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

Neste relatório será explicado o funcionamento do jogo *Air Hockey* desenvolvido no âmbito da Unidade Curricular de Laboratório de Programação Orientado por Objetos (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* 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 constituido por duas aplicações, uma *desktop* e uma *Android*:

- ✓ **aplicação *desktop*:** a primeira aplicação, que trabalha independendentemente 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 o utilizador 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*

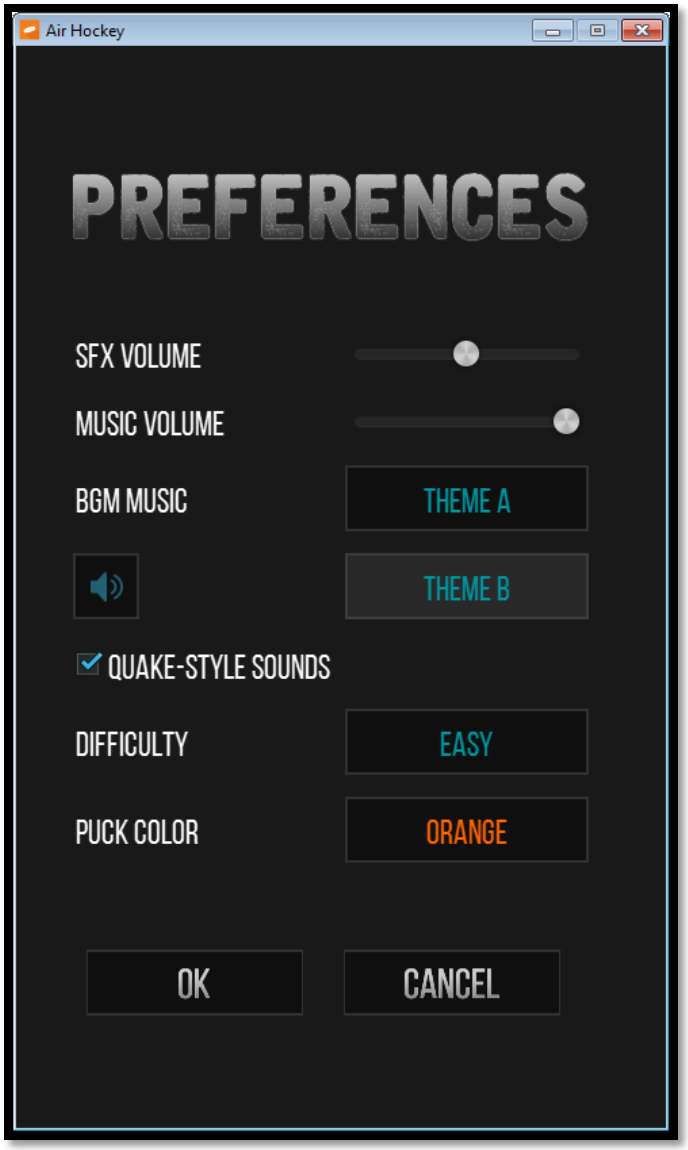
✓ **Botão *"MULTIPLAYER"*** - inicia uma nova partida no modo *multiplayer*

✓ **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"*

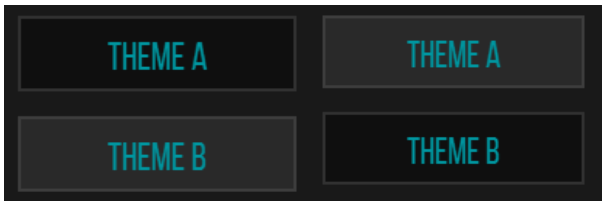


No menu *Preferences* o utilizador pode alterar as definições da aplicação e do modo *singleplayer*, como o volume da música de fundo e efeitos sonoros, a própria música de fundo, a cor do *puck* e a dificuldade do adversário controlado por computador no modo *singleplayer*. As definições são guardadas 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.

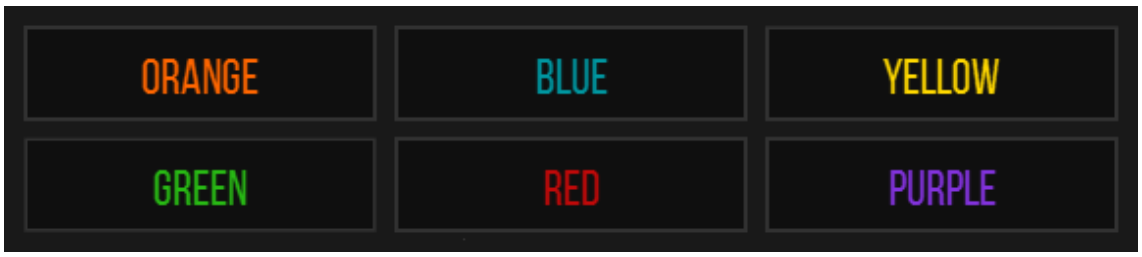
Botão "BGM MUSIC" - permite escolher o tema da música de fundo. O respetivo botão do tema escolhido apresentará uma cor cinza.



***Checkbox* "QUAKE-STYLE SOUNDS"** - ativa/desativa as vozes e efeitos sonoros adaptados do clássico *Quake*, que podem ser escutados quer no início das partidas, quer em situações de golo.

Botão *"DIFFICULTY"* – permite selecionar a dificuldade do adversário controlado pelo computador no modo *singleplayer*. Estão disponiveis quatro niveis de dificuldade distintos que influenciam o tempo de reação e a velocidade de movimento do adversário controlado artificialmente.

Botão *"PUCK COLOR"* – permite seleccionar o cor do *puck*. Estão disponiveis seis cores distintas, sendo a cor do texto no botão a mesma cor que será aplicada ao *puck*.



Botão *"OK"* – guarda as preferências atuais do utilizador de forma permanente para futuras partidas e regressa ao menu inicial.

Botão *"CANCEL"* – regressa ao menu inicial sem guardar as preferências atuais do utilizador, 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 ás 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 *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 defrontar-se-ão dois jogadores humanos.

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

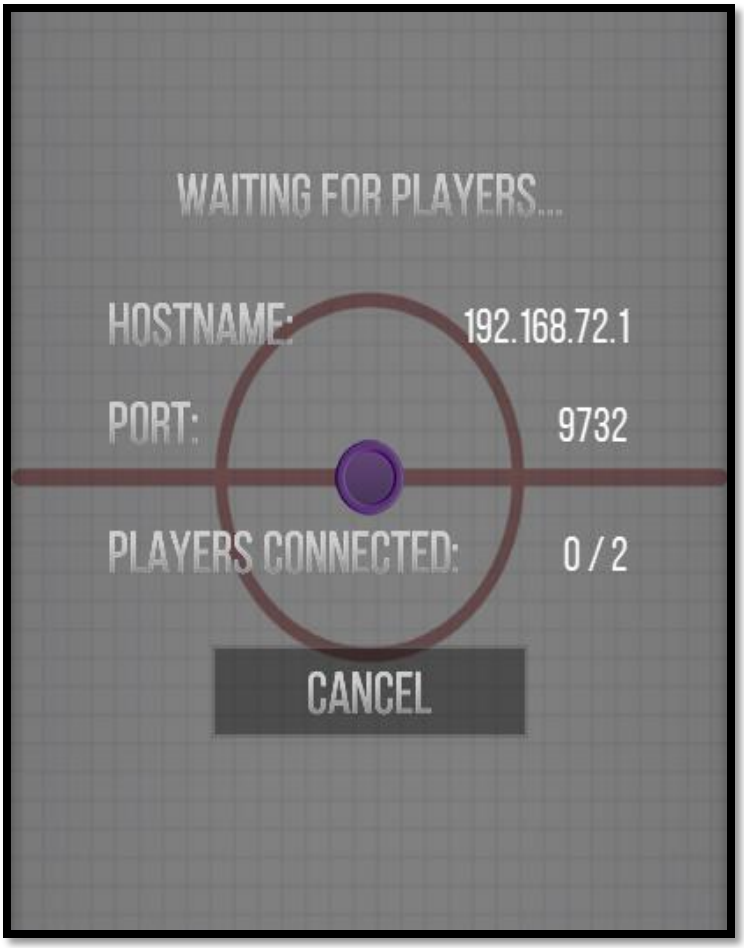
Botão *"BEST OF 10"* - inicia uma nova partida do tipo "melhor de dez". Igual ao modo *singleplayer*.

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

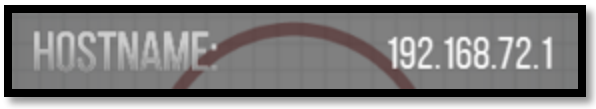
Botão *"PUCK ATTACK"* - inicia uma nova partida do tipo *"Puck Attack"*

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

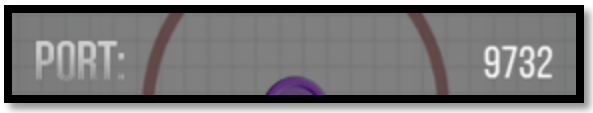
Após selecionado o tipo de partida, é necessário que ambos os jogadores se conectem. É com este fim que aparece o seguinte ecrã de espera:



"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" – representa o número de jogadores já conectados ao servidor naquele momento, bem como o respetivo número máximo.

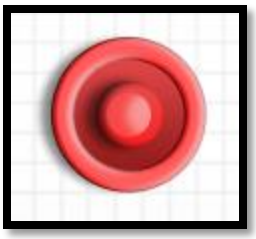
Botão *"CANCEL"* – cancela o processo de procura de jogadores, fechando o servidor e regressando ao menu inicial.

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 direcionar o disco em direcção à baliza do adversário, bem como para defender a sua própria baliza. No ecrã de jogo podem-se 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. O *paddle* de cada jogador não pode entrar no campo do adversário.



No lado esquerdo de cada campo é possível encontrar ainda 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 verifica-se uma situação de golo e começa uma nova ronda, sendo atribuída um ponto ao jogador marcador. O *puck* perde alguma 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 apresentada no ecrã uma mensagem semelhante, onde o nome no vencedor é mostrado, bem como a pontuação final obtida por ambos os jogadores que participaram nessa partida.

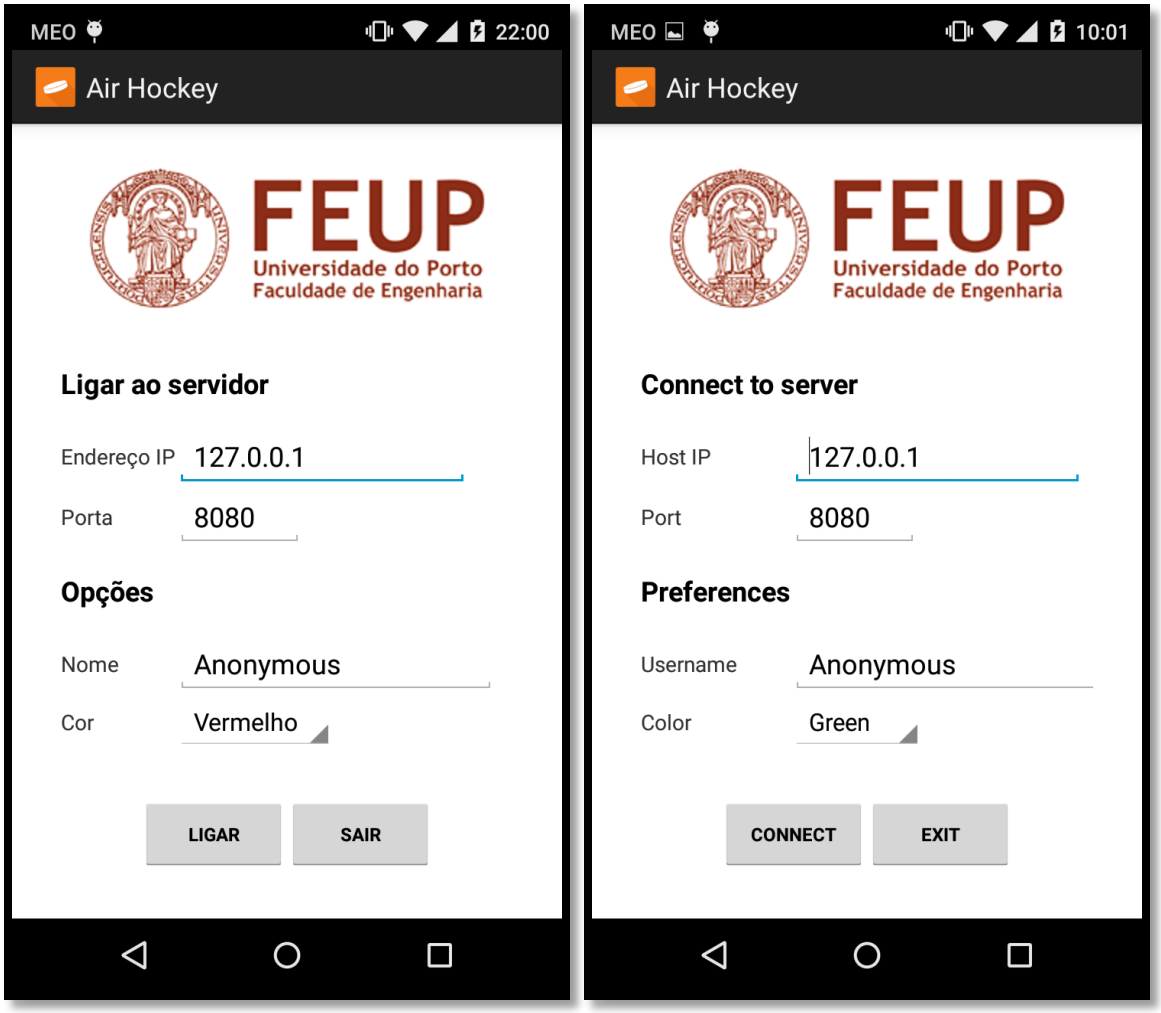
É possível colocar uma partida em pausa sempre que necessário, sendo mostrado ao utilizador a seguinte mensagem no ecrã. Para regressar à partida basta pressionar o botão "*RESUME*".

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"*, a partida continua.

3. Manual (aplicação *Android*)

Através do menu inicial da aplicação *Android* do *Air Hockey* 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* que identifica a máquina anfitriã na rede

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

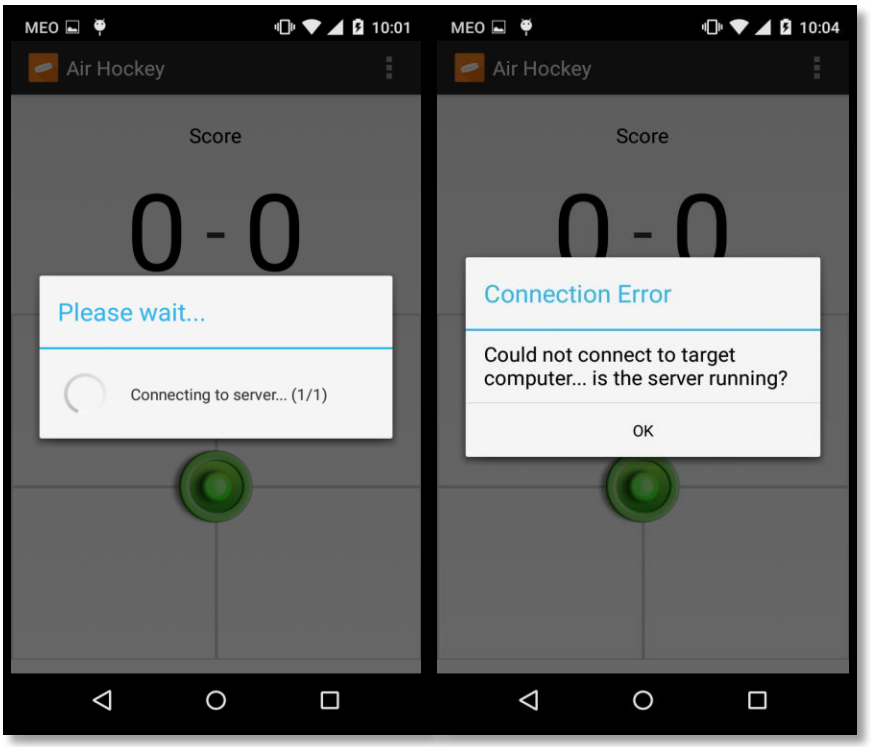
- ***Username / Nome*** – serve para introduzir o nome ou alcunha que o jogador irá tomar durante a partida

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

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

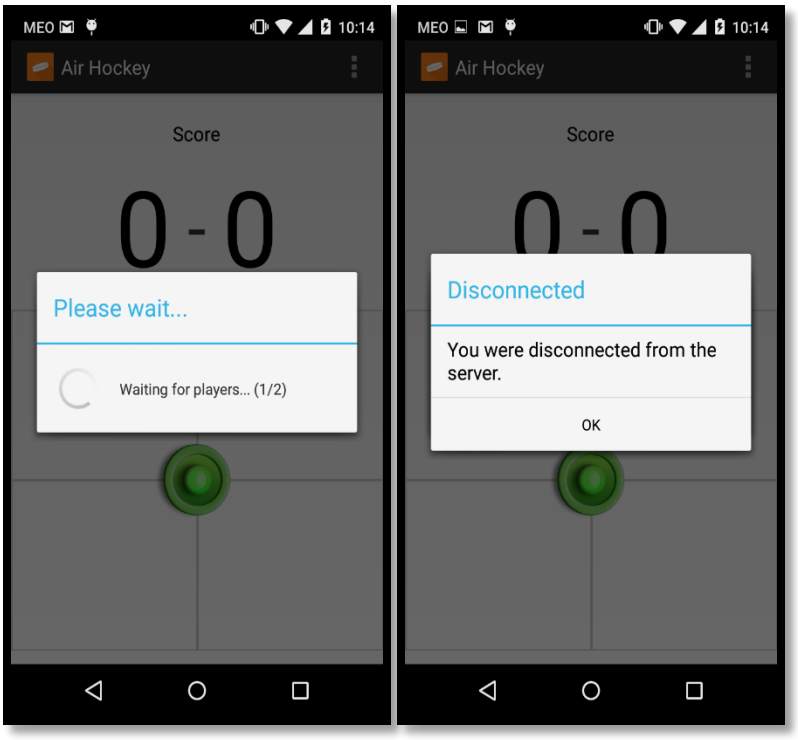
Ao carregar no botão *"Connect"* é criada uma nova *activity* na aplicação do cliente que representa o campo do jogador. Se essa tentativa de ligação falhar será apresentada uma mensagem ao utilizador para o informar de tal acontecimento.

Ao pressionar o botão "OK", o utilizador é encaminhado de volta ao menu inicial da aplicação. Como os dados que introduziu na *activity* principal são memorizados durante a transição para a *activity* do Game, o utilizador pode voltar a conectar-se ao servidor carregando simplesmente no botão "*Connect*".



O utilizador terá de esperar até que o seu adversário se conecte ao servidor. À semelhança do que acontecia na aplicação *desktop*, assim que ambos os jogadores se conectarem devidamente será feita a transição para a *activity* do jogo e dar-se-á início a uma nova partida no modo *multiplayer*.

Se durante esta partida a ligação ao servidor falhar ou o servidor for fechado subitamente, será mostrada a seguinte mensagem de erro, e os jogadores que já estivessem ligados ao servidor serão desconectados deste.



Dado ser necessário ter uma ligação do tipo *wireless* ativa para se conectar ao servidor, caso o dispositivo *Android* do utilizador tenha este tipo de ligação desligada, será apresentada a seguinte mensagem ao utilizador a pedir permissão para ativar e utilizar os serviços *wireless* do seu dispositivo móvel.

4. 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 objetos 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 sobre os 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 simples.

- ✓ **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 *display 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 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 interação entre o programador/utilizador e o sistema complexo 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:** classe *Listener* implementada 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:** criação de classes de baixo nível que "observam" interações do utilizador com a aplicação ou com um determinado dispositivo de entrada (rato, teclado ou placa de rede) e que comunicam às classes dependentes sempre que o estado dos mesmos é atualizado, para processarem então a informação recebida
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 de 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 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 ignoradas quando o utilizador cancelava 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 endereços IP inválidos e a combinações de 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/wireless 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 terminasse imediatamente 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 *display densities*:

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

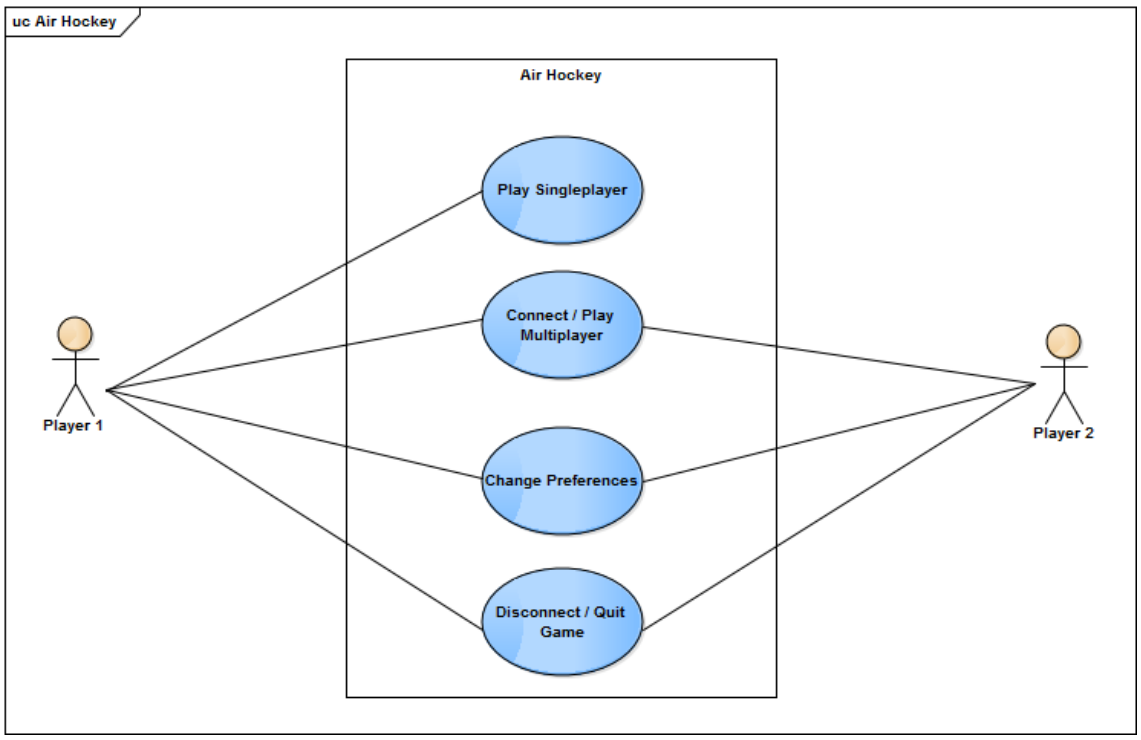
4.4 Mecanismos de comunicação

A comunicação em rede implementada consiste na troca de objetos entre cliente e servidor. Estes objetos são instâncias de classes relativamente simples e de tamanho reduzido, 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 ótimo em todas as redes móveis/Wi-Fi e na maior gama de 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 terminou e o nome do jogador que se sagrou o vencedor da mesma.
- **PlayerLogin** – utilizado pelo cliente para informar o servidor de que este se tentou conectar; é enviado ao servidor o nome/alcunha do jogador e a cor do *paddle* escolhida.
- **PlayerConnected** – utilizado pelo servidor para informar os restantes clientes de que um novo jogador entrou no servidor.
- **PlayerDisconnected** – utilizado pelo servidor para informar os restantes clientes de que um cliente se desligou do servidor, tendo 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 objeto 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 na aplicação *Android*; são enviadas as pontuações de ambos os jogadores (*p1Score* e *p2Score)* sempre que o servidor recebe este pedido.

5. Diagrama de casos de utilização



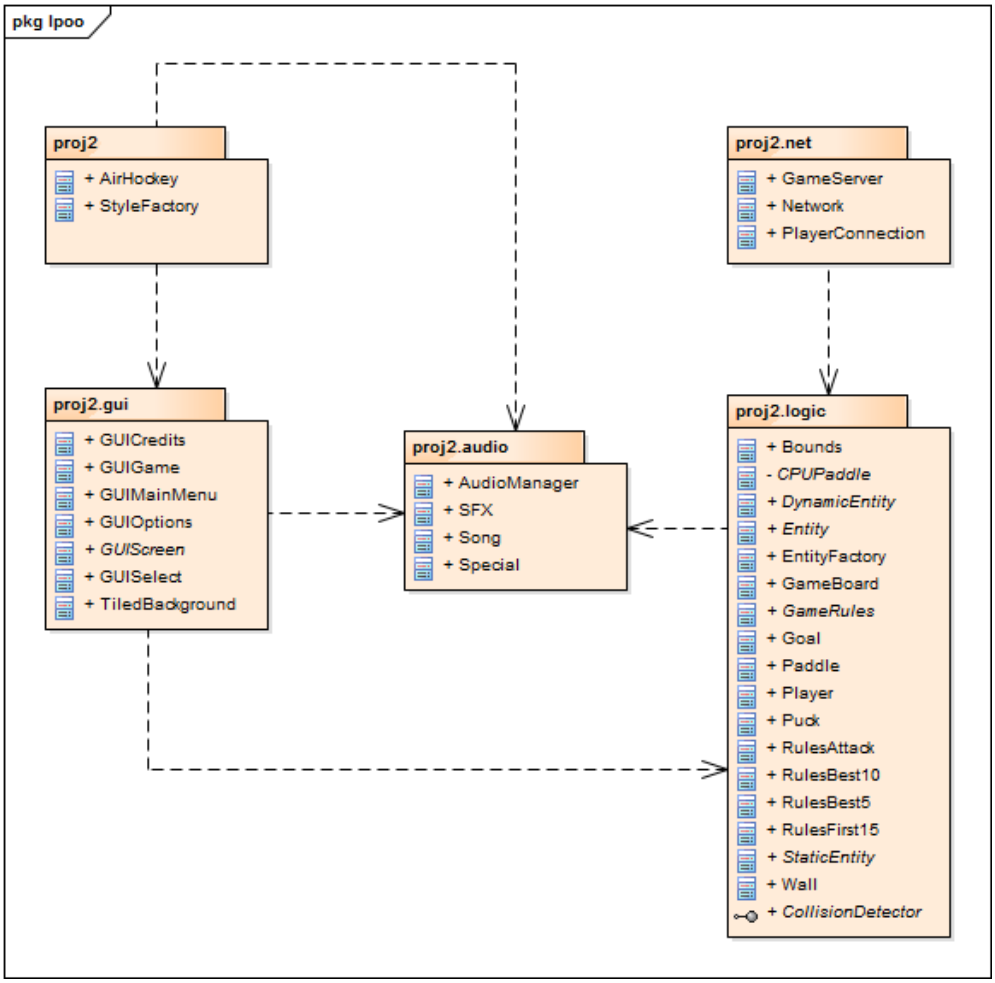
***Player* 1**

- ✓ (aplicação *desktop*) iniciar uma nova partida no modo *singleplayer*
- ✓ (aplicação *desktop*) iniciar uma nova partida no modo *multiplayer* (criar um servidor)
- ✓ (aplicação *Android)* entrar numa partida existente do modo *multiplayer*
- ✓ (aplicação *desktop*) alterar a configuração global (volume dos sons, volume da música, música de fundo, cor do disco)
- ✓ aplicação *desktop*) alterar a configuração do modo *singleplayer* (dificuldade do adversário)
- ✓ (aplicação *Android)* alterar o perfil do jogador (alcunha e cor do *paddle*)
- ✓ (aplicação *Android)* abandonar uma partida no modo *multiplayer*, desligando-se do servidor
- ✓ (aplicação *desktop*) encerrar a aplicação

***Player* 2**

- ✓ (aplicação *Android)* entrar numa partida existente do modo *multiplayer*
- ✓ (aplicação *Android)* alterar o perfil do jogador (alcunha e cor do *paddle*)
- ✓ (aplicação *Android)* abandonar uma partida no modo *multiplayer*, desligando-se do servidor

6. Diagrama de pacotes *(packages)*

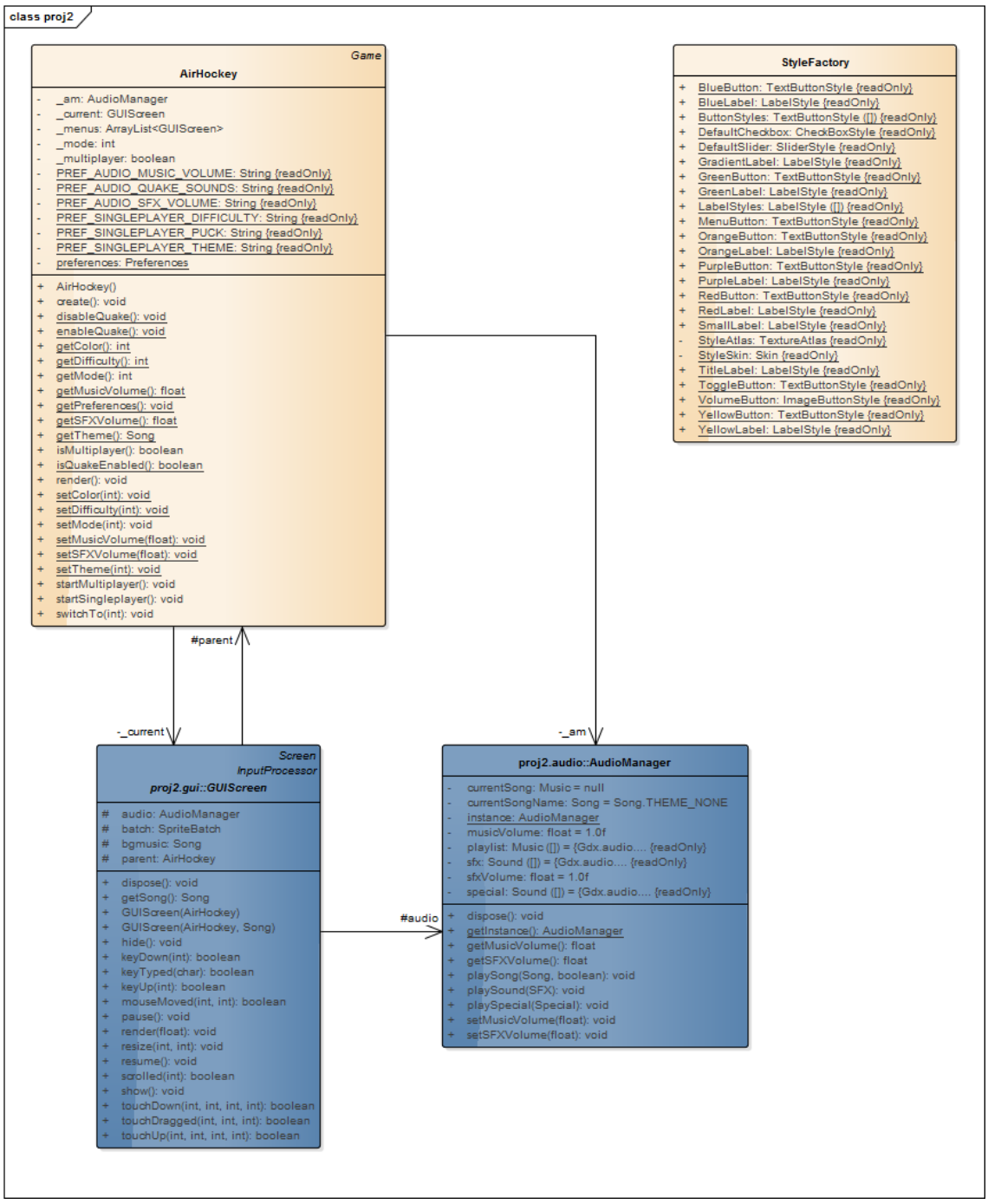


Package	**Descrição**
proj2	▪ camada principal (topo hierárquico) da aplicação, constituída por apenas duas classes, *AirHockey* e *StyleFactory* ▪ contém o gestor de ecrãs, responsável pela transição entre ecrãs e menus da aplicação, bem como métodos de serialização para carregamento e armazenamento permanente 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

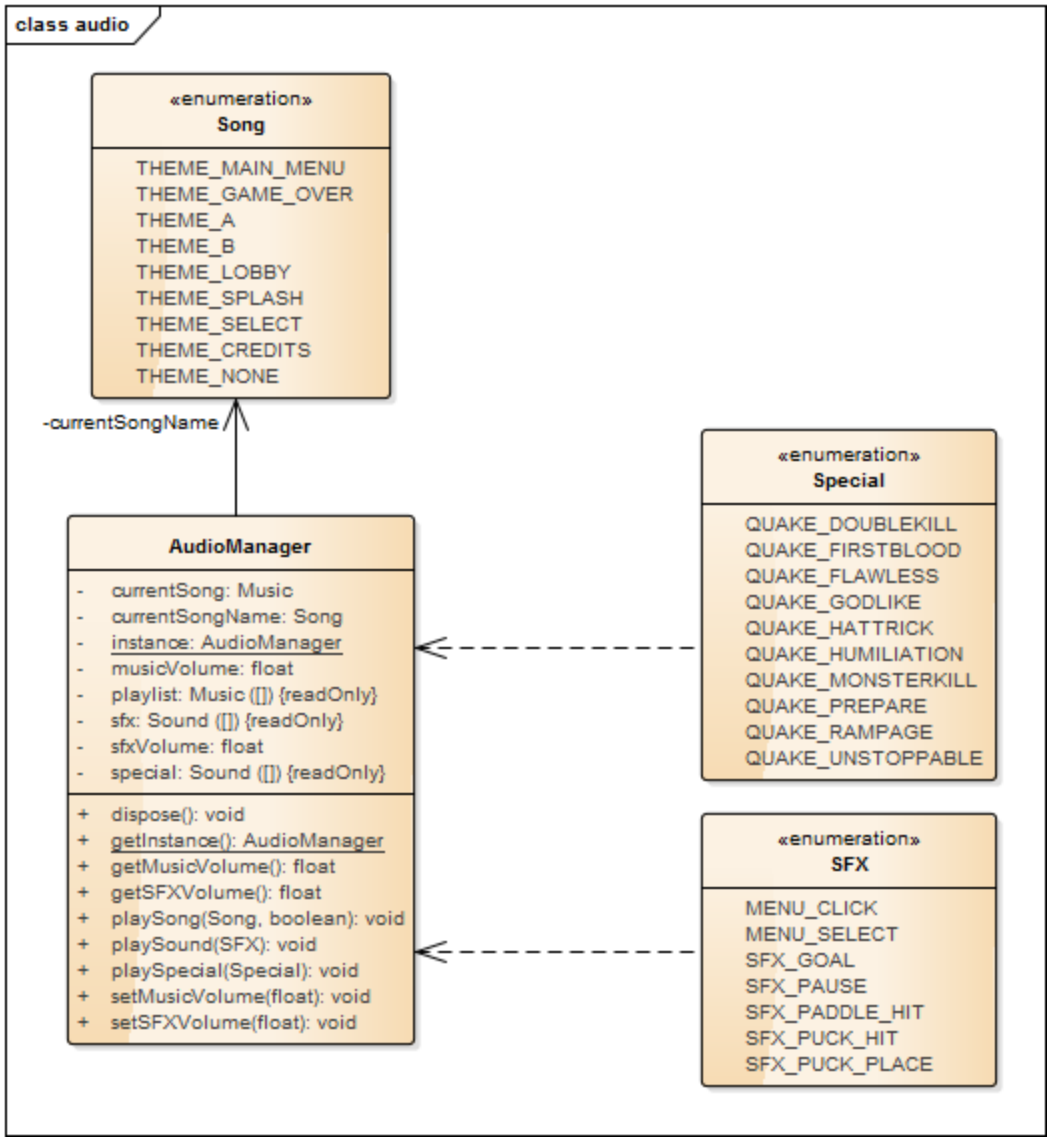
proj2.*audio*	▪ contém toda a camada de áudio da aplicação, funcionando como gestor e controlador de som integrado ▪ durante o arranque da aplicação armazena em memória todos os *assets* de áudio da aplicação, de forma a serem utilizados posteriormente com tempo de acesso mínimo ▪ constituído por uma 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 exceção 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 que permite a construção de um campo de jogo com paredes, balizas, *paddles* e um disco, bem como uma implementação das regras e modos de jogo disponíveis • a criação das *sprites* e a simulação da física é feita dentro das classes dos elementos do jogo respetivos • 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*
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* ou outro dispositivo móvel com sistema operativo *Android* (arquitetura cliente-servidor) ▪ constituída por apenas duas classes, *GameServer* e *Network*, sendo a última 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 também pela classe *GameServer*

7. Diagrama de classes

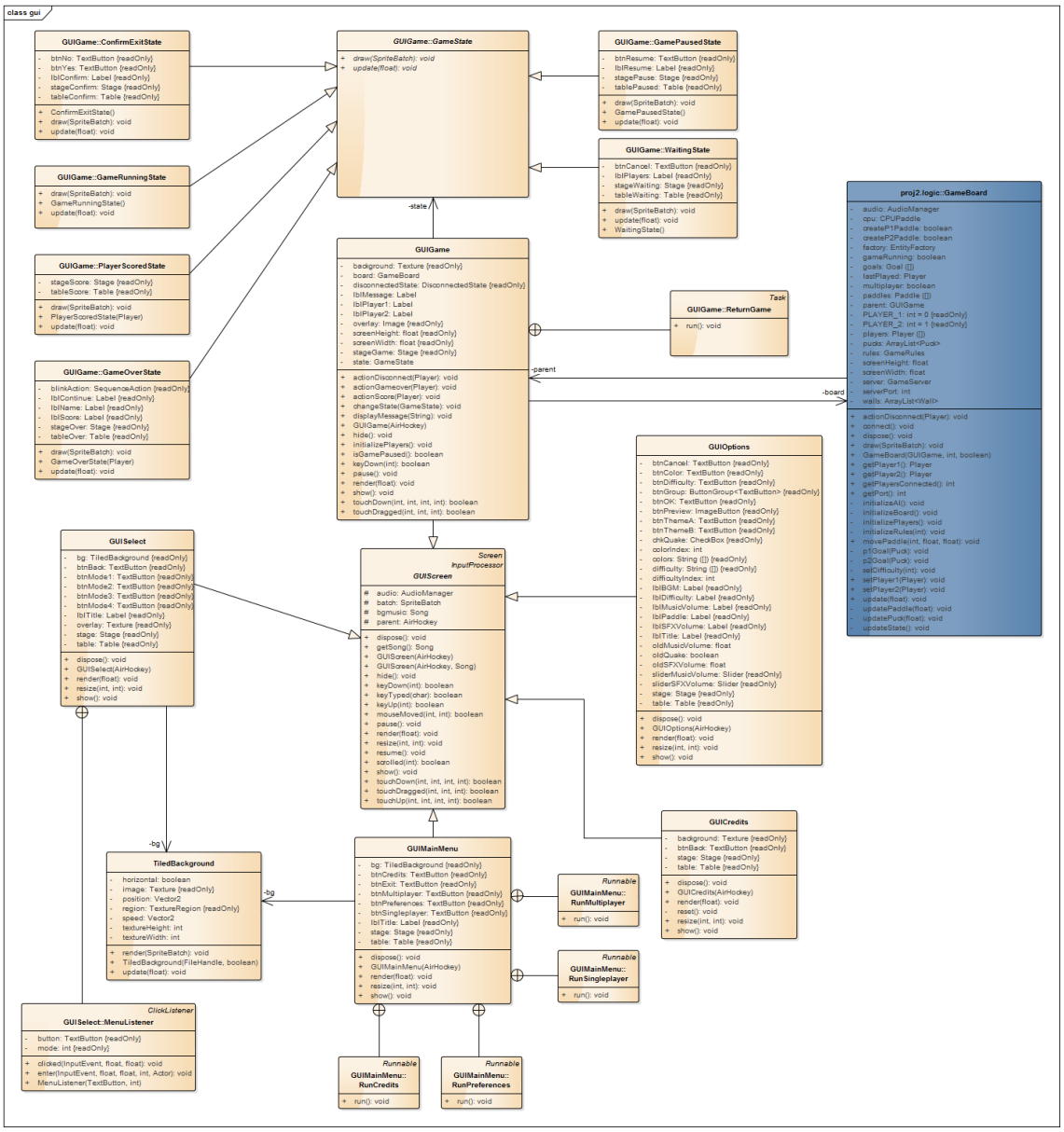
7.1 *Package* lpoo.proj2



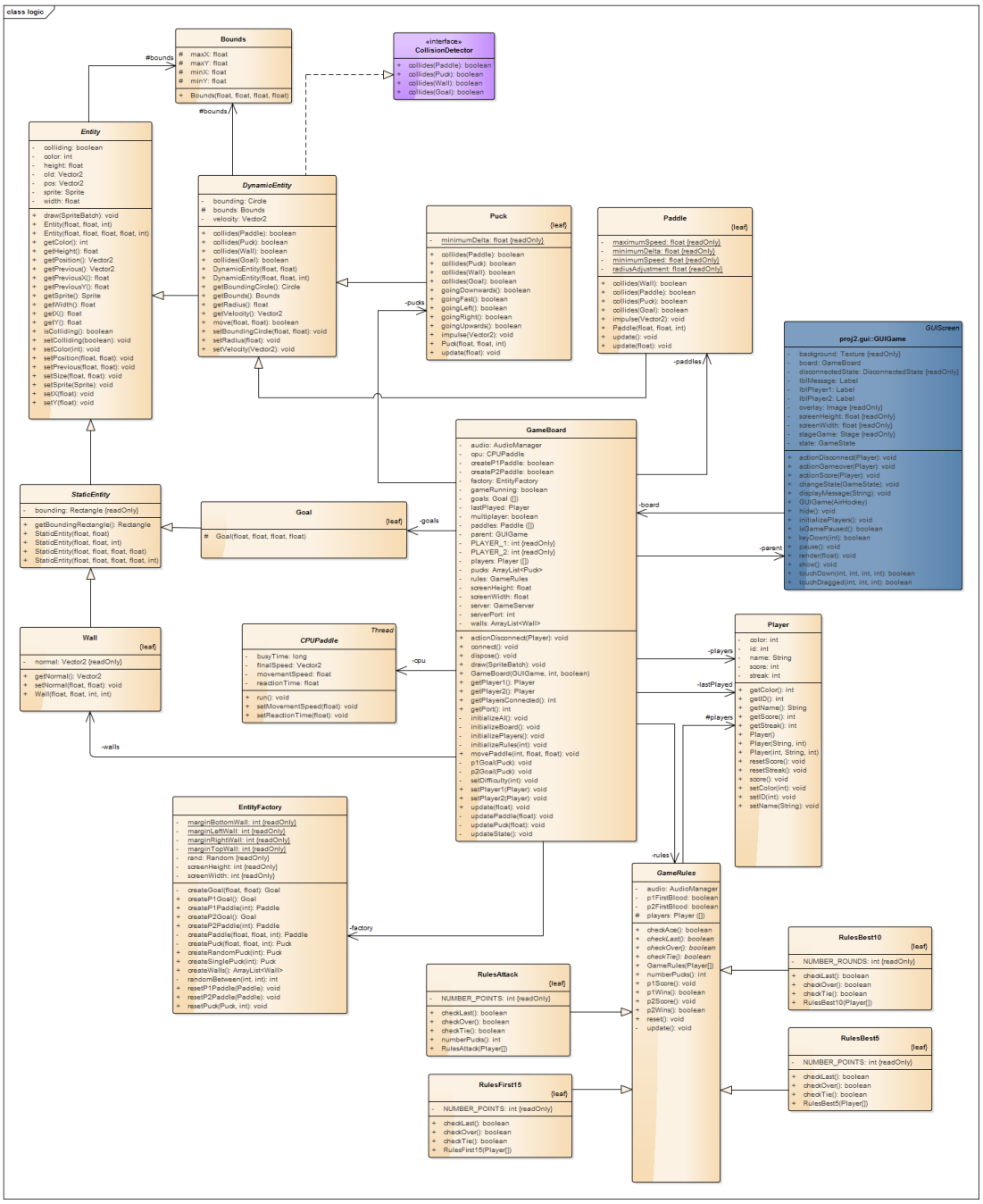
7.2 *Package* lpoo.proj2.audio



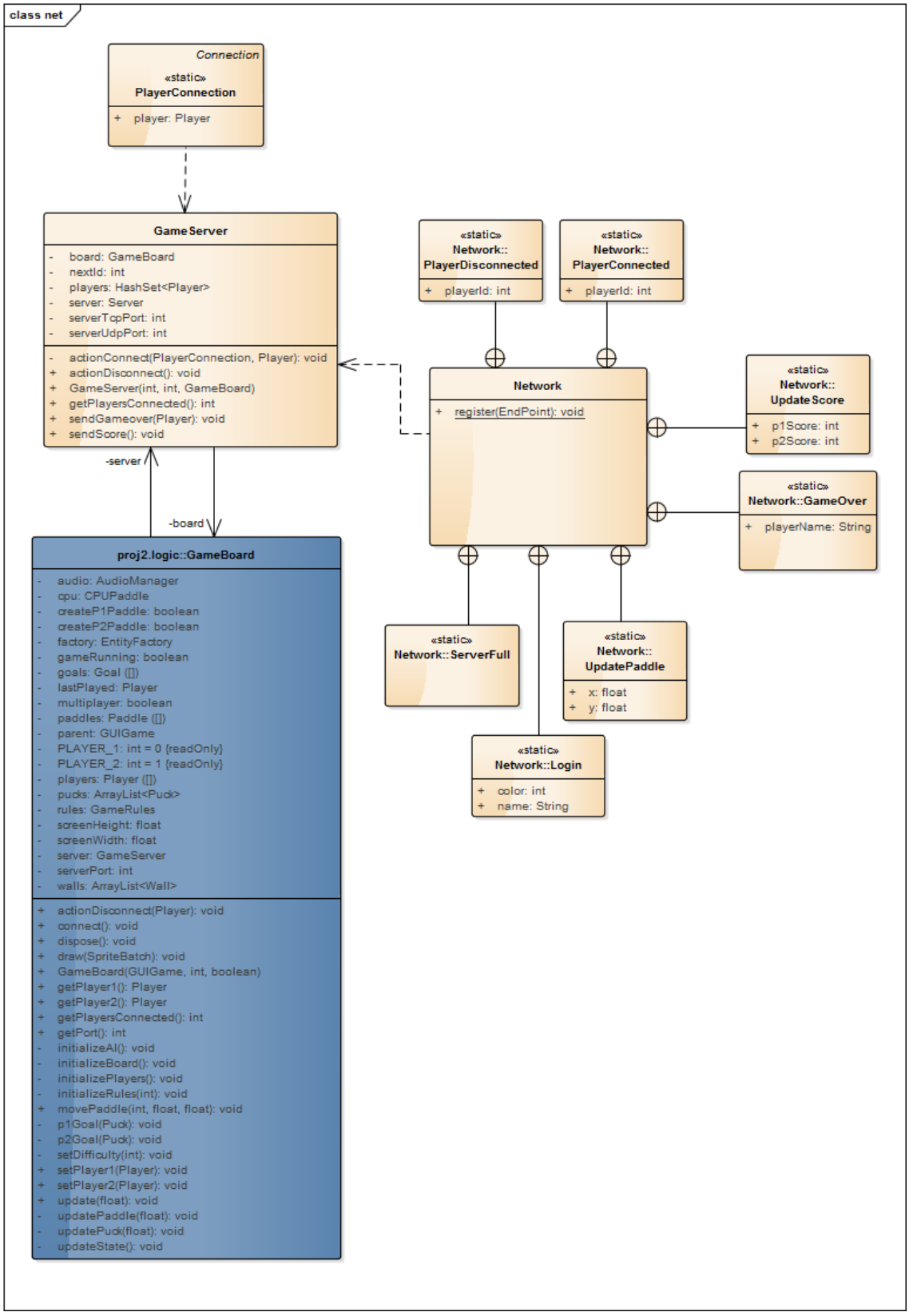
7.3 *Package* lpoo.proj2.gui



7.4 *Package* lpoo.proj2.logic



7.5 *Package* lpoo.proj2.net



8. Conclusão

O Air Hockey está completamente funcional, permitindo jogar em ambos os modos singleplayer e multiplayer, em diversos tipos de partida. Além disso, apresenta uma *interface* gráfica intuitiva e simples, permitindo ao utilizador configurar vários aspetos gráficos do jogo consoante as suas necessidades. Durante o jogo são também utilizados diversos efeitos sonoros e animações por forma a ser mais apelativo a qualquer jogador, independetemente da sua faixa etária. Conclui-se assim que foram atingidos os objectidos pretendidos para este projeto.

Embora satisfeitos com o resultado obtido, poderiamos ainda aperfeicoar a intelegencia artificial responsável pelo controlo do paddle inimigo no modo singleplayer por forma a torna-lo mais competitivo, simulando uma estratégia de jogo que se pareca mais com a de um humano. Outra melhoria a realizar seria o aperfeicoamento da deteção de colisões entre os diversos componentes durante as partidas, por forma a evitar as raras situações estranhas e não realistas em que o puck pode não se comportar como o previsto.

Ambos os elementos do grupo participaram na realização do Air Hockey e na seguinte proporção:

- ✓ Diogo Marques: %
- ✓ Pedro Melo: %