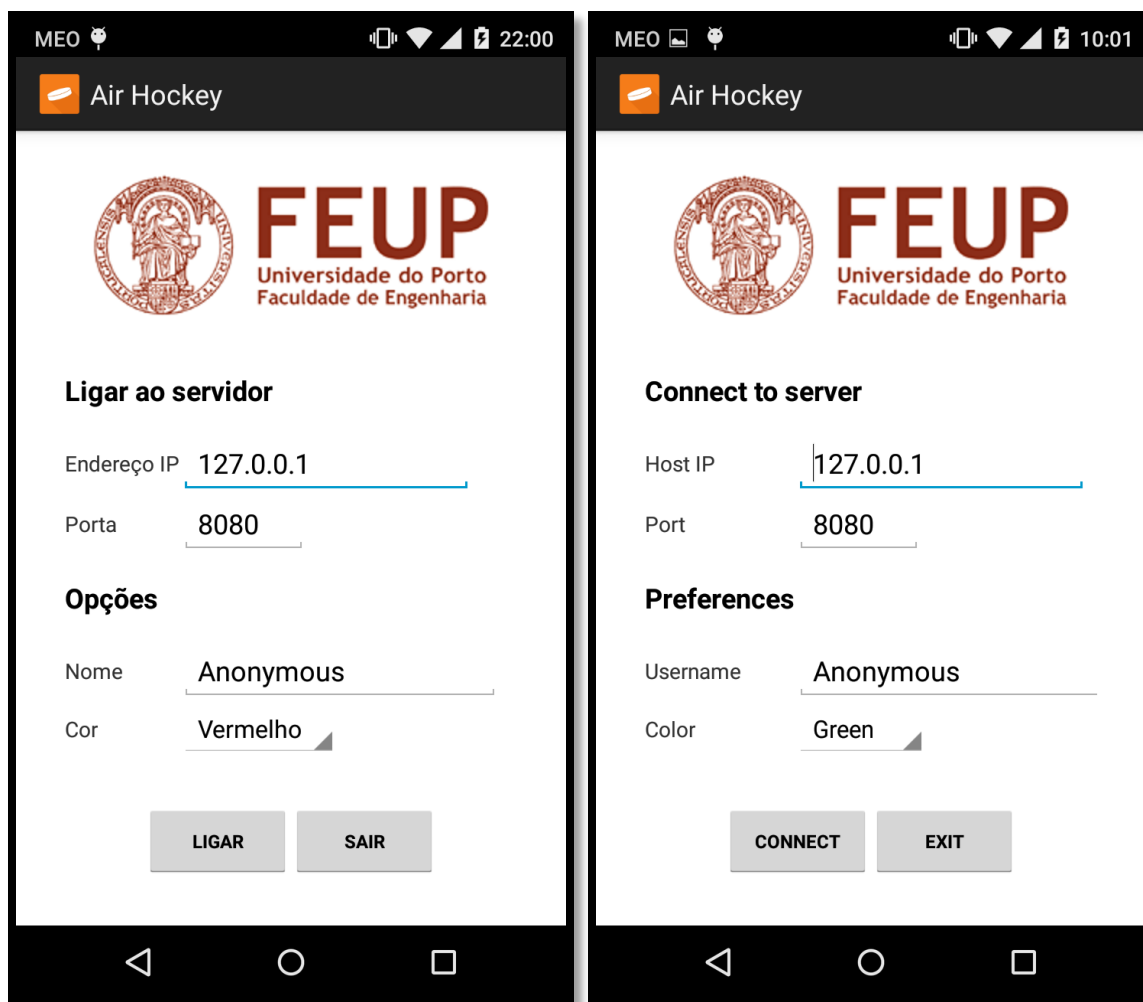
Caso um jogador não pretenda continuar uma dada partida, é possível abortar a partida. Para tal basta carregar na tecla *ESC*, aparecendo a seguinte mensagem:



Carregando no botão “YES” a partida é abortada, regressando ao menu inicial. Carregando no botão “NO”

3. Manual (aplicação *Android*)

O menu inicial da aplicação *Android* do Air Hockey surge quando se inicia a aplicação. Através dele o utilizador pode conectar-se a uma máquina anfitriã (servidor) de forma a participar numa partida *multiplayer*.



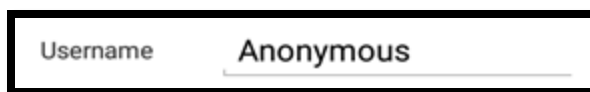
- **Host IP / Endereço IP** - serve para introduzir o endereço local/*internet* onde a máquina anfitriã se encontra a correr

Endereço IP 127.0.0.1

- **Port / Porta** - serve para introduzir a porta TCP onde o servidor se encontra a correr na máquina anfitriã

Porta 8080

- **Username / Nome** - serve para introduzir o nome ou alcunha que o jogador irá possuir durante a partida

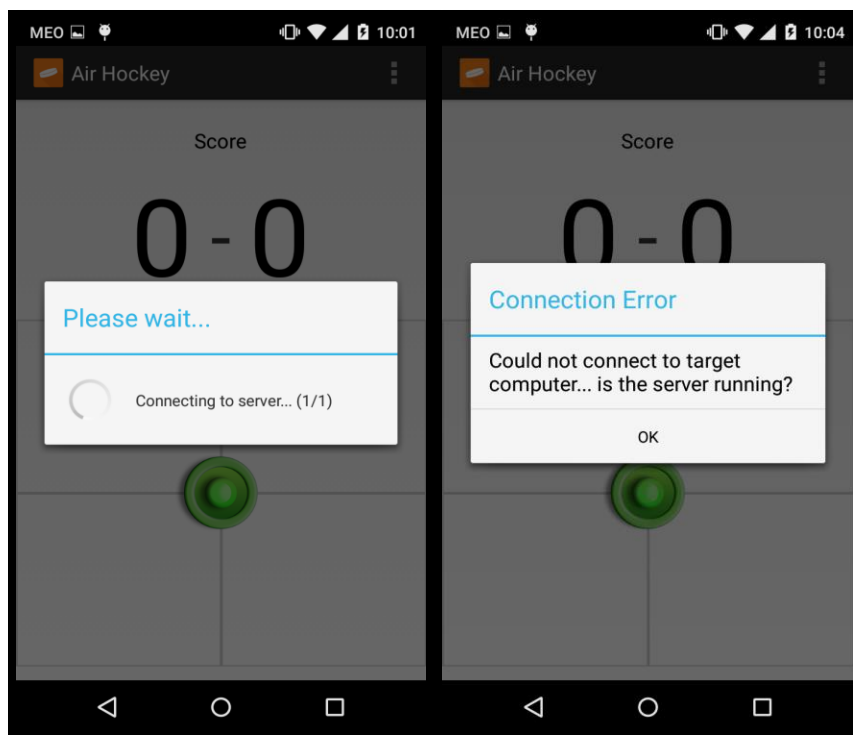


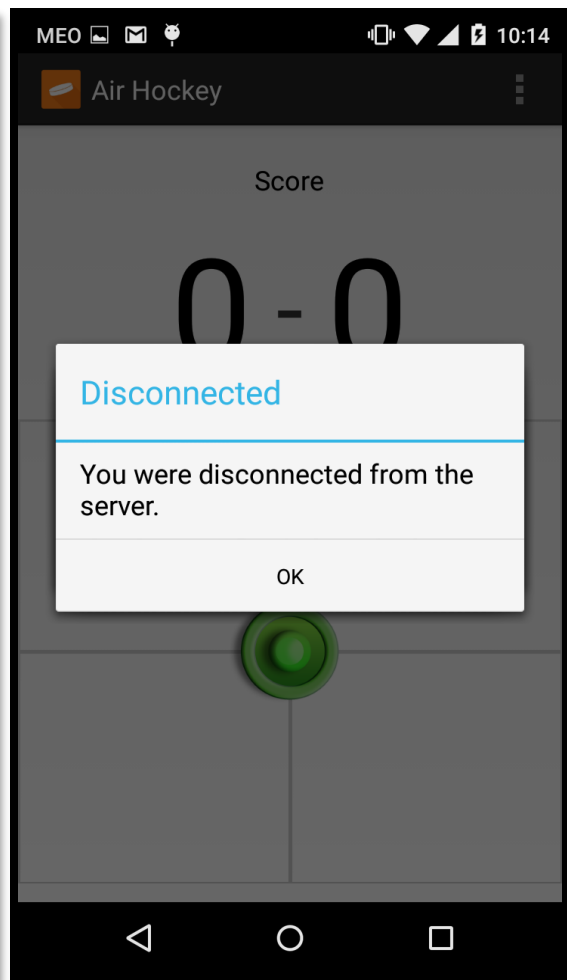
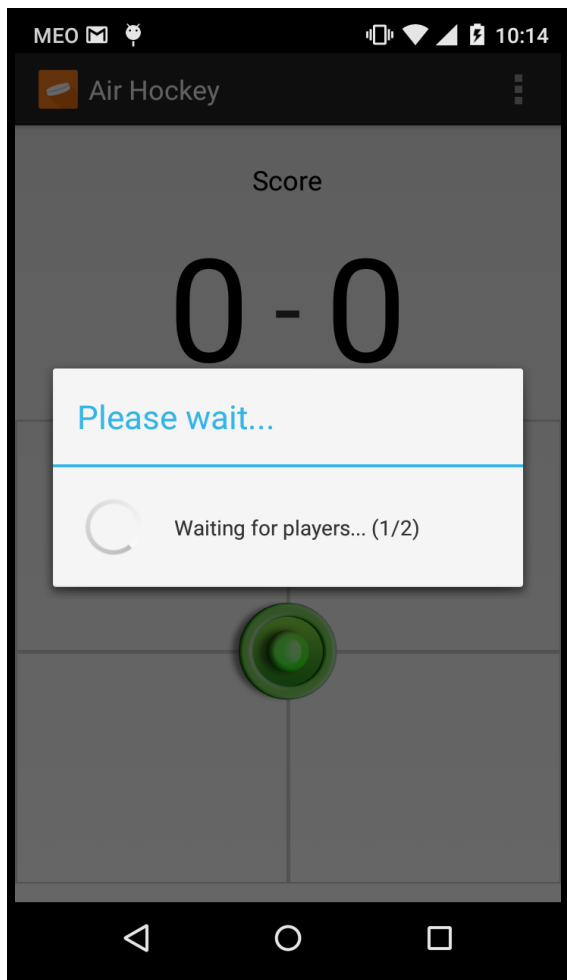
- ✓ **Color / Cor** - serve para escolher a cor do *paddle* do utilizador

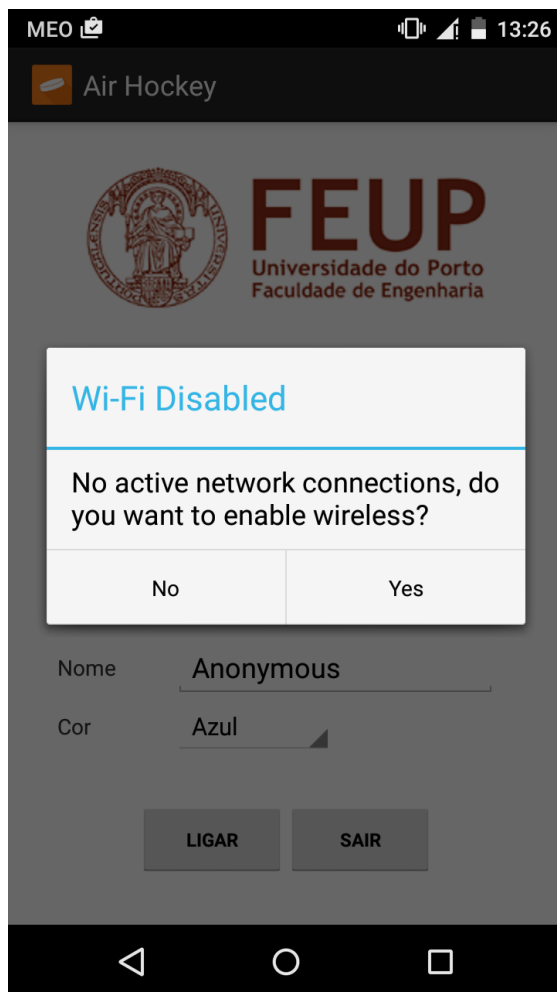


É possível escolher uma das seis cores disponíveis para o *paddle* do utilizador: vermelho (*red*), azul (*blue*), amarelo (*yellow*), verde (*green*), laranja (*orange*) e violeta (*purple*).

Ao carregar no botão “Ligar” é criada uma nova *activity* na aplicação do cliente que representa o campo do jogador. Se uma tentativa de ligação falhar será 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)

4. Conceção 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 *non-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 distribuição *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	<p><i>“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.”</i></p> <p>onde: classe <i>AudioManager</i></p> <p>porquê: encapsulamento de um sistema complexo (o gestor de áudio da aplicação e os seus subsistemas de gestão de ficheiros de áudio, controlo de volume, controlo de reprodução) numa classe que através da implementação de métodos mais simples (<i>setSFXVolume</i>, <i>setMusicVolume</i>, <i>playSong</i>, <i>playSound</i>) facilita a interacção entre o programador/utilizador e o sistema complexo em questão</p>
Observer	<p><i>“Define a one-to-many dependency between objects so that when one object changes state, all its dependents are notified and updated automatically.”</i></p> <p>onde: classes <i>ServerListener</i>, <i>ClientListener</i> implementadas pela biblioteca <i>Kryonet</i> para a comunicação em rede entre servidor e cliente, <i>ClickListener</i> e <i>InputProcessor</i> implementadas pela biblioteca <i>LibGDX</i> para processamento de eventos de <i>input</i> do rato e teclado, <i>MenuListener</i> como classe derivada de <i>ClickListener</i></p> <p>porquê: ...</p>
Singleton	<p><i>“Ensure a class has only one instance, and provide a global point of access to it.”</i></p> <p>onde: classe <i>AudioManager</i></p> <p>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</p>
State	<p><i>“Allow an object to alter its behavior when its internal state changes. The object will appear to change its class.”</i></p> <p>onde: superclasse <i>GameState</i> e todas as suas classes derivadas (<i>GameRunningState</i>, <i>GamePausedState</i>, <i>GameOverState</i>, <i>PlayerScoredState</i>, <i>WaitingState</i>, <i>ConfirmExitState</i>, <i>DisconnectedState</i>)</p> <p>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 <i>input</i> do utilizador</p>

<p>Template Method</p>	<p><i>“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.”</i></p> <p>onde: superclasse <i>GameRules</i> e todas as suas classes derivadas (<i>RulesBest5</i>, <i>RulesBest10</i>, <i>RulesFirst15</i> e <i>RulesAttack</i>), que implementam os métodos em questão</p> <p>porquê: subclasses partilham parte dos métodos e atributos, no entanto existem diferenças em três nos métodos implementados <i>checkOver()</i>, <i>checkLast()</i> e <i>checkTie()</i> 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)</p>
<p>Visitor</p>	<p><i>“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.”</i></p> <p>onde: interface <i>CollisionDetector</i>, implementada pela classe abstracta <i>DynamicEntity</i></p> <p>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</p>

4.3 Testes unitários

Como se trata de um jogo maioritariamente 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 receção 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)

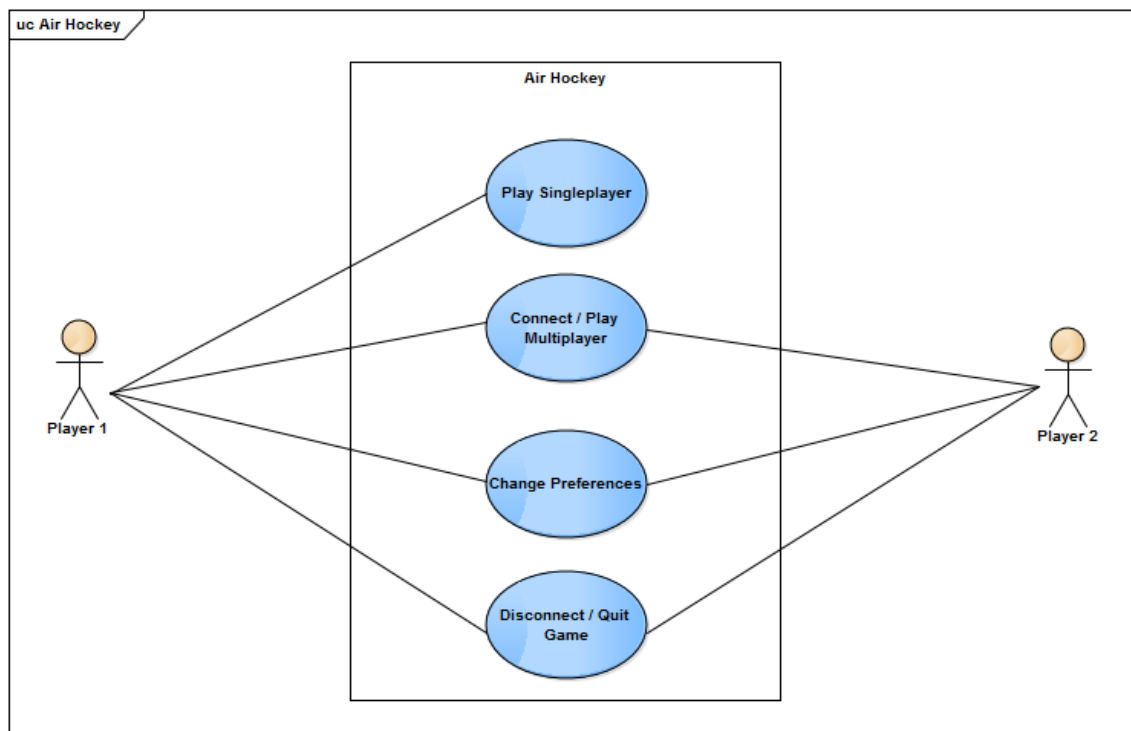
4.4 Mecanismos de comunicação

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.

5. Diagrama de casos de utilização



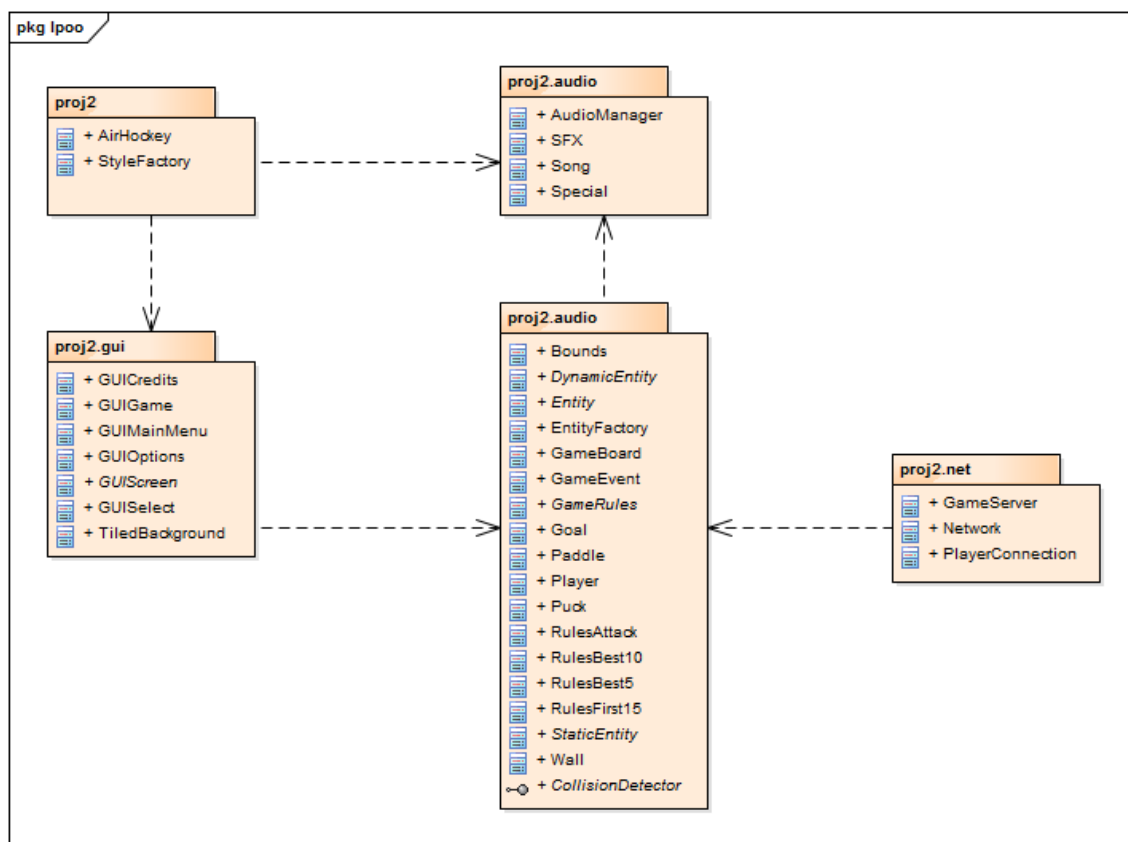
Player 1

- ✓ 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 do disco)
- ✓ 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*, desligando-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*, desligando-se do servidor

6. Diagrama de pacotes (*packages*)

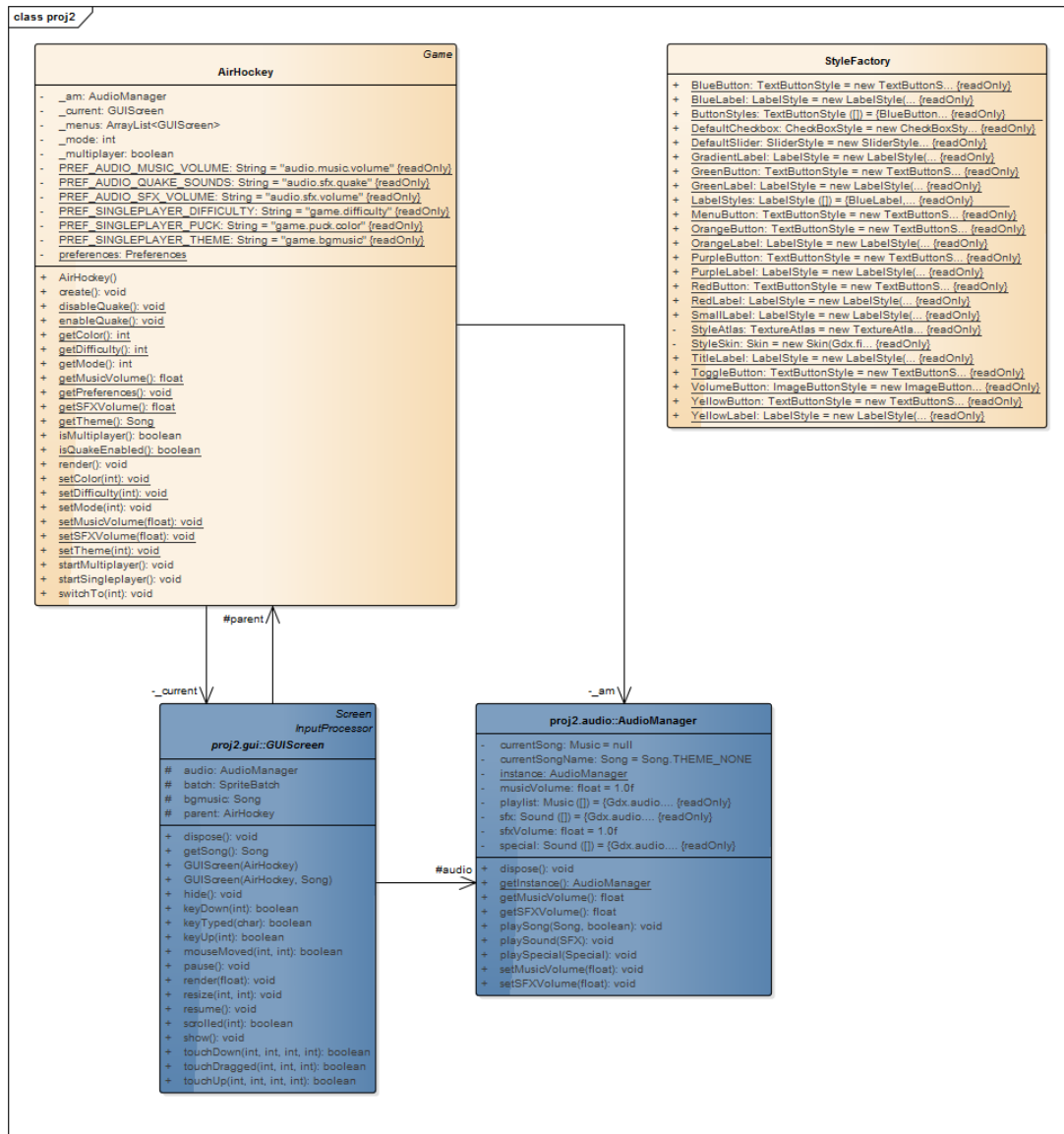


Package	Descrição
<i>proj2</i>	<ul style="list-style-type: none"> camada principal (topo hierárquico) da aplicação, constituída por apenas duas classes, <i>AirHockey</i> e <i>StyleFactory</i> a classe <i>AirHockey</i> contém o gestor de ecrãs, responsável pela transição entre ecrãs e menus da aplicação, 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 (<i>singleplayer</i> ou <i>multiplayer</i>) e da dificuldade do adversário no modo <i>singleplayer</i> a classe <i>StyleFactory</i> armazena e carrega para a memória vários <i>assets</i> e estilos visuais de forma a serem utilizados posteriormente pelas classes da camada gráfica com tempo de acesso mínimo
<i>proj2.audio</i>	<ul style="list-style-type: none"> 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 <i>assets</i> de áudio da aplicação, de forma a serem utilizados posteriormente com tempo de acesso mínimo constituído por uma classe (<i>AudioManager</i>), contendo a lógica de controlo e três enumeráveis (<i>SFX</i>, <i>Song</i> e <i>Special</i>) que

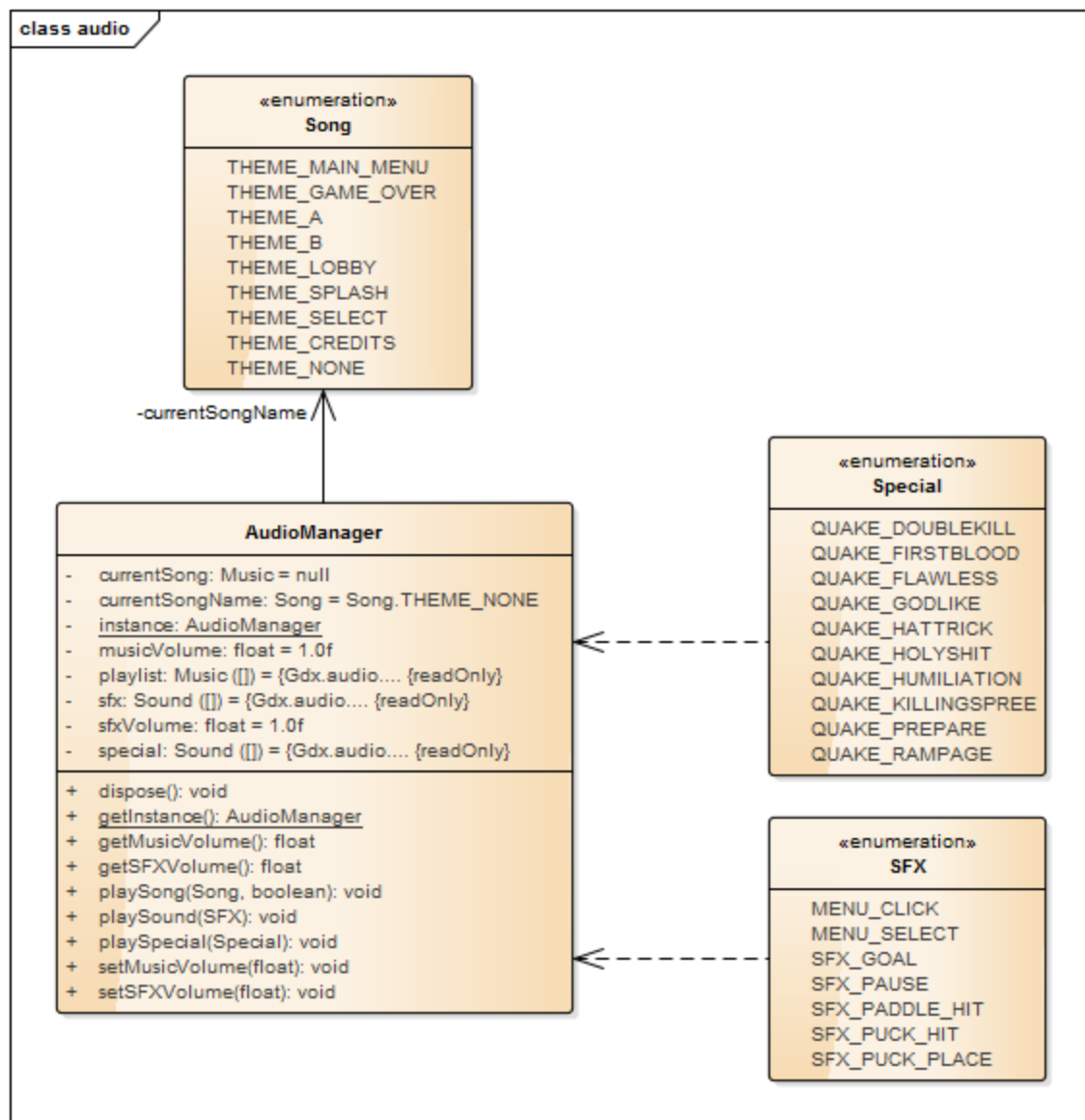
	<p>identificam o ficheiro de som a ser reproduzido e representam, respetivamente, efeitos sonoros, música de fundo e voz</p> <ul style="list-style-type: none"> ▪ existe ainda a possibilidade de aumentar/diminuir o volume da música de fundo e dos efeitos sonoros de forma independente e de desativar a voz, sendo esta substituída por outros efeitos sonoros
proj2.gui	<ul style="list-style-type: none"> ▪ contém toda a camada da <i>interface</i> gráfica da aplicação, isto é, tudo o que possa ser apresentado no ecrã ao utilizador, bem como métodos e <i>listeners</i> 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 <i>TiledBackground</i> ▪ <i>TiledBackground</i> é uma componente gráfica dos menus (responsável pela criação de imagens de fundo com <i>scrolling</i> infinito), tendo sido também incluída na camada gráfica da aplicação
proj2.logic	<ul style="list-style-type: none"> • contém toda a camada lógica do jogo que permite a construção de um campo de jogo com paredes, balizas, <i>paddles</i> e um disco, bem como uma implementação das regras e modos de jogo disponíveis • a criação das <i>sprites</i> e a simulação da física é feita dentro das classes dos elementos do jogo respetivos • estabelece também uma ponte de comunicação entre a <i>interface</i> gráfica e a camada de rede, ao permitir que eventos do servidor sejam também notificados ao utilizador através da <i>interface</i>
proj2.net	<ul style="list-style-type: none"> ▪ camada de comunicação por rede que permite a criação de partidas <i>multiplayer</i> entre dois jogadores (clientes), que podem interagir com o servidor através de um telemóvel <i>smartphone</i> com sistema operativo <i>Android</i> (arquitetura cliente-servidor) ▪ constituída por apenas duas classes, <i>GameServer</i> e <i>Network</i>, sendo esta constituída por várias <i>inner classes</i> que representam mensagens utilizadas comunicação entre os clientes e o servidor e vice-versa ▪ a gestão dos clientes é feita pela classe <i>GameServer</i>

7. Diagrama de classes

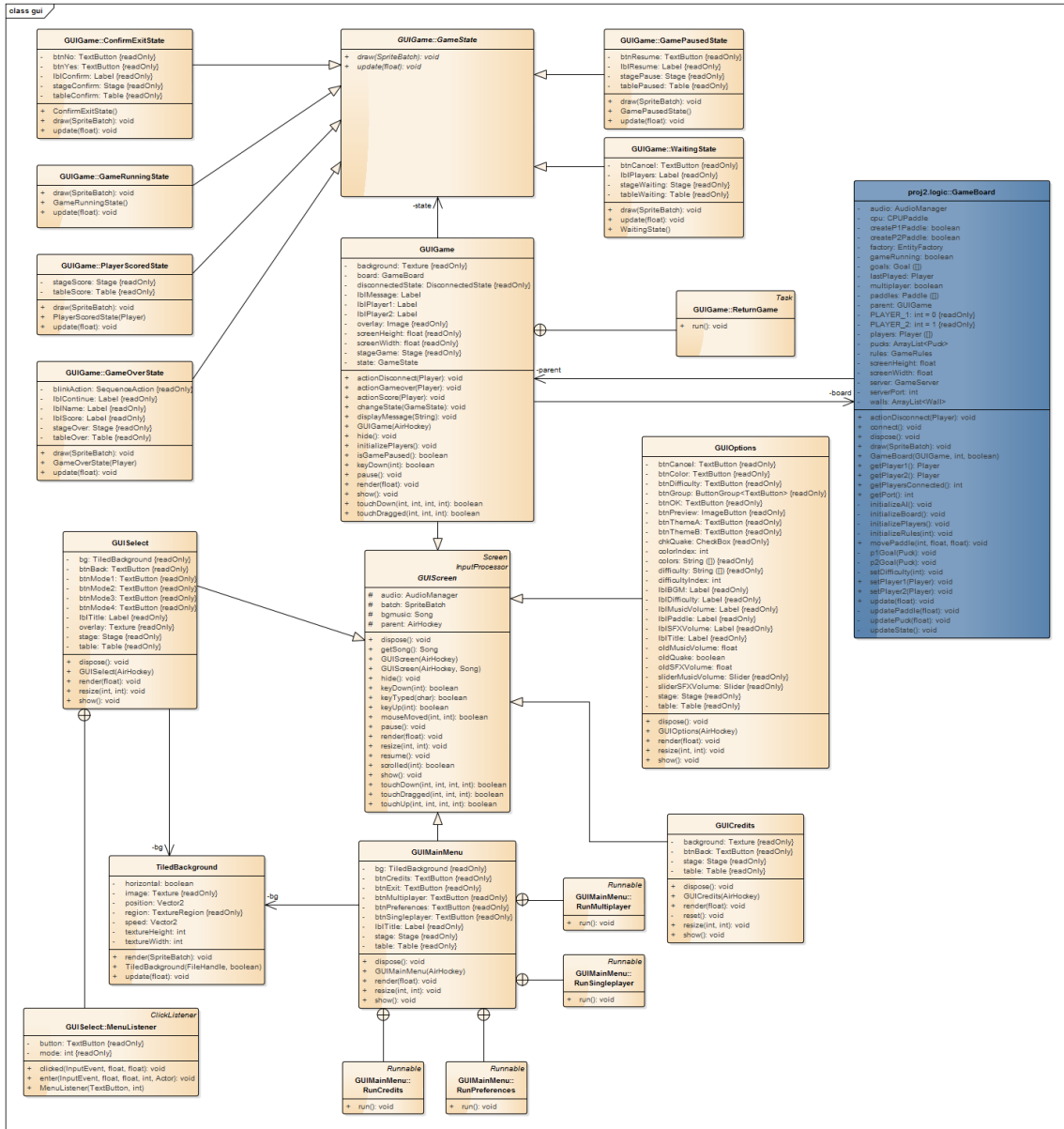
7.1 Package lpoo.proj2



7.2 Package lpoo.proj2.audio



7.3 Package lpoo.proj2.gui



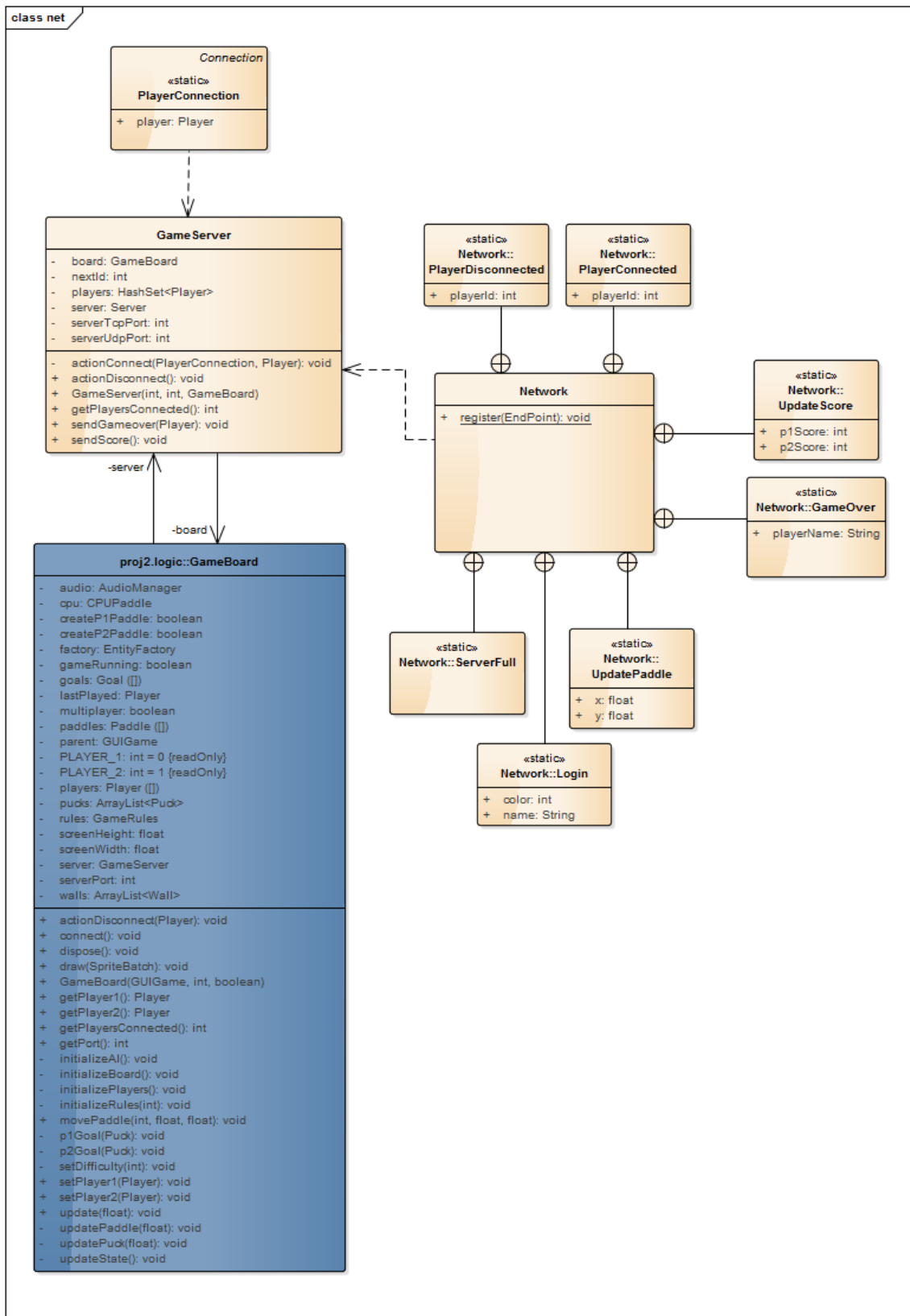
7.4 Package lpoo.proj2.logic

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. Assim, os jogadores não precisam de entrar simultaneamente no modo *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 *Receiver 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



8. 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.