



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

The Code of Iga



Hugo Sousa (46304)

José Siopa (46338)

Orientador

Professor Doutor Hugo Cordeiro

16 de setembro de 2021

Resumo

No atual mundo onde vivemos, cada vez mais é frequente uma pessoa ter uma vida agitada e com muito “stress”. É por isso necessário criar formas de entretenimento para nos abstrairmos do mundo real durante um tempo, sendo os videojogos uma boa forma para tal.

Este projeto debruça-se na criação de um videojogo para computador, utilizando o motor de jogo Unity.

O jogo é do tipo combate 3D e tem como objetivo final escapar de uma prisão futurista, tendo o jogador de planear corretamente os seus movimentos e a sua estratégia.

O projeto terá como pontos focais o desenvolvimento de uma inteligência artificial utilizando árvores de comportamento, criando assim uma maior imersão durante o jogo, como também o uso da ferramenta Cinemachine, que permitirá implementar diferentes efeitos e perspetivas para a câmara do jogador.

Este projeto também pretende explorar o Universal Render Pipeline com o objetivo de melhorar e otimizar a qualidade gráfica.

O projeto implementado conclui assim a qualidade gráfica desejada num mapa bem detalhado, bem como uma jogabilidade desafiadora e envolvente.

Abstract

In the current world in which we live, it is increasingly common for a person to have a busy life with a lot of stress. It is therefore necessary to create forms of entertainment to abstract ourselves from the real world for a while and video games are a good way to do that.

This project focuses on the creation of a videogame for computer, using the Unity game engine for its development.

The game is of 3D combat type and has the ultimate objective to escape from a futuristic prison, having the player correctly plan their movements and their strategy.

The project will have as focal points the development of an artificial intelligence using behavior trees, thus creating greater immersion during the game, as well as the use of the Cinemachine tool, which will allow the implementation of different effects and perspectives for the player's camera.

This project also intends to explore the Universal Render Pipeline in order to improve and optimize graphic quality.

The implemented project thus completes the desired graphic quality in a very detailed map, as well as a challenging and immersive gameplay.

Índice

Resumo	i
Abstract	iii
Índice	v
Lista de Tabelas	vii
Lista de Figuras	ix
Lista de Listagens	xi
1 Introdução	1
2 Trabalho Relacionado	3
2.1 Ghostrunner	3
2.2 Nioh	4
2.3 Metal Gear Solid 1	5
3 Modelo Proposto	7
3.1 Requisitos	7
3.1.1 Caraterização Geral	7
3.1.2 Caraterização Pormenor	8
3.2 Fundamentos	11
3.2.1 Universal Render Pipeline	11
3.2.2 HDRI	12
3.2.3 Árvore de comportamentos	13

4 Implementação do Modelo	15
4.1 Mapa	15
4.1.1 Nível 1	17
4.1.2 Nível 2	18
4.1.3 Nível 3	19
4.1.4 Nível 4	19
4.1.5 Loading	20
4.1.6 Iluminação	21
4.2 Câmaras	23
4.3 Jogador e personagem principal	25
4.3.1 Animações	25
4.3.2 Movimentação e controlos	25
4.3.3 Combate	27
4.4 Inimigos	29
4.4.1 Animações	29
4.4.2 Implementação da árvore de comportamentos	31
4.5 Interface Gráfica	33
4.5.1 Menu Inicial	33
4.5.2 Interface In-game	34
4.6 Áudio	36
4.7 Diagrama do projeto	37
5 Conclusões e Trabalho Futuro	39
A Controlo de Versões	41
Bibliografia	43

Lista de Tabelas

3.1	Funções do Sistema	9
3.2	Atributos do Sistema	9
3.3	Caso de Utilização - Começar jogo	10
3.4	Caso de Utilização - Sair do jogo	10
3.5	Caso de Utilização - Controlar o volume do áudio	10
3.6	Caso de Utilização - Combater um inimigo	11
4.1	Controlos de movimentos do jogador	26

Lista de Figuras

2.1 Jogo Ghostrunner	4
2.2 Jogo Nioh	5
2.3 Jogo Metal Gear Solid	6
3.1 Exemplo da ferramenta Shader Graph	12
3.2 Comparação entre LDR (à esquerda) e HDR (à direita)	13
4.1 'The Panopticon' by Jenni Fagan	16
4.2 Esboço Final do Mapa	17
4.3 Nível 1 - Celas	18
4.4 Nível 2 - Laboratórios	18
4.5 Nível 3 - Armazéns	19
4.6 Nível 4 - Quartos	20
4.7 Loading Screen	21
4.8 Skybox	22
4.9 Exemplo da implementação das luzes ("light probes" - círculos amarelos e "reflection probes" - círculo branco)	23
4.10 Exemplo da distribuição das câmeras e aspecto no inspetor	24
4.11 Árvore de animações do jogador	25
4.12 Input Manager	26
4.13 Exemplo da chamada de um método durante uma animação	27
4.14 Personagem e as suas colisões	28
4.15 Árvore de animações dos inimigos	29
4.16 Correto posicionamento das mãos na arma utilizando cinematática inversa	30
4.17 Animação de recarregar a arma utilizando cinematática inversa	31
4.18 Árvore de comportamentos dos inimigos	32

4.19 Ecrã inicial	34
4.20 Menu de opções	34
4.21 Interface do jogador	35
4.22 Menu de pausa	36
4.23 Audiomixer	37
4.24 Diagrama do projeto	38
A.1 Controlo de versões	42
A.2 Upload de nova versão	42

Lista de Listagens

4.1	Função Loading	20
4.2	Calculo da posição do jogador no mini-mapa	35
4.3	Conversão de decibéis para um valor entre 0.001 e 1	37

1

Introdução

De modo a criar mais formas de entretenimento para as pessoas, este projeto visa a criação de um videojogo de combate em 3D para computador, utilizando o motor de jogo Unity para o seu desenvolvimento.

O foco principal do projeto foca-se na exploração do Universal Render Pipeline para a criação de gráficos otimizados mais apelativos e com maior qualidade.

O jogador irá assumir o papel de um ninja e terá de escapar de uma prisão espacial, tendo que enfrentar vários inimigos pelo caminho. Estes irão possuir uma árvore de comportamentos, havendo uma maior variedade de movimentos para criar uma maior imersão para o jogador.

Também será explorada a ferramenta Cinemachine para criar uma maior diversificação de câmaras, havendo assim diferentes perspetivas ao longo do jogo.

Os principais aspetos que motivaram o desenvolvimento deste projeto relacionam-se com o interesse pelo mundo dos videojogos, podendo assim criar um jogo que inclua os temas mais apreciados pelos programadores.

A disciplina de Animação em Ambientes Virtuais também serviu de motivação, onde se pode verificar que esta é uma área sempre em fases de crescimento, dando a possibilidade de criar qualquer tipo de aplicação nos

motores de jogos disponibilizados.

Este relatório está dividido em seis capítulos. O primeiro capítulo refere a introdução e motivação do trabalho, o segundo apresenta os trabalhos relacionados, o terceiro descreve o modelo proposto, seguido pelo capítulo onde é descrito o processo de implementação. O penúltimo capítulo apresenta os testes realizados ao jogo e o último capítulo contem as conclusões e as ideias para o trabalho futuro.

2

Trabalho Relacionado

Neste capítulo irão ser apresentados jogos em que este projeto está relacionado e que serviram de inspiração para o seu desenvolvimento.

Além da sua apresentação, serão referidas as semelhanças e as diferenças em relação a este projeto, bem como melhorias idealizadas.

2.1 Ghostrunner

O jogo Ghostrunner, figura 2.1, é um videojogo do estilo “hack and slash”, em primeira pessoa, que decorre num futuro apocalíptico, seguindo um estilo visual género “Cyberpunk”, originado em filmes como “Blade Runner” e “The Matrix”.



Figura 2.1: Jogo Ghostrunner, [One More Level, 2020]

O jogo centra-se no conceito de velocidade e agilidade, em que tanto o jogador e os inimigos são mortos em apenas um golpe, sendo o objetivo usar a mobilidade extra do personagem, como correr nas paredes e poder dar duplos saltos, para conseguir chegar ao final ileso no menor tempo possível.

Esta foi a nossa inspiração inicial, principalmente para a estética geral do jogo (“Cyberpunk” / Futurista) e com uma jogabilidade de alta dificuldade em combate.

2.2 Nioh

O segundo jogo, que serviu de inspiração para as animações de combate da personagem principal, é Nioh, figura 2.2, um “Action RPG”, que decorre no Japão feudal (1600) contando os acontecimentos reais, com elementos míticos no meio.



Figura 2.2: Jogo Nioh, [Koei Tecmo Games Co., LTD, 2017]

Nioh foca-se na ação, em que o jogador dispõe de uma grande abundância de ataques, com animações fluidas, onde se podem realizar diferentes golpes em sequência (“combos”).

2.3 Metal Gear Solid 1

O terceiro jogo, que serviu de inspiração para o estilo/tema de furtividade do jogo, é Metal Gear Solid, figura 2.3, o pioneiro desta categoria de jogos, estabelecendo novos padrões para a indústria com cada sequela, visto como um dos maiores ícones dos videojogos.



Figura 2.3: Jogo Metal Gear Solid, [Konami Computer Entertainment Japan, 1998]

3

Modelo Proposto

De seguida, encontrar-se-ão expostas as características do projeto, assim como os requisitos e os fundamentos aplicados para a sua realização. Será ainda mencionada a abordagem tomada na implementação de certos aspectos críticos do projeto.

3.1 Requisitos

3.1.1 Caraterização Geral

Contextualização

A Okatsu, a personagem principal, foi capturada na Terra e levada para uma prisão espacial, dado ela ser a última ninja Iga da sua geração.

A força inimiga, denominada por Oicipsoh, é composta por um exército futurista quer conquistar o universo aprendendo todas as técnicas de combate, incluindo os golpes ninjas para se tornarem a unidade mais forte existente.

Assim, o objetivo da Okatsu é lutar para escapar da prisão e evitar que os seus segredos e técnicas sejam expostos aos inimigos.

Objetivos

A progressão do jogo será baseada em salas. Isso significa que cada sala é considerada um nível com o objetivo principal de “limpá-la” (lutar contra todos os inimigos) ou de passar por eles.

O objetivo principal do jogo é chegar à última sala para conseguir escapar da prisão espacial.

Mecânicas

Sendo o personagem principal um ninja, a arma deste é uma espada e é possível realizar várias sequências de ataques (chamados “combos”) para infligir mais dano aos inimigos. O jogador também pode rebolar e utilizar um “dash” para se desviar das balas ou aproximar-se dos inimigos.

Para seguir para a próxima sala, o jogador terá que interagir com a porta e os objetos do próximo nível serão carregados.

Relativamente aos inimigos, estes estarão a patrulhar sobre uma rota previamente definida e equipados com armas de fogo, disparando sobre o jogador sempre que o virem.

3.1.2 Caraterização Pormenor

Funções do sistema

As funções do sistema (tabela 3.1) representam as funcionalidades que o sistema deverá cumprir. Algumas destas serão visíveis para o utilizador, enquanto as outras não, mas serão necessárias para o funcionamento do projeto.

Ref. #	Função	Categoria
R1.1	Movimentação do personagem	Visível
R1.2	Inimigos atacam o personagem	Visível
R1.3	Inimigos avaliam e percepcionam o ambiente	Invisível
R1.4	Carregamento dos objetos ao atravessar uma porta ou ao iniciar o jogo	Invisível
R1.5	Jogo recomeça sempre que o jogador morre	Visível
R1.6	Transição de câmaras dependendo da localização do jogador	Visível
R1.7	Pausar o jogo	Visível
R1.8	Alterar o volume dos efeitos sonoros ou música	Visível

Tabela 3.1: Funções do Sistema

Atributos do sistema

Os atributos do sistema (tabela 3.2) são características do sistema. Poderão haver um conjunto de restrições associados a estes atributos, inserindo um detalhe adicional aos atributos descritos. A coluna referente à categoria indica se o atributo referido é obrigatório ser cumprido ou opcional.

Atributo	Detalhe	Categoria
Plataformas	Computador (Windows)	Obrigatório
Performance	Jogabilidade fluída com pelo menos 60 FPS	Desejável
Interação Pessoa-Máquina	Utilização de teclado/rato ou comando	Obrigatório
	Tempo de resposta instantâneo	Obrigatório
Gráficos	Alta qualidade gráfica incluindo reflexos e sombras	Obrigatório
	Interface simples e de fácil compreensão	Desejável
Dificuldade	Dificuldade aumenta com a progressão do jogo	Desejável

Tabela 3.2: Atributos do Sistema

Casos de utilização

De seguida, são apresentadas as tabelas 3.3 à 3.6, que representam os casos de utilização do projeto, fazendo referência às funções do sistema.

Caso de Utilização 1	
Nome:	Começar jogo
Resumo:	Jogador clica no botão “Play” e os objetos do primeiro nível são carregados, podendo movimentar-se
Referências:	R1.1, R1.4, R1.6

Tabela 3.3: Caso de Utilização - Começar jogo

Caso de Utilização 2	
Nome:	Sair do jogo
Resumo:	Sair do jogo utilizando o botão “Exit” no menu inicial ou no menu de pausa
Referências:	R1.1, R1.4, R1.6

Tabela 3.4: Caso de Utilização - Sair do jogo

Caso de Utilização 3	
Nome:	Controlar o volume do áudio
Resumo:	Jogador pausa o jogo e pode controlar o áudio dos efeitos sonoros ou da música
Referências:	R1.7, R1.8

Tabela 3.5: Caso de Utilização - Controlar o volume do áudio

Caso de Utilização 4	
Nome:	Combater um inimigo
Resumo:	Jogador dispõe de um conjunto de controlos para lutar contra os inimigos, onde estes reagem aos comportamentos do jogador
Referências:	R1.1, R1.2, R1.3, R1.6

Tabela 3.6: Caso de Utilização - Combater um inimigo

3.2 Fundamentos

3.2.1 Universal Render Pipeline

A “Universal Render Pipeline”, URP, é uma “pipeline” de renderização do Unity, feita com um foco no desempenho da plataforma, como também na flexibilidade e personalização da “pipeline” através de “scripts C#”, em contraste com a que já se encontrava integrada.

O foco no desempenho, permite ter uma qualidade gráfica que se adapta ao desempenho corrente da plataforma, tornando-se indicado para o desenvolvimento de jogos cuja plataforma seja de baixo desempenho, como plataformas moveis ou consolas portáteis.

A “pipeline” contém já efeitos “post-processing” integrados diretamente, como “Anti-aliasing, Depth of Field, Motion Blur, Panini Projection, Bloom”, oferecendo facilidade na customização dos mesmos.

Ao contrário da “pipeline built-in”, esta oferece suporte ao “Shader Graph”, uma ferramenta visual, que simplifica a criação de novos shaders customizáveis, através de “nodes” que se ligam entre si, cada um com uma designada função, estando um exemplo do mesmo visível na figura 3.1.

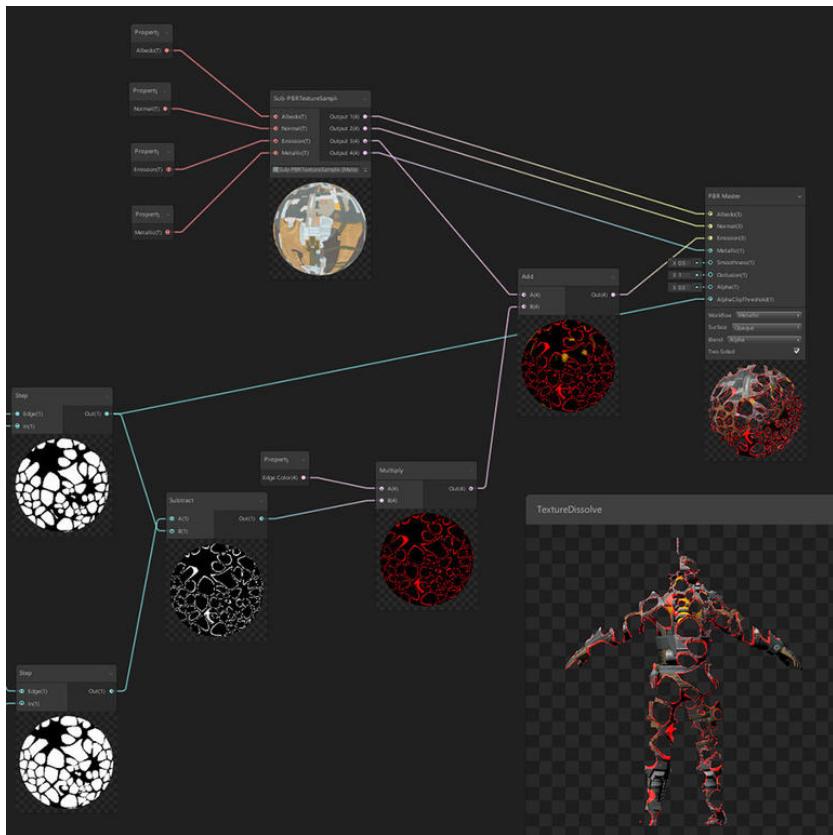


Figura 3.1: Exemplo da ferramenta Shader Graph

A URP ainda está em desenvolvimento, e ainda carece de funções que a “pipeline built-in” contém, mas oferece mais flexibilidade nas que dispõe, sendo o objetivo tornar esta a de-facto “pipeline” integrada, substituindo a existente, após esta estar completa.

3.2.2 HDRI

Uma HDRI, sigla para “High Dynamic Range Image”, trata-se de uma imagem panorâmica que, como o nome indica, tem um alto alcance dinâmico.

Aludindo à informação extra que possuem sobre a iluminação da imagem, as HDRI podem conter até 32 bits por píxel em cada canal de luz, em contraste com as imagens LDR, “Low Dynamic Range”, que guardam apenas 8 bits por cada canal RGB em cada píxel.

Assim, as HDRI, ao contrário do formato JPG, podem ter valores de luminosidade maiores e mais detalhados, conseguindo imagens quase tão re-



Figura 3.2: Comparação entre LDR (à esquerda) e HDR (à direita)

alistas como as que o ser humano perceciona. Uma comparação entre ambas pode ser vista na figura 3.2.

As HDRI, devido ao elevado número de informação extra que possuem, podem ser usadas para iluminar cenas 3D de uma forma mais realista, nomeadamente quanto à luminosidade e sombras da imagem, sem ter de recorrer a, no caso do Unity, luzes extras.

3.2.3 Árvore de comportamentos

Uma árvore de comportamentos é representada graficamente como uma árvore direcionada onde os nós são classificados como raiz, nós de fluxo de controlo ou nós de execução (tarefas). Para cada par de nós conectados, o nó de saída é chamado pai e o nó de entrada é chamado filho. Graficamente, os filhos de um nó são colocados debaixo dele, ordenados por importância, da esquerda para a direita.

A execução de uma árvore de comportamento começa na raiz, que envia tiques com uma certa frequência para o seu filho. Um tique é um sinal de habilitação que permite a execução de um nó filho. Quando a execução de um nó na árvore de comportamento é permitida, ele retorna ao pai um “status”: “running”, se a sua execução ainda não terminou; “success”, se atingiu o seu objetivo ou “failure” caso contrário.

Nós de fluxo de controlo

Um nó de fluxo de controlo é usado para controlar as subtarefas das quais ele é composto. Este pode ser um nó seletor ou um nó de sequência. As suas

subtarefas são sempre executadas à vez. Quando uma subtarefa é concluída e retorna o seu “status” (sucesso ou falha), o nó do fluxo de controle decide se executa a próxima subtarefa ou não.

Os nós seletores são usados para localizar e executar o primeiro nó filho que não falha. Um nó seletor retornará imediatamente com um código de “status” de sucesso ou execução quando um dos seus filhos retornar sucesso ou execução.

Os nós de sequência são usados para localizar e executar o primeiro nó filho que ainda não foi bem-sucedido. Um nó de sequência retornará imediatamente com um código de “status” de falha ou execução quando um dos seus filhos retornar falha ou execução.

4

Implementação do Modelo

Este capítulo será responsável por descrever as várias facetas da implementação do nosso projeto, como as respetivas explicações dos conceitos e decisões tomadas em cada uma.

4.1 Mapa

A inspiração principal para a construção do mapa foi o conceito de “Panopticon” (Panótico em português), uma categoria de estabelecimento prisional, desenhado e proposto por Jeremy Bentham, filósofo inglês e investigador de ciências sociais, durante o século XVIII (figura 4.1).

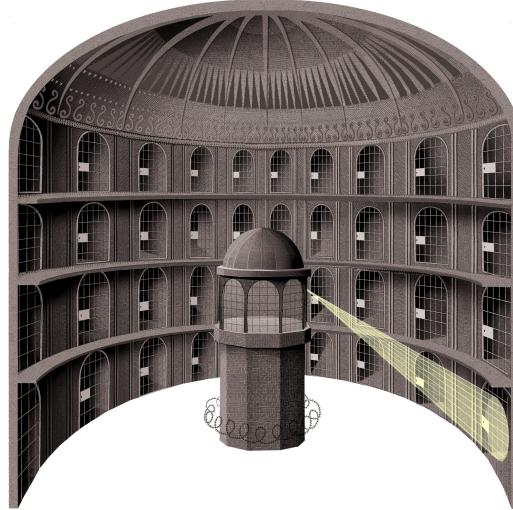


Figura 4.1: 'The Panopticon' by Jenni Fagan

"The Building circular – an iron cage, glazed – a glass lantern about the size of Ranelagh – The Prisoners in their Cells, occupying the Circumference – The Officers, the Centre. By Blinds, and other contrivances, the Inspectors concealed from the observation of the Prisoners: hence the sentiment of a sort of invisible omnipresence. – The whole circuit reviewable with little, or, if necessary, without any, change of place." - Jeremy Bentham (1791)

O conceito principal do edifício consiste em que todos os prisioneiros possam ser observados por apenas um guarda prisional, localizado na peça central do edifício, sem saberem que estão a ser observados. Mesmo que seja praticamente impossível o guarda conseguir vigiar todos os prisioneiros de uma só vez, devido a estes estarem na incógnita de estarem a ser vigiados, sentem-se compelidos a comportarem-se de forma adequada.

Com base neste conceito, foi criada uma prisão futurista, com uma peça central, semelhante à do edifício em que foi inspirado, que irá ser a ligação entre as restantes partes do mapa, nomeadamente conjuntos de celas, laboratórios e armazéns.

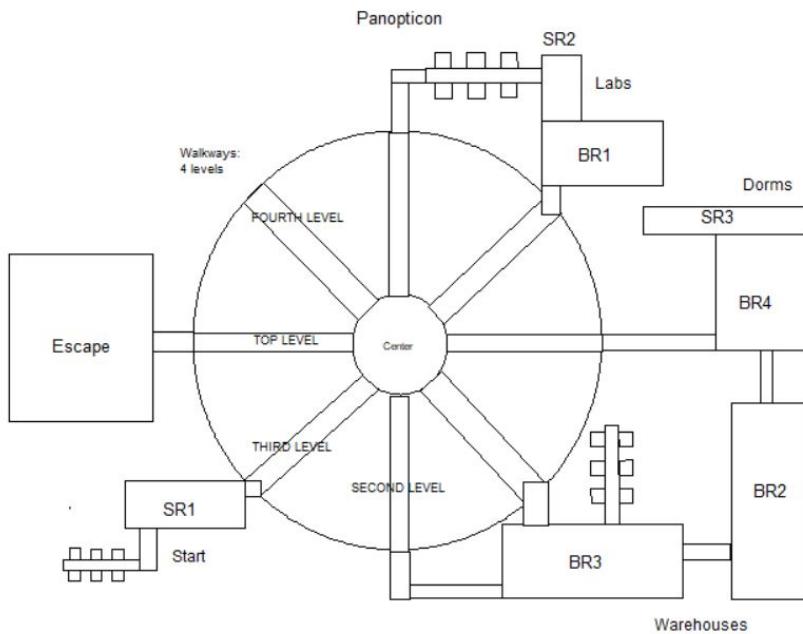


Figura 4.2: Esboço Final do Mapa

4.1.1 Nível 1

Esta componente consiste numa pequena sala e num bloco de celas, onde o protagonista começa a sua fuga, sendo possível observar parte desse cenário na figura 4.3. Aqui é apresentado ao jogador o seu primeiro desafio, apenas com um inimigo a patrulhar a zona. Após chegar ao fim, o jogador poderá avançar para o centro da prisão.



Figura 4.3: Nível 1 - Celas

4.1.2 Nível 2

Esta componente consiste em duas salas com um bloco de celas adjacente, compondo a parte laboratorial da prisão, onde são realizadas experiências nos prisioneiros, sendo possível observar parte desse cenário na figura 4.4. Esta zona irá acrescentar uma dificuldade acrescida, com uma maior quantidade de inimigos e a presença de escadas para níveis superiores da mesma sala, adicionando novas formas de abordar o nível.



Figura 4.4: Nível 2 - Laboratórios

4.1.3 Nível 3

Esta componente consiste em duas salas de grande comprimento com um bloco de celas incorporado, compondo os armazéns da prisão, onde se encontra grande parte do armamento e os geradores que dão eletricidade ao edifício, sendo possível observar parte desse cenário na figura 4.5. Esta zona irá novamente aumentar a dificuldade, com uma maior quantidade de inimigos e com uma disposição mais complexa.



Figura 4.5: Nível 3 - Armazéns

4.1.4 Nível 4

A componente final consiste em duas salas, onde se encontra os quartos da tripulação e a sala do chefe prisional, contendo o cartão que o jogador terá de obter para sair da prisão, pela saída que se encontra no lado oposto do nível, sendo possível observar parte desse cenário na figura 4.6. Esta zona irá apresentar o último desafio ao jogador, contendo vários inimigos numa sala fechada, com o chefe prisional na sua sala respetiva.



Figura 4.6: Nível 4 - Quartos

4.1.5 Loading

Para melhorar o desempenho do jogo, cada nível acima descrito foi separado para uma cena individual, sendo apenas carregada aquela em que o jogador está. Assim não são carregadas partes desnecessárias do mapa, ou seja, partes em que o jogador não está nem vê, havendo menos área para processar. Assim, atinge-se regularmente 60 FPS, como desejado, em contraste dos 10-20 FPS caso tudo estivesse carregado de uma só vez.

Para transitar entre zonas, foram colocadas regiões de “trigger” nas portas que marcam o fim de um nível, cujas o jogador pode interagir para mudar de área. O “loading” dos níveis é feito de forma assíncrona através da função no excerto 4.1.

Listagem 4.1: Função Loading

```

1 private void Load()
2 {
3     scenesToLoad.Add(SceneManager.UnloadSceneAsync(
4         sceneToUnload));
5     scenesToLoad.Add(SceneManager.LoadSceneAsync(gameObject.
6         name, LoadSceneMode.Additive));
7     StartCoroutine>LoadingScreen()
```

Foi criado um ecrã para “loading”, na figura 4.7, enquanto a próxima cena não é carregada, possuindo uma barra para mostrar o progresso do carregamento (a laranja).

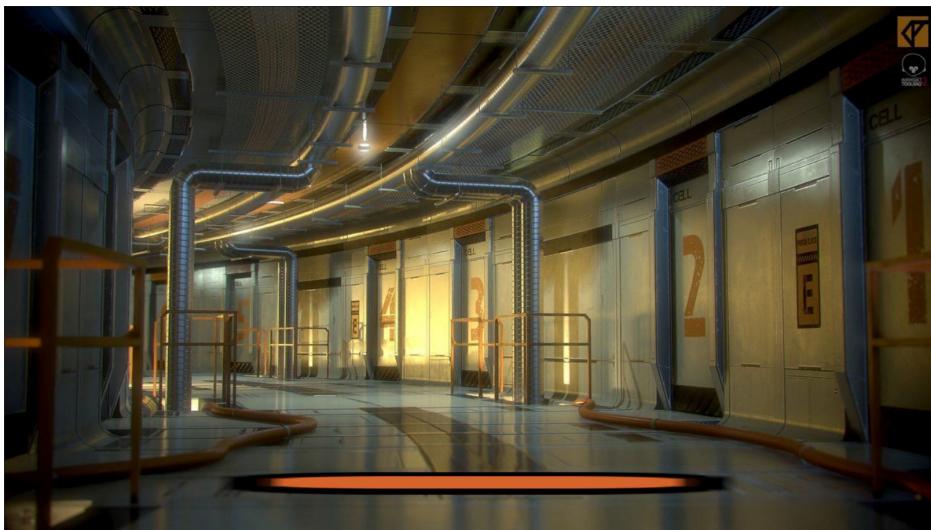


Figura 4.7: Loading Screen

4.1.6 Iluminação

A iluminação geral do mapa foi feita recorrendo a “Point Lights”, pois são as que melhor simulam lâmpadas, indicadas para ambientes interiores na maior parte dos casos. Normalmente, as “Point Lights” estariam no modo “mixed”, permitindo ter a iluminação “baked” em “lightmaps” para os objetos estáticos, como também luz dinâmica para objetos não estáticos. No entanto, a “URP” não permite usar este modo neste tipo de luzes, devido à possibilidade de piorarem o desempenho geral do jogo, tendo assim de as “Point Lights” estar no modo “baked”. Isto impossibilita ter sombras nos personagens sendo estes objetos dinâmicos. Mesmo assim, continua-se a usar esta categoria de luzes, pois são as que melhor simulam a iluminação pretendida.

No decorrer deste trabalho, o problema acima descrito foi eventualmente resolvido na versão mais atual do Unity (v2021.1), mas decidiu-se não a usar devido a possíveis problemas de instabilidade e de portabilidade da nova versão.

A iluminação da parte central, sendo esta a espaço aberto, é garantida pela luz natural proveniente da “skybox” escolhida, visível na figura 4.8, uma HDRI (High Dynamic Range Image), que permite emitir luz a partir da mesma, não sendo necessário usar uma “Directional Light”, que iria interferir com a iluminação dos espaços interiores.

Além disso, sendo que a prisão se encontra no espaço e a única fonte de luz natural ser o sol fixo na “skybox”, pode-se fixar o ponto de onde a luz é emitida, em vez de se usar a “Directional Light” onde apenas é possível ajustar o ângulo, tornando assim a iluminação exterior mais realista.



Figura 4.8: Skybox

Para melhorar a iluminação geral dos interiores foram utilizados mais duas componentes: “reflection probes” e “light probes”. Um exemplo de ambos pode ser visível na figura 4.9.

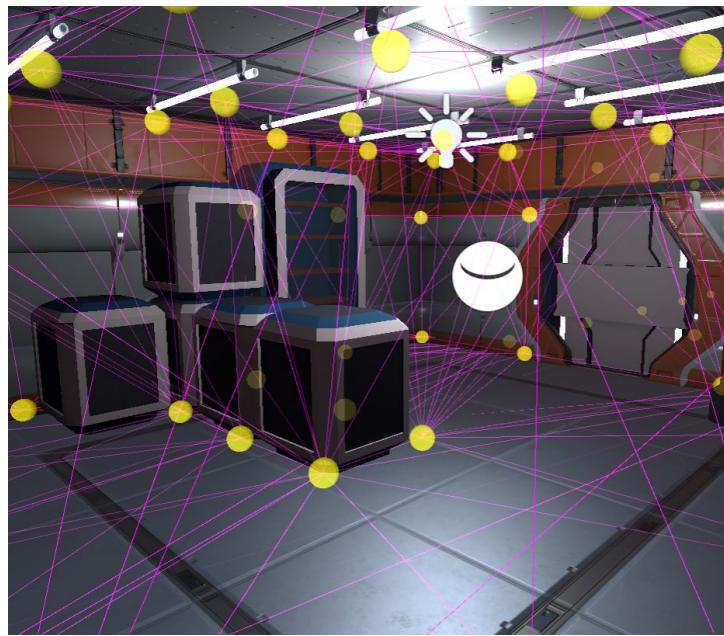


Figura 4.9: Exemplo da implementação das luzes (“light probes” - círculos amarelos e “reflection probes” - círculo branco)

O primeiro permite introduzir reflexão em objetos com componentes metálicos, ou semelhante desde que tenha um componente refletivo. O “probe” age como uma câmara, que captura uma imagem ao seu redor e guarda-a como um “cubemap”, usada nos objetos que interagem com o “probe”.

Os “light probes” permitem guardar informação sobre a luz a passar no espaço vazio da cena, usando essa informação para passar a luz a objetos dinâmicos. Como estes não guardam a informação da luz, não reagem a diferenças na iluminação no ambiente, podendo causar problemas de imersão ao jogador, caso estes “probes” não estejam implementados.

4.2 Câmaras

O sistema de câmaras do mapa foi montado com o “package” Cinemachine, havendo uma câmara principal (o “brain”), que irá mudar de comportamento e de posição dependendo de onde o jogador se encontra, que estará designado numa câmara virtual (“Virtual Camera”).

Para saber que câmara virtual ativar, foram espalhadas zonas, designadas de “camera zones”, que possuem um “collider” em modo “trigger” e um

“script” `Camera Zone`, ativando a câmera pretendida caso o jogador entre nessa zona.

Foram colocadas 48 câmeras ao longo do mapa inteiro, variando entre dois tipos de câmeras virtuais: “Do Nothing” e “Tracked Dolly”. Um exemplo pode ser visto na figura 4.10.

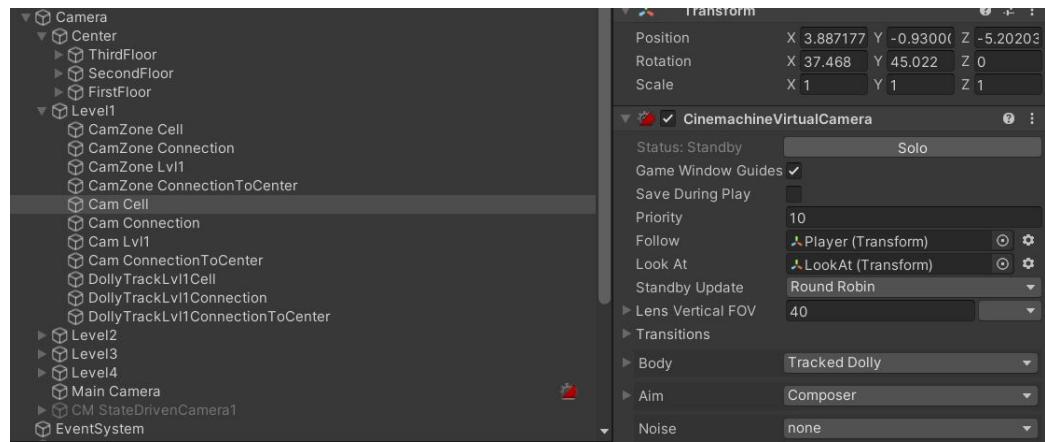


Figura 4.10: Exemplo da distribuição das câmeras e aspeto no inspetor

O primeiro, como o nome indica, não movimenta a câmera, apontando-a apenas para o jogador, seguindo o seu movimento até este sair da zona associada. Esta categoria de câmera foi escolhida, ao invés de seguir o jogador por trás como normal, como decisão estilística, para fazer parecer que são as câmeras de segurança da prisão. apesar de, por vezes, obstruir ligeiramente o campo de visão em certas zonas.

O segundo utiliza outro componente do Cinemachine, o “dolly track”, que dispõe inicialmente de dois pontos, podendo-se adicionar mais, onde a câmera irá seguir o alvo a ela atribuído pelo caminho delineado, movimentando-se suavemente entre os pontos mais próximos do alvo.

Quando o jogador muda de zona, há uma transição entre câmeras, que pode ser editada entre câmeras específicas, utilizando “custom blends”, mas optou-se por deixar uma transição “default” utilizada por todas. Neste jogo, os tipos de transições usadas foram o “cut”, permitindo uma transição instantânea, e o “Ease In Out”, que executa uma transição suave entre câmeras, podendo ser modificado o tempo de duração da mesma.

4.3 Jogador e personagem principal

4.3.1 Animações

Como foi referido na secção 3.1.1 em Mecânicas, a personagem principal irá utilizar uma espada como arma. Assim sendo, recorrendo a um personagem retirado do “website” Mixamo [Adobe, 2021] e animações obtidas através da loja de “assets” do Unity [wemakethegame, 2017].

Utilizou-se a componente “animator” para utilizar as animações transferidas e procedeu-se à sua ligação através de condições, para suavizar as transições entre animações. Na figura 4.11 observam-se todas as animações existentes, sendo as setas brancas representativas de condições de transição.

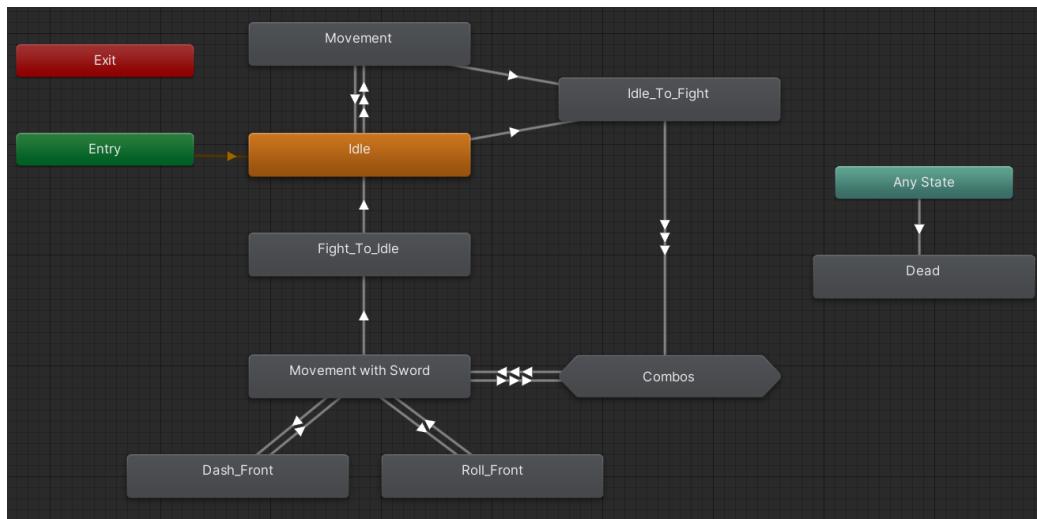


Figura 4.11: Árvore de animações do jogador

4.3.2 Movimentação e controlos

De seguida, criou-se o “script” que recebe os “inputs” do jogador e envia os valores necessários para as condições do “animator” e adicionou-se a componente “character controller” ao objeto do personagem.

Com recurso ao “package input manager”, é possível juntar e atribuir os “inputs” do teclado e do comando pretendidos a uma só ação, o que facilita na construção do código. Na figura 4.12 observam-se várias ações, estando expandida a ação de ataque principal (“Fire1”) associada ao “joystick button 0”, gatilho direito do comando.

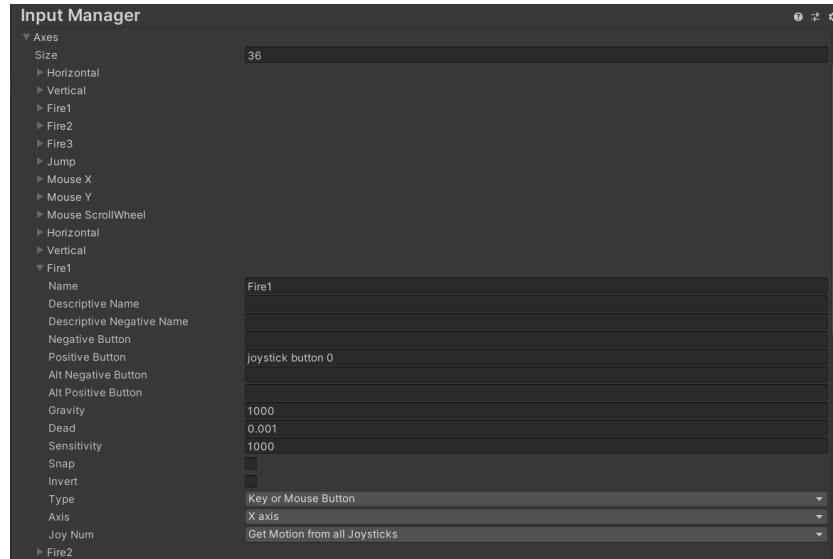


Figura 4.12: Input Manager

O “script” do jogador que trata dos movimentos e das ações do jogador avalia em cada instante se houve alguma tecla pressionada (teclado, rato ou comando). Ativando a opção que aplica “root motion”, a movimentação da personagem fica da responsabilidade das animações, não sendo assim necessário realizar código para movimentá-la.

Na tabela 4.1, encontra-se a lista de controlos de movimentos do jogador e a sua função (os controlos referentes ao comando pertencem ao comando XBOX).

Teclado/Rato	Comando	Função
Teclas W, A, S, D	Analógico esquerdo	Movimentar o personagem
Shift esquerdo	Trigger esquerdo	Correr
Tecla Q	Botão B	Dash
Tecla X	Bumper esquerdo	Rebolar
Clique esquerdo	Botão A	Ataque leve
Clique direito	Botão X	Ataque pesado

Tabela 4.1: Controlos de movimentos do jogador

Este “script” também possui dois temporizadores sempre a contar de-

crescentemente. Um deles serve para registar a sucessão de ataques para realizar os “combos”, ou seja, por exemplo, se o jogador clicar duas vezes no rato num curto espaço de tempo, são realizadas duas animações de ataque diferentes, escalando o dano de cada uma. O outro temporizador serve para alternar entre o modo passivo e agressivo, ou seja, enquanto o temporizador não chegar a zero, o jogador tem a espada na mão e está na pose de ataque, quando chegar a zero, guarda a espada e muda para uma pose “relaxada”. Este temporizador redefine-se sempre que o jogador ataca.

4.3.3 Combate

Como já foi referido anteriormente, a personagem possuí uma arma, a espada, e esta aparece sempre que o jogador ataca e, caso não realize um ataque durante um período, é arrumada. Isto é possível utilizando uma das funcionalidades das animações do Unity, onde se definir um certo “frame” para chamar um método, podendo até passar atributos.

Na figura 4.13 observa-se que o método “Dash” (selecionado a azul), que trata de deslocar instantaneamente a personagem para um ponto à sua frente, está atribuído a uma certa “frame” quando a animação é executada.

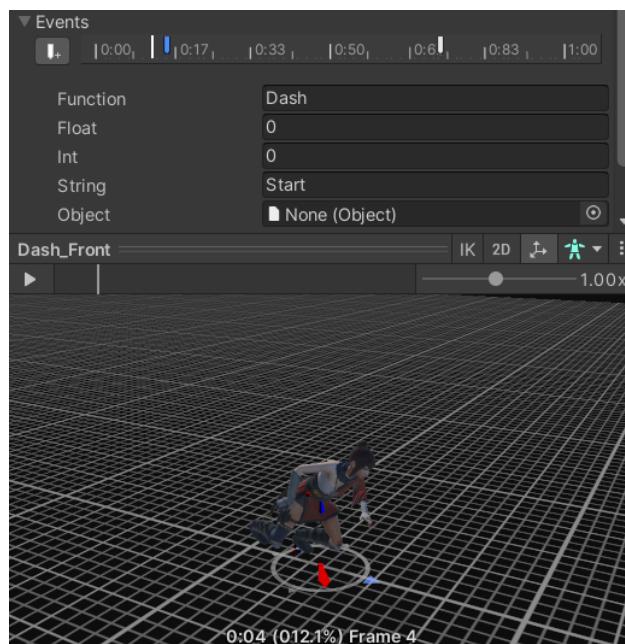


Figura 4.13: Exemplo da chamada de um método durante uma animação

Para causar dano aos inimigos, primeiramente criou-se um “script” adicional que guarda o valor da vida atribuída a um certo objeto e que possui um método público que recebe como argumento o dano a ser descontado sobre esse valor total.

De seguida, utilizou-se o método `OverlapCapsule` que, indicando dois pontos para construir uma cápsula e uma máscara, devolve todos os objetos pertencentes à máscara indicada e que possuem a componente “collider”. Esta máscara (ou “layer”) é aplicada a todos os inimigos. Posteriormente, percorrem-se todos os inimigos detetados, chamando o método público referido anteriormente para lhes causar dano.

Na figura 4.14 apresenta-se a personagem principal, delimitado a verde estão as componentes “collider” que realizam as colisões com o mundo. A esfera branca serve apenas como referência para se saber o tamanho da área de deteção de inimigos.

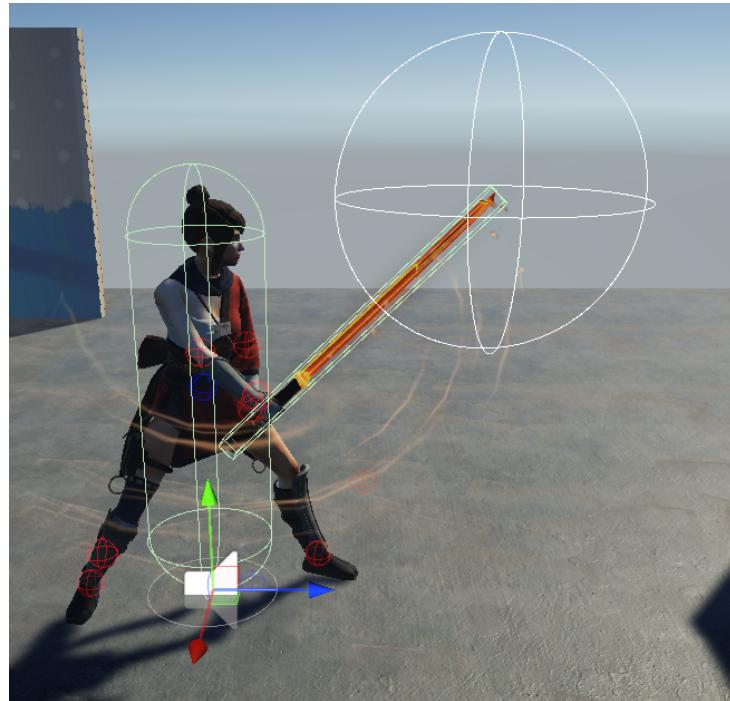


Figura 4.14: Personagem e as suas colisões

Por fim, utiliza-se a mesma funcionalidade referida na figura 4.13 para realizar a deteção de inimigos no instante pretendido, que será a “frame”

onde se considera que a espada devia causar dano.

4.4 Inimigos

Os inimigos foram um dos focos principais deste projeto. Foi decidido que se iriam utilizar árvore de comportamentos, ou “behavior tree”, para a inteligência artificial do jogo, ao invés da tradicional máquina de estados do Unity, o “Animator”.

Também se recorreu à biblioteca externa “Animation Rigging” para se poder criar ou modificar animações com maior facilidade.

Ao objeto do inimigo, atribuiu-se a componente do Unity “Navigation Mesh Agent” que, quando atribuído a um objeto e dando um ponto como argumento, cabe ao motor de jogo calcular o caminho mais próximo para esse ponto e movimentar o objeto.

4.4.1 Animações

Antes de se proceder à codificação da árvore de comportamentos, criou-se primeiro a árvore de animações, com recurso ao “animator”. Assim como na personagem, as setas a branco representam condições de transição na figura 4.15 e as caixas a cinzento e laranja são animações.

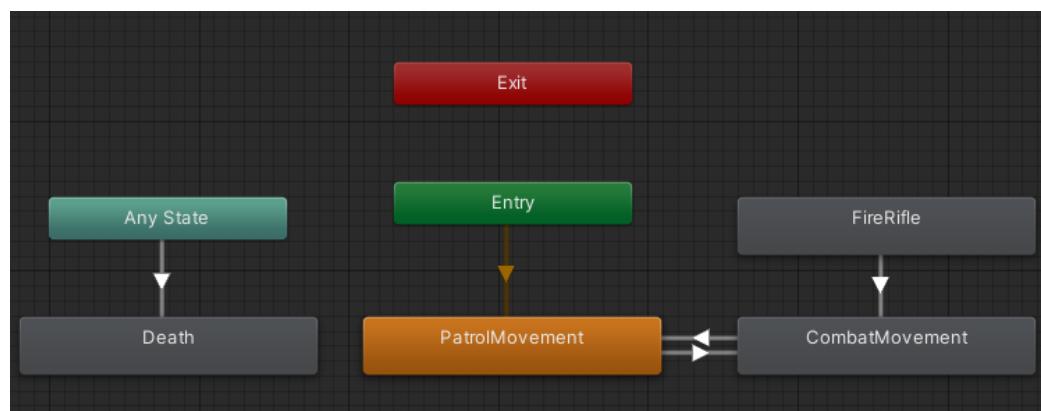


Figura 4.15: Árvore de animações dos inimigos

Como as animações usadas não traziam uma arma a elas associada, foi necessário ir adquirir uma na loja do Unity. No entanto, as mãos na animação

não estavam corretamente posicionadas em relação à arma. Para solucionar este problema, utilizou-se a biblioteca “Animation Rigging” para corrigir este problema e também para criar uma animação para recarregar a arma.

Esta biblioteca destaca-se por se poder sobreescrivver partes de uma animação para ter o comportamento desejado, utilizando a cinemática inversa para criar articulações. Neste caso, criam-se os dois pontos na arma, que serão onde as mãos vão ser colocadas e, dando a conhecer os pontos do braço completo e os pontos criados na arma, pode-se movimentá-la livremente que os braços e as mãos vão corresponder à movimentação dos ossos de um ser humano, ignorando completamente as posições dos braços definidas pela animação.

Na figura 4.16 está o resultado do posicionamento da arma e das mãos na pose de “idle”.



Figura 4.16: Correto posicionamento das mãos na arma utilizando cinemática inversa

Sendo possível sobreescrivver uma animação, também se podem criar animações com esta biblioteca que executam sobre outras. Neste caso, manteve-se a postura corporal do inimigo (animação de “idle”) e mexeu-se apenas no posicionamento da arma e dos pontos atribuídos à mão. Utilizando a aba “Animation”, figura 4.17, podem-se criar “keyframes”, que guardam diversos valores como a posição, rotação entre outros, construindo assim uma

animação para recarregar a arma, que irá afetar apenas os braços e a arma.

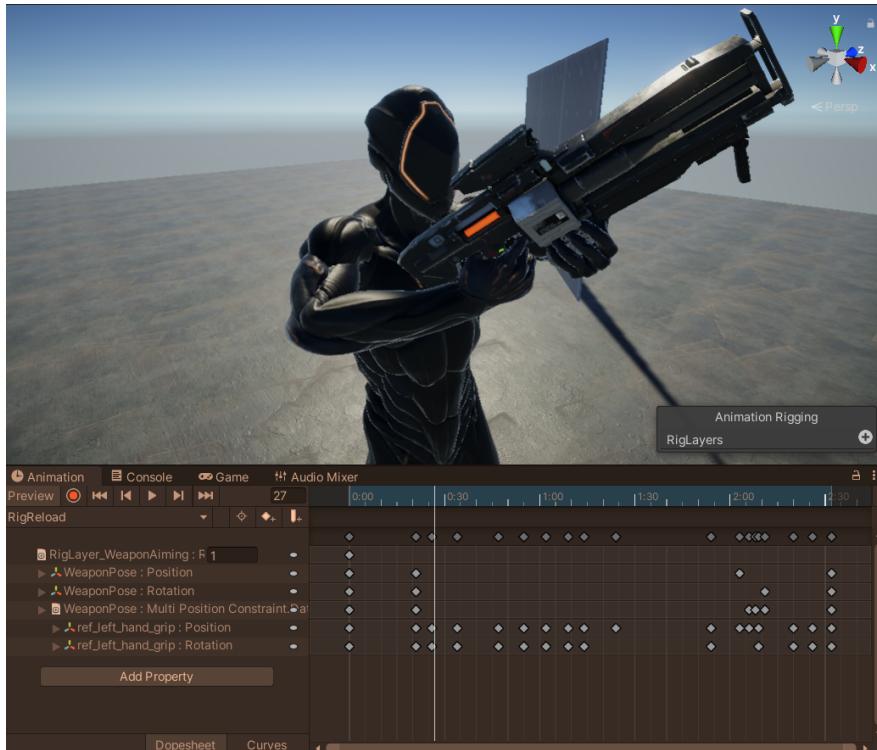


Figura 4.17: Animação de recarregar a arma utilizando cinemática inversa

4.4.2 Implementação da árvore de comportamentos

As “behavior trees” têm como vantagem poder realizar várias tarefas em simultâneo, podendo realizar diferentes ações dependendo do resultado dessas tarefas.

Assim como foi descrito na subsecção 3.2.3, esta árvore é composta por diferentes tipos de nós. Em código, apenas foram necessários criar quatro “scripts” para abranger estes tipos de nó. O primeiro chama-se `Node` e é a classe abstrata que contém o “status” dos nós e o método abstrato `Evaluate` onde serão executadas as tarefas definidas, retornando o seu “status”. O segundo e o terceiro são as classes dos nós de fluxo de controlo, `Selector` e `Sequence`, explicadas na subsecção 3.2.3. O último “script” é apenas uma classe auxiliar, `Inverter`, que apenas inverte o “status” de um nó.

Definidas as classes base, criou-se um diagrama com todos os nós a ser criados, com as tarefas que cada um tem que realizar, na figura 4.18.

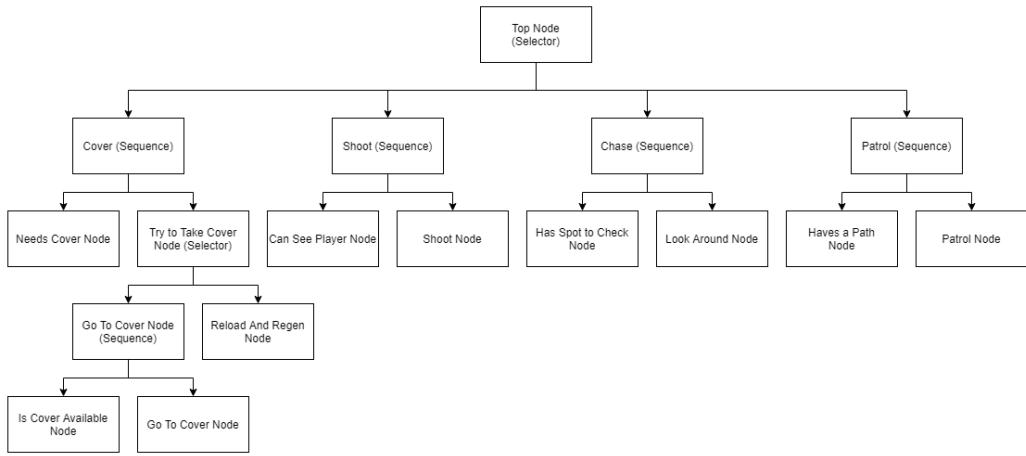


Figura 4.18: Árvore de comportamentos dos inimigos

Cover Nodes

Sendo os nós mais à esquerda os de maior importância, o inimigo irá sempre primeiramente avaliar se necessita de deslocar para uma posição com cobertura. Definiu-se que apenas quando o inimigo está sem balas, precisa de encontrar o lugar mais próximo para se cobrir. Quando este lá chegar, recarrega a arma e restaura a vida.

Shoot Nodes

Caso não precise de cobertura, o inimigo irá avaliar se encontra um alvo.

O inimigo possui um “script” auxiliar que cria campo de visão em formato de cone e deteta todos os objetos com a componente “collider” pertencentes a uma “layer” desejada.

Assim, o jogador é detetado a uma maior se aparecer pela frente, mas se tentar aproximar-se pelas costas e aproximar-se em demasia do inimigo, este também irá ser detetado.

Estando o jogador na linha de fogo, o inimigo irá olhar para ele e disparar, tendo um pequeno valor aleatório a alterar a posição da mira, para simular o coice da arma e para não dificultar em demasia o jogo.

Caso o jogador se esconda, o inimigo guarda a última posição em que o jogador esteve e para de disparar.

Chase Nodes

Não havendo alvo para disparar e tendo em memória a última posição do jogador, o inimigo irá movimentar-se para esse lugar e espera um pouco para garantir que o jogador não volta a aparecer.

Caso não encontre o jogador, a posição em memória é limpa.

Patrol Nodes

Por fim, é executado último ramo de nós caso todos os anteriores não tenham sucesso, ou seja, se o inimigo tiver balas e não se lembrar onde esteve o jogador, irá patrulhar pelo mapa.

Pode-se definir qual o caminho que deve ser seguido durante a patrulha, criando diversos pontos, ou pode-se deixar que o jogo escolha um ponto aleatório no mapa. O inimigo ao chegar a um ponto esperará um pouco antes de seguir para o próximo.

4.5 Interface Gráfica

4.5.1 Menu Inicial

A interface de início, visível na figura 4.19, dispõe de três botões: “Play” para começar o jogo, “Settings” para um menu de opções (figura 4.20) e “Exit” para sair do jogo.



Figura 4.19: Ecrã inicial

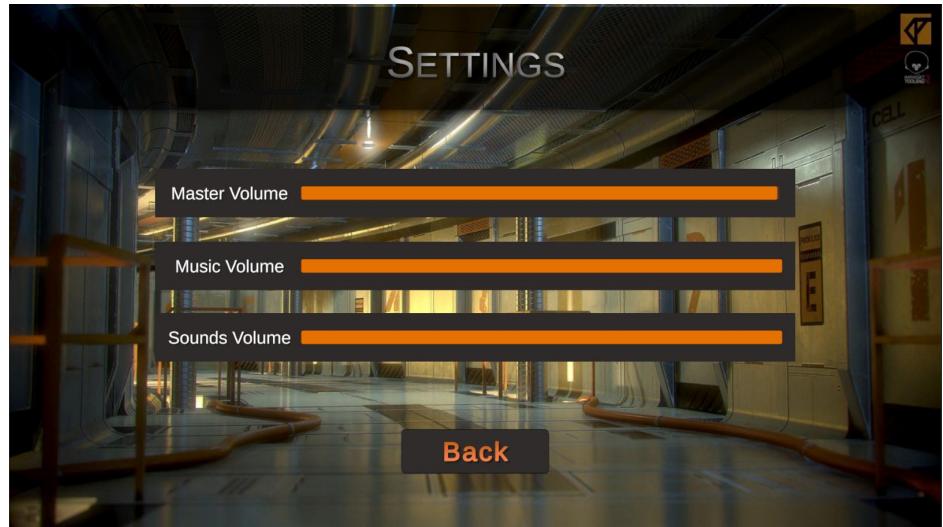


Figura 4.20: Menu de opções

4.5.2 Interface In-game

Durante o jogo, o jogador dispõe da interface visível na figura 4.21, estando sempre à vista a barra de vida (canto superior esquerdo a vermelho) e também um mini-mapa, que mostra a posição do jogador no mapa, onde pode ser consultado ao clicar no “Tab”.



Figura 4.21: Interface do jogador

Para a implementação do mini-mapa, foi mapeada a posição do jogador numa imagem do mapa no “canvas”. É calculado o vetor de escala usando a posição central do mapa e o final, que irá corresponder ao canto direito superior da imagem do “canvas”, sendo depois usado como escala para a localização do jogador no mini-mapa. O código em detalhe pode ser consultado no excerto de código 4.2.

Listagem 4.2: Calculo da posição do jogador no mini-mapa

```

1 if (isShowing)
2 {
3     normalized = Divide(
4         map3dParent.InverseTransformPoint(transform.position), ////
5             calcular a posicao do jogador relativa ao centro do mapa
6             map3dEnd.position - map3dParent.position // vetor de
7             escala
8     );
9     normalized.y = normalized.z; //ao mapear de 3d para 2d a
10    coordenada z deixa de ser relevante
11     mapped = Vector3.Scale(normalized, map2dEnd.localPosition);
12         //aplica a escala ao indicador do jogador no mapa
13     playerInMap.localPosition = mapped;
14 }
```

Durante o jogo, o jogador pode pausar o jogo ao clicar no “Esq”, o que irá mostrar a interface de pausa, figura 4.22. Aí poderá modificar as opções no botão “Settings”, semelhante à figura 4.20 ou sair do jogo no botão “Exit”.

Caso o jogador pretenda resumir o jogo pode clicar no botão “Resume” ou volta a clicar no “Esq”.



Figura 4.22: Menu de pausa

4.6 Áudio

Para adicionar áudio ao jogo, criou-se um “script” comum utilizado por todos os objetos que produzem sons. Cada som está assim associado a um método, podendo ser chamados, por exemplo, durante uma animação, semelhante à figura 4.13.

Para adicionar mais imersão, alguns dos sons, como os passos ou os disparos, quando chamados, é-lhes sempre aplicado um valor de “pitch” aleatório.

Relativamente às barras que controlam o volume do jogo, foram definidas três, como na figura 4.20. A primeira controla o volume geral (“Master”), a segunda controla os sons (“Sound Effects”) e a terceira controla a música (“Music”), como se observa na figura 4.23.

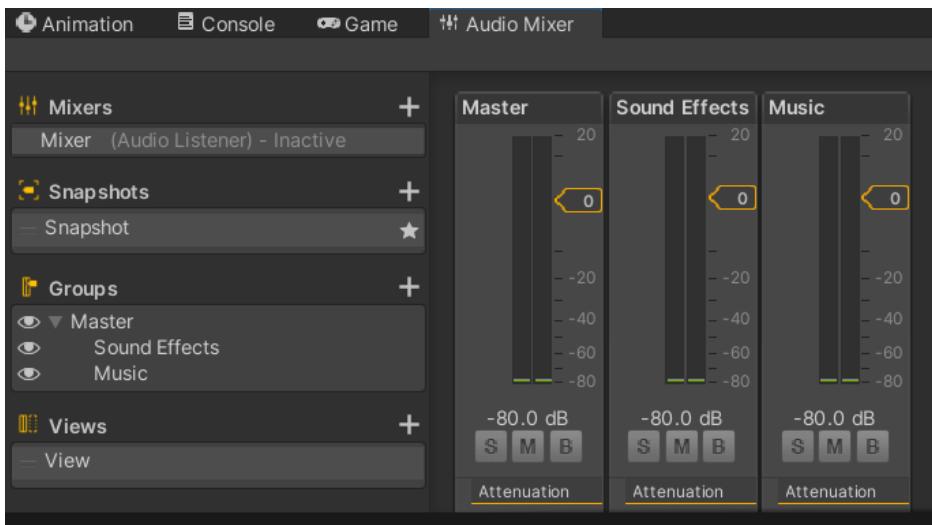


Figura 4.23: Audiomixer

Estas barras também precisam de estar sincronizadas com o menu inicial e “in-game”. Por isso, sempre que se entra numa destas cenas, são lidos os valores do “mixer”, em decibéis, e convertidos para um valor entre 0.001 e 1, para serem aplicados à barra. Ao mexer na barra, o mesmo acontece, mas vice-versa. Esta operação encontra-se no excerto de código 4.3.

Listagem 4.3: Conversão de decibéis para um valor entre 0.001 e 1

```

1 private void Start()
2 {
3     if (mixer.GetFloat(slider, out float value)) // ler o
4         valor pretendido do mixer
5     GetComponent<Slider>().value = Mathf.Pow(10, value /
20f); // converter para o range 0.001-1
}

```

4.7 Diagrama do projeto

Nesta secção é apresentado o diagrama do projeto na figura 4.24, que contém os objetos principais para o funcionamento do jogo, representados pelos retângulos. Relativamente aos “MonoBehaviors”, representados por elipses, são “scripts” associados por uma seta aos seus respetivos “Game Objects”. Estes são responsáveis por controlar todas as características, funções ou movi-

mentos do objeto. Também podem estar associados a um objeto “placeholder” apenas para correr um excerto de código na execução do jogo.

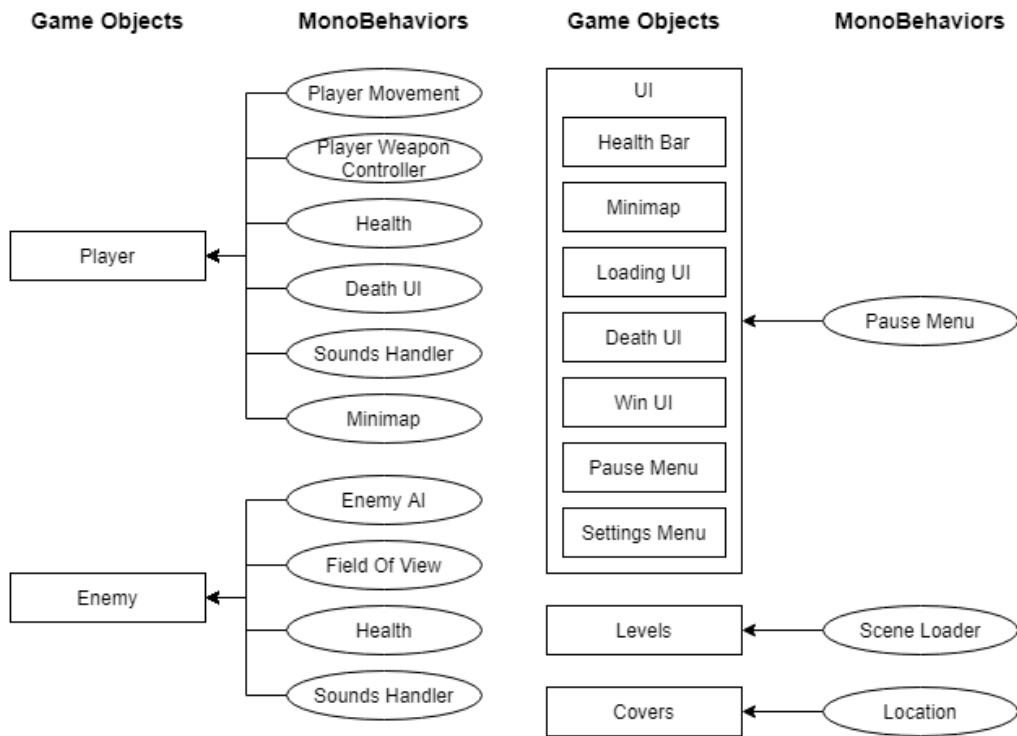


Figura 4.24: Diagrama do projeto

5

Conclusões e Trabalho Futuro

Ao longo do projeto, construiu-se o mapa com a respetiva iluminação e câmaras associadas, seguido da implementação da personagem principal e dos inimigos, com todas as suas animações e controlos. A inteligência artificial foi implementada utilizando “behavior trees”. Por fim, foram adicionados os detalhes finais como os menus, a interface gráfica e o áudio.

Remetendo para os objetivos listados no capítulo 1, este jogo foi criado para a exploração de várias ferramentas, muitas delas que são usadas em jogos tanto recentes, como em marcos da indústria dos últimos anos, servindo de exemplo os já listados no capítulo 2.

Como tal, grande parte dos conceitos apresentados neste projeto, tiveram origem na curiosidade para expandir os conhecimentos base adquiridos na cadeira de Animação em Ambientes Virtuais, pretendendo ter uma janela para os padrões de jogos atuais, numa indústria que está constantemente em rápida evolução, servindo como experiência importante para uma futura carreira.

Relativamente a futuras implementações, visto que o projeto se trata de um jogo, as adições são infinitas, mas especificamente, falta adicionar “cut-scenes” para complementar a história do jogo, dando possibilidade para explorar mais a ferramenta Cinemachine. A adição de “puzzles” pelas salas,

como mini-jogos do tipo “browser”, também é uma adição futura a se fazer.

A

Controlo de Versões

Para fazer controlo de versões utilizou-se o “Unity Collaborate”, visto que o Unity já possuí um por defeito (colocar depois na biografia). Devido à natureza experimental do nosso projeto decidiu-se optar por este em relação aos recomendados, pois, embora ainda recente, já tem bastantes funcionalidades e é de fácil configuração, podendo ser útil em projetos semelhantes no futuro.

O “Unity Collaborate” permite a gestão de versões no editor. Após a instalação, tem-se acesso à janela “Collaborate”, visível na figura A.1. Na aba “History”, pode-se visualizar e restaurar entre versões mais antigas. Na aba “Changes”, figura A.2, pode-se selecionar os arquivos que o utilizador criou ou modificou para dar “upload” das mesmas como uma nova versão, colocando uma breve descrição na caixa “Summary”.

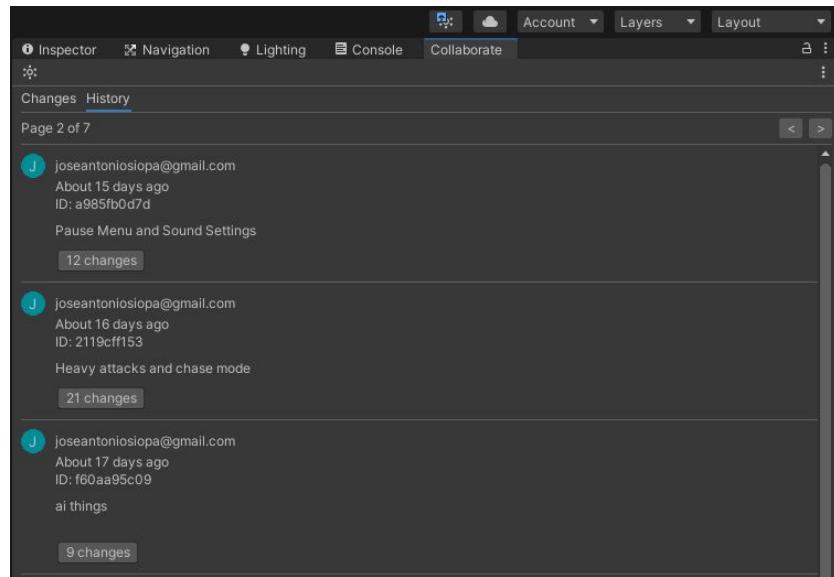


Figura A.1: Controlo de versões

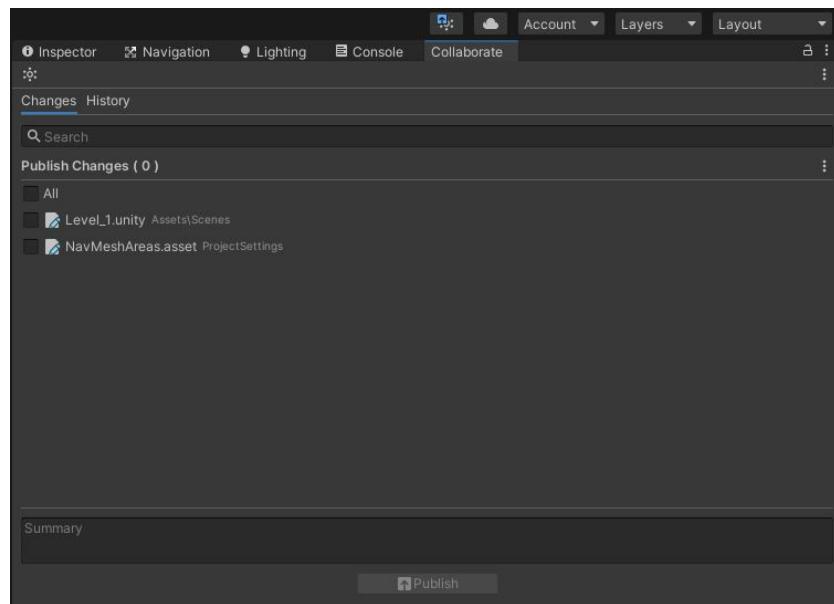


Figura A.2: Upload de nova versão

Posteriormente, foi usado o GitHub para entrega e partilha deste projeto e do respetivo relatório.

Bibliografia

[Adobe, 2021] Adobe (2021). Mixamo. <https://www.mixamo.com/>.

[Koei Tecmo Games Co., LTD, 2017] Koei Tecmo Games Co., LTD (2017). Nioh. https://store.steampowered.com/app/485510/Nioh_Complete_Edition__Complete_Edition/.

[Konami Computer Entertainment Japan, 1998] Konami Computer Entertainment Japan (1998). Metal gear solid.

[One More Level, 2020] One More Level (2020). Ghostrunner. <https://ghostrunnergame.com/>.

[wemakethegame, 2017] wemakethegame (2017). Oriental sword animation. <https://assetstore.unity.com/packages/3d/animations/oriental-sword-animation-71318>.