



ISEL – INSTITUTO SUPERIOR DE ENGENHARIA DE LISBOA
ADEETC – ÁREA DEPARTAMENTAL DE ENGENHARIA DE
ELECTRÓNICA E TELECOMUNICAÇÕES E DE COMPUTADORES

LEIM

LICENCIATURA EM ENGENHARIA INFORMÁTICA E MULTIMÉDIA
UNIDADE CURRICULAR DE PROJETO

Shuman Craft

**SHUMAN
CRAFT**

Shun Wang (45410)

Arman Freitas (45414)

Orientador

Professor [Doutor] Hugo Cordeiro

Setembro, 2020

Resumo

Este projeto consiste no desenvolvimento de um jogo 3d *co-op multiplayer* num estilo mais simples, não realista.

O objetivo consiste na cooperação dos jogadores de modo a manterem-se vivos. De acordo com as suas necessidades e ambições, terão de caçar, construir, etc. O jogo ronda à volta da sobrevivência e, portanto, cada jogador deve estar sempre atento às características do seu personagem (vida, fome, sede).

O jogo é constituído por ilhas sendo que cada uma representa um bioma. Cada uma tem animais diferentes, consoante o ambiente, assim como materiais diferentes que os jogadores podem encontrar (ferro, cobre, etc.).

A motivação para desenvolver o jogo foi o facto de poder realizar construções e, qualquer tipo de pessoa se possa divertir com amigos (ou mesmo sozinho, caso queira). Outro aspeto importante na decisão da criação do projeto foi o facto de jogos de sobrevivência, tais como este, têm uma característica diferente dos outros tipos. Não é necessário seguir uma linha de história e, não existe um fim ao jogo. A situação ideal seria os programadores continuarem a desenvolver elementos que enriquecem o mesmo, consoante o *feedback* dos utilizadores.

Abstract

This project consists of the development of a multiplayer 3D co-op game in a simple, unrealistic style.

The objective consists in the cooperation of the players in order to stay alive. According to your needs and ambitions, you will have to hunt, build, etc. The game is about survival, so each player must always be aware of his character's characteristics (life, hunger, thirst).

The game is made up of islands, each of which represents a biome. Each one has different animals, depending on the environment, as well as different materials that the players can find (iron, copper, etc.).

The motivation for developing the game was the fact that it could perform contructions and, any type person can have fun with friends (or even yourself, if you want). Another important aspect in the decision to create the project was the fact that survival games, such as this one, have a different characteristic from the other types. It isn't necessary to follow a history line and there is no end to the game. The ideal situation would be for programmers to continue to develop elements that enrich it, depending on users' feedback.

Agradecimentos

Gostaríamos de agradecer a todos os colegas e docentes que ajudaram com este percurso académico, especialmente o docente Hugo Cordeiro que esteve sempre disposto a ajudar no desenvolvimento do projeto, e o docente Paulo Trigo que nos deu os alicerces para desenvolver um projeto mais elaborado e coerente e também o facto que se mostrar disponível para responder a quaisquer questões.

Insanity is doing the same thing over and over again

and expecting different results... .

Albert Einstein

Índice

Resumo	i
Abstract	iii
Agradecimentos	v
Índice	ix
Lista de Tabelas	xiii
Lista de Figuras	xv
1 Introdução	1
2 Trabalho Relacionado	5
2.1 The Sims 2: Castaway	5
2.2 Raft	6
2.3 Minecraft	7
2.4 Ark: Survival Evolved	8
3 Modelo Proposto	11
3.1 Requisitos	11
3.2 Caracterização Geral	12
3.2.1 Síntese de Objectivos	12
3.2.2 Clientes alvo	12
3.2.3 Metas a alcançar	12
3.3 Caracterização Pormenor	12
3.3.1 Funções do Sistema	12
3.3.2 Atributos do Sistema	14

3.3.3	Atributos e funções do sistema	15
3.3.4	Casos de Utilização	15
3.4	Fundamentos	19
3.5	Abordagem	21
4	Implementação do Modelo	23
4.1	Tecnologias	23
4.1.1	Adobe Photoshop	25
4.1.2	Blender, Maya e Mixamo	26
4.1.3	Unity	28
4.1.4	Audacity	29
4.2	Itens	30
4.2.1	Sistema de itens	30
4.2.2	Sistema de Inventário	33
4.2.3	Sistema de <i>Crafting</i>	34
4.3	Criação do Mapa	35
4.4	Câmara	38
4.4.1	Espaço de cores	41
4.4.2	Aspetos técnicos	41
4.4.3	Pós-Processamento	44
4.5	Animações 3D	50
4.5.1	Movimentos	50
4.5.2	Animações Simples	51
4.6	Som	53
4.6.1	Som Ambiente	54
4.6.2	Som dos passos do personagem	55
4.7	Arquitetura da rede	56
4.7.1	Sincronização de objetos pela rede	56
4.7.2	Instanciamento de objetos pela rede	57
4.7.3	Guardar e Carregar jogo	57
4.8	Sistema de mensagens	60
4.9	Interface Gráfica (GUI)	61
4.9.1	<i>Canvas</i> do Unity	61
4.9.2	Animações do UI	68
4.9.3	Implementação da Interface Gráfica	69
4.10	Máquinas de Estados	73

<i>CONTENTS</i>	xi
5 Validação e Testes	75
6 Conclusões e Trabalho Futuro	81
A Questionário de Usabilidade do jogo	83

Lista de Tabelas

3.1	Tipos de funções do sistema	13
3.2	Funções do Sistema	13
3.3	Tipos de atributos do sistema	14
3.4	Atributos do Sistema	14
3.5	Relações entre funções e atributos do sistema	15
3.6	Caso de utilização - Interface Gráfica	16
3.7	Caso de utilização - <i>Multiplayer</i>	17
3.8	Caso de utilização - Animações	17
3.9	Caso de utilização - Ambiente	17
3.10	Caso de utilização - Máquina de Estados	17
3.11	Caso de utilização - Inputs	18
3.12	Cenário do jogador iniciar um jogo	18
3.13	Cenário de sincronização <i>multiplayer</i>	19

Lista de Figuras

1.1	<i>UnReal World</i>	2
1.2	<i>Lowpoly</i> vs <i>Highpoly</i>	3
2.1	The Sims 2: Castaway	6
2.2	Raft	6
2.3	Minecraft	7
2.4	Ark: Survival Evolved	8
3.1	Casos de utilização	16
3.2	Exemplo de floresta	20
3.3	Exemplo de ilha de gelo	20
3.4	Exemplo de ilha	20
3.5	Exemplo de savana	20
3.6	Timeline do projeto	21
3.7	Ilhas e Animais	22
4.1	Interface - <i>Photoshop</i>	25
4.2	Interface - <i>Blender</i>	26
4.3	Interface - <i>Maya</i>	27
4.4	Interface - <i>Mixamo</i>	27
4.5	Interface - <i>Audacity</i>	29
4.6	Diagrama de classes dos itens	31
4.7	Exemplo de parâmetros de um item	33
4.8	Inventário do jogador	34
4.9	Hotbar do jogador	34
4.10	<i>Hotbar</i> do jogador	35
4.11	Design inicial da ilha	36
4.12	Hexágono base da criação da ilha	36

4.13 Ilha base	37
4.14 Ilha com texturas	37
4.15 Ilha com detalhes	37
4.16 Exemplo de câmera em primeira pessoa	38
4.17 Exemplo de câmera em terceira pessoa	39
4.18 Exemplo de câmera MOBA	40
4.19 <i>Linear vs Gamma</i>	41
4.20 Exemplo de colisão de câmera - 1	43
4.21 Exemplo de colisão de câmera - 2	44
4.22 Pós-processamento Detalhes - (<i>Occlusion</i>)	45
4.23 <i>Occlusion OFF vs ON</i>	46
4.24 Pós-processamento Detalhes - (<i>Color Grading</i>)	47
4.25 Pós-processamento Detalhes - (<i>Bloom Effect</i>)	48
4.26 <i>Bloom Effect OFF vs ON</i>	48
4.27 Demonstração sem pós-processamento	49
4.28 Demonstração do uso de pós-processamento	49
4.29 Gráfico cartesiano da <i>Blend Tree</i>	51
4.30 Gráfico cartesiano das curvas	52
4.31 Component <i>Audio Source</i>	53
4.32 Gráfico de som 3D - <i>Audio Source</i>	54
4.33 Arquitetura da rede através do servidor do <i>Photon</i>	56
4.34 Processo de guardar o jogo, quando um cliente saí	58
4.35 Processo de carregar a informação de um cliente, quando o mesmo entra	59
4.36 Posicionamento dos elementos	62
4.37 Screen Space - Overlay	64
4.38 Screen Space - Camera	64
4.39 World Space	64
4.40 Opções disponíveis do GridLayout	66
4.41 Exemplo de uma implementação do GridLayout	67
4.42 Opções disponíveis do VerticalLayout	67
4.43 Exemplo de uma implementação do VerticalLayout	67
4.44 Opções disponíveis do GridLayout	68
4.45 Exemplo de uma implementação do GridLayout	68
4.46 <i>Tweening</i> exemplo	69

4.47	Menu Inicial	70
4.48	Menu de Itens	71
4.49	Hotbar	71
4.50	Menu dos créditos	72
4.51	<i>Stats</i> do jogador	72
4.52	Máquina de estados - predadores	74
4.53	Máquina de estados - presas	74
5.1	Estatísticas do Questionário - Género	76
5.2	Estatísticas do Questionário - Idade	76
5.3	Estatísticas do Questionário - Conhecimento de jogos <i>survival</i>	76
5.4	Estatísticas do Questionário - Tipos de jogos que costuma jogar	77
5.5	Estatísticas do Questionário - Diversão	77
5.6	Estatísticas do Questionário - Performance	78
5.7	Estatísticas do Questionário - Visuais	78
5.8	Estatísticas do Questionário - Compreensão do objetivo do jogo	78
5.9	Estatísticas do Questionário - Jogabilidade	79
5.10	Estatísticas do Questionário - Compraria o Jogo?	79
5.11	Estatísticas do Questionário - Problemas com o jogo	79
5.12	Estatísticas do Questionário - Sugestões	80
5.13	Estatísticas do Questionário - Classificação geral	80

Capítulo 1

Introdução

Um jogo de sobrevivência, encontra-se sub-categorizado no género de jogos de ação. Neste tipo de jogos, o jogador é liberto num mapa, com mínimos recursos possíveis. O mapa tipicamente contém adversários hostis, que tentam impedir o jogador de passar certas áreas.

Este ambiente, exige que o jogador recolha recursos, construa armas, ferramentas e abrigos para que possa sobreviver maior tempo possível. Geralmente, os jogos de sobrevivência não têm fim, isto é , não existe nenhum objectivo para terminar o jogo. O utilizador apenas se diverte e encontra diferentes formas de sobreviver. O jogo considerado como primeiro exemplo deste género de sobrevivência é o UnReal World, criado por Sami Maaranen em 1992 que, ainda hoje continua a ser atualizado. O jogador é colocado em condições rigorosas na Finlândia, durante a Idade do Ferro. A única meta a alcançar é sobreviver o maior tempo possível contra criaturas selvagens e perigos criados pela tempestade de neve. De seguida, na Figura 1.1 podemos observar o aspetto do UnReal World.



Figura 1.1: *UnReal World*

O objetivo deste trabalho prende-se com o desenvolvimento de um jogo de sobrevivência (na plataforma *Windows*) *online* onde, em cada jogador são simuladas características de um ser humano tais como fome e sede. Assim, o jogador deve explorar, caçar, entre outras coisas, para poder sobreviver.

Em termos de mapa, o jogo será constituído por várias ilhas, cada uma com um ambiente diferente (deserto, floresta, etc.). O jogador só poderá transitar de ilha em ilha caso consiga passar os obstáculos que as separam.

Não irá ser adoptado um estilo realista em termos de aparência, mas sim a técnica *lowpoly*. *Lowpoly* caracteriza um objeto 3D como tendo uma malha poligonal relativamente simples, isto é, de baixa resolução. Ao contrário de objetos que tentam ter o máximo detalhe possível, objetos *lowpoly* são mais simples, sendo uma mais valia no que diz respeito ao processamento dos mesmos por parte do *hardware*.

A Figura 1.2, representa a diferença do uso de *lowpoly* vs realismo, respectivamente.

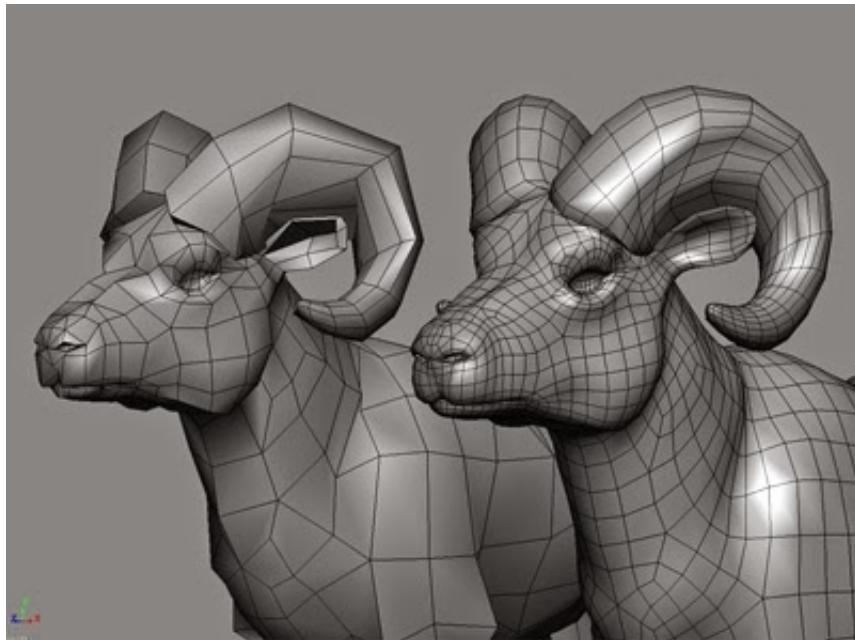


Figura 1.2: *Lowpoly* vs *Highpoly*

A motivação no desenvolvimento deste projeto, foi o facto de podermos criar um jogo que possa abranger qualquer idade, onde os jogadores possam se divertir.

Uma grande mais valia é o jogo não ter um percurso pré-definido pelos criadores. O jogador não deve seguir um caminho, mas sim pensar no seu próprio, ele é que decide quando precisa de caçar ou explorar, consoante o que necessita.

O seguinte capítulo (2) pretende mostrar uma variedade de jogos que se podem relacionar com o projeto desenvolvido.

Serão apresentados jogos como: *The Sims 2: Castaway*, *Raft*, *Minecraft*, *Ark: Survival Evolved* e o *Rust*. Alguns destes forneceram inspiração na criação de algumas características do projeto.

No capítulo 3, será explicado o modelo proposto. Antes do desenvolvimento do projeto foi proposto um modelo. Este capítulo serve para mostrar as características que pretendemos implementar no projeto.

A implementação do modelo, no capítulo 4, procura aprofundar um pouco mais o modelo proposto. Neste capítulo será explicado cada aspeto do projeto mais detalhadamente. Irá ser mostrada a nossa linha de pensamento e, as

arquiteturas adotadas.

Alguns dos tópicos abordados são os seguintes:

- Câmara
- Animações 3D
- Itens
- Som
- Arquitetura da rede
- Interface Gráfica (GUI)
- Máquina de Estado

Juntos, estes segmentos dão origem ao processo de desenvolvimento do jogo em si.

No seguinte capítulo (5), segue-se a validação e testes. Este passa-se por demonstrar o jogo a colegas e, após os mesmos jogarem-no, pedir o seu *feedback*.

Conforme o *feedback*, positivo ou negativo, são realizados pequenos ajustes que no fim dão por terminado o estado de desenvolvimento do projeto.

Por fim, as conclusões servem para refletirmos sobre o projeto desenvolvido, assim como mostrar os recursos que podem ser implementados futuramente.

Capítulo 2

Trabalho Relacionado

Os jogos de sobrevivência têm vindo a aumentar a sua popularidade nos últimos tempos. Existem vários jogos diferentes, no entanto também alguns clássicos, que nunca saíram de moda. No segmento a seguir, serão expostos alguns jogos presentes que nos inspiraram para o desenvolvimento deste projeto.

2.1 The Sims 2: Castaway

The Sims 2: Castaway, *banner* do jogo encontra-se Figura 2.1, é um jogo lançado pela *EA Games* em Outubro de 2007, é o terceiro spin-off de *The Sims 2* mas com género Survival, exclusivo para consolas. Como nos demais jogos da série, deve-se controlar a vida dos personagens, suprindo suas necessidades básicas, porém agora numa ilha deserta, pois os Sims naufragaram e precisam se unir para tentar voltar para casa.

Este género, de simulação da vida humana e carácter de sobrevivência, inspirou-nos profundamente para fazer o nosso projeto.

Não só o estilo de sobrevivência numa ilha foi o tema principal do projeto, como se pode observar ao ver o mapa do jogo, como também os modelos dos objectos e personagens que o jogo apresenta.



Figura 2.1: The Sims 2: Castaway

2.2 Raft

Raft, banner do jogo encontra-se Figura 2.2, é um jogo produzido pela *Redbeet Interactive* e publicado pela *Axolot Games* em 2018. Consiste em sobreviver num meio do oceano dentro de uma jangada. A forma de ir colectando recursos que aparecem na água para expandir e sobreviver na jangada. Este é bastante parecido ao nosso projeto, mas uma vez por ser um jogo de sobrevivência que não tem fim e de ter a possibilidade de jogar *multiplayer co-op*, ou seja, cooperação multi-jogador com amigos.



Figura 2.2: Raft

O jogo têm um sistema de construção que nos inspirou em fazer um parecido no nosso jogo, de seguida estão alguns aspectos do jogo que mais se destacam:

- *Multiplayer* com cooperação
- Criar/construir armas e utensílios para ajudar a sobreviver
- Modo de construção

2.3 Minecraft

O *Minecraft*, banner do jogo encontra-se Figura 2.3, foi desenvolvido pelo sueco Markus ”Notch” Persson em 2009 e posteriormente desenvolvido e publicado pela Mojang Studios, cujo foi obtida pela Microsoft em 2014. Em *Minecraft*, os jogadores exploram um mundo aberto tridimensional intencionalmente em blocos, pixelizado e gerado proceduralmente, podendo descobrir e extrair matérias-primas, ferramentas artesanais, construir estruturas ou terraplanagens e, dependendo do modo de jogo, podem combater NPC's (*Non Playing Characters*), bem como cooperar ou competir contra outros jogadores no mesmo mundo.



Figura 2.3: Minecraft

Este jogo têm vários aspectos que nos deram ideias no projeto nomeadamente o sistema de itens, que divide vários itens em categorias e o sistema de inventário que contém uma barra de acesso rápido e um inventário tipo mochila, de seguida temos mais alguns aspectos:

- *Multiplayer* com cooperação
- Criar/construir armas e utensílios para ajudar a sobreviver
- Defender contra perigos da natureza
- Sistema de items
- Sistema de inventário

2.4 Ark: Survival Evolved

ARK Survival Evolved, *banner* do jogo encontra-se Figura 2.4, é um jogo desenvolvido pela Studio Wildcard do gênero Ação-Aventura com um mapa Mundo Aberto. O jogo foi desenvolvido utilizando o motor gráfico Unreal Engine. O jogo consiste em sobreviver em uma ilha repleta de dinossauros e outras criaturas pré-históricas, pode-se sobreviver sozinho, em servidores dedicados ou em servidores privados com os amigos.



Figura 2.4: Ark: Survival Evolved

Este jogo têm vários aspectos parecidos com o nosso projeto, nomeadamente:

- *Multiplayer* com cooperação
- Criar/construir armas e utensílios para ajudar a sobreviver
- Defender contra perigos da natureza
- Sistema de items
- Sistema de inventário
- Motor do jogo semelhante ao *Unity* (*Unreal Engine*)

Capítulo 3

Modelo Proposto

Neste capítulo será abordado a análise de requisitos que é um aspecto importante na gestão de projetos, esta é responsável por coletar dados indispensáveis, necessários para solucionar um problema e alcançar os nossos objetivos.

3.1 Requisitos

Essa análise de requisitos é vital para o desenvolvimento do sistema, ela vai determinar o sucesso ou o fracasso do projeto. Os requisitos colhidos devem ser bem analisados, detalhados e relevantes para o projeto para tornar este mais coeso e bem estruturado. Serão abordados nesta fase:

- Caracterização geral do problema apresentado pelo trabalho
- Caracterização pormenor onde será feita uma análise mais pormenorizada sobre funções e atributos do sistema
- Funções e atributos do sistema que foram referidos na caracterização pormenor
- Casos de utilização com base no que já foi exposto nos pontos anteriores

3.2 Caracterização Geral

3.2.1 Síntese de Objectivos

Neste projeto foi proposto a criação de um jogo em *low-poly* 3D no Unity, que têm o intuito de simular a sobrevivência um humano na natureza num mapa extenso para simular um *open-world*.

3.2.2 Clientes alvo

O público-alvo serão todos os que queiram jogar um videojogo do tipo *survival*, um jogo que não requer competição entre jogadores, nem necessita de estar sempre a jogar, um jogo sem fim e sem objetivo final para descontrair e jogar com amigos no tempo livre.

3.2.3 Metas a alcançar

Pretende-se com este jogo alcançar:

- Proporcionar uma simulação do instinto humano para sobreviver
- Possibilitar o jogador de jogar em *multiplayer*
- Permitir uma boa interação usando menus gráficos
- Oferecer uma experiência de jogo agradável e relaxada

3.3 Caracterização Pormenor

3.3.1 Funções do Sistema

As funções do sistema também chamados de requisitos funcionais, representam aquilo que o sistema é suposto fazer. Cada um deste requisitos deve ser categorizada de modo a definir as prioridades, identificar os requisitos mais importantes e custos associados (temporais). Existem 3 tipos de categorias que se pode dividir os requisitos:

Categoría	Significado
Evidente	Tem que ser realizada. O utilizador tem que ter conhecimento da sua realização.
Invisível	Tem que ser realizada. Não é visível para os utilizadores.
Adorno	Opcional. Não afecta significativamente o custo ou outras funções.

Tabela 3.1: Tipos de funções do sistema

De seguida apresenta-se todas as funções do nosso sistema usando as categorias expostas anteriormente:

Ref. #	Função	Categoría
Ref. 1.1	Sistema Multiplayer	Evidente
Ref. 1.2	Registar os inputs do utilizador	Evidente
Ref. 1.3	Sincronização em tempo real entre os jogadores do servidor	Evidente
Ref. 1.4	Guardar o progresso feito de todos os jogadores do servidor	Evidente
Ref. 1.5	Captar e reproduzir sons	Evidente
Ref. 1.6	Apresentar animações do jogador	Evidente
Ref. 1.7	Aceder aos menus todos	Evidente
Ref. 1.8	Sistema de inventário e crafting	Evidente
Ref. 1.9	Sistema de colocação de objetos no mapa	Evidente
Ref. 1.10	Gerir as físicas do jogo	Invisível
Ref. 1.11	Inteligência artificial dos agentes	Invisível
Ref. 1.12	Criar um ciclo dia e noite	Adorno
Ref. 1.13	Mudar de cena cada vez que transita para outra ilha	Adorno

Tabela 3.2: Funções do Sistema

3.3.2 Atributos do Sistema

Os atributos do sistema também chamados de requisitos não-funcionais estes são as características ou dimensões do sistema. Tal como nas funções do sistema existe também diferentes formas de classificar dependendo da sua prioridade:

Categoría	Significado
Obrigatório	Tem que ser contemplado. É usual ser uma Restrição de Fronteira
Desejável	Deve estar preparado para alcançar

Tabela 3.3: Tipos de atributos do sistema

De seguida apresenta-se todos os atributos do nosso sistema usando as categorias expostas anteriormente:

Atributos	Atributo Detalhe/ Restrição de Fronteira	Categoría
Plataforma	Windows	Obrigatória
Interação Homem-Máquina	Utilização do teclado e rato	Obrigatória
Interação Homem-Máquina	Interação com a interface gráfica do jogo	Obrigatória
Networking	Necessidade de ter ligação à internet	Obrigatória
Facilidade de utilização	Aprendizagem fácil das mecânicas do jogo	Desejável
Tempos de resposta	Instantâneo	Desejável
Tolerância a erros de execução	Minimizar o quanto possível	Desejável
Custo do projeto	Menor custo possível	Desejável
Desempenho	Ter um frame rate razoável	Obrigatório

Tabela 3.4: Atributos do Sistema

3.3.3 Atributos e funções do sistema

É importante evidenciar todas as relações entre funções do sistema e atributos do sistema, descrever atributos relacionados com funções específicas. De seguida encontram-se estas relações:

Atributos do Sistema	Funções do Sistema
Plataforma	R1.1, R1.2, R1.3, R1.4, R1.5, R1.6, R1.7, R1.8, R1.9, R1.10, R1.11, R1.12, R1.13
Interação Homem-Máquina	R1.2, R1.5, R1.6, R1.7, R1.8, R1.9, R1.10, R1.13
Networking	R1.3, R1.4
Facilidade de utilização	R1.7, R1.8, R1.9
Tempos de resposta	R1.1, R1.2, R1.3, R1.4, R1.5, R1.6, R1.7, R1.8, R1.9, R1.10, R1.11, R1.12, R1.13
Tolerância a erros de execução	R1.1, R1.3, R1.4, R1.5, R1.7, R1.8, R1.9, R1.12, R1.13

Tabela 3.5: Relações entre funções e atributos do sistema

3.3.4 Casos de Utilização

Os casos de utilização descrevem o processo de uma sequência de eventos necessárias para completar uma ação. Existem duas formas de identificar casos de utilização, focado nos actores ou focado nos eventos, no nosso projeto vamos usar o focado nos actores para isso precisamos de:

- Identificar os actores relacionados com o sistema
- Por actor, identificar os processos que ele inicia ou participa

De seguida estão os actores que estão presentes no nosso sistema também como as suas ações:

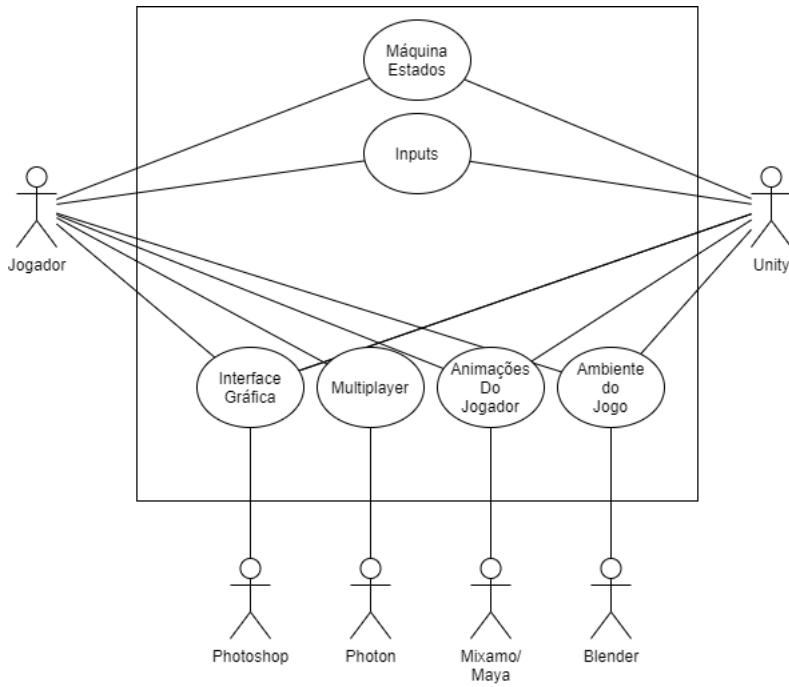


Figura 3.1: Casos de utilização

Casos de utilização - Resumido

De seguida seguem-se tabelas de casos de utilização em particular que contém um nome ilustrativo do caso, resumo da ação do caso e as referências às funções dos sistemas.

Cabeçalho	
Nome:	Interface Gráfica
Resumo:	Interação do jogador com os menus e interface gráfica
Referências:	R1.7, R1.8, R1.9

Tabela 3.6: Caso de utilização - Interface Gráfica

Cabeçalho	
Nome:	<i>Multiplayer</i>
Resumo:	Conectar a um servidor e jogar em <i>multiplayer</i>
Referências:	R1.1, R1.2, R1.3, R1.4

Tabela 3.7: Caso de utilização - *Multiplayer*

Cabeçalho	
Nome:	Animações
Resumo:	Animações dos jogadores e dos NPCs
Referências:	R1.6

Tabela 3.8: Caso de utilização - Animações

Cabeçalho	
Nome:	Ambiente do jogo
Resumo:	Interação entre os assets que compõe o mapa com o jogador e os NPCs
Referências:	R1.5, R1.6, R1.10, R1.12, R1.13

Tabela 3.9: Caso de utilização - Ambiente

Cabeçalho	
Nome:	Máquina de Estados
Resumo:	Inteligência Artificial dos NPCs para poderem interagir com o ambiente e os jogadores
Referências:	R1.5, R1.6, R1.10, R1.11

Tabela 3.10: Caso de utilização - Máquina de Estados

Cabeçalho	
Nome:	Inputs
Resumo:	Registar todos os eventos do rato e teclado que o jogador faz
Referências:	R1.2, R1.5, R1.6, R1.7, R1.8, R1.9, R1.10

Tabela 3.11: Caso de utilização - Inputs

Casos de utilização - Expandido

Estes casos de utilização englobam uma representação completa do fluxo de um determinado cenário.

Cenário Principal (fluxo típico de eventos)		
Ação do Ator		Resposta do Sistema
1	O Caso de utilização inicia quando o jogador começa o jogo	
2	O jogador põe ao servidor a lista de lobbies	
		3 Servidor devolve a lista de <i>lobbies</i> disponíveis
4	Jogador escolhe o <i>lobby</i>	
		5 Servidor cria a sessão do jogador
6	Jogador pode jogar no <i>lobby</i>	

Cenário Alternativo 1		
Número de Sequência	Alternativa	
2	O jogador pede para criar um novo lobby	Cria um novo lobby. 3 sem efeito

Tabela 3.12: Cenário do jogador iniciar um jogo

Cenário Principal (fluxo típico de eventos)		
	Ação do Ator	Resposta do Sistema
1	O Caso de utilização inicia quando está mais que 1 jogador no lobby	
2	O jogador faz uma ação	
3	Ação enviada para o servidor	
		4 Servidor realiza a ação (ação fica sincronizada para todos os jogadores do lobby)

Tabela 3.13: Cenário de sincronização *multiplayer*

3.4 Fundamentos

Nesta secção serão abordados os passos que tornaram este projeto possível. Inicialmente numa abordagem inicial foi analisar/investigar o jogo que deu a ideia para fazermos o nosso projeto, 2.1, deste conseguimos retirar bastantes ideias chave que foram fundamentais para o desenvolvimento do projeto, todas estas foram sendo apontadas à medida que surgiam mais, posteriormente teve de se escolher e alterar as melhores para enquadrarem no ambiente do projeto. Para uma melhor gestão do tempo e uma melhor organização do trabalho foi criado uma *timeline* com todas as ideias, que agora já são objetivos a realizar, e com as suas respectivas datas previstas de conclusão.

Após a realização da *timeline*, procedeu-se à pesquisa de assets, após umas procura chegou-se à conclusão que teria de se mudar o *timeline* devido à falta de modelos que adequassem no género do nosso projeto.

Antes de começar a implementação do projecto no *Unity*, pesquisaram-se paisagens em *low-poly*, por exemplo ilhas desertas, florestas, savanas, desertos, para servirem de base para fazer o terreno do projeto.



Figura 3.2: Exemplo de floresta



Figura 3.3: Exemplo de ilha de gelo



Figura 3.4: Exemplo de ilha



Figura 3.5: Exemplo de savana

3.5 Abordagem

Esta parte do projeto, pretende mostrar a nossa abordagem inicial ao desenvolvimento de um jogo *survival multiplayer*.

Inicialmente foi feita uma consulta à *timeline* abordada na secção 3.4, esta encontra-se apresentada na figura 3.6.

Semana	Topicos	Objetivos específicos
14/mar - 18/mar	Multiplayer (protótipo)	Text Chat, Juntar a servidor
	Terreno	Base (praia, pantano)
19/mar - 24/mar	UI	Menu start ingame, inventário(bagpack), atributos do jogador(vida, fome/sede, experiência, stamina) asdasdasdasdasdasdas
	Sistema de inventario	-
	Terreno	Adicionar algo que esteja em falta ou achamos essencial
25/mar - 2/abr	Personagem	Adicionar personagem lowpoly e animações com mixamo e blendtree
	Camera	Adicionar características da camera
	Terreno	Adicionar algo que esteja em falta ou achamos essencial
3/abr - 10/abr	Itens	Implementação de itens (Materiais e Utensílios), Criação de modelos em Blender caso necessário, apanhar e soltar itens
	Atributos de personagem	Utilizar o UI construído para implementar Stamina, Vida, Hunger, Experiencia da personagem
	Terreno	Adicionar algo que esteja em falta ou achamos essencial
11/abr - 18/abr	NPCs	Adicionar animais ao jogo
	Sistema de crafting	UI do sistema de crafting e o backend
19/abr - 25/abr	Modo Construção	Modo onde o player poderá instanciar modelos no mapa (caso tenha recursos para tal)
26/abr - 10/maio	Som	Adicionar Som ao jogador (steps), sons de fundo e animais
	Ciclo dia e noite	Adicionar ciclo dia e noite
11/maio - 20/maio	Tutorial	Criacao de um tutorial
21/maio - 10/junho	Cutscenes	Cutscene incial do jogo
	PostProcessing	Postprocessing da camera
11/junho - 21/junho	Bug fixes	Testar as funcionalidades todas do jogo
20/Junho - Até entrega	Relatório Final	

Figura 3.6: Timeline do projeto

Um dos aspetos principais abordados antes do começo do desenvolvimento do projeto em si, foi qual o estilo visual que seria a adotar para o jogo.

Como foi falado anteriormente, foi decidido fazer a parte 3D no estilo *lowpoly* (explicado na introdução). Esta característica permite que o jogo seja mais fácil de correr em máquinas menos capazes e, dá um caráter mais divertido ao mesmo.

De seguida, foi para a estrutura e história do jogo. Existem jogos em que o único objetivo é completar missões, de forma a chegar ao fim. No entanto, a

ideia era que o jogador não necessitasse de seguir missões como num manual, e, decidir por ele como é pode seguir em frente no jogo.

Para isso, foi escolhido um jogo numa espécie de níveis. Estes níveis seriam ilhas no mapa, correspondentes a biomas. O jogador começaria numa ilha, e, apenas consegue transitar para outras ilhas caso possua as ferramentas necessárias para ultrapassar os obstáculos que as separam.

Outro aspecto importante para implementar foram animais. É relevante mencioná-los pois se encontram diretamente relacionados com cada ilha. Como em cada uma o clima é completamente diferente das outras, os tipos de animais devem ser diferentes. Por exemplo, uma zebra, apenas existe num clima de savana, seco e quente.

Por fim, apresenta-se um pequeno diagrama, Figura 3.7, que mostra todas as ilhas que decidimos colocar no jogo, assim como todos os animais.

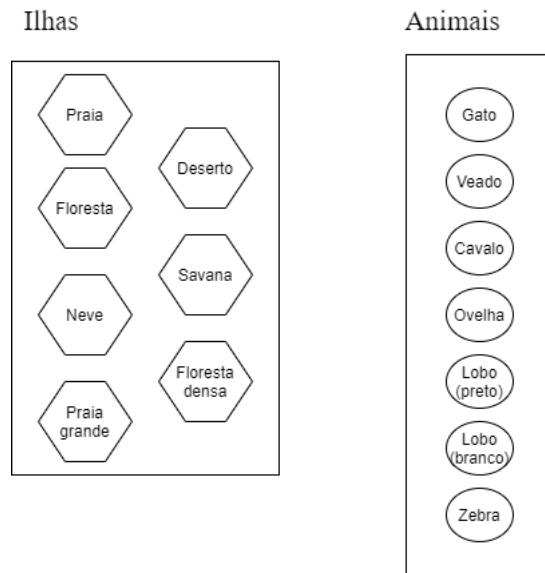


Figura 3.7: Ilhas e Animais

Capítulo 4

Implementação do Modelo

No presente segmento será demonstrada a implementação do modelo, descrito no capítulo 3. Desta forma, iremos descrever cada aspecto, mais em particular, permitindo ao leitor compreender a linha de pensamento, utilizada no desenvolvimento deste mesmo projeto. Descrever as tecnologias que permitiram transformar as nossas ideias no que veio a ser o projeto e, assim, num jogo completo onde qualquer jogador se possa divertir, da forma como foi inicialmente foi planeado que o jogo fosse experinciado.

4.1 Tecnologias

Num jogo existem vários aspetos a ter em conta, este projeto não foge à exceção. Primeiramente, existe a parte **visual**. Esta é a parte que mais impacto tem no utilizador. Isto porque é o primeiro estímulo obtido quando é iniciado um jogo/aplicação. Foram considerados os seguintes tópicos:

- **UI** - Relativo ao *design* das interfaces presentes, tais como o menu inicial, as interfaces relativas ao inventário *in-game* e visualização das características do jogador (vida, fome, etc.).
- **Design 3D** - Relativo ao *design* de tudo o que está presente no mapa, assim como o jogador em si e, objetos que possam ser utilizados (Pica-reta, Lança, etc.).
- **Efeitos especiais** - Relativo a filtros, mais conhecidos como *Post Processing*, aplicados na câmara que renderiza os gráficos do jogo, dando uma aparência mais primorosa ao jogo.

De seguida, a parte **lógica**, mesmo que não seja diretamente visível pelo utilizador é uma parte bastante importante, que apenas é possível sentir ao jogar um jogo. O motor do jogo, impacta a fluidez do jogo e daí é de grande importância e, deve ser adequado para o tipo de jogo que se pretende (neste caso um jogo *multiplayer co-op*). Algumas vertentes afetadas por esta parte seguem-se a seguir:

- **Controles do Jogador**
- **Inteligência Artificial de NPCs**
- **Tratamento de dados no envio e receção de pacotes pela rede**

Por fim, mas não menos importante, o **som** presente, também é um forte aspeto que influencia a percepção do jogador relativa ao jogo. Dá uma maior satisfação ao utilizador poder ouvir o que se está a passar na realidade presente no ambiente virtual. Existem alguns tipos de sons, particularmente os seguintes:

- **Som Ambiente** - Permite ao utilizador perceber onde se encontra, em termos de natureza.
- **Sons da personagem** - Tais como as pegadas e o cansaço.
- **Sons da interface do jogo** - Presentes mais em botões que exijam o *input* do utilizador.

4.1.1 Adobe Photoshop

O *Adobe Photoshop* é um programa caracterizado como um editor de imagens, desenvolvido pela *Adobe*. Considerado o líder no mercado dos editores de imagem profissionais. A razão pela escolha deste programa ao invés de outros tais como o GIMP, foi tanto pelo facto do nível de profissionalismo que o *Photoshop* nos trás, tanto como já havia conhecimento prévio deste *software*. Esta tecnologia encaixa-se no segmento visual do jogo, mais precisamente no UI do jogo. Todos os menus, ícones e, imagens foram, ou criados de raiz no *Photoshop*, ou sofreram alterações no mesmo. Na Figura 4.1 encontra-se um exemplo do uso do photoshop.

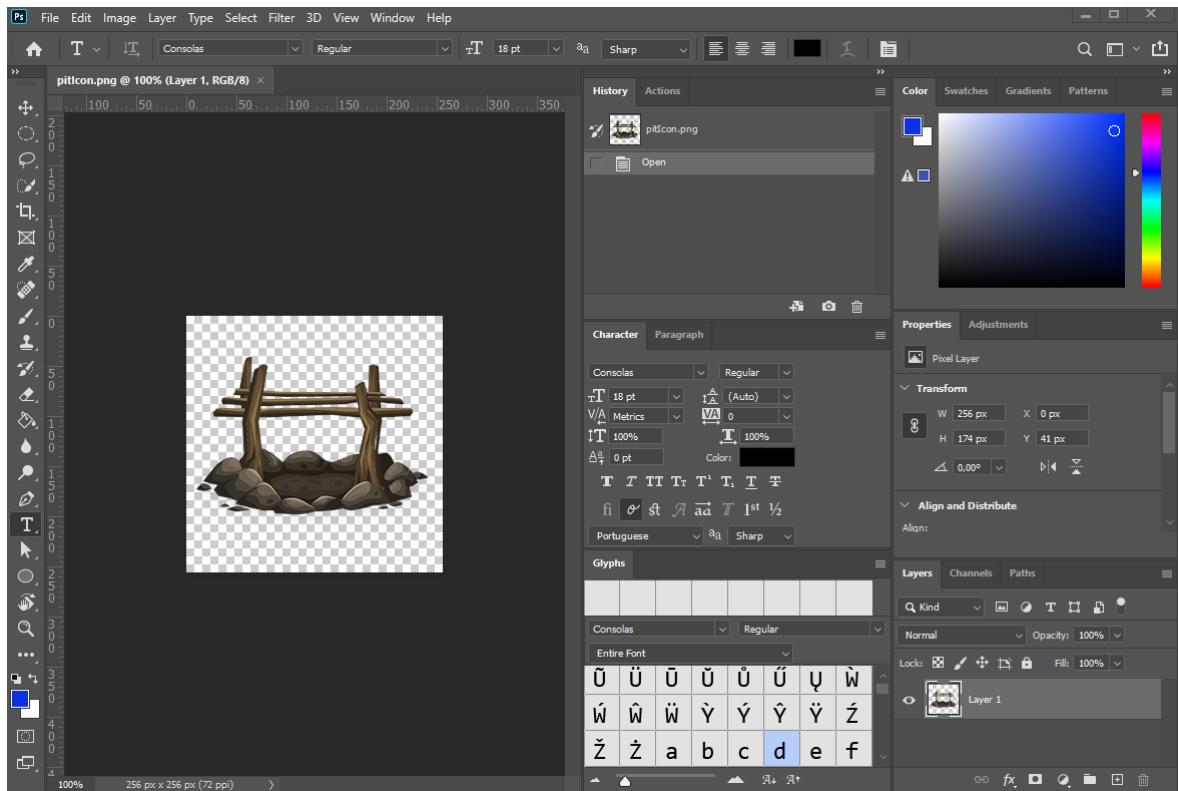


Figura 4.1: Interface - *Photoshop*

4.1.2 Blender, Maya e Mixamo

O *Blender* e o *Maya* são programas com o intuito de modelação, animação, texturização, composição e, renderização. Já a *Mixamo*, é outro *software* desenvolvido por parte da *Adobe* que permite, através de uma extensa biblioteca grátis de animações *motion capture* fornecidas pela *Adobe*, animar modelos 3d, mais especificamente com estrutura do esqueleto humano (*humanoids*).

Apesar do *Blender* e o *Maya* terem as mesmas capacidades, foram utilizados ambos os programas. Isto pois, devido à forma de como o *Blender* exporta animações para o formato ".fbx", não foi possível usar o mesmo com o *Unity*.

Assim, foi resolvido utilizar o *Blender* apenas para a modelação de modelos 3d que considerámos necessários. Já o *Maya*, foi utilizado para a criação de animações. É de notar que a maior parte das animações foram fornecidas pelo *Mixamo*. Mas, como eram necessárias algumas que não constavam na *Mixamo*, recorreu-se à criação das mesmas.

Nas Figuras 4.2, 4.3 e 4.4, é possível verificar as interfaces da cada uma das tecnologias mencionadas.

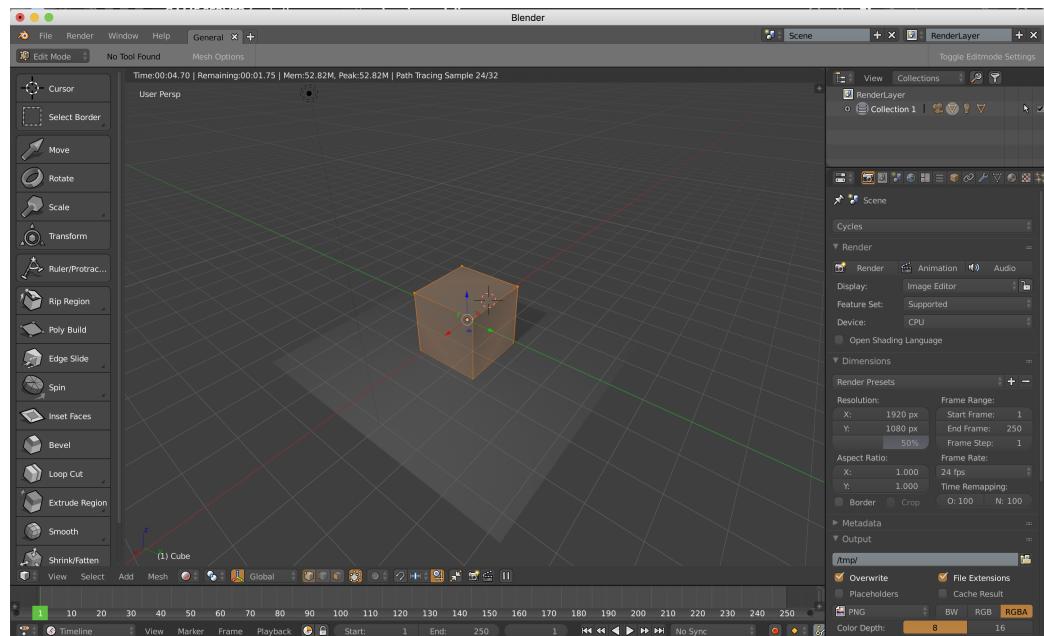


Figura 4.2: Interface - Blender

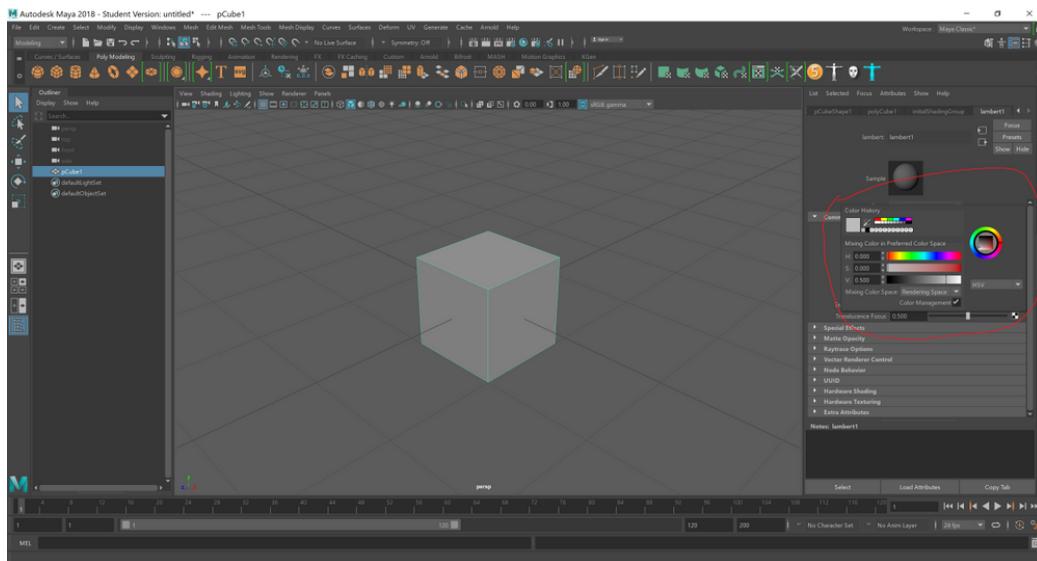


Figura 4.3: Interface - Maya

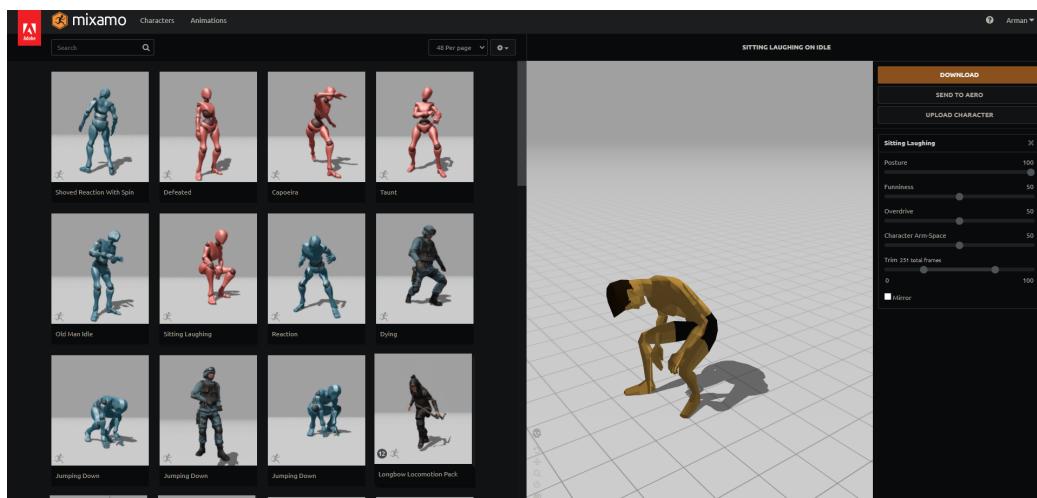


Figura 4.4: Interface - Mixamo

4.1.3 Unity

O *Unity* é um motor de jogo desenvolvido pela *Unity Technologies*. Este, oferece a capacidade de criar jogos tanto em 2D, como em 3d (como no nosso caso). É também possível a criação não só para sistemas Windows/Linux/-Mac, como também para tvOS, PS4, iOS, Xbox One, Android e, WebGL. A linguagem de programação disponibilizada pelo mesmo é o C#.

Este, é o motor do jogo. Foi uma ferramenta que já havia sido utilizada anteriormente na Unidade Curricular de Animação em Ambientes Virtuais (AAV) e, por isso já havia bastante conhecimento em como trabalhar neste ambiente. Uma grande mais valia, foi a *Asset Store*, fornecida pelo *Unity* que permite encontrar *assets* (pagos ou gratuitos) que facilitam diversas coisas. O *asset* que demos mais uso foi o *Photon Pun 2* que disponibiliza um servidor para poder jogar o jogo online, mas também foram utilizados mais alguns pequenos *assets*.

4.1.4 Audacity

Audacity é um software livre de edição digital de áudio disponível principalmente nas plataformas: Windows, Linux e Mac. O Audacity é muito popular entre os *podcasters* pelos seus recursos de edição, sua grande disponibilidade em múltiplas plataformas, suporte e *open-source* que permite ao programa ser gratuito.

O Audacity encaixa-se no segmento de **som** e, foi usado como editor de sons porque é um dos mais populares e é completamente gratuito, esta ferramenta permitiu nos fazer *trim*, ou seja, cortar os sons de forma a serem úteis para o projeto, também permitiu fazer uns efeitos nos sons tais como *fade-in* e *fade-outs* para os sons cortados parecerem mais naturais e por fim também foi utilizado a ferramenta de *noise reduction*, ou seja, redução de ruído para melhorar alguns sons que tinham ruído de fundo desnecessário.

A interface do Audacity, Figura 4.5 tem a seguinte aparência:

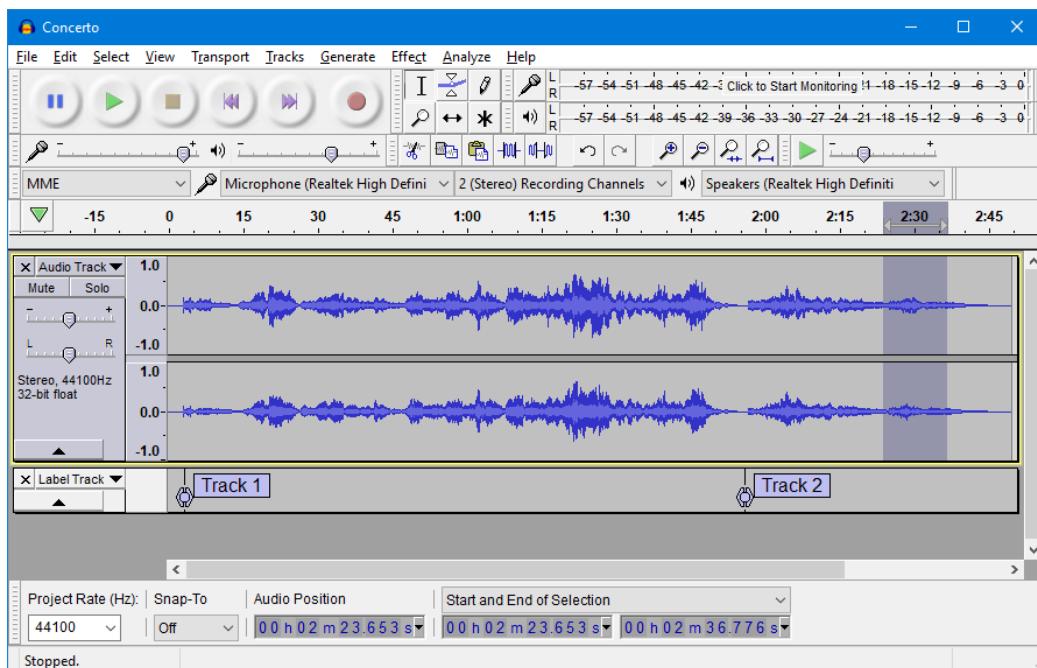


Figura 4.5: Interface - Audacity

4.2 Itens

Uma das características principais do nosso jogo é a sua variedade de itens diferentes, cada item tem funcionalidades diferentes e têm propósitos diferentes para cada situação. Todos os itens podem ser obtidos no jogo quer seja apanhados do chão, partir objetos no mapa ou matar animais, à medida que se vai conseguindo mais itens vai se podendo fazer itens cada vez melhores para ajudar a sobreviver ou progredir no jogo.

4.2.1 Sistema de itens

Para haver uma melhor ordenação dos itens estes foram divididos em várias categorias diferentes, como podemos observar de seguida:

- Materiais - estes itens não tem utilidade por si só, normalmente são usados em conjunto com outros materiais para criar ferramentas ou armas.
- Comida - itens que recuperam os *stats* do jogador, podem recuperar fome, sede e vida.
- Armadura - serve para proteger o jogador e diminuir o dano causado pelos animais agressivos
- Ferramentas - as ferramentas são utensílios para realizar alguma tarefa, no nosso jogo servem para partir árvores ou pedras.
- Armas - as armas servem para defender o jogador dos perigos da natureza

Como podemos observar cada categoria de itens têm uma utilidade diferente e vários atributos diferentes para cada cenário do jogo, ao dividir em categorias foi possível criar diferentes mecânicas e formas de uso para cada tipo de item.

Para implementar os itens e resolver este problema das categorias no nosso jogo foram usados *ScriptableObjects*, estes são contentores de dados que podem ser usados para guardar grandes quantidades de informações, são completamente independentes das instâncias das classes. Uma das principais

utilidades dos *ScriptableObjects* é reduzir a quantidade de memória utilizada pelo jogo porque evita a cópia de valores guardando valores estáticos, cada vez que é instanciado ele faz uma cópia de si mesmo em vez de duplicar os dados passa só a referência e faz isso para todas as instâncias. Quer dizer, por exemplo o jogador tiver 100 itens do mesmo tipo no seu inventário, que só existe sempre 1 *ScriptableObject* a consumir memória do nosso jogo. O *ScriptableObject* é igual a um asset do Unity, os valores têm de ser alterados no editor para que tenham efeito dentro o jogo, os parâmetros que pode tomar são definidos por um script que está ligado ao mesmo.

Para fazer os scripts dos itens foi usado o conceito de herança e classe abstracta onde existe uma classe genérica *ItemObject* que vai ter todos os parâmetros necessários para definir um item e depois as classes que herdam desta têm parâmetros mais específicos para cada uma das categorias, *FoodObject*, *ArmourObject*, *ToolObject*, *PlaceableObject*, *WeaponObject*, como mostra a Figura 4.6.

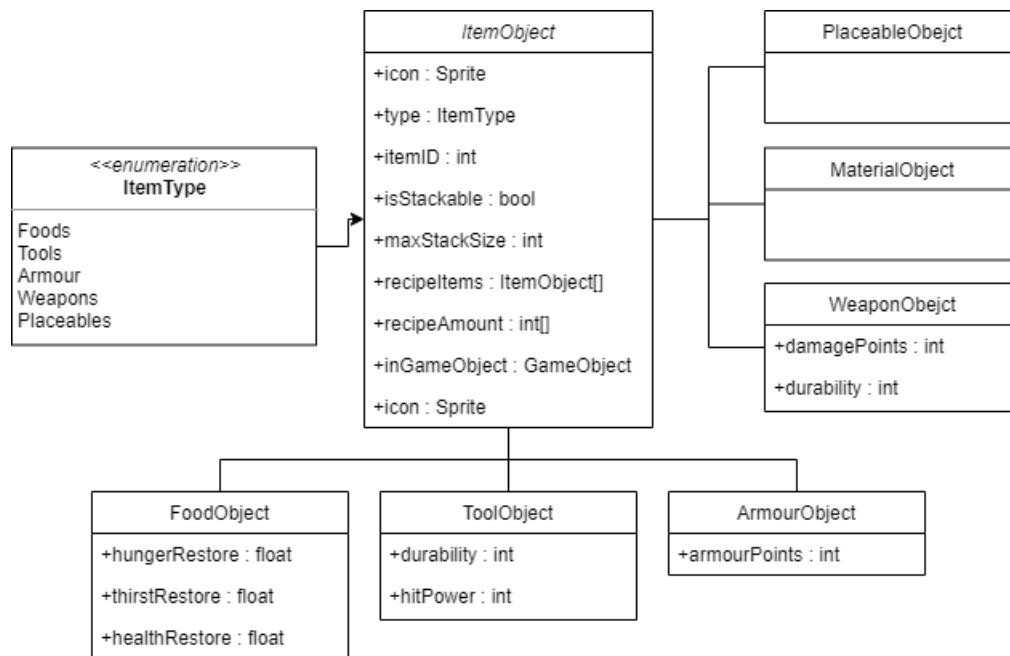


Figura 4.6: Diagrama de classes dos itens

Aqui estão expostos todos os parâmetros genéricos que um item pode tomar:

- Icon - ícone 2d que vai aparecer no inventário
- Type - categoria do item
- ItemID - identificador único de cada item
- IsStackable - se pode haver mais que um item no mesmo slot
- MaxStackSize - se o anterior for verdade qual é o valor máximo
- RecipeItems - itens para fazer receita
- ReceipeAmount - número de itens para fazer receita
- InGameObject - é o prefab que vai aparecer na mão do jogador
- Description - descrição do item

Como cada categoria de item pode ter parâmetros diferentes de seguida estão demonstrados mais especificamente o que cada categoria têm a mais que os itens genéricos, a figura 4.7 é um exemplo de uma arma e os seus parâmetros dentro de um *ScriptableObject* no editor do Unity.

- ToolObject
 - Durability - durabilidade do item
 - HitPower - dano que causa num recurso
- WeaponObject
 - Durability - durabilidade do item
 - DamagePoints - dano que a arma causa no inimigo
- ArmourObject
 - ArmourPoints - quando mais menor dano levamos do inimigo
- FoodObject
 - Não têm nenhum parâmetro especial
- MaterialObject

- Não têm nenhum parâmetro especial

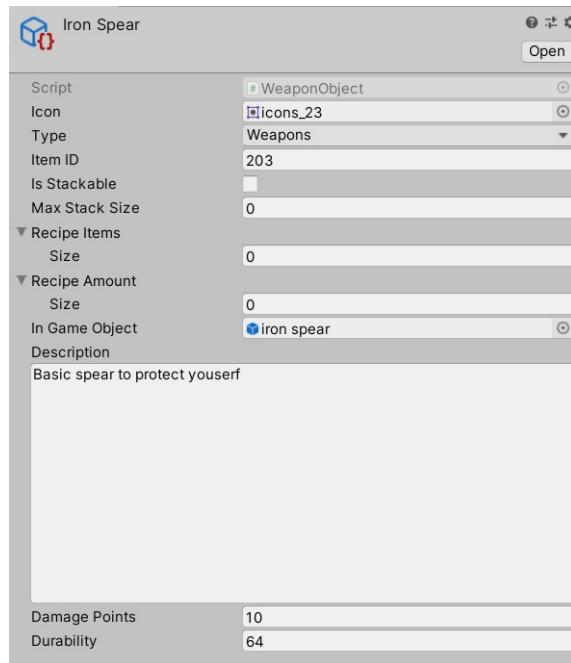


Figura 4.7: Exemplo de parâmetros de um item

4.2.2 Sistema de Inventário

Agora que já foram apresentados os itens todos do jogo e as suas funcionalidades, vai ser explicado aqui um forma de poder armazenar os mesmos. Cada jogador vai ter consigo um inventário que é uma matriz 4x5 que ao todo são 20 *slots* onde se pode guardar itens, o jogador pode colectar itens até que este fique completamente cheio. O inventário tem uma funcionalidade de poder fazer *drag and drop* dos itens para onde achar melhor, trás a vantagem de poder organizar o seu inventário e organizar ao seu gosto. De seguida, na Figura 4.8 está um exemplo do inventário do jogador.



Figura 4.8: Inventário do jogador

Hotbar é igualmente um inventário onde o jogador pode guardar itens mas este inventário é de acesso rápido. Existem 10 *slots*, cada um pode ser acedido usando os números que estão no teclado, por exemplo primeiro *slot* pode ser acedido pelo número 1 do teclado e assim em diante até ao último que é acedido usando o número 0, outra forma de interagir com este menu é usar o *scrollwheel* do rato. Este inventário é serve para poder usar as armas/ferramentas dependendo o *slot* seleccionado e está um exemplo deste na Figura 4.9.



Figura 4.9: Hotbar do jogador

4.2.3 Sistema de *Crafting*

Sistema de *crafting* foi introduzido no jogo para criar uma espécie de progressão no jogo, ao colectar mais itens o jogador pode fazer outros tipos de ferramentas melhores. Como foi referido anteriormente todos os itens que não da categoria dos materiais têm receitas para fazer os mesmos, estas receitas dizem que materiais são necessários e a respectiva quantidade para conseguir fazer um determinado item. Quando o jogador quer *craftar* algum item só precisa de abrir o menu de *crafting* e se este tiver os requisitos necessários ao lado do item que quer *craftar* vai aparecer um botão para poder criar o item.

Figura 4.10: *Hotbar* do jogador

4.3 Criação do Mapa

Para começar a fazer *design* e modelar o mapa foi necessário primeiro procurar ideias de níveis, progressão, história, continuidade e fluidez. Primeiramente foi inspirada no nível usado no jogo *SIMS* que foi referido no Capítulo 2.1, uma ilha desabitada onde o jogador depois de andar a deriva por muito tempo no oceano foi parar. Inicialmente para criar a ilha foi usado as ferramentas de terreno que o *Unity* oferece mas o resultado a que chegamos não agradou ao grupo, pode se ver o resultado na seguinte Figura 4.11.

Depois de alguma procura de inspiração encontramos um modelo que nos serviu como base de todo o terreno do nosso mapa, um hexágono, representado na Figura 4.12, começamos por fazer um pequeno exemplo do mapa com este modelo e saiu de agrado do grupo, como iriam ser feitos vários biomas diferentes o mapa foi logo dividido em "ilhas", cada parte do mapa é considerada uma ilha diferente e que contém animais diferentes, climas diferentes, texturas e cores diferentes.

Após a escolha da base das ilhas começou-se a pensar nos vários biomas que poderiam integrar o nosso jogo, inicialmente juntou-se vários hexágonos para criar o formato da ilha depois adicionou-se texturas e detalhes de uma planície como árvores, plantas, pedras, rios para tornar tudo um pouco mais

vivo, nas seguintes imagens têm a evolução da construção das ilhas, 4.13, 4.15, 4.14. As restantes ilhas foram feitas com as mesmas etapas só diferindo na parte do tipo de bioma, texturas e detalhes.

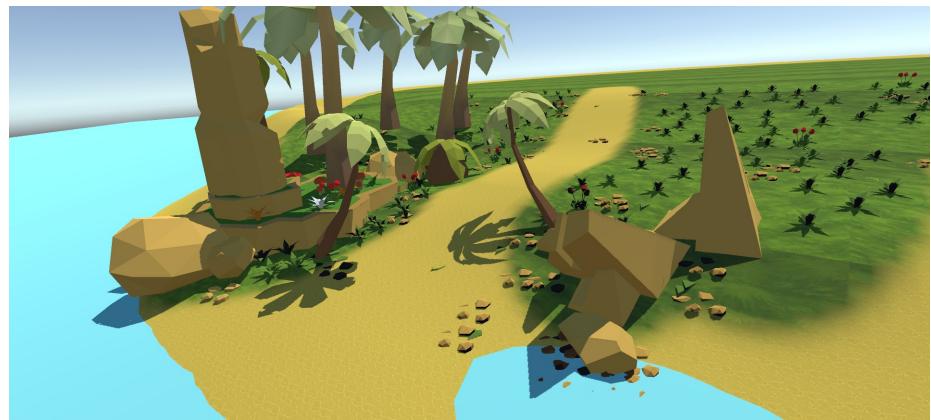


Figura 4.11: Design inicial da ilha

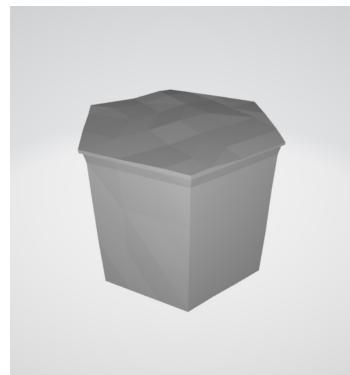


Figura 4.12: Hexágono base da criação da ilha

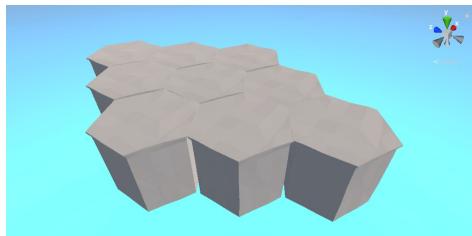


Figura 4.13: Ilha base

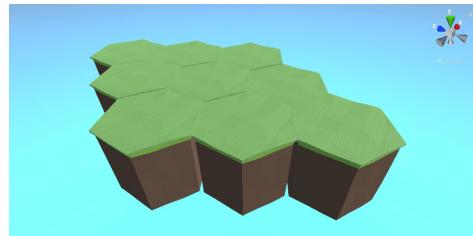


Figura 4.14: Ilha com texturas

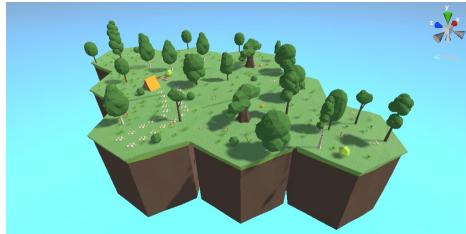


Figura 4.15: Ilha com detalhes

Em cada ilha, existem pequenos detalhes tais como árvores, pedras etc. Uma forma que foi encontrada de, não só ser mais rápido no posicionamento destes objetos, mas também de fazê-los serem aleatórios, foi utilizado um *asset* da *asset store* do *Unity*.

O *asset* utilizado foi o *Prefab Painter* e, este, deu a oportunidade de escolher um diâmetro e, com o cursor espalhar vários tipos de objetos no mapa.

4.4 Câmara

Uma câmara, em vídeo jogos é a componente pela qual o cliente consegue renderizar o mundo.

Existem alguns tipos de câmaras. De entre todos os mais conhecidos são:

- **Primeira Pessoa**
- **Terceira Pessoa**
- **MOBA**

Uma câmara em primeira pessoa é bastante utilizada maioritariamente em jogos de *shooter*. Nesta, o utilizador não consegue ver o modelo do personagem 3d totalmente. Um dos jogos conhecidos que utiliza este tipo de câmara é o *Counter Strike Global Offensive*, apresentado na Figura ??:



Figura 4.16: Exemplo de câmara em primeira pessoa

Já a câmara em terceira pessoa pretende mostrar o personagem 3d por completo e, foi o tipo de câmara escolhido para o projeto. Geralmente, esta mostra o modelo 3d do personagem visto de trás e, é muito provavelmente o tipo de câmara mais utilizado nos jogos do dia-a-dia, juntamente com a de primeira pessoa. Um jogo conhecido que usa este tipo de câmara é o *Fortnite* e apresenta a seguinte Figura 4.17:



Figura 4.17: Exemplo de câmera em terceira pessoa

Por fim, existe a câmara do tipo MOBA (*Multiplayer Online Battle Arena*). Este é um tipo de câmara *top down* (vista de cima para baixo), mas com um certo ângulo.

Encontra-se presente em vários jogos conhecidos tais como o *League of Legends*. De seguida, é possível verificar um exemplo do uso desta câmara no jogo mencionado acima.

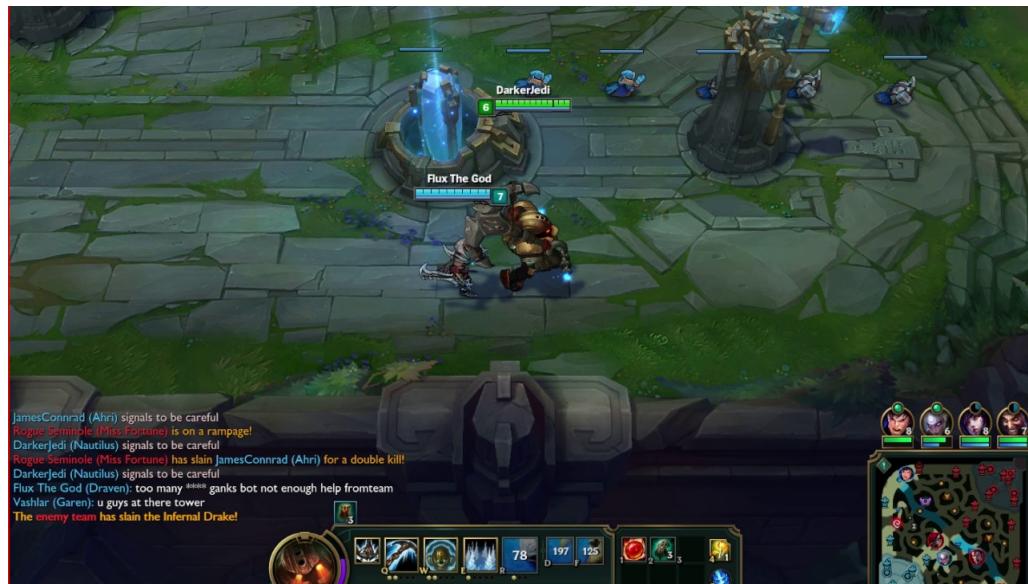


Figura 4.18: Exemplo de câmara MOBA

4.4.1 Espaço de cores

Antes de prosseguir para as características da câmara utilizada no jogo, é relevante mencionar o espaço de cores (ou *color space*).

Todas as texturas tendem a ser guardadas num espaço de cores do tipo *gamma*. Todavia, existe também o tipo *linear*.

A principal diferença entre estes dois espaços de cor, é a forma de como tratam a luz que incide nas texturas. Quando colocada em uso a renderização *linear*, os valores das equações de luz são diferentes do espaço *gamma*, resultando em imagens diferentes.

Em seguida, é possível verificar o impacto da intensidade da luz em objetos iguais, nestes dois espaços de cor, *linear* e *gamma*, respetivamente:

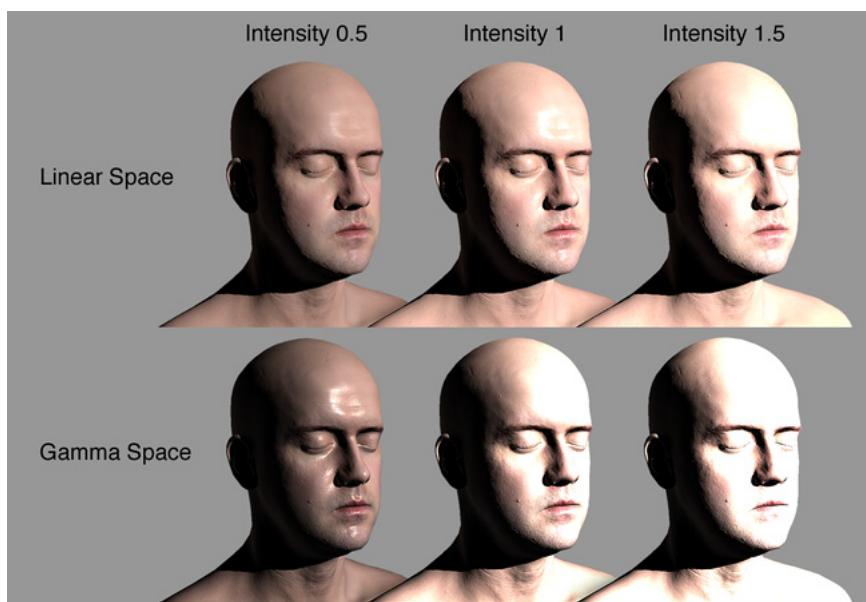


Figura 4.19: *Linear* vs *Gamma*

É claramente visível que o espaço *linear* tem um aspeto mais realista, não tendendo para a cor branca, mesmo em ambientes com uma luz de intensidade alta, daí ter sido escolhido este espaço.

4.4.2 Aspetos técnicos

Foi usado um *asset* da *Asset Store* do *Unity*, *Camera Controller*.

Este *asset* disponibiliza-nos várias características que, tornam a câmara muito mais realista, em termos profissionais. De seguida, serão apresentados os aspetos que levaram a escolher esta câmara.

É de notar que, decidimos não construir uma câmara nossa pois não precisamos de nenhuma *feature* específica ao nosso projeto. E, como a câmara não é o tópico principal do jogo, não valia a pena perder tempo com a construção de uma câmara que, já se encontra construída.

Motores

O motor da câmara é a parte que trata de processar todos os movimentos e rotações da mesma baseado no *input* do jogador ou do ambiente.

Este controlador de câmara (do *asset*: *Camera Controller*) encontra-se equipado com os seguintes motores:

- Primeira Pessoa
- Terceira pessoa
- MOBA
- *Fixed*
- *Spline*

Como é possível verificar acima, estes são os tipos de câmara anteriormente falados excepto os motores *Fixed* e *Spline*.

O *Fixed*, como o nome explica, apenas apresenta uma posição e rotação à câmara. A câmara não se altera.

Já o *Spline*, dá nos a facilidade de escolher um caminho (conjunto de pontos) por onde a câmara deve passar, fazendo uma pequena animação.

Âncora

A âncora é o que a câmara tem de ver sempre que neste caso é o jogador. Este *asset* permite que, facilmente seja escolhida a âncora para a câmara como o jogador. Desta forma, a câmara encontra-se sempre em órbita do mesmo.

É possível também adicionar um vetor *offset*, caso seja necessário subir, descer, ou movimentar a câmara em qualquer sentido em relação à sua âncora.

Colisões

Este controlador também tem conta as colisões da câmara. Caso a câmara tenha uma colisão contra algum objeto esta deve adaptar os movimentos, assim como, se algo se opuser entre a câmara e o jogador, a mesma deverá fazer o possível para que o modelo do jogador seja visível.

Assim, é usada a técnica *line-of-sight collision* que, baseia a sua colisão caso consiga ou não ver o modelo do jogador.

Seguidamente, é apresentado na Figura 4.20 alguns exemplos de colisão da câmara que, esta mesma tenta evitar.

Neste exemplo, a câmara não consegue ver o modelo do jogador, pois o caminho se encontra obstruído, assim a câmara limita-se a ir para a frente até que consiga avistar o modelo do jogador.

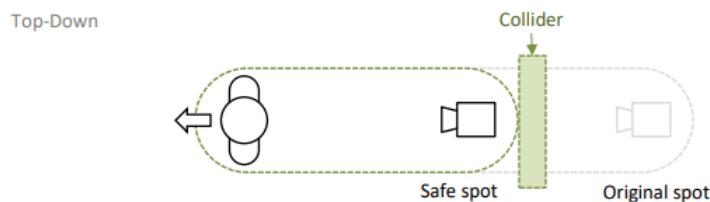


Figura 4.20: Exemplo de colisão de câmara - 1

Já aqui na Figura 4.21, o *collider* pode representar uma parede. Caso o utilizador movimente a câmara contra o *collider*, a mesma apenas seguirá o único caminho para o modelo do jogador, aproximando-se do mesmo.

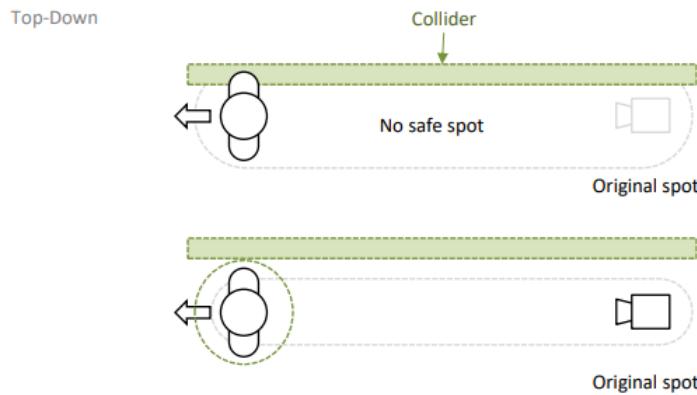


Figura 4.21: Exemplo de colisão de câmera - 2

4.4.3 Pós-Processamento

Pós-Processamento, mais referido por *Post Processing* é o processo de aplicar filtros a uma imagem. Neste processo, a imagem do jogo é renderizada e, antes de ser enviada para o monitor, ocorre um processamento sobre a imagem renderizada, este é o pós-processamento.

Esta técnica permite-nos ajustar o que o utilizador final vê. Tanto podemos fazer o jogo parecer mais realista, como mais estilo *cartoon*. Em boa verdade, dá-nos a capacidade de editar todos os *frames* a serem enviados para o monitor e, dar uma aparência completamente diferente da prévia.

Como pós-processamento, foi usado alguns *assets* provindos da *Asset Store*. Estes foram:

- *Amplify Color*
- *Amplify Occlusion*
- *Amplify Bloom*

Através deste conjunto de *assets*, foi adicionada à câmera do jogador estes componentes.

Amplify Occlusion

Esta componente tem o intuito de adicionar oclusão de ambiente ao meio. Oclusão Ambiente é um sombreamento e técnica de renderização utilizada para calcular quão exposto cada ponto numa imagem está. Assim é possível saber que pontos são os que estão menos expostos à luz ambiente, desvane- cendo conforme a intensidade do filtro.

Existem vários parâmetros, Figura 4.22 que podem ser ajustados para que qualquer pessoa tenha a liberdade de mudar o filtro como apetecer. Após adicionado à câmara, esta componente aumentou em grande intensidade o nível de realismo da cena, e ficou com o seguinte aspeto, Figura 4.23:



Figura 4.22: Pós-processamento Detalhes - (*Occlusion*)

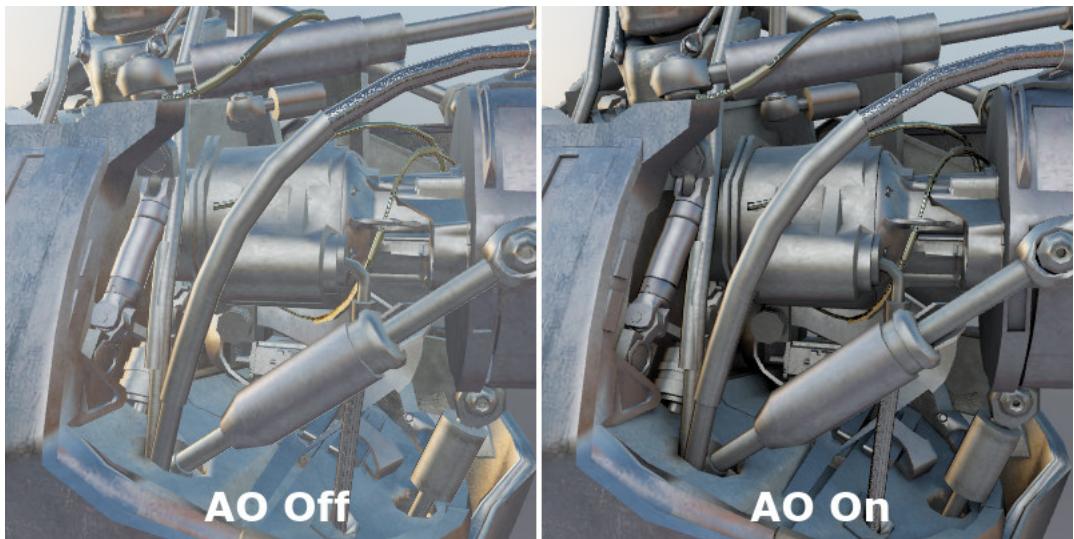


Figura 4.23: *Occlusion* OFF vs ON

Amplify Color

De seguida para o *Amplify Color*, este pós-processamento deve-se a ajustar as cores da renderização. O processo chama-se de *Color Grading* e, é a técnica usada para aprimorar uma imagem. É possível ajustar o contraste, as cores, a saturação, os detalhes, os níveis de branco e/ou preto. A Figura 4.24 mostra todas as características que este *asset* permite alterar:

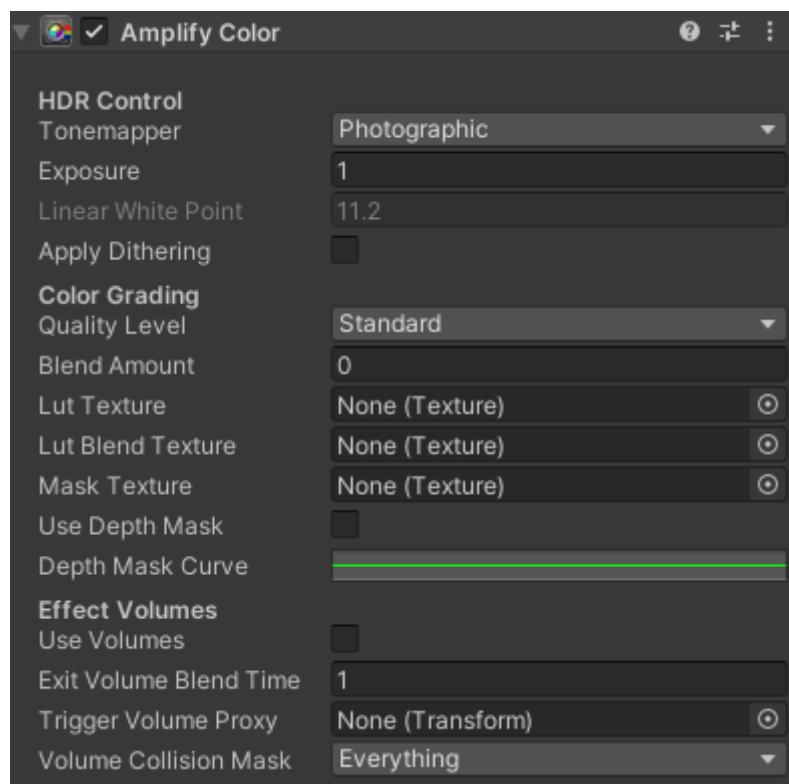


Figura 4.24: Pós-processamento Detalhes - (*Color Grading*)

Amplify Bloom

Por fim, este ultimo módulo adicionado ao pós-processamento tem como principal objetivo adicionar o efeito *Bloom*.

Este, é um efeito de computação gráfica utilizado muito em vídeo jogos para produzir uma simulação de um elemento nas imagens gravadas por câmaras da vida real. Mais em concreto, produz franjas de luz em volta de objetos de alto brilho numa imagem, contribuindo com a ilusão de uma luz extremamente clara, afetando a câmara que está a capturar a cena.

O *Amplify Bloom* têm as seguintes opções (Figura 4.25):

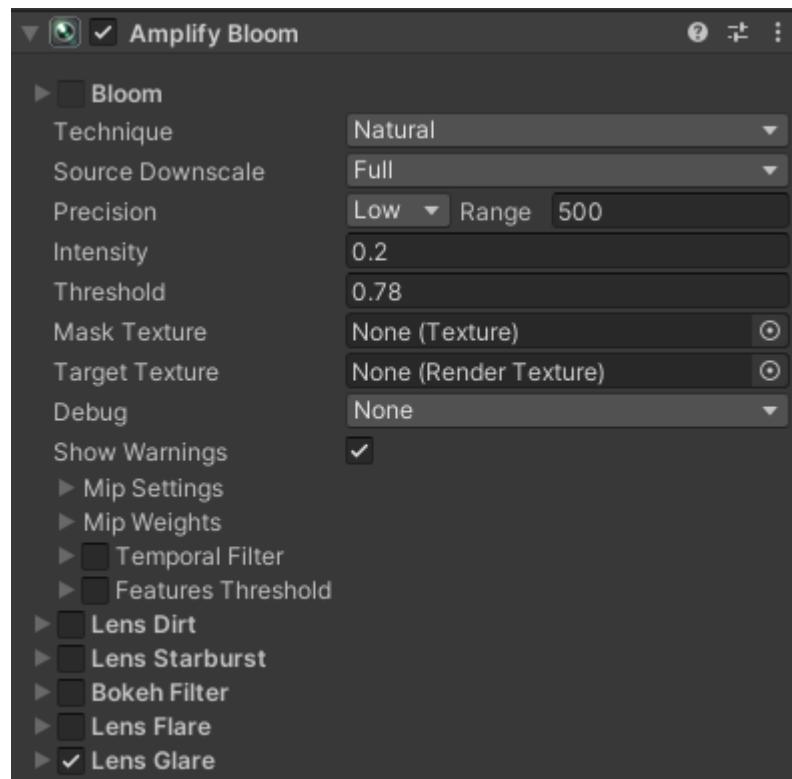


Figura 4.25: Pós-processamento Detalhes - (*Bloom Effect*)

De seguida, Figura 4.26, é possível ver como é que o efeito impacta a imagem final.



Figura 4.26: *Bloom Effect* OFF vs ON

Renderização Final

Através do uso destes três *assets* de pós-processamento, a renderização final ficou a seguinte, apresentados nas Figuras 4.27 e 4.28.



Figura 4.27: Demonstração sem pós-processamento



Figura 4.28: Demonstração do uso de pós-processamento

É possível notar os pequenos sombreamentos adicionados nos cantos dos objetos, realizado pelo *Amplify Occlusion*, assim como o tom de cor meio alaranjado e, a reflexão dos objetos tratados pelo *Amplify Color* e o *Amplify Bloom*.

4.5 Animações 3D

Como explicado anteriormente, no segmento 4.1.2, a maioria das animações foram retiradas da *Mixamo* e, outras foram feitas manualmente no *Maya*.

4.5.1 Movimentos

Uma tarefa comum em animações de jogos é a mistura entre duas ou mais animações. Um exemplo disto é por exemplo, a transição entre um personagem andar e correr adequadamente de acordo com a velocidade do personagem.

Para o funcionamento das mesmas no *Unity*, foi utilizado uma técnica fornecida pelo próprio, *Blend Trees*.

É importante destacar que Transições e *Blend Trees* são dois conceitos diferentes. Embora serem utilizados para criar transições suaves, são usados para diferentes situações.

- **Transições** são utilizadas para, ao transitar entre animações, haver uma certa suavidade.
- ***Blend Trees*** são utilizadas para permitir que múltiplas animações, ao transitarem, tenham suavidade, incorporando-as todas (as animações) a diversos ângulos.

Uma *Blend Tree* contém um mapa 2D com todas as animações presentes na mesma. Todas as animações que são movimentos para cada personagem (andar para a frente, correr, etc.) foram aglomeradas numa *Blend Tree*. Ficamos assim com a seguinte *Blend Tree*:

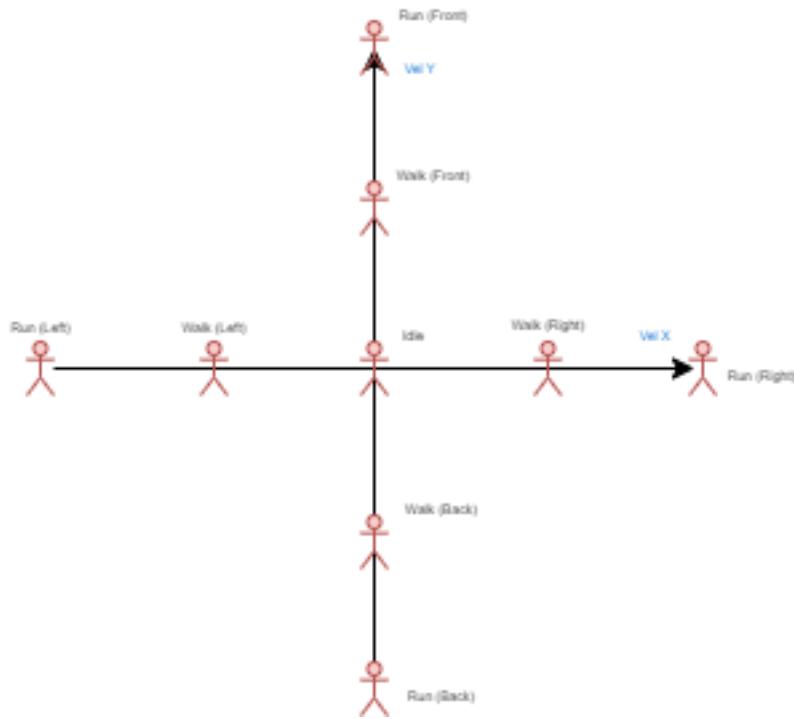


Figura 4.29: Gráfico cartesiano da *Blend Tree*

A cada ponto (ator) corresponde uma animação e, entre pontos, o *Unity* suaviza as transições. O que controla o gráfico é o vetor de velocidade do personagem. Esta velocidade, tanto em x, como em y varia entre -1 e 1 e, determina que animação é realizada.

4.5.2 Animações Simples

Já foi falado das animações de movimento que, de forma a serem suaves entre si, fizemos uso das *Blend Trees*. Porém, um último aspecto a adicionar nesta parcela do projeto, são as animações mais simples. Desde uma animação do personagem cair, saltar, ou mesmo pegar alguma coisa.

Todas elas utilizam *triggers*. Um *trigger* é nada mais nada menos que um interruptor. No entanto quando clicado no interruptor, este desliga logo de seguida, o que faz com que caso seja accionado um *trigger* uma animação apenas corre uma vez, não mais que isso.

Salto

Apesar de animação em si ser simples de efetuar, o salto é uma animação deveras um pouco mais complexa. Isto deve-se ao *collider** ter de subir com a animação. Para isso, foi operado um parâmetro do *Unity* chamado de *Curves* (ou curvas).

Uma curva é uma função matemática que define uma variável à nossa escolha no espaço de tempo da animação. Como se quer que o *Collider** suba ao longo do tempo, apenas criámos uma curva que sobe enquanto o personagem salta e, volta a descer quando este se aproxima do chão e, ficou com a seguinte aparência apresentada na Figura 4.30.

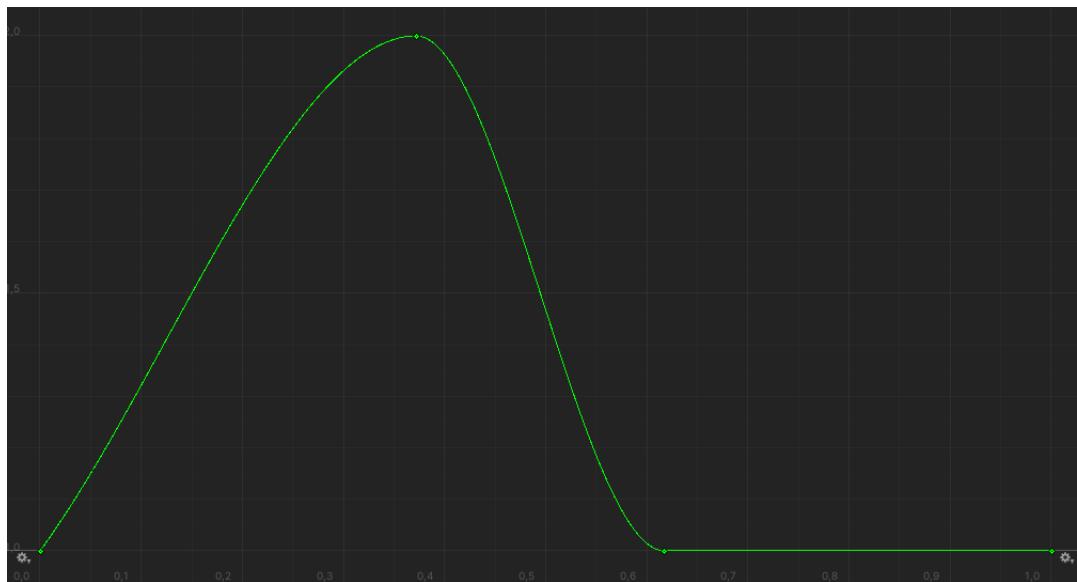


Figura 4.30: Gráfico cartesiano das curvas

Finalmente, a variável que a curva define é tratada em código, subindo ou descendo conforme o valor da curva em que se encontra.

*Um *Collider* determina as zonas de colisão de um determinado objeto que o contenha nos seus componentes.

4.6 Som

Este segmento procura explorar os tipos de som utilizados no projeto, assim como a sua implementação num jogo 3d.

O som é, por vezes subvalorizado nas áreas de vídeo jogos. Existem vários fatores que contribuem para um ambiente mais imersivo, no que diz respeito à experiência de utilizador. O som é um destes fatores mas, devido a não poder ser visto, muitas pessoas tomam este aspeto como dado.

A maior parte dos sons encontrados no projeto foram descarregados do freesound.org, e, foram todos melhorados utilizando o *Audacity*, em partes como o corte de sons, *fade-in/fade-out*, etc.

O *Unity* fornece logo ferramentas que tornam o processo de adição de som bastante simples. A componente que disponibiliza as ferramentas é o *Audio Source*.

Esta componente dá as seguintes opções de customização, Figura 4.31.

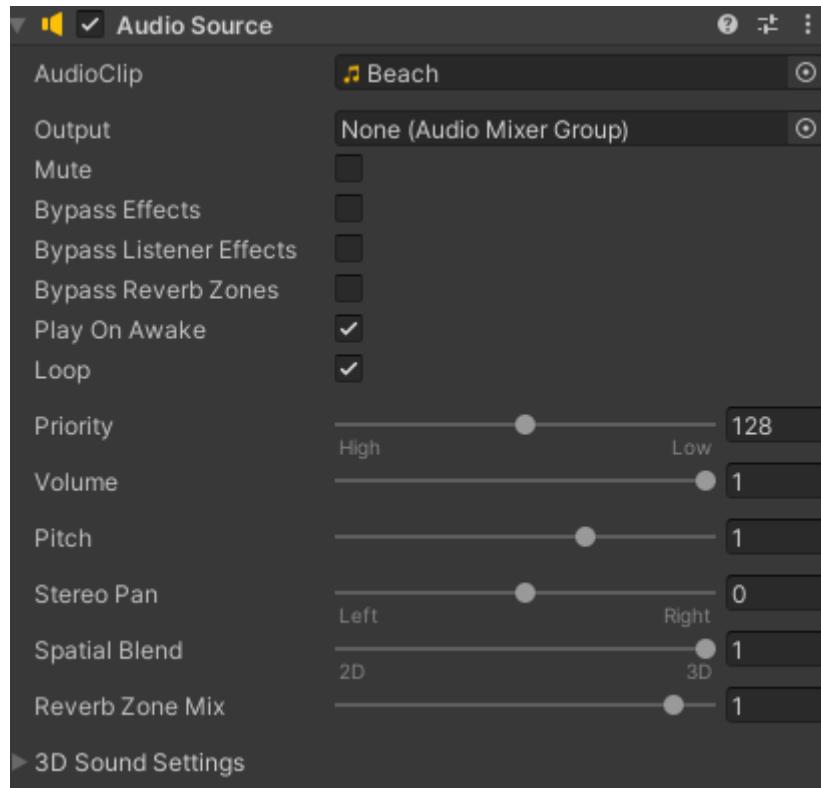


Figura 4.31: Component *Audio Source*

É de notar que, de forma ser possível ouvir som, a câmara do jogador deve conter a componente *Audio Listener*.

4.6.1 Som Ambiente

O som ambiente refere-se ao ruído de fundo, presente numa determinada cena ou localização. Isto pode incluir chuva, tráfego das estradas, grilos, pássaros, etc.

No *Unity* é possível definir o quanto abrange o som, em termos de espaço, no janela de cena. De forma a criar uma transição de ilha em ilha, foi utilizada a opção do *Audio Source* ”*3D Sound Settings*” mostrada na figura 4.31.

Mais em detalhe, esta opção permite definir o gráfico que relaciona o nível de som (entre 0 e 1), presente no eixo das abcissas, com a distância ao ouvinte, presente no eixo das ordenadas, Figura 4.31.

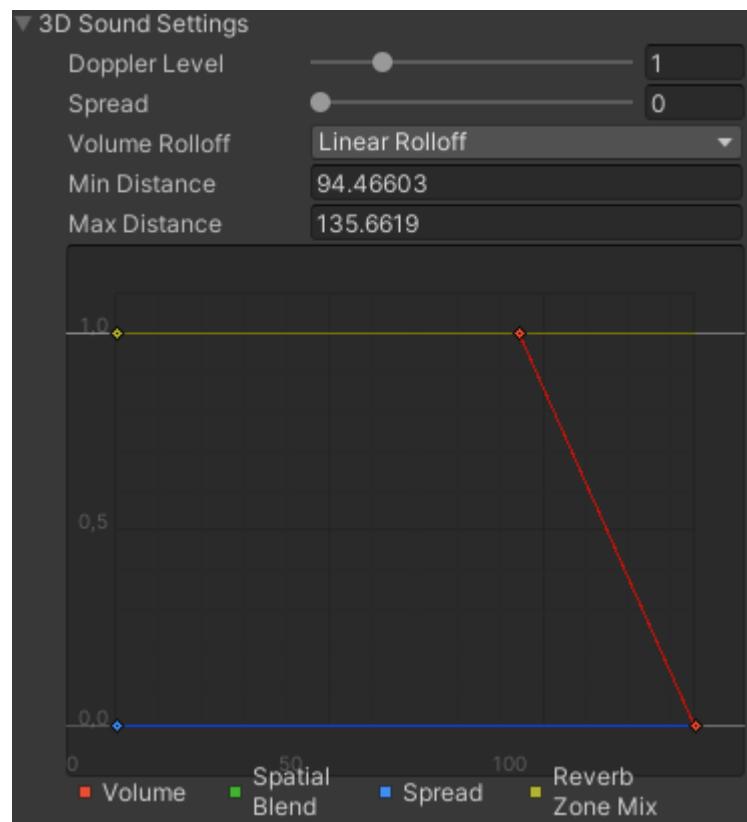


Figura 4.32: Gráfico de som 3D - *Audio Source*

À medida que o ouvinte se afasta desde a distância de 94.4 até à distância 135.7, o som vai gradualmente descendo até se deixar de ouvir completamente.

4.6.2 Som dos passos do personagem

Um tópico importante a respeito do som dos passos do personagem, é o facto de os passos poderem ser em diferentes tipos de solo. Um jogador pode estar a andar sobre neve, relva, etc.

Para desenvolver um jogo em que o som dos passos é coerente com o ambiente em que o jogador se situa, é preciso de detetar onde está o jogador.

Isto foi feito através do uso de *tags* e *Raycasts*. Uma *tag* é uma palavra de referência que pode ser atribuída a um ou a mais objetos numa cena. Por exemplo, usualmente os modelos de cada personagem pertencem à *tag* "Player".

Cada solo pertencente a cada bioma possui então a sua própria *tag*. Por exemplo, um pedaço de solo na neve tem a *tag* de "neve".

Já um *raycast*, como o nome indica, é um "lançamento de um raio". É bastante utilizado em vídeo-jogos. São utilizados *raycasts* para verificar onde é que uma reta num referencial (2D ou 3D) toca em algum objeto e, qual foi o objeto que tocou na reta.

Fazendo uso de ambos estes conceitos, utilizamos um *raycast* no modelo do jogador, sendo este uma reta a apontar para baixo (para os pés do modelo), e, é possível verificar qual a *tag* do objeto que o *raycast* devolve.

Conforme a *tag*, o som dos passos do personagem é trocado para o que é representante do solo.

É de notar que, para que o *raycast* não detete o jogador em si, foi necessário envolver regras que fazem com que o mesmo ignore o próprio modelo do jogador.

4.7 Arquitetura da rede

Conforme explicado anteriormente, foram utilizados alguns *assets* da *Asset Store* do *Unity*. No entanto, o que permitiu dar um caráter *Multiplayer* ao mesmo, o *Photon Pun 2*. Através deste *asset* foi possível conectar a um servidor do *Photon* e, conectar um cliente em qualquer lado (desde que tenha acesso à internet) ao *host*. Assim, o servidor do *Photon* age como um *relay server*. Todos os clientes conseguem chegar a todos os clientes existentes numa sala e, ao *host* por este servidor. Supondo que existe uma partida, onde estão presentes 3 jogadores, um deles o *host* e, o Cliente A pretende comunicar uma mensagem ao *host*, temos o seguinte diagrama, Diagrama 4.33:

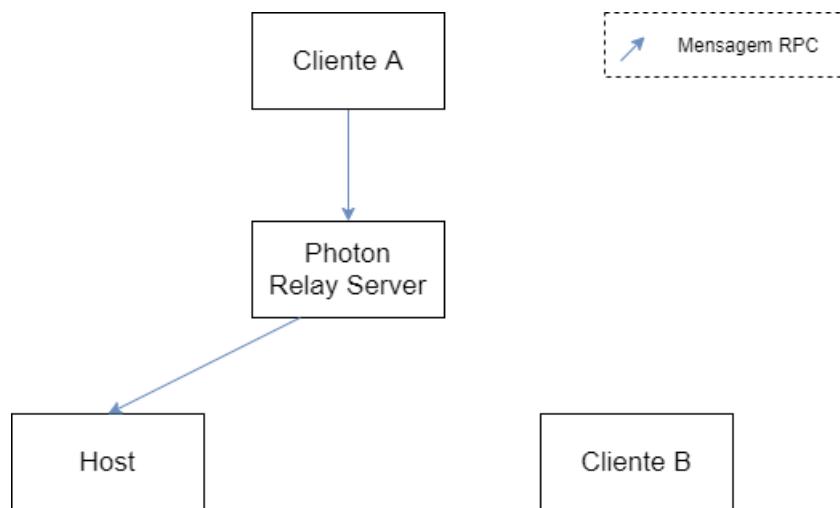


Figura 4.33: Arquitetura da rede através do servidor do *Photon*

As mensagens transmitidas através do servidor chamam-se de RPC (*Remote Procedure Calls*) e, permitem enviar mensagens a clientes específicos, a todos os clientes conectados, ou só ao *host*.

4.7.1 Sincronização de objetos pela rede

Num jogo *Multiplayer*, os objetos devem ser devidamente sincronizados com a rede. Isto é, todos os jogadores devem ver o mesmo a acontecer. Todavia, nem tudo precisa de ser sincronizado, existem algumas características que podem ser apenas locais à máquina a correr o jogo.

Por exemplo, a câmara de cada jogador não deve ser visível pela rede, ou seja, deve ser local, assim como os *scripts* que a controlam.

A sincronização pela rede, pelo *Photon* é feito da seguinte forma. Cada objeto a ser sincronizado pela rede deve conter um componente *PhotonView*.

Este componente serve para identificar um objeto pela rede (através de um id único). Dentro de deste componente, cabe ao programador decidir que atributos devem ser sincronizados. Podemos sincronizar a posição, a rotação e a escala do objeto e/ou as suas animações.

É de notar que, objetos que constam no mapa como estáticos, não devem ser sincronizados, pois não existem alterações aos mesmos.

4.7.2 Instanciamento de objetos pela rede

Instanciar objetos é um módulo que já está implementado na versão base do *Unity*. Contudo, caso um cliente (ou *host*) instancie um objeto apartir deste módulo, o objeto não será visível na rede, apenas localmente.

Através de mensagens RPC, mencionadas no segmento 4.7, é possível ”avisar” os jogadores presentes de modo a estes instanciarem o objeto localmente. Esta solução não é a melhor, devido à sua complexidade e, aumentar o tráfego de mensagens na rede.

Para isto, o *Photon* disponibiliza o seu próprio módulo de instanciamento de objetos na rede. Basta chamar o método *PhotonNetwork.Instantiate()* e, com alguns parâmetros, é possível instanciar objetos na rede em instantes.

Constate-se que para instanciar na rede um objeto, o mesmo deve ser sincronizado, devendo possuir a componente *PhotonView* (4.7.1).

4.7.3 Guardar e Carregar jogo

O principal objetivo ao guardar e carregar o estado de um jogo *multiplayer* é garantir que os dados sejam acessíveis ao *host*. Como o *Photon* não nos permite guardar informação no servidor, foi necessário utilizar o sistema de ficheiros do *host*, de modo a guardar e carregar informação.

Existem dois tipos de informação que é necessário guardar para preservar o estado do jogo, estas são os seguintes:

- **Dados dos jogadores** - Vida, Fome, Sede, etc.

- **Alterações no mapa** - Objetos instanciados no mapa por parte dos jogadores.

Foi utilizado o JSON como estrutura de dados.

JSON (*JavaScript Object Notation*), é um formato leve de troca de dados entre sistemas.

Foram operados dois ficheiros JSON, um para cada tipo de informação (Dados dos jogadores ou Alterações no mapa).

Apenas o *host* guarda a informação e, da seguinte forma: Cada vez que um jogador sai do jogo, o servidor do *Photon* deteta a saída e, informa o *host* que tal jogador saiu. Neste jogador, encontram-se os seus dados (dados de jogador e, alterações que este fez ao mapa). Assim, o *host* apenas guarda (ou atualiza, caso o jogador já conste no ficheiro) em ambos os ficheiros JSON as informações fornecidas pelo servidor.

É de notar que, caso o *host* pretenda guardar os seus dados, deverá o fazer manualmente através do menu.

Em seguida, encontra-se um pequeno diagrama que mostra o processo de guardar dados de um jogador.

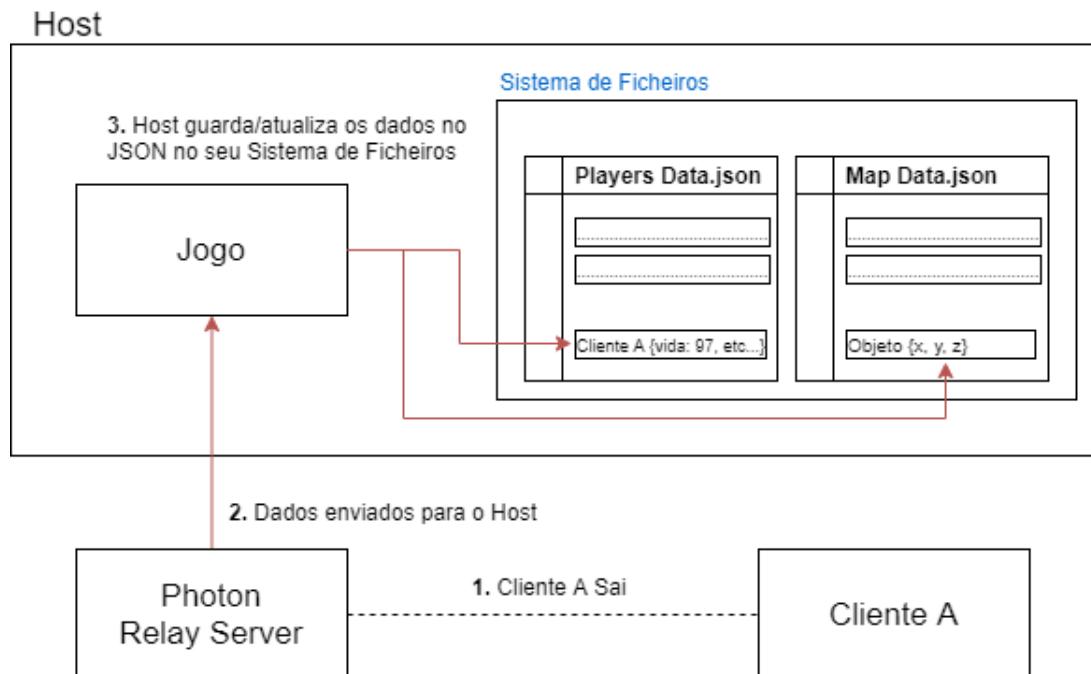


Figura 4.34: Processo de guardar o jogo, quando um cliente saí

Já quando o jogo começa, é carregado todo o ficheiro **Map Data** (Acima no diagrama), de forma a manter o estado do mapa. À medida que cada cliente entra, o servidor *Photon* "avisa" o *host* que, por sua vez verifica se o cliente consta no **Players Data** e, caso sim, envia um pacote pela rede com a sua informação. Este processo é bastante parecido ao de guardar o jogo, mas reversamente:

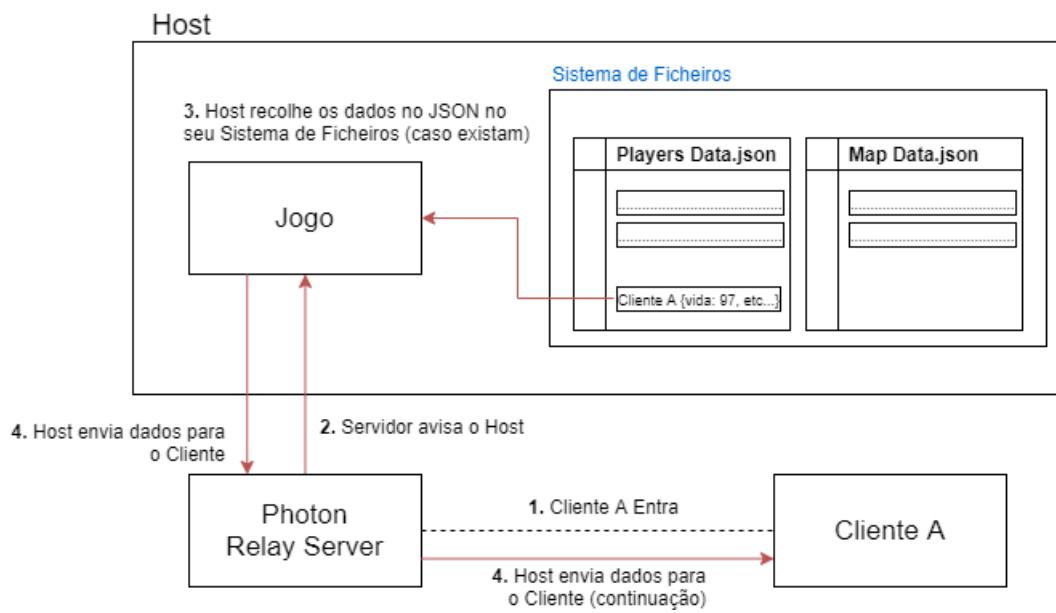


Figura 4.35: Processo de carregar a informação de um cliente, quando o mesmo entra

4.8 Sistema de mensagens

Um sistema de mensagens foi também adicionado. Assim, jogadores podem falar entre si e, juntos descobrir formas de prosseguir com a continuação do jogo.

Um jogo de cooperação precisa de um mecanismo de comunicação entre jogadores. Só assim é que os mesmos irão trabalhar em conjunto para que consigam chegar aos seus objetivos, sejam esses recolher recursos, construir infraestruturas, entre outros.

O sistema de mensagens funciona da seguinte forma, as mensagens não se encontram guardadas em nenhuma base de dados, isto pois não foi considerado que utilizadores necessitem ou mesmo queiram ler mensagens antigas. Foi criado um objeto que, se encontra sempre sincronizado com a rede. Este objeto tem o intuito de guardar as mensagens dos jogadores ao longo de uma sessão de jogo e, após o término da sessão é apagado, não criando nenhum histórico de mensagens.

Os utilizadores podem clicar na tecla "T" para enviar mensagens, aparecendo um pequeno UI (*User Interface*) que disponibiliza duas caixas de texto, uma para escrever e, outra para ler. Cada vez que um utilizador envia uma mensagem, é enviada uma mensagem RPC (mencionado no segmento 4.7) para todos os jogadores, guardando a mensagem no objeto de mensagens sincronizado pela rede.

Caso algum utilizador agora decida abrir o painel de mensagens, a mensagem enviada pela rede aparecerá lá.

4.9 Interface Gráfica (GUI)

Interface gráfica do utilizador (abreviadamente, o acrónimo GUI, do inglês Graphical User Interface) é um tipo de interface do utilizador que permite a interação com dispositivos digitais por meio de elementos gráficos como ícones e outros indicadores visuais, em contraste a interface de linha de comando. Foi criada pela Xerox mas somente se tornou um produto com a Apple.

Um aspecto muito importante de um jogo é a sua interface gráfica, a parte onde o utilizador vai interagir e vai observar quando tenta fazer alguma coisa quer num jogo. Uma interface apelativa dá logo pontos extra a qualquer tipo de aplicação ou jogo, incentiva naturalmente uma pessoa a utilizar mais porque se deparamos com um interface feia é muito provável que não haverá mais visitas depois da primeira.

4.9.1 *Canvas* do Unity

Implementação da interface gráfica foi concebida através do uso do *Gameobject Canvas* do Unity, este *Canvas* é uma espécie de tela de pintura, como o nome sugere, onde se pode colocar vários tipos de elementos gráficos, por exemplo botões, painéis, imagens, e fazer design intuitivamente e dinamicamente.

Canvas do Unity têm vários truques que tornam a criação de menus muito mais fácil, de seguida vamos falar sobre alguns mais importantes e mais úteis que foram usados na implementação do nosso trabalho.

Ordem de renderização dos elementos

Os elementos que se encontram num *Canvas* são desenhados na mesma ordem que aparecem na Hierarquia. O primeiro *child* será o primeiro a ser desenhado e por aí em diante, se um elemento sobrepor a outro o último vai ficar em cima do primeiro. Para mudar qual elemento fica em cima é só reordenar a Hierarquia no editor.

Por exemplo, na Figura 4.36 existem 3 camadas de elementos, a mais atrás que será o fundo, a do meio que são os botões onde se pode interagir e por fim mais acima está uma moldura para dar a sensação de uma janela.



Figura 4.36: Posicionamento dos elementos

Modos de Renderização

O *Canvas* tem várias opções que deixam este ser renderizado no ecrã da pessoa, na câmara do jogo ou no próprio jogo. De seguida está apresentado em detalhe cada uma.

- Screen Space - Overlay, este modo de renderização é o mais usado para a criação da interface gráfica porque adapta-se automaticamente ao tamanho do ecrã. Usado globalmente para fazer a interface gráfica, representado pela Figura 4.37.
- Screen Space - Camera, fazer renderização na câmara dá a liberdade de colocar o *Canvas* a qualquer distância de uma determinada câmara, dependendo das opções e efeitos colocadas na câmara pode se mudar a aparência dos elementos que aparecem no *Canvas*. Por exemplo ao mudar a perspectiva da câmara podemos ver um *Canvas* com ângulos diferentes, representado pela Figura 4.38.
- World Space, neste modo o *Canvas* comporta-se como qualquer outro objeto que esteja no jogo. O tamanho, posição, rotação deste pode ser manualmente configurado no jogo para contemplar com as diferentes necessidades. Útil para fazer barras de vida/informação em cima dos NPCs, representado pela Figura 4.39.



Figura 4.37: Screen Space - Overlay

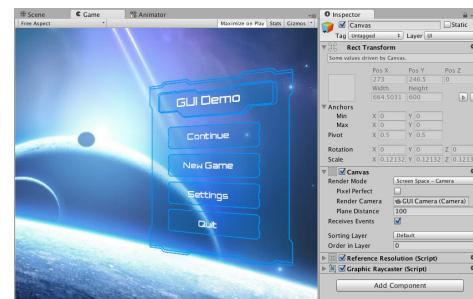


Figura 4.38: Screen Space - Camera

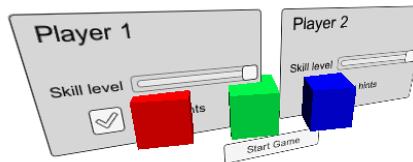


Figura 4.39: World Space

Elementos do *Canvas*

Com a integração da interface gráfica no Unity, vários novos elementos foram adicionados para ajudar a criar uma GUI com funcionalidades específicas. Cada funcionalidade têm objetivos diferentes que juntos conseguem facilitar muito a criação de GUI dinâmicos e interactivos.

- **Text** - elemento básico do UI que cria uma caixa de texto onde se pode escrever caracteres e também deixa formatar o tamanho da fonte, estilo da fonte, alinhamento do texto.
- **TextMeshPro** - é igual ao elemento Text mas contém mais formas de formatar o texto, tais como tamanho automático da fonte, trata dos *overflows* de texto, deixa usar fontes personalizadas entre muitos outros.
- **Image** - é um elemento que deixa por um *sprite*, ou seja, um *bitmap graphic* que pode ser uma imagem estática ou animada.
- **Slider** - elemento que pode tomar qualquer valor dentro de um intervalo definido nos parâmetros, normalmente usado para ajustar valores

volume mas pode ser usado em outras aplicações tais como barras de vida, barras de progresso e etc....

- **Button** - quando clicado faz *trigger*, dispara um evento que depois é apanhado por um *listener* que depois pode ser usado para fazer diferentes ações no jogo.
- **Dropdown menu** - é um elemento da interface gráfica que é semelhante a uma lista, que permite que o usuário escolha um valor de uma lista de opções.

Layouts

Nesta secção vamos observar as formas de posicionamento dos elementos no *Canvas*, estas ferramentas podem para mover, rodar, escalar, fixar, ancorar os elementos na tela, estes são fundamentais para poder gerir o posicionamento de cada elemento na posição correta do *Canvas*.

- Rect tool - todos os elementos da interface gráfica são representados como retângulos para serem mais facilmente manuseados nos layouts. Este retângulo pode ser manipulado usando a ferramenta de *Rect Tool*, este dá a possibilidade de alterar o tamanho diretamente no *Canvas* sem ter que recorrer aos valores do *Rect transform*, tornando o mais fácil e prático para ajustes rápidos.
- Anchor - esta opção deixa fazer layouts responsivos para se adaptarem para qualquer tipo de janela ou para manterem as proporções independentemente da posição do elemento "pai".

Auto Layouts

O sistema de layout do *Rect Transform* é flexível o suficiente para lidar com muitos tipos diferentes de layouts e também permite colocar elementos de uma forma totalmente livre. No entanto, às vezes, algo um pouco mais estruturado pode ser necessário.

O sistema de layout automático fornece maneiras de colocar elementos em grupos de layout em hierarquia, como grupos horizontais, grupos verticais ou matrizes. Também permite que os elementos sejam dimensionados automaticamente de acordo com o conteúdo nele contido. Por exemplo, um botão

pode ser redimensionado dinamicamente para ter exactamente o tamanho do conteúdo de texto.

O sistema de layout automático é um sistema construído em cima do sistema de layout básico Rect Transform. Pode ser usado opcionalmente em alguns ou todos os elementos. Para tornar implementações mais rápidas existem vários layout que já vem feitos no Unity, estes ajudam muito a cortar o tempo necessário para desenvolver uma interface gráfica.

- GridLayout - este layout cria uma matriz com espaçamento igual no eixo vertical e horizontal entre todos os elementos da mesma, esta pode ter qualquer tamanho, é de notar que o número de colunas pode ser diferente do número de linhas e vice versa dando a liberdade total a quem vai implementar mas com algumas opções é possível obrigar a matriz a ser quadrada se for preciso. Útil para criar sistemas de inventários ou tabuleiros. Nas figuras seguintes, 4.40 e 4.41, podemos observar todas as opções possíveis e uma implementação possível.

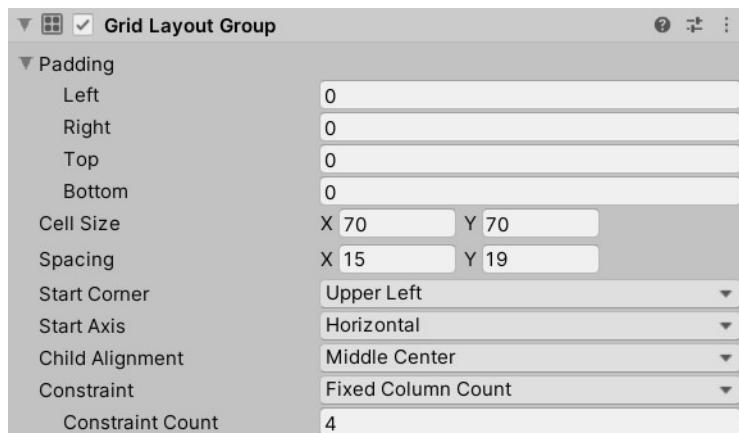


Figura 4.40: Opções disponíveis do GridLayout

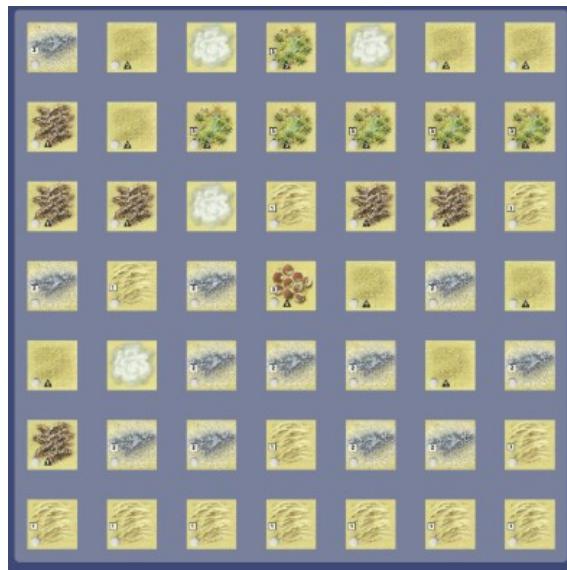


Figura 4.41: Exemplo de uma implementação do GridLayout

- Vertical Group Layout - este layout cria uma lista vertical de elementos onde todos os elementos estão separados igualmente no eixo vertical, este layout é uma versão simplificada do GridLayout uma vez que só controla os elementos no eixo vertical. Serve normalmente para criar menus que contém vários botões em lista, criar listas de informações e etc...

Nas figuras seguintes, 4.42 e 4.43, podemos observar todas as opções possíveis e uma implementação possível.

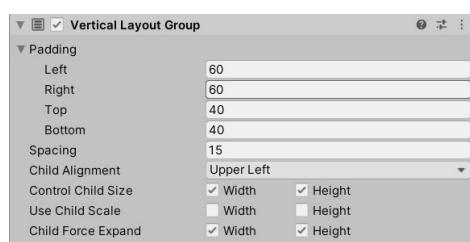


Figura 4.42: Opções disponíveis do VerticalLayout

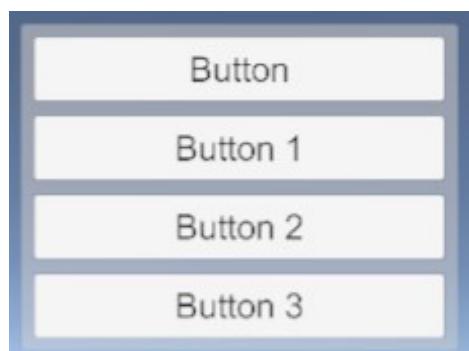


Figura 4.43: Exemplo de uma implementação do VerticalLayout

- Horizontal Group Layout - este layout cria uma linha horizontal de elementos, como foi falado anteriormente este layout é uma simplificação do GridLayout por controlar os elementos no eixo horizontal. Normalmente usados como separadores ou opções rápidas no topo das janelas.

Nas figuras seguintes, 4.44 e 4.45, podemos observar todas as opções possíveis e uma implementação possível.

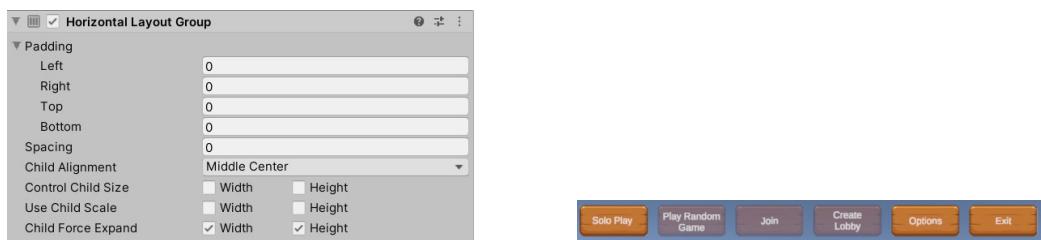


Figura 4.44: Opções disponíveis do GridLayout

Figura 4.45: Exemplo de uma implementação do GridLayout

4.9.2 Animações do UI

Já foram mencionadas as animações dos modelos dos personagens, agora, segue-se as animações da interface gráfica. Animações para as interfaces gráficas são, por exemplo, o aumento de botões quando o cursor sobrepor o mesmo.

Estas animações têm a finalidade de dar ao utilizador uma experiência mais agradável. Para a criação destas pequenas animações fizemos uso de uma técnica de animação chamada de *tweening*. Para isto, utilizámos um *asset*, da *asset store* do *Unity*, chamado de *Lean Tween*.

Tweening

Tweening é um processo de animação que procura gerar *frames* intermédios entre duas imagens, chamadas de *key frames*, para dar a aparência de que a primeira imagem evolui para a segunda, suavemente. Os *frames* gerados entre estas duas imagens, chamam-se de *inbetweens*.

A seguinte Figura 4.46 mostra dois estados de uma animação, assim como os *inbetweens*:

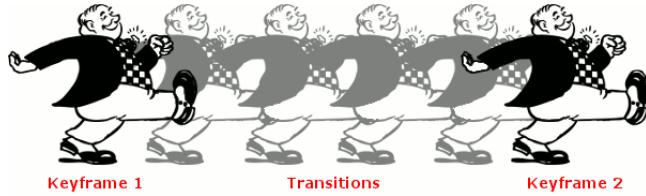


Figura 4.46: *Tweening* exemplo

4.9.3 Implementação da Interface Gráfica

De seguida apresentamos os resultados finais dos vários menus implementados no nosso jogo,

Menu Inicial

O menu inicial, Figura 4.51, é a primeira interface que o utilizador vai entrar em contacto antes de iniciar o jogo em si. Este menu têm várias funcionalidades distintas que vamos apresentar a seguir:

- Play Random Game - automaticamente entra num jogo que esteja ativo na rede, se não encontrar nenhum, vai procurar localmente se existe algum jogo guardado se sim faz *load* desse jogo e por fim se não encontrar cria um jogo novo.
- Join - abre uma janela com todos os jogos disponíveis na rede e da a possibilidade de escolher qualquer um para entrar
- Create Lobby - cria uma sala com um nome escolhido pelo jogador e este torna-se disponível para qualquer pessoa entrar.
- Options - mostra as várias opções que estão disponíveis no jogo, atualmente só é possível ajustar o volume e a qualidade do jogo.
- Exit - forma de sair do jogo quando se está farto

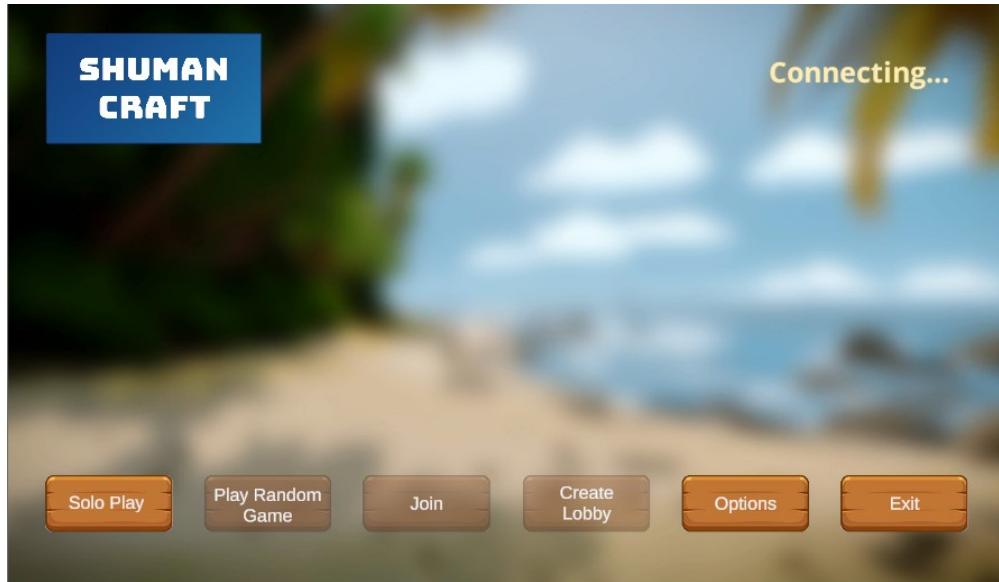


Figura 4.47: Menu Inicial

Menu de Itens

Este menu é um dos mais importantes do nosso jogo, este divide-se em duas secções uma delas é a secção de crafting, a outra é a secção dos itens.

Na secção de *crafting* estão listados todas as receitas para craftar os itens, no lado esquerdo destas têm um botão para craftar, se tiver ativo é que têm os requisitos para tal senão o botão estará disabled.

Na secção de itens estão listados todos os itens que existem no jogo incluindo a sua descrição, imagem, receita para craftar e outras informações relevantes, os itens estão separados em secções para facilitar a navegação.



Figura 4.48: Menu de Itens

Inventário

Neste menu residem todos os itens que o jogador possuí no momento, referenciada na Figura 4.8.

Hotbar

Neste menu estão os itens de acesso rápido, referenciada na Figura 4.9



Figura 4.49: Hotbar

Menu das informações

Aqui está o menu onde o utilizador pode encontrar várias informações sobre o jogo, por exemplo tutoriais, introdução ao jogo, criadores e eventuais informações:



Figura 4.50: Menu dos créditos

Stats do jogador

Esta parte, comprehende as características do jogador, tais como, a sua vida, a *stamina*, a fome e, a sede. Situa-se no canto inferior esquerdo do ecrã sempre, para que o utilizador não perca essa informação de vista.



Figura 4.51: *Stats* do jogador

4.10 Máquinas de Estados

Nesta parte do trabalho, iremos rever as máquinas de estados colocadas nos NPC's.

Um NPC (*Non Playing Character*), tal como o nome indica, é um personagem não jogável. Nenhum jogador tem forma de o controlar. São totalmente controlados por máquinas de estados, implementadas no próprio *Unity*.

Neste caso, todos os NPC's são animais, que podem ser ou **presas**, ou **predadores**.

- **Presas** - Quando avistam qualquer jogador, fogem do mesmo para não serem atacadas.
- **Predadores** - Ao verem um jogador, seguem-no até que o possam atacar.

No *Unity*, a criação de máquinas de estados é feita na janela *Animator*, que trabalha com as animações.

Dentro desta, é possível criar *Sub-State Machines*. Uma *Sub-State Machine* pode ser utilizada quando um personagem contém ações complexas, consistindo num certo número de etapas. Assim, podemos ter várias animações seguidas dentro de uma *Sub-State Machine*, tudo englobado em apenas um estado.

Contudo, a razão pela qual foi escolhido utilizar estas máquinas de estados é, o facto de ser mais fácil de adicionar *scripts*, chamados de *Behaviours* (traduzindo para o português, comportamentos) a cada estado.

Um *Behaviour*, dá-nos a oportunidade de detetar quando a máquina de estados entra num determinado estado (estado associado ao *Behaviour*), assim como a sua saída, e, caso permaneça no estado também.

De seguida vão ser escolhidos os estados para cada tipo de NPC (presa ou predador). Foi concluído que, ambos os tipos apenas necessitam de três estados:

- **Idle** - Encontra-se parado.
- **Walk** - Encontra-se a movimentar para algum lado.

- ***Chase/Flee*** - *Chase* para predadores e, *Flee* para as presas, persegue o jogador caso o aviste ou foge do mesmo, respetivamente.

Nas Figuras 4.52 e 4.53, estão representadas as máquinas de estados para os dois tipos de NPC:

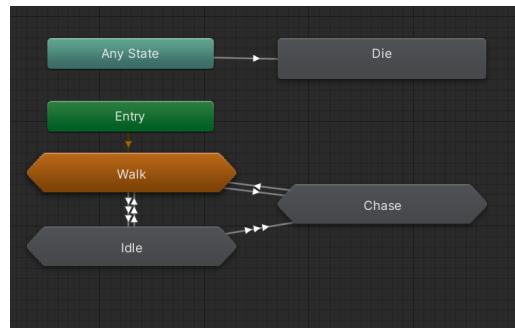


Figura 4.52: Máquina de estados - predadores

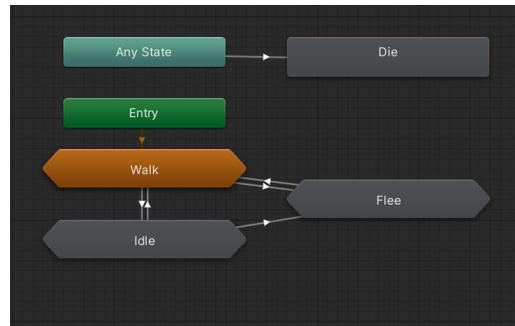


Figura 4.53: Máquina de estados - presas

Os estados *Idle* e *Walk* estão sempre a correr seguidos um do outro. Primeiro começa o *Idle* e, após o seu término, segue-se para o estado *Walk*, e assim em diante.

Só existe transição para o estado *Chase* ou *Flee* caso o NPC detete um jogador no seu espaço de visão.

Capítulo 5

Validação e Testes

Por fim, após a conclusão da implementação do modelo proposto foi necessário realizar testes para validar todas as componentes do jogo.

Uma das formas mais fáceis de conseguir disponibilizar o jogo para várias pessoas seria aproveitar o FEIM, mas devido à situação atual não se pode usar esta feira para realizar muitos testes de uma vez.

Para adaptar à situação atual foi só possível mostrar a familiares e amigos próximos fisicamente para poder mostrar o jogo, levando o computador a eles. Por fim, depois de jogarem brevemente era proposto responderem a um questionário breve para recolher as opiniões e críticas das pessoas que jogaram.

As primeiras questões procuram perceber o perfil das pessoas que testaram o jogo. Assim, facilmente se reconhece que a maior parte dos utilizadores teriam de entre a 18 e 25 anos. Também é possível verificar que a maioria já havia jogado pelo menos um jogo *survival* anteriormente. Outro aspeto é, que o estilo de jogos mais famoso de entre esta amostra são os jogos de ação. Estas estatísticas encontram-se nas Figuras 5.1, 5.2, 5.3, 5.4.

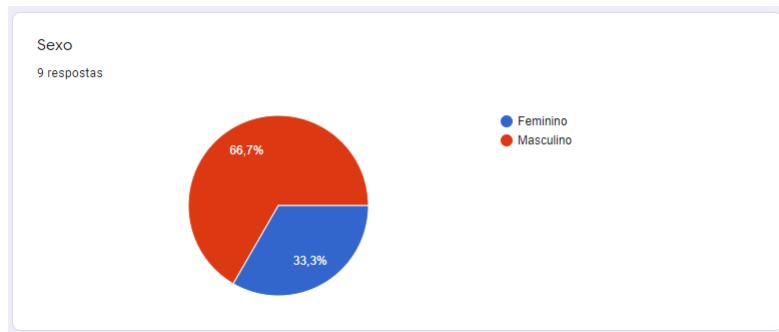


Figura 5.1: Estatísticas do Questionário - Género

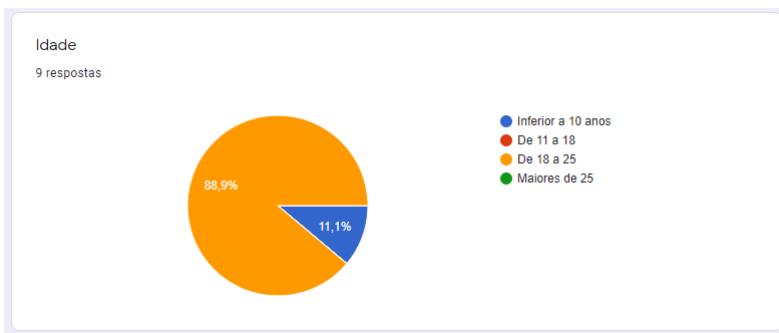
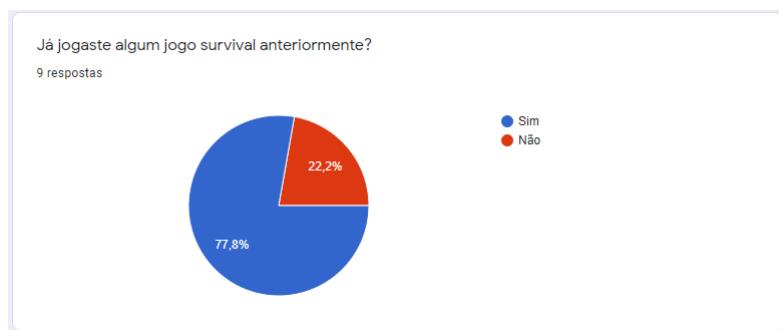


Figura 5.2: Estatísticas do Questionário - Idade

Figura 5.3: Estatísticas do Questionário - Conhecimento de jogos *survival*

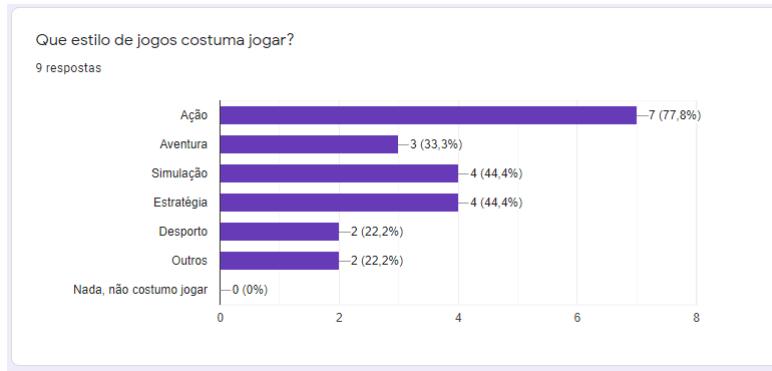


Figura 5.4: Estatísticas do Questionário - Tipos de jogos que costuma jogar

De seguida, as questões tornam-se mais relativas à experiência que os utilizadores tiveram no jogo. Pela maioria, consegue-se perceber que, não houve problemas de performance nem *bugs*, muitas das pessoas classificam o jogo como elegante e divertido. É possível também verificar que o objetivo do jogo foi algo que não houve muita dificuldade em compreender e, que a jogabilidade foi, em geral, boa. A maioria da amostra talvez compraria o jogo, mas na pergunta ”Compraria o jogo?” a resposta é relativamente mista. Tanto existem pessoas que comprariam como não.

Estes resultados encontram-se abaixo, nas Figuras 5.5, 5.6, 5.7, 5.8, 5.9, 5.10 e 5.11.

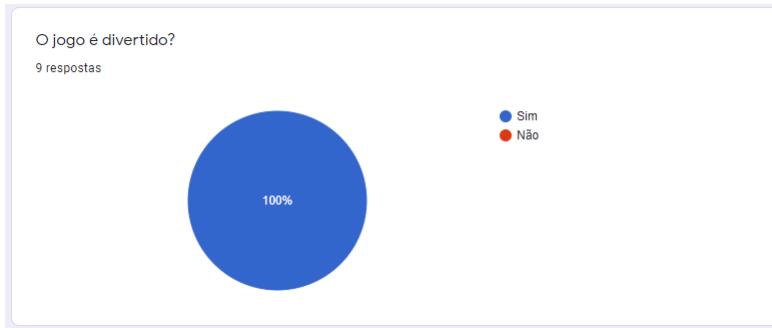


Figura 5.5: Estatísticas do Questionário - Diversão

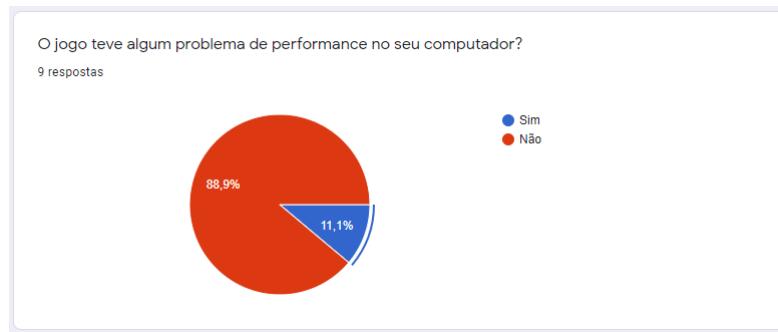


Figura 5.6: Estatísticas do Questionário - Performance

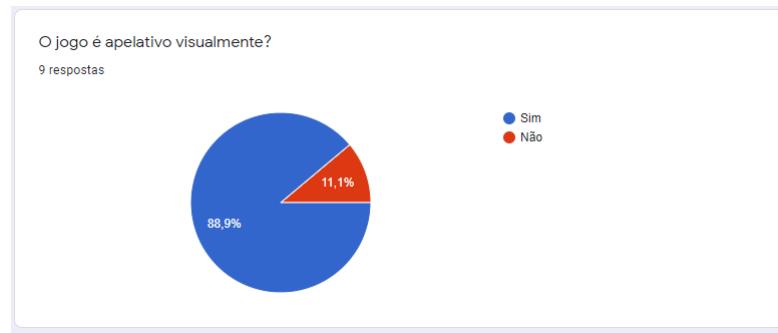


Figura 5.7: Estatísticas do Questionário - Visuais

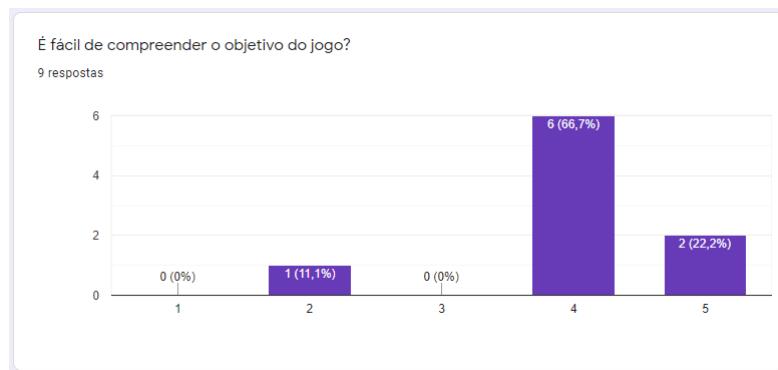


Figura 5.8: Estatísticas do Questionário - Compreensão do objetivo do jogo

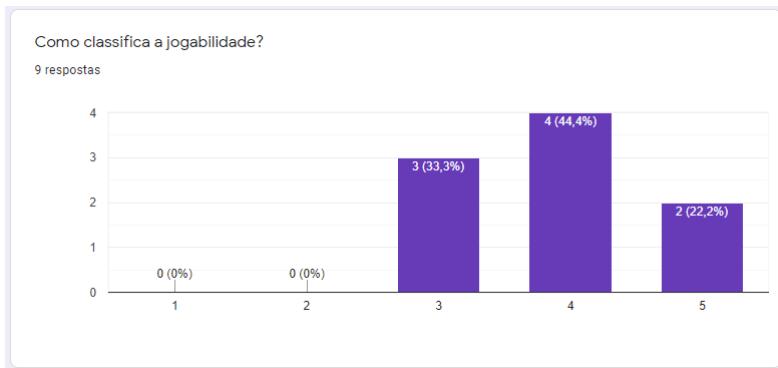


Figura 5.9: Estatísticas do Questionário - Jogabilidade

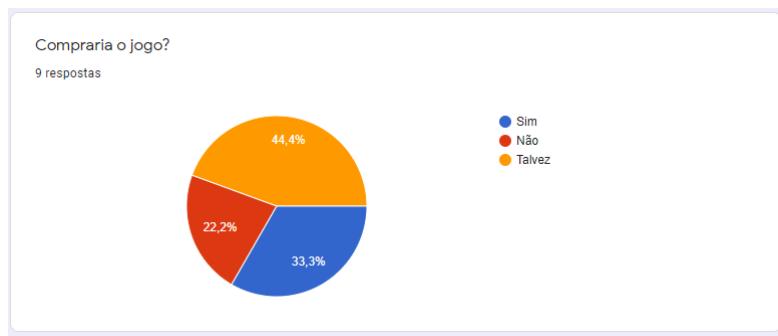


Figura 5.10: Estatísticas do Questionário - Compraria o Jogo?

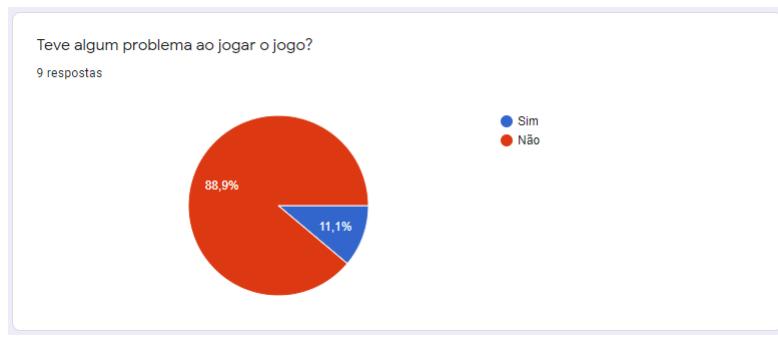


Figura 5.11: Estatísticas do Questionário - Problemas com o jogo

Por fim, pedimos aos utilizadores para colocarem sugestões. Apesar de não terem havido muitas sugestões, foi sugerido à adição de mais animais (“Sim, faltam pinguins”) e, mais introdução e tutoriais (“Falta um pouco de introdução e tutorial do jogo”).

A ultima questão pretende ver o que as pessoas pensam em geral do jogo. Estas respostas apresentam-se nas Figuras 5.12 e 5.13.

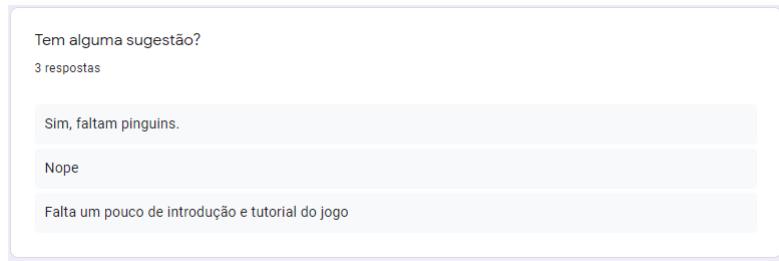


Figura 5.12: Estatísticas do Questionário - Sugestões

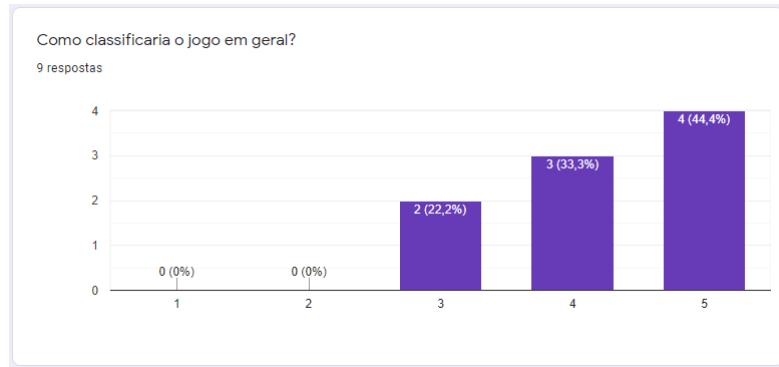


Figura 5.13: Estatísticas do Questionário - Classificação geral

Capítulo 6

Conclusões e Trabalho Futuro

O presente projeto permitiu-nos uma expansão de conhecimentos nas áreas de modelação, multimédia e, mesmo na modelação relativa à programação envolvida no jogo.

Conseguimos aprofundar os conhecimentos dados em unidades curriculares tais como Modelação de Ambientes Virtuais (MAV) e, Animação em Ambientes Virtuais (AAV). Nestas duas unidades apreendemos conhecimentos na área de modelação (com o uso do *Blender*) e, a trabalhar em ferramentas como o *Unity*. Assim, viémos a investigar mais estas áreas e, obter mais informação para que pudéssemos construir um projeto mais completo e apresentável. Ao explorar o *Unity*, encontrámos ferramentas que, se calhar não teriam muito uso nas cadeiras acima, mas que no nosso projeto têm bastante relevância.

Através do FEIM (Forum de Engenharia Informática e Multimédia) 2020, tivemos a chance mostrar o projeto a um maior número de pessoas. Desta forma, recebemos *feedback* relativo ao aspeto visual do jogo, no que diz respeito a *design* centrado no utilizador.

O uso do *Photon* como servidor expandiu também as áreas abrangidas pelo projeto. Mais facilmente compreendemos como funcionava a sincronização de objetos pela rede.

O trabalho em equipa foi também uma característica que consideramos ter tido muito impacto no desenvolvimento do jogo. No início, começámos por desenvolver o projeto em dois PC's diferentes e, sempre que queríamos atualizar alguma coisa tínhamos de enviar todas as alterações. O *GitHub*, neste aspetto acelerou muito o processo de junção de partes.

Em suma, foi realizado um jogo 3d *lowpoly* cooperativo em modo *multiplayer* onde os jogadores podem explorar um grande mapa constituído por vários biomas (praias, florestas, deserto, etc.).

Através do UI (*User Interface*), desenhado no *Photoshop*, os jogadores podem verificar o seu inventário, *craftar* ferramentas e/ou armas e, até mesmo construir estruturas no mapa.

O jogadores podem também comunicar entre si, com o uso da ferramenta de *chat*.

Tudo isto, junto com um sistema de guardar e carregar jogo que, dá aos jogadores ainda mais vontade de jogar, sabendo que os seus progressos são todos guardados.

Em termos prospectivos, existem ainda algumas ideias que, foram pensadas durante o *brainstorming* do projeto.

A geração do mapa processualmente era algo que não só dá mais enfase à ideia de que este jogo não tem um fim. Isto pois o utilizador teria sempre zonas novas a explorar.

Adicionalmente, poderá também ser melhorada a experiência de utilizador, caso haja uma maior variedade de objetos que o jogador possa construir, assim como animais e biomas. Quantas mais coisas o utilizador pode encontrar no jogo para fazer, mais divertido e desafiador se torna o mesmo.

Apêndice A

Questionário de Usabilidade do jogo

Shuman Craft

Formulário de avaliação do utilizador após jogar o jogo survival Shuman Craft.

*Obrigatório



Sexo *

Feminino
 Masculino

Idade *

Inferior a 10 anos
 De 11 a 18
 De 18 a 25
 Maiores de 25

Já jogaste algum jogo survival anteriormente? *

Sim

Não

Que estilo de jogos costuma jogar? *

Ação

Aventura

Simulação

Estratégia

Desporto

Outros

Nada, não costumo jogar

O jogo é divertido? *

Sim

Não

O jogo teve algum problema de performance no seu computador? *

Sim
 Não

O jogo é apelativo visualmente? *

Sim
 Não

É fácil de compreender o objetivo do jogo? *

1 2 3 4 5
Difícil Muito Fácil

Como classifica a jogabilidade? *

1 2 3 4 5
Má Muito Boa

Compraria o jogo? *

Sim
 Não
 Talvez

Teve algum problema ao jogar o jogo? *

Sim
 Não

Tem alguma sugestão?

A sua resposta

Como classificaria o jogo em geral? *

1 2 3 4 5

Mau Excelente

Submeter

Bibliografia

- [1] Unity Technologies. Unity Manual, 2018
<https://docs.unity3d.com/Manual/index.html>
- [2] Brackeys Youtube Tutorials
<https://www.youtube.com/user/Brackeys>
- [3] Photon Unity Networking (PUN)
<https://doc.photonengine.com/en-us/pun/current/getting-started/pun-intro>
- [4] Sons do jogo
<https://freesound.org>
<https://freesfx.co.uk>
- [5] Photoshop Tutorials
<https://helpx.adobe.com/pt/photoshop/tutorials.html>
- [6] Minecraft
<https://www.minecraft.net>
- [7] Ark Survival Evolved
https://store.steampowered.com/app/346110/ARK_Survival_Evolved/

[8] Raft

<https://store.steampowered.com/app/648800/Raft/>

[9] Sims2 Cast Away

https://sims.fandom.com/wiki/The_Sims_2:_Castaway