Universidade da Beira Interior

Departamento de Informática

Nº 61 - 2025: *Treetening*

Elaborado por:

Filipe António Semedo Mesquita
Nº 49701

Orientador:

Professor Abel João Padrão Gomes

30 de junho de 2025

Agradecimentos

Ao longo do desenvolvimento deste projeto, tive a sorte de contar com o apoio e incentivo de várias pessoas, a quem deixo aqui o meu mais sincero agradecimento.

Em primeiro lugar, agradeço ao Professor Doutor Abel João Padrão Gomes, meu orientador, pela orientação rigorosa, disponibilidade constante e valiosos conselhos, que foram determinantes para o desenvolvimento deste trabalho.

Expresso também a minha profunda gratidão aos meus pais, pelo seu amor incondicional, apoio e encorajamento durante todo o processo.

Aos meus amigos e colegas de curso, agradeço pela amizade, partilha de conhecimentos e pelos momentos de descontração que ajudaram a equilibrar esta jornada.

Por fim, agradeço à Universidade da Beira Interior, não apenas pela formação académica de qualidade, mas também pelo ambiente enriquecedor que proporcionou durante este projeto.

Conteúdo

Lista de Tabelas

Lista de Figuras

Lista de Excertos de Código

Acrónimos

FPS *First Person Shooter* - FPS. 1, 3, 19

GDD *Game Design Document* - GDD (*Game Design Document*). 2

GGJ25 *Global Game Jam* 2025. 21

HP *Health Points*. 11–13, 33

HUD *Heads-Up Display*. iv, vii, viii, 41, 42, 47

IA Inteligência Artificial. 21, 22

UE5 *Unreal Engine* 5. 21

UI *User Interface*. 35, 46, 47

Glossário

.cs Extensão de ficheiros usada para código-fonte escrito em C#, uma linguagem de programação orientada a objetos desenvolvida pela Microsoft, comummente utilizada em aplicações Windows, jogos com Unity e desenvolvimento web. 22, 23

Scriptable Object Classe da Unity que permite criar objetos de dados independentes de instâncias de componentes ou cenas, facilitando a gestão e reutilização de configurações, variáveis e conteúdos partilhados entre vários objetos do jogo. 23, 30

animator Componente da Unity responsável por controlar animações de objetos, permitindo gerir estados, transições e *blends* entre várias animações para criar movimentos fluidos e dinâmicos em personagens e objetos. vii, 28, 30

asset Recurso digital utilizado na criação de videojogos, como modelos 3D, *sprites* 2D, texturas, sons, animações ou interfaces, que compõem os elementos visuais e auditivos do jogo. 2, 4, 21

blueprints Sistema de programação visual da Unreal Engine 5 que permite criar lógica de jogo e comportamentos sem necessidade de escrever código. Utiliza nós e ligações gráficas para facilitar o desenvolvimento, ideal para designers e programadores. 21

canvas Elemento fundamental do sistema UI da Unity que funciona como uma superfície para desenhar e organizar elementos gráficos, como botões, textos e imagens, permitindo criar interfaces visuais interativas no jogo. 38, 40–43

engine Plataforma de software utilizada para desenvolver videojogos, que fornece ferramentas e funcionalidades como motor gráfico, física, áudio, *scripting* e gestão de recursos. 1, 21

game object Elemento fundamental na Unity que representa qualquer objeto na cena, podendo conter componentes que definem o seu comportamento, aparência e interação no jogo. 40

hitbox Área invisível definida em personagens ou objetos de um videojogo que determina onde podem ser detetados impactos, colisões ou ataques, usada para calcular interações e danos durante o jogo. 27, 29, 34

low poly Estilo de modelação 3D caracterizado por um número reduzido de polígonos, resultando em modelos com aparência simples e angulosa. É frequentemente utilizado para otimizar o desempenho e criar uma estética visual distinta. 4, 7

overlap Situação em videojogos onde duas ou mais áreas, objetos ou colisores se sobrepõem ou intersectam, podendo desencadear eventos como colisões, deteção de contacto ou outras interações no jogo. 35

prefab Objeto pré-configurado na Unity que pode ser reutilizado e instanciado várias vezes em diferentes cenas. Permite guardar hierarquias de objetos, componentes e configurações para facilitar a criação e manutenção de elementos repetidos no jogo. iv, vii, 23, 27, 29, 30, 33, 34, 36–39, 46, 47

skybox Técnica gráfica utilizada para simular o céu ou o ambiente envolvente de uma cena tridimensional, normalmente através de uma textura aplicada a um cubo ou esfera que rodeia a cena. viii, 45

sprite Imagem ou animação 2D usada para representar objetos visuais num videojogo, como personagens, itens ou efeitos.. vii, 4, 10, 12, 13, 15, 16, 23, 30, 38, 42

tag Etiqueta atribuída a objetos na Unity para os identificar e agrupar, facilitando a deteção, organização e manipulação de grupos específicos de objetos durante o jogo. 35, 39

corrotina Método especial em Unity que permite executar código de forma assíncrona, pausando e retomando a execução em vários *frames*. Muito usada para criar temporizações, animações e sequências sem bloquear o fluxo principal do jogo. 38

FPS Género de videojogo conhecido como "*First-Person Shooter*", com foco em em que o jogador vê a ação através dos olhos do protagonista. xi

GDD (*Game Design Document*) Documento que descreve detalhadamente a visão, mecânicas, história, personagens, estética e requisitos técnicos de um videojogo, servindo como guia de referência durante o desenvolvimento. xi

jogos AAA Videojogos com elevado orçamento de desenvolvimento e *marketing*, produzidos por grandes estúdios. Caracterizam-se por alta qualidade gráfica, complexidade técnica e grande ambição em termos de conteúdo e alcance no mercado. 21

Capítulo

1

Introdução

1.1 Enquadramento e Objetivo

Este projeto tem como objetivo o desenvolvimento de *Treetening*, um videojogo 3D em primeira pessoa (um FPS) *singleplayer*, com um *loop* de *gameplay* centrado na recolha de recursos e na melhoria de equipamento. O jogador utiliza armas para abater árvores e recolher as suas raízes, que contêm cristais valiosos. Estes concedem dinheiro, permitindo ao jogador investir em novas armas e melhorias, dando assim continuidade ao ciclo de progressão do jogo.

1.2 Motivação

A escolha deste projeto partiu de um gosto e interesse pessoal pela área do *game development* e computação gráfica.

A ideia do jogo surgiu cerca de um ano antes da realização do projeto, com o avistamento de uma árvore que, devido a uma tempestade, caiu e deixou a sua raiz exposta, o que despertou a ideia de fazer um videojogo que se relacionasse com isso.

1.3 Calendarização

O desenvolvimento de um jogo passa por várias fases, sendo muitas destas extensas e por vezes até mesmo sobrepostas.

Assim, o desenvolvimento deste projeto seguiu a seguinte calendarização:

- **Fase 1** - Planeamento e Setup do Projeto (23 de fevereiro a 8 de março):
  - Definição do conceito do jogo, mecânicas, armas e sementes;
  - Instalação da *engine* (Unity) e criação do projeto;
  - Criação de um repositório no GitHub para controlo de versões e backups, caso necessário;
  - *Import* de um *first person controller* disponibilizado na *Unity Asset Store*.

- **Fase 2** - Desenvolvimento e Implementação (9 de março a 20 de junho):
  - Criação de *asset*s 3D e 2D;
  - Desenvolvimento e implementação de mecânicas e *features*.
- **Fase 3** - Relatório (GDD), Adição final de *Features* e Entrega (21 de junho a 30 de junho):
  - Escrita do realtório (GDD);
  - Correção de bugs e implementação de algumas features em falta;
  - Entrega do projeto.

1.4 Organização do Documento

Este documento é composto por seis capítulos, sendo estes:

- **Capítulo 1** - Introdução:

  Apresenta o contexto do projeto, com uma descrição geral do mesmo e da motivação que o originou. Inclui ainda a calendarização seguida para o seu desenvolvimento e a estrutura adotada ao longo deste relatório.

- **Capítulo 2** - Descrição do Projeto:

  Faz uma descrição do que é o *Treetening*. Fala sobre a visão geral do jogo, qual é a sua proposta, os seus diferenciais e como é o seu estilo visual e de *gameplay*.

- **Capítulo 3** - Audiência, Plataforma e Marketing:

  Aborda o público-alvo ao qual o jogo será mais apelativo, a plataforma sobre a qual foi desenvolvido, aquelas onde estará disponível e o plano de marketing e disponibilidade do jogo.

- **Capítulo 4** - *Gameplay*, Progressão e Mecânicas:

  Explora as principais mecânicas que *Treetening* oferece e de que forma estas influenciam a experiência do jogador.

- **Capítulo 5** - Desenvolvimento e Implementação:

  Aborda a criação de *asset*s e a implementação de mecânicas que levaram ao resultado final.

- **Capítulo 6** - Conclusão e Trabalho Futuro:

  Reflete sobre o resultado final do projeto e trabalhos que podem vir a ser realizados para melhorar o mesmo.

Capítulo

2

Descrição do Projeto

Iremos então definir com mais detalhe o que é o *Treetening*. No decorrer deste capítulo vamos falar sobre a visão geral do jogo, qual é a sua proposta, os seus diferenciais e como é o seu estilo visual e de *gameplay*.

2.1 Conceito do Jogo

Treetening é um FPS *singleplayer* casual onde o jogador tem acesso a uma variedade de armas que usa para derrubar árvores e recolher as suas raízes, quem contêm um cristal valioso. Ao fazê-lo o jogador ganha dinheiro que tem de gerir entre melhorar as armas que já possui, expandir o seu arsenal ou comprar mais sementes para plantar mais árvores e assim continuar a ganhar dinheiro.

2.1.1 Diferenciais

Algo que diferencia este jogo de outros do mesmo género é a plataforma onde está disponível. Uma grande parte dos jogos casuais, com jogabilidade simples e foco em progressão satisfatória, está concentrada no mercado *mobile* (telemóveis, *tablets*, etc.). Segundo Nick Galov, em "Key Mobile Gaming Statistics: How Many People Play in 2023?" [4], estes jogos representam também uma parte significativa dos *downloads* nessas plataformas. Isto mostra que há espaço para este tipo de jogo noutras plataformas, especialmente tratando-se de um jogo gratuito, sem publicidade nem qualquer tipo de monetização.

2.1.2 Objetivos

O principal objetivo do jogador é desbloquear todas as armas disponíveis no jogo e melhorar os seus atributos o máximo possível. Para isso, deve gerir o dinheiro disponível entre a aquisição e melhoria de armas e a plantação de sementes, já que apenas ao plantar novas árvores poderá garantir uma fonte de rendimento.

2.1.3 Género

Treetening é um FPS *singleplayer* casual, ou seja, um jogo em primeira pessoa, com foco total numa experiência individual que consiste num *loop* de *gameplay* composto pela recolha de recursos e melhoria de equipamentos.

2.1.4 Estilo Visual e Experiência de *Gameplay*

2.1.4.1 Estilo Visual

Os elementos 3D do jogo seguem uma estética *low poly*, algo muito parecido com o estilo visual de *Grow Home* (ver figura 2.1), com modelos com poucos polígonos, cores vivas e texturas simples. Este estilo não dá apenas um aspeto descontraído e característico ao jogo como também proporciona um desenvolvimento mais rápido, uma vez que simplifica bastante a criação de *asset*s 3D.

Figura 2.1: *Grow Home*

No que toca a elementos 2D, como interfaces e *sprite*s, estes têm um estilo simples e minimalista. Com destaque para as *sprite*s das armas, que são desenhos 2D feitos digitalmente (figura 2.2) com base nos modelos 3D que representam (figura 2.3).

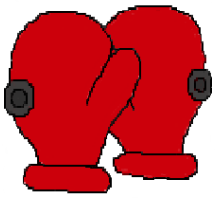

Figura 2.2: *Sprite* das *Rocket Gloves*

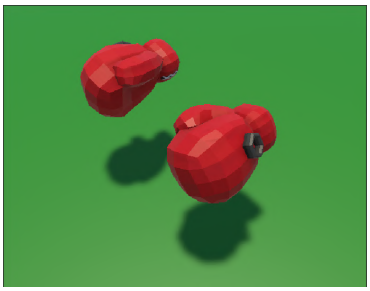

Figura 2.3: Modelo 3D das *Rocket Gloves*

2.1.4.2 Experiência de *Gameplay*

Treetening propõe uma experiência de *gameplay* agradável e divertida de dominar, com cada arma tendo um estilo de *gameplay* e forma de utilização únicos.

Uma boa comparação seria *Overwatch*, onde cada herói dispõe de uma arma completamente diferente, permitindo uma grande variedade de estilos de jogo (ver figuras 2.4 e 2.5).

Figura 2.4: *Overwatch*: arma da Tracer. Fonte: [1]

Figura 2.5: *Overwatch*: arma da Mei. Fonte: [2]

Capítulo

3

Audiência, Plataforma e Marketing

Este capítulo aborda o público-alvo ao qual o *Treetening* será mais apelativo, a plataforma sobre a qual foi desenvolvido e estará disponível e o plano de marketing e disponibilidade do jogo.

3.1 Público-Alvo

O público-alvo de *Treetening* são adolescentes e jovens adultos (13 a 25 anos) que procuram uma experiência de jogo casual, divertida e acessível, ideal para os seus momentos de lazer.

Este tipo de jogador valoriza uma *gameplay* envolvente e fluida, com mecânicas simples mas gratificantes, que lhes permita descontrair, sem a pressão dos jogos altamente competitivos ou complexos, nem a exigência de atenção que os jogos com forte componente narrativa normalmente requerem.

O estilo visual apelativo em *low poly* e a ênfase na diversidade de armas e no ciclo de ação e recompensa — cortar árvores, recolher cristais, melhorar o arsenal e replantar — tornam o jogo especialmente atrativo para quem privilegia experiências de jogo centradas na *gameplay*, com um toque de originalidade e humor.

3.1.1 Personas

A seguir, apresentam-se duas personas que ilustram os principais tipos de público-alvo.

- **José Vasco, 23 anos:**
  - O José é um recém licenciado em Engenharia Informática, que entrou no mercado de trabalho e que cresceu a jogar videojogos nos seus tempos livres. No entanto, com as responsabilidades impostas pela vida adulta ele tem pouco tempo para jogar. Assim, durante a semana ele põe de parte jogos competitivos, para evitar stress a mais depois de um longo dia de trabalho, ou com foco em narrativa, para que não se sinta tentado a continuar a jogar fora do horário devido por estar muito imerso na história e ansiar por saber mais sobre esta. Assim, sem ser nos fins de semana ele procura jogos casuais, simples, mas cativantes, que o entretenham.

- **Joana Fernandes, 15 anos:**
  - A Joana está no 10º ano, no ensino secundário, ela sempre gostou de jogar videojogos, ao contrário do seu circulo de amigos. No entanto eles todos gostam de conversar online através do Discord, assim a Joana gosta de aproveitar os momentos em que está em chamada com os seus amigos para jogar um jogo mais casual, sem necessidade de prestar atenção a diálogos ou *cutscenes*, mas com uma *gameplay* fluída e recompensadora que a mantém investida no jogo. Desta forma ela pode jogar algo enquanto conversa com os seus amigos sem que uma coisa atrapalhe a outra.

3.2 Plataforma

O jogo foi desenvolvido num computador com as especificações que podemos ver na tabela abaixo (tabela 3.1):

Componente	**Especificação**
Sistema Operativo	Windows 11 (64 bits) versão 24H2
CPU	Intel Core i7-12700H
RAM	16GB
GPU	NVIDIA GeForce RTX 3050 Laptop (VRAM 16GB)

Tabela 3.1: Especificações do computador

Não há garantia de que o jogo corra como esperado num computador com *hardware* inferior a este. No entanto, um requisito mínimo para correr a *build* do jogo é ter um sistema operativo Windows na versão Windows 10 version 21H1 (build 19043) ou superior. Esta é a versão mínima recomendada em *Unity Documentation*.

3.3 Marketing

O jogo foi desenvolvido no âmbito da unidade curricular de Projeto, assim este servirá, pelo menos inicialmente, apenas como prática na área do *game development* e conteúdo para um portefólio pessoal, sem qualquer divulgação dedicada na *Internet* ou redes sociais.

Contudo, uma versão atualizada do *Treetening* estará disponível no *itch.io*, de forma gratuita para qualquer pessoa que tenha interesse neste.

Capítulo

4

Gameplay, Progressão e Mecânicas

Quando falamos de *gameplay* e mecânicas, referimo-nos à experiência proporcionada ao jogador pelas diferentes *features* disponíveis no jogo, por outras palavras, o que é possível fazer dentro do jogo e qual o impacto dessas ações.

Ao longo deste capítulo, vamos explorar as principais mecânicas que *Treetening* oferece e de que forma estas influenciam a experiência do jogador.

4.1 *Gameplay* e Progressão

Como mencionado anteriormente, *Treetening* oferece uma experiência de *gameplay* agradável e divertida de dominar, em que cada arma apresenta um estilo e forma de utilização únicos e uma progressão gradual mas recompensadora.

4.1.1 Início do Jogo

Quando o jogo começa, o jogador é colocado no centro de uma ilha flutuante, sem dinheiro e com uma arma incial, a *Hatchet* (ver figura 4.1). Espalhados pela ilha é possível ver uma loja, onde serão feitas as compras e os *upgrades*, uma máquina onde devem ser colocadas sementes para que estas sejam plantadas e algumas árvores já prontas para serem abatidas. O jogador deve então derrubar as árvores para obter dinheiro e começar assim a progredir no jogo, reinvestindo os seus ganhos na loja, onde se encontram todas as sementes e armas disponíveis no jogo.

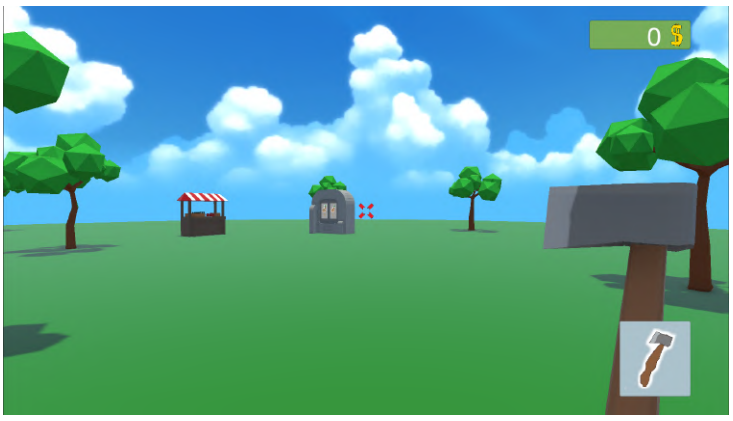

Figura 4.1: *Setup* inicial do jogo

Este *setup* inicial foi desenhado de forma a que o jogador compreenda intuitivamente qual o caminho a seguir para progredir no jogo. Ao começar sem dinheiro, apenas com uma machadinha na mão e algumas árvores à sua frente, este deduz que deve usar a arma para derrubar as árvores. Para além disso, ao ver as *sprite*s das sementes no painel do *planter* com o número 0 em baixo e também na loja, ele percebe então que deve comprar estas para depois as depositar na máquina.

4.1.2 Progressão

4.1.2.1 Sistema de Preços

Conforme vai jogando, derrubando árvores, recolhendo as suas raízes e fazendo compras, o jogador deparar-se-á com o sistema de preços da loja. Este pode ser resumido pela seguinte frase:

"Dependendo do produto adquirido, o preço da próxima compra pode ser ou igual ou superior o preço atual."

Isto significa que temos **dois** casos possíveis:

1. **Compra de Sementes** - o preço é sempre constante, assim como o valor da raiz da árvore que a semente origina, o que garante que valha sempre a pena a compra destas e haja progressão no jogo.
   (É importante notar que o valor da raiz da árvore é sempre maior do que o preço da semente.)
2. **Compra de Armas e *Upgrades*** - todas as armas e *upgrades* partilham o mesmo preço, no entanto, sempre que é feita uma compra, este sofre um aumento de 1.2%, o que faz com que o jogador tenha de definir a sua prioridade em relação ao que desbloquear agora e o que "deixar para depois", pois a próxima compra será sempre mais cara do que a atual.

O preço base para a compra de armas e *Upgrades* é **50$**.

4.1.2.2 Itens na Loja

A loja tem disponíveis todas as sementes e armas presentes no jogo (ver figura 4.2).

Figura 4.2: Menu da Loja

Lista de Sementes:

- ***Mahogany Seed*** - semente mais barata, com tempo de crescimento menor, origina uma árvore com menos HP e a sua raiz não é tão valiosa (ver figuras 4.3 e 4.4).

  Parâmetros da semente:

  - Preço -> 3$;
  - Tempo de crescimento -> 5s;
  - Árvore -> *Mahogany Tree*.

  Parâmetros da árvore:

  - HP -> 5;
  - Valor da raiz -> 5$.

Figura 4.3: *Mahogany Seed*

Figura 4.4: *Mahogany Tree*

- ***Pine Seed*** - semente mais cara, com tempo de crescimento maior, origina uma árvore com mais HP, mas com uma raiz com mais valor (ver figuras 4.5 e 4.6).

  Parâmetros da semente:

  - Preço -> 7$;
  - Tempo de crescimento -> 10s;
  - Árvore -> *Pine Tree*.

  Parâmetros da árvore:

  - HP -> 10;
  - Valor da raiz -> 10$.

Figura 4.5: *Pine Seed*

Figura 4.6: *Pine Tree*

Lista de Armas:

- ***Hatchet*** - uma machadinha que dá pouco dano mas que cumpre bem o seu trabalho (ver figuras 4.7 e 4.8).

  Esta é a arma inicial do jogo, logo esta está desbloqueada desde o início.

  Parâmetros da arma:

  - Nome do primeiro atributo -> *Attack Damage*;
  - Valor base do primeiro atributo -> 1;
  - Nome do segundo atributo -> *Attack Speed*;
  - Valor base do segundo atributo -> 1;

Figura 4.7: *Hatchet Sprite*

Figura 4.8: Modelo 3D da *Hatchet*

- ***Rocket Gloves*** - um par de luvas de box com foguetes na parte de trás com as quais o jogador dá socos poderosos (ver figuras 4.9 e 4.10).

  Tem de ser desbloqueada antes de ser utilizada.

  Parâmetros da arma:

  – Nome do primeiro atributo -> *Attack Damage*;
  – Valor base do primeiro atributo -> 3;
  – Nome do segundo atributo -> *Attack Speed*;
  – Valor base do segundo atributo -> 1;

Figura 4.9: *Rocket Gloves Sprite*

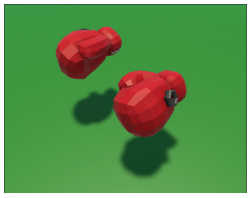

Figura 4.10: Modelo 3D das *Rocket Gloves*

4.2 Mecânicas e Controlos

4.2.1 Mecânicas

4.2.1.1 Ataque, Abate de Árvores e Recolha de Raízes

O jogador pode usar a arma que tem equipada para atacar as árvores, ao fazê-lo irá diminuir o HP delas consoante o dano causado pela arma. Quando o HP da árvore fica menor ou igual a 0 esta "morre" (ver figura 4.11). Então a árvore cai e a sua raiz fica exposta para que o jogador a recolha interagindo com ela, ganhando assim dinheiro que depois pode gastar na loja.

Figura 4.11: Árvore Caída

4.2.1.2 Loja

O jogador pode gastar o dinheiro, que ganha ao recolher raízes, na loja (ver figura 4.12). Para tal deve interagir com a mesma, o que abrirá o menu da loja (ver figura 4.13).

Este menu mostra a lista de todas as sementes e armas disponíveis, e a quantidade de dinheiro que o jogador possui.

Figura 4.12: Modelo 3D da loja

Figura 4.13: Menu da Loja

Ao selecionar um item, aparece o painel com informações sobre o mesmo e o/os botão/botões de compra.

Caso uma **semente** seja selecionada, vemos as seguintes informações (ver figura 4.14):

- *Sprite* da semente;
- Nome da semente;
- Tempo de crescimento (*Growth Time*);
- Valor da raiz (*Root Value*);
- Preço (*Seed Price*);
- Quantidade possuída no inventário (*Owned Seeds*).

Figura 4.14: Informação de uma semente

Se selecionarmos uma **arma**, há duas possibilidades.

Se a arma estiver **desbloqueada**, vemos (ver figura 4.15):

- *Sprite* da arma;
- Nome da arma;
- Preço do *upgrade* (*Upgrade Price*);
- Nome do primeiro atributo;
- Nível do primeiro atributo;
- Nome do segundo atributo.
- Nível do segundo atributo;

Figura 4.15: Informação de uma arma desbloqueada

Caso ainda esteja **bloqueada**, vemos (ver figura 4.16):

- *Sprite* da arma;
- Nome da arma;
- Preço do desbloqueio (*Unlock Price*).

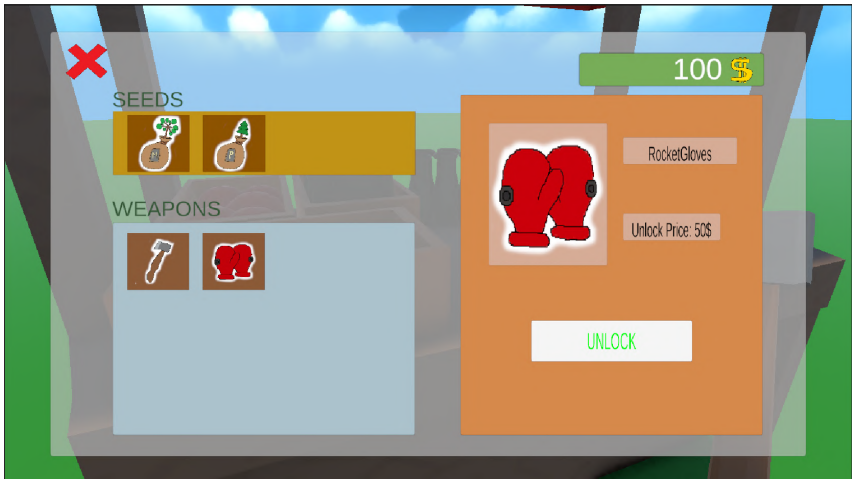

Figura 4.16: Informação de uma arma bloqueada

Relembrando que o preço de aquisição de uma semente é constante mas o custo de aquisição ou melhoria de armas aumenta a cada compra feita.

4.2.1.3 Plantação de Sementes

Para plantar as sementes que compra na loja, o jogador deve interagir com o *Planter* (ver figura 4.17), uma máquina que recebe sementes e as planta num local aleatório da ilha.

Figura 4.17: *Planter*

O *Planter* dá prioridade àquelas que têm menor tempo de crescimento e apenas planta uma semente de cada vez, ou seja, enquanto a semente não crescer a máquina não começa a plantar outra.

Na frente do *Planter* temos um painel que mostra a quantidade de cada semente que está dentro dele (ver figura 4.18).

Figura 4.18: Painel do *Planter*

É também relevante referir que a ideia para o funcionamento do *planter* foi inspirada no *auto-feeder* do jogo *Slime Rancher* (ver figura 4.19) em que itens são colocados dentro da máquina e este gere-os e dispensa-os automaticamente.

Figura 4.19: *Auto-feeder* do *Slime Rancher*. Fonte: [3]

4.2.2 Controlos

Os controlos do jogo seguem os *standarts* dos jogos FPS, movimentação com "WASD", controlo da câmera é feito com o rato e o ataque/disparo é feito clicando no *mouse 1*. No entanto temos também alguns outros controlos que também devem ser mencionados. Podemos ver estes na tabela abaixo (tabela 4.1):

Tecla/Controlo	**Ação**
W	Andar para a frente
A	Andar para a esquerda
S	Andar para trás
D	Andar para a direita
Espaço	Saltar
Shift esquerdo	Correr
Rato	Mover a câmera
Mouse 1/Botão esquerdo do rato	Atacar/Disparar e Selecionar (dentro de menus)
Scroll Wheel/Roda do rato	Trocar entre armas desbloqueadas
F	Interagir com objetos do jogo
ESC	Abrir o menu de pausa

Tabela 4.1: Controlos do jogo

Capítulo

5

Desenvolvimento e Implementação

Após termos definido o conceito do jogo e tudo o que este engloba, desde público alvo e estética a mecânicas e progressão, vamos então falar sobre o desenvolvimento do mesmo.

Neste capítulo iremos abordar a criação de *asset*s e a implementação de mecânicas que levaram ao resultado final.

5.1 Escolhas Iniciais, Tecnologias e Ferramentas Utilizadas

Em termos de ferramentas, a escolha que mais impacta o desenvolvimento de um jogo é com certeza o motor gráfico (*engine*), então, primeiramente foi feita uma pesquisa para decidir qual seria utilizado. A escolha era entre UE5 e Unity. As duas tinham as suas vantagens, no entanto a UE5, de inicio, era mais apelativa, devido a já ter tido contacto com a *engine* no curso "Unreal Engine 5 para iniciantes", ainda ter acesso aos materiais fornecidos nas aulas do referido curso e também devido ao facto de a UE5 ser bastante utilizada por grandes estúdios presentes no mercado de jogos AAA, como a CD Projekt Red ou a Epic Games. No entanto, alguns fatores levaram às escolha da Unity como *engine* para este projeto, nomeadamente:

- O curso referido anteriormente apenas fez uso do sistema de programação com *blueprints*, ou seja, não havia qualquer experiência prévia com C++ na UE5. Para além disso, a integração da linguagem na *engine* mostrou-se um tanto diferente e mais confusa do que era esperado, o que levaria a um desenvolvimento mais lento e difícil;
- Mesmo tendo acesso aos materiais fornecidos no curso de UE5, a interface e a dinâmica de funcionamento da Unity encontrava-se muito mais avivada devido ao uso da mesma na GGJ25 pouco tempo antes do início do desenvolvimento do projeto.

Assim, devido à maior familiaridade com a *engine* e tendo em conta o tempo disponível para a concretização do projeto, a escolha final foi a Unity, o que proporcionou um desenvolvimento mais ágil e eficiente.

Em termos de outras ferramentas utilizadas, como o Blender para modelação 3D, Paint e Gimp 3 para arte 2D, o VS Code para programação e o GitHub para controlo de versões e *backups*, a escolha destas deve-se apenas ao facto de já existir experiência prévia com a utilização das mesmas.

Para além disso, também foi utilizada IA, nomeadamente o ChatGPT-3.5 e o ChatGPT-4-turbo, como auxílio em algumas partes criativas do projeto e no esclarecimento de dúvidas técnicas ao longo do desenvolvimento.

Face a isto, de forma resumida, no desenvolvimento deste projeto foram utilizadas as seguintes tecnologias e ferramentas:

- **Motor Gráfico**:
  - Unity Engine - versão do editor 6000.0.40f1.
- **Programação**:
  - VS Code - versão 1.101.1;
  - C#.
- **Modelação 3D e Texturas**:
  - Blender - versão 4.4.3.
- **Arte 2D**:
  - Paint - versão 11.2504.531.0;
  - Gimp 3 - versão 3.4.0.
- **Controlo de Versões e Backups**:
  - GitHub e GitHub desktop.
- **IA**:
  - ChatGPT-3.5 e ChatGPT-4-turbo.

5.2 Desenvolvimento

O desenvolvimento do jogo foi pensado desde o início para facilitar a expansão e manutenção do projeto a longo prazo. Procurou-se estruturar o código e os sistemas de forma clara e reutilizável, permitindo, por exemplo, a adição de novas armas, árvores ou funcionalidades sem necessidade de alterações profundas na base do código existente. Nesta secção serão abordados, em detalhe, os principais componentes implementados, tais como as armas, árvores e sementes, a loja, o *planter*, a cena principal e a interface de utilizador.

5.2.1 Armas

5.2.1.1 Base Geral

Todas as armas seguem o mesmo método de implementação, sendo que, dos 6 ficheiros ".cs" que todas as armas utilizam, apenas 3 são únicos para cada arma, os restantes têm apenas uma instância que é partilhada por todas.

Os 6 ficheiros ".cs" mencionados são os seguintes:

- **WeaponData.cs** (exerto de código 5.1)

  Ficheiro que estende a classe *ScriptableObject*. Este permite a criação de *Scriptable Objects* diretamente a partir do editor, que são objetos que guardam as **informações** base de cada arma.

  Sendo estas:

  - ID da arma (*weaponID*);
  - Nome da arma (*weaponName*);
  - *Prefab* da arma (*weaponPrefab*);
  - Valor base do primeiro atributo da arma (*baseAttribute1*);
  - Valor base do segundo atributo da arma (*baseAttribute2*);
  - Nome do primeiro atributo (*attribute1Name*);
  - Nome do segundo atributo (*attribute2Name*);
  - *Sprite* da arma (*weaponSprite*);

```
[CreateAssetMenu(fileName = "WeaponData", menuName = "Scriptable
    Objects/WeaponData")]
public class WeaponData : ScriptableObject
{
    public string weaponId;
    public string weaponName;
    public GameObject weaponPrefab;

    public float baseAttribute1;
    public float baseAttribute2;

    public string attribute1Name;
    public string attribute2Name;

    public Sprite weaponSprite;
}
```

Excerto de Código 5.1: *WeaponData.cs*

- **WeaponInstance.cs** (exerto de código 5.2)

  Este tem o atributo [*Serializable*], o que permite que instâncias de *WeaponInstance* sejam exibidas no Editor da Unity e, no futuro, caso seja implementado um sistema de *saves*, permite que sejam convertidas para JSON. Um objeto deste tipo é criado sempre que uma arma é desbloqueada e fica armazenado na lista de armas desbloqueadas em *PlayerInventory.cs*.

  Um objeto do tipo *WeaponInstance* tem o ID da arma ao qual está associado e guarda o nível das habilidades da mesma. Quando é necessário saber qual o valor atual de um atributo, por exemplo o dano de uma arma, este *script* tem os métodos para calcular e retornar esse valor com base no nível do atributo -> $valorAtual = valorBase \times (1 + 0.1 \times nivel)$.

```
[Serializable]
public class WeaponInstance
{
    public string weaponId;
    public int attribute1Level;
    public int attribute2Level;

    public float GetAttribute1Value(WeaponData data)
    {
        return data.baseAttribute1 * (1 + 0.1f * attribute1Level);
    }

    public float GetAttribute2Value(WeaponData data)
    {
        return data.baseAttribute2 * (1 + 0.1f * attribute2Level);
    }
}
```

Excerto de Código 5.2: *WeaponInstance.cs*

- **PlayerInventory.cs** (exerto de código 5.3)

  O *PlayerInventory.cs* é responsável por armazenar todas as armas, sementes e dinheiro que o jogador possui, mas por enquanto vamos abordar apenas o que é relativo a armas, mais à frente no documento falaremos sobre a implementação dos restantes elementos.

  O ficheiro tem uma lista com todos os objetos do tipo *WeaponData* criados e outra com a *WeaponInstance* de cada arma já desbloqueada. Para além disso guarda também o preço atual de desbloqueio ou melhoria de armas e também o multiplicador que lhe é aplicado cada vez que uma compra relativa a armas é feita.

  Aqui encontramos também algumas funções relacionadas a armas que são utilizadas por outros *scripts*, funções que servem, por exemplo, para obter o *WeaponData* ou *WeaponInstance* de uma arma, desbloquear uma arma (adicionar a sua *instance* à lista de armas desbloqueadas), obter a próxima arma desbloqueada da lista, entre outras, e ainda, dentro da função *Awake*(), que corre no inicio do ciclo de vida do programa, a arma inicial, a *Hatchet*, é desbloqueada e equipada.

```
public class PlayerInventory : MonoBehaviour
{
    public List<WeaponData> allWeaponData;
    public List<WeaponInstance> ownedWeapons = new List<
       WeaponInstance>();
    [SerializeField] private int weaponPrice;
    [SerializeField] private float weaponPriceMult;

    //...

    public WeaponInstance GetWeaponInstance(string weaponID);

    public WeaponData GetWeaponData(string weaponID);

    public void UnlockWeapon(string weaponID);

    public string nextWeapon(string currentWeaponID);

    public string previousWeapon(string currentWeaponID);

    public int getWeaponPrice();

    public void IncreaseWeaponPrice();

    //...

    void Awake();
}
```

Excerto de Código 5.3: *PlayerInventory.cs* - armas (cabeçalhos)

- **WeaponManager.cs** (exerto de código 5.4)

  Script responsável por guardar a informação de qual é a arma equipada, equipar a nova arma quando é feita uma troca e, dentro da função *Update*(), processar os *inputs* relativos à arma feitos pelo utilizador, como disparar (*shoot*()) e trocar de arma.

```
public class WeaponManager : MonoBehaviour
{
    //...

    public WeaponBehaviour currentWeapon;

    //...

    public void EquipWeapon(string weaponId);

    void Update();
}
```

Excerto de Código 5.4: *WeaponManager.cs* (cabeçalhos)

- **WeaponBehaviour.cs** (exerto de código 5.5)

  Este *script* serve de base para um dos ficheiros únicos de cada arma, os ficheiros **Nome-DaArma*Behaviour.cs*, que vão estender o *WeaponBehaviour.cs*. Aqui declaramos quais são os métodos que as classes extensoras vão ter de obrigatoriamente implementar. Por exemplo, o método *shoot*() irá sempre ser chamado pelo *WeaponManager.cs*, então tem de estar sempre definido nas classes "*behaviour*" das armas.

```
public abstract class WeaponBehaviour : MonoBehaviour
{
    public WeaponInstance instance;
    public WeaponData data;

    public virtual void Initialize(WeaponInstance i, WeaponData d);

    public abstract void shoot();
}
```

Excerto de Código 5.5: *WeaponBehaviour.cs* (cabeçalhos)

5.2.1.2 Hatchet

A *Hatchet* é a arma inicial do jogo, e por tanto já vem desbloqueada desde o início e tem atributos mais baixos (ver figura 5.1).

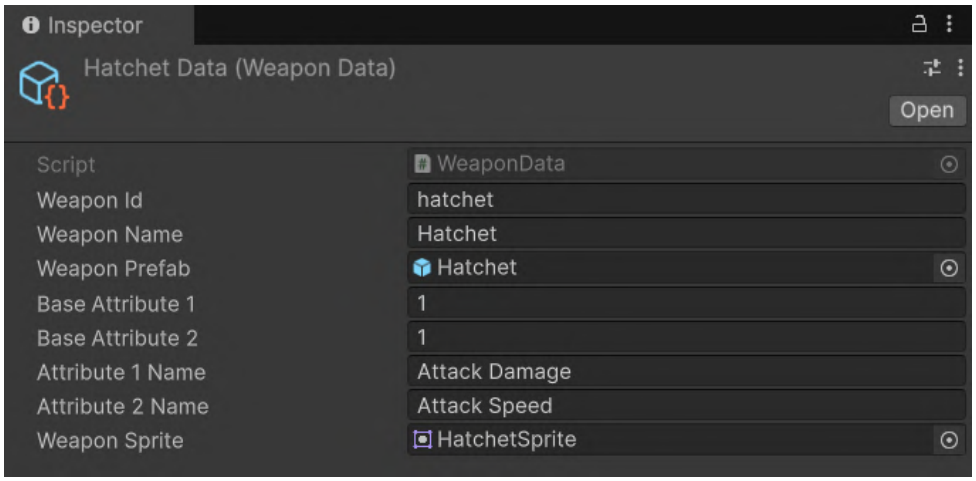

Figura 5.1: *Hatchet Data*

O *prefab* desta (ver figura 5.2) tem o modelo 3D da arma e uma *hitbox* que é ativada durante a animação de ataque para que seja feita a deteção de colisão com árvores.

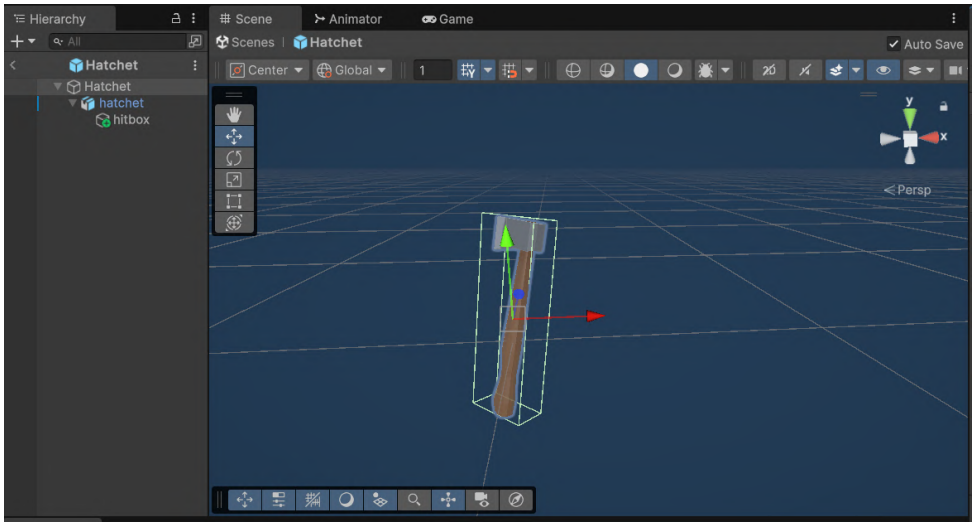

Figura 5.2: *Prefab* da *Hatchet*

Dentro do *prefab* temos uma instância do *HatchetBehaviour.cs* (excerto de código 5.6), que estende a classe *WeaponBehaviour.cs*. Esta inicializa (prepara) a arma para ser utilizada no método *Start*(), implementa a função *shoot*(), tal como é imposto pela classe *WeaponBehaviour.cs* e contém algumas outras funções que são utilizadas quando é feito um ataque (*shoot*()) e quando uma árvore é atingida.

```
public class HatchetBehaviour : WeaponBehaviour
{
    //...

    void Start();

    public override void shoot();

    public void EnableHitbox();

    public void DisableHitbox();

    public void setCanShoot(bool freeToShoot);

    public void applyDamage(Collider hit);

    private IEnumerator HandleRootCollider(Collider hit);
}
```

Excerto de Código 5.6: *HatchetBehaviour.cs* (cabeçalhos)

Em termos de animações, a *Hatchet* apenas tem a animação de ataque, que engloba a "ida" e a "volta" do golpe. Quando a função *shoot*() é chamada, o *animator* (ver figura 5.3) da arma passa do estado "*New State*" para o estado "*Attack*", executa a animação de ataque e volta para o "*New State*". A velocidade a que a animação é feita é controlada pelo nível do segundo atributo da arma, "*Attack Speed*".

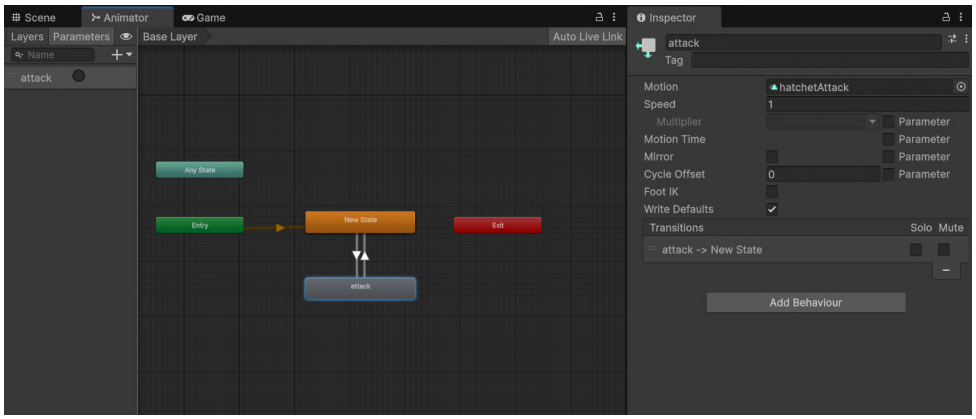

Figura 5.3: *Animator* da *Hatchet*

5.2.1.3 Rocket Gloves

Esta arma deve ser desbloqueada antes de poder ser utilizada, e por tanto tem o seu dano base um pouco mais elevado (ver figura 5.4).

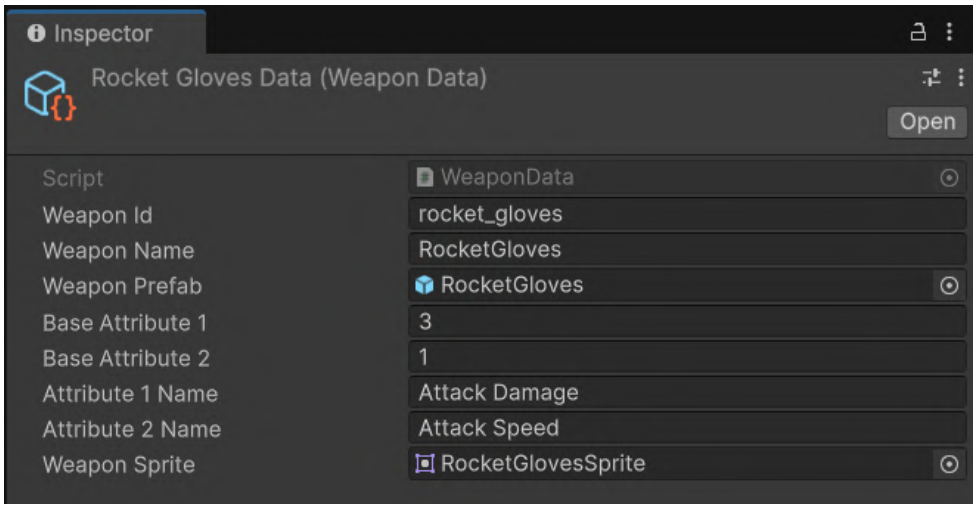

Figura 5.4: *Rocket Gloves Data*

O seu *prefab* (ver figura 5.5) contém o modelo 3D de cada luva cada uma tem uma *hitbox* associada. Estas são usadas na deteção de colisão com árvores quando é feito um ataque.

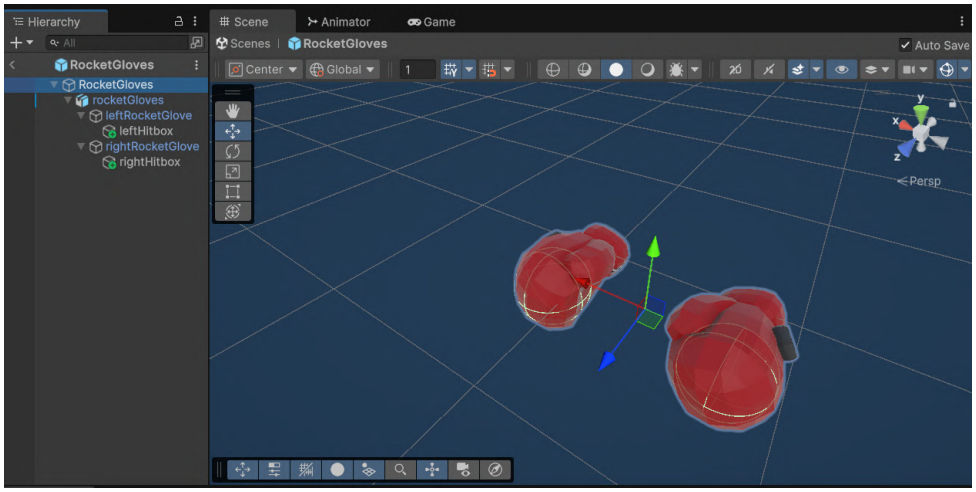

Figura 5.5: *Prefab* das *Rocket Gloves*

Tal como na *Hatchet*, este *prefab* contém o *behaviour* das *Rocket Gloves*, o *RocketGlovesBehaviour.cs* (excerto de código 5.7). Este é bastante semelhante ao da *Hatchet*, no entanto controla duas *hitbox*es e duas animações diferentes, enquanto que a *Hatchet* tem apenas uma de cada.

```
public class RocketGlovesBehaviour : WeaponBehaviour
{
    /...

    void Start();

    public override void shoot();

    public void EnableHitbox();

    public void DisableHitbox();

    public void setCanShoot(bool freeToShoot);

    public void applyDamage(Collider hit);

    private IEnumerator HandleRootCollider(Collider hit);
}
```

Excerto de Código 5.7: *RocketGlovesBehaviour.cs* (cabeçalhos)

Como já foi mencionado, as *Rocket Gloves* possuem duas animações, o soco com a mão direita e o soco com a mão esquerda. Assim, quando a função *shoot*() é chamada, uma variável booleana alterna entre *true* e *false*, o que controla se, dentro do *animator* (ver figura 5.6), a transição feita é de "*New State*" para "*rightRocketGlovePunch*" ou para "*leftRocketGlovePunch*", que são respetivamente os estados que tocam as animações de soco da mão direita e soco da mão esquerda. A velocidade da animação também é controlada pelo atributo "*Attack Speed*" da arma.

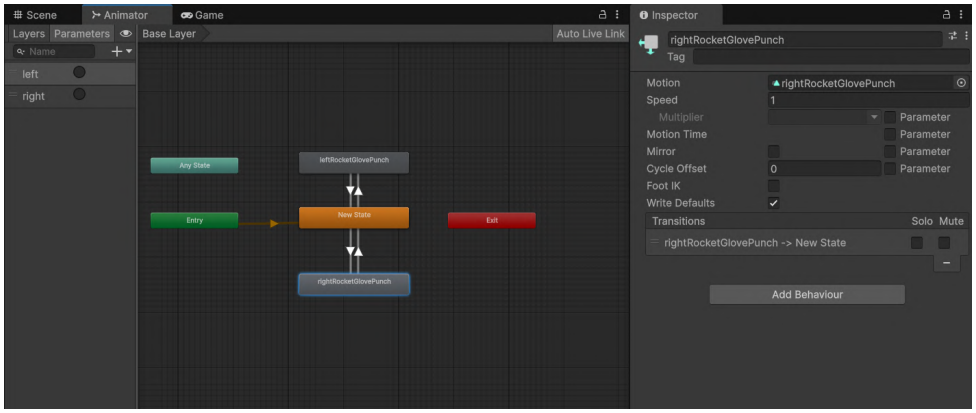

Figura 5.6: *Animator* das *Rocket Gloves*

5.2.2 Árvores e Sementes

As árvores e as sementes, apesar de serem objetos diferentes, trabalham juntas para que o jogo funcione como esperado. Além disso, alguma da lógica da implementação das armas também se aplica a estas, visto que também é feito o uso de *Scriptable Object*s para as sementes e *prefab*s para as árvores.

5.2.2.1 *SeedData*

Todas as sementes têm um *Scriptable Object*, feito a partir do *SeedData.cs* (excerto de código 5.8), que guarda as seguintes informações:

- ID da semente (*Seed ID*);
- Nome da semente (*Seed Name*);
- Tempo de crescimento (*Growth Time*);
- Preço da semente (*Seed Price*);
- *Prefab* da árvore (*Tree Prefab*);
- *Sprite* da semente (*Seed Sprite*);

```
[CreateAssetMenu(fileName = "SeedData", menuName = "Scriptable Objects/
    SeedsData")]
public class SeedData : ScriptableObject
{
    public string seedID;

    public string seedName;

    public float growthTime;

    public int seedPrice;

    public GameObject treePrefab;

    public Sprite seedSprite;
}
```

Excerto de Código 5.8: *SeedData.cs*

Portanto, temos dois objetos do tipo *SeedData*, um para cada semente:

1. ***Mahogany Seed***

   Semente mais barata e com tempo de crescimento menor mas cuja árvore não tem uma raiz tão valiosa (ver figura 5.7).

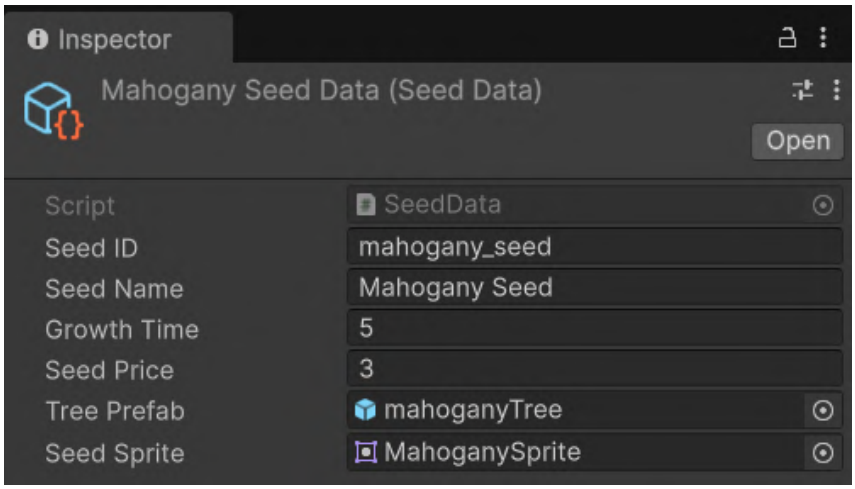

Figura 5.7: *Mahogany Seed Data*

2. ***Pine Seed***

   Mais cara, com maior tempo de crescimento mas a sua raiz vale mais dinheiro (ver figura 5.8).

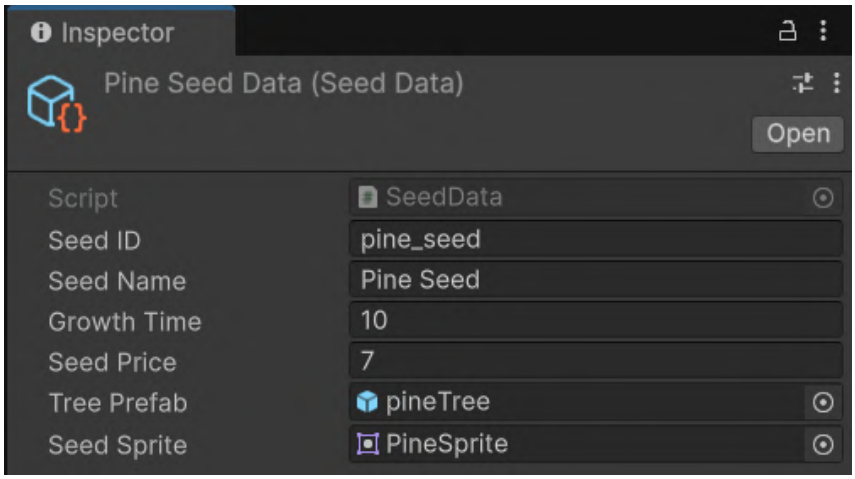

Figura 5.8: *Pine Seed Data*

5.2.2.2 Compra e Uso das Sementes

Como mencionado antes, para que as sementes sejam plantadas e originem árvores, estas primeiro devem ser adquiridas na loja, o que aumenta o número de sementes possuídas dentro do *PlayerInventory.cs* (excerto de código 5.9), e só depois colocadas dentro do *Planter* para que sejam então plantadas.

Tudo começa dentro da loja, onde, quando é feita a compra de uma semente, é chamada a função *incOwnedSeed*(*string seedI*) do *PlayerInventory.cs*, que aumenta a quantidade de uma dada semente, identificada pelo seu ID, em uma unidade.

Quando o jogador coloca todas as suas sementes dentro do *Planter*, depois de guardar o respetivo número dentro da máquina, é chamada a função *setOwnedSeed*(*string seedID, int qtt*) para cada semente para colocar o número destas dentro do inventário a zero.

```
public class PlayerInventory : MonoBehaviour
{
    //...

    public List<SeedData> allSeedData;
    private List<int> ownedSeeds = new List<int>();

    //...

    public List<int> getOwnedSeeds();

    public void setOwnedSeed(string seedID, int qtt);

    public void incOwnedSeed(string seedID);

    void Awake();
}
```

Excerto de Código 5.9: *PlayerInventory.cs* - sementes (cabeçalhos)

5.2.2.3 *Prefab*s das árvores

Já dentro do *Planter*, a semente é então "plantada", isto é, é esperado o tempo de crescimento da semente e, após o fim deste, um *prefab* da árvore que lhe corresponde é colocado num local aleatório da ilha. Sendo que todos os *prefab*s de árvores seguem a mesma estrutura, com modelos 3D para tronco e raiz, uma instância do *TreeScript.cs* e um *Box Collider* no tronco e uma instância de *RootScript.cs* e um *Sphere Collider* no objeto da raiz.

O *TreeScript.cs* (excerto de código 5.10) é responsável por guardar informações, como o HP da árvore e o valor da raiz e por tratar o dano que a árvore recebe, o caso em que o HP da mesma fica menor ou igual a zero e o "desaparecimento" da mesma do mapa após a sua raiz ser recolhida.

```
public class TreeScript : MonoBehaviour
{
    [SerializeField] private float hp;
    [SerializeField] private int rootValue;

    //...

    void Start();

    void FreezeTree();

    public void UnfreezeTree();

    public bool getFallenTree();

    public void setFallenTree(bool fallenTree);

    public bool takeDamage(float dmg);

    public int getRootValue();

    public void destroyTree();

    private IEnumerator waitAndDestroy();
}
```

Excerto de Código 5.10: *TreeScript.cs* (cabeçalhos)

O *RootScript.cs* (excerto de código 5.11) é o que controla principalmente a *hitbox* da raiz. Esta começa sempre desativada e o seu estado apenas é alterado x segundos após a árvore cair, para que o jogador possa então recolher a raiz.

```
public class RootScript : MonoBehaviour
{
    void Start();

    public void DestroyRoot();

    public void DisableRootCollider();

    public void EnableRootCollider();

    private IEnumerator waitAndEnable();

    public int getRootValue();
}
```

Excerto de Código 5.11: *RootScript.cs* (cabeçalhos)

No fundo, as únicas diferenças entre os *prefab*s das árvores são os modelos 3D (ver figuras 5.9 e 5.10) e os valores dos parâmetros definidos em cada *TreeScript.cs*.

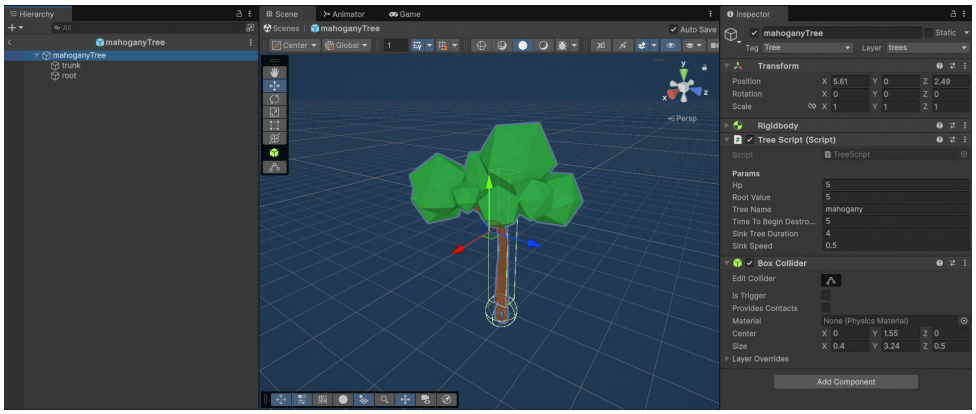

Figura 5.9: *Prefab* da *Mahogany Tree*

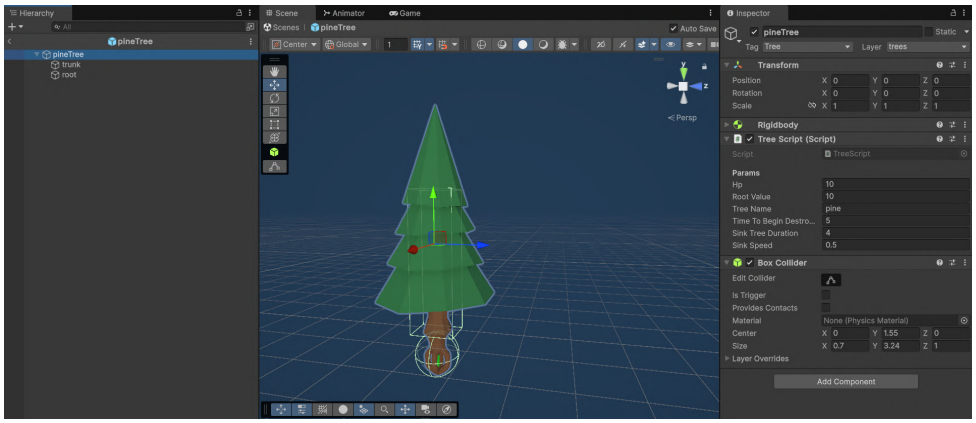

Figura 5.10: *Prefab* da *Pine Tree*

5.2.3 Interações com Objetos

Todas as interações do jogo (com a tecla F) partem de um mesmo *script*, o *InteractScript.cs* (excerto de código 5.12). Este, ao detetar que o jogador clicou na tecla F, chama a função *Detect()* que cria uma *OverlapSphere* invisível à sua frente, esta deteta se está em *overlap* com objetos das camadas "*trees*" ou "*interactables*" e, para cada objeto encontrado, analisa a sua *tag* e chama a função correspondente a esse caso.

Para além disso, se a tecla ESC for clicada, este *script* trata a ativação e desativação da interface correspondente ao menu de pausa, recorrendo aos métodos *handlePause*() e *getIsPaused*().

```
public class InteractScript : MonoBehaviour
{
    //...

    private bool isShoping = false;
    private bool isPaused = false;

    void Start();

    void Update();

    private void Detect();

    private void handleRoot(Collider hit);

    private void handlePlanter(Collider hit);

    public void handleShop(Collider hit);

    public bool getIsShoping();

    public void handlePause();

    public bool getIsPaused();

    /*----------------------------//---------------------------------*/
    //  Creates a sphere that represents the area affected by the hit
       detection (Debug purposes)
    private void VisualizeDetection(Vector3 explosionPoint);
}
```

Excerto de Código 5.12: *InteractScript.cs* (cabeçalhos)

5.2.3.1 Loja

Quando a loja é detetada, a UI do menu da loja é ativada e todas as outras UIs são desativadas.

Para além disso, o movimento do jogador, da câmera, e a ação de disparar ficam desativados enquanto a loja está aberta. Isto é controlado pela variavél booleana "*isShoping*" que está dentro do *InteractScript.cs*.

O menu da loja é controlado pelo *ShopUIScript.cs* (excerto de código 5.13), que tem referências ao elementos da UI feita no editor e os altera conforme necessário.

Um ponto importante do funcionamento deste são as funções *openSeedInfo*() e *openWeaponInfo*(), que abrem o painel com as informações específicas do item selecionado.

```
public class ShopUIScript : MonoBehaviour
{
    //...

    public void closeShop();

    public void UpdateMoney();

    public void openSeedInfo(string seedID);

    public void openWeaponInfo(string weaponID);


    public void desableAllInfos();

    public void buySeed();

    public void buyUpgrade(int attNumber);


    public void unlockWeapon();
}
```

Excerto de Código 5.13: *ShopUIScript.cs* (cabeçalhos)

Vale também a pena referir as funções *buySeed*(), *buyUpgrade*(*int attNumber*) e *unlockWeapon*(). Estas são a funções chamadas pelos botões de compra do painel de informação do item seleciondado. Um ponto em comum entre estas 3 funções é que todas interagem com o *PlayerInventory.cs* (excerto de código 5.14) para verificar se o jogador tem dinheiro suficiente para fazer a compra e, após a conclusão desta, para atualizar o seu valor.

```
public class PlayerInventory : MonoBehaviour
{
    //...

    private int money;

    //...

    public int getMoney();

    public void setMoney(int money);

    //...
}
```

Excerto de Código 5.14: *PlayerInventory.cs* - dinheiro (cabeçalhos)

O *prefab* da loja (ver figura 5.11) tem o seu modelo 3D e um *box collider*, que serve para que a loja seja identificada pelo *interactScript.cs* e também para que o jogador não a consiga atravessar.

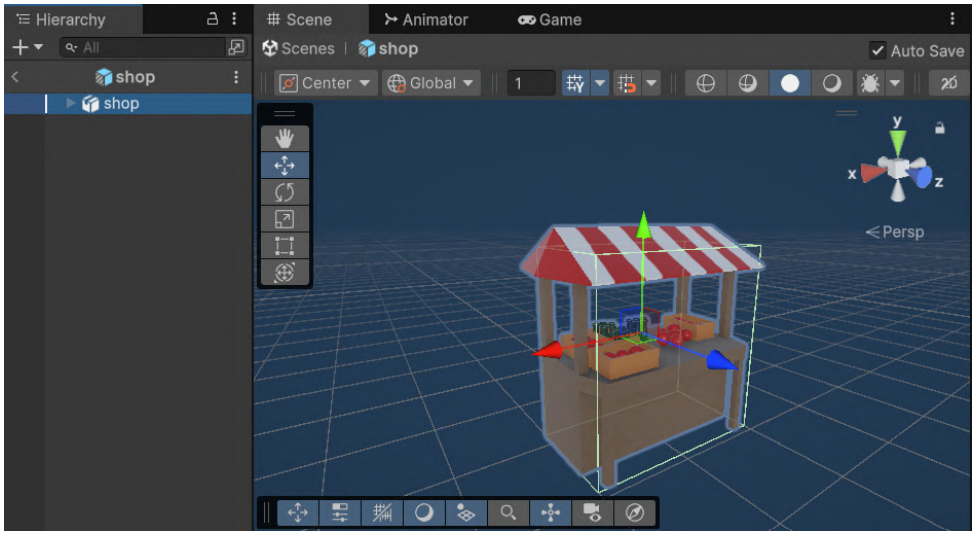

Figura 5.11: *Prefab* da loja

5.2.3.2 *Planter*

Se o *Planter* for detetado não há qualquer alteração a nível da interface principal do jogo, o que acontece é que todas as sementes dentro do inventário são transferidas para dentro do *Planter* e o painel deste, que mostra o números de sementes dentro dele, é atualizado, da mesma maneira que atualiza sempre que uma árvore é plantada e o número de sementes diminui.

Todas as funcionalidades relativas ao *Planter* envolvem o *PLanterScript.cs* (excerto de código 5.15). Este tem as funções usadas na adição e remoção de sementes e na plantação de árvores.

```
public class PlanterScript : MonoBehaviour
{
    //...

    public GameObject groundObject;

    //...

    void Start();

    void Update();

    public int GetSeedQuantity(string seedID);

    public void addSeeds(string seedID, int seedQTT);

    public void updateCard(int ID, int newQtt);


    IEnumerator PlantFreeSeeds();

    IEnumerator PlantSeed(int index);
    public void TryPlantSeed(int index);

}
```

Excerto de Código 5.15: *PlanterScript.cs* (cabeçalhos)

De entre as funções, cujos cabeçalhos se encontram na figura a cima, devemos dar especial atenção a *PlantFreeSeeds*(). Esta é executada dentro de uma corrotina que é iniciada no método *Start*() e é responsável por adicionar uma semente de *mahogany* ao planter, de forma gratuita, a cada 60 segundos (excerto de código 5.16). Isto serve para evitar o caso em que o jogador não tem dinheiro para comprar sementes nem árvores no cenário para abater, o que destruiria completamente a progressão do jogo.

```
IEnumerator PlantFreeSeeds()
    {
        while (true)
        {
            yield return new WaitForSeconds(timeToPlantFreeTree);

            availableSeeds[0]++;
            updateCard(0, availableSeeds[0]);
            StartCoroutine(PlantSeed(0));
        }
    }
```

Excerto de Código 5.16: *PlantFreeSeeds*()

O *prefab* do *Planter* (ver figura 5.12), tal como a loja, tem o modelo 3D e um *box collider* para ser detetado pelo *InteractScript.cs* e não ser atravessado pelo jogador. No entanto este tem alguns elementos a mais, como o *PlanterScript.cs*, que aqui está colocado no objeto pai dentro do *prefab* e o *canvas* que mostra o painel com a *sprite* e o número de cada semente dentro da máquina.

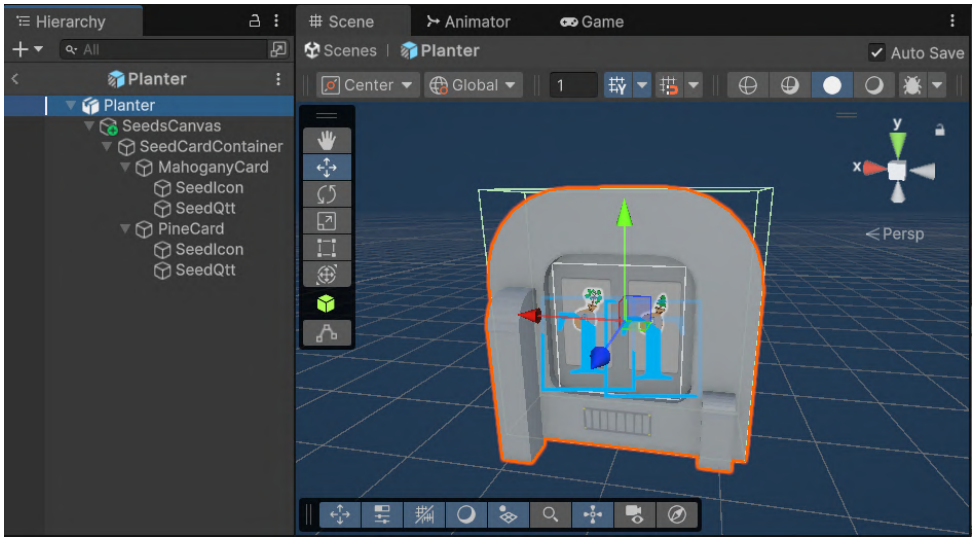

Figura 5.12: *Prefab* do *Planter*

5.2.4 *PlayerCatcher*

Algo que poderia colocar em causa o funcionamento o jogo e acabar com a experiência do jogador é o que acontece quando o mesmo cai da ilha. Visto que não existe nenhuma barreira física que impeça o jogador de cair ou de saltar, foi necessário implementar uma outra solução para este problema, o *PlayerCatcher*. Este é um objeto invisível que fica debaixo da ilha e quando o jogador entre em contacto com ele é transportado para as coordenadas (0,0,0), que é justamente o centro da cena e da superfície da ilha.

O funcionamento deste objeto é controlado pelo *PlayerCatcherScript.cs* (excerto de código 5.17) que, sempre que algum outro objeto entra em contacto consigo, verifica se este possui a *tag* "*Player*" e, caso possua, altera a sua posição na cena para "*Vector3.zero*", ou seja, para (0,0,0).

```
public class PlayerCatcherScript : MonoBehaviour
{
    private void OnTriggerEnter(Collider other)
    {
        // Checks if the object that entered the trigger is the Player
        if (other.CompareTag("Player"))
        {

            Debug.Log("Inside the collider");
            Debug.Log($"{other.gameObject.name}");

            // Gets the Player's CharacterController
            CharacterController controller = other.GetComponent<
                CharacterController>();
            if (controller != null)
            {
                // Teleports the player to (0,0,0)
                controller.enabled = false; // Temporarily disable to
                    avoid conflicts
                other.transform.position = Vector3.zero;
                controller.enabled = true; // Reactivates
                    CharacterController
            }
        }
    }
}
```

Excerto de Código 5.17: *PlayerCatcherScript.cs*

Este objeto também tem um *prefab* (ver figura 5.13) que contem o *PlayerCatcherScript.cs* e um *Box Collider*.

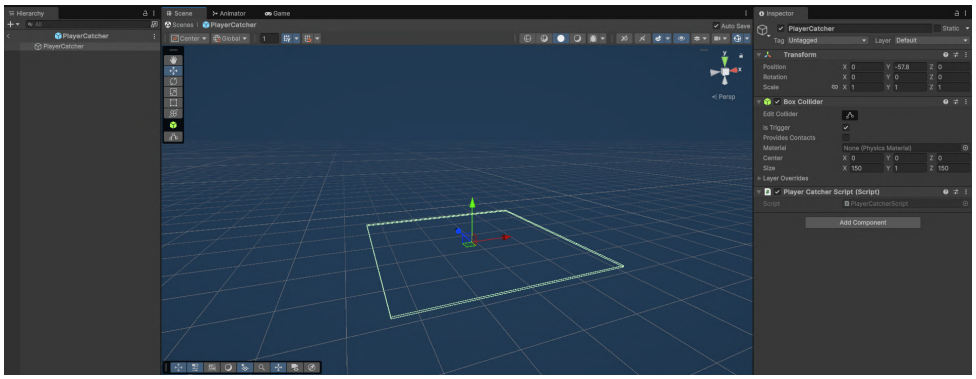

Figura 5.13: *Prefab* do *PlayerCatcher*

Importante notar que a opção "*Is Trigger*" do *Box Collider* está ativa. Isto serve para que o *collider* identifique o momento em que colide com outro objeto.

5.2.5 Interfaces e Menus

Todas as interfaces do jogo têm como base um *canvas* que aloja todos os elementos que são visualizados no ecrã, como painéis, imagens, textos, botões, etc. (ver figura 5.14).

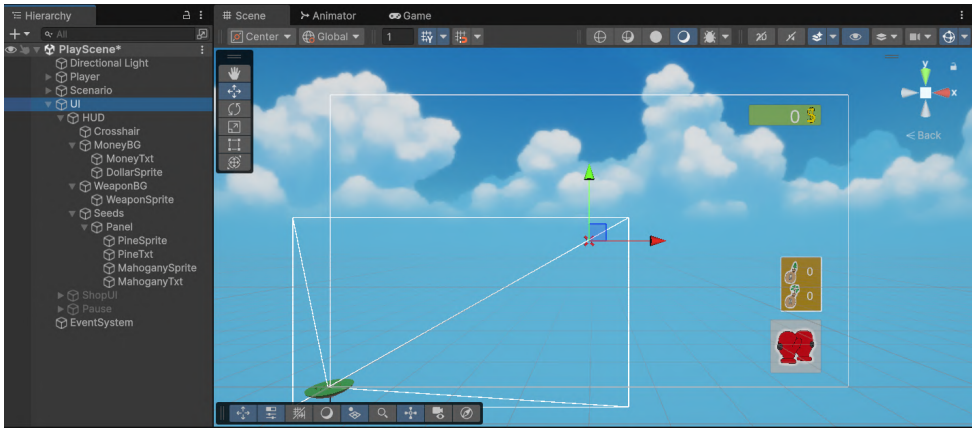

Figura 5.14: Exemplo de hierarquia de uma interface

Para fazer a troca entre interfaces, é feita a ativação e desativação de cada uma conforme necessário usando a função *SetActive*() já definida em todos os *game objects* da Unity (excerto de código 5.18).

```
Transform shopUI = UI.transform.Find("ShopUI");
Transform hud = UI.transform.Find("HUD");

if (hud != null) hud.gameObject.SetActive(false);
if (shopUI != null) shopUI.gameObject.SetActive(true);
```

Excerto de Código 5.18: Ativação/Desativação de Interfaces

5.2.5.1 Menu Inicial

Este é o primeiro menu que aparece quando o jogo é aberto. Nele podemos ver o logótipo do jogo w os botões "*PLAY*", que troca para a *PlayScene* e inicia o jogo, e "*EXIT*", que fecha o jogo (ver figura 5.15).

Figura 5.15: Menu Inicial

Tanto o logótipo e os botões mencionados estão dentro de um painel, que serve como fundo, e este são compostos por:

- **Logótipo**: Uma imagem;
- **Botão "*PLAY*"**: Um botão;
- **Botão "*EXIT*"**: Um botão.

A interface é controlada pelo *TitleScreenScript.cs* que se encontra dentro do *canvas* "pai" (excerto de código 5.19).

```
public class TitleScreenScript : MonoBehaviour
{
    public void playBtn();

    public void exitBtn();
}
```

Excerto de Código 5.19: *TitleScreenScript.cs* (cabeçalhos)

5.2.5.2 HUD

O HUD é a interface que fica visível durante o jogo ativo, isto é, quando o jogador não está dentro de nenhum menu. Este dispõe o retículo (*crosshair*) no centro do ecrã, no canto inferior direito, a arma equipada no momento, a cima deste e caso o número de sementes dentro do *PlayerInventory.cs* seja superior a 0, mostra a quantidade possuída de cada semente e, no canto superior direito, o dinheiro disponível (ver figuras 5.16 e 5.17).

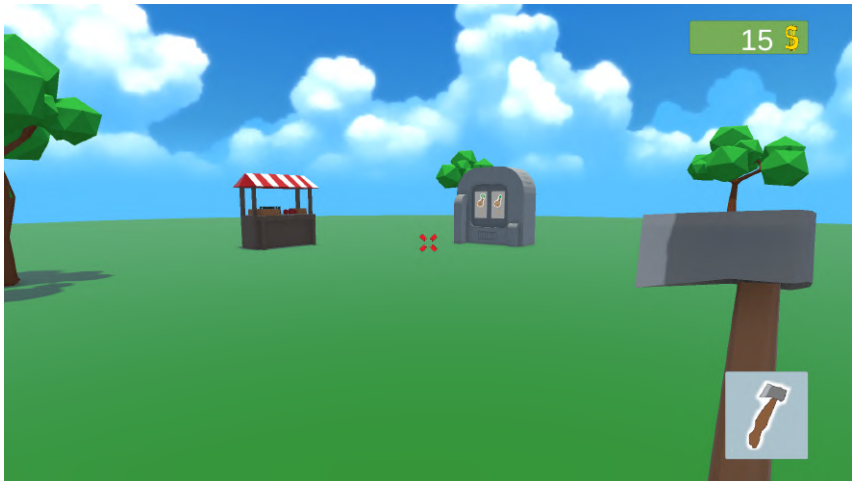

Figura 5.16: HUD sem sementes

Figura 5.17: HUD com sementes

Os elementos do HUD, em termos de hierarquia no editor, são compostos por:

- ***Crosshair***: Uma imagem;
- **Arma equipada**: Um painel com uma imagem;
- **Sementes**: Um *canvas* com um painel que contem uma imagem e uma caixa de texto para cada semente;
- **Dinheiro**: Um painel com uma imagem e uma caixa de texto.

O *canvas* "pai" desta interface tem ainda um *HudScript.cs* (excerto de código 5.20) associado, que controla as informações que aparecem no ecrã, isto é, a quantidade de dinheiro, a *sprite* da arma e o painel com a informação sobre as sementes.

```
public class HudScript : MonoBehaviour
{
    //...

    public void UpdateMoney();

    public void UpdateWeapon();

    public void UpdateSeeds();
}
```

Excerto de Código 5.20: *HudScript.cs* (cabeçalhos)

5.2.5.3 Menu da Loja

O menu da loja aparece quando o jogador entra na loja. Este contém um botão de saída, a informação de quanto dinheiro o jogador tem, a lista de sementes e a lista de armas, sendo estas listas compostas por botões que, ao serem clicados abrem o painel de informação do respetivo item. Estes painéis contêm uma imagem, caixas e texto com informações do item e um ou mais botões de compra (ver figura 5.18).

Figura 5.18: Menu da loja

Os elementos mencionados são compostos por:

- **Botão de saída**: Um botão;
- **Dinheiro**: Um painel com uma imagem e uma caixa de texto;
- **Lista de sementes**: Uma caixa de texto e por baixo um painel com, para cada semente, um painel com um botão dentro;
- **Lista de armas**: Uma caixa de texto e por baixo um painel com, para cada arma, um painel com um botão dentro;
- **Painel de informação**: Um painel com 3 painéis dentro, sendo que apenas um destes fica ativo por vez. Estes contêm uma imagem, caixas de texto e um ou mais botões, sendo que todos estes, à exceção dos botões, estão dentro de um painel próprio.

Este menu é controlado pelo *ShopUIScript.cs* (excerto de código 5.13) que se encontra no *canvas* "pai" da interface.

5.2.5.4 Menu de Pausa

Este menu aparece quando o jogador carrega na tecla ESC. Nele podemos ver o logótipo do jogo e dois botões, um de "*RESUME*", para fechar o menu e retomar o jogo, e um de "*EXIT TO TITLE*", que retorna para a cena inicial do jogo (ver figura 5.19).

Figura 5.19: Menu de Pausa

Todos os elementos deste menu estão dentro de um painel e estes são compostos por:

- **Logótipo**: Uma imagem;
- **Botão "*RESUME*"**: Um botão;
- **Botão "*EXIT TO TITLE*"**: Um botão.

O que controla esta interface é o *PauseUIScript.cs* (excerto de código 5.21) colocado dentro do seu "pai".

```
public class PauseUIScript : MonoBehaviour
{
    //...

    public void exitBtn();

    public void resumeBtn();
}
```

Excerto de Código 5.21: *PauseUIScript.cs* (cabeçalhos)

5.2.6 Cenas (*Scenes*)

Na Unity, cada nível/ambiente é chamado **cena**. Conforme o jogo avança é feita a troca entre cenas para colocar o jogador num novo ambiente, por exemplo, quando, no menu inicial, o botão "*PLAY*" é clicado o jogo troca da cena que contem este menu para a cena onde o jogo acontece, com a ilha, as armas, etc.

Desta forma, no desenvolvimento do *Treetening*, foram criadas duas cenas, a "*TitleScene*" e a "*PlayScene*". Estas têm alguns pontos em comum, nomeadamente a *skybox*, que é a mesma nas duas cenas. Esta foi importada da *Unity Asset Store* e aplicada como material, como podemos ver na figura a baixo (figiura 5.20).

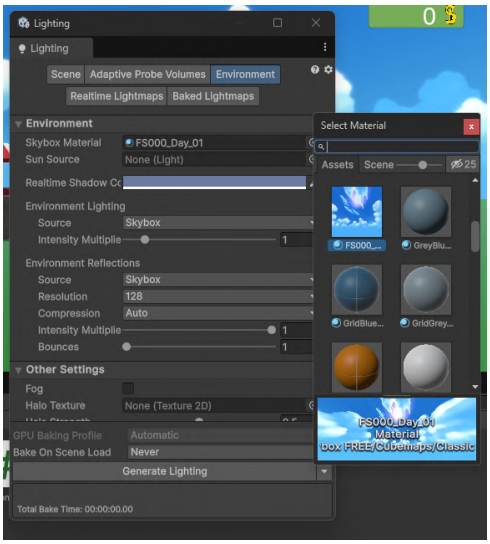

Figura 5.20: *Skybox*

5.2.6.1 *TitleScene*

Este é a primeira cena que é carrega quando o jogo é aberto. Nela podemos ver o menu inicial, que é uma interface que está sempre ativa. De fundo, vemos a ilha com a loja, o *planter* e algumas árvores no topo. Sob a ilha, um sistema de partículas simula um efeito de nevoeiro (ver figura 5.21).

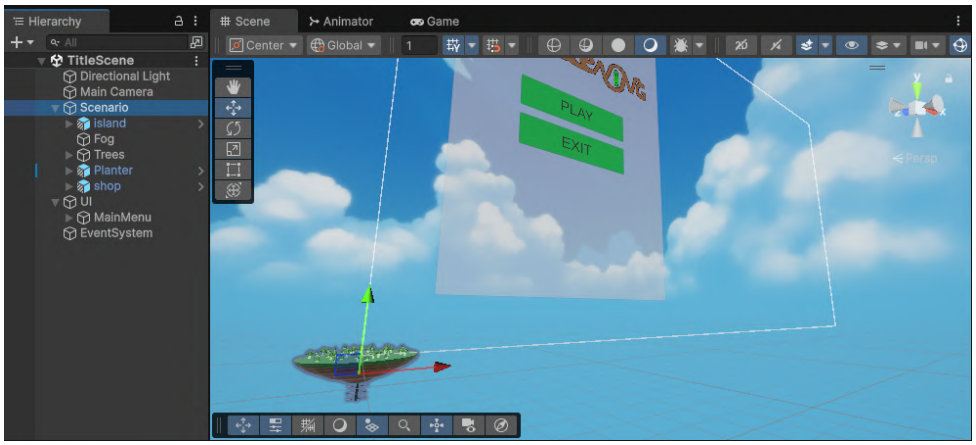

Figura 5.21: *TitleScene*

Na hierarquia da cena, à direta, podemos ver que esta é composta por:

- ***Directional Light***: A "fonte" de luz da cena. Esta influencia como os objetos são iluminados e como as suas sobras são projetadas;
- ***Main Camera***: A câmera principal da cena. É esta que define onde estão os "olhos" da cena, ou seja, o que é visto e de onde é visto;
- ***Scenario***: Um objeto "*empty*" que aloja todos os elementos do cenário da cena. Neste caso, *Scenario* tem como filhos o sistema de particulas que simula o nevoeiro e os *prefab*s da ilha (*island*), de todas as árvores em cena (*trees*), do *planter* e da loja (*shop*);
- **UI**: Semelhante ao *Scenario*, UI também aloja elementos mas, neste caso, os relativos ao menu inicial 5.2.5.1 (*MainMenu*);
- ***EventSystem***: Elemento que é adicionado automaticamente sempre que se coloca algo interativo na UI, como por exemplo um botão. É responsável pelo tratamento de *inputs* relativos à UI

5.2.6.2 *PlayScene*

Quando o botão "*PLAY*" na *TitleScene* é clicado é esta cena que é carregada. É aqui que o jogo em si decorre. Esta contém todos os elementos necessários para o funcionamento do jogo, desde a ilha com a loja, as árvores e o *planter*, ao objeto "*Player*" que o jogador controla ao jogar o jogo (ver figura 5.22).

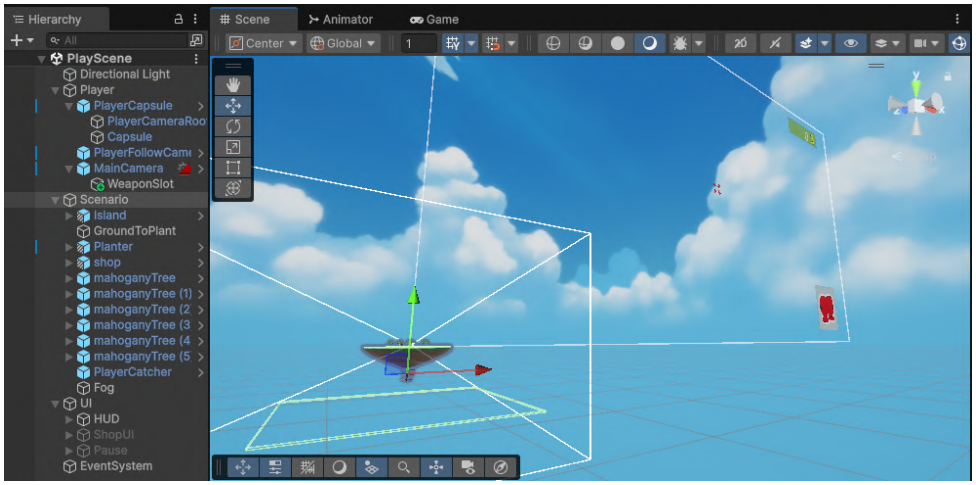

Figura 5.22: *PlayScene*

Podemos observar que esta cena tem alguns elementos em comum com a *TitleScene* 5.2.6.1. No entanto também conta com alguns novos:

- ***Directional Light***: Igual à da *TitleScene*;
- ***Player***: Objeto *empty* que contém todos os *scripts* e filhos relativos ao *player* e às armas, como a capsula que representa o jogador, câmera que este controla e o *WeaponSlot* que é o local na cena onde a arma equipada é colocada;
- ***Scenario***: Semelhante ao da *TitleScene*, com exceção ao *prefab* do *PlayerCatcher* e do "*GroundToPlant*", que é um objeto invisível que sobrepõe o chão da ilha e diz ao *planter* qual a área onde plantar;
- **UI**: Da mesma forma que o da *TitleScene*, contem as UIs utilizadas nesta cena, seno neste caso o HUD 5.2.5.2 (HUD), o menu da loja 5.2.5.3 (*shopUI*) e o menu de pausa 5.2.5.4 (*Pause*);
- ***EventSystem***: Igual ao da *TitleScene*.

Capítulo

6

Conclusão e Trabalho Futuro

Falemos então sobre os objetivos alcançados e aquilo que foi planeado mas não chegou a ser concretizado. Este capítulo reflete sobre o resultado final do projeto e trabalhos que podem vir a ser realizados para melhorar o mesmo.

6.1 Resultado Final e Objetivos Alcançados

O resultado final do projeto corresponde, quase na sua totalidade, à visão inicial que existia sobre este. Tanto o estilo visual como as principais mecânicas de jogo e o sentimento base da experiência foram concretizados de forma satisfatória. No entanto, certos elementos não chegaram a ser implementados dentro do prazo estabelecido, nomeadamente o sistema de áudio, o sistema de *saves*, mais conteúdos, como armas e tipos de árvore e alguns pormenores visuais e efeitos que poderiam enriquecer ainda mais a experiência de quem joga.

O jogo apresenta um *loop* de *gameplay* bem definido e que oferece uma experiência concisa ao jogador, em que todas as mecânicas que apresenta funcionam na sua totalidade e que tem em atenção inclusive alguns problemas que poderiam vir a ocorrer e portanto foram prevenidos.

Não é ainda a versão final do jogo mas pode se considerar que se trata de uma versão *beta* bastante avançada.

6.2 Trabalho Futuro

Como mencionado anteriormente, alguns aspetos que definitivamente deveriam vir a ser implementados são os sistemas de áudio e de *saves*.

A ausência de áudio e efeitos sonoros é algo que é imediatamente notado por quem joga e que por consequência quebra a sua experiência com o jogo.

O sistema de *saves* é uma mecânica que certamente enriqueceria bastante a sensação de progressão do jogador. Com este o jogo não seria reiniciado sempre que é aberto.

Para além disso, especialmente tratando-se de um jogo com foco em melhoria e desbloqueio de equipamentos, mais conteúdo deveria ser adicionado. Principalmente armas. Algumas, já projetadas, seriam os *Bomber Planes*, aviões de papel que explodem ao colidir com algo e o *Saw Hat*, uma cartola com um serra na aba que o jogador atira numa linha reta e que depois volta para ele.

Bibliografia

[1] Brett Molina, USA TODAY. Five reasons you need to play 'Overwatch', 2016. [Online] https://eu.usatoday.com/story/tech/gaming/2016/06/01/five-reasons-you-need-play-overwatch/85242348/, Último acesso a 29 de Junho de 2025.

[2] Nic Healey, CNET. Overwatch: Release date, gameplay and everything else you need to know, 2016. [Online] https://www.cnet.com/reviews/overwatch-preview/, Último acesso a 29 de Junho de 2025.

[3] The1AndOnlyMike, fandom. CorralAutofeeder, 2025. [Online] https://slimerancher.fandom.com/wiki/Corral_(Slime_Rancher)?file=CorralAutofeeder.png, Último acesso a 29 de Junho de 2025.

[4] Nick Galov, WebTribunal. Key Mobile Gaming Statistics: How Many People Play in 2023?, 2023. [Online] https://webtribunal.net/blog/mobile-gaming-statistics, Último acesso a 29 de Junho de 2025.

[5] Joey Carlino. Character modeling for beginners - Blender, 2024. [Online] https://youtu.be/O6HQhs-gk50?si=YisJMMFZzLSWQgGU. Último acesso a 21 de Abril de 2025.

[6] LMHPOLY. GAME ASSET BEGINNER TUTORIAL - Blender to Unity (Introduction), 2021. [Online] https://youtu.be/KFEb51rinwI?si=fo5tjhu6uu_VRpk1. Último acesso a 11 de Maio de 2025.

[7] Jamie Dunbar. Blender 4.0 - Texture Painting quick start guide, 2023. [Online] https://youtu.be/iwWoXMWzC_c?si=8AzE59bgUsXr1evT. Último acesso a 11 de Maio de 2025.

[8] Ryan King Art. Animation for Beginners! (Blender Tutorial), 2022. [Online] https://youtu.be/CBJp82tlR3M?si=kF-gFzJcTcYLYk0w. Último acesso a 20 de Maio de 2025.

[9] The Game Dev Cave. How to Export Animation From Blender For Game Engines Like Unity or Unreal, 2023. [Online] https://youtu.be/kUn4jUj4gOk?si=IAETkP6borXZq8yl. Último acesso a 20 de Maio de 2025.

[10] RSDevelopment. Idle, Walk, and Run Animations in Unity Tutorial, 2022. [Online] https://youtu.be/5mlwvbu1fxQ?si=45Y_uEjad1lsa5SI. Último acesso a 9 de Junho de 2025.

[11] Synty Studios. How to use Animation Events in Unity (Tutorial) by SyntyStudios, 2023. [Online] https://youtu.be/92P2Zz6K9vA?si=e311cVVdwCB4KeHB. Último acesso a 9 de Junho de 2025.

[12] Unitips. Unity Change Scene With Button - EASIEST Method, 2021. [Online] https://youtu.be/zQH7RRb3CnY?si=4VnhXNMY7BU_MMnG. Último acesso a 26 de Junho de 2025.

[13] ASCENDER. Mastering Unity Responsive UI: Pivot and Anchors Tutorial for Mobile Design, 2024. [Online] https://youtu.be/MXg7zqfZBmc?si=q42x34x8Nsf3ve74. Último acesso a 26 de Junho de 2025.