INSTITUTO SUPERIOR DE ENGENHARIA DO PORTO
MESTRADO EM ENGENHARIA INFORMÁTICA




Aplicação para intervenção em terapia ocupacional com o Leap Motion

PEDRO MIGUEL FERREIRA COSTA
Outubro de 2014

POLITÉCNICO
DO PORTO

Instituto Superior de Engenharia do Porto

Aplicação para intervenção em terapia ocupacional com o Leap Motion

À Descoberta das Ilhas

Pedro Miguel Ferreira Costa

Dissertação para obtenção do Grau de Mestre em Engenharia Informática, Área de Especialização em Sistemas Gráficos e Multimédia

Orientador: Paula Maria de Sá Oliveira Escudeiro

Porto, Outubro de 2014

Resumo

O projeto "À Descoberta das Ilhas" surge das lacunas de atenção e motivação por parte das crianças na realização de exercícios na terapia ocupacional, aliadas a uma subjetividade na análise do seu progresso.

Direcionado para crianças com dificuldades de integração bilateral motora, com idades compreendidas entre os cinco e nove anos, este projeto tem como base um jogo 3D para as plataformas *Windows*, *Mac OS X* e *Linux*, controlado com os movimentos dos membros superiores através do dispositivo *Leap Motion*. Através do controlo de um avião, a criança descobre várias ilhas e desbloqueia componentes do mesmo, alcançando os diversos bónus e *checkpoints* ao longo de cada percurso. Ao terapeuta são apresentados gráficos com dados obtidos pelo dispositivo aquando do momento lúdico da criança que permitem acompanhar a sua evolução a cada nível.

O sucesso no cumprimento dos objetivos do projeto permitiu confirmar a utilidade da aplicação na intervenção e avaliação do público-alvo.

Palavras-chave: integração bilateral motora, *Leap Motion*, jogo 3D, terapia ocupacional

Abstract

The project “À Descoberta das Ilhas” comes from the lack of attention and motivation in children when preforming the traditional exercises in occupational therapy sessions, furthermore, the analysis of their progress is based on subjectivity.

Aimed at the children with bilateral motor integration difficulties, aged between five and nine years, this project is based on a 3D game for Windows, Mac OS X and Linux platforms, that is controlled by the upper limb’s movements through the Leap Motion device. By controlling a plain, the child is able to discover several islands and unlock plain’s components, achieving bonus and several checkpoints along each journey. Graphs with data obtained by the device are then presented to the therapist enabling him monitoring and evaluation the child’s performance.

The success in meeting the objectives of the project confirmed the usefulness of the application in the assessment and intervention of the target audience.

Key-words: bilateral motor integration, Leap Motion, 3D game, occupational therapy.

Índice

Lista de Figuras

Lista de Tabelas

Acrónimos e Símbolos

Lista de Acrónimos

ISEP	**I**nstituto **S**uperior de **E**ngenharia do **P**orto
BD	**B**ase de **D**ados
IDE	***I**ntegrated **D**evelopment **E**nvironment*
QEF	***Q**uality **E**valuation **F**ramework*
SDK	***S**oftware **D**evelopment **K**it*
USB	***U**niversal **S**erial **B**us*
XML	***E**xtensible **M**arkup **L**anguage*
API	***A**pplication **P**rogramming **I**nterface*

Introdução

Neste capítulo inicial do relatório será feita uma apresentação inicial do projeto, bem como o seu enquadramento, objetivos e resultados. Será também aqui apresentada a empresa, as reuniões efetuadas, o planeamento e uma breve descrição das tecnologias utilizadas no desenvolvimento do projeto.

1.1 Enquadramento

Este projeto foi realizado no âmbito da Tese do Mestrado em Engenharia Informática – Sistemas Gráficos e Multimédia no Instituto Superior de Engenharia do Porto. Esta unidade curricular decorre no segundo ano e o principal objetivo é a aplicação dos conhecimentos adquiridos pelo aluno ao longo do curso num contexto real. Toda esta experiência deve também permitir o desenvolvimento de novas competências profissionais, sociais e pessoais.

O projeto foi iniciativa própria do aluno cuja oportunidade surgiu de uma conversa informal entre o mestrando e o seu amigo Marco Leão. O último é terapeuta ocupacional na clínica *Getting It*, Pediatria e Desenvolvimento Lda. Este projeto tem como finalidade auxiliar a intervenção ao nível das dificuldades motoras em crianças e permitirá tornar os exercícios mais motivadores através do dispositivo *Leap Motion*.

1.2 Motivação

Este projeto surge da curiosidade e entusiasmo do mestrando na área dos videojogos além da convicção de que estes podem ser transformados numa ferramenta útil para o uso na área da saúde.

A oportunidade de explorar conhecimentos e investir esforço e dedicação no auxílio do saudável crescimento de crianças foi considerada a melhor para a criação de um projeto que encerra este ciclo académico.

1.3 Apresentação do projeto

O projeto "À Descoberta das Ilhas" consiste num jogo 3D cujo objetivo principal é auxiliar a intervenção ao nível das dificuldades de motricidade global e fina e coordenação motora, frequentes em crianças com atrasos de desenvolvimento.

O público-alvo do jogo varia entre os 5 e os 9 anos de idade e caracteriza-se pelas suas dificuldades de lateralidade e habilidade motora, moderadas, não existindo entre as possíveis causas do diagnóstico apresentado crianças invisuais. No presente estudo verifica-se o treino da habilidade motora fina, definida pelo emprego de força muscular mínima, no caso a mão e também pela visão.

Através do dispositivo *Leap Motion*, um sensor de movimentos que suporta mãos e dedos, a criança controla um avião com os movimentos verticais de ambos os membros superiores, alcançando os diversos bónus e *checkpoints* ao longo de cada percurso. Os *checkpoints* permitem abastecer o depósito do avião, que se vai esvaziando ao longo do tempo, enquanto os bónus permitem desbloquear novos níveis e *upgrades* ao avião, motivando assim o jogador a continuar e a superar-se.

O projeto desenvolvido permite às crianças melhorarem as suas capacidades de graduação dos movimentos dos membros superiores, lateralidade e precisão, essenciais à integração bilateral motora, ou seja, à capacidade de coordenar simultaneamente os dois lados do corpo e desenvolver especialização hemisférica (lado dominante).

Para acompanhar todo o progresso das crianças no jogo foi implementada uma área que apresenta ao terapeuta o progresso de cada jogador, nomeadamente o campo de ação de

cada mão. Com base em dados adquiridos através do *Leap Motion*, a plataforma permite ao terapeuta analisar os limites mínimos e máximos atingidos, bem como as distâncias percorridas por cada mão em cada eixo numa representação tridimensional. Ou seja, são adquiridos e apresentados dados ao terapeuta acerca dos movimentos de cada mão das crianças em largura (eixo de X), altura (eixo de Y) e profundidade (eixo de Z).

Uma vez que o avião é exclusivamente controlado através de movimentos alternados no eixo Y, ou seja, os membros superiores movem-se na vertical, todos os movimentos efetuados no plano horizontal (eixos X e Z) são dispensáveis e representam uma precária coordenação do jogador. Como estes dados são individualizados por mão, é possível a comparação de ambas, avaliando assim o hemisfério dominante. Todos os dados são apresentados por nível, permitindo uma análise do sucessivo progresso do jogador.

O desenvolvimento de jogos, na sua maioria, envolve uma equipa multidisciplinar composta por profissionais de cada área, nomeadamente *game designers*, artistas e programadores. Neste projeto o mestrando realizou individualmente toda a análise, desenho, implementação e testes da aplicação, explorando assim as suas competências nestas diversas áreas.

1.4 Apresentação da organização

A *Getting It*, Pediatria e Desenvolvimento Lda é uma instituição que atenta às necessidades das crianças e jovens com alterações do neurodesenvolvimento ou problemas de comportamento. Reúne um leque variado de especialistas que, trabalhando em equipa num espaço acolhedor e informal, se propõem:

- Diagnosticar e avaliar;

- Tratar e orientar;

- Elaborar e implementar programas de intervenção personalizados;

- Informar sobre apoios sociais, educacionais e de saúde;

- Encaminhar para associações específicas de pais e doentes. (Getting It, sem data)

Esta organização foi fundamental para o desenvolvimento do projeto na medida em que permitiu a interação do projeto com o público-alvo. A articulação entre o mestrando e a organização foi estabelecida pelo terapeuta Marco Leão.

1.5 Contributos do projeto

As crianças e jovens são o público-alvo da clínica *Getting It* e, face às novas tecnologias, tornou-se quase obrigatório expandir as metodologias através desta área.

O uso de novas ferramentas possibilita assim a criança mais autónoma, permitindo uma maior estimulação, motivação, interação e concentração perante os exercícios. Para além destas vantagens para a criança, o projeto permite ainda armazenar e apresentar informação sobre o progresso do jogador, esta última muito útil para o diagnóstico e acompanhamento por parte do terapeuta.

O acesso das crianças às novas tecnologias tende a ser mais precoce, tornando a utilização destas cada vez mais necessária para cativar os mais novos.

Deste modo, o projeto mostra-se como um grande avanço face às perspetivas da *Getting It* neste campo.

1.6 Planeamento de projeto

Na fase inicial do projeto foi realizado um planeamento de forma a definir as tarefas necessárias e o tempo a despender em cada uma. Este planeamento teve como objetivo organizar o tempo disponível para realizar as várias iterações. Após reflexão e análise dos diferentes métodos de organização e planeamento de tarefas, foi decidido aquele que parece ser o mais adequado à elaboração deste projeto.

Deste modo, para auxiliar na conceção do projeto foi utilizada a metodologia de processo de desenvolvimento de jogos referenciada pela empresa *E-Games - Educational Games at Purdue* (ver Figura 1), juntamente com o sistema de gestão de desenvolvimento de *software Scrum*.

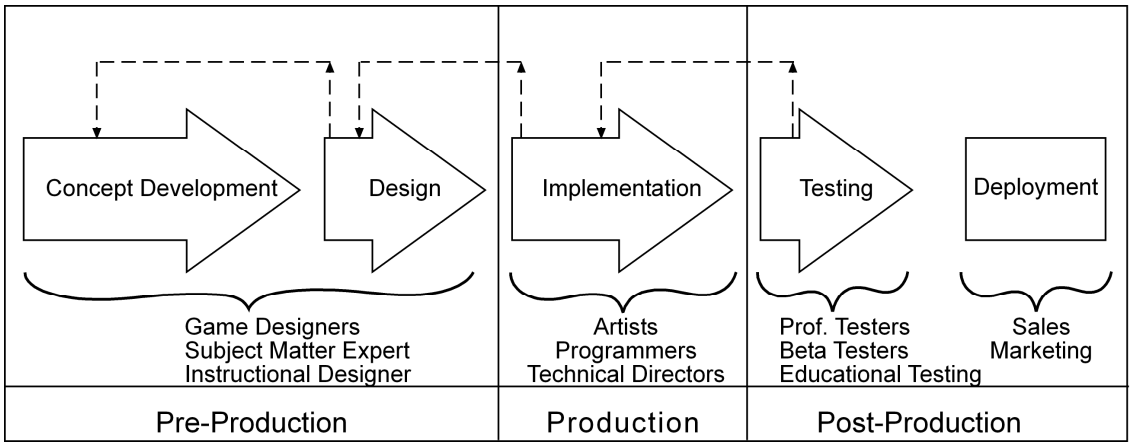


Figura 1 - Processo de Desenvolvimento de Jogos proposto pela *Educational Games as Purdue* (E-Games, sem data)

O *Scrum* é um processo de desenvolvimento ideal para gestão de projetos constituídos por várias pessoas cujos requisitos não estão totalmente definidos no início. Neste projeto, a metodologia foi adaptada para um único integrante (o mestrando) e foi elaborado um *master backlog* com os requisitos principais e vários *sprint logs* com as próximas tarefas a efetuar. A periocidade dos *sprints* foi de um mês e o planeamento consistiu na divisão do projeto pelas seguintes fases:

1ª - Análise:

- Análise do estado da arte;

- Análise do *Leap Motion*;

- Análise de requisitos do jogo;

- Descrição global da aplicação;

- Definição de regras do jogo;

- Análise de requisitos da área de progresso do jogador.

2ª - Desenho:

- Desenvolvimento de esboços;

- Desenvolvimento dos modelos;

- Desenvolvimento dos cenários;

- Desenvolvimento de texturas;

- Desenvolvimento de menus;

- Desenvolvimento de sons.

3ª - Implementação:

- Implementação do jogo;

- Implementação da área de progresso do jogador.

4ª - Testes:

- Testes da aplicação;

- Testes com o público-alvo.

5ª – Documentação:

- Plano de Projeto;

    - Game Design Document;

    - QEF (Quality Evaluation Framework);

    - *QEF* – Resultados;

    - Elaboração do relatório.

A documentação foi iniciada após a fase de análise e prolongou-se até ao final do projeto.

No decorrer do projeto foram encontrados obstáculos que dificultaram o cumprimento rigoroso dos prazos estipulados no planeamento, conduzindo a um desvio na linha temporal de execução. Este foi causado principalmente pela mudança da entidade patronal do mestrando, refletindo-se num aumento da carga horária laboral. Esta alteração ocorreu na fase de desenvolvimento do projeto e reduziu significativamente o tempo disponível para a realização do mesmo.

Tabela 1 - Tarefas e prazos planeados

Fase	Tarefa	Data / Sprint
Análise	Pesquisa e análise da problemática	31 de Março
Análise	Pesquisa e análise do *Leap Motion*	
Análise	Pesquisa e análise do *Leap Motion* aplicado à saúde	
Análise	Visita à *Getting It*, clínica parceira do projeto	
Análise	Definição das metodologias e técnicas a utilizar	
Análise	Definição do conceito do jogo	30 de Abril
Análise	Definição da cenografia do jogo	
Análise	Definição dos requisitos do jogo	
Desenho	Protótipo do jogo de avião (controlo com *Leap Motion*)	
Desenho	Desenvolvimento dos modelos	31 de Maio
Desenho	Desenvolvimento dos cenários	
Documentação	Elaboração do Plano de Projeto	
Documentação	Elaboração do *Action Table*	
Documentação	Elaboração do *Game Rules*	
Documentação	Elaboração do *Storyboard*	
Documentação	Elaboração do *QEF*	30 de Junho
Documentação	Elaboração do Game Components	
Documentação	Elaboração do Learning Knowledge	
Documentação	Elaboração do Screen Architecture	
Documentação	Elaboração do *Game Flowchart*	
Análise	Sistema de BD (Base de Dados) de jogadores	
Implementação	Implementação do sistema de BD de jogadores	
Implementação	Implementação dos menus	
Implementação	Integração dos menus com o *Leap Motion*	
Documentação	Elaboração do Game Design Document	31 de Julho
Desenho	Desenvolvimento da animação do início do nível	
Implementação	Implementação da animação do início do nível	
Implementação	Implementação da animação do fim do nível	
Desenho	Desenvolvimento da animação do fim do nível	31 de Agosto
Implementação	Implementação da animação do fim do Nível	
Implementação	Implementação das escalas de níveis	
Desenho	Desenvolvimento dos *upgrades* do avião	
Implementação	Implementação dos *upgrades* do avião	
Desenho	Redefinição de alguns modelos	
Implementação	Implementação da animação quando acaba o combustível	30 de Setembro
Implementação	Alteração do arranque do avião	
Documentação e Implementação	Alteração das regras de jogo (acrescentar vidas por nível)	
Implementação	Leitura de dados do *Leap Motion* para a área de progresso do jogador	
Implementação	Integração dos menus com o jogo	
Desenho	Sons	
Implementação	Integração dos sons	
Desenho e	Elaboração do *Storyboard* (área de progresso do	26 de Outubro

Documentação	jogador)
Implementação	Implementação da área de progresso do jogador
Implementação	Correção de erros
Desenho	Animação de introdução do jogo
Implementação	Implementação da animação de introdução do jogo
Testes	Testes ao jogo com o utilizador final (crianças)
Testes	Testes à área de progresso do jogador com o utilizador final (terapeuta)
Documentação	*QEF* - Resultados
Documentação	Relatório Final

1.6.1 Reuniões de acompanhamento

Ao longo do projeto foram agendadas reuniões entre o mestrando, o terapeuta Marco Leão da *Getting It*, a Professora Paula Escudeiro e a Psicóloga Carla Ribeiro.

De forma a definir e discutir o projeto na globalidade, reuniões foram agendadas com a professora Paula Escudeiro. Para perceber o problema que a tese visa colmatar, o Marco Leão foi o terapeuta que apresentou a realidade ao mestrando, representado também o papel de cliente final do projeto. Para auxílio numa vertente mais focada no público-alvo do jogo, as crianças, foram realizadas reuniões com a psicóloga Carla Ribeiro.

Devido à ocupação da agenda do mestrando, foram reuniões que se realizaram com menor frequência do que inicialmente estaria previsto. No entanto, esta menor frequência permitiu ter uma maior autonomia quer no desenvolvimento técnico quer no processo de levantamento de requisitos, o que acabou por surtir resultados favoráveis.

Tabela 2 - Reuniões

Data	Local	Participantes	Assuntos Discutidos
19 de Setembro de 2013	ISEP	Pedro Costa e Paula Escudeiro	- Apresentação da proposta de tese.
24 de Outubro de 2013	ISEP	Pedro Costa e Paula Escudeiro	- Discussão da estrutura do relatório; - Decisão da metodologia de desenvolvimento.
8 de Março de 2014	Getting It	Pedro Costa e Marco Leão	- Primeiro contacto com a instituição; - Demonstração de dois jogos a crianças para verificar o seu comportamento face ao Leap Motion.
25 de Abril	Via Skype	Pedro Costa e	- Apresentação da proposta de tese;

de 2014		Carla Ribeiro	- Orientação e aconselhamento relativamente a características do jogo mais adequadas ao seu objetivo e ao público-alvo: *design*, conceito e motivação.
30 de Agosto de 2014	Consultório	Pedro Costa e Carla Ribeiro	- Apresentação e discussão sobre o jogo (quase terminado); - Considerações relativamente ao *design*; - Discussão sobre teorias de aprendizagem inerentes no jogo.
18 de Setembro de 2014	Via Skype	Pedro Costa e Marco Leão	- Discussão sobre rascunhos da área de progresso do jogador; - Revisão dos dados a apresentar na área de progresso do jogador;
23 de Setembro de 2014	ISEP	Pedro Costa e Paula Escudeiro	- Discussão sobre o *Game Design Document* e da sua inserção no relatório final; - Apresentação do jogo final; - Apresentação do protótipo da área de progresso do jogador.

1.7 Tecnologias utilizadas

Um dos objetivos do projeto foi uniformizar toda a aplicação para que o terapeuta possa acompanhar todo o processo, desde o jogo até à área de progresso do jogador, de forma simples e intuitiva.

Para a interação do utilizador com o computador foi utilizado o dispositivo *Leap Motion*, um sensor de movimentos USB (Universal Serial Bus) que suporta as mãos e os dedos sem necessidade de contato.

A ferramenta principal utilizada foi o *Unity 3D*, um motor de jogos e *IDE* (*Integrated Development Environment*) gráfico. Através desta foram implementados o jogo, a área de progresso do jogador e a leitura de dados do *Leap Motion*, sendo necessário para o segundo a utilização da *SDK* (*Software Development Kit*) disponível no *site* da própria empresa. A linguagem de programação utilizada nesta ferramenta foi o *C Sharp*.

Para armazenar o registo dos jogadores e *upgrades* do avião foi desenvolvida uma base de dados sob o modelo relacional e armazenada no formato *XML* (*eXtensible Markup Language*).

1.8 Organização do relatório

Neste primeiro capítulo, Introdução, é efetuado o enquadramento e a apresentação geral do projeto e dos seus objetivos. De seguida, é efetuada uma breve apresentação da organização que parceira neste projeto, dos contributo do projeto, do seu planeamento e também das tecnologias utilizadas.

No segundo capítulo, Contexto, é efetuada uma contextualização do projeto. Neste capítulo será apresentada a patologia, o problema detetado e o estado de arte elaborado no projeto.

No terceiro capítulo, Descrição Técnica, é apresentada a análise da solução pretendida, seguida da descrição de todo o processo de desenvolvimento do projeto. São também descritos os testes realizados e são apresentadas as dificuldades encontradas no processo de desenvolvimento.

Por último, no quarto capítulo, Conclusões, é feita uma visão geral do trabalho desenvolvido e dos possíveis trabalhos a desenvolver no futuro.

Contexto

2.1 Integração bilateral motora

A integração bilateral motora refere-se ao uso conjunto e coordenado de ambos os lados do corpo. A coordenação dos dois lados do corpo é a base importante para o desenvolvimento de muitas capacidades motoras (finas e grossas). Gradualmente leva à consciência dos dois lados do corpo, definição de um lado como dominante (lateralidade) e sentido de direccionalidade (discriminação entre direita e esquerda, cima e baixo, frente e trás). Além disso, também a capacidade de cruzar a linha média (imaginária) é um importante passo adaptativo para o qual é indispensável uma boa integração bilateral motora. (Ayres, 1974)

As capacidades de coordenação motora desenvolvem-se naturalmente na maioria das crianças, no entanto, isso não acontece com grande parte das que apresentam necessidades educativas especiais. Distúrbios no desenvolvimento na coordenação estão presentes em aproximadamente 5 a 6% das crianças em idade escolar. (Zwicker et al., 2012)

As crianças começam por movimentar ambos os lados do corpo simetricamente (i.e. bater palmas), depois alternadamente (i.e. gatinhar) e mais tarde assimetricamente (i.e. pintar). As atividades que envolvam o uso das duas mãos tornam-se difíceis de completar. A criança pode não ser capaz de coordenar uma mão enquanto a outra se mantém estável, ou pode trocar de mão durante a realização de uma tarefa de motricidade fina por estar a ter dificuldades em usar ambas as mãos em simultâneo.

Desta forma, uma boa coordenação dos dois lados do corpo é muito importante para o correto desempenho de múltiplas atividades quotidianas.

2.2 Problema e proposta de solução

Tradicionalmente, com o objetivo de estimular as capacidades motoras precariamente desenvolvidas nas crianças com dificuldades de integração bilateral motora, são realizadas atividades na terapia ocupacional tais como: apanhar e atirar bolas com as duas mãos; realizar trabalhos de arte plástica (colar; recortar; etc.); fazer construções com legos; compor um colar de enfiamentos; abrir caixas ou frascos usando ambas as mãos.

O principal problema detetado na terapia dos distúrbios de integração bilateral motora é a dificuldade na concentração e motivação nos exercícios. Atualmente assiste-se a um precoce acesso aos meios digitais por parte das crianças, facto responsável pela desatualização dos exercícios tradicionais. Por outro lado, no que diz respeito aos terapeutas, estes deparam-se com a dificuldade na uniformização da avaliação e acompanhamento destas crianças nas diferentes sessões. Isto deve-se a uma falta de critérios validados para uso generalizado, o que torna o diagnóstico e resposta à terapia algo subjetivos.

No desenho deste projeto direcionado para crianças com problemas de integração bilateral motora pretende-se, para além do desenvolvimento das competências motoras necessárias ao desempenho das tarefas diárias, fazer face às dificuldades que estas crianças enfrentam quando percebem ineficácia nas suas tentativas de realização da tarefa e frustração, inibindo o seu desempenho em vários contextos, nomeadamente o escolar. Assim, surge a importância de aliar o mundo virtual às mais variadas terapias nestes domínios, nomeadamente a terapia ocupacional, uma vez que se permite à criança com este diagnóstico interagir com o mundo virtual, tal como outra criança o faz, e em seu próprio benefício. Neste sentido, o projeto visa treinar e desenvolver competências quer motoras, quer cognitivas.

2.3 Estado da arte

Com a evolução tecnológica no mundo atual, cada vez mais se encontram vastas ofertas de videojogos que facilmente cativam as crianças e as prendem horas a fio ao computador, consola, *smarpthone* ou *tablet*.

De facto, a dedicação e atenção que as crianças atualmente aplicam a estes jogos alertaram profissionais na área da saúde para que tentassem integrar na sua atividade videojogos para aumentarem a motivação das crianças para o processo de intervenção.

"A realidade é que as crianças se estão a tornar mais sedentárias e passam mais tempo em frente à televisão e a jogar videojogos. Esta atração das crianças e adolescentes pelos videojogos e novas tecnologias confere-lhes valor enquanto um potencial instrumento terapêutico", afirmou a diretora do laboratório de psicologia e tecnologia (*LabPsiTec*) das Universidades *Jaume I* e Universidade de Valência.

Como anteriormente referido, na área das perturbações de desenvolvimento é frequente encontrarem-se casos de crianças com dificuldades de coordenação motora (global e fina) que interferem no desempenho das suas ocupações diárias, e que necessitam de intervenção na área de terapia ocupacional para que esse impacto seja minimizado. Contudo, quando se percorre a literatura percebe-se que poucos recursos estão disponíveis ao nível da utilização de videojogos para se avaliar e intervir ao nível das dificuldades motoras.

Wii

A consola *Wii* da empresa *Nintendo* foi lançada no mercado em 2006 e destaca-se pelo seu controlador *Wii Remote,* sem fios, equipado com botões e capaz de detetar movimentos 3D através de uma ligação *Bluetooth* (ver Figura 2). Este controlador, parte integrante da consola, é detetável a uma distância máxima de aproximadamente cinco metros e apresenta as dimensões em centímetros de 14,8 (altura) x 3,6 (largura) x 3,1 (espessura). Em 2013, o seu preço em Portugal era de, aproximadamente, 110 euros (*Wii* + *Wii Remote*).

Figura 2 - *Wii Remote* (imagem retirada de www.fastgames.com.br)

Em 2007 foi lançada a *Balance Board* (ver Figura 3), um acessório que consiste numa balança com sensores de pressão. Com as dimensões em centímetros de 31 (altura) x 51 (largura) x 6 (espessura) e peso de aproximadamente quatro kg, apresentava um custo de 70 euros em 2013.

Figura 3 - *Wii Balance Board* (imagem retirada de www.engadget.com)

Na área da terapia ocupacional, foram analisados os seguintes jogos da *Wii*:

- *Wii Fit Plus*: Conjunto de mini jogos que permite treinar o equilíbrio, coordenação e consciência corporal;

- *Wii Sports*: coordenação visomotora.

Na pesquisa efetuada, concluiu-se que estas ferramentas e respetivos jogos são, sobretudo, utilizados na intervenção em doentes com sequelas de lesões neurológicas (i.e. AVC - Acidente Vascular Cerebral), sendo uma boa ferramenta terapêutica na reabilitação da força e capacidades motores perdidas.

DESVANTAGENS: comando, precisão, não é direcionado para crianças com necessidades especiais e não devolve dados.

Kinect

Nesta área de tecnologias existe também o popular *Kinect* (ver Figura 4). Este dispositivo é um sensor utilizado principalmente em jogos nas consolas *Xbox 360* e *Xbox One* da *Microsoft*. Com as dimensões em centímetros de 7,5 (altura) x 28 (largura) x 7,5 (espessura), o *Kinect*

permite a interação numa área de 0,4 a quatro metros realizando duas operações: o rastreio do esqueleto total e parcial e a deteção da profundidade.

Figura 4 - Kinect (imagem retirada de www.kanguruinfo.com.br)

A última versão do *Kinect* (em 2013) também mede a frequência cardíaca, reconhece expressões, acompanha seis utilizadores em simultâneo, determina as forças em várias articulações do corpo e faz uma simulação muscular. Foi também lançada uma versão do dispositivo para *Windows* com mais funcionalidades do que a versão para a consola. O preço do *Kinect* para *Windows* em 2013 era cerca de 197 euros e para a *Xbox* cerca de 55 euros.

Na área da terapia ocupacional, foram analisados os seguintes jogos do *Kinect*:

- *Kinect Dance Central:* força e motricidade grossa;
- *Kinect Training*: força, resistência e consciência corporal;
- *Kinect Sports*: coordenação visomotora.

Uma vez deteta os movimentos de todo o corpo, este dispositivo tem ganho realce na área do condicionamento físico junto da população idosa, permitindo a realização de exercício físico de forma confortável sem acompanhamento permanente e presencial de um terapeuta. Realça-se ainda a sua utilidade na reabilitação de habilidades em doentes a recuperar de lesões cerebrais.

Leap Motion

O *Leap Motion* (ver Figura 5) apresentou-se no mercado em 2013 como um dispositivo inovador que aumenta a dimensão da interação do utilizador com o computador. Com a leitura de uma área hemisférica 3D de aproximadamente meio metro de raio, a tecnologia suporta mãos, dedos e ferramentas semelhantes a dedos (i.e. lápis) e é uma das mais precisas

do mercado. Com as dimensões em centímetros de 1,3 (altura) x 8 (largura) x 3 (espessura), o preço do *Leap Motion* em 2013 era cerca de 60 euros.

Figura 5 - *Leap Motion* (imagem retirada de www.robobshop.com)

Uma vez que a tecnologia é recente, ainda existem poucos projetos na área da saúde. O único passível de análise foi o projeto *Richard RIESE*. Este foi desenvolvido especificamente para a reabilitação de pacientes com diferentes tipos de deficiências que afetam a sua função da mão-motor, incluindo crianças com deficiência motora fina, pessoas com lesões nas mãos ou pacientes que tenham sofrido AVC (Acidente Vascular Cerebral) com paralisias parciais. O projeto é constituído por um videojogo que permite ao utilizador fazer os exercícios por conta própria em casa de uma forma fácil e divertida, com o auxílio do *Leap Motion*. Contém também um *backoffice* que permite o acompanhamento e análise de um terapeuta, como também a personalização de alguns parâmetros do jogo, nomeadamente a qualidade dos gráficos, útil para pessoas mais idosas ou com dificuldades visuais. Existe pouca informação sobre este projeto e o mestrando tentou contactar o seu autor mas, até ao momento, não obteve resposta. (Sturm, 2013)

Discussão

Perante o levantamento do estudo do mercado anteriormente apresentado verificou-se a escassez de aplicações especialmente direcionadas a crianças com dificuldades de integração bilateral motora. Por outro lado, uma grande desvantagem transversal à maioria dos jogos existentes prende-se com o facto de estes apenas permitirem a "avaliação" através da pontuação obtida, não devolvendo dados específicos do desempenho do utilizador. Deste

modo mostram-se apenas uteis no campo da intervenção, sem capacidade de acompanhamento.

Após análise das diferentes tecnologias e respetivos projetos, foi realizada uma comparação entre as mesmas, sumariando-se os principais requisitos na Tabela 3.

Tabela 3 - Comparação de alguns requisitos entre a *Wii*, *Kinect* e *Leap Motion*

Requisitos	***Wii***	***Kinect***	***Leap Motion***
Hardware	Consola Wii + TV	Consola XBox + TV / computador	Computador
Acessórios	Comando e balança	Nenhum	Nenhum
Reconhecimento	Movimentos	Gestos, expressões faciais e voz	Gestos
Sensibilidade	Apenas o comando	Corpo todo	Mãos e dedos
Precisão	Nenhuma	Muita na motricidade global	Muita na motricidade fina
Portabilidade	Pouca	Pouca (XBox)/ média (Computador)	Muita
Preço (€)	110	55 / 197	60

Importa realçar que há vantagem nos movimentos das mãos e dedos livres de qualquer acessório, permitindo um maior *feedback* do posicionamento dos membros superiores, bem como autocorreção postural.

Através da análise da Tabela 3 conclui-se que o *Leap Motion* se evidencia neste domínio pela sua precisão, portabilidade e preço.

A integração deste dispositivo no projeto permite, não só resolver os problemas de motivação do público-alvo, como também auxiliar o terapeuta no acompanhamento do progresso da criança, apresentando dados mensuráveis, passiveis de uma avaliação objetiva. Assim sendo, o *Leap Motion* revela-se uma mais-valia para o projeto que visa colmatar os problemas inerentes aos exercícios tradicionais.

Descrição técnica

3.1 Tecnologias e ferramentas

Microsoft Visual Studio

O *Microsoft Visual Studio* é, em termos básicos, um *IDE* de desenvolvimento de *software*. Neste projeto foi utilizada a versão 2013 para escrita de código e também para a construção de diagramas de sistema, classes e sequência.

CSharp

Também escrito como C#, é uma linguagem de programação orientada a objetos, fortemente tipada, desenvolvida pela *Microsoft* como parte da plataforma .NET. Esta foi a linguagem maioritariamente utilizada no projeto.

XML

Sigla de *eXtensible Markup Language*, é uma linguagem de marcação utilizada para criar documentos com dados organizados de forma hierárquica e tem como principal objetivo facilitar a partilha de informação através da internet. Uma vez que o *XML* é na sua essência um ficheiro de texto (TXT), possui como grande vantagem o facto de ser independente de plataformas, podendo por isso ser interpretado por diferentes sistemas. Esta linguagem foi utilizada para a base de dados do jogo.

Adobe Illustrator CS6

O *Illustrator* é um editor de imagens vetoriais criado pela mesma empresa que desenvolveu o *Photoshop* e o *Premiere*. Este produto da Adobe é considerado um dos melhores programas do género, sendo sinónimo de profissionalismo. As ferramentas disponíveis são tudo o necessário para criar imagens perfeitas, redimensionando sem perdas de qualidade e exportando para vários formatos. Esta ferramenta foi utilizada para desenhar todos os gráficos existentes no projeto.

Adobe Photoshop CS6

O *Photoshop* é um poderoso editor de imagens bidimensionais da empresa Adobem que possuí também algumas funcionalidades do editor de vetores *Adobe Illustrator*. Esta ferramenta serviu de auxílio para tratamento de alguns gráficos do projeto.

Blender

O *Blender* é uma ferramenta que permite a criação de vastos conteúdos de 3D. Oferece funcionalidades completas para modelação, renderização, animação, pós-produção, criação e visualização de conteúdo 3D interativo e é dirigido a profissionais e artistas desta área. Esta ferramenta foi utilizada para desenhar todos os objetos 3D utilizados no jogo, importados posteriormente para o *IDE* de jogo.

Unity 3D

Também conhecido como *Unity*, o *Unity* 3D é um motor de jogo 3D proprietário e uma *IDE* criado pela *Unity Technologies*. Existem duas versões principais: *Unity Pro*, com o preço de, aproximadamente, 1124 euros, e a versão gratuita, simplesmente chamada *Unity*, que contém menos funcionalidades que a paga e pode ser usada tanto para fins educacionais, quanto para fins comerciais. Ambas as versões permitem compilar e exportar projetos para múltiplas plataformas. Neste projeto foi utilizada a versão gratuita.

Importa referir que umas das linguagens de programação que a *SDK* do *Leap Motion* suporta é o *CSharp* para o motor de jogo e *IDE Unity* 3D. Uma vez que o mestrando está à vontade com esta linguagem de programação, foi então decidida a ferramenta *Unity* 3D para o desenvolvimento do jogo.

Audacity

O *Audacity* é um editor de áudio que pode gravar, reproduzir e importar/exportar sons nos formatos WAV, AIFF, MP3 e OGG. Com esta ferramenta é possível editar músicas e aplicar cortes, copiar e colar recursos (com funcionalidades de desfazer/refazer ilimitadas), mixar faixas e aplicar efeitos na gravação. Esta ferramenta foi utilizada para editar os sons utilizados no projeto.

Mendeley

O *Mendeley* é um *software* para gestão de referências bibliográficas. Numa interface amigável destacam-se como características e funcionalidades o uso intuitivo do programa, a possibilidade de utilização do programa sem ligação à internet através da versão *desktop* (grátis) e a importação de referências bibliográficas a partir de bases de dados e catálogos bibliográficos. Permite a integração com processadores de texto, a seleção e aplicação de diferentes estilos bibliográficos e a sincronização automática entre as versões *desktop e Web*. (Donato, 2014)

3.1.1 Leap Motion

O *Leap Motion* é uma tecnologia inovadora integrada no projeto e de grande importância para o mesmo, merecendo assim especial destaque tendo sido criada esta secção para uma descrição mais pormenorizada.

É uma tecnologia criada pela empresa homónima apresentada ao mercado em 2013. Consiste num pequeno dispositivo com um sensor capaz de captar movimentos dos 10 dedos das mãos do utilizador com uma precisão aproximada de 1/100 milímetros. O dispositivo permite, portanto, controlar computadores com *Windows*, *Mac OS X* ou *Linux* usando apenas movimentos no ar, detetando as ações de: pintar, desenhar, agarrar, beliscar, dedilhar, entre outros.

O preço atual do *Leap Motion* é de 89.99€.

O grande destaque da tecnologia face aos seus concorrentes deve-se às suas pequenas dimensões, com as medidas 8 x 3 x 1,3 cm (ver Figura 6).

Figura 6 - *Leap Motion* e respetivas dimensões (imagem retirada de www.leapmotion.com)

Para leitura de dados, a tecnologia utiliza duas câmaras monocromáticas de infravermelhos e três *LEDs* para captar movimentos precisos numa área hemisférica, como descrito na Figura 7.

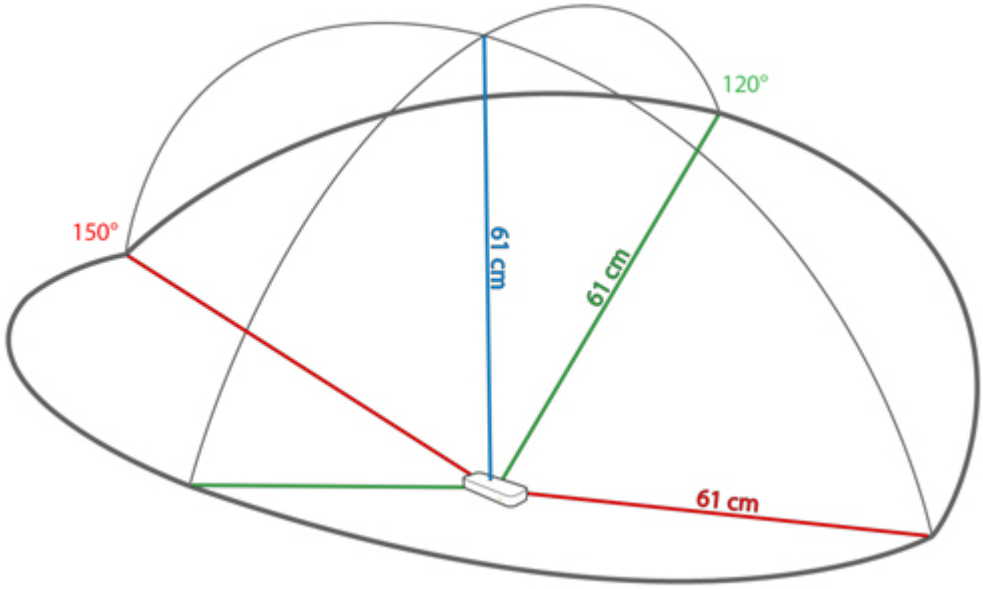


Figura 7 - Área de captura do *Leap Motion*. Os valores apresentados referem-se à totalidade das distâncias em cada eixo. (imagem retirada de www.clubic.com)

Os *LEDs* geram um padrão 3D de pontos de luz infravermelha e as câmaras geram mais de 200 *frames* por segundo (ver Figura 8). Estes dados são enviados através de um cabo *USB* para o computador, onde a informação é analisada pelo *software* do controlador e são sintetizados dados de posição 3D através da comparação das *frames* 2D geradas pelas duas câmaras. Além disso, a latência é inexistente para os olhos humanos, sendo inferior à taxa de atualização dos monitores de computador.

Figura 8 - A captura do *Leap Motion* das mãos do utilizador (imagem retirada de www.leapmotion.com)

Sistema de Coordenadas

O *Leap Motion* emprega um sistema de coordenadas cartesianas da mão direita com a origem centrada no topo do dispositivo. Os eixos X e Z encontram-se no plano horizontal e o eixo Y no plano vertical. As orientações de cada eixo estão representadas na Figura 9.

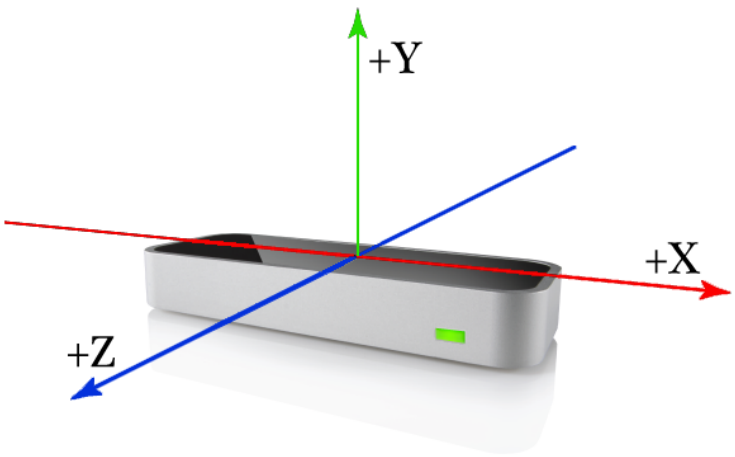


Figura 9 - Sistema de coordenadas do *Leap Motion* (imagem retirada de www.leapmotion.com)

A tecnologia mede grandezas físicas nas unidades descritas na Tabela 4.

Tabela 4 - Grandezas físicas e respetivas unidades do *Leap Motion*

Grandeza	**Unidade**
Distância	Milímetros
Tempo	Microssegundos
Velocidade	Milímetros/segundo
Ângulo	Radianos

Requisitos mínimos do sistema

- *Windows* 7 ou 8 ou *Mac OS X* 10.7 *Lion*;

- AMD Phenom II ou Intel Core i3, i5 ou i7;

- 2 *GB* de *RAM*;

- Porta *USB* 2.0;

- Ligação à internet para instalação.

Leap Motion SDK

Disponível no *site* da própria empresa existe uma *SDK* e necessária para o desenvolvimento de aplicações integradas com o *Leap Motion.* Suporta *Windows, Mac OS X, e Linux* em seis linguagens de programação, entre as quais *CSharp* para o motor de jogos e *IDE Unity 3D*, utilizado neste projeto.

Esta ferramenta contém uma poderosa *API (Application Programming Interface)* que permite aceder aos dados que o *Leap Motion* está a captar de forma simples. Existem várias versões deste *SDK*, no entanto, não importa referir qual foi utilizada neste projeto, pois o código é compatível com todas estas.

O código utilizado desta interface será apresentado na secção 3.4 Implementação nas respetivas fases do projeto onde foi implementado.

3.2 Análise

Esta fase é crucial para um projeto pois é a primeira no processo de desenvolvimento de *software*. Uma boa análise reduz substancialmente o trabalho de desenvolvimento de *software*, enquanto aproxima a solução final da solução pretendida.

Nesta secção será então apresentada toda a análise efetuada no projeto.

A análise do jogo teve em principal consideração a interação entre as crianças e a aplicação. Um dos objetivos cruciais do jogo é a cativação e motivação das crianças durante a utilização da aplicação e, em simultâneo, a adquisição de dados para posterior análise. Esta última efetuada na área de progresso do jogador, onde a interação ocorre entre o terapeuta e o menu.

3.2.1 Breve descrição do jogo

O jogo "À Descoberta das Ilhas" consiste no controlo de um avião de forma a conquistar ilhas desconhecidas. É direcionado para crianças com idade compreendida entre os cinco e nove anos e é *single-player*. Através do dispositivo *Leap Motion*, o jogador controla o avião com movimentos dos membros superiores, alcançando os diversos bónus e checkpoints ao longo de cada percurso, o que lhe permite desbloquear novas ilhas e componentes do avião. O jogo está disponível para *Windows*, *Mac OS* e *Linux*.

Figura 10 - Fotografia de uma criança a jogar

3.2.2 Requisitos funcionais

- Gráficos 3D;
- Gestão de perfis de jogador;
- Aumento automático da dificuldade;
- Avião com *upgrades*;
- Controlo do avião através do *Leap Motion* com as duas mãos em simultâneo*;*
- Aquisição dos dados da posição tridimensional de cada mão durante o jogo;
- Apresentação das distâncias totais percorridas por cada mão em cada eixo tridimensional através de um gráfico de barras;
- Apresentação dos valores mínimos e máximos atingidos por cada mão em cada eixo tridimensional através de um gráfico de barras e em forma 3D (campo de ação);
- Utilização de uma base de dados em *XML*;
- Utilização e controlo de som.

3.2.3 Requisitos não funcionais

Usabilidade

A interface do jogo deve estar adequada ao público-alvo, isto é, deve ser simples e intuitiva tendo em conta que o utilizador final é uma criança. Por sua vez, também o menu deve ter as mesmas características para fácil utilização pelo terapeuta.

Uma vez que as crianças em questão podem ter problemas de lateralidade e direccionalidade, o percurso do jogo tem de ser bem sinalizado, dirigido percetível.

Design

A escolha de cores deve ser cuidadosa tendo em conta o público-alvo. Uma vez que a cor tem preponderância no comportamento da criança face ao jogo, é necessário o cuidado de optar por de cores apelativas e estimulantes, evitando-se o uso excessivo das cores monocromáticas e agressivas. (Naranjo-Bock, 2011)

Competitividade

Face à dificuldade na realização de diversas tarefas, a frustração é frequente no público-alvo. Deste modo, a utilização de obstáculos e a promoção de competitividade devem ser cuidadosas e evitadas, permitindo assim às crianças obter satisfação ao longo do jogo, não tornando este último em mais uma fonte de frustração.

3.2.4 Casos de uso

Nos casos de uso do jogo "À Descoberta das Ilhas" foram identificados dois atores: o terapeuta e a criança. O terapeuta é responsável pela interação com o menu e a criança com o controlo do avião nos cenários das ilhas. Primeiro serão descritos os casos de uso do terapeuta e depois da criança.

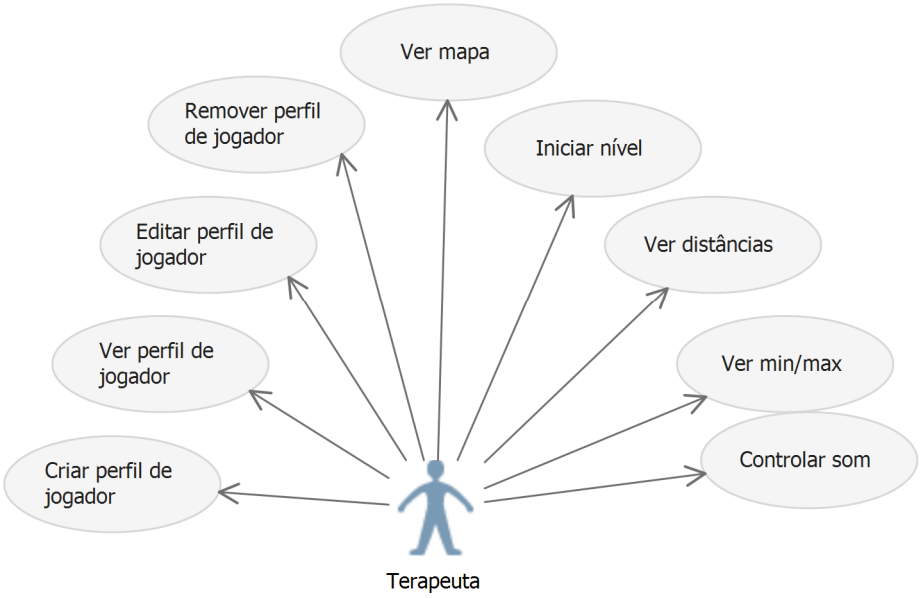


Figura 11 - Diagrama de casos de uso do terapeuta

3.2.4.1 Caso de Uso "Criar perfil de jogador"

Breve descrição

Esta funcionalidade permite ao terapeuta criar o perfil de um novo jogador na base de dados.

Descrição estruturada

Tabela 5 - Descrição estruturada do caso de uso "Criar perfil de jogador"

Ator Principal	**Terapeuta**
Pré-condições	Jogo instalado no computador.
Cenário de sucesso principal	1. O terapeuta seleciona a opção "Criar Jogador"; 2. O sistema apresenta o respetivo ecrã; 3. O terapeuta introduz o nome, mão dominante e género da criança e seleciona a opção "Gravar"; 4. O sistema grava os dados na base de dados e apresenta o ecrã inicial com a grelha de perfis de jogador atualizada.
Caminhos alternativos	*a. A qualquer momento o terapeuta pode cancelar o processo e voltar ao ecrã inicial selecionando a opção "Voltar".

Diagrama de sequência do sistema

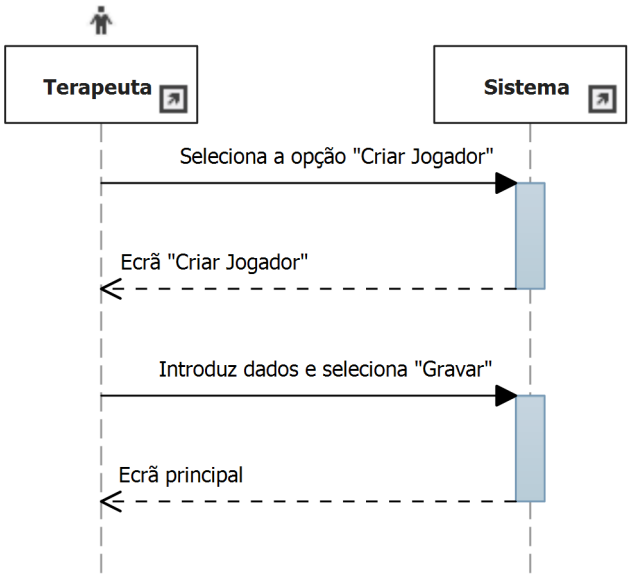


Figura 12 - Diagrama de sequência do sistema para o caso de uso "Criar perfil de jogador"

3.2.4.2 Caso de uso "Ver perfil de jogador"

Breve descrição

Esta funcionalidade permite ao terapeuta ver o perfil de um jogador existente na base de dados.

Descrição estruturada

Tabela 6 - Descrição estruturada do caso de uso "Ver perfil de jogador"

Ator Principal	**Terapeuta**
Pré-condições	Jogo instalado no computador.
Cenário de sucesso principal	1. O terapeuta seleciona o jogador; 2. O sistema apresenta o respetivo ecrã.
Cenário alternativo	*a. A qualquer momento o terapeuta pode sair do jogo selecionando a opção "Sair".

Diagrama de sequência do sistema

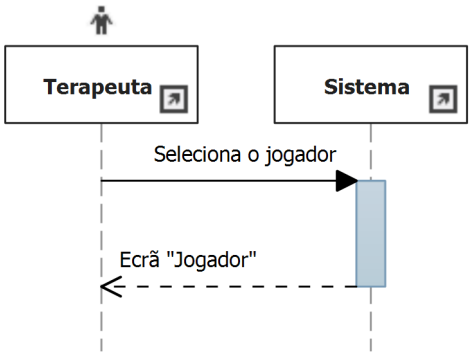


Figura 13 - Diagrama de sequência do sistema para o caso de uso "Ver perfil do jogador"

3.2.4.3 Caso de uso "Editar perfil de jogador"

Breve descrição

Esta funcionalidade permite ao terapeuta editar o perfil de um jogador existente na base de dados.

Descrição estruturada

Tabela 7 - Descrição estruturada do caso de uso "Editar perfil de jogador"

Ator Principal	**Terapeuta**
Pré-condições	Jogo instalado no computador. Perfil de jogador selecionado (caso de uso "Ver perfil de jogador").
Cenário de sucesso principal	1. O terapeuta seleciona a opção "Editar"; 2. O sistema apresenta o respetivo ecrã com os dados pré preenchidos do jogador selecionado. 3. O terapeuta altera os dados pretendidos e seleciona a opção "Gravar". 4. O sistema grava os dados na base de dados e apresenta o ecrã inicial com a grelha de perfis de jogador atualizada.
Cenário alternativo	*a. A qualquer momento o terapeuta pode cancelar o processo e voltar ao ecrã inicial selecionando a opção "Voltar".

Diagrama de sequência do sistema

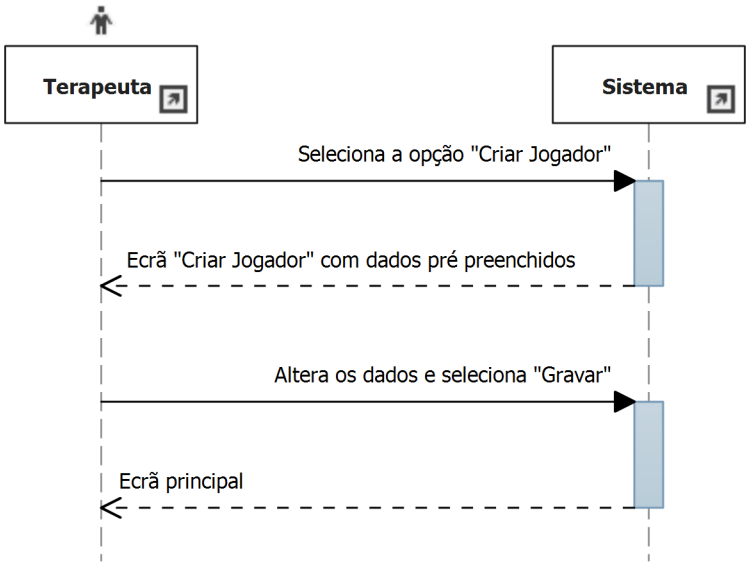


Figura 14 - Diagrama de sequência do sistema para o caso de uso "Editar perfil do jogador"

3.2.4.4 Caso de uso “Remover perfil de jogador”

Breve descrição

Esta funcionalidade permite ao terapeuta remover o perfil de um jogador existente na base de dados.

Descrição estruturada

Tabela 8 - Descrição estruturada do caso de uso “Remover perfil de jogador”

Ator Principal	**Terapeuta**
Pré-condições	Jogo instalado no computador. Perfil de jogador selecionado (caso de uso “Ver perfil de jogador”).
Cenário de sucesso principal	1. O terapeuta seleciona a opção “Remover”; 2. O sistema apresenta uma solicitação de confirmação ao terapeuta. 3. O terapeuta confirma a ação. 4. O sistema remove os dados na base de dados e apresenta o ecrã inicial com a grelha de perfis de jogador atualizada.
Cenário alternativo	*a. A qualquer momento o terapeuta pode cancelar o processo e voltar ao ecrã inicial selecionando a opção “Voltar”.

Diagrama de sequência do sistema

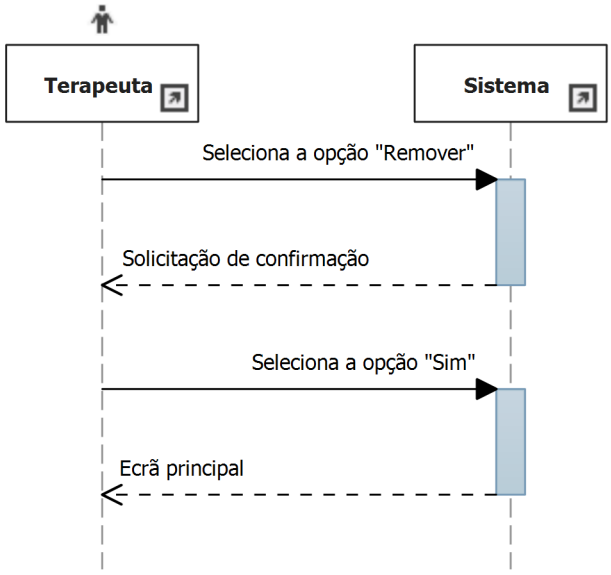


Figura 15 - Diagrama de sequência do sistema para o caso de uso "Remover perfil do jogador"

3.2.4.5 Caso de uso “Ver mapa”

Breve descrição

Esta funcionalidade permite ao terapeuta ver o mapa de jogo de um jogador existente na base de dados com os dados (bónus) dos níveis conquistados até ao momento e o próximo nível disponível.

Descrição estruturada

Tabela 9 - Descrição estruturada do caso de uso "Ver mapa"

Ator Principal	**Terapeuta**
Pré-condições	Jogo instalado no computador. Perfil de jogador selecionado (caso de uso “Ver perfil de jogador”).
Cenário de sucesso principal	1. O terapeuta seleciona a opção “Ver Mapa”; 2. O sistema apresenta o respetivo ecrã.
Cenário alternativo	*a. A qualquer momento o terapeuta pode sair do jogo selecionando a opção “Sair”.

Diagrama de sequência do sistema

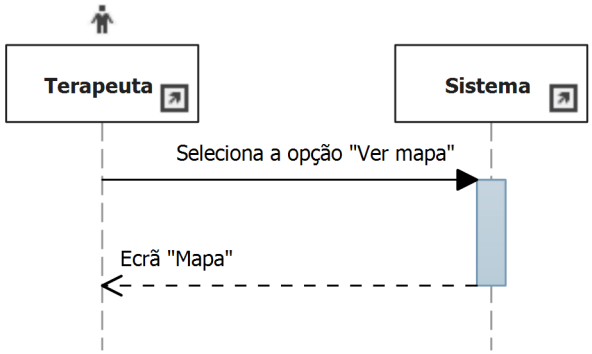


Figura 16 - Diagrama de sequência do sistema para o caso de uso "Ver mapa"

3.2.4.6 Caso de uso “Iniciar nível”

Breve descrição

Esta funcionalidade permite ao terapeuta iniciar o nível de um jogador existente na base de dados. No fim deste caso de uso é apresentado o cenário de jogo para criança a jogar. A mesma é remetida para o caso de uso “Calibrar mãos”.

Descrição estruturada

Tabela 10 - Descrição estruturada do caso de uso "Iniciar nível"

Ator Principal	**Terapeuta**
Pré-condições	Jogo instalado no computador. Mapa de jogo de jogador selecionado (caso de uso “Ver mapa”).
Cenário de sucesso principal	1. O terapeuta seleciona o nível desejado para a criança jogar; 2. O sistema apresenta o respetivo cenário.
Cenário alternativo	*a. A qualquer momento o terapeuta pode voltar ao ecrã de perfil do jogador selecionando a opção “Voltar”.

Diagrama de sequência do sistema

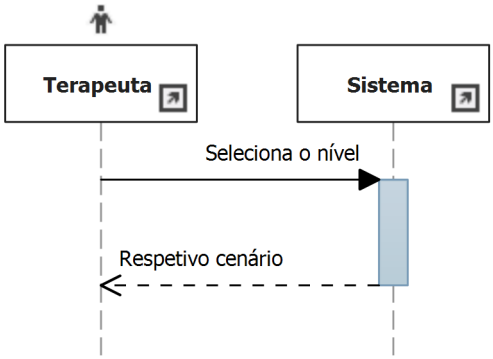


Figura 17 - Diagrama de sequência do sistema para o caso de uso "Iniciar nível"

3.2.4.7 Caso de uso “Ver distâncias”

Breve descrição

Esta funcionalidade permite ao terapeuta ver os valores das distâncias totais percorridas por mão em cada eixo tridimensional de um jogador existente na base de dados. Os valores são apresentados em forma de gráfico de barras e o utilizador dispõe das opções “X” (selecionado por defeito), “Y” e “Z” para selecionar o respetivo eixo do qual pretende ver as distâncias.

Descrição estruturada

Tabela 11 - Descrição estruturada do caso de uso "Ver distâncias"

Ator Principal	**Terapeuta**
Pré-condições	Jogo instalado no computador. Perfil de um jogador selecionado (caso de uso “Ver perfil de jogador”).
Cenário de sucesso principal	1. O terapeuta seleciona a opção “Progresso”. 2. O sistema apresenta o respetivo ecrã. 3. O terapeuta seleciona a opção “Distâncias”; 4. O sistema apresenta um gráfico de barras com os valores do eixo X; 5. O terapeuta altera o eixo tridimensional pretendido; 6. O sistema apresenta um gráfico de barras com os valores do eixo selecionado. O terapeuta repete os passos 3-4 as vezes que pretender.
Cenário alternativo	*a. A qualquer momento o terapeuta pode voltar ao ecrã de progresso do jogador selecionando a opção “Voltar”.

Diagrama de sequência do sistema

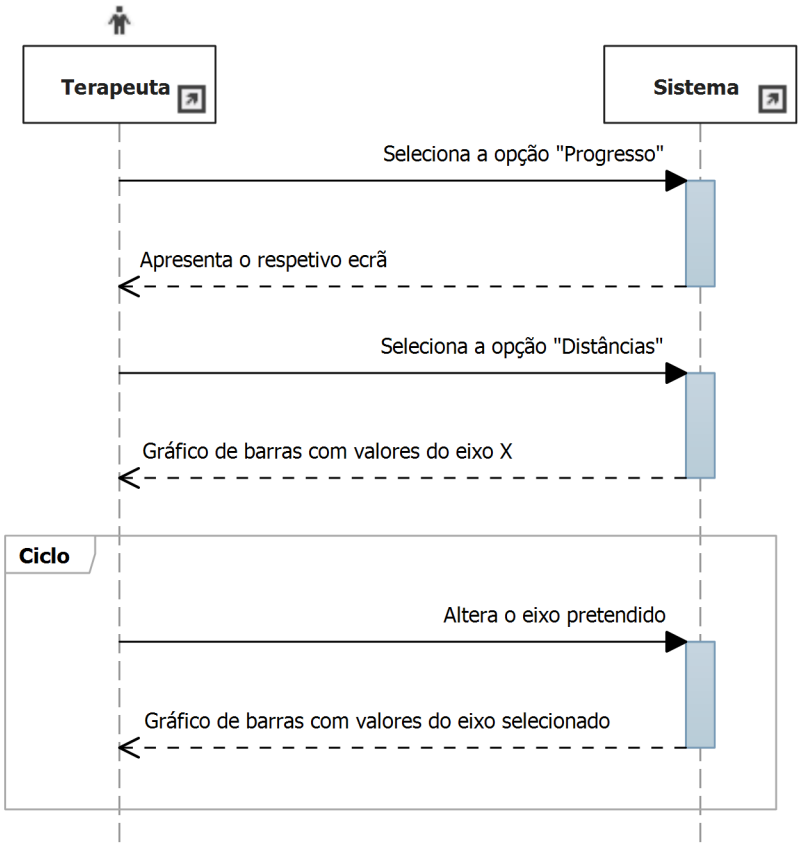


Figura 18 - Diagrama de sequência do sistema para o caso de uso “Ver distâncias”

3.2.4.8 Caso de uso “Ver min/max”

Breve descrição

Esta funcionalidade permite ao terapeuta ver os valores mínimos e máximos atingidos por mão em cada eixo tridimensional de um jogador existente na base de dados. Os valores podem ser apresentados em forma de gráfico de barras e em forma 3D (campo de ação). No gráfico de barras, o terapeuta dispõe das opções “X” (selecionado por defeito), “Y” e “Z” para selecionar o respetivo eixo do qual pretende ver os valores. No campo de ação, o terapeuta dispõe de opções para incrementar e decrementar o nível e a tentativa dos quais pretende ver os valores, assim como pode rodar horizontalmente e aproximar e afastar todo o campo de ação. A ação de rotação executa-se “arrastando” todo o campo de ação para a esquerda ou direita, aplicando-lhe uma rotação no eixo Y na direção pretendida e a ação de aproximar ou afastar executa-se através do *scroll* do rato.

Descrição estruturada

Tabela 12 - Descrição estruturada do caso de uso "Ver min/max"

Ator Principal	**Terapeuta**
Pré-condições	Jogo instalado no computador. Perfil de um jogador selecionado (caso de uso “Ver perfil de jogador”).
Cenário de sucesso principal	1. O terapeuta seleciona a opção “Progresso”. 2. O sistema a presenta o respetivo ecrã. 3. O terapeuta seleciona a opção “Min/Max”; 4. O sistema apresenta um gráfico de barras com os valores do eixo X; 5. O terapeuta altera o eixo tridimensional pretendido; 6. O sistema apresenta um gráfico de barras com os valores do eixo selecionado; 7. O terapeuta altera a forma de representação de valores pretendida; 8. O sistema apresenta a forma de representação de valores selecionada; 9. O terapeuta altera o nível e tentativa pretendidos; 10. O sistema apresenta os valores para a seleção pretendida; 11. O terapeuta seleciona e roda horizontalmente e aproxima ou afasta o campo de ação; 12. O sistema aplica ao campo de ação a rotação no eixo Y e movimentação no eixo Z pretendida; O terapeuta repete os passos 3-4, 5-6, 7-8 e 9-10 as vezes que pretender.
Cenário alternativo	*a. A qualquer momento o terapeuta pode voltar ao ecrã de progresso do jogador selecionando a opção “Voltar”.

Diagrama de sequência do sistema

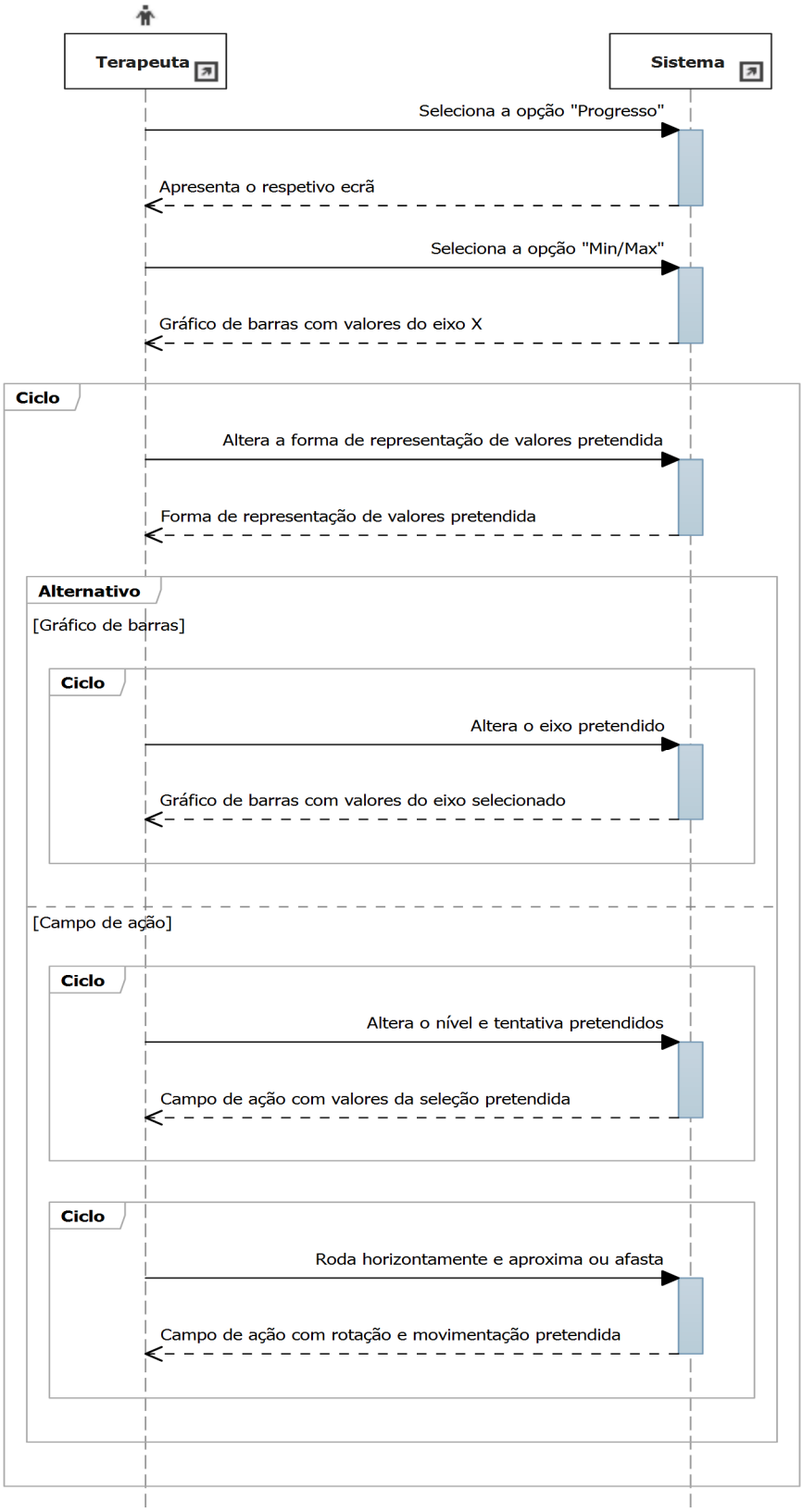


Figura 19 - Diagrama de sequência do sistema para o caso de uso "Ver min/max"

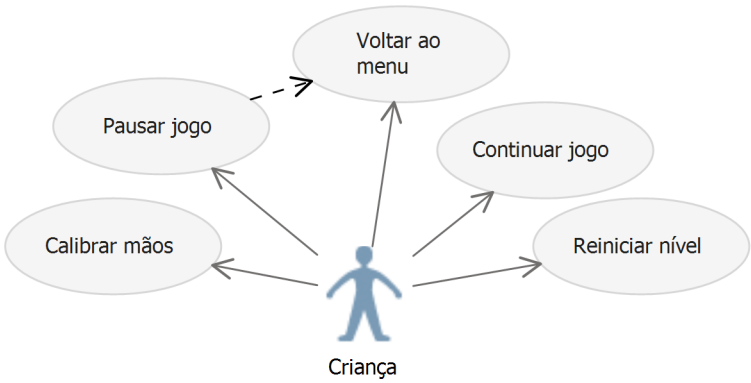


Figura 20 - Diagrama de casos de uso da criança

3.2.4.9 Caso de uso "Calibrar mãos"

Breve descrição

Esta funcionalidade permite à criança calibrar as mãos no início de cada nível, nivelando-as horizontalmente e posicionando-as no centro da área captável pelo *Leap Motion*. No fim deste caso de uso, a criança começa a controlar o avião.

Descrição estruturada

Tabela 13 - Descrição estruturada do caso de uso "Calibrar mãos"

Ator Principal	**Criança**
Pré-condições	Jogo instalado no computador. Leap Motion. Nível selecionado (caso de uso "Iniciar nível").
Cenário de sucesso principal	1. A criança posiciona ambas as mãos nas zonas indicadas graficamente; 2. O sistema apresenta uma notificação que as mãos estão na zona correta; 3. A criança espera três segundos com as mãos na mesma posição; 4. O sistema inicia o jogo. A criança repete os paços 1-2 até conseguir posicionar corretamente as mãos três segundos.
Cenário alternativo	*a. A qualquer momento a tecla "c" é pressionada. 1. O sistema inicia o jogo.

Diagrama de sequência do sistema

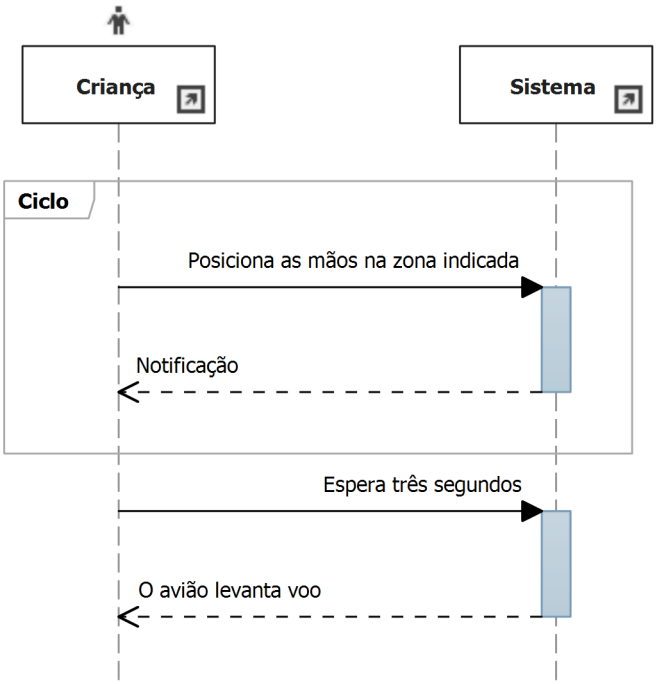


Figura 21 - Diagrama de sequência do sistema para o caso de uso "Calibrar mãos"

3.2.4.10 Caso de uso “Pausar jogo”

Breve descrição

Esta funcionalidade permite à criança fazer uma pausa no nível em que se encontra. No fim deste caso de uso, o avião é parado e a criança deixa de o controlar.

Descrição estruturada

Tabela 14 - Descrição estruturada do caso de uso "Pausar jogo"

Ator Principal	**Criança**
Pré-condições	Jogo instalado no computador. Leap Motion. Nível selecionado (caso de uso “Iniciar nível”).
Cenário de sucesso principal	5. A criança fecha ambas as mãos; 6. O sistema pausa o jogo e apresenta o respetivo ecrã.
Cenário alternativo	1a. A tecla “Esc” é pressionada.

Diagrama de sequência do sistema

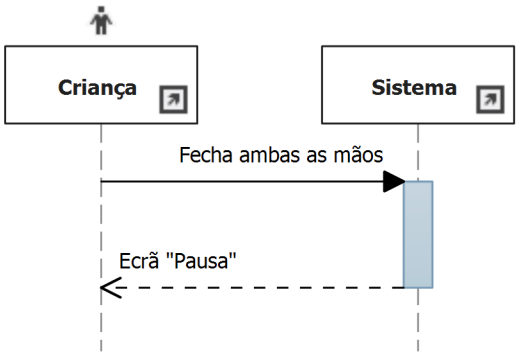


Figura 22 - Diagrama de sequência do sistema para o caso de uso "Pausar jogo"

3.2.4.11 Caso de uso “Voltar ao menu”

Breve descrição

Esta funcionalidade permite à criança voltar ao menu após ter ficado sem combustível quatro vezes ou após ter chegado ao fim do nível. No fim deste caso de uso, o terapeuta é remetido para o caso de uso “Ver mapa”. Se o nível foi completado com sucesso, são apresentados os bónus conquistados pela criança (moedas e estrelas) e todos os dados são gravados na base de dados.

Descrição estruturada

Tabela 15 - Descrição estruturada do caso de uso "Voltar ao menu"

Ator Principal	**Criança**
Pré-condições	Jogo instalado no computador. Nível selecionado (caso de uso “Iniciar nível”). Avião sem combustível e fim das quatro vidas do nível ou final do nível.
Cenário de sucesso principal	1. A criança seleciona a opção “Voltar ao Menu”; 2. O sistema apresenta o menu, os resultados do nível e grava os dados (se o nível foi completado).

Diagrama de sequência do sistema

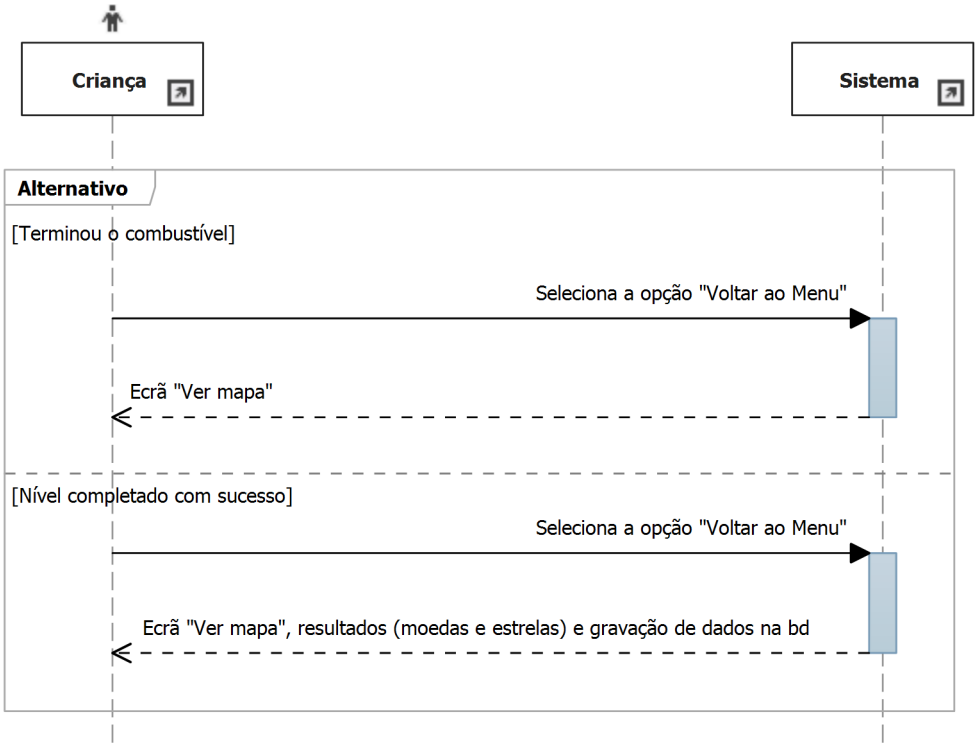


Figura 23 - Diagrama de sequência do sistema para o caso de uso "Voltar ao menu"

3.2.4.12 Caso de uso “Continuar jogo”

Breve descrição

Esta funcionalidade permite à criança continuar a jogar o nível em que se encontra após ter efetuado uma pausa. No fim deste caso de uso, o avião move-se novamente a criança volta a controlá-lo.

Descrição estruturada

Tabela 16 - Descrição estruturada do caso de uso "Continuar jogo"

Ator Principal	**Criança**
Pré-condições	Jogo instalado no computador. *Leap Motion*. Jogo pausado (caso de uso “Pausar jogo”).
Cenário de sucesso principal	1. A criança seleciona a opção “Continuar”; 2. O sistema continua o jogo.

Diagrama de sequência do sistema

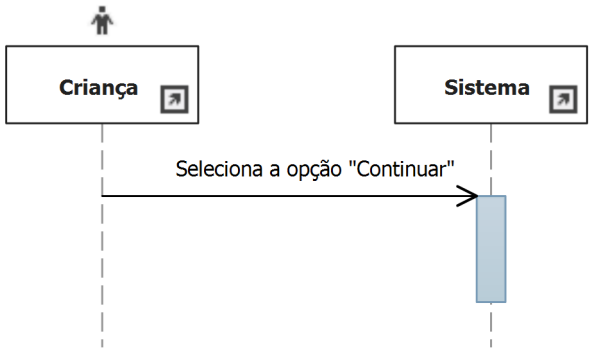


Figura 24 - Diagrama de sequência do sistema para o caso de uso "Continuar jogo"

3.2.4.13 Caso de uso “Reiniciar nível”

Breve descrição

Esta funcionalidade permite à criança reiniciar o nível em que se encontra após ter ficado sem combustível quatro vezes. No fim deste caso de uso, a criança é remetida para o caso de uso “Calibrar mãos”.

Descrição estruturada

Tabela 17 - Descrição estruturada do caso de uso "Reiniciar nível"

Ator Principal	**Criança**
Pré-condições	Jogo instalado no computador. Nível selecionado (caso de uso “Iniciar nível”). Avião sem combustível e fim das quatro vidas do nível.
Cenário de sucesso principal	1. A criança seleciona a opção “Reiniciar”; 2. O sistema reinicia o nível atual.
Cenário alternativo	*a. A qualquer momento a criança pode voltar ao menu inicial selecionando a opção “Voltar ao Menu”.

Diagrama de sequência do sistema

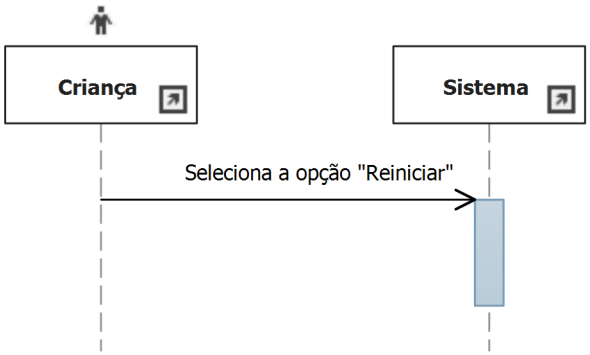


Figura 25 - Diagrama de sequência do sistema para o caso de uso "Reiniciar nível"

3.2.5 Regras e componentes do jogo

No fim da análise estavam reunidas as condições para serem definidas as regras do jogo.

Cada jogador tem à sua disposição um simples avião para começar a jogar na vista de terceira pessoa. Depois de criado e selecionado o perfil do jogador, no mapa de jogo é apresentada a próxima ilha disponível e as ilhas conquistadas até ao momento, podendo o jogador repeti-las vezes ilimitadas.

Para completar um nível (conquista da ilha), o jogador tem de alcançar sequencialmente 10 *checkpoints*. Os mesmos permitem encher o depósito de combustível que o avião necessita para se mover. Quando o jogador inicia um nível dispões de quatro tentativas: a que começa mais três vidas. Caso o combustível termine, o jogador perde uma vida e recomeça no último *checkpoint* alcançado. Se perder as quatro vidas, todo o progresso da tentativa do nível será perdido.

Em cada nível existem também três estrelas. Para desbloquear a próxima ilha, o jogador necessita de 2/3 das estrelas existentes até ao nível corrente. Para não ser possível avançar um nível sem o jogar, para além da regra anterior, é também necessário alcançar, no mínimo, uma estrela por nível.

Entre cada dois *checkpoints* existem três moedas de jogo. Logo, existem no total 27 moedas de jogo por nível. As moedas de jogo permitem ao utilizador fazer *upgrades* ao avião. Os *upgrades* são feitos automaticamente e são adquiridos como descrito na seguinte tabela:

Tabela 18 - Sistema de aquisição de *upgrades* do avião

ID	Moedas de jogo necessárias	*Upgrade*
1	40	Cor
2	95	Pala traseira e beiras nas asas
3	149	Um motor em cada asa
4	203	Fumos nos motores
5	257	Dois motores em cada asa
6	311	Um motor na pala traseira

Para uma melhor visualização de todos os componentes do jogo é apresentado o respetivo diagrama na Figura 26.

Figura 26 - Diagrama de componentes do jogo

O jogador apenas pode obter uma vez cada moeda de jogo e cada estrela. As que não forem obtidas ficam disponíveis na mesma posição sempre que o nível for repetido. É importante voltar a salientar que apenas são gravadas as moedas e estrelas alcançadas se o nível for conquistado com sucesso.

Cada ilha é representada por um cenário independente. O número de cenários é limitado mas o número de níveis não. Logo, todos os cenários são percorridos sequencialmente num ciclo infinito. Por exemplo, se existirem quatro ilhas, o quinto nível do jogo terá o mesmo cenário que o primeiro.

Os *checkpoints* são colocados no cenário dinamicamente, segundo um algoritmo dependente do nível. À medida que se vai avançando no jogo a dificuldade vai aumentando pois o tamanho e disposição dos *checkpoints* dificulta o seu alcance.

O tempo necessário para que o jogador complete um nível é de aproximadamente um minuto e meio e um depósito combustível dá para 25 segundos.

3.2.6 Competências Transversais

O jogo desenvolvido no projeto é individual e orientado quer pelo terapeuta numa fase inicial, quer virtualmente através de sinalética (setas, ver Figura 33). Pelo seu desenho dirigido, promove a orientação e o foco seletivo face aos objetos (*checkpoints*) e em simultâneo promove a capacidade motora de estabilização / equilíbrio e direção através do dispositivo *Leap Motion*. O mesmo foi concebido para promover a atenção, a assimilação e acomodação da lateralidade e direccionalidade (esquerda e direita) e a integração motora dos movimentos. Deste modo, consideram-se reunidas condições para a aquisição de competências ao nível da coordenação bilateral e viso-motora bem como do equilíbrio e estabilidade, todos necessários para a utilização do *Leap Motion*.

Pretende-se então estimular a atenção e agilidade ao mesmo tempo que se motiva para uma tomada de consciência de autoeficácia na integração bilateral motora, experienciada nos mais variados contextos, nomeadamente o escolar.

A criança, ao nível do jogo aprende, num primeiro momento, seguindo as orientações do terapeuta e condicionada pelas regras do jogo, e num segundo momento pela validação cognitiva da sua própria experiência. Pelo que se realça a importância das perspetivas psicológicas de aprendizagem, quer comportamentais, quer as cognitivas construtivistas nesta interação lúdica entre criança e mundo virtual.

3.3 Desenho

Nesta secção será descrita a fase de desenho, posterior à de análise, onde a arte e a criatividade mais se evidenciam. Esta fase do projeto é também bastante trabalhosa e extensiva, justificando assim a sua apresentação detalhada.

Depois de definidos os requisitos e o conceito do jogo foi desenhada a arquitetura e foram iniciados os esboços que permitiram a transição do conceito à prática.

O desenvolvimento do projeto foi efetuado maioritariamente num ecrã com a resolução 1440x900 e, uma vez que os requisitos mínimos de um computador para suportar o dispositivo *Leap Motion* são consideravelmente bons, não foi estabelecida uma resolução adequada para o jogo.

Uma vez que o jogo utiliza o *Leap Motion* para o controlo do avião, foi decidido que a navegação dos menus seria também efetuada com este dispositivo. Como os menus vão ser acedidos pelo terapeuta para análise de dados, a utilização do rato pode ser mais prática e, por isso, optou-se também pelo suporte deste último dispositivo. Portanto, a interação com os menus pode ser feita através de ambos os dispositivos. Para inserção de dados do jogador, é necessário também um teclado.

Uma fácil e intuitiva interação é crucial para a utilização de qualquer *software*. Se a primeira experiência do utilizador for frustrante, complexa ou cansativa, este acaba por desistir mesmo antes de chegar ao essencial da aplicação.

O desenvolvimento de interfaces para *software* que utilize o *Leap Motion* requer uma atenção especial, portanto, foram analisadas e seguidas as diretrizes de boas práticas disponíveis no site da própria marca. (Motion Leap, sem data)

Depois desta análise e da experiência em várias aplicações existentes, foram tidas em consideração as seguintes características: dimensão dos botões e alinhamento dos mesmos - os principais ao centro e os secundários nos cantos do ecrã.

De seguida é apresentada a fase de desenho dos cenários das ilhas e do menus, pela mesma ordem que foi efetuada na elaboração deste projeto.

3.3.1 Cenários das ilhas

A fase de desenho do jogo foi iniciada pelo esboço do avião (ver Figura 27). Sendo este o componente principal do jogo, o seu desenho foi auxiliado por um *designer* profissional.

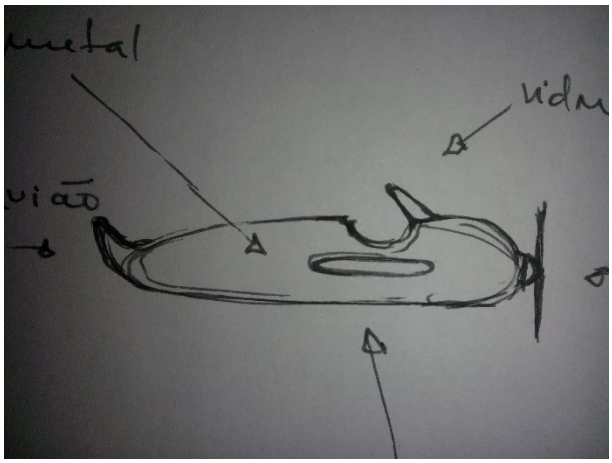


Figura 27 - Esboço do avião

Depois do esboço do avião foi iniciada a sua modelação 3D (ver Figura 28), assim como a dos outros componentes do jogo que, devido à sua forma simples, foram diretamente desenhados em formato digital.

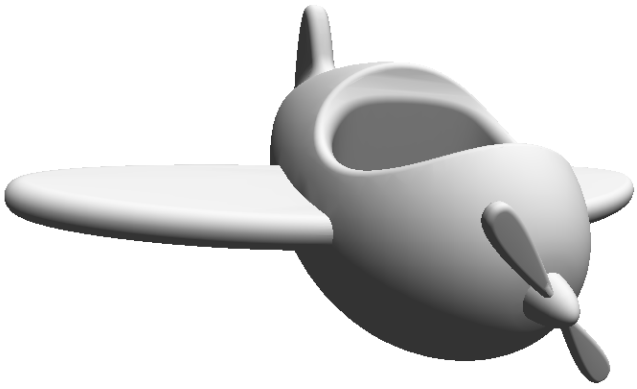


Figura 28 - Avião 3D

A maioria dos objetos 3D do jogo foi modelada com o *software Blender*, sendo os restantes construídos no *Unity 3D* devido às suas características simples. Depois de terminados os objetos, exportaram-se para o formato .obj e subsequentemente importados no *Unity 3D*.

As personagens masculina e feminina são o mesmo objeto 3D apenas alterando a cor do capacete: castanho no primeiro caso e rosa no segundo.

A Figura 29 apresenta a construção da personagem e as Figuras 30-33 apresentam a última versão dos bónus (estrela e moeda), *checkpoint* e setas indicadoras do próximo *checkpoint* a alcançar. Todos estes objetos foram modelados no *Blender*.

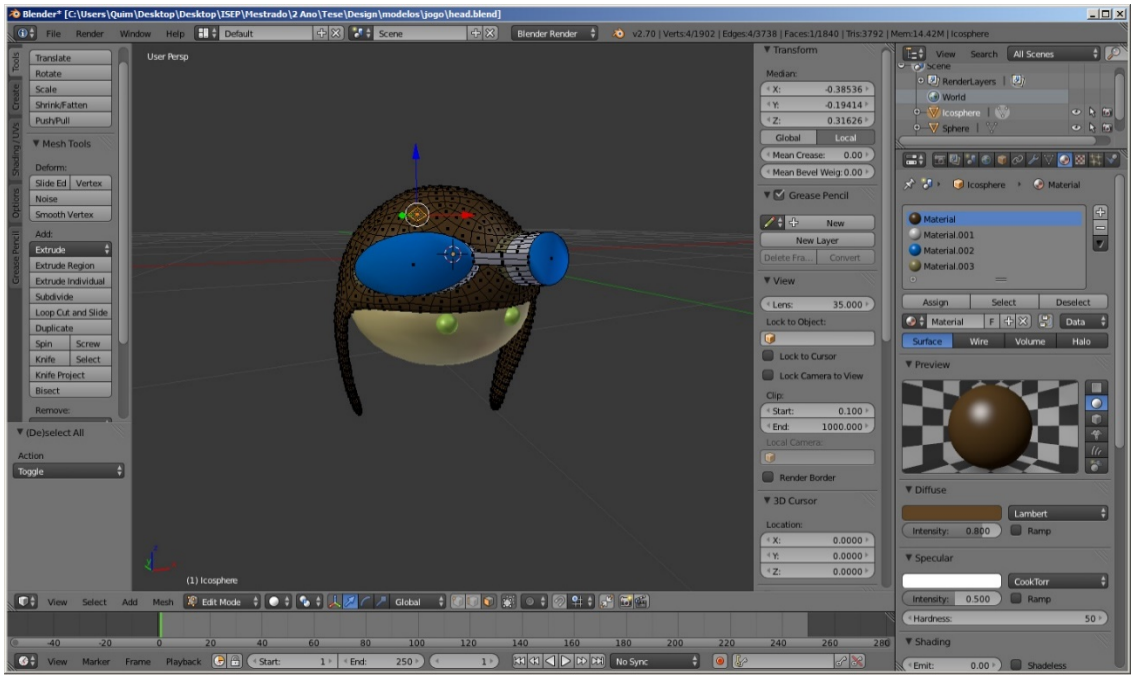


Figura 29 - Construção da personagem no Blender

Figura 30 - Bónus estrela

Figura 31 - Bónus moeda

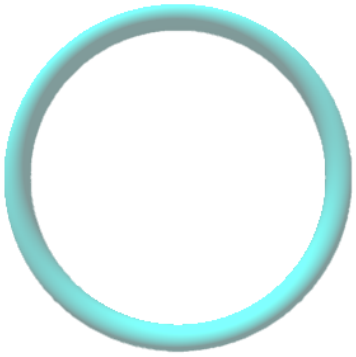


Figura 32 – *Checkpoint*

Figura 33 - Setas de indicação do próximo *checkpoint* a alcançar

Uma vez que os *checkpoint* têm de ser alcançados sequencialmente, foram definidos três estados do *checkpoint*: normal, próximo a alcançar e alcançado. Para uma melhor perceção da criança foram utilizadas diferentes cores para estes estados, representados nas Figuras 32, 34 e 35, respetivamente.

Figura 34 - Próximo *checkpoint* a alcançar

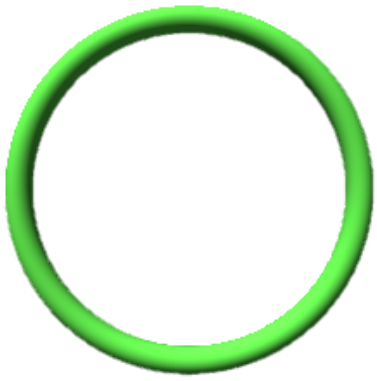


Figura 35 - *Checkpoint* alcançado

No caso das ilhas, estas foram construídas na *IDE Unity 3D* com a sua ferramenta de desenho de terrenos. Esta ferramenta permite, através de um plano, elevar ou aplanar áreas, aplicar texturas e acrescentar árvores e ervas por zonas. Na Figura 36 é apresentada a modelação da primeira ilha do jogo, com texturas de areia e pedra e com árvores e arbustos. Ambas as texturas e elementos da natureza foram importados da loja do *Unity 3D*, que o mesmo disponibiliza para livre utilização. Ilhas pequenas, por vezes apenas montes de areia, também foram desenhadas ao longo do percurso para dar mais ânimo ao cenário.

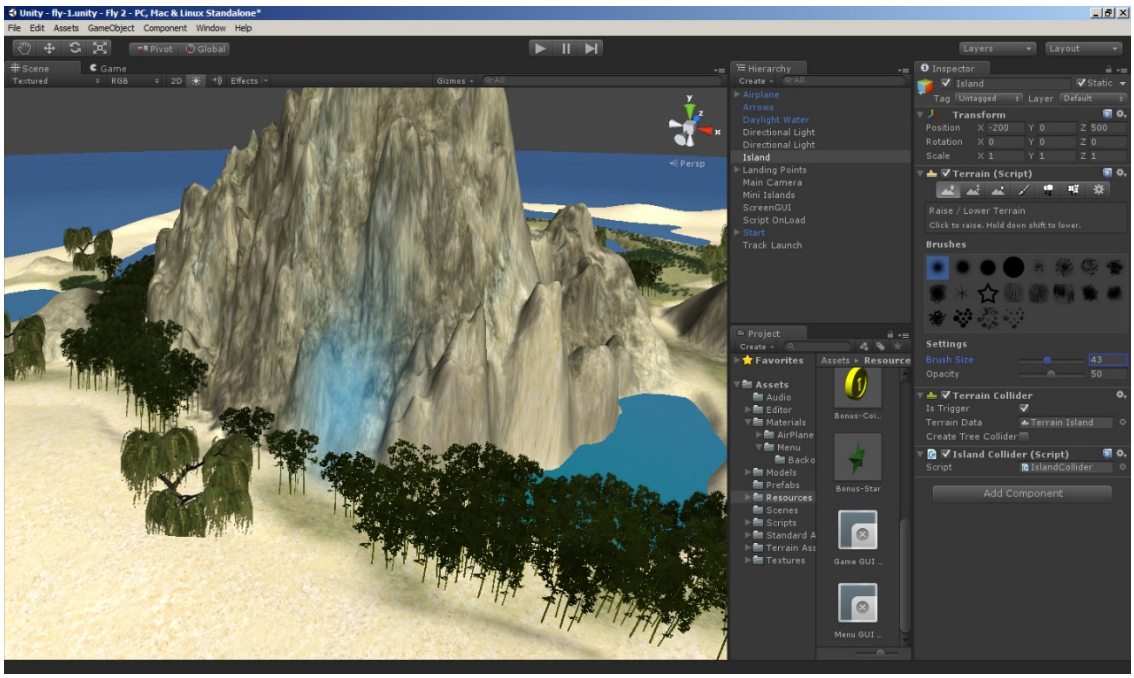


Figura 36 - Construção da primeira ilha do jogo no *Unity 3D*

De seguida são apresentadas as últimas versões dos três cenários de ilhas existentes no jogo.

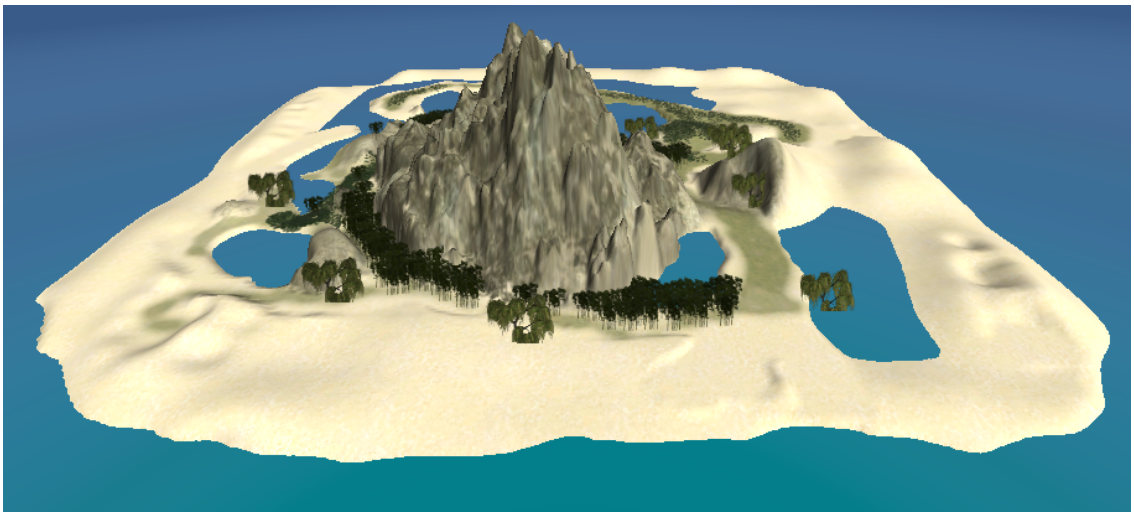


Figura 37 - Primeira ilha

Figura 38 - Segunda ilha

Figura 39 - Terceira ilha

Os *upgrades* do avião foram construídos no *Unity 3D*, após importação do objeto 3D avião. As Figuras 40-46 apresentam a evolução do avião com a personagem masculina, desde o primeiro disponível no jogo até ao último *upgrade* do mesmo.

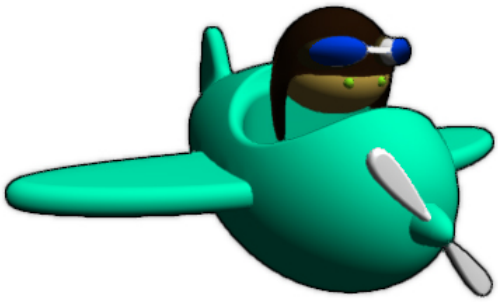


Figura 40 – Primeiro avião no jogo

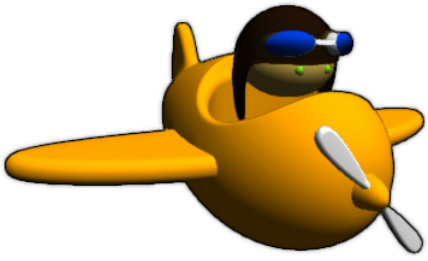


Figura 41 - Avião com o primeiro *upgrade*

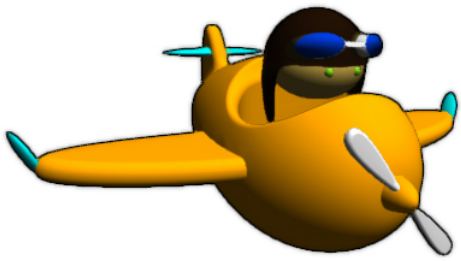


Figura 42 - Avião com o segundo *upgrade*

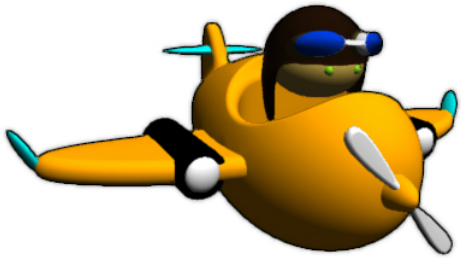


Figura 43 - Avião com o terceiro *upgrade*

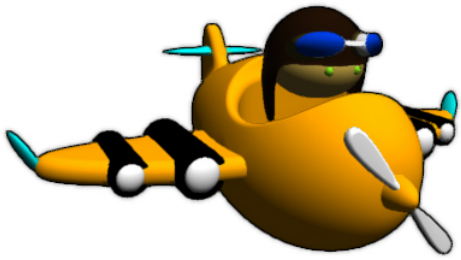


Figura 44 - Avião com o quinto *upgrade*

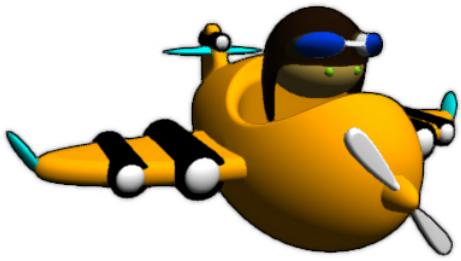


Figura 45 - Avião com o sexto *upgrade*

O quarto *upgrade* consiste na emissão de fumo pelos motores sob a forma de partículas, propriedade disponível no *Unity 3D*, como apresentado na Figura 46. Os *upgrades* seguintes somam este, ou seja, todos os motores passam a emitir fumo a partir do quarto *upgrade*.

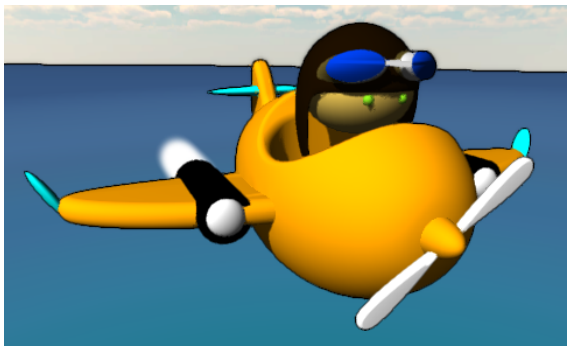


Figura 46 - Avião com o quarto *upgrade*

Para a calibração das mãos no início de cada nível foi utilizada uma textura de uma mão *cartoon*, como apresentado na Figura 47. Quando o *Leap Motion* capta a mão da criança é apresentada essa textura no ecrã que acompanha movimento da mão, cujo processo será descrito sucintamente na secção 3.4 Implementação. A fonte utilizada nos textos apresentados ao terapeuta e à criança denomina-se de *Showcard Gothic* e é gratuita.

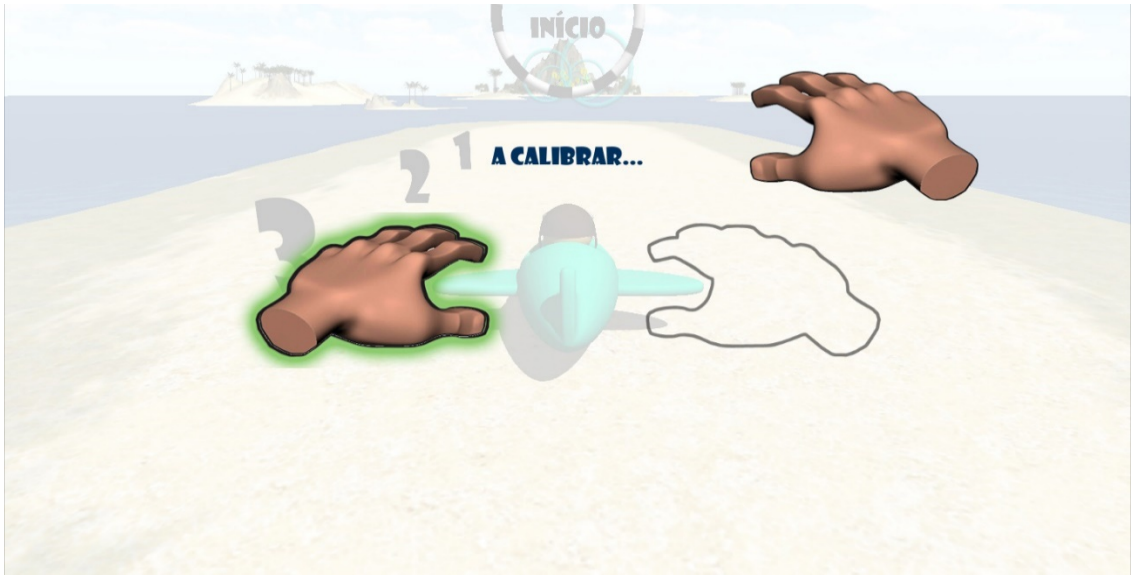


Figura 47 - Calibração das mãos

Depois do jogo estar implementado foi apresentado à psicóloga Carla Ribeiro que alertou a importância de enfatizar o início de cada nível, momento em que a criança começa a controlar o avião, depois da calibração. No seguimento deste conselho acrescentou-se uma contagem de três segundos, uma argola com a textura típica da bandeira da meta de uma corrida e o texto "início", como apresentado na Figura 48. Da mesma forma foi utilizada a mesma textura para o último *checkpoint* de cada nível.

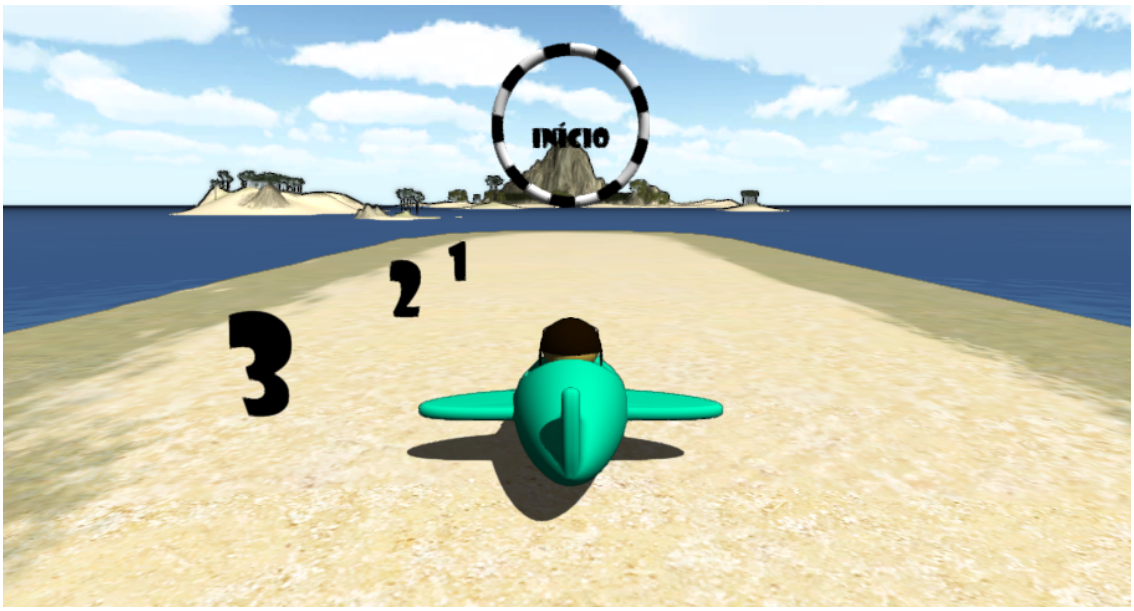


Figura 48 - Início de um nível

3.3.2 Menus

A fase de desenho dos menus foi iniciada pelo menu principal, elaborando-se um diagrama com a sua arquitetura de ecrãs (ver Figura 49).

Figura 49 - Diagrama de arquitetura de ecrãs

Após o desenho da arquitetura de ecrãs do menu principal, foi definido o fluxo de todo o jogo que o terapeuta e a criança podem seguir (ver Figura 50).

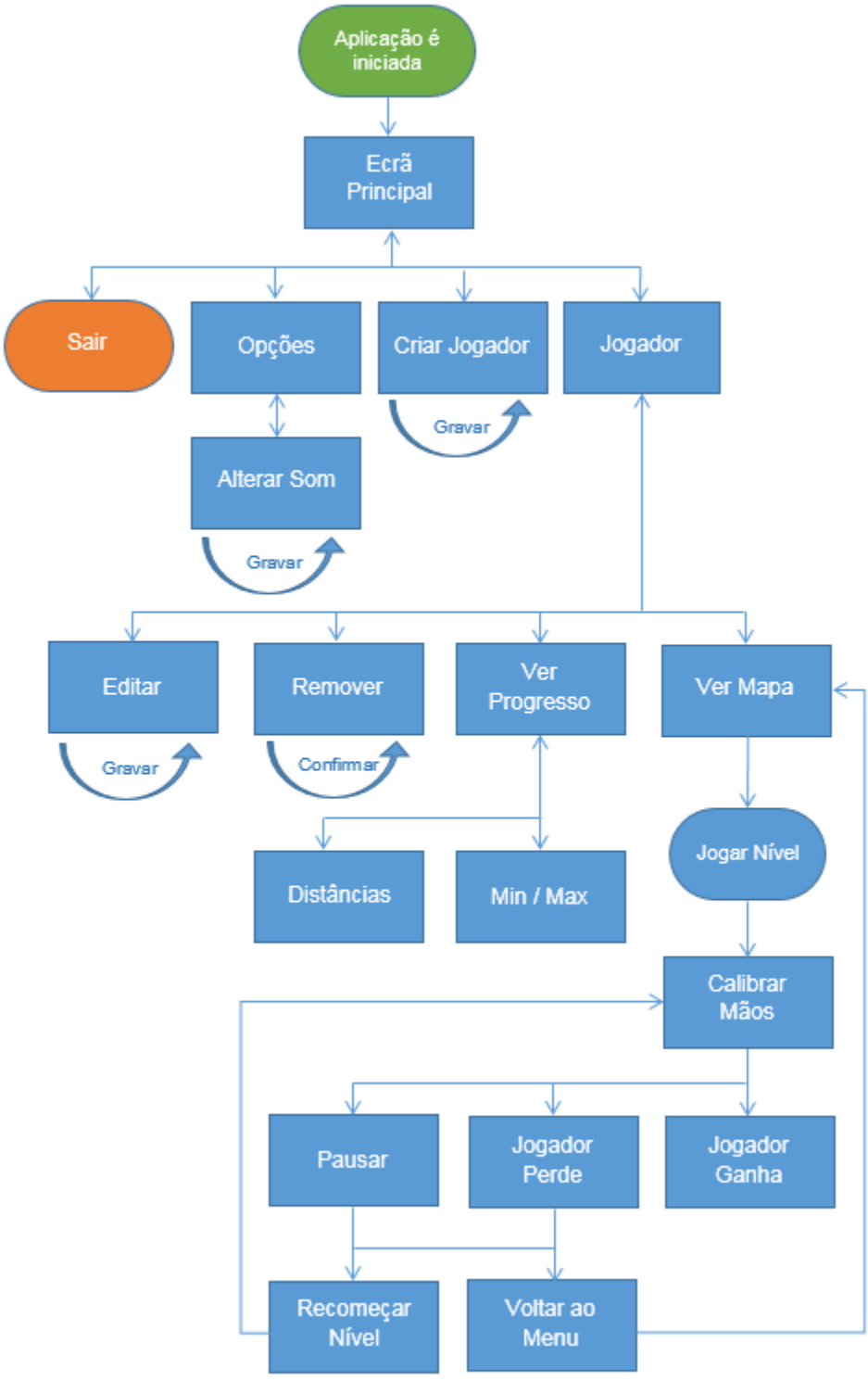


Figura 50 - Diagrama de fluxo do jogo

Depois de definida a arquitetura de ecrãs do menu e o fluxo dos menus começaram a ser desenhados à mão os respetivos esboços.

Ecrã principal

Este ecrã é o primeiro a aparecer no jogo. Neste são apresentados os jogadores existentes na base de dados, o botão "Sair" para fechar a aplicação, "Criar Jogador" e "Opções". Os ícones dos jogadores contém a personagem do seu género e são clicáveis, permitindo ao terapeuta avançar para o ecrã do jogador selecionado.

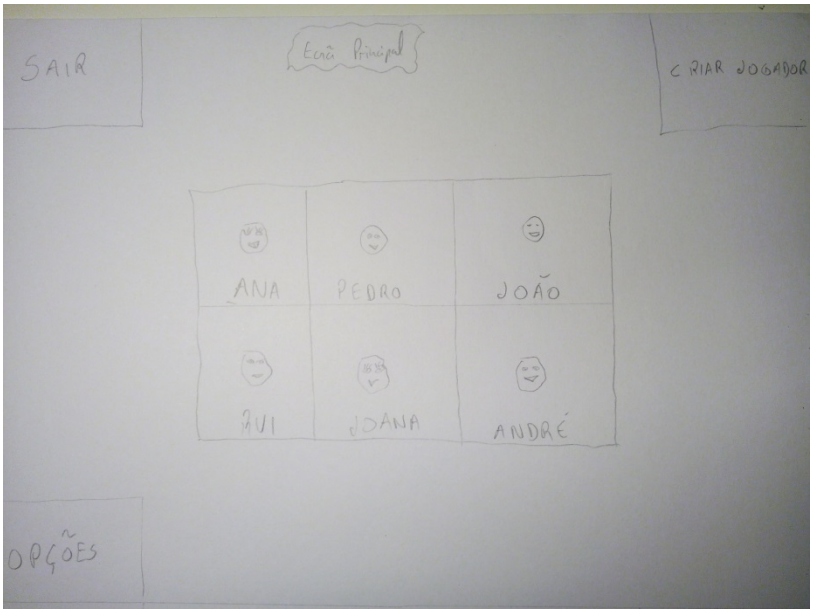


Figura 51 - Esboço o ecrã principal

Ecrã "Criar Jogador", "Editar Jogador" e "Eliminar Jogador"

Este ecrã permite ao terapeuta criar, editar ou eliminar um jogador da base de dados, consoante a opção anteriormente selecionada. Neste é apresentado um campo de texto para se introduzir/alterar (através do teclado) o nome, dois conjuntos de botões de escolha única: um para o género e outro para a mão dominante, o botão "Voltar" para voltar ao ecrã principal e o botão "Gravar" para a respetiva ação. Se a opção anteriormente selecionada foi "Criar Jogador", os botões do género masculino e da mão direita aparecem selecionados por defeito. Se a opção anteriormente selecionada foi "Editar", os dados são pré-preenchidos consoante o perfil da respetiva criança e é também apresentado ao terapeuta o botão "Eliminar" no canto inferior esquerdo para a respetiva ação.

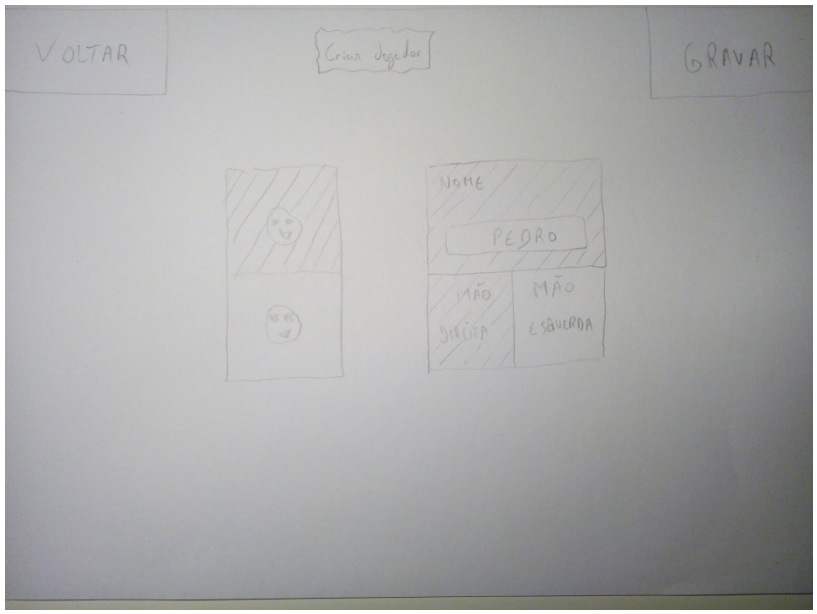


Figura 52 - Esboço do ecrã "Criar Jogador"

Ecrã "Jogador"

Este ecrã permite ao terapeuta ver o perfil do jogador selecionado anteriormente no ecrã principal. É apresentado o nome, género, mão dominante, nível atual (última ilha conquistada), número de estrelas e moedas conquistadas, o botão "Voltar" para voltar ao ecrã principal, o botão "Editar" para editar os dados do perfil do jogador e o botão "Ver Mapa" para a respetiva ação.

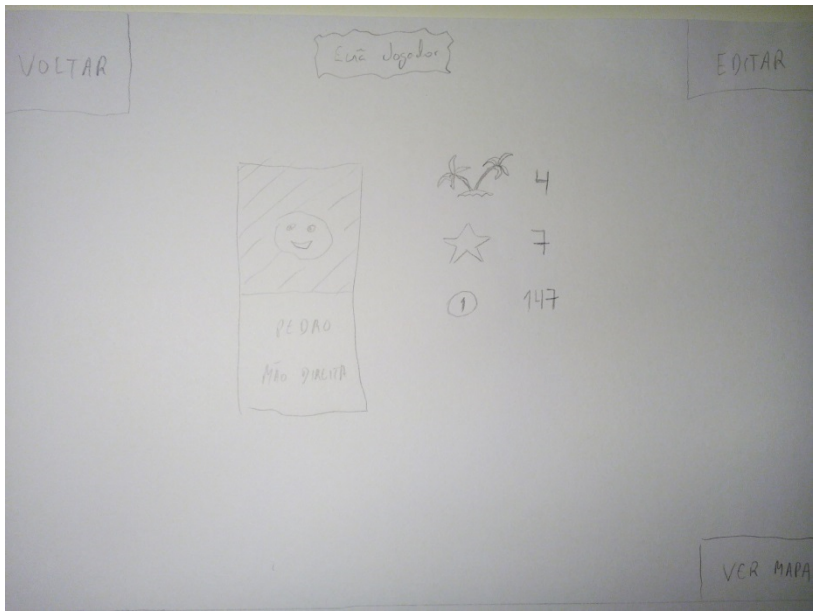


Figura 53 - Esboço do ecrã "Jogador"

Ecrã "Ver Mapa"

Este ecrã permite ao terapeuta aceder ao mapa de jogo do jogador selecionado anteriormente no ecrã principal. São apresentadas as ilhas conquistadas até ao momento, com os respetivos bonús alcançados, a próxima ilha a conquistar com o respetivo número de estrelas necessárias, o número total de estrelas conquistadas até ao momento no topo do ecrã, dois botões para efetuar *scroll* no caso de haverem ilhas suficientes para ultrapassar a largura do ecrã e o botão "Voltar" para voltar ao ecrã do jogador. As ilhas são clicáveis e permitem ao terapeuta avançar para o respetivo nível para a criança jogar.

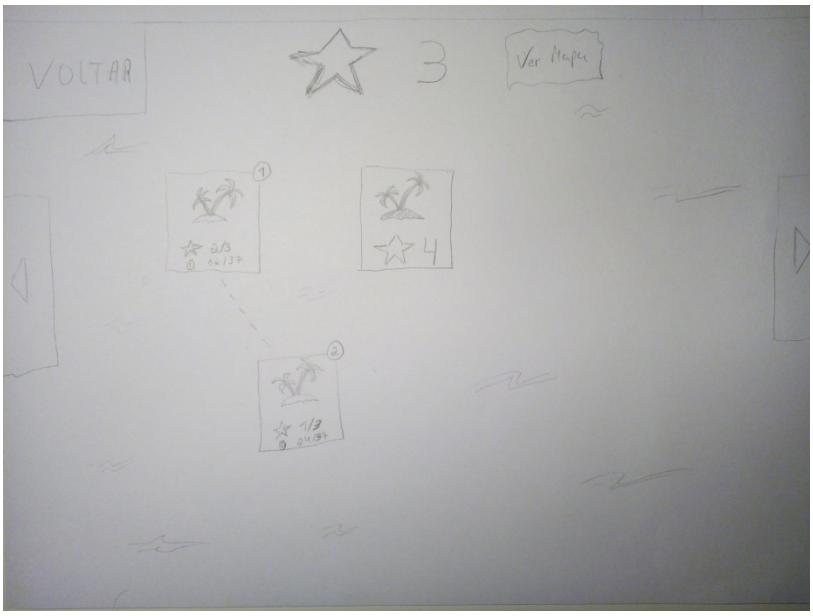


Figura 54 - Esboço do ecrã "Ver Mapa"

Ecrã "Ver Progresso"

Este ecrã permite ao terapeuta aceder ao progresso do jogador anteriormente selecionado. São apresentados os botões "Distâncias" e "Min/Max" para avançar para o respetivo ecrã e botão "Voltar" para voltar ao ecrã do jogador.

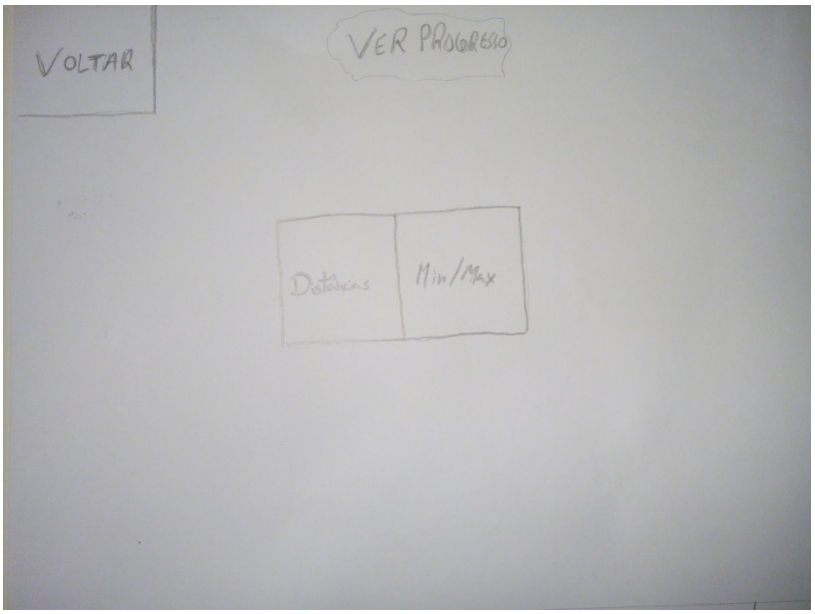


Figura 55 - Esboço do ecrã "Ver Progresso"

Ecrã do gráfico de barras "Distâncias" e "Min/Max"

Este ecrã permite ao terapeuta aceder aos valores das distâncias e dos mínimos e máximos do progresso do jogador anteriormente selecionado, consoante a opção selecionada. São apresentados dois gráficos de barras, um para cada mão, com os valores de todos os níveis e respetivas tentativas jogados pela criança, três botões de escolha única com cada eixo tridimensional que permitem selecionar os valores do respetivo eixo e o botão "Voltar" que permite voltar ao ecrã "Ver Progresso". Se a opção anteriormente selecionada foi "Min/Max" é apresentado também o botão "Campo de Ação" no canto superior direito que permite a alteração de apresentação de dados para uma vista 3D, explicada de seguida.

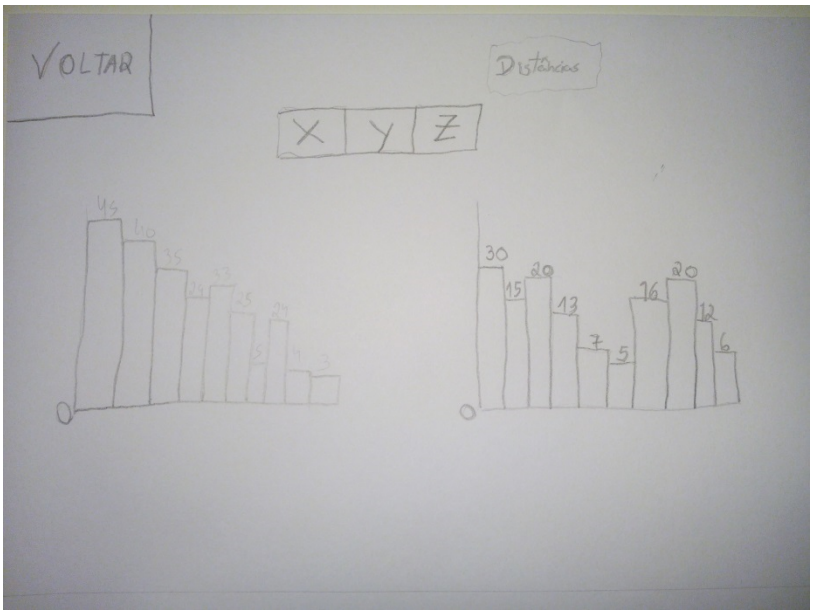


Figura 56 - Esboço do ecrã de gráficos de barras

Ecrã "Campo de Ação"

Este ecrã permite ao terapeuta aceder aos valores mínimos e máximos do progresso da criança anteriormente selecionada. São apresentados dois paralelepípedos, um para cada mão, com os valores mínimos e máximos de cada eixo atingidos pela criança no nível e tentativa selecionados, um botão para aumentar e outro para diminuir o nível e a tentativa pretendida, o botão "Gráfico de Barras" que permite alterar a apresentação de dados para gráficos de barras (de todos os níveis e tentativas como anteriormente explicada) e o botão "Voltar" que permite voltar ao ecrã "Ver Progresso".

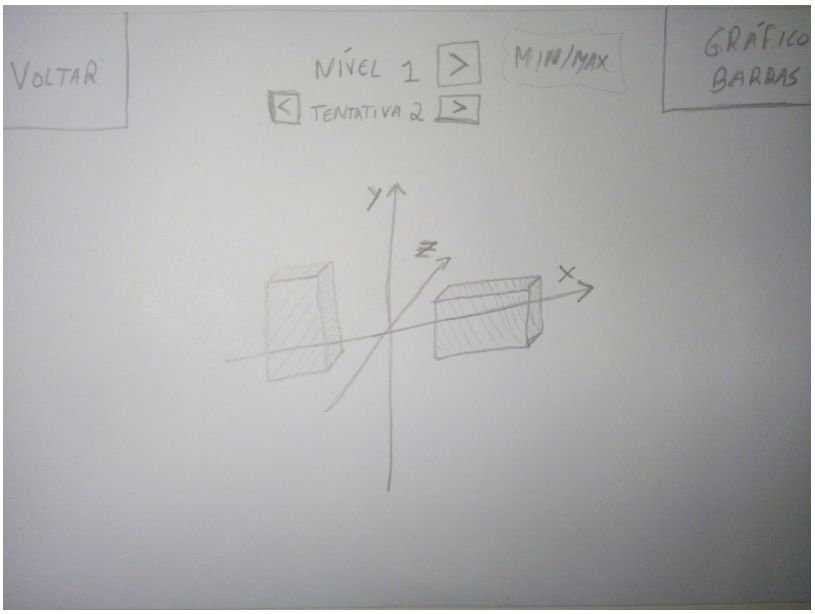


Figura 57 - Esboço do ecrã de vista 3D ou campo de ação

Para uma melhor visualização dos valores representados pelos objetos 3D foi também decidida a rotação no eixo Y e a movimentação no eixo Z de todo o campo de ação. A ação de rotação executa-se "arrastando" todo o campo de ação para a esquerda ou direita que, com o rato, consiste num movimento simples de arrastar (i.e. pastas), e no caso do *Leap Motion*, com ponteiro em cima de qualquer área, fecha-se a mão, move-se na horizontal e abre-se a mão para parar a rotação. A ação de movimentação é efetuada com o *scroll* do rato e no caso do *Leap Motion* é efetuada através dos movimentos da mão no mesmo eixo.

Depois de desenhados os esboços do menu em papel, foi iniciada a sua construção em formato digital através da criação de vetores. Para este processo foi utilizado o *software Adobe Illustrator CS6* e, numa fase posterior, o *Adobe Photoshop CS6* que serviu para tratar algumas imagens.

A navegação nos menus utilizando o *Leap Motion* é efetuada com uma mão que controla o ponteiro e para a seleção de botões foi utilizada a seguinte técnica: após três segundos com o ponteiro em cima de um botão, o mesmo é selecionado. Para ilustrar e indicar ao utilizador esta técnica foi utilizado o ponteiro vazio representado na Figura 58 que vai preenchendo como representado na Figura 59 ao longo desses três segundos.

Figura 58 - Cursor dos menus vazio

Figura 59 - Cursor dos menus quase cheio

A Figura 60 é um exemplo do ecrã principal nos primeiros desenhos vetoriais da interface do projeto. Era uma interface simples e "limpa".

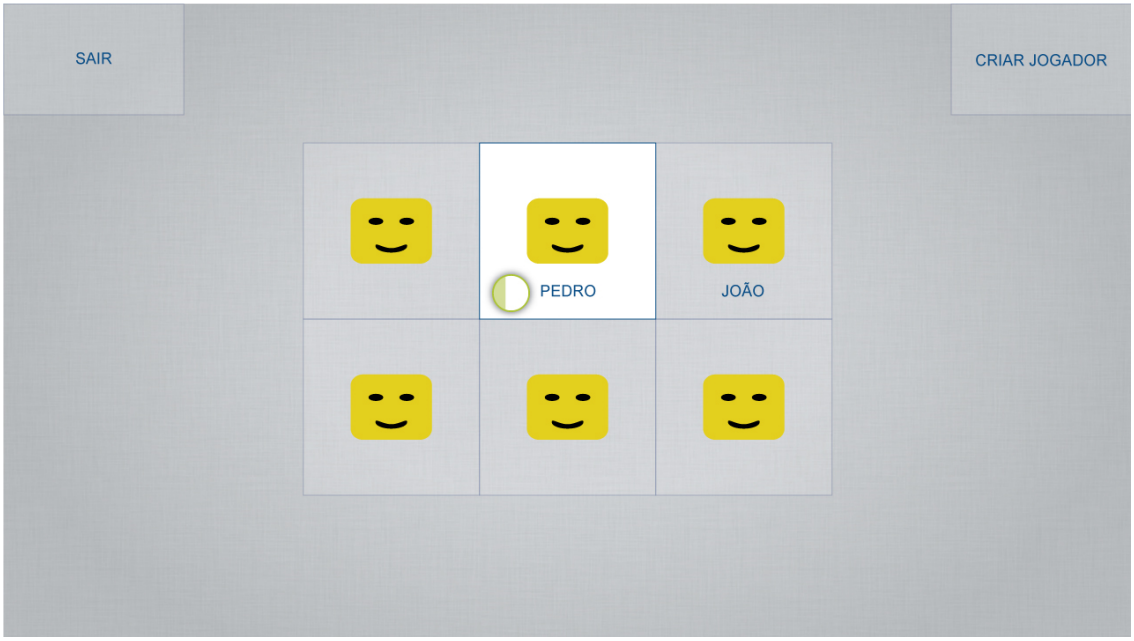


Figura 60 - Primeira versão da interface do ecrã principal do menu

Ao longo do desenho e também da implementação do projeto, a interface foi sofrendo pequenas alterações devido a conselhos de *designers*, tendo-se chegado a uma solução que se achou ser a mais adequada ao projeto e seus objetivos, nomeadamente o público-alvo. Na Figura 61 é possível ver algumas versões dos desenhos vetoriais na ferramenta *Adobe Illustrator CS6*.

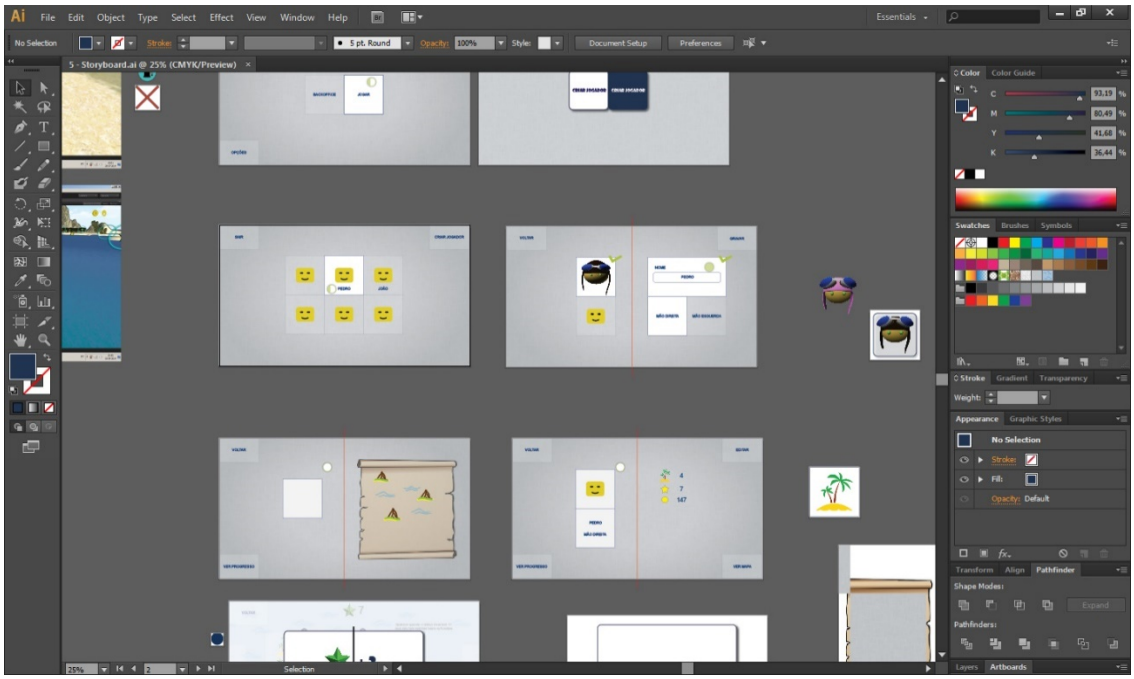


Figura 61 - Projeto de desenho vetorial no *Adobe Illustrator CS6*

As Figura 62 e 63 são exemplos da versão mais atualizada da interface, respetivamente os ecrãs "Criar Jogador" e "Ver Mapa". Os restantes ecrãs do menu são parecidos com estes dois, à exceção dos que permitem ver dados do jogador ("Distâncias" e "Min/Max") que, devido à sua complexidade, serão detalhados na secção 3.4 Implementação.

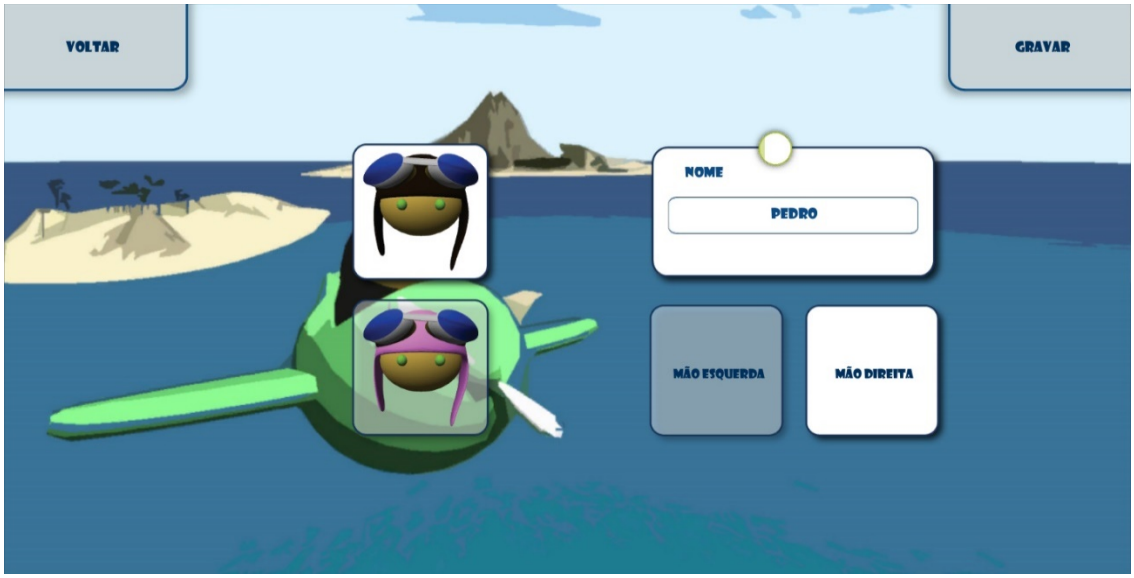


Figura 62 - Versão atualizada da interface do ecrã "Criar Jogador"

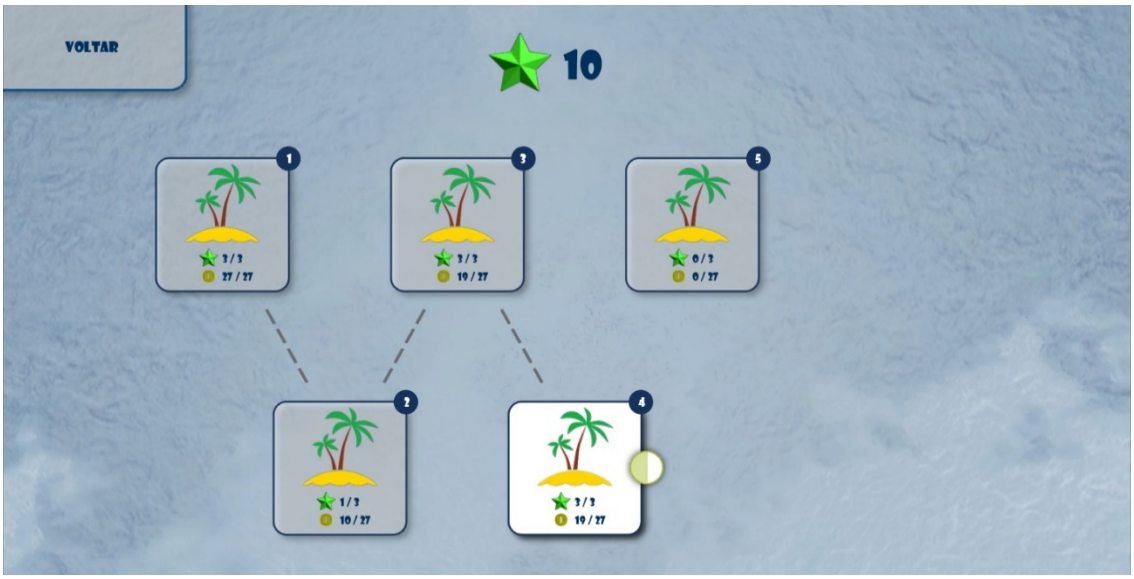


Figura 63 - Versão atualizada da interface do ecrã "Ver Mapa"

3.3.3 Sons

Como parte importante do projeto, a criação e escolha dos sons foi efetuada com cuidado, de modo a adequarem-se ao estilo de jogo e público-alvo. Todos eles foram descarregados gratuitamente.

Para a moeda foi criado um efeito sonoro no *site* www.bfxr.net.

Para a hélice do avião foi utilizado um som do *site* www.soundbible.com, posteriormente foi editado com o *sofwtare Audacity* como na Figura 64.

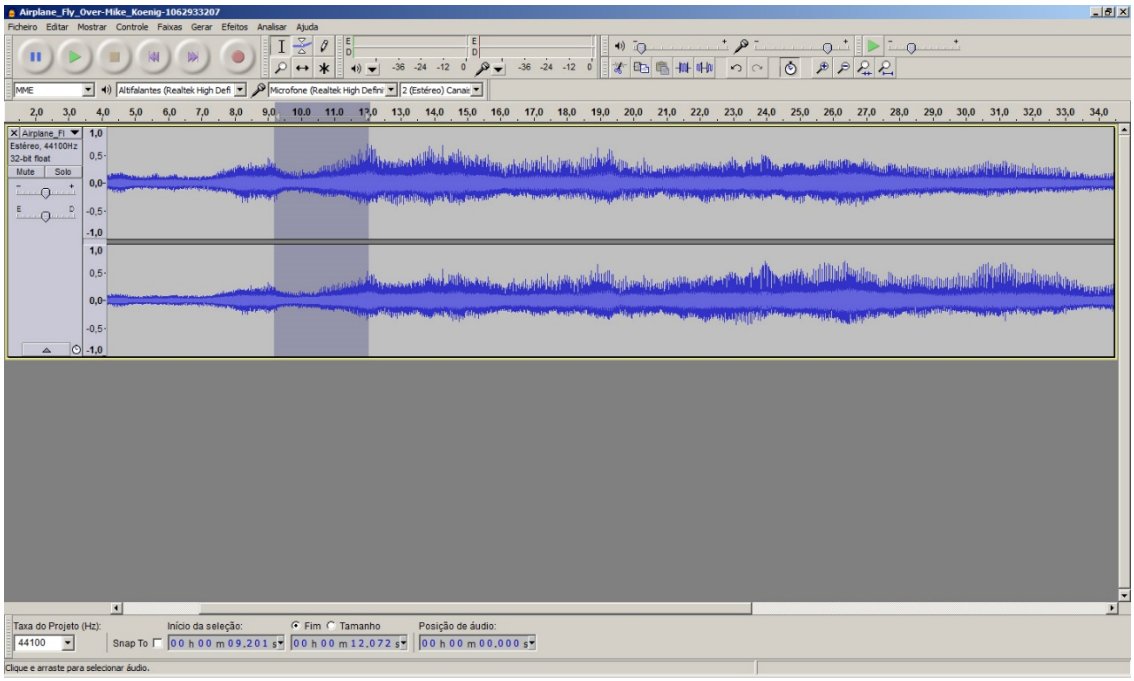


Figura 64 - Edição de som no *software Audacity*

Os restantes sons utilizados no jogo foram descarregados da loja do *Unity 3D* www.assetstore.unity3d.com.

3.4 Implementação

Ao longo desta secção será descrito o modo de funcionamento do jogo, bem como toda a componente técnica elaborada na fase de desenvolvimento, que permitiu assim chegar à solução final do projeto.

3.4.1 Funcionamento do *Unity 3D*

Um projeto no *Unity 3D* consiste num cenário constituído por objetos. Estes últimos são representados pela classe *GameObject* que contém as propriedades da sua posição, rotação e escala. Para sua a utilização em diferentes cenários foram criados *prefabs*, que consistem num *template* de um objeto que, ao ser alterada alguma propriedade, esta é aplicada a todas as instâncias. Por exemplo, nos três cenários existentes foi utilizado o objeto do avião. Foi então criado um *prefab* do avião pois assim, qualquer alteração feita, como a alteração do tamanho das asas, irá ser replicada em todos os cenários.

MonoBehaviour

No *Unity 3D* cada objeto pode ter agregada uma classe que estenda a *MonoBehaviour* que receba eventos do utilizador, manipule todos os comportamentos do objeto e tenha acesso às suas propriedades. Esta interface disponibiliza os métodos *Update()* e *FixedUpdate()* muito utilizados no jogo: o primeiro é executado uma vez por *frame* enquanto o segundo pode ser executado uma vez, zero, ou várias vezes por *frame*, dependendo da quantidade de *frames* por segundo de física são definidos nas configurações de tempo, e quão rápido/lento o *framerate* é.

A interface disponibiliza também a função *OnGUI()* que permite ter acesso à camada *GUI* (***G****raphical* ***U****ser* ***I****nterface*), ou interface gráfica do utilizador, muito útil para apresentar dinamicamente texto e imagens 2D ao utilizador. Os elementos aqui desenhados sobrepõem-se ao cenário filmado. Para uma uniformização de estilos de objetos (i.e. botões) utilizados nesta camada foram criadas duas *GUI Skin*: a *Game GUI Skin* para o cenário da ilha e a *Menu GUI Skin* para o menu principal. Por exemplo, no caso do botão, foram personalizados o tamanho e tipo de letra, imagens de fundo por defeito e *onMouseOver*, etc. Depois de personalizados os objetos, apenas é necessário indicar no método *OnGUI()* que se pretende utilizar a *GUI Skin*. Para o desenho de textos, texturas, e botões foram utilizados os métodos *GUI.Label()*, *GUI.DrawTexture()* e *GUI.Button()*, respetivamente, que recebem uma posição e um tamanho (altura e largura). No caso do botão, é também necessário definir o método para a ação do clique.

Collider

O *Collider* é um componente do *Unity 3D* que pode ser acrescentado a um objeto *GameObject*. Através deste componente, é possível detetar dinamicamente colisões entre objetos através do método *OnTriggerEnter()*, disponível na interface *MonoBehavior*. Por exemplo, neste jogo, os objetos do avião, moedas, estrelas e *checkpoints* têm a componente *Collider*. Dentro de cada classe associada aos três últimos *GameObject* está declarado o método *OnTriggerEnter()* onde é processada informação no momento da colisão (i.e. incrementar score, encher depósito de combustível, etc), que ocorre entre estes objetos e o avião.

ApplicationModel

Quando se troca de um cenário para o outro, o *Unity 3D* destrói toda a informação.

Para a comunicação entre cenários foi criada a classe *ApplicationModel* que contém objetos estáticos e armazena a informação durante o tempo de vida da aplicação, permitindo assim a troca de dados entre os cenários das ilhas e o menu principal. A Figura 65 apresenta o diagrama de comunicação entre os respetivos cenários onde o primeiro fornece os dados sobre o utilizador e o nível selecionados e o segundo os dados sobre os resultados do jogador nesse nível.

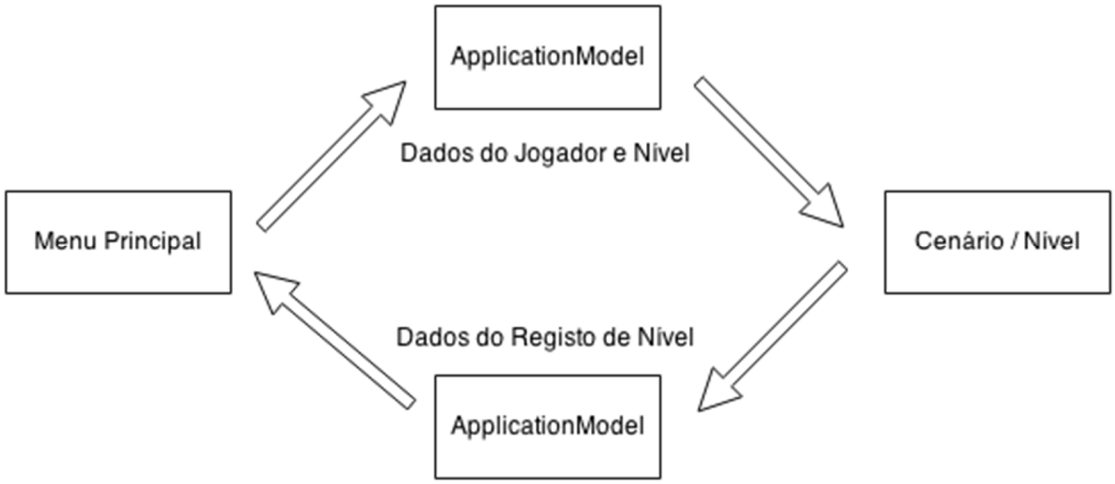


Figura 65 - Diagrama de comunicação entre o menu principal e o cenário da ilha

3.4.2 Base de dados

Para armazenar toda a informação dos jogadores foi modelada e implementada uma base de dados. A informação gravada na base de dados é relativa apenas aos jogadores e respetivos registos dos níveis, representada no modelo de domínio da Figura 66.

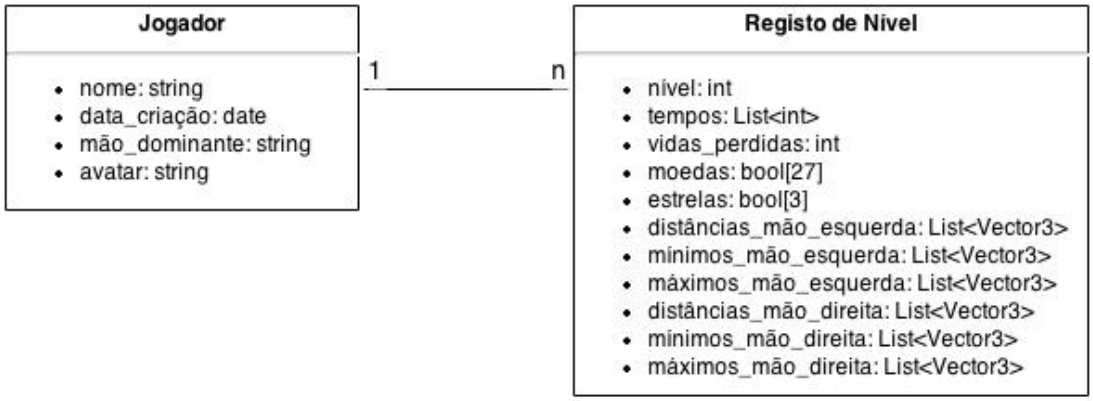


Figura 66 - Modelo de domínio da base de dados

Os dados armazenados provenientes do jogo, nomeadamente os que são captados pelo *Leap Motion* e analisados pelo terapeuta, são de enorme importância e o desenvolvimento do sistema de base de dados foi efetuado com prudência.

Considerando a hipótese de exportação ou migração da base de dados no futuro, decidiu-se que a mesma seria armazenada no formato *XML*, um dos mais padronizados atualmente, promovendo assim a sua escalabilidade. Tendo em conta estas características foram utilizados três padrões no sistema de base de dados: *Adapter, Singleton* e *Factory*.

O padrão *Adapter* foi utilizado para a leitura e gravação de dados, permitindo assim acrescentar facilmente diferentes formatos. Todos os adaptadores criados têm de estender a classe *InterfaceSaveLoad*, contendo assim os métodos obrigatórios *Load()* e *Save()*. Inicialmente estavam a ser utilizadas métodos simples que o *Unity 3D* disponibiliza através de processos de serialização binária (classe *AdapterSaveLoadApp*), que resulta num ficheiro de base de dados em código binário e obriga ao processo inverso para que os dados fiquem legíveis. Logo, foi acrescentado o formato *XML*, necessitando-se apenas de mais uma classe *Adapter* (classe *AdapterSaveLoadXML*).

O padrão *Factory* permite encapsular as condições que decidem qual adaptador utilizar e o *Singleton* garante existir apenas uma instância da mesma classe, acessível por toda a aplicação. Graças à *InterfaceSaveLoad*, a classe *SingleFactory* pode instanciar um adaptador sem conhecer o tipo de dados do mesmo, não criando assim dependências na estrutura e permitindo uma fácil inserção de novos adaptadores (formatos de base de dados). O *Singleton* também foi implementado na classe que contém a lista de jogadores, garantindo assim que existe apenas uma durante todo o jogo.

A Figura 67 apresenta o diagrama de classes do sistema de base de dados.

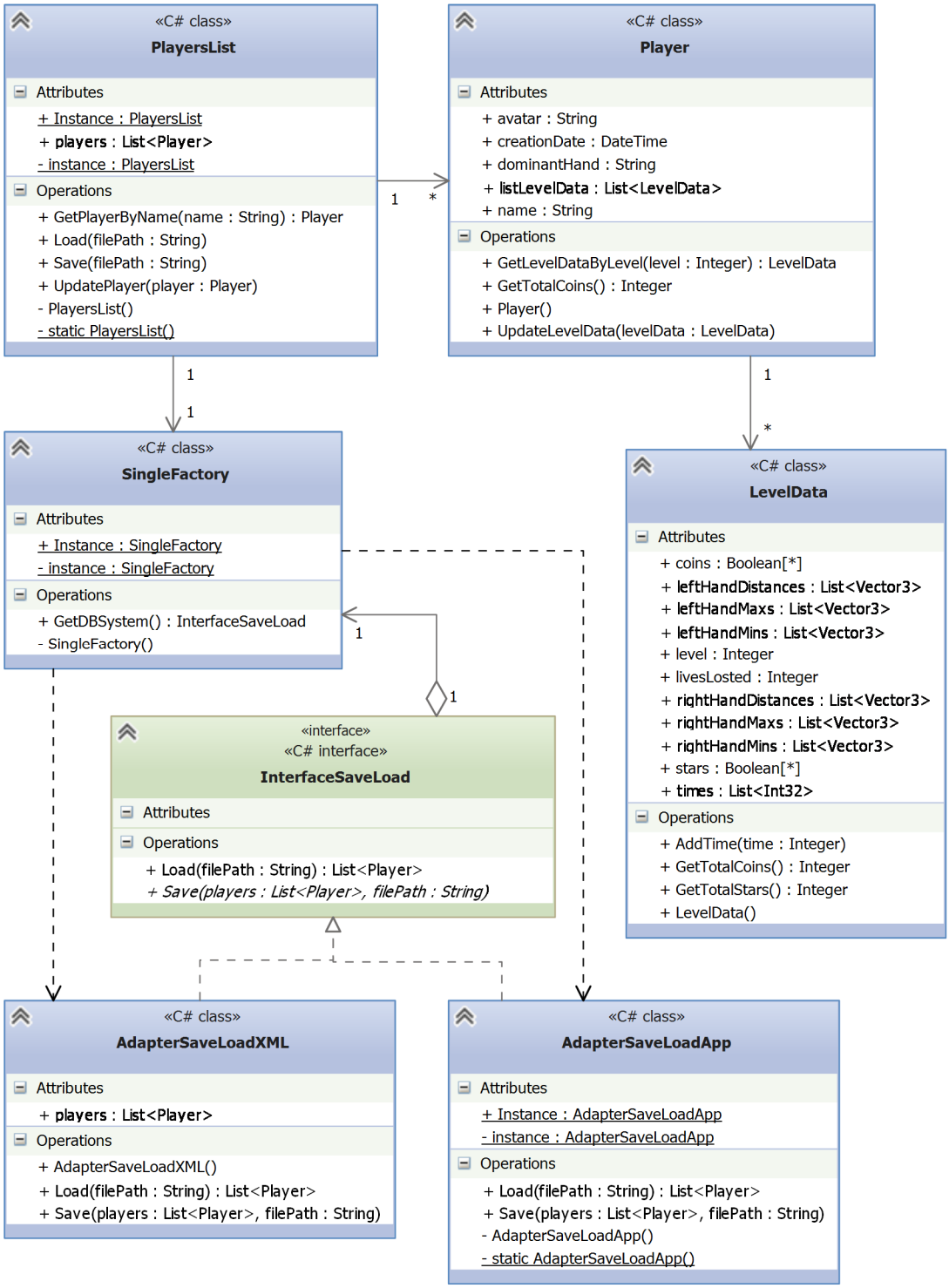


Figura 67 - Diagrama de classes do sistema de base de dados

Para a leitura e gravação de dados em *XML* foi utilizado o tipo de dados *XmlSerializer*, que permite a leitura e gravação de ficheiros através dos métodos *Serialize()* e *Deserialize()*. Este

necessita que o elemento raiz de cada ficheiro *XML* seja indicado com o atributo *XmlRoot*. Com *XmlArray* e *XmlArrayItem* declaramos as listas existentes. O Código 1 apresenta as declarações na classe *AdapterSaveLoadXML*:

```
[XmlRoot("PlayersCollection")]
public class AdapterSaveLoadXML : InterfaceSaveLoad
{
[XmlArray("players"), XmlArrayItem("player")]
        public List<Player> players;
        ...
}
```

Código 1 – Excerto da classe *AdapterSaveLoadXML* com declarações de *XML*

O Anexo 1 apresenta um exemplo do resultado do ficheiro *XML* que armazena os dados dos jogadores e níveis.

A leitura de dados da base de dados (ficheiro *XML*) apenas é efetuada uma vez durante o jogo, no momento em que é iniciado. Após a leitura do ficheiro de base de dados, a lista de jogadores da classe *PlayersList* é preenchida com a respetiva informação. A gravação de dados na base de dados ocorre em cinco momentos: criação, edição e remoção de perfil do jogador, conclusão com sucesso de um nível e fim das quatro vidas de um nível. Ambos os processos de leitura e gravação são idênticos, sendo o segundo representado no diagrama de sequência da Figura 68.

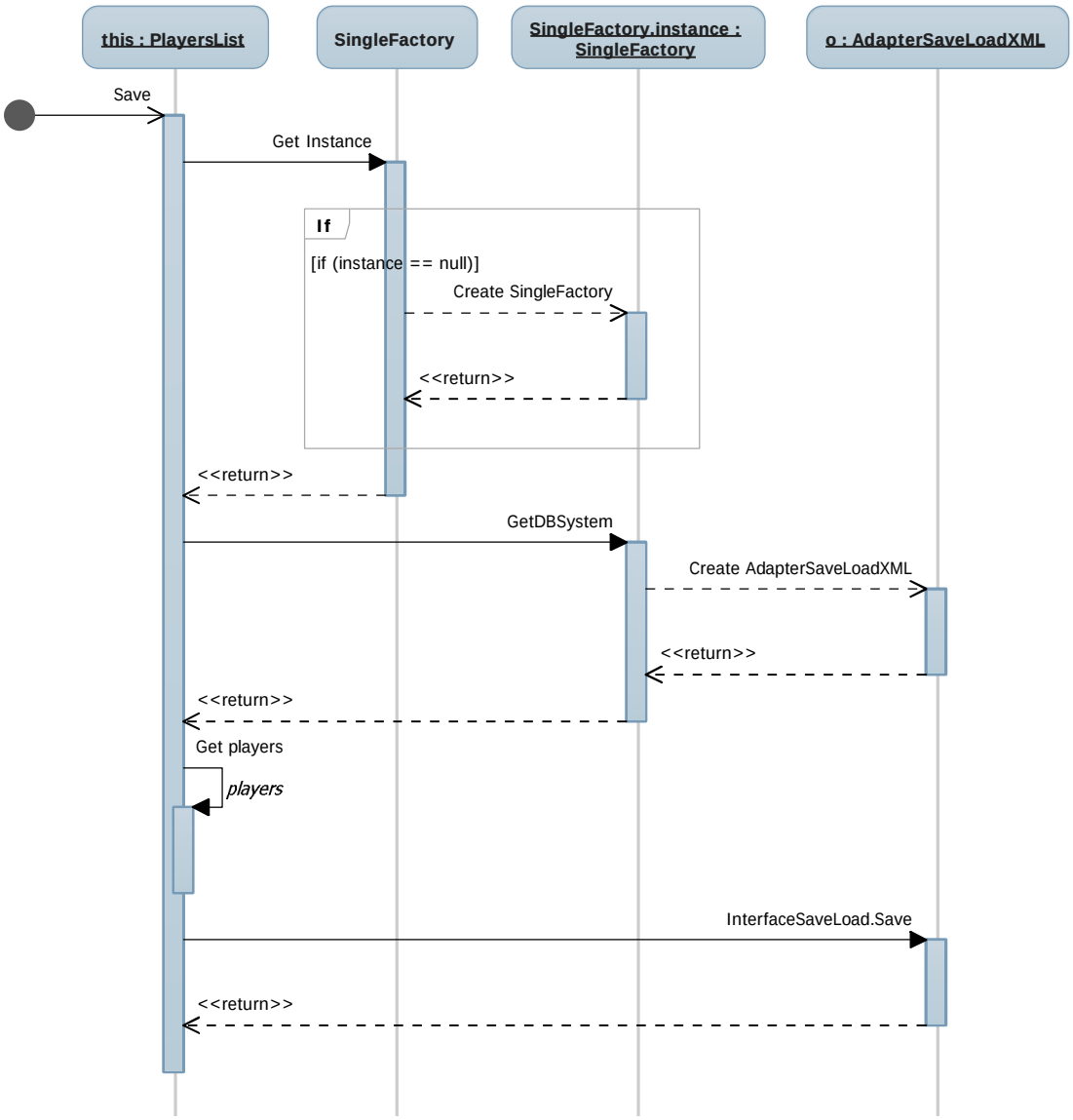


Figura 68 - Diagrama de sequência da gravação de dados

3.4.3 Leap Motion

Nesta secção serão descritas as classes que diretamente utilizam e manipulam dados recebidos através do *LeapMotion*.

LeapSingleton* e *LeapUnityExtensions

Para o auxílio de dados do *Leap Motion* no *Unity 3D* foram implementadas as classes *LeapSingleton* e *LeapUnityExtensions*. A primeira é responsável por instanciar e armazenar o objeto *Controller* do *LeapMotion*, que representa o dispositivo, e a segunda acrescenta o

método *ToUnityTranslated()* que inverte a direção do eixo Z (do vetor recebido do dispositivo), alinhando assim os dados com o *Unity 3D.*

PlaneController

Para o controlo do avião foi criada a classe *PlaneController* associada ao *GameObject* do avião que manipula a informação recebida do *LeapMotion* e a aplica ao objeto. No seu método *FixedUpdate()* (ver Código 2) é verificado se o dispositivo está a captar duas mãos e, caso afirmativo, calcula a distância vertical entre ambas. Esta distância é aplicada devidamente à rotação nos eixos Y e Z do avião, rodando-o assim em simultâneo com o movimento das mãos, como demonstrado na Figura 69.

```
void FixedUpdate() {
    . . .
    Frame frame = controller.Frame();
       If (frame.Hands.Count >= 2) {
           Vector3 leftHandPosition = new Vector3(),
                    rightHandPosition = new Vector3();
           foreach (Hand hand in frame.Hands) {
               if (hand.IsLeft)
                   leftHandPosition = hand.PalmPosition.ToUnity();
               else if(hand.IsRight)
                   rightHandPosition = hand.PalmPosition.ToUnity();
           }
           Vector3 avgPalmForward = (frame.Hands[0].Direction.ToUnity()
                    + frame.Hands[1].Direction.ToUnity()) * 0.5f;
           Vector3 handDiff = (leftHandPosition - rightHandPosition)
                    * 0.02f;
           Vector3 newRot = transform.localRotation.eulerAngles;
           newRot.z = -handDiff.y * 20.0f;
           newRot.y += - newRot.z * 0.03f *
                    transform.rigidbody.velocity.magnitude;
           newRot.x = 0;
           transform.localRotation = Quaternion.Slerp(
                                        transform.localRotation,
                                        Quaternion.Euler(newRot),
                                        0.1f);
           transform.rigidbody.velocity = transform.forward * speed;
       }
       . . .
}
```

Código 2 - Excerto do método *FixedUpdate()* da classe *PlaneController*

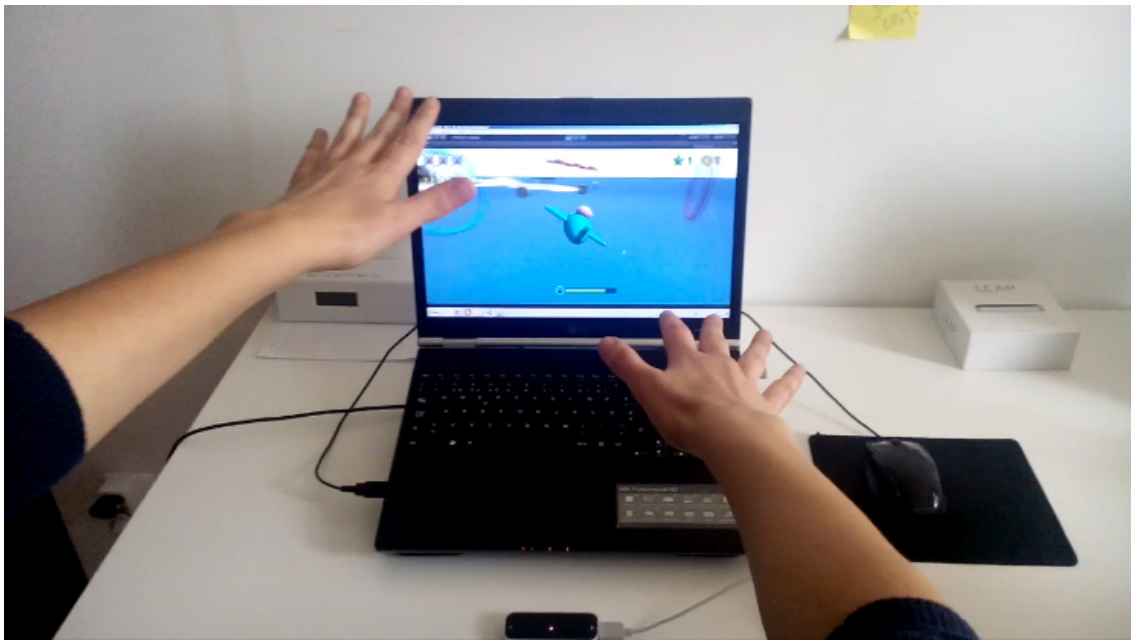


Figura 69 - Fotografia da vista do jogador a controlar o avião através do *Leap Motion*

Na classe *PlaneController* também está definido o método *SetPlayerLevelData()*, importante para a análise de dados por parte do terapeuta. Este método é invocado na função *FixedUpdate()* e efetua os cálculos do mínimo, máximo e distância percorrida em cada eixo tridimensional de cada mão da criança, através da comparação dos objetos da posição de ambas que recebe por parâmetro.

LeapMouseController* e *MouseCursor

Para o controlo do cursor nos menus foram implementadas as classes *LeapMouseController* e *MouseCursor*. A primeira é responsável por manipular os dados recebidos pelo *LeapMotion* e aplicar à segunda, responsável por mover o cursor e por aplicar um clique (utilizado ao fim de três segundos do cursor por cima de um botão). Para o último caso, foram utilizadas duas funções da API Win32, como apresentado no Código 3.

```
public class MouseCursor {
    [DllImport("user32.dll")]
    private static extern bool SetCursorPos(int x, int y);
    public static void MoveCursor(int x, int y) {
        SetCursorPos(x, y);
    }

    [DllImport("user32.dll", CharSet = CharSet.Auto, CallingConvention =
       CallingConvention.StdCall)]
    public static extern void mouse_event(
                    long dwFlags, long dx, long dy,
                    long cButtons, long dwExtraInfo);

    private const int MOUSEEVENTF_LEFTDOWN = 0x02;
    private const int MOUSEEVENTF_LEFTUP = 0x04;
    private const int MOUSEEVENTF_RIGHTDOWN = 0x08;
    private const int MOUSEEVENTF_RIGHTUP = 0x10;

    public static void DoMouseClick() {
        mouse_event(MOUSEEVENTF_LEFTDOWN | MOUSEEVENTF_LEFTUP, 0, 0, 0, 0);
    }
}
```

Código 3 – Excerto da classe *MouseCursor*

A classe *LeapMouseController* verifica o dedo mais adiantado da mão mais adiantada do utilizador, comparando a posição no eixo Z dos dedos dos respetivos membros. De seguida, é posicionado o cursor no ecrã consoante a posição da ponta desse dedo na área captável pelo *Leap Motion*, como apresentado no Código 4.

```
void FixedUpdate() {
    currentFrame = _leapController.Frame();
    Hand primeHand = frontMostHand();
    Finger primeFinger = Finger.Invalid;
    if(primeHand.IsValid) {
        primeFinger = pointingFigner(primeHand);
        if (primeFinger.IsValid) {
            int mousePosX =
            (int)((primeFinger.TipPosition.ToUnity().normalized.x + 1) /
            2 * UnityEngine.Screen.width);

            int mousePosY = UnityEngine.Screen.height
                    (int)primeFinger.TipPosition.ToUnity().y;
            MouseCursor.MoveCursor(mousePosX, mousePosY);
        }
    }
}
```

Código 4 - Método *FixedUpdate()* da classe *LeapMouseController*

3.4.4 Cenários das ilhas

Nesta secção serão descritas todas as classes utilizadas nos cenários das ilhas, bem como a implementação de partes importantes para o jogo.

OnLoad

A classe *OnLoad* é responsável por preparar os cenários, como criar e dispor os checkpoints e bónus consoante o nível, e por trocar toda a informação entre os diversos componentes do jogo.

Através do seu método *LoadMenu()*, esta classe lê e armazena os dados da classe *ApplicationModel* e atualiza a mesma através do seu método *BuildPlayerData()*, respetivamente no início e no fim do nível.

Como o jogo consiste no controlo da direção lateral do avião, a disposição dos *checkpoints* e bónus foi desenvolvida para inicialmente ser quase em linha reta, necessitando de poucos movimentos para os alcançar, e com o avançar do jogo, ser mais afastada, obrigando a criança a maiores movimentos dos membros superiores. Desta forma, não é exigido muito à criança numa fase inicial, o que permite uma melhor ambientação com o dispositivo *Leap Motion*.

A classe *OnLoad* contém o método *BuildCheckPointsAndBonus()* responsável por construir e dispor os *checkpoints*, moedas e estrelas. Todos estes componentes estão numa pasta denominada de *Resources* pois assim poder ser acedidos dinamicamente no *Unity 3D*. Depois de instanciados os objetos são guardados num *array* de *GameObject's* para posteriormente serem manipulados de forma simples.

No caso dos *checkpoints*, o método *BuildCheckPointsAndBonus()* cria os 10 existentes no nível e posiciona-os em "zig zag" a uma distância frontal de 13 unidades e lateral que aumenta consoante o nível, com uma componente aleatória, como representado na Figura 70. As distâncias são calculadas a cada objeto e a partir do centro dos mesmos, ou seja, o primeiro *checkpoint* pode estar 10 unidades à direita e o segundo pode estar seis unidades à esquerda.

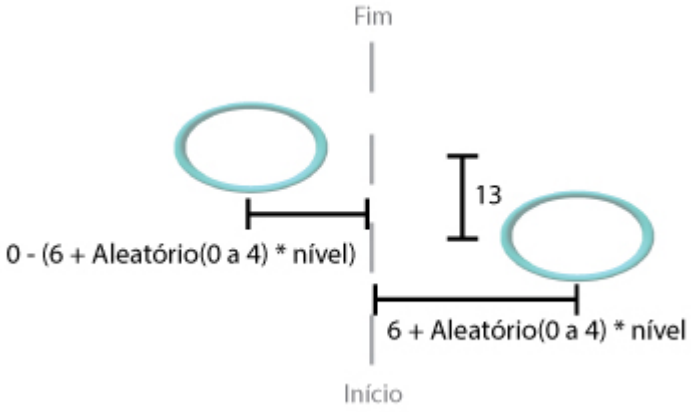


Figura 70 - Esquema da disposição dos *checkpoints*

Também a escala dos *checkpoints* diminuí dinamicamente dependendo do nível, aumentando ligeiramente assim a dificuldade do seu alcance. Para que o *checkpoint* não diminua para menos do que o avião necessita para passar por dentro, foi criada uma condição que limita o mínimo da escala para 100. O Código 5 representa a fase de instanciação dos *checkpoints*.

```
void BuildCheckPointsAndBonus() {
        float auxScale = 350 - 10 * level;
        if (auxScale < 100)
            auxScale = 100;
        Vector3 scale = new Vector3(auxScale, auxScale, auxScale);

        checkPoints = new GameObject[10];
        bonusCoins = new GameObject[27];
        bonusStars = new GameObject[3];

        int sign = 1;
        for (int nCP = 1; nCP <= 10; nCP++) {
            float distanceX = 6 + Random.Range(0, 4 * level);
            distanceX *= sign;
            GameObject checkpoint = (GameObject)
Instantiate(Resources.Load("CheckPoint/CheckPoint"));
            checkpoint.transform.localScale = scale;
            checkpoint.transform.position = new Vector3(distanceX,
cpDistanceY, cpInitialDistanceZ * nCP);
            checkPoints[nCP - 1] = checkpoint;
            if (nCP > 1)
                SetBonusCoins(nCP, totalCheckPoints, distanceX);

            // invert sign
            sign *= -1;
            ...
        }
        SetBonusStars();
    }
```

Código 5 – Excerto do método *BuildCheckPointsAndBonus()*

As moedas também são construídas e posicionadas pelo método *SetBonusCoins()*, sendo que entre dois *checkpoints* são instanciadas três moedas. Imagine-se uma linha imaginária entre o centro de dois *checkpoints* cujo comprimento é igual à distância entre os mesmos. Esta distância é dividida em quatro, resultando em cinco posições (zero, um, dois, três e quatro); as posições zero e quatro são respetivas à posição dos *checkpoints* e as restantes são utilizadas para a posição das moedas. Para que a disposição das mesmas não seja em linha reta (fácil para aquisição pela criança) foi também criada uma fórmula que as dispõe de forma aleatória dependendo do nível, como representado na Figura 71.

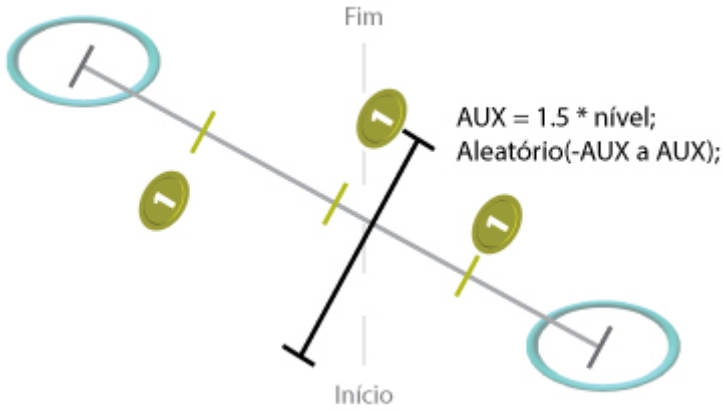


Figura 71 - Esquema da disposição das moedas

As estrelas são simplesmente colocadas sempre dentro do terceiro, sexto e nono *checkpoint* em cada nível.

Na vista superior dos níveis apresentada nas Figuras 72 e 73 é possível ver dois exemplos da disposição de todos os *checkpoints* e bónus nos níveis um e 20, respetivamente.

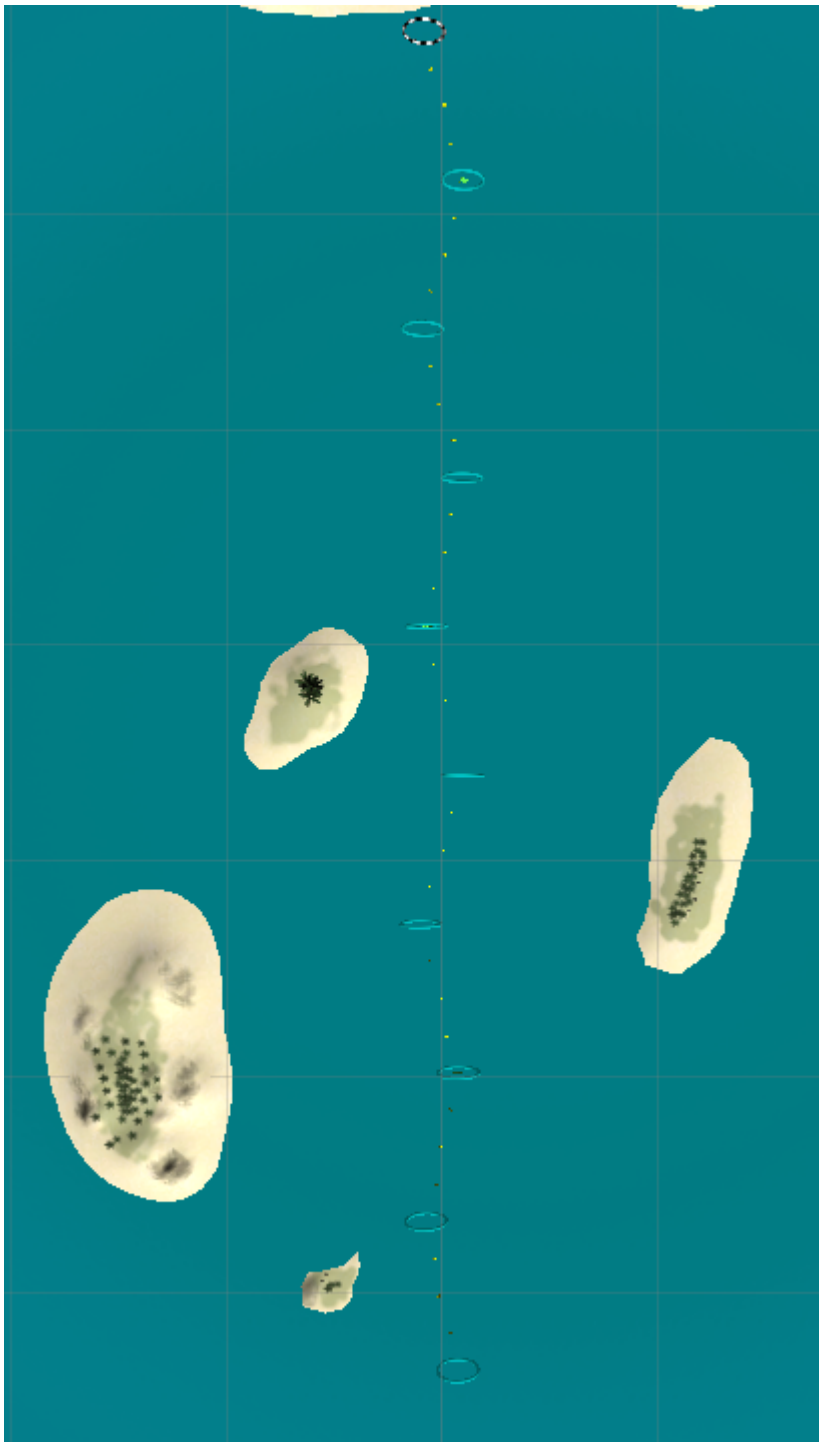


Figura 72 - Disposição dos *checkpoints* e bónus no primeiro nível

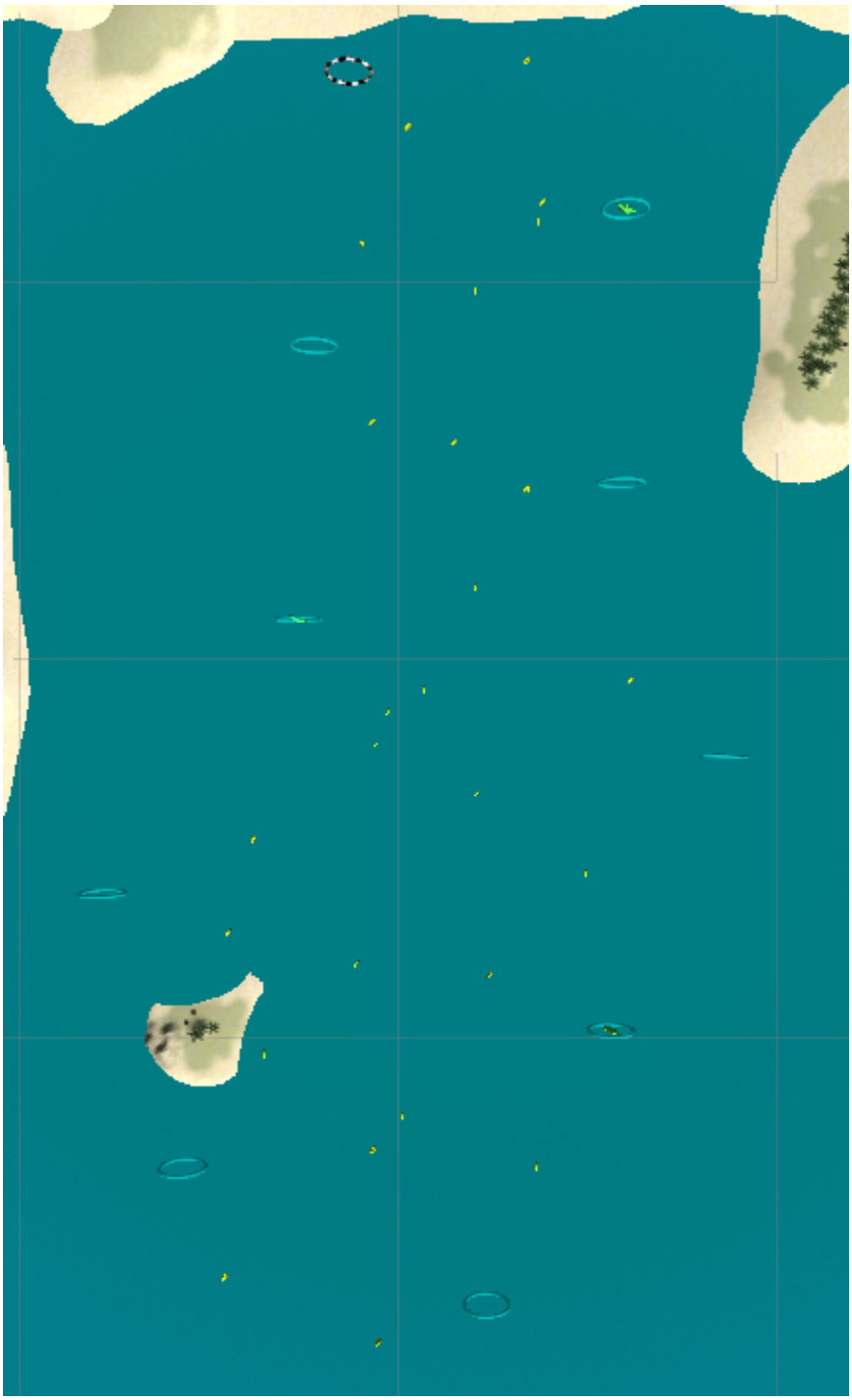


Figura 73 - Disposição dos *checkpoints* e bónus no vigésimo nível

Outro método relevante da classe *OnLoad* que importa descrever é o *NextCheckPoint()*. Este é responsável pelo processamento da aquisição de um *checkpoint* quando o avião o alcança e é invocado pela classe *CheckPointCollider*, explicada mais abaixo nesta secção. Se o *checkpoint* alcançado não for o último do nível, este método determina o próximo *checkpoint* (que é sequencial), altera a sua cor para vermelho, ativa o seu *Collider*, indica às setas o próximo

checkpoint e comunica à classe ScreenGUI, explicada mais abaixo nesta secção, para o abastecimento do depósito de combustível. O Código 6 representa um excerto deste método e do processo explicado anteriormente.

```
public void NextCheckpoint() {
        currentNCheckpoint++;
        if (currentNCheckpoint != checkPoints.Length) {
              arrowsScript.SetNextCheckpoint(checkPoints[currentNCheckpoin
       t].transform);

       checkPoints[currentNCheckpoint].transform.renderer.material.color =
new Color32(255, 0, 0, 127);
 checkPoints[currentNCheckpoint].transform.GetChild(0).collider.enabled =
true;

            if (currentNCheckpoint != 0)
                screenGUIScript.CheckpointGas();
        }
       ...
}
```

Código 6 - Excerto do método *NextCheckpoint()* da classe *OnLoad*

Caso o *checkpoint* alcançado seja o último do nível, este método inativa as setas de indicação, inativa o controlo do avião, comunica à classe *ScreenGUI* o fim do nível e ativa a classe *Landing*, responsável pela aterragem do avião e explicada mais abaixo nesta secção. Na Figura 74 é apresentado o diagrama de sequência do processo dispoltado pela aquisição onde é possível verificar que a classe *OnLoad* é a responsável pelas comunicações com outras classes.

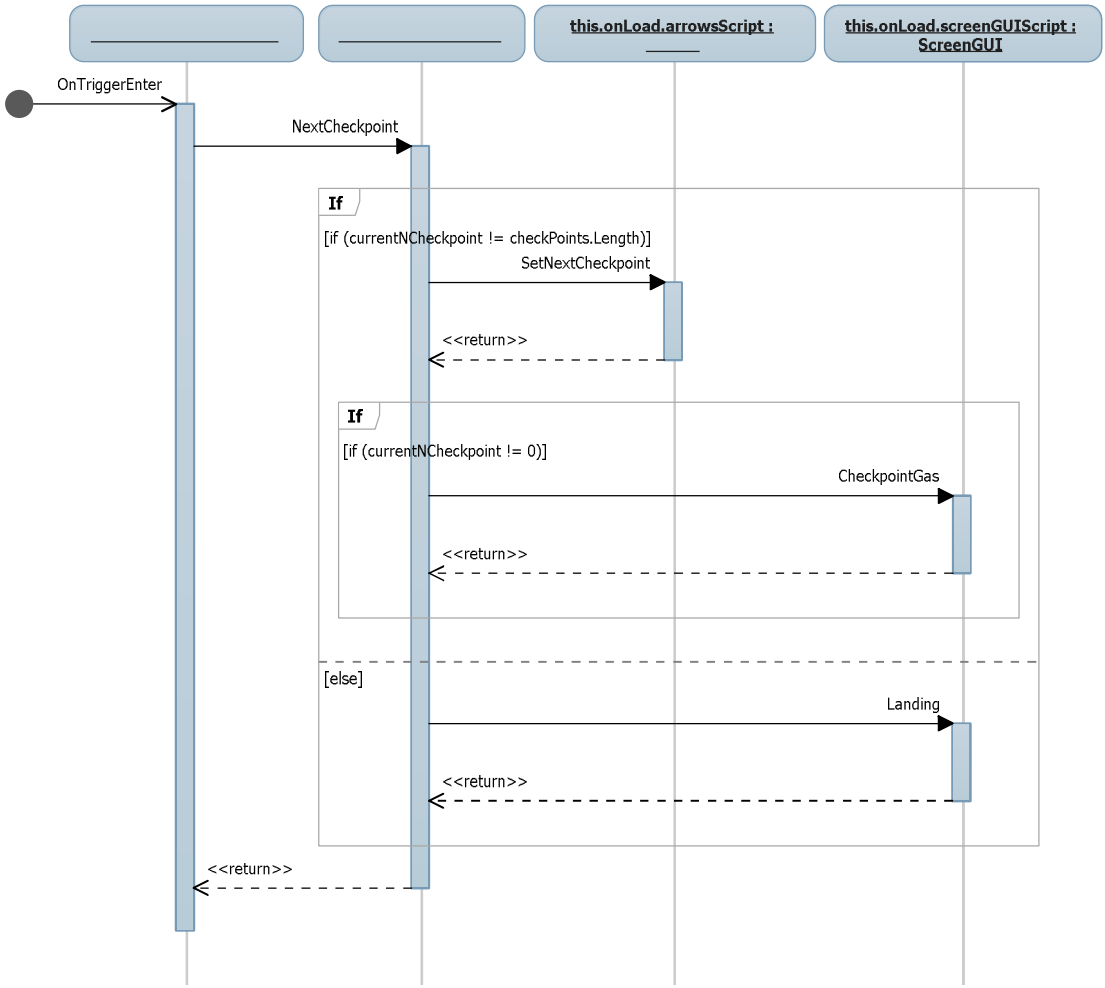


Figura 74 - Diagrama de sequência do método *OnTriggerEnter()* da classe *CheckPointCollider*

ScreenGUI

A classe *ScreenGUI* é responsável pela apresentação de textos e elementos 2D ao utilizador, como o *score* atual, combustível e os ecrãs de calibração e pausa, entre outros.

Para apresentar o ecrã de calibração (ver Figura 75) das mãos foi criado o método *CalibrateScreen()*, chamado na função *OnGUI()*. No primeiro método, definimos um fundo de imagem transparente em todo o ecrã através da função *SetBackground()* que utiliza a textura *bg* (imagem toda branca de 32x32 pixéis) e aplica uma transparência de 80%, como apresentada no Código 6. Este método é também utilizado nos ecrãs de pausa, fim de gasolina e fim do nível.

```
void SetBackground() {
        Color initialColor = GUI.color;
        GUI.color = new Color(initialColor.r, initialColor.g,
initialColor.b, 0.8f);
```

```
            GUI.DrawTexture(new Rect(0, 0, screenWidth, screenHeight), bg);
            GUI.color = initialColor;
        }
```

Código 7 - Função *SetBackground()* da classe *ScreenGUI*

Depois de aplicado o fundo branco e o texto "A CALIBRAR..." são colocadas duas texturas para indicar à criança onde colocar as mãos, estas representadas por outras duas texturas que são adicionadas quando o *Leap Motion* deteta as mãos da criança. Para acrescentar estas duas últimas texturas foram utilizadas duas variáveis *bool (isLeftHand* e *isRightHand)* e para as mover consoante a posição indicada pelo *Leap Motion* foram utilizadas duas variáveis *Vector3* (*leftHandPosition* e *rightHandPosition*), sendo todas as variáveis atualizadas na função *Update()*, como apresentado no Código 8.

```
void FixedUpdate() {
        if (calibrating) {
            Frame frame = controller.Frame();
            isLeftHand = isRightHand = false;
            foreach (Hand hand in frame.Hands) {
                if (hand.IsLeft) {
                    isLeftHand = true;
                    leftHandPosition.x =
(int)((hand.PalmPosition.ToUnity().normalized.x + 1) / 2 * screenWidth -
handX / 2);
                    leftHandPosition.y = screenHeight -
(int)hand.PalmPosition.ToUnity().y - handY;
                    leftHandPosition.z =
hand.PalmPosition.ToUnity().normalized.z;
                }
               ...
            }
        }
```

Código 8 - Excerto da função *FixedUpdate()* da classe *ScreenGUI*

Para validar a calibração das mãos é verificado se o extremo da textura de cada mão está perto do extremo da respetiva textura que marca a posição ideal, como representado na Figura 75. Em caso positivo, é apresentada uma imagem verde na respetiva posição e, depois de contados três segundos seguidos com ambas as mãos posicionadas corretamente, o avião inicia o levantamento da pista.

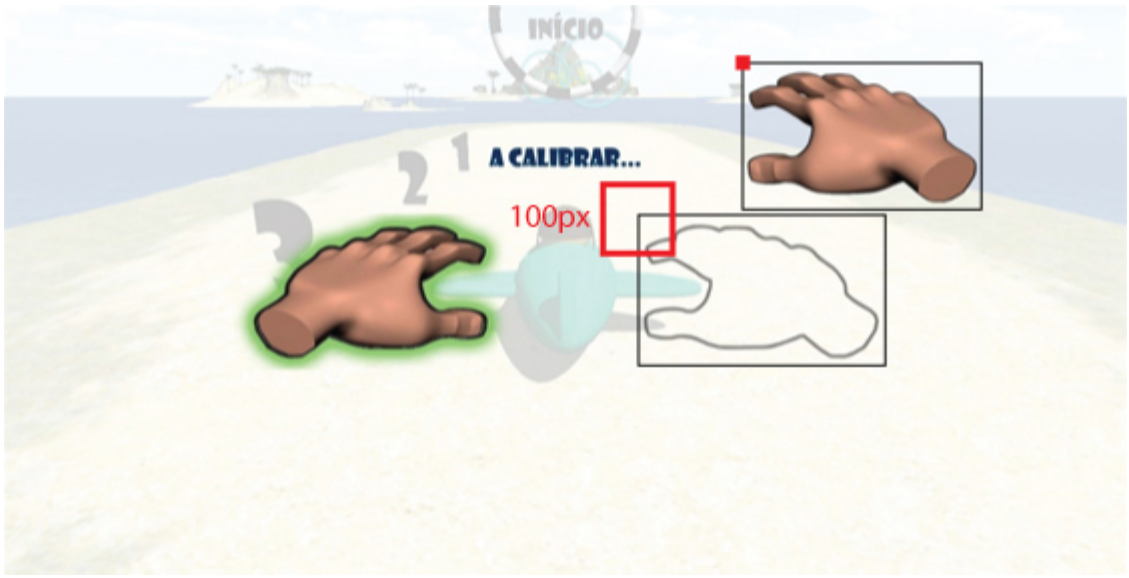


Figura 75 - Representação do processo de calibração das mãos

Os ecrãs de pausa, fim de combustível e fim do nível são processados de igual forma, diferenciando apenas no método invocado à classe *OnLoad* no evento da ação dos botões apresentados.

Enquanto a criança controla o avião são apresentados o combustível e o total de moedas e estrelas adquiridas através do método *PlayingScreen()*. Tanto para o combustível como para o cursor utilizado nos menus é utilizada a seguinte técnica: define-se uma área onde se vão desenhar as texturas pretendidas e, no seu interior, desenham-se, na mesma posição, as texturas que representam o estado vazio e cheio. Estas texturas vão se sobrepor e prevalece a textura do estado "cheio", da qual apenas é apresentada uma percentagem. As Figuras 76 e 77 representam respetivamente as texturas do estado vazio e cheio do combustível. A Figura 78 apresenta o resultado da utilização desta técnica quando o depósito está a 35%.

Figura 76 - Depósito de combustível do avião vazio

Figura 77 - Depósito de combustível do avião cheio

Figura 78 - Depósito de combustível do avião 35% cheio

A percentagem de combustível é armazenada numa variável e vai decrementando ao longo do tempo e é calculada na função *FixedUpdate()*, dando um depósito para 25 segundos. O combustível é abastecido totalmente quando o avião passa por um *checkpoint* com sucesso, momento em que o último avisa a classe *OnLoad* do sucedido.

O total de moedas e estrelas adquiridas é armazenado em duas variáveis atualizadas pela classe *OnLoad*, que recebe os eventos dos respetivos objetos.

CheckPointCollider

Esta classe é responsável pelo *GameObject checkpoint*. Este objeto é composto por dois *GameObject's:* uma argola grande visível com uma esfera pequena dentro invisível. O objetivo deste último é verificar a passagem bem-sucedida do avião pelo *checkpoint*, sendo esta detetada pela colisão do avião com a pequena esfera. Logo, é este objeto que contem a componente *Collider* e a classe *CheckPointCollider*. Como os *checkpoints* apenas são alcançáveis uma vez, o seu componente *Collider* fica inativo após a primeira colisão. O Código 9 representa um excerto do método *OnTriggerEnter()* que recebe como parâmetro o componente *Collider* do objeto, desativa-o, altera a cor da argola visível para verde (*GameObject* parente da esfera invisível) e comunica à classe *OnLoad* (responsável pelo fluxo principal), instanciada no objeto *onLoad*.

```
void OnTriggerEnter(Collider collider) {
            ...
        this.collider.enabled = false;
this.transform.parent.renderer.material.color = new Color32(0, 255, 0,
127);
        onLoad.NextCheckpoint();
    }
```

Código 9 – Excerto do método *OnTriggerEnter* da classe *CheckPointCollider*

BonusCoin* e *BonusStar

As classes *BonusCoin* e *BonusStar* estão associadas a cada moeda e cada estrela, respetivamente. Estas utilizam a componente *ParticleSystem* do *GameObject* que permite simular líquidos, nuvens e chamas, gerando e animando pequenas imagens 2D. No jogo "À Descoberta das Ilhas" foram configuradas *ParticleSystem's* de forma a parecer que as estrelas e moedas explodem e se dissolvem em pequenas partículas quando o avião colide com as mesmas. Ambas as classes são idênticas, efetuando o seguinte processamento no método

OnTriggerEnter(): atualiza o score do utilizador, ativa as partículas, esconde e desativa o *GameObject*. O Código 10 representa um excerto da mesma função.

```
void OnTriggerEnter(Collider collider) {
       ...
       this.particleSystem.Play();
        this.renderer.enabled = false;
        StartCoroutine(Hide());
}
```

Código 10 - Excerto do método *OnTriggerEnter()* das classes *BonusCoin e BonusStar*

O método *StartCoroutine()* da classe *MonoBehaviour* pode ser interrompido utilizando o código *yield*. No caso da animação das estrelas e moedas o objetivo é desativar o *GameObject* apenas quando a animação das partículas termina e para isso foi definido método *Hide(),* apresentado no Código 11.

```
IEnumerator Hide() {
        float time = this.particleSystem.duration;
        yield return new WaitForSeconds(time);
        this.gameObject.SetActive(false);
}
```

Código 11 - Método *Hide()* das classes *BonusCoin e BonusStar*

Upgrade* e *UpgradeList

Para o avião foi criado um *prefab* que contém o avião simples e todos os seus *upgrade*s, cada um agrupado num *GameObject* inativo para que mais tarde possa ser dinamicamente ativo. A Figura 79 apresenta a vista do *Unity 3D* com o avião num cenário (vazio, neste caso) com o segundo *upgrade* ativo e é possível ver a estrutura do mesmo (separador *Hierarchy,* canto superior direito): os *GameObject Back*, Left e *Right* representam as respetivas beiras na pala traseira e nas asas do avião, agrupadas no *GameObject Upgrade2* que se encontra ativo (para demonstração).

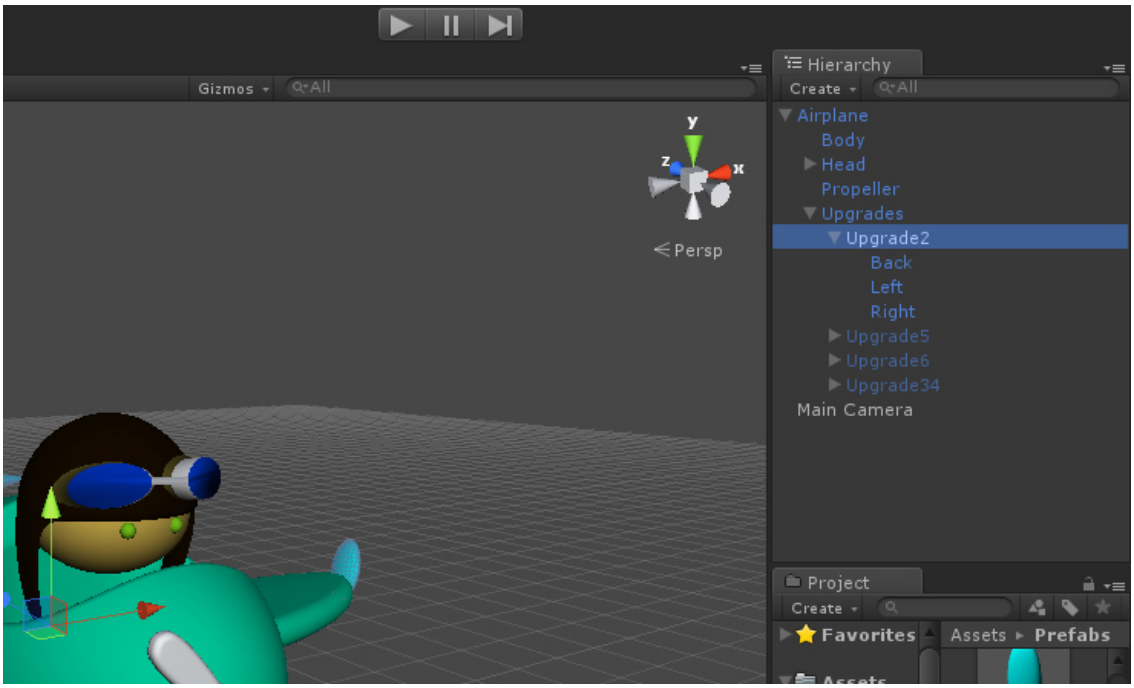


Figura 79 - Vista do *Unity 3D* do avião e da estrutura dos seus *upgrades*

O *prefab* do avião é colocado no cenário como um *GameObject* e para auxiliar a gestão dos seus *upgrades* foram criadas as classes *Upgrade* e *UpgradesList*. A primeira permite um tipo de dados com os campos *id*, moedas e descrição, que representa um item da Tabela 18 - Sistema de aquisição de *upgrades* do avião, e a segunda classe permite ter uma lista desses items e é responsável por ativar cada *upgrade*. O Código 12 apresenta a função *GetUpgrades()* da classe *UpgradesList*, responsável por ativar *upgrades* consoante o número de moedas do jogador.

```
public void SetUpgrades(int coins) {
        foreach (Upgrade upgrade in upgrades) {
            if (upgrade.coins <= coins) {
                try {
                    MethodInfo mi = this.GetType().GetMethod("SetUpgrade"
+ upgrade.id);
                    mi.Invoke(this, null);
                }
                catch (Exception e) {
                    Debug.Log(e);
                }
            } else
                break;
        }
}
```

Código 12 - Método *SetUpgrades()* da classe *UpgradesList*

O Código 13 apresenta a função *SetUpgrade2()* da classe *UpgradesList*, responsável por ativar o segundo *upgrade*.

```
public void SetUpgrade2() {
          GameObject.Find("Airplane").transform.
              FindChild("Upgrades").transform.
              FindChild("Upgrade2").gameObject.SetActive(true);
}
```

Código 13 - Método *SetUpgrade2()* da classe *UpgradesList*

CameraController

A classe *CameraController* está associada ao *GameObject* da câmara, objeto do *Unity 3D* que apresenta o cenário ao utilizador. A função desta classe é fazer com que a câmara esteja sempre posicionada perto e apontada para o avião. Para a primeira, a posição da camara é igualada à do avião menos o valor da distância inicial, para não ficarem exatamente na mesma posição, resultando numa vista de terceira pessoa. Para que a camara também rode com os movimentos laterais do avião é multiplicado ao valor da distância entre ambos a mesma rotação do avião no eixo Y. Para a segunda função da classe *CameraController* é utilizado o método *LookAt()* da classe *Transform* (disponível no *GameObject*) que aponta a sua instância para o objeto que recebe por parâmetros (avião). Se o avião estiver a aterrar a distancia a subtrair é aumentada, resultando numa posição diferente de quando a criança está a controlar o avião. Todo este processo é efetuado no método *FixedUpdate()*.

Head

A classe *Head* está associada ao avatar (boneco/cabeça que pilota o avião) e a sua função é rodar ligeiramente para o mesmo lado que o avião roda, sendo-lhe aplicada a mesma rotação no eixo Y. Para que o movimento não seja linear e direto foi utilizada o método *Quaternion.Slerp()* que permite suavizar a rotação. Todo este processo é efetuado no método *FixedUpdate()*.

Propeller

Associada à hélice do avião, a classe *Propeller* é responsável pela sua rotação e dispõe de dois métodos públicos, invocados pela classe *OnLoad*: *ON()* e *OFF()* que alteram a rotação para o respetivo estado.

IslandCollider

A classe *IslandCollider* está associada à ilha principal dos cenários e a sua função é detetar a colisão do avião com a mesma. No método *OnTriggerEnter()* é invocada a função *CollisionBackward()* da classe *PlaneController* associada ao avião que inverte a velocidade do mesmo para negativo, originando assim no movimento contrário do avião. De seguida, a velocidade vai aumentando até ao valor normal, movendo o avião em frente.

Landing e LandingPointCollider

O processo de aterragem do avião consiste no seguinte: após a passagem pelo último *checkpoint* a criança deixa de ter controlo sobre o avião que se desloca para dois pontos (invisíveis) sequencialmente, estando o primeiro no inicio da pista, à mesma altura do avião, e o segundo a meio da pista, mais baixo. Quando o avião colide com o segundo, a sua velocidade começa a diminuir até parar. A classe *LandingPointCollider* é responsável por detetar as colisões do avião com ambos os pontos de aterragem e comunicar à classe *Landing*, responsável pelo restante processamento e ativada pelo método *NextCheckPoint()* da classe *OnLoad*. Quando o avião é para, a classe *Landing* comunica à classe *OnLoad*, invocando o seu método *Finished()*.

StartText

A classe *StartText* está associada a um *GameObject* que contém o texto "Início" e números três, dois e um que auxiliam a perceção da criança no levantamento do avião. A função desta classe é detetar a colisão do avião com o *GameObject* do texto, comunicar à classe *OnLoad* através do método *TakedOff()* e desativar o seu conteúdo: texto e números. Após esta colisão, a criança assume o controlo do avião.

3.4.5 Menu principal

MenuScreenGUI

A classe *MenuScreenGUI* é a responsável pela apresentação e gestão do menu principal, dividindo cada ecrã (descritos na secção 3.3 Desenho) pelo respetivo método. O processo de apresentação de ecrãs consiste no desenho de texturas 2D e textos, apresentado

anteriormente neste documento, e é muito semelhante em todo o projeto, por isso, apenas se evidenciam a implementação do gráfico de barras e do campo de ação.

Importa aqui descrever o método *PlayerGameResults()*, implementado já numa fase final do projeto, responsável por apresentar a informação do nível jogado pela criança logo após o retorno do cenário da ilha ao menu principal, como apresentado na Figura 80.

Figura 80 - Ecrã "Mostrar resultados"

O Código 14 apresenta o método *PlayerGameResults()* que compara o resultado anterior do jogador no nível jogado com o novo resultado, atualiza a lista de jogadores, grava os novos resultados e apresenta os novos bónus conquistados. O método *InvokeRepeating()* permite invocar uma função repetidamente, após determinados segundos e num determinado intervalo de segundos, e foi utilizada neste caso para incrementar o resultado espaçadamente de forma a criar uma animação para ambos o terapeuta e a criança.

```
void PlayerGameResults() {
    currentPlayer = playersList.GetPlayerByName(
                        ApplicationModel.currentPlayerName);
    if (ApplicationModel.levelResult != -1) {
        onShowPlayerResults = true;
        int oldTotalStars = 0, oldTotalCoins = 0;
        LevelData levelDataOld = currentPlayer.GetLevelDataByLevel(
                        ApplicationModel.currentLevelData.level);
        if (levelDataOld != null) {
            oldTotalStars = levelDataOld.GetTotalStars();
            oldTotalCoins = levelDataOld.GetTotalCoins();
        }
        resultsStars = ApplicationModel.currentLevelData.GetTotalStars()
                        - oldTotalStars;
        resultsCoins = ApplicationModel.currentLevelData.GetTotalCoins()
                        - oldTotalCoins;
        currentPlayer.UpdateLevelData(ApplicationModel.currentLevelData);
        playersList.Save(Path.Combine(Application.dataPath,"players.xml"));
        if (resultsStars > 0)
            InvokeRepeating("IncrementResultStars", 1, 0.2f);
        else if (resultsCoins > 0)
            InvokeRepeating("IncrementResultCoins", 1, 0.2f);
        else
            Invoke("ClosePlayerGameResults", 2);
    }
}
```

Código 14 - Método *PlayerGameResults()* da classe *MenuScreenGUI*

Min/Max

Os gráficos de barras têm dimensões fixas no ecrã e são constituídos por barras verticais e linhas horizontais. As primeiras, denominadas de *Regist's* (ver Figura 83), apresentam os valores a analisar e as segundas orientam as escalas.

O processo descrito de seguida aplica-se ao desenvolvimento do gráfico de barras para a apresentação, tanto dos valores "Min/Max", como dos valores "Distâncias".

Para melhor descrever o processo de desenvolvimento são apresentadas as Figuras 81 e 82 que representam os ecrãs de visualização do progresso da criança, respetivamente nos mínimos e máximos e nas distâncias atingidos pela criança.

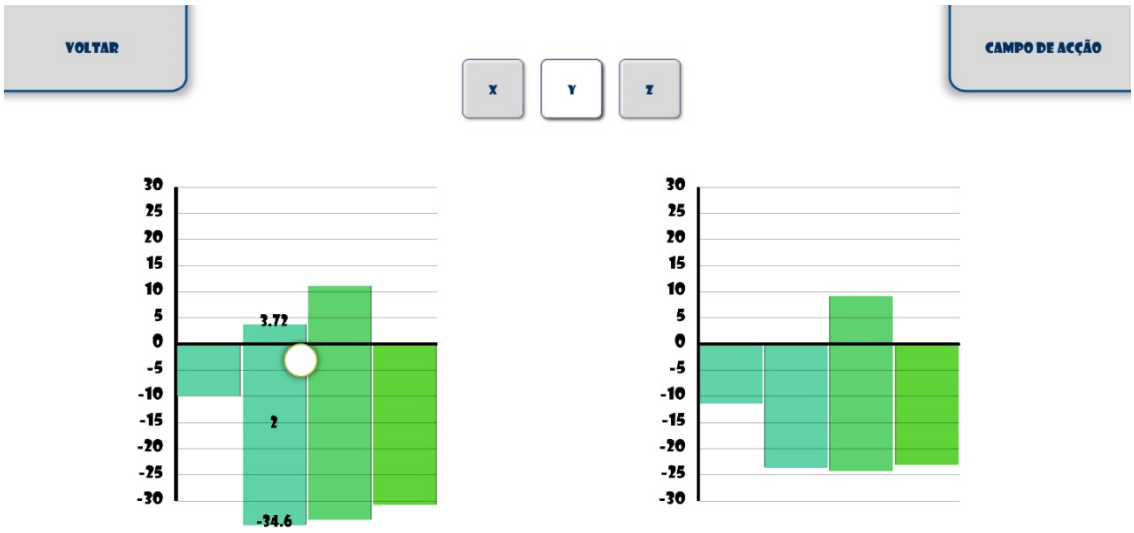


Figura 81 - Ecrã do gráfico de barras dos mínimos e máximos atingidos pela criança

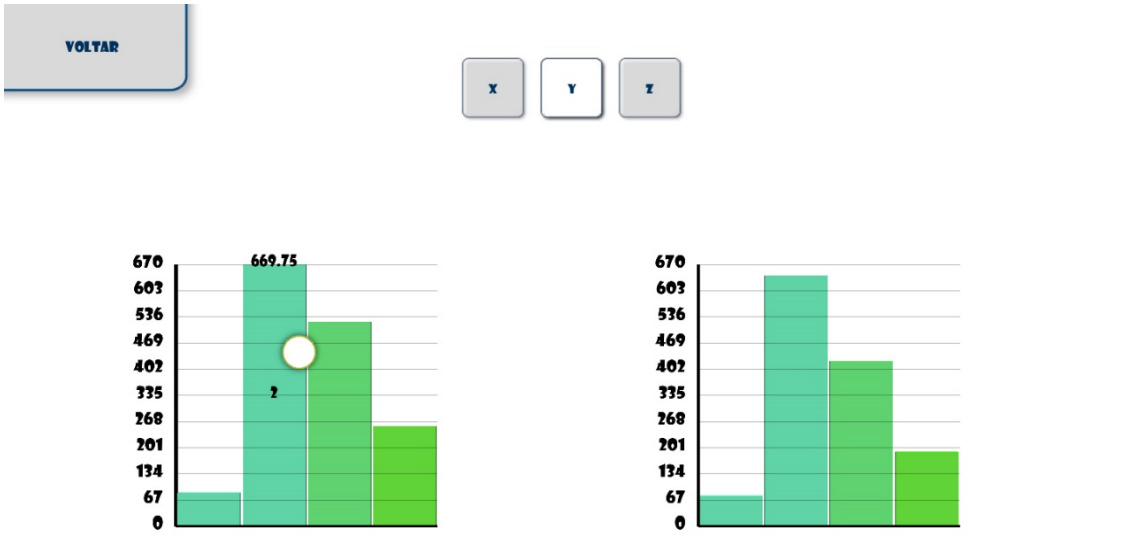


Figura 82 - Ecrã do gráfico de barras das distâncias percorridas pela criança

A classe *BarChart* é responsável pela construção de ambos os gráficos (mão esquerda e direita). Depois de recebidos os dados do jogador é definida a escala de valores do gráfico, baseada nos valores a apresentar. No caso "Min/Max", os valores mínimos e máximos são sempre os respetivos da área captável pelo *Leap Motion*. No caso das distâncias, o valor mínimo é sempre zero e o máximo é a respetiva distância, sendo a escala do gráfico aumentada de acordo com a necessidade. Depois é calculado o número de barras verticais a apresentar e a sua largura, consoante o número de tentativas e níveis jogados pela criança. De seguida, para cada dado da criança, é instanciado o objeto *Regist* (ver Figura 83), constituído

por um paralelepípedo e três textos: mínimo, máximo e tentativa. São preenchidos os textos e o paralelepípedo é escalado e posicionado devidamente.

Figura 83 - *GameObject Regist* que representa um registo do gráfico de barras

Os textos estão invisíveis, ficando visíveis quando o cursor é passado por cima do registo.

Para diferenciar os registos por nível é definida uma escala de cores e é atribuída uma cor única para cada nível. Na Figura 82 é apresentado um exemplo onde é possível verificar a segunda tentativa do primeiro nível, onde ambos os registos têm a mesma cor.

Ao ser alterado o eixo tridimensional a classe *OnScreenGUI* obtém os respetivos dados da criança e envia-os novamente para a classe *BarChart*, que efetua novamente o processo de construção do respetivo gráfico de barras.

Campo de Ação

Para a apresentação dos valores mínimos e máximos atingidos pela criança em forma 3D, denominado de campo de ação, foram criados dois cubos 3D, um para cada mão da criança, como apresentado na Figura 84. Os eixos ilustrados representam a área captável pelo *Leap Motion*.

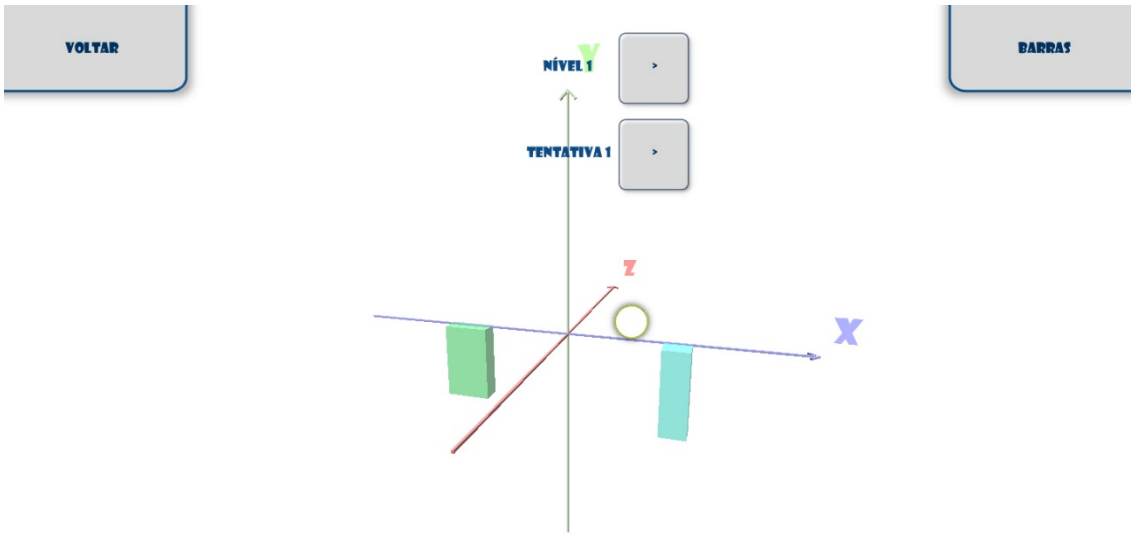


Figura 84 – Ecrã do campo de ação com os valores mínimos e máximos atingidos pela criança

A classe *ActionFieldCube* está associada ao *GameObject* cubo e, após receber dois objetos *Vector3* (contém as propriedades x, y e z) com os valores mínimos e máximos de cada eixo, é responsável por efetuar a devida escala e posicionamento. O Código 15 apresenta o método *SetVectors3()* da classe *ActionFieldCube* invodado pela classe *MenuScreenGUI*. Os valores *scale* e *localPosition* são aplicados ao *GameObject* no método *FixedUpdate()* com o auxílio da função *Vector3.Lerp()* quer permite suavizar a transição, resultando assim a escala e posicionamento numa animação.

```
public void SetVectors3(Vector3 mins, Vector3 maxs) {
        scale = new Vector3(maxs.x - mins.x,
                            maxs.y - mins.y,
                        maxs.z - mins.z);
        scale /= totalScale;
        localPosition = new Vector3(maxs.x / totalScale - (scale.x / 2),
                                    maxs.y / totalScale - (scale.y / 2),
                                    maxs.z / totalScale - (scale.z / 2));
}
```

Código 15 - Método *SetVectors3* da classe *ActionFieldCube*

A classe *ActionField* está associada a todo o campo de ação e é responsável pela sua rotação no eixo Y e movimentação no eixo Z. A primeira é efetuada através da ação de "arrastar" pelo rato e a segunda através do *scroll* da roleta do rato, como apresentado no Código 16.

```
void Update() {
    if (Input.GetMouseButton(0)) {
        transform.Rotate(Vector3.up * Input.GetAxis("Mouse X") * 5f);
    }
    if (Input.GetAxis("Mouse ScrollWheel") != 0)
        newPosition = transform.position - Vector3.forward *
                            Input.GetAxis("Mouse ScrollWheel") * 50f;
    else
        transform.position = Vector3.Lerp(transform.position,
                                    newPosition, Time.deltaTime * 5);
}
```

Código 16 – Método *Update()* da classe *ActionField*

Ao ser alterada a tentativa ou o nível, a classe *OnScreenGUI* obtém os respetivos dados da criança e envia-os novamente para a classe *ActionField*, que efetua novamente o processo de construção do campo de ação.

3.4.6 Sons

Para a reprodução de sons no *Unity 3D* são necessários dois componentes associados a um *GameObject*: *AudioSource*, responsável por emitir o som que tem atribuído, e *AudioListener*, responsável por captar todos os sons.

Tanto nos cenários das ilhas como no menu principal foram associadas à camara os dois componentes: *AudioListener*, que representa a vista do utilizador e portanto a sua posição, e *AudioSource*, com a música ambiente.

Para a emissão de sons no clique dos botões foi associado um *AudioSource* com o respetivo som ao *GameObject* que contém as classes responsáveis pela apresentação de objetos na camada *GUI: ScreenGUI* e *MenuScreenGUI.* O componente *AudioSource* foi também associado aos *GameObject's* da moeda, estrela, *checkpoint*, hélice e texto "início", nos cenários das ilhas, e o som é imitido aquando a colisão do avião com os mesmos.

Para a reprodução do som importado para o *AudioSource* apenas é necessário invocar o método *Play()* da propriedade *audio* do objeto *MonoBehavior*, como apresentado no Código 17.

```
this.audio.Play();
```

Código 17 - Reprodução do som do componente *AudioSource*

Para os sons de ambiente foi configurada a opção de ciclo infinito do *AudioSource* que permite que a reprodução do seu som se prolongue indeterminadamente.

Para o som da hélice foi utilizada a propriedade *pitch* do *AudioSource* que permite manipular o tom. Neste caso, o *pitch* do som da hélice varia entre zero e um, consoante a velocidade da mesma, resultando no efeito de som pretendido e mais próximo da realidade. O Código 18 apresenta o método *Update()* da classe *Propeller*, associada ao *GameObject* da hélice, onde o *pitch* é igualado à percentagem da velocidade da rotação da mesma.

```
void Update() {
        this.audio.pitch = speed / maxSpeed;
}
```

Código 18 - Método *Update()* da classe *Propeller*

3.5 Testes

As seguintes secções apresentam os testes realizados ao longo do desenvolvimento do projeto.

3.5.1 Tipo de testes

Para uma melhor compreensão dos testes realizados ao longo do projeto os mesmos são divididos em cinco tipos distintos: Testes Unitários, Teste de Integração, Testes de Sistema, Testes *Alpha* e Testes *Beta*.

Os Testes Unitários foram efetuados sempre que era implementado um novo método, validando especialmente os que envolviam entrada e saída de dados. A técnica escolhida para a realização destes denomina-se de "Teste de Caixa-Negra", que consiste na comparação de resultados obtidos com resultados esperados.

Sempre que foi encontrado um erro foi efetuada uma traçagem ao código até ser corrigido o problema.

Após conclusão dos métodos, foram realizados Testes de Integração aos mesmos. Esta abordagem incremental permitiu isolar os erros facilitando o processo da sua correção. Estes testes permitem verificar o funcionamento dos vários componentes como um todo e têm como principal objetivo verificar os requisitos funcionais e não funcionais, procurando por possíveis falhas nas funcionalidades do sistema.

Os Testes de Sistema seguem-se aos Testes de Integração e têm como objetivo testar o sistema do ponto de vista do utilizador final, validando assim todas as funcionalidades do sistema.

Numa fase final, são efetuados os Testes *Alpha* que consistem na utilização da aplicação pelo utilizador final, em ambiente de desenvolvimento, na presença do responsável pelo desenvolvimento que regista erros e problemas de usabilidade.

Por último, os Testes *Beta*. Estes seguem-se aos Testes *Alpha* e diferenciam-se no facto da aplicação ser utilizada em ambiente de produção e, geralmente, na ausência do responsável do projeto.

3.5.2 Cenários das ilhas

Durante a realização dos Testes Unitários e de Integração foram encontradas algumas dificuldades, principalmente nos métodos de criação e disposição dos bónus e *checkpoints.* Todos os obstáculos foram resolvidos graças à abordagem incremental utilizada que facilitou a localização dos erros.

Como os Testes de Integração foram intensos, na realização dos Testes de Sistema foram detetados poucos erros, sendo o mais relevante na transição do jogo com o menu principal, que facilmente foram corrigidos.

Uma vez que o público-alvo são crianças com necessidades especiais, os testes *Alpha* e *Beta* foram fundamentais para o sucesso do projeto. Na realização dos Testes de *Alpha* foram detetadas poucos erros, sendo as mais relevantes na altura do avião, que ia aumentando ligeiramente, e na repetição do nível. Para além de falhas técnicas, estes testes permitiram também detetar dificuldades na calibração das mãos através do dispositivo *Leap Motion*, que obrigaram a um cuidado especial na sua correção.

Após efetuadas as melhorias e correções dos problemas detetados nos Testes *Alpha*, foi entregue um executável do jogo ao terapeuta juntamente com o *Leap Motion* durante quatro dias (Testes *Beta*). Foram detetados problemas de compatibilidade do jogo e *Leap Motion* com o sistema *Mac OS X*, que serão resolvidos numa fase posterior, quando o mestrando tiver à sua disposição uma máquina equipada com esse sistema operativo. Contudo, os resultados

deste período foram positivos revelando uma fácil e motivante interação entre as crianças e o jogo através do *Leap Motion*.

Nas Tabelas 19 e 20 são apresentados os testes mais significativos efetuados aos cenários das ilhas e respetivos resultados, bem como uma breve descrição dos erros encontrados.

Tabela 19 - Lista dos testes mais significativos dos cenários das ilhas e respetivos resultados

Descrição	Resultado	Resultado após correção
Colisões	Sucesso	
Apresentação do *score*	Sucesso	
Ecrã com diferentes resoluções	Sucesso	
Preparação dos bónus e *checkpoints*	Falha	Sucesso
Levantamento do avião	Falha	Sucesso
Aterragem do avião	Falha	Sucesso
Controlo do avião com o *Leap Motion*	Sucesso	
Ações de pausa e reinício do nível	Falha	Sucesso
Aquisição de dados através do *Leap Motion*	Falha	Sucesso
Reprodução de sons	Sucesso	
Correr a aplicação no sistema *Mac OS X*	Falha	(em espera)
Conclusão do primeiro nível pela criança	Sucesso	

Tabela 20 - Lista das falhas mais significativas dos cenários das ilhas e respetivas descrições

Erro	Descrição
Preparação dos bónus e checkpoints	A escala do checkpoint não estava a ser bem aplicada devido à escala existente no próprio objeto e foi corrigido o cálculo.
Levantamento do avião	Teve de ser aumentada a rotação do avião no eixo X para que este colidisse com sucesso no texto “Início”.
Aterragem do avião	O segundo ponto de aterragem estava demasiado próximo do chão e em alguns casos o avião colidia com o mesmo, sendo corrigida a sua posição.
Ações de pausa e reinício do nível	O tempo de jogo não estava a ser parado na segunda tentativa do nível e o reinício dava erro, sendo corrigido.
Aquisição de dados através do Leap Motion	As variáveis estavam a ser instanciadas na mesma linha (min = max = new Vector3[2];) e uma das variáveis ficava como apontador da outra.
Correr a aplicação no sistema *MAC OS X*	Foram efetuados testes rápidos e bibliotecas do *Leap Motion* não são reconhecidas, mesmo seguindo as

	instruções do site do dispositivo. Para a sua correção é necessário um computador com o sistema *MAC OS X* para testes mais demorados.

Na Tabela 20 são apresentadas as dificuldades detetadas nos Testes *Alpha* e *Beta* efetuados aos cenários das ilhas e uma breve descrição da solução aplicada para colmatar as mesmas.

Tabela 21 - Lista das dificuldades encontradas nos Testes *Alpha e Beta* dos cenários das ilhas e respetivas melhorias

Dificuldade	**Melhoria**
Calibração das mãos	Foi confirmada a importância da altura da superfície onde se posiciona o *Leap Motion* face à altura dos diferentes utilizadores. Foi também diminuída a sensibilidade na calibração, abrangendo assim uma maior área correta para colocar as mãos no início do nível.

3.5.3 Menu principal

Nos Testes Unitários efetuados ao menu principal foram encontrados algumas falhas principalmente nos gráficos de barras e campo de ação, sendo todas elas resolvidas de seguida.

Já na realização dos Testes de Integração e Sistema foram encontradas dificuldades na apresentação de texturas 2D na camada *GUI* pois o resultado destas não estava com a qualidade pretendida. Estas dificuldades obrigaram a um desenho e implementação mais precisos na resolução correta.

No menu principal, os Testes *Alpha* mostraram-se importantes, não só para detetar alguma falhas, como também para receber o *feedback* do terapeuta. Foram detetados erros na transição do cenário da ilha para o menu principal quando existia mais do que um jogador na base de dados. Durante estes testes, para a navegação no menu principal foi preferida a utilização do rato em detrimento do *Leap Motion*, uma vez que o mesmo não apresenta vantagens face ao rato para este propósito pois torna a navegação um pouco mais difícil e

demorada. Este facto não foi surpreendente, uma vez que o próprio mestrando já tinha previsto o mesmo. A experiência da interação do terapeuta com o menu principal revelou-se positiva, inclusive a análise dos dados de uma criança foi de imediato utilizada no auxílio do seu diagnóstico, nomeadamente no respeitante à mão dominante.

Durante os Testes *Beta* não foram detetados erros no menu principal mas foram sugeridas melhorias, nomeadamente legendas no campo de ação.

Nas Tabelas 22 e 23 são apresentados os testes mais significativos efetuados ao menu principal e respetivos resultados, bem como uma breve descrição dos erros encontrados.

Tabela 22 - Lista dos testes mais significativos do menu principal e respetivos resultados

Descrição	Resultado	Resultado após correção
Apresentação de texto na *GUI*	Falha	Sucesso
Apresentação de texturas na *GUI*	Falha	Sucesso
Navegação entre ecrãs	Sucesso	
Importação e exportação de dados *XML*	Sucesso	
Ecrã com diferentes resoluções	Sucesso	
Gestão de jogadores	Falha	Sucesso
Controlo do cursor com o *Leap Motion*	Falha	Sucesso
Apresentação dos resultados do nível	Falha	Sucesso
Reprodução de sons	Sucesso	
Construção dos gráficos de barras e campo de ação	Falha	Sucesso

Tabela 23 - Lista das falhas mais significativas do menu principal e respetivas descrições

Erro	Descrição
Apresentação de texto na *GUI*	Não era possível a apresentação de caracteres especiais dinamicamente e a alteração da codificação dos ficheiros *CSharp* resolveu o problema.
Apresentação de texturas na *GUI*	As imagens não estavam nítidas e foi concluído que as mesmas, quando importadas para o *Unity 3D,* são transformadas para uma resolução cujos números estejam contidos na função 2^{x}, sendo **x** um número inteiro. Todas as imagens da aplicação foram ajustadas e importadas novamente.
Gestão de jogadores	A edição do jogador não estava a funcionar a 100% pois uma variável não era atualizada devido a um problema com a

	caixa de inserção de texto.
Controlo do cursor com o *Leap Motion*	O método apresentado no fórum do *Unity 3D* contém falhas quando a camara não está na posição de origem (0,0,0). Foi então desenvolvido de raiz um novo método de movimento do cursor.
Apresentação dos resultados do nível	O total das estrelas e moedas estava a ser erradamente acumulado no menu principal em diferentes jogadores e foi imediatamente corrigido.
Construção dos gráficos de barras e campo de ação	O valor das escalas a aplicar às barras e cubos não estava a ser corretamente calculado para diferentes escalas de valores e foi corrigido de seguida.

Na Tabela 24 são apresentadas as sugestões dadas pelo terapeuta após os Testes *Alpha* e *Beta* efetuados ao menu principal. As mesmas são de grande valor para a melhoria do projeto e análise e desenvolvimento destas serão efetuados numa fase posterior, não sendo registados ainda neste documento.

Tabela 24 - Lista das sugestões dadas após os Testes *Alpha* e *Beta* do menu principal e respetivas melhorias

Ecrã	Sugestão
Gráficos de barras e campo de ação	Utilização de legendas para os níveis, tentativas e mão respetiva do cubo.

3.5.4 QEF

O *QEF* (*Quantitative Evaluation Framework*) é uma ferramenta que permite avaliar o *software* ao longo do seu ciclo de vida. O seu planeamento foi elaborado cuidadosamente na fase de análise do projeto onde foram decididos os objetivos de avaliação, posteriormente avaliados por uma pessoa que não o responsável pelo desenvolvimento. A utilização da ferramenta *QEF* neste projeto garante a qualidade do produto final. (Escudeiro & Escudeiro, 2012)

Para o *QEF* desenvolvido para este projeto foram definidos 53 requisitos divididos em sete categorias e a avaliação final resultou em 90%.

O documento encontra-se no Anexo 2.

3.6 Dificuldades encontradas

Ao longo do desenvolvimento do projeto foram encontrados vários desafios. Cada desafio obrigou a uma pesquisa rigorosa, em que foram estudadas as várias alternativas possíveis, sendo algumas delas muitas vezes implementadas e testadas para verificar se realmente funcionavam.

Sem dúvida, o grande desafio do projeto foi a criação de uma aplicação de raiz unicamente pelo mestrando, fator que, embora muito estimulante, o obrigou a um enorme esforço nas diferentes áreas abrangidas.

Os principais desafios técnicos encontrados prendem-se com a utilização do *Leap Motion* e do *Unity 3D* uma vez que o mestrando não tinha qualquer tipo de experiência nestas ferramentas. Enfatiza-se aqui a grande dificuldade de desenvolvimento de gráficos no *Unity 3D*, não sendo esta ferramenta de todo a mais indicada para este propósito.

O maior contratempo ocorrido no desenvolvimento do projeto foi a mudança de emprego do mestrando. Este teve um impacto negativo nas horas disponíveis para dedicar ao projeto e aconteceu na fase de implementação do projeto, o que dificultou o cumprimento do planeamento. Consequentemente, foi encurtado o desenvolvimento de tarefas, nunca prejudicando a qualidade nem os objetivos do projeto, mas condicionando o tempo disponível para tratamento do *feedback* obtido nos Testes *Alpha* e *Beta*. Nenhuma informação foi descartado mas sim adiada para trabalho futuro, não sendo documentado ainda neste relatório.

Conclusões

Neste último capítulo do relatório serão apresentadas as principais conclusões do projeto, abordando-se também as limitações do projeto e possíveis desenvolvimentos futuros.

4.1 Objetivos realizados

Considera-se que os objetivos a que o projeto se propôs foram alcançados com sucesso, noção que foi comprovada no êxito da fase de testes. Foi clara a motivação e entusiasmo mostrados pelas crianças ao jogar bem como a utilidade da aplicação para o terapeuta na avaliação do progresso, tornando-a assim uma mais-valia na terapia de crianças com dificuldades de integração bilateral motora.

4.2 Limitações e trabalho futuro

Relativamente a este subcapítulo, podem ser referidos alguns aspetos a ter em conta. Apesar de todos os objetivos principais terem sido cumpridos com sucesso, existem aspetos passiveis de serem desenvolvidos.

Os próximos desenvolvimentos do projeto incidirão na interface apresentada ao terapeuta, nomeadamente na área de progresso da criança. Também é pretensão do mestrando acrescentar, a longo prazo, novas componentes ao jogo, como novas ilhas, *upgrades* do avião e bónus ao longo do nível (i.e. rampa de aceleração). Pretende-se também acrescentar ao

início da aplicação uma pequena animação 3D introdutória alusiva ao jogo, bem como a apresentação de algumas dicas sobre as regras do mesmo durante a transição entre o menu principal e os cenários das ilhas.

Apesar de haver algum *feedback* do utilizador final obtido na fase de testes, o tempo permitirá uma ampla utilização da aplicação que resultará numa resposta mais completa e detalhada, possibilitando assim a análise e melhoria do projeto por parte do mestrando.

Um aspeto a acrescentar ao projeto, já discutido com o terapeuta, consiste na apresentação de uma média que será assumida como ideal. Esta média será obtida pela análise dos dados devolvidos pela aplicação numa amostra de crianças com desenvolvimento bilateral motor normal, com o mesmo intervalo de idades do público-alvo do projeto. Esse valor médio funcionará como termo de comparação que permitirá uma melhor análise do progresso de crianças pertencentes ao público-alvo.

Com o possível crescimento do projeto, pondera-se a hipótese de uma análise de dados em ferramentas externas ao *Unity 3D*, mais adequadas a este propósito, necessitando apenas da exportação de dados das crianças, funcionalidade já contemplada pelo projeto.

4.3 Apreciação final

A realização do projeto foi um grande marco no meu percurso curricular e pessoal, uma vez que me permitiu explorar áreas desafiantes, nas quais não tinha experiência.

Foi de enorme prazer desenvolver este projeto de raiz e sentir-me capaz de representar diferentes papéis no desenvolvimento do *software*.

A comunicação com diferentes áreas profissionais foi enriquecedora e demonstrou a importância da integração multidisciplinar para a elaboração de projetos.

Por último, a confirmação da utilidade do projeto é motivo de enorme gratificação e, sem dúvida, vale todo o esforço e dedicação despendidos na elaboração do projeto.

Referências

Anón (sem data) «Bfxr. Make sound effects for your games.», [em linha] Available from: http://www.bfxr.net/ (Acedido 19 Outubro 2014a).

Anón (sem data) «CodeProject - For those who code», [em linha] Available from: http://www.codeproject.com/ (Acedido 24 Outubro 2014b).

Anón (sem data) «Saving and Loading Data: XmlSerializer - Unify Community Wiki», [em linha] Available from: http://wiki.unity3d.com/index.php?title=Saving_and_Loading_Data:_XmlSerializer (Acedido 20 Outubro 2014c).

Anón (sem data) «Unity 3D Asset Store», [em linha] Available from: https://www.assetstore.unity3d.com/en/ (Acedido 19 Outubro 2014d).

Ayres, A. J. (1974) *The development of sensory integrative theory and practice: A collection of the works of A. Jean Ayres*,.

Barcala, L., Grecco, L. A. C., Colella, F., Lucareli, P. R. G., Salgado, A. S. I. e Oliveira, C. S. (2013) «Visual biofeedback balance training using wii fit after stroke: a randomized controlled trial.», *Journal of physical therapy science*, 25, pp 1027–32, [em linha] Available from: http://www.pubmedcentral.nih.gov/articlerender.fcgi?artid=3820213&tool=pmcentrez&rendertype=abstract.

Deng, S., Li, W.-G., Ding, J., Wu, J., Zhang, Y., Li, F. e Shen, X. (2014) «Understanding the mechanisms of cognitive impairments in developmental coordination disorder.», *Pediatric research*, 75, pp 210–6, [em linha] Available from: http://www.ncbi.nlm.nih.gov/pubmed/24192703.

Donato, H. (2014) «Gestores de Referências Bibliográficas»,.

E-Games (sem data) «E-Games - Educational Games as Purdue», [em linha] Available from: http://www.e-games.tech.purdue.edu/default.asp (Acedido 7 Outubro 2014).

Escudeiro, P. e Escudeiro, N. (2012) «Evaluation of serious games in mobile platforms with QEF: QEF (Quantitative Evaluation Framework)», Em *Proceedings 2012 17th IEEE International Conference on Wireless, Mobile and Ubiquitous Technology in Education, WMUTE 2012*, pp 268–271.

Getting It (sem data) «GETTING it | Pediatria e Desenvolvimento Lda», [em linha] Available from: http://www.gettingit.org/ (Acedido 5 Outubro 2014).

kinnealey, M., & Miller, L. J. (1993) «kinnealey_miller.pdf», Em *Sensory Integration/ Learning Disabilities.*, pp 474–489.

Li, X., Atkins, M. S. e Stanton, B. (2006) «Effects of Home and School Computer Use on School Readiness and Cognitive Development Among Head Start Children: A Randomized Controlled Pilot Trial», *Merrill-Palmer Quarterly*.

Lin, C. K. e Wu, H. M. (2013) «Development and validation of the computerized bilateral motor coordination test», *Research in Developmental Disabilities*.

Magalhães, L. C., Nascimento, V. C. S. e Resende, M. B. (2004) «Avaliação da coordenação e destreza motora-ACOORDEM: etapas de criação e perspectivas de validação», *Revista de Terapia Ocupacional da Universidade de São Paulo*, 15, pp 17–25.

Microsoft (sem data) «MSDN Library», [em linha] Available from: http://msdn.microsoft.com/library/ (Acedido 13 Outubro 2014).

Motion Leap (sem data) «Menu Design Guides», [em linha] Available from: https://developer.leapmotion.com/documentation/csharp/practices/Leap_Menu_Design_Guidelines.html.

Naranjo-Bock, C. (2011) «Effective Use of Typography in Applications for Children :: UXmatters», *www.uxmatter.com*, [em linha] Available from: http://www.uxmatters.com/mt/archives/2011/06/effective-use-of-typography-in-applications-for-children-3.php (Acedido 26 Outubro 2014).

Sturm, D. (2013) «Richard RIESE», [em linha] Available from: http://damiansturm.prosite.com/182175/1912152/work/richard-riese (Acedido 8 Outubro 2014).

Tarakci, D., Ozdincler, A. R., Tarakci, E., Tutuncuoglu, F. e Ozmen, M. (2013) «Wii-based Balance Therapy to Improve Balance Function of Children with Cerebral Palsy: A Pilot Study.», *Journal of physical therapy science*, 25, pp 1123–7, [em linha] Available from: http://www.pubmedcentral.nih.gov/articlerender.fcgi?artid=3818755&tool=pmcentrez&rendertype=abstract.

Zwicker, J. G., Missiuna, C., Harris, S. R. e Boyd, L. a (2012) «Developmental coordination disorder: a review and update.», *European journal of paediatric neurology : EJPN : official journal of the European Paediatric Neurology Society*, 16, pp 573–81, [em linha] Available from: http://www.ncbi.nlm.nih.gov/pubmed/22705270.

Anexo 1

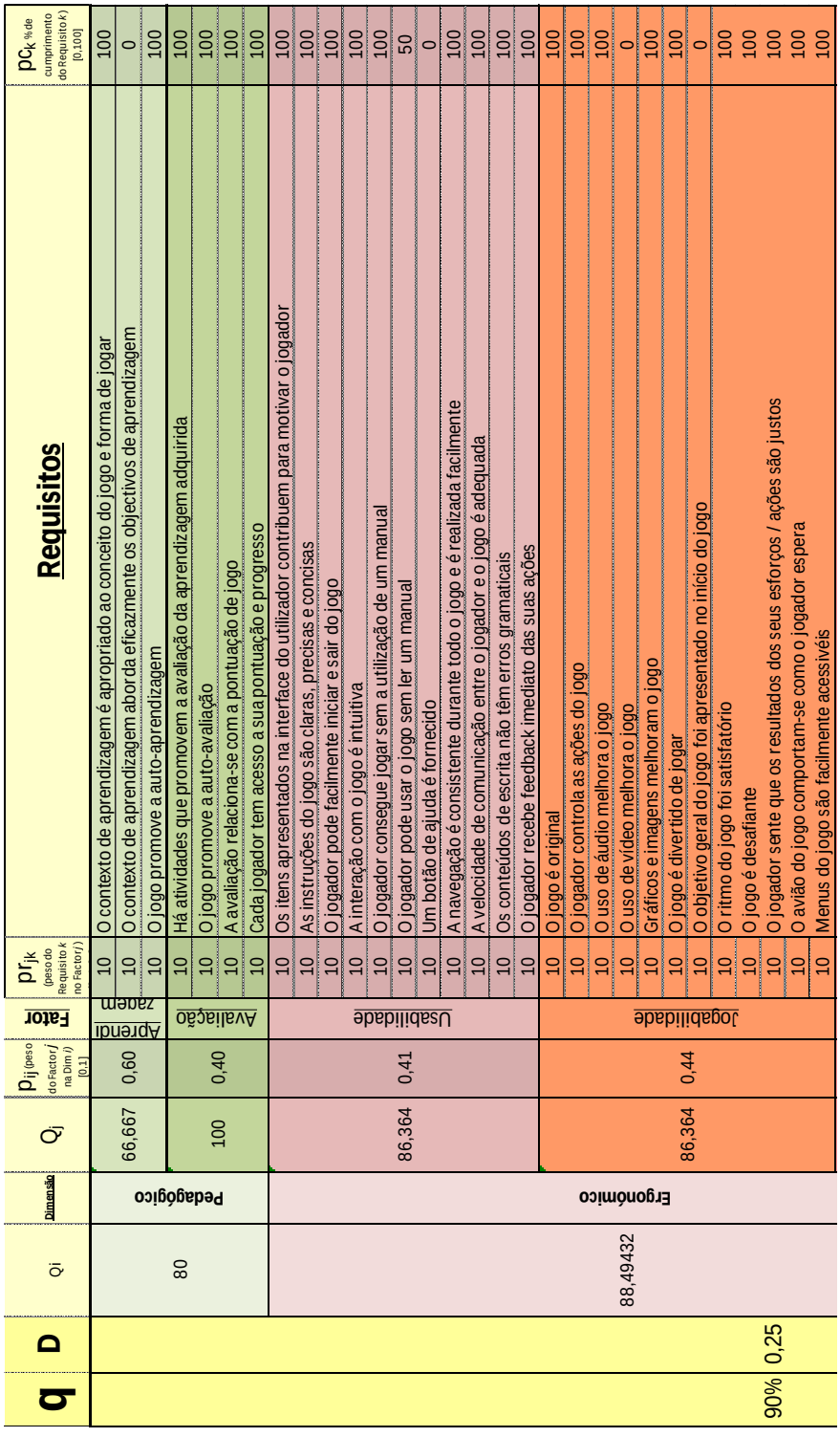
Exemplo de ficheiro *XML* da base de dados com um jogador e um registo de nível


```
<?xml version="1.0" encoding="Windows-1252"?>
<PlayersCollection xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <players>
       <player>
      <name>Pedro</name>
      <avatar>menino</avatar>
      <dominantHand>direita</dominantHand>
      <creationDate>2014-10-15T23:37:58.1597525+01:00</creationDate>
      <listLevelData>
        <LevelData>
          <level>1</level>
          <times>
            <int>75</int>
          </times>
          <livesLosted>1</livesLosted>
          <coins>
            <boolean>true</boolean>
            <boolean>true</boolean>
            <boolean>true</boolean>
            <boolean>true</boolean>
            <boolean>true</boolean>
            <boolean>true</boolean>
            <boolean>true</boolean>
            <boolean>true</boolean>
            <boolean>true</boolean>
            <boolean>true</boolean>
            <boolean>true</boolean>
            <boolean>true</boolean>
            <boolean>true</boolean>
            <boolean>true</boolean>
            <boolean>true</boolean>
            <boolean>true</boolean>
            <boolean>true</boolean>
            <boolean>true</boolean>
            <boolean>true</boolean>
            <boolean>true</boolean>
            <boolean>true</boolean>
            <boolean>true</boolean>
            <boolean>true</boolean>
            <boolean>true</boolean>
            <boolean>true</boolean>
            <boolean>true</boolean>
            <boolean>true</boolean>
          </coins>
          <stars>
            <boolean>true</boolean>
            <boolean>true</boolean>
            <boolean>true</boolean>
```

```
            </stars>
            <leftHandDistances>
              <Vector3>
                <x>41.73</x>
                <y>198.44</y>
                <z>34.99</z>
              </Vector3>
            </leftHandDistances>
            <rightHandDistances>
              <Vector3>
                <x>28.11</x>
                <y>112.48</y>
                <z>22.57</z>
              </Vector3>
            </rightHandDistances>
            <leftHandMins>
              <Vector3>
                <x>-13.86</x>
                <y>-4.84</y>
                <z>-8.65</z>
              </Vector3>
            </leftHandMins>
            <leftHandMaxs>
              <Vector3>
                <x>-9.04</x>
                <y>7.84</y>
                <z>-3.38</z>
              </Vector3>
            </leftHandMaxs>
            <rightHandMins>
              <Vector3>
                <x>12.45</x>
                <y>-4.05</y>
                <z>-6.85</z>
              </Vector3>
            </rightHandMins>
            <rightHandMaxs>
              <Vector3>
                <x>13.93</x>
                <y>3.03</y>
                <z>-3.36</z>
              </Vector3>
            </rightHandMaxs>
          </LevelData>
        </listLevelData>
      </player>
    </players>
  </PlayersCollection>
```

Anexo 2

QEF (Quality Evaluation Framework)



q	D	Qi	Dimensão	Qj	p_{ij} (peso do Factor *j* na Dim *i*) [0,1]	Factor	pr_{jk} (peso do Requisito *k* no Factor *j*)	Requisitos	pc_k % de cumprimento do Requisito *k*) [0,100]
90%	0,25	80	Pedagógico	66,667	0,60	Aprendizagem	10	O contexto de aprendizagem é apropriado ao conceito do jogo e forma de jogar	100
							10	O contexto de aprendizagem aborda eficazmente os objectivos de aprendizagem	0
							10	O jogo promove a auto-aprendizagem	100
				100	0,40	Avaliação	10	Há atividades que promovem a avaliação da aprendizagem adquirida	100
							10	O jogo promove a auto-avaliação	100
							10	A avaliação relaciona-se com a pontuação de jogo	100
							10	Cada jogador tem acesso a sua pontuação e progresso	100
		88,49432	Ergonómico	86,364	0,41	Usabilidade	10	Os itens apresentados na interface do utilizador contribuem para motivar o jogador	100
							10	As instruções do jogo são claras, precisas e concisas	100
							10	O jogador pode facilmente iniciar e sair do jogo	100
							10	A interação com o jogo é intuitiva	100
							10	O jogador consegue jogar sem a utilização de um manual	100
							10	O jogador pode usar o jogo sem ler um manual	50
							10	Um botão de ajuda é fornecido	0
							10	A navegação é consistente durante todo o jogo e é realizada facilmente	100
							10	A velocidade de comunicação entre o jogador e o jogo é adequada	100
							10	Os conteúdos de escrita não têm erros gramaticais	100
							10	O jogador recebe feedback imediato das suas ações	100
				86,364	0,44	Jogabilidade	10	O jogo é original	100
							10	O jogador controla as ações do jogo	100
							10	O uso de áudio melhora o jogo	100
							10	O uso de vídeo melhora o jogo	0
							10	Gráficos e imagens melhoram o jogo	100
							10	O jogo é divertido de jogar	100
							10	O objetivo geral do jogo foi apresentado no início do jogo	0
							10	O ritmo do jogo foi satisfatório	100
							10	O jogo é desafiante	100
							10	O jogador sente que os resultados dos seus esforços / ações são justos	100
							10	O avião do jogo comportam-se como o jogador espera	100
							10	Menus do jogo são facilmente acessíveis	100

(continuação do Anexo 2)

								10	O idioma do jogo é adequado ao público-alvo	100
						Socio-Cultural		10	O jogo não apresenta nenhum conteúdo ofensivo ou estereótipos em termos de gênero, raça, religião e diversidade cultural	100
								10	O jogo não inclui conteúdo que incite à violência	100
								10	O jogo respeita as normas de acessibilidade que garantem o acesso para todos os utilizadores, incluindo aqueles com necessidades especiais	100
						Suporte		10	O jogo é estável e é executado sistematicamente sem falhas	100
								10	A instalação do jogo é fácil para o público alvo	0
								10	Os dados do jogador são assegurados em caso de falha do jogo	100
								10	O jogo pode ser facilmente atualizado	100
								10	Há um ponto de entrada para o jogo claro e simples	100
						Outros		10	A apresentação de conceitos é clara e oportuna	100
								10	Os símbolos e metáforas são adequados e coerentes	100
								10	O jogo é agradável para repetir	100
								10	Os efeitos das ações do jogador podem ser visto claramente no ambiente de jogo	100
								10	O som do jogo ajuda a envolver o jogador no ambiente	100
								10	Os gráficos do jogo ajudam a envolver o jogador no ambiente	100
								10	O jogo oferece recompensa suficiente para as acções de sucesso do jogador	100
								10	Feedback audio-visual é fornecido com cada ação do jogador	100
								10	Os controles para o jogo foram os mesmos durante todo o jogo	100
								10	As cores utilizadas no jogo foram consistentes ao longo do jogo	100
								10	O jogo permite o desenvolvimento cognitivo para que o utilizador seja capaz de o usar noutros contexto	100
								10	O jogo desenvolve no utilizador a capacidade de avaliar criticamente e produzir novos conhecimentos	0
								10	A aplicação permite ao terapeuta seguir o progresso do jogador	100
								10	Os dados são devolvidos ao terapeuta de forma mensurável	100