ESCOLA SECUNDÁRIA DE CAMÕES

CURSO PROFISSIONAL TÉCNICO DE GESTÃO E PROGRAMAÇÃO DE SISTEMAS INFORMÁTICOS

PROVA DE APTIDÃO PROFISSIONAL

(RELATÓRIO)

Wild Soul: Fragmentos Perdidos

Alisson Teixeira de Paula
NÚMERO: 1 TURMA: 12N

Valdir Fátima Reis Pires
NÚMERO: 16 TURMA: 12N

Professor(es) orientador(es):

Professor Alexandre Barão
Professor Nuno Padeiro

Julho, 2025.

Página em branco

Resumo

Este projeto consiste na criação de um jogo intitulado *Wild Soul: Fragmentos Perdidos*, um *site* de apoio e um servidor Discord.

A narrativa do jogo situa-se num universo inspirado na mitologia nórdica, antes de um evento catastrófico conhecido como *Permafrost*, que trará um inverno rigoroso e um desequilíbrio entre o plano físico e espiritual. O jogador controla uma alma perdida, sem memória, que desperta numa floresta esquecida e embarca numa missão para recuperar fragmentos do seu passado. Durante a jornada, enfrenta criaturas corrompidas e os Guardiões que guardam toda a história para compreender e proteger o mundo.

Além do jogo, foi desenvolvido um *site* que funciona como plataforma de apresentação e divulgação do projeto, permitindo a comunicação com o público, a partilha de atualizações e a disponibilização de conteúdos complementares.

A plataforma Discord possui um servidor que serve como canal direto para interagir com os utilizadores, recolher *feedback* e fomentar uma comunidade em torno do jogo, facilitando a troca de ideias e o envolvimento dos interessados no desenvolvimento.

Palavras-chave: Wild Soul, jogo *indie*,RPG, mitologia nórdica, *game design*, *web design,* html5, css3, php, javas*cript*, mySql.

Página em branco

Notações

2D	*Bidimensional*
3D	*Tridimensional*
AES	*Advanced Encryption Standart*
AIFF	*Audio Interchange File Format*
CSS	*Cascade Style Sheet*
FLAC	*Free Lossless Audio Codec*
GIF	Graphics Interchange Format
GML	*GameMaker Language*
HTML	*Hypertext Markup Language*
ID	Identificador
IDE	*Integrated Development Environment*
MP3	*MPEG Audio Layer 3*
MPEG	*Moving Picture Experts Group*
NPC	*Non-Playable-Character*
PNG	*Portable Network Graphic*
RPG	*Role-Playing-game*
SSL	*Secure Sockets Layer*
TGPSI	Técnico de Gestão e Programação de Sistemas Informáticos
UML	*Unified Modelling Language*
WAV	*Waveform Audio File Format*

Página em branco

Agradecimentos

Expressamos aqui o nosso mais sincero reconhecimento à **Escola Secundária de Camões**, cuja formação ao longo destes três anos nos conferiu aptidões indispensáveis para o nosso percurso académico e profissional.

Ao nosso orientador, Professor **Alexandre Barão**, manifestamos a nossa profunda gratidão, foi ele, de forma totalmente voluntária e (também) nos seus tempos livres, quem nos direcionou, ajustou e amparou em cada fase, possibilitando a concretização deste relatório. Agradecemos igualmente ao Professor **Nuno Padeiro**, cujo suporte permanente ao longo do curso enriqueceu o nosso crescimento técnico e pessoal.

Apesar de termos trabalhado em equipa, segue-se o reconhecimento individual.

Alisson Paula
Registo a minha gratidão à família e aos amigos que dedicaram o seu tempo a testar o jogo "*Wild Soul*" e, em especial, a **Orlando Emanuel Costa**, fundador e responsável da produtora musical *LX PRO*, onde realizei o meu estágio com missões de desenvolvimento/atualização da página *web* da empresa. A sua orientação constante e a oportunidade de aplicar conhecimentos técnicos foram decisivas para reforçar a minha experiência e ultrapassar os desafios do *site* deste projeto.

Valdir Pires
Quero agradecer à minha família pelo apoio constante e por me lembrarem, sempre que foi preciso, para não desistir. Esse incentivo foi essencial para chegar até aqui. Agradeço também aos amigos que ajudaram a dar forma à ideia do *Wild Soul*, com opiniões, ideias e aquela força nos momentos certos. Sou também grato ao meu amigo Alisson Paula, cuja colaboração e empenho foram fundamentais para a concretização deste trabalho.

A todos, o nosso profundo agradecimento.

Página em branco

Índice

Índice de Figuras

Página em branco

Índice de Tabelas

Página em branco

1. Introdução

Neste capítulo, apresentam-se os principais elementos introdutórios que servem de base para a compreensão do trabalho realizado. Na Secção 1.1, são descritos o contexto e a motivação do projeto, destacando a relevância e os fatores que impulsionaram o seu desenvolvimento. Na Secção 1.2, é apresentado o problema que se pretende abordar, incluindo os desafios e as necessidades identificadas.

De seguida, na Secção 1.3, são descritos os resultados alcançados, fornecendo uma visão geral sobre os progressos realizados. Na Secção 1.4, são delineados os objetivos do trabalho, definindo de forma clara as metas a atingir.

A Secção 1.5 apresenta o cronograma, detalhando a organização temporal das atividades realizadas ao longo do projeto. Finalmente, na Secção 1.6, descreve-se a estrutura do documento, oferecendo ao leitor uma visão geral da organização e conteúdo dos capítulos subsequentes.

1.1. Contexto e Motivação

Este projeto centra-se na área da indústria dos jogos. Em 2010, Portugal começou a receber destaque internacional na criação de jogos *indie*, em particular, com o sucesso da Lisboa *GameWeek*. O ano de 2014 foi relevante ao nível de novas criações de jogos em Portugal, mas apenas em 2021 que o país obteve uma notoriedade mais expressiva com o jogo "*Out of Line*" do grupo de programadores *Nerd Monkey*, e com o jogo "*Decoherence*" da empresa *Efecto Studio*.

Inspirado pelo contexto da criação de jogos, os autores desenvolveram o conceito *Wild Soul*. Trata-se de uma narrativa que se passa num mundo tomado por uma *Permafrost* em que mais de 70% da humanidade foi extinta, e por outro lado, animais sofrem drásticas mudanças para que possam sobreviver à catástrofe climática. Este ambiente deu origem a novas criaturas, consideradas místicas.

O personagem principal da narrativa é Thorin, um humano da tribo Ulvhir . Esta tribo tem como a sua habilidade inata, o ligação com o mundo espiritual.

O mundo se depara com uma catástrofe climática e Thorin se vê numa situação onde terá que derrotar os 6 Guardiões lendários e a Lendária *Phoenix* Congelada.

1.2. Problema

Apesar do crescimento e reconhecimento gradual da indústria de jogos *indie*, verifica-se uma escassez dos mesmos.

Por esta razão, este projeto visa contribuir para a criação de mais um título.

1.3. Resultados alcançados

Os principais resultados alcançados consistem na elaboração formal do presente relatório, desenvolvimento do jogo *Wild Soul*, e, finalmente, desenvolvimento do *site* para suporte aos jogadores.

1.4. Objetivos

Considerando o contexto e motivação, perante o problema identificado, definiu-se um conjunto de objetivos principais:

- **O1** – Analisar o enquadramento teórico do projeto;
- **O2** - Definir os casos de uso do projeto, i.e. identificação dos principais atores, processos e fronteiras do sistema;
- **O3** - Elaborar o modelo de domínio que traduza as relações estáticas das principais classes do projeto;
- **O4** - Definir a arquitetura do sistema;
- **O5** - Definir os requisitos das aplicações;
- **O6** - Definir mecanismos de persistência de dados;
- **O7** - Realizar o protótipo aplicacional.

1.5. Cronograma

Nesta secção ilustra-se o cronograma de tarefas executadas relativas ao projeto (Diagrama de Gantt, Figura 1). Assim, observa-se no cronograma:

- **Tarefa 1,** *Definição de Contexto/Motivação*;
- **Tarefa 2,** *Definição de Problema/Objetivos*;
- **Tarefa 3,** Analise d*o Enquadramento Teórico;*
- **Tarefa 4,** *Definição de Casos de Uso/Modelo de Domínio;*
- **Tarefa 5,** *Arquitetura do Sistema;*
- **Tarefa 6,** *Definição de Requisitos;*
- **Tarefa 7,** *Desenvolvimento do Protótipo; e,*
- **Tarefa 8,** *Elaboração do Relatório*.

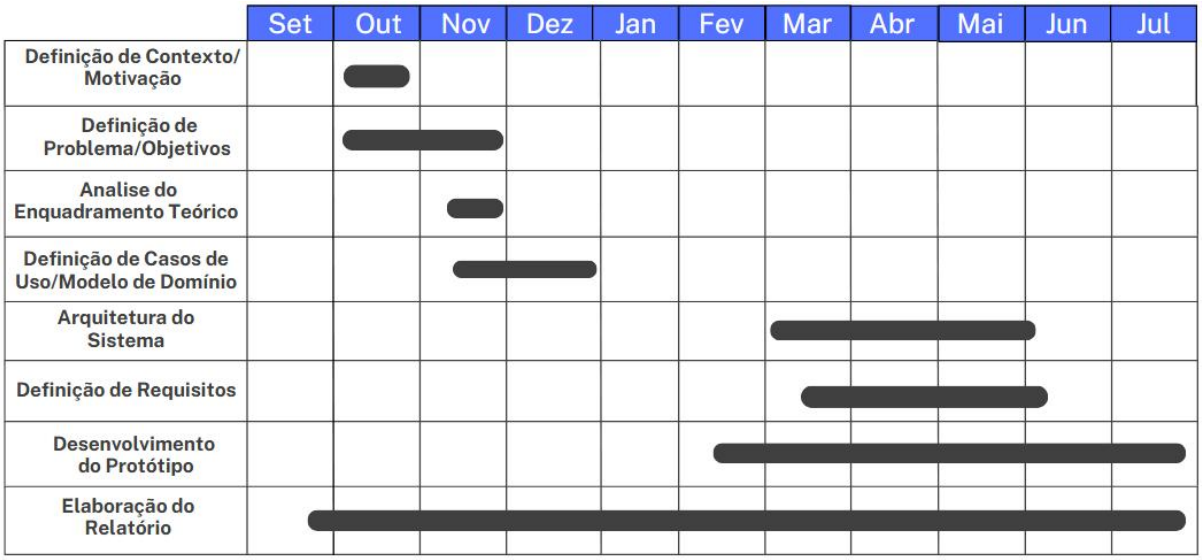

Figura 1 Cronograma

1.6. Estrutura do documento

Este documento encontra-se organizado em capítulos. Assim, o capítulo 1 refere-se à introdução, incluindo as secções: 1.1 Contexto e Motivação, 1.2 Problema, 1.3 Objetivos, 1.4 Cronograma, 1.5 Resultados Alcançados e 1.6 Estrutura do Documento.

O Capítulo 2 inclui o enquadramento teórico, através das secções: 2.1 Videojogos, 2.2 Metodologias de Análise e 2.3 Tecnologias Utilizadas.

O Capítulo 3 apresenta o Projeto *Wild Soul.* Este capítulo inclui as secções: 3.1 Casos de Uso do Projeto *Wild Soul*, 3.2 Modelo de Domínio do Projeto *Wild Soul*, 3.3 Arquitetura do Sistema, 3.4 Requisitos, 3.5 Persistência de dados, e 3.6 Protótipo.

Finalmente, o Capítulo 4 refere-se às conclusões preliminares do projeto, que inclui as secções: 4.1 Discussão e 4.2 Trabalho Futuro.

Página em branco

2. Enquadramento teórico

Neste capítulo, apresentam-se os principais elementos teóricos que visam dar suporte ao leitor para uma melhor compreensão do trabalho realizado. Na Secção 2.1, procede-se à caracterização do modelo de negócio para videojogos, abordando as suas principais características e os diferentes tipos existentes. De seguida, na Secção 2.2, são introduzidas as metodologias de análise utilizadas, incluindo os casos de uso, o modelo de domínio, e os requisitos funcionais e não funcionais. Finalmente, na Secção 2.3, abordam-se as tecnologias utilizadas no projeto, bem como as ferramentas adotadas para programação e desenvolvimento.

2.1. Videojogos

O modelo de negócio associado ao mercado de videojogos gera receitas muito significativas a nível mundial.

Os videojogos são pacotes de s*oftware* interativos com uma *interface* gráfica, que permite ao jogador controlar personagens e/ou elementos dentro de um ambiente virtual. Este ambiente digital promove uma experiência dinâmica, imersa e orientada por objetivos onde o jogador atua como a entidade ativa e interage através de uma *interface* visual e periféricos. Para alcançar esse propósito, os videojogos apresentam várias características, que incluem [CG, 24]:

- **Interatividade** - O jogador executa ações e comandos que influenciam a narrativa e a dinâmica do jogo possibilitado uma experiência ativa e personalizada;

- **Gráficos** - Os videojogos utilizam componentes visuais que variam em qualidade e estilo, desde gráficos bidimensionais (2D) até ambientes tridimensionais (*3D*). Além disso, existem gráficos que combinam os elementos de ambos os estilos (2.5D);

- **Sons** - A música e os efeitos sonoros complementam a experiência do videojogo, contribuindo para a criação de atmosferas e reforçando emoções e reações às ações do jogador;

- **Objetivos** - Cada jogo estabelece metas e/ou desafios que orientam o jogador, acrescentam uma estrutura motivacional e um sentido de progresso;

- **Regras** - As limitações e permissões de ações são definidas por um conjunto de regras, que estruturam a experiência e garantem a coerência do universo do jogo. De resto, existe uma área científica que se denomina *Teoria dos Jogos* [TM+,24].

2.1.1. Plataformas para Videojogos

As plataformas de videojogos são os dispositivos onde os jogos são executados e determinam a qualidade da experiência e a forma de interação. As três principais plataformas para videojogos incluem computadores, consolas e dispositivos móveis, cada uma com características únicas que influenciam o desempenho, os gráficos e o tipo de controlo [JC, 24]:

- **Computadores (PCs)** - Oferecem uma plataforma versátil e altamente configurável, permitindo jogos com gráficos de alta resolução e uma grande diversidade de controlos, como teclados, ratos e comandos. Os computadores permitem ampla personalização de *hardware*, o que pode melhorar significativamente o desempenho gráfico e a capacidade de processamento, sendo ideais para jogos mais pesados computacionalmente, que exigem alta qualidade visual e maior capacidade de recursos (e.g. memória RAM);

- **Consola de Videojogos** - *PlayStation 5*, *Xbox Series X*, *Nintendo Switch*, são exemplos de consolas usadas especificamente para videojogos, desenhados para oferecer uma experiência otimizada e consistente. Com *hardware* padronizado e comandos dedicados, permite ao jogador uma utilização simplificada, sem necessidade de configurações complexas;

- **Dispositivos Móveis** - *Smartphones* e *tablets*, são dispositivos móveis que possibilitam uma experiência de jogo portátil, com interações baseadas em ecrãs táteis e por vezes sensores de movimentos. Embora sejam menos poderosos que computadores e consolas, tipicamente, nos dispositivos móveis as sessões são mais curtas, idealmente para jogos casuais e rápidos. Estas plataformas são apreciadas pela conveniência e portabilidade, permitindo a experiência de jogo *anytime*/*anywhere*.

2.1.2. Videojogos *online* vs.*offline*

Os modos de jogo *online* e *offline*, oferecem experiências distintas e são adequados a diferentes perfis e preferências dos jogadores.

No modo *offline*, o jogador não depende de uma ligação à Internet para desfrutar do jogo. Normalmente este modo de jogo é mais focado na experiência individual e na exploração de elementos internos do mesmo, com a narrativa e desafios estabelecidos. Este modo de jogo é ideal para quem prefere um ritmo próprio, concentrando-se na progressão e na imersão sem depender de fatores externos (e.g. acesso à Internet).

Por outro lado, no modo de jogo *online*, é necessária ligação à Internet e pode proporcionar uma experiência de jogo coletiva e interativa, permitindo que os jogadores interajam entre si em tempo real. Este modo de jogo é normalmente caracterizado pela possibilidade de colaboração ou competição com outros jogadores, que frequentemente podem ser de outras regiões. Alem disso os videojogos *online* são habitualmente atualizados com novos conteúdos e desafios,

oferecendo aos jogadores uma experiência diversificada e em expansão. Todavia, a dependência de uma ligação à Internet pode implicar o risco de problemas de conectividade, segurança, latência e interrupções que podem afetar a fluidez da experiência [JC, 24].

2.1.3. Videojogos 2D, 3D e 2.5D

As dimensões gráficas dos videojogos, que incluem 2D, 3D e 2.5D, influenciam significativamente como o jogador irá interagir com o ambiente do jogo.

Videojogos de duas dimensões (2D) operam no eixo horizontal e vertical, proporcionando uma jogabilidade linear e normalmente simplificada. Esta estética gráfica permite um foco claro na mecânica de jogo e na narrativa, sendo mais comum em videojogos de género plataforma e de *puzzle*, onde a clareza visual e a acessibilidade são cruciais.

Por outro lado, os videojogos de três dimensões (3D) oferecem uma representação mais realista e imersiva, por operarem em três eixos – horizontal, vertical e de profundidade. Esta dimensão adicional permite a criação de ambientes complexos e dinâmicos. Videojogos com gráficos 3D são amplamente utilizados em géneros como ação, aventura e simulação, que acentua a sensação de presença e imersão no mundo virtual.

Contudo, a criação de videojogos 3D requer um *hardware* mais avançado, o que poderá limitar ou restringir o acesso de certos jogadores. A complexidade visual inerente a estes videojogos também implicará frequentemente um maior tempo de desenvolvimento, que resulta em experiências de utilização mais detalhadas e aprofundadas.

Como referido anteriormente (Secção 2.1.1), o conceito dos gráficos 2.5D, surge como uma intersecção entre o 2D e o 3D, combinando os elementos de ambas. Os videojogos 2.5D utilizam gráficos tridimensionais, mas limitando a jogabilidade a um plano bidimensional, assim proporcionando uma estética visual que preserva a simplicidade do 2D enquanto oferece a profundidade e os detalhes do 3D. Normalmente esta abordagem é frequentemente utilizada em jogos de género plataforma. Este tipo de conceito gráfico permite que os jogadores desfrutem de um ambiente visualmente atrativo, sem perder a clareza das mecânicas dos videojogos 2D [JC, 24].

2.1.4. Tipos de videojogos

A tabela seguinte resume diversos géneros e tipos de videojogos [CG, 4]. O género de vídeojogo deste projeto é **Aventura**.

Género	SubGénero	Descrição	Exemplo
Ação	Ação/Aventura	Exploração e combate	Uncharted Tomb Raider
	Arena shooter	Combate em arenas fechadas com ritmo rápido	Quake Unreal Tournament
	Beat' em up	Progresso em fase com combate contra múltiplos inimigos	Street of Rage Double Dragon
	Hack and Slash	Combate corpo-a-corpo com combos	Devil My Cry Bayonetta
	Hero Shooter	Personagens com capacidades únicas	Paladins Overwatch
	FPS	Tiro na primeira pessoa	Halo Valorant
Aventura	Point and *Click*	Resolução de Puzzles e Narrativas	Monkey Island Myst
	Text Adventure	Jogo narrativo usando texto como principal interação	Zork
	Visual Novel	Baseado em textos e ilustrações, com escolhas que afetam a narrativa	Doki Doki Literature Club
	Walking Simulator	Exploração e narrativa sem grandes desafios	FireWatch Gone home
Battle Royale	N/A	*Multiplayer* onde o último jogador ou equipa vence	Fortnite PUBG
Corrida	Corrida *Arcade*	Corrida com física simplificada	Need for Speed Mario Kart
	Simulação de corrida	Corrida realista e com física precisa	Forza Motorsport iRacing
Desporto	Desporto *Arcade*	Versão menos realista e mais acessível	NBAJam Rocket League
	Simulação de Desportos	Simulação realista do desporto	Madden NFL Pro Evolution Sccorer
Estratégia	Moba	Combate de equipa em mapas divididos em rotas	League of Legends Dota 2
	RTS (Real Time Strategy)	Estratégia em tempo real, com construções de base e combate	StarCraft Age of Empires
	TBS (Turn Based Strategy)	Estratégia baseada em turnos	Civilization Heroes of Migth and Magic
	Tower Defense	Defesa de pontos com torres e armadilhas	Plants vs. Zombies Blood TD
Musical e Rítmico	Jogos de Musica e ritmo	Sincronização com música	Guitar Hero Just Dance
***Party* e Casual**	Jogos casuais	Jogos simples e acessíveis	Bejeweled Crossy Road
	*Party game*s	Jogos para grupos ou família, com foco em diversão e interação	Mario Party JackBox Party Pack
Plataforma	Metroidvania	Exploração e aquisição de competências para avançar	Holo Knigth Castlevania: Symphony of the Nigth
	Plataforma 2D	Ação e saltos em cenários 2D	Super Mario Bros. Celest
	Plataforma 3D	Exploração e saltos em cenários tridimensionais	Super Mario 64 Crash Bandicoot

Género	SubGénero	Descrição	Exemplo
Puzzle	Logic Puzzlekk	Baseado em lógica	Portal The Witness
	Physics-Based Puzzle	Puzzle que envolve físico para resolver problemas	Angry Birds World of Goo
	Quebra-Cabeças Clássicos	Resolução de problemas	Tetris Candy Crush
RPG	JRPG (Japonese RPG)	Narrativa linear e combate baseado em turnos	Final Fantasy Persona
	MMORPG	RPG *online* de mundo aberto para milhares de jogadores	World of Warcraft Guild Wars
	RPG de Ação	Combate em tempo real com progressão de personagens	The Wicht Dark Souls
	RPG de Ocidental	Liberdade de escolha e mundo aberto	Skyrim Fallout
	Roguelike e Roguelite	Mapas diferentes com grau de dificuldade acima da média	Hades The Binding of Isaac
	Tatical RPG	RPG com combate em mapas táticos e por turnos	Fire Emblem XCOM
	RPG Narrativo	Jogos com foco em diálogos e decisões, muitas vezes com elementos de RPG.	Disco Elysion The Council
Simulação	Simulador de Construção/Gestão	Gerir recursos, cidades ou negócios	SimCity RollerCoaster Tycoon
	Simulador de Desporto	Simula desportos realistas como futebol e basquetbol	FIFA NBA 2K
	Simulador de Veículos	Condução visual	Fight Simulador Gran Turismo
	Simulador de Investigação	Jogos que simulam o trabalho de um detetive, como resolver casos, analisar pistas, ect.	Her Story Telling Lies
	Simulador de vida	Simula o dia a dia	The Sims Animal Crossing
Terror	Horror de Ação	Mistura elementos de terror com ação imensa	Dead Space The Evil Within
	Horror de Investigação	Focado em mistérios com elementos de terror e suspense	Phasmonphobia Call of Cthulhu
	Psychological Horror	Terror psicológico e narrativo	Amnesia Layers of Fear
	Survival Horror	Enfatizar a sobrevivência e atmosfera tensa	Resident Evil Silent Hill
VR e AR	Realidade Aumentada (AR)	Elementos do jogo integrados com o mundo real	Pokémon GO
	Realidade Virtual (VR)	Jogos desenvolvidos para óculos VR. com imensão total	Beat Saber Half-Life: Alyx

Tabela 1 Géneros de videojogos e respetivas características

2.1.5. *E-Sports* e aspetos de competitividade dos videojogos

Competições com base em jogos eletrónicos, também conhecidos como *e-sports*, referem-se a competições profissionais de videojogos de diversos géneros (ver tabela da secção anterior). Onde jogadores individuais ou equipas competem em videojogos com uma componente competitiva de alto nível. *E-sports* envolve uma vasta gama de géneros de jogos, como jogos de estratégia em tempo real, jogos de combate, jogos de tiro na primeira pessoa (*FPS*), jogos de luta, entre outros.

Estes tipos de competições podem ocorrer tanto em plataformas *online* como também em eventos presenciais. Frequentemente são transmitidas ao vivo, atraindo grandes audiências em várias partes do mundo.

No contexto competitivo *e-sports*, a prática envolve a organização de torneios e ligas, onde os participantes podem ser jogadores amadores e/ou jogadores profissionais que competem por prémios monetários e prestígio. A competição em *e-sports* é um campo onde competências técnicas, trabalho em equipa, capacidade de estratégia e reações rápidas são determinantes para o sucesso. Além disso, o ambiente competitivo dos *e-sports* é sustentado por uma infraestrutura robusta, que pode incluir: treinadores, analistas, gestores de equipas e outros profissionais que suportam o desenvolvimento dos jogadores e das equipas.

Os *e-sports*, ao longo dos últimos anos, tornou-se um fenómeno mundial, com uma crescente profissionalização do setor e uma enorme base de fãs. Plataformas como a *Twitch* e o *Youtube*, eventos como *League of Legends World Championship* do jogo *League of Legends* e *Major Championships* do jogo *CS2*, contribuem para visibilidade e o crescimento no cenário competitivo dos videojogos.

Torneios e competições de *e-sports* são reconhecidos pelo seu alto nível de organização, e interação entre jogadores, público e patrocinadores, sendo atualmente uma das formas de entretenimento mais consumidas na Internet a nível mundial [JC,24].

2.2. Metodologias

Nesta secção, são apresentados os casos de uso, o modelo de domínio e, por fim, os requisitos funcionais e não funcionais. Na secção 2.2.1, são definidos os diagramas de casos de uso, detalhando as interações entre os atores e o sistema. Seguidamente, na secção 2.2.2, apresenta-se o modelo de domínio, que ilustra a estrutura conceptual e as principais entidades envolvidas. Por fim, na secção 2.2.3, procede-se à especificação dos requisitos funcionais e não funcionais, estabelecendo as funcionalidades principais do sistema.

2.2.1. Casos de Uso

Os diagramas de casos de uso (em UML) são apresentações gráficas utilizadas em engenharia de s*oftware*, em contexto de análise de sistemas. Estes diagramas descrevem as interações entre os atores (utilizadores ou sistemas externos) e o sistema em desenvolvimento. O seus objetivos é mostrar as principais funcionalidades do sistema sobre a perspetiva do utilizador.

A tabela seguinte descreve os principais componentes de um diagrama de casos de uso (adaptado de [CG, 4]).

Componente	Descrição
Ator	Representa uma entidade externa que interage com o sistema (pode ser uma pessoa, ou outro sistema, e.g. *Hardware*). Podem existir relações de herança entre os atores
Caso de Uso	Representa uma funcionalidade ou comportamento do sistema que gera valor ao ator. Normalmente é representado por uma oval e inclui um verbo e um substantivo.
Relação	Linhas que conectam atores e casos de uso. Existem relações de inclusão extensão e generalização.
Sistema	Pode ser representado por um retângulo que inclui os casos de uso do sistema (define os limites do sistema)

Tabela 2 Principais componentes dos diagramas de casos de uso

A figura seguinte ilustra um exemplo de diagrama de casos de uso para um sistema *online* de reserva de bilhetes de avião (Fonte: [EDIO, 24]). Como se observa, existem dois atores: 1) Cliente; e, 2) Administrador. Cada processo associado ao ator é caracterizado por um verbo e substantivo. O cliente pode efetuar reservas, pesquisas e cancelamentos. O administrador pode efetuar cancelamentos e atualizações da informação de voo. Observa-se ainda, o título e a fronteira do sistema (retângulo).

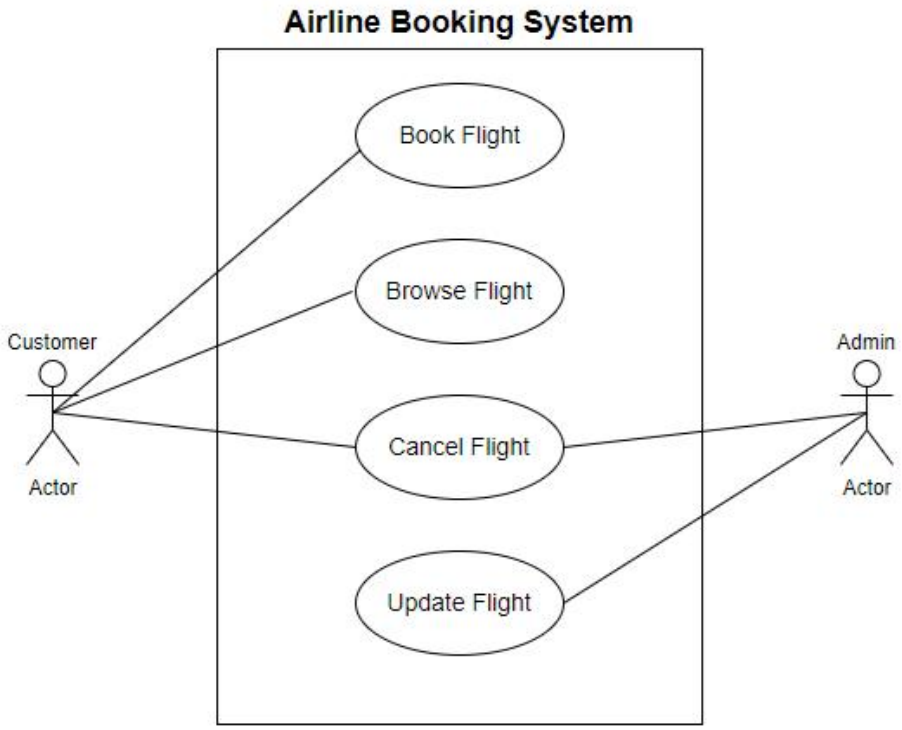

Figura 2 Caso de uso do Sistema de Reservas Online

2.2.2. Modelo de Domínio

O Modelo de Domínio é uma representação conceptual das entidades relevantes dentro de um determinado sistema ou área de interesse. Ele fornece uma visão clara dos principais conceitos e relacionamentos que compõem o domínio, funcionando como uma base fundamental para a compreensão e desenvolvimento

de um sistema. O modelo de domínio traduz as relações estáticas entre as principais classes do sistema e não se preocupa com a implementação técnica ou operações detalhadas, mas sim com os elementos essenciais do problema a ser resolvido. Ele é frequentemente representado por meio de diagramas de classes UML **(***Unified Modeling Language***)**, uma linguagem poderosa que permite visualizar entidades, seus atributos, e os relacionamentos entre elas [QN_Modelo, 24].

No contexto de desenvolvimento de s*oftware*, um Modelo de Domínio é utilizado para capturar o vocabulário do domínio e as interações entre os conceitos, o que facilita o entendimento entre a equipe técnica e as partes interessadas. Ele serve como um mapa conceptual que define as fronteiras do sistema e ajuda a alinhar as expectativas entre programadores e *stakeholders*. A criação de um modelo de domínio robusto permite evitar ambiguidades e lacunas de entendimento, além de ser um ponto de partida essencial para as fases de *design* e implementação do projeto [QN_Modelo, 24].

Como referido anteriormente, o diagrama de classes UML é uma das formas mais comuns de ilustrar um modelo de domínio. Ele organiza as entidades (classes) que representam conceitos ou objetos do mundo real, e os relacionamentos entre essas classes. Um diagrama de classes inclui classes, atributos das classes, e associações (relacionamentos) entre elas, representando a estrutura estática do sistema [QN_Modelo, 24].

A figura seguinte ilustra um diagrama de classes para o sistema "Escola" (Exemplo).

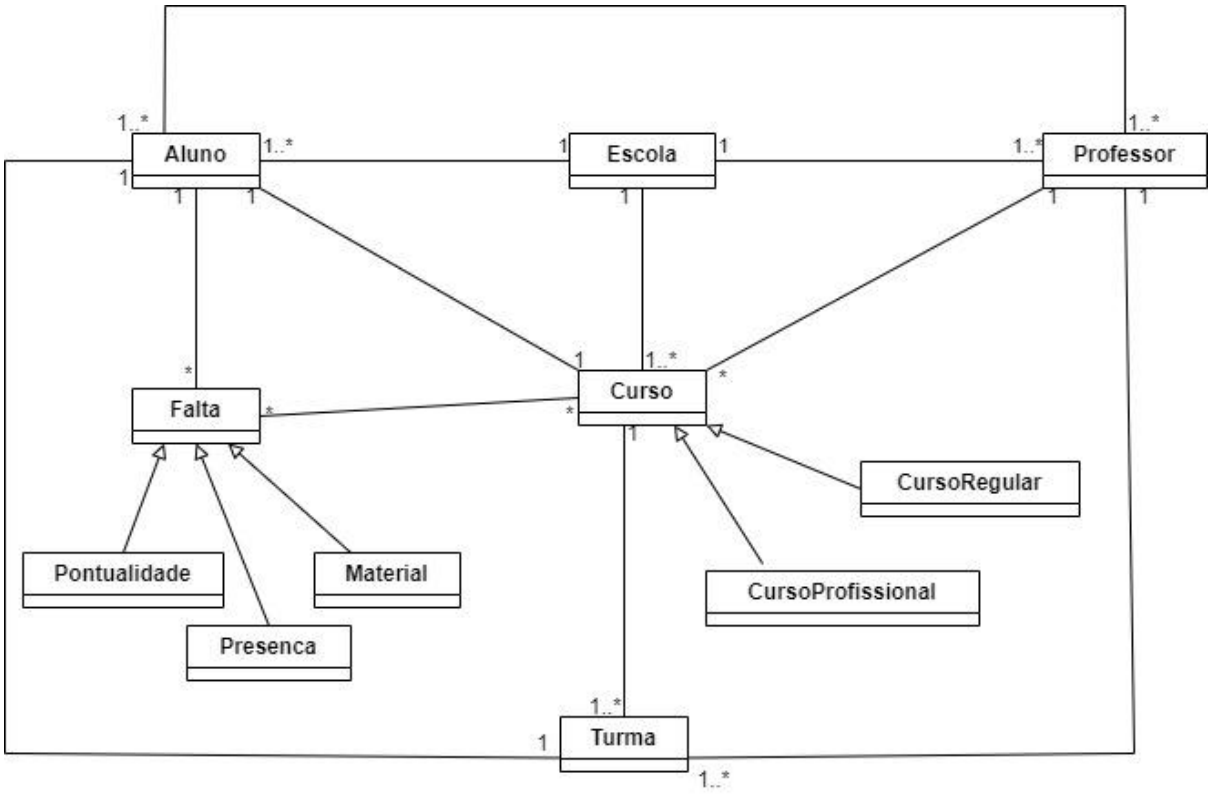

Figura 3 Modelo de Domínio UML do sistema "Escola"

Como se observa, o sistema é composto por várias classes principais que representam entidades no domínio educativo, como Escola, Aluno, Professor, Turma, e, Curso. As classes estão interligadas através de relacionamentos que refletem suas interações no sistema. A escola é responsável por manter cursos, agregar professores, agregar turmas, e gerir alunos. A classe Aluno representa um ou mais estudantes que estão matriculados na escola. Os alunos estão associados a uma turma e participam em diferentes cursos. Cada aluno está sujeito a um controle de presença, que pode ser registado através da classe Falta. Existem três tipos de faltas: pontualidade, presença e material. A classe Professor representa um ou mais docentes que pertencem á escola. Cada professor pode ser responsável por diversas turmas, que por sua vez têm vários alunos. As turmas também são caracterizadas em função do tipo de curso a que estão associadas, por exemplo, Curso Regular ou Curso Profissional.

Como referido anteriormente, o diagrama de classes UML é uma das formas mais comuns de ilustrar um modelo de domínio. Ele organiza as entidades (classes) que representam conceitos ou objetos do mundo real, e os relacionamentos entre essas classes. Um diagrama de classes inclui classes, atributos das classes, e associações (relacionamentos) entre elas, representando a estrutura estática do sistema [OS, 24].

2.2.3. Requisitos funcionais/não funcionais

Os requisitos funcionais definem o que o sistema deve fazer para resolver problemas e satisfazer as necessidades dos utilizadores. Incluem as ações, entradas e saídas, fluxos de trabalho, cálculos e comportamentos que o sistema deve ter de forma clara e objetiva. Ou seja, são relacionados ao que o sistema pode fazer. Tipicamente, os requisitos são numerados para facilitar a sua identificação e acompanhamento (eg: RF1, RNF1).

Eis alguns exemplos de requisitos funcionais:

- Inserir dados num formulário;
- Realizar compras*;*
- Selecionar opções num menu;
- Permitir que o utilizador procure produtos por categoria ou nome.

Os requisitos não funcionais, são os que estão relacionados ao que o sistema irá realizar o que está a ser planeado; envolvem aspetos tal como: usabilidade, desempenho, confidencialidade, manutenção, etc. Ou seja, enquanto os requisitos funcionais especificam o que está a ser feito pelo sistema, os não funcionais dizem como irão ser feitos.

Eis alguns exemplos de requisitos não funcionais [MW, 24], [CG, 4]:

- Tipo de Sistema Operativo;
- *Hardware* a ser utilizado;
- Disponibilidade do sistema;
- Os dados do utilizador serem encriptados usando algoritmos AES-256 (algoritmo de criptografia que usa uma chave de 256 *bits* para converter texto numa cifra) [PG,22].

A tabela seguinte resume as principais diferenças entre requisitos funcionais e não funcionais (tabela adaptada de [CG,4]):

Aspeto	Requisitos Funcionais	Requisitos não Funcionais
Foco	O que o sistema deve fazer.	Como o sistema se deve comportar.
Objetivo	Descrever as funcionalidades.	Definir restrições e qualidades.
Exemplo	Autorizar um registo de um utilizador	O registo ser realizado em menos de 2 segundos
Visibilidade	Geralmente visíveis pelo utilizador	Raramente vistos pelo utilizador

Tabela 3 Principais Diferenças Entre os Requisitos Funcionais e Não Funcionais

2.3. Tecnologias utilizadas

No presente capítulo, são abordadas as tecnologias utilizadas no desenvolvimento de um jogo 2D. O Godot foi a principal plataforma utilizada, integrando ferramentas como o Godot *2D IDE*, o Godot *2D Runner* e o Godot 2D Editor de Ativos, as quais desempenharam um papel essencial na simplificação do processo de desenvolvimento, organização e execução do jogo.

Para a criação de cenários e personagens, foi utilizado o *LibreSprite*, uma aplicação especializada em arte pixelizada (*Pixel Art*), que possibilitou a conceção de elementos visuais com um elevado nível de precisão e detalhe.

Adicionalmente, para a criação de um *site* de suporte ao jogo, foram utilizadas tecnologias *web* que complementaram o projeto, como o HTML, o CSS e o Javas*cript*, as quais contribuíram para o desenvolvimento de *interface*s dinâmicas e integradas, proporcionando uma experiência mais interativa e atrativa para os utilizadores. Paralelamente, recorreu-se à linguagem PHP para implementar funcionalidades associadas à gestão de dados e à comunicação entre sistemas, garantindo maior eficiência e robustez nas operações do projeto.

2.3.1. Godot

O Godot *engine* é uma plataforma gratuita e de código aberto para o desenvolvimento de jogos 2D e 3D. Reconhecida por sua flexibilidade e eficiência, o motor destaca-se principalmente em projetos 2D devido ao fluxo de processamento otimizado e ao conjunto robusto de ferramentas nativas para esse tipo de desenvolvimento [GDDC, 25].

Com suporte para múltiplas plataformas, o Godot permite criar jogos de qualidade profissional, enquanto promove acessibilidade e liberdade criativa para programadores de todos os níveis de experiência.

2.3.2. Godot 2D IDE

O ambiente de desenvolvimento integrado (IDE) do Godot *engine* é projetado para simplificar e acelerar a criação de jogos 2D. Ele fornece ferramentas abrangentes que cobrem todas as etapas do desenvolvimento, desde a concepção inicial até a exportação final.

- ***Interface* Intuitiva** - O IDE é personalizável, com janelas organizáveis e atalhos que facilitam o fluxo de trabalho*;*
- ***GDScript* e Suporte a Outras Linguagens** - O *GDScript*, a linguagem nativa do Godot, é fácil de aprender e ideal para iniciantes. Além disso, o motor suporta linguagens como C#, Visual*Script* e até C++*;*
- **Editor de Cena (*Scene Editor*)** - Um dos principais recursos do Godot é o sistema de cenas, que permite estruturar o jogo em nós reutilizáveis, que facilita a gestão e a modularidade do projeto.

A *interface* do Godot é intuitiva, oferecendo ferramentas visuais e opções para personalização, sendo uma solução versátil tanto para iniciantes quanto para programadores experientes.

A figura seguinte ilustra o IDE do Godot ([GDDC, 25]).

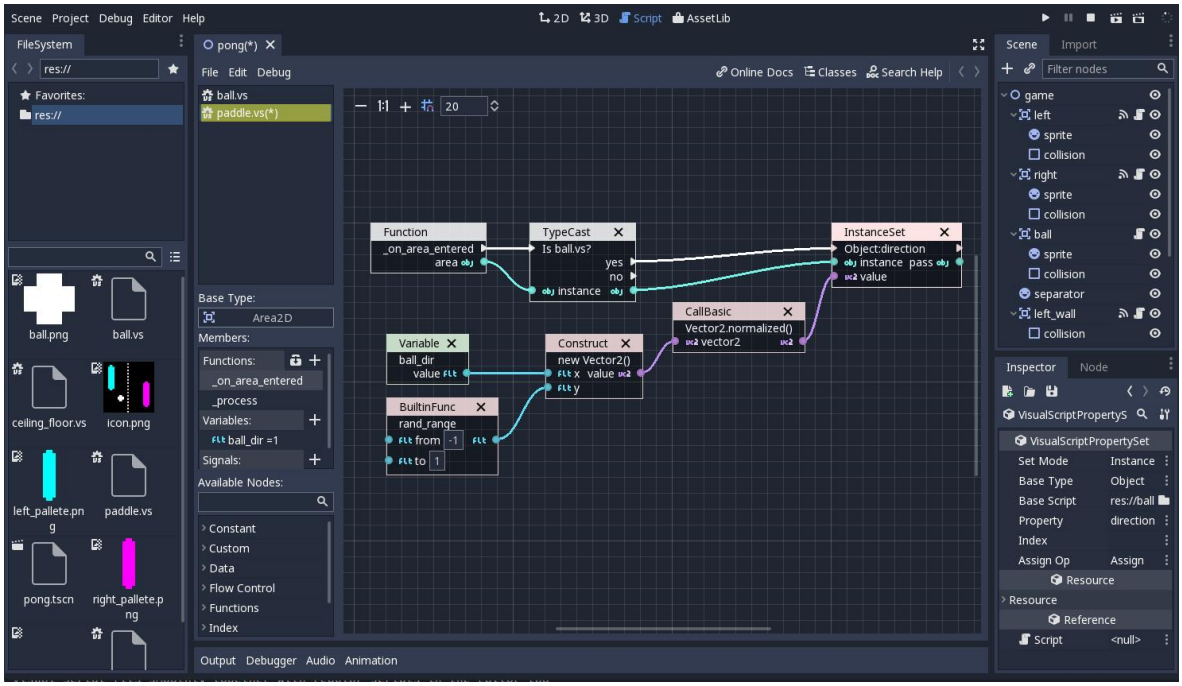

Figura 4 Exemplo Godot IDE 2D

2.3.3. Godot 2D Runner

O Godot 2D *Runner* refere-se a exemplos e estruturas predefinidas disponíveis para auxiliar programadores na criação de jogos de plataforma, *endless runners* e outros gêneros em 2D.

O Godot oferece uma variedade de sistemas pré-fabricados que incluem *scripts* prontos para movimentação de personagens, deteção de colisões, física e geração de níveis. Estes sistemas são altamente personalizáveis graças à integração com o sistema de nós, que permite aos utilizadores modificar as funcionalidades padrão e ajustá-las às necessidades específicas de cada projeto. Além disso, o motor gráfico é ideal para prototipagem rápida, oferecendo recursos de teste e a iteração de mecânicas de jogo, acelerando o processo de *design* e desenvolvimento [GDDC, 25].

2.3.4. Godot 2D Editores de Ativos

Os editores de ativos do Godot *Engine* são ferramentas integradas ao IDE que simplificam o processo de criação e gestão de recursos visuais e sonoros para jogos.

- **Editor de *Sprites*** - Suporte nativo para importar e configurar *sprites*, com funcionalidades para atlas de texturas e animações;

- **Animação 2D** - O editor de animações do Godot permite criar sequências quadro a quadro ou interpoladas diretamente no motor, garantindo movimentos suaves e personalizáveis;

- **Gestão de Áudio** - Ferramentas para manipulação e integração de sons e músicas, incluindo suporte a mixagem em tempo real e efeitos de áudio.

Estes editores proporcionam uma abordagem integrada e eficiente para a gestão de recursos, otimizando o fluxo de trabalho e minimizando a necessidade de utilização de ferramentas externas [GDDC, 25].

2.3.5. LibreSprite

O *LibreSprite* é uma aplicação de código aberto, concebida para a criação de arte pixelizada e animações 2D [CG, 4].

Amplamente utilizado por artistas digitais e programadores de jogos, o LibreSprite destaca-se pela sua facilidade de interação através de ferramentas especializadas que simplificam a criação de gráficos em estilo retro e animações fluídas, tornando-se uma escolha de referência para projetos relacionados com jogos e ilustrações.

Este *software* surgiu como um *fork* do Aseprite, preservando muitas das funcionalidades essenciais do programa original, mas disponibilizando-as de forma totalmente gratuita e com o apoio de uma comunidade dedicada ao código aberto.

À semelhança do Aseprite, o LibreSprite oferece recursos como camadas, paletas de core, suporte a transparência e exportação de animações em formatos como *GIF* ou *spritesheets*, destacando-se como uma alternativa acessível e livre de custos.

Embora frequentemente comparado ao Aseprite, o LibreSprite diferencia-se por ser uma solução gratuita e robusta, ideal para utilizadores que procuram um *software* para a criação de *Pixel Art*, mantendo o compromisso com a acessibilidade e a partilha colaborativa. [CG, 4].

A figura seguinte ilustra a *Interface* do ambiente de trabalho do LibreSprite.

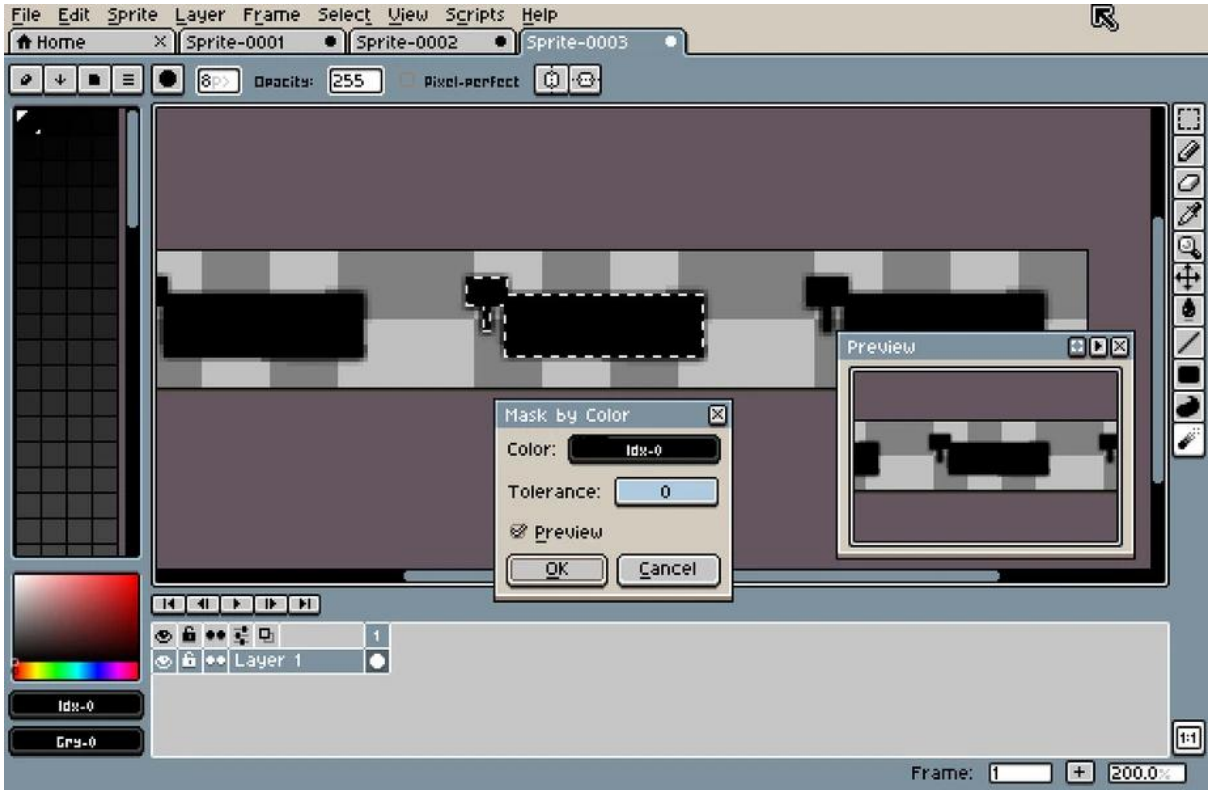

Figura 5 Exemplo do ambiente de trabalho do LibreSprite

2.3.6. Audacity

O Audacity é um *software* de código aberto amplamente reconhecido para gravação e edição de áudio digital [AC,25]. Disponível para Windows, macOS e Linux, destaca-se pela sua *interface* intuitiva, ampla gama de ferramentas e totalmente gratuito.

Entre as suas funcionalidades principais incluem-se a gravação de áudio de múltiplas fontes, edição precisa de faixas e aplicação de efeitos como equalização, redução de ruído e compressão. O Audacity suporta vários formatos de áudio, como WAV, MP3, FLAC e AIFF, permitindo tanto a importação como a exportação de ficheiros com elevada qualidade.

Sustentado por uma comunidade ativa, o Audacity mantém-se como uma solução robusta e acessível para projetos diversos, incluindo música, *podcasts* e narrações [AC,25].

A figura seguinte ilustra a *Interface* do ambiente de trabalho do Audacity.

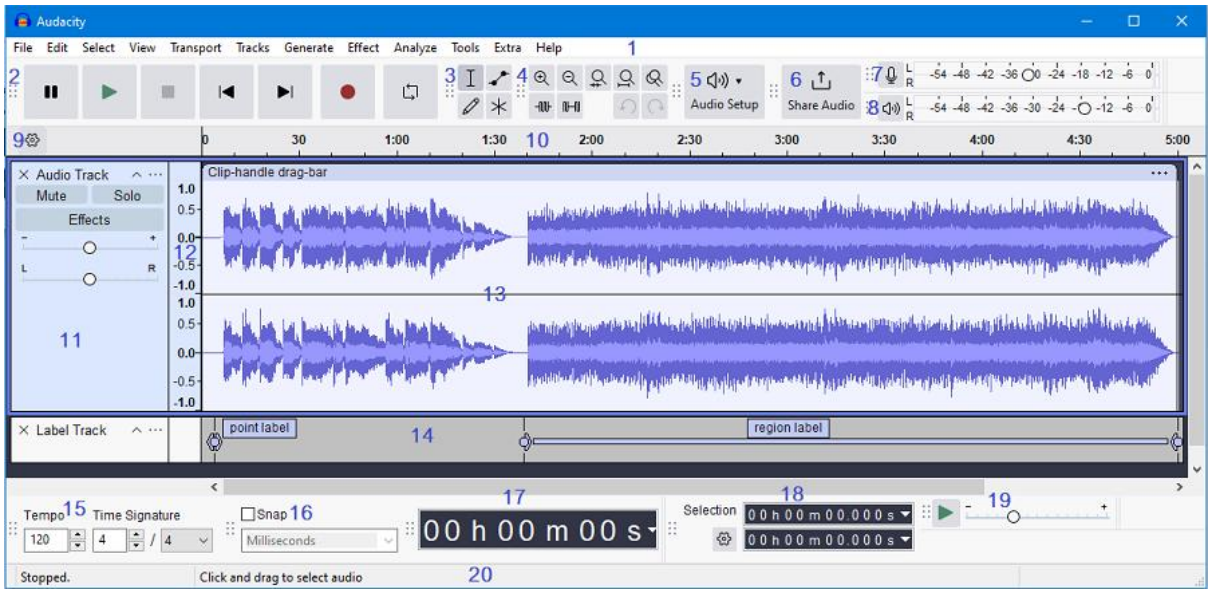

Figura 6 Exemplo do ambiente de trabalho do Audacity

2.3.7. Discord

O Discord é uma plataforma de comunicação digital gratuita amplamente utilizada para mensagens de texto, chamadas de voz e vídeo, e partilha de ficheiros. Disponível para Windows, macOS, Linux, iOS e Android, destaca-se pela sua versatilidade e *interface* intuitiva.

Inicialmente concebido para a comunidade *gamer*, o Discord evoluiu para atender a uma ampla variedade de utilizadores, incluindo contextos educativos, trabalho remoto e socialização. Os servidores, organizados em canais de texto e voz, oferecem uma estrutura eficiente e altamente personalizável. Entre as suas principais funcionalidades incluem-se a utilização de *bots* automatizados, partilha de ecrã, transmissões em direto e integração com serviços externos, como o Spotify e o Twitch.

A plataforma garante segurança e privacidade aos utilizadores, que disponibiliza opções avançadas de moderação e autenticação de dois fatores. Apoiado por atualizações regulares e uma comunidade dinâmica, o Discord apresenta-se como uma solução robusta para comunicação e colaboração em grupo [DC, 25].

A figura seguinte ilustra a interface de uma comunidade no Discord.

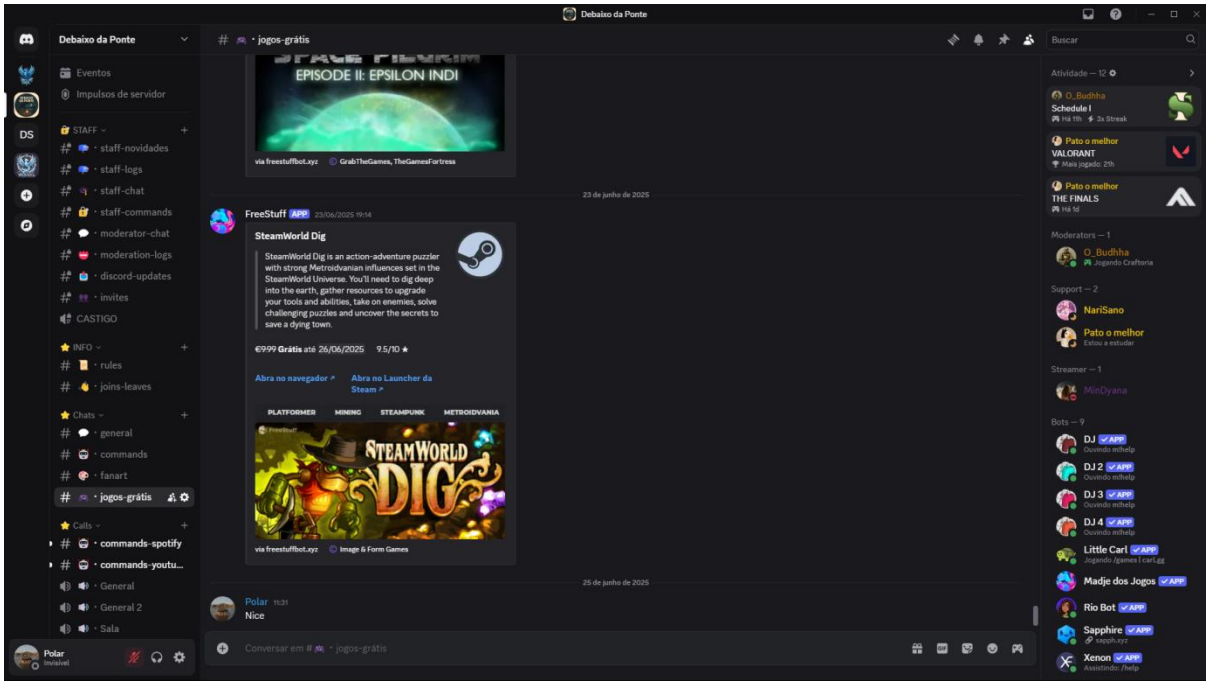

Figura 7 Exemplo do ambiente de uma comunidade do Discord

2.3.8. HTML5 (*Hyper Text Markup Language, versão 5*)

O HTML5 é uma linguagem de marcação utilizada para estruturar o conteúdo de *site*. Sua principal função é organizar textos, imagens, *links* e outros elementos em uma hierarquia lógica, que permite que navegadores interpretem e exibam essas informações de forma estruturada [W3HTML5, 25].

2.3.9. CSS3 (*Cascading Style Sheets*, versão 3)

O CSS3 é uma linguagem de estilos usada para controlar a aparência visual de documentos HTML.

Com a tecnologia CSS3, é possível definir e reutilizar *layouts*, paletas de cores, tipografias e aplicar efeitos visuais, como transições e animações. O CSS3 desempenha um papel essencial na criação de *interfaces* visuais esteticamente agradáveis e adaptáveis a diferentes dispositivos [W3CSS3, 24].

2.3.10. PHP (*Hypertext Preprocessor*)

O PHP é uma linguagem de programação de código aberto vocacionada para o desenvolvimento no lado servidor [PHPGRP, 25].

PHP permite processar dados de formulários, executar lógica de programação e conectar-se com bases de dados para gerar páginas dinâmicas de conteúdo personalizado. O PHP é amplamente utilizado para construir aplicações que integram soluções algorítmicas com bases de dados [PHPGRP, 25].

2.3.11. Java*Script*

O Javas*cript* é uma linguagem de programação interpretada que opera no lado cliente e, em alguns casos, no lado servidor. A sua principal aplicação é adicionar interatividade às páginas da web, facto que permite manipulação em tempo real de elementos visuais, validações de dados e comunicação assíncrona com servidores via APIs [W3JS, 25].

2.3.12. MySQL

O MySQL é um Sistema de Gerenciamento de Banco de Dados (SGBD) que utiliza a linguagem SQL para realizar consultas. Propriedade da Oracle. Sua portabilidade permite a implementação em uma ampla gama de plataformas e sistemas operacionais, compatibilidade com linguagens de programação como PHP, Java, C, Python, Ruby. Entre suas principais funcionalidades estão o suporte a *Unicode*, índices de texto completo, replicação, *backup* em tempo real, controle transacional, *triggers*, cursores, *stored procedures* e funções, proporcionando robustez e escalabilidade a qualquer tipo de aplicação[OC, 25].

2.3.13. Visual Studio Code (VS *Code*)

É um editor de código-fonte desenvolvido pela Microsoft, amplamente utilizado no setor de tecnologia para programação [VSM, 25].

O VS *Code* permite o desenvolvimento em várias linguagens de programação e oferece funcionalidades como realce de sintaxe, auto-completar inteligente, depuração integrada e um terminal embutido. Além disso, possui uma vasta biblioteca de extensões que permite personalizar o ambiente de desenvolvimento e integrar ferramentas adicionais, como suporte a *frameworks* e controle de versão via Git.

O Visual Studio *Code* é compatível com sistemas Windows, macOS e Linux.

A figura seguinte ilustra a *Interface* do Visual Studio *Code* onde se observa a barra de atividades (A), barra principal (B), grupos de edição (C), painel de execução (D), e finalmente a barra de estado (E).

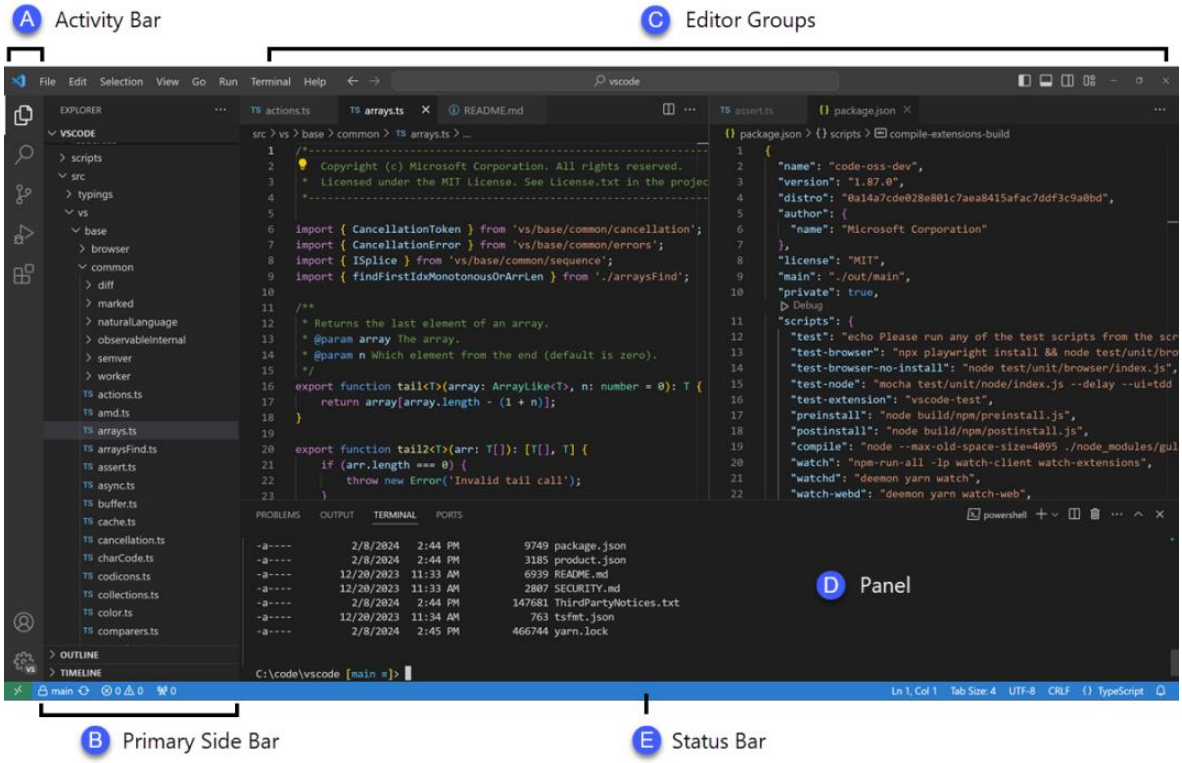

Figura 8 Exemplo da Interface do Visual Studio Code

2.3.14. Plataforma para alojamento do projeto

O InfinityFree é um serviço de alojamento *web*. Pode disponibilizar 5 GB de espaço em disco, largura de banda ilimitada, 400 bases de dados MySQL e certificados SSL gratuitos, garantindo segurança e desempenho para *sites*. Os utilizadores podem usar um subdomínio gratuito ou um domínio próprio [IF, 24]. A figura seguinte ilustra uma captura de ecrã da página InfinityFree com um resumo de características técnicas desta plataforma de alojamento gratuito.

Figura 9 Características técnicas do serviço de alojamento InfinityFree

3. Projeto Wild Soul

Este capítulo apresenta os principais aspetos técnicos e funcionais do sistema desenvolvido. São descritos os Casos de Uso (secção 3.1), o Modelo de Domínio (3.2) e a Arquitetura do Sistema (3.3), bem como os Requisitos Funcionais e Não Funcionais (3.4).

A Persistência de Dados (3.5) detalha o armazenamento de informações dos utilizadores do *site*.

Por fim, o Protótipo (3.6) é dividido entre o Jogo (3.6.1), com características gerais, navegação, cenário e exemplos de código, e a página de suporte aos jogadores (3.6.2), abordando características gerais, esquema navegacional e exemplos de código.

Este capítulo estrutura assim a base técnica e funcional do projeto.

3.1. Casos de Uso do Projeto Wild Soul

A figura seguinte ilustra o diagrama de casos de uso do *site* de suporte ao jogo.

Como se observa, existem quatro atores: 1) Anónimo, 2) Registado, 3) Programador e 4) Jogador.

O anónimo pode aceder notícias, registar-se e fazer *download* do jogo. O registado pode fazer *login*, interagir com a comunidade além de participar em sessões de perguntas e respostas com os programadores e fazer doações. O programador pode inspecionar contas, verificar punições e *bugs*, além de gerir contas, pode alterar dados relativos à conta do usuário. O jogador pode começar e carregar o jogo , selecionar opções e gerir créditos.

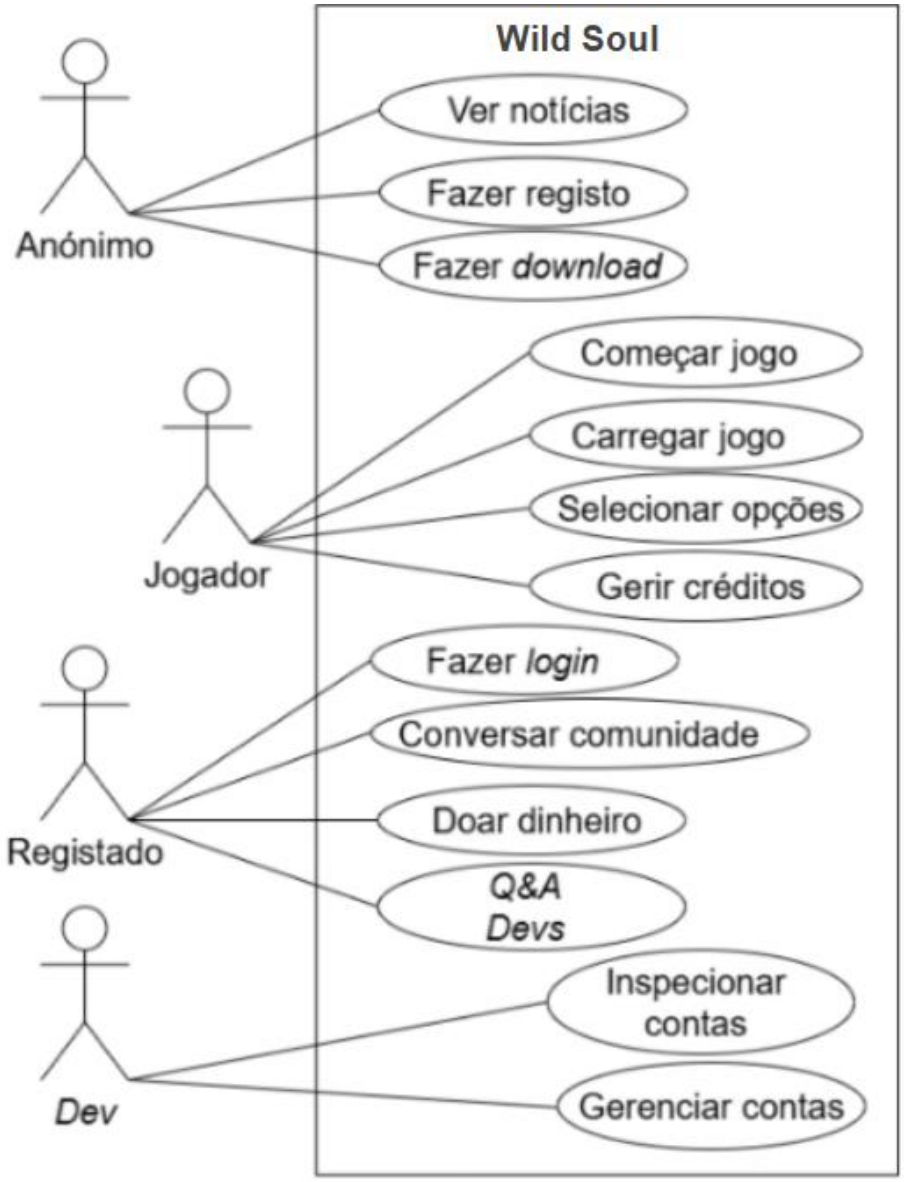

Figura 10 Caso de Uso do projeto Wild Soul

3.2. Modelo de Domínio do Projeto *Wild Soul*

No diagrama apresentado nesta secção, observa-se o modelo de domínio para o *site* e o jogo *Wild Soul*. As duas classes principais são *site* e Jogo.

O *site* possui diferentes tipos de utilizadores, incluindo Utilizador Anónimo, Utilizador Registado e Programador. Este também contém as classes Comunidade e Notícia. Somente o Utilizador Registado e o Programador têm acesso à Comunidade, enquanto todos os tipos de utilizadores têm acesso à Notícia.

O Jogo é composto por diversas classes, como Criaturas, subdivididas em Criaturas Passivas e Criaturas Hostis. Outras classes incluem Personagem, *Player*, *non-player character* (NPC) e História, que define a narrativa do jogo.

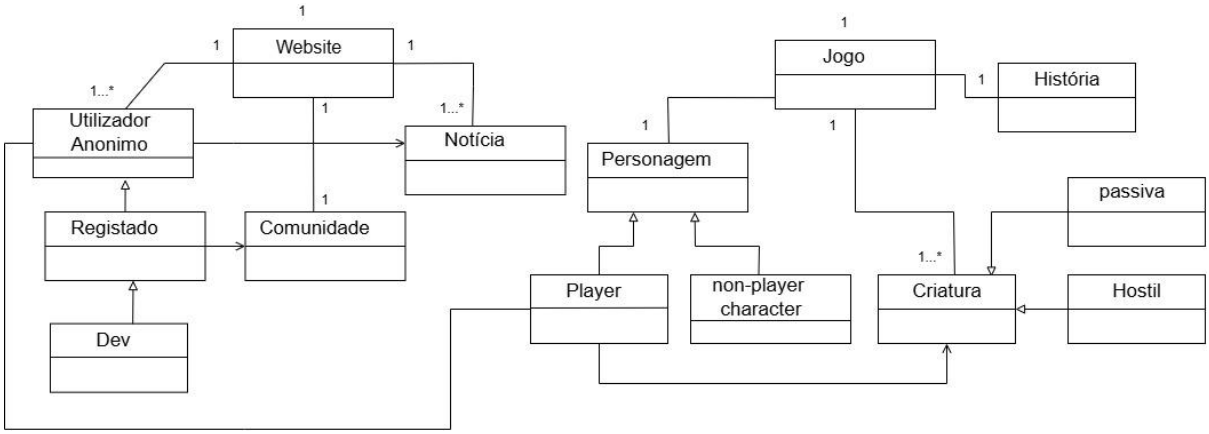

Figura 11 Modelo de Domínio do projeto Wild Soul

3.3. Arquitetura do Sistema

A arquitetura do sistema desempenha um papel fundamental no desenvolvimento e funcionamento de um projeto, definindo a forma como os componentes interagem entre si e com o ambiente externo. Nesta secção, são descritas as duas principais vertentes do sistema: o *site*, que segue um modelo cliente-servidor para a interação com o utilizador, e o jogo, desenvolvido na *Engine* Godot e baseado numa arquitetura modular que assegura flexibilidade e eficiência. Cada uma destas vertentes é detalhada nas secções seguintes, destacando os seus elementos constituintes e o fluxo de trabalho associado.

3.3.1. *Site*

A figura seguinte representa a arquitetura cliente-servidor do *site*, onde um cliente acede ao sistema através de um *browser*, enviando pedidos HTTP para um servidor. O servidor, utiliza PHP, HTML, CCS3 e MySQL, processa esses pedidos no seu servidor aplicacional e interage com a base de dados (BD) para obter ou armazenar informações. Após o processamento, o servidor responde ao cliente com os dados solicitados, permitindo a navegação e a interação com o *site*.

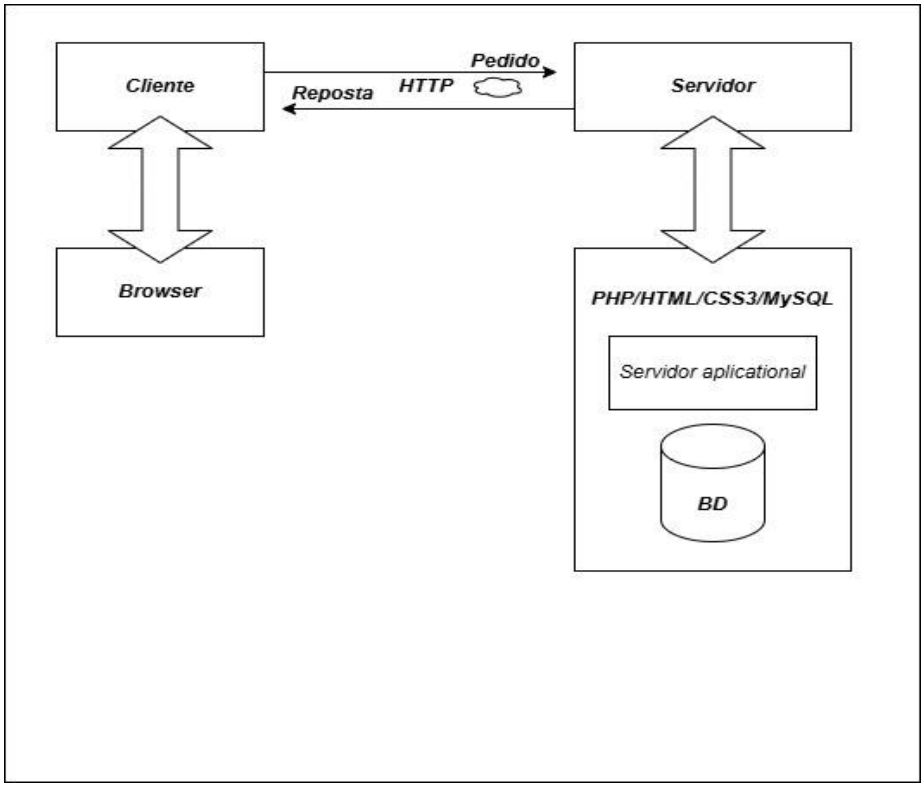

Figura 12 Arquitetura do Site

3.3.2. Jogo

A arquitetura do sistema do Godot é composta por quatro componentes principais: *Core*, *Scene*, *Servers* e *Drivers*, cada um desempenhando um papel essencial no funcionamento da *Engine*, assegurando flexibilidade e modularidade no desenvolvimento de jogos e aplicações.

O *Core* constitui o núcleo central da *Engine*, responsável por disponibilizar as funcionalidades fundamentais que suportam todo o sistema. Este componente gere recursos como texturas, modelos e *Scripts*, assegurando o seu carre*gamen*to e manipulação eficientes. Adicionalmente, o *Core* implementa um sistema de sinalização que facilita a comunicação entre os nós através de eventos. Este componente inclui ainda classes fundamentais, como vetores, matrizes e *strings*, indispensáveis para cálculos matemáticos, operações de manipulação de dados e tarefas de baixo nível.

O componente *Scene* é responsável por estruturar o conteúdo da *Engine*, organizando-o numa hierarquia de nós. Este sistema permite que os programadores criem e reutilizem cenas e nós como elementos modulares, promovendo uma abordagem eficiente à construção de projetos. O *Scene* disponibiliza também ferramentas no editor, que facilitam a criação e a visualização da hierarquia de nós, assegurando um fluxo de trabalho intuitivo e organizado.

Os *Servers* são componentes dedicados à gestão dos sistemas subjacentes da *Engine*. Estes incluem o *Render Server*, que gere os gráficos renderizados; o *Physics Server*, que processa as simulações físicas, como colisões e efeitos

gravitacionais; o *Audio Server*, responsável pela mistura e reprodução de áudio; e o *Navigation Server*, que realiza cálculos relacionados com a navegação de personagens e objetos. Estes servidores operam de forma independente, garantindo uma execução eficiente dos diferentes aspetos técnicos do projeto.

Por último, os *Driver*s constituem a camada de *interface* entre a *Engine* e o *Hardware*. Estes *Driver*s permitem a comunicação com APIs gráficas, como OpenGL e Vulkan, dispositivos de áudio e periféricos de entrada, garantindo ainda a compatibilidade com diversos sistemas operativos e dispositivos móveis. Esta camada é essencial para assegurar que o Godot funcione de forma consistente em múltiplas plataformas.

A figura seguinte ilustra arquitetura, composta pelos componentes *Core*, *Scene*, *Servers* e *Drivers*, oferece uma estrutura robusta e flexível, permitindo ao Godot atender às exigências de programadores, independentemente da complexidade dos seus projetos.

Figura 13 Arquitetura de Sistema

3.4. Requisitos

Na Secção 3.4, procede-se à identificação e descrição dos requisitos funcionais para o desenvolvimento do jogo e do respetivo *site* de suporte. Esta secção encontra-se dividida em duas subsecções principais para melhor organização.

Na Secção 3.4.1, são apresentados os requisitos funcionais, detalhando as funcionalidades específicas que o jogo e o *site* devem oferecer para garantir o cumprimento dos seus objetivos e a satisfação das necessidades dos utilizadores.

De seguida, na Secção 3.4.2, são descritos os requisitos não funcionais, que incluem aspetos como desempenho, segurança, escalabilidade, entre outros fatores para otimizar o jogo e o *site*.

3.4.1. Requisitos funcionais

As tabelas seguintes identificam respetivamente os principais requisitos funcionais para o jogo *Wild Soul* e *site*. O prefixo “RFJ” refere-se a requisitos funcionais do jogo, O prefixo “RFWS” refere-se a requisitos funcionais do *site*.

Na primeira tabela reúnem-se requisitos essenciais para o desenvolvimento do jogo *Wild Sou.* Estes requisitos estão organizados em cinco categorias principais:

- **Mecânicas do jogo (RFJ-01 a RFJ-14)**: Inclui o combate em tempo real, exploração do mapa aberto, sistema de *Loot*, desafios, interação com o ambiente inimigo;
- **História e Narrativa (RFJ-15 a RFJ-21)**: Abrange a narrativa linear, *NPC*s com missões e diálogos, bem como gestão do inventário;
- **Exploração e ambiente (RFJ-22 a RFJ-28)**: Contempla o mapa, clima, criaturas, arquitetura e elementos culturais relacionados com a mitologia nórdica;
- **Progressão e Personagem (RFJ-29 a RFJ-31)**: Trata do controlo do personagem Thorin, personalização de armas e armaduras, evolução e desbloqueio competências ofensivas.

ID	Descrição
RFJ-01	Implementar combate em tempo real com ataques e esquivas.
RFJ-02	Permitir exploração livre de todo o mapa com restrições de progresso.
RFJ-03	Adicionar um sistema de *Loot* para recolher armas, artefatos e recursos.
RFJ-04	Incorporar um sistema de combate.
RFJ-05	Introduzir inimigos com padrões de ataque variados.
RFJ-06	Criar baús espalhados pelo mapa.

ID	Descrição
RFJ-07	Permitir interação com elementos do ambiente, como destruir obstáculos.
RFJ-08	Adicionar Inimigos Chefe em áreas específicas do mapa.
RFJ-09	Implementar um sistema de energia para ataques e movimentos.
RFJ-10	Adicionar um inventário limitado para gerir recursos e itens.
RFJ-11	Adicionar um diário de missões para acompanhar o progresso.
RFJ-12	Incluir missões secundárias com recompensas.
RFJ-13	Implementar um sistema de economia simples para trocas com *NPC*s.
RFJ-14	Adicionar zonas de descanso onde o jogador pode recuperar vida.
RFJ-15	Criar uma narrativa linear com progressão em capítulos.
RFJ-16	Introduzir *NPC*s com diálogos para dar missões e informações.
RFJ-17	Permitir que os *NPC*s atualizem os diálogos conforme o progresso do jogador.
RFJ-18	Criar diálogos que forneçam contexto sobre os Guardiões.
RFJ-19	Criar missões principais focadas em encontrar artefactos dos Guardiões.
RFJ-20	Incorporar eventos narrativos nos quais Thorin interage diretamente com os Guardiões.
RFJ-21	Incluir um prólogo interativo que apresente Thorin e o mundo.
RFJ-22	Implementar ciclos de dia e noite.
RFJ-23	Adicionar mudanças climáticas dinâmicas.
RFJ-24	Adicionar pontos de interesse no mapa, como ruínas ou vilas abandonadas.
RFJ-25	Implementar animais selvagens no ambiente para caçar ou evitar.
RFJ-26	Criar um sistema de navegação com marcas no mapa personalizáveis.
RFJ-27	Adicionar um sistema de viagem rápida entre pontos.
RFJ-28	Implementar um sistema de rastreamento para encontrar tesouros escondidos.
RFJ-29	Adicionar um sistema de níveis para o jogador.
RFJ-30	Permitir troca de equipamentos com *NPC*s.
RFJ-31	Adicionar habilidades passivas.
RFJ-31	Introduzir melhorias cosméticas para armas e armaduras.

Tabela 4 Requisitos Funcionais do jogo

Na segunda tabela reúnem-se os requisitos essenciais para a construção do *site*.

ID	Descrição
RFWS-01	O utilizador anónimo deve poder aceder à página inicial.
RFWS-02	O utilizador anónimo deve poder visualizar a lista de notícias.
RFWS-03	O utilizador anónimo deve poder iniciar o processo de registo.
RFWS-04	O utilizador deve poder registar-se com *e-mail* e senha.
RFWS-05	O utilizador anónimo deve poder descarregar o jogo.
RFWS-06	O utilizador deve poder criar conta fornecendo nome, *e-mail* e senha.
RFWS-07	O utilizador deve poder fazer *Login* com *e-mail* e senha.
RFWS-08	O sistema deve manter a sessão autenticada até que o utilizador proceda ao *logout* ou ocorra expiração.
RFWS-09	O utilizador autenticado deve poder editar o perfil (nome, avatar, preferências).
RFWS-10	O utilizador deve poder efetuar *logout*.
RFWS-11	A página inicial deve exibir um menu de navegação.
RFWS-12	A página inicial deve incluir um vídeo demonstrativo do jogo.
RFWS-13	A página inicial deve apresentar um texto explicativo sobre a proposta e ambientação do jogo.
RFWS-14	A página inicial deve destacar as três últimas notícias com título, resumo e data.
RFWS-15	A página de novidades deve listar todas as atualizações do jogo, ordenadas da mais recente para a mais antiga.
RFWS-16	A página de novidades deve exibir título, data de publicação e resumo para cada notícia.
RFWS-17	A página de comunidade deve incluir um texto introdutório explicando o servidor presente na plataforma do Discord.
RFWS-18	A página de comunidade deve apresentar imagens ou ícones representativos do Discord.
RFWS-19	A página de comunidade deve disponibilizar um botão para aceder ao servidor do Discord oficial.
RFWS-20	O botão de acesso ao Discord deve abrir o convite num separador novo do navegador.
RFWS-21	A página deve indicar que toda a interação decorre no Discord e não diretamente no *site.*
RFWS-22	A página de *download* deve fornecer links diretos para as versões Windows, macOS e Linux.
RFWS-23	A página de *download* deve exibir o tamanho do ficheiro e a data da última versão disponibilizada.
RFWS-24	Todas as páginas devem ser responsivas, garantindo boa usabilidade em dispositivos móveis e *desktop*.
RFWS-25	Todos os botões e *links* devem ter estados de foco e *hover* visíveis para navegação por teclado.

Tabela 5 Requisitos Funcionais do Site

3.4.2. Requisitos não funcionais

As tabelas seguintes identificam os principais requisitos não funcionais para o jogo e para o *site*, respetivamente. O prefixo “RNFJ” refere-se a requisitos não funcionais do jogo, enquanto o prefixo “RNFWS” refere-se a requisitos não funcionais da *site*.

Na primeira tabela reúnem-se os requisitos não funcionais do jogo *Wild Soul*.

ID	Descrição
RNFJ-01	O jogo deve garantir um mínimo de 60 *FPS* em configurações médias em PCs recomendados.
RNFJ-02	Deve suportar taxas de *FPS* desbloqueadas para monitores de alta frequência (120 Hz ou mais).
RNFJ-03	O jogo será desenvolvido exclusivamente para Windows 10 ou superior.
RNFJ-04	Deve funcionar em sistemas com arquitetura de 64 *bits*.
RNFJ-05	Deve suportar monitores com múltiplas resoluções, incluindo 1080p, 1440p e 4K.
RNFJ-06	O sistema operativo deve proteger os ficheiros principais contra modificações de utilizadores sem privilégios de escrita.
RNFJ-07	Os ficheiros de texto devem ser armazenados localmente em formato encriptado.
RNFJ-08	O tamanho total do jogo instalado não deve ultrapassar 20 GB.
RNFJ-09	O processo de desinstalação deve remover todos os ficheiros associados ao jogo.

Tabela 6 Requisitos Não Funcionais do jogo

Na segunda tabela reúnem-se os requisitos não funcionais do s*ite*.

ID	Descrição
RNFWS-01	O sistema deve suportar até 10 000 utilizadores em simultâneo.
RNFWS-02	A página deve carregar até 3 segundos.
RNFWS-03	Toda a comunicação entre cliente e servidor deve realizar-se via HTTPS.
RNFWS-04	Os dados pessoais devem ser armazenados encriptados.
RNFWS-05	O s*ite* deve ser compatível com as versões mais recentes de Chrome, Firefox, Safari e Edge.
RNFWS-06	.O site deve ser submetido a validadores W3C (World Wide Web Consortium).
RNFWS-07	Todas as páginas devem ser entregues em UTF-8.
RNFWS-08	Os Ficheiros CSS e Java*Script* devem ser reutilizados.
RNFWS-09	O sistema deve recuperar de falha crítica em menos de 5 minutos.

Tabela 7 Requisitos Não Funcionais da Site

3.5. Persistência de dados

Nesta secção apresenta-se o modelo físico de dados que é utilizado no *site* de suporte aos utilizadores, em concreto, dada a simplicidades, resume-se à tabela "utilizador".

Esta tabela armazena informações essenciais sobre os utilizadores registados no sistema, incluindo um identificador único (*id*), o nome do utilizador (*nome*), o endereço de *Email* (*Email*), a palavra-passe (*senha*) e a data de criação do registo (*criado_em*).

A sua estrutura foi concebida para garantir a integridade e segurança dos dados, permitindo uma gestão eficiente dos utilizadores.

A figura seguinte ilustra a estrutura física da tabela.

Figura 14 Atributos da tabela "utilizador"

3.6. Protótipo

Nesta secção, é apresentado o protótipo desenvolvido, que inclui o jogo *Wild Soul* e o *site*. Para o jogo, são descritas as suas características gerais, o esquema navegacional e apresentados exemplos de código fonte. De forma semelhante, para o *site*, são abordadas as características gerais das páginas *web*, o esquema navegacional e exemplos de código fonte que suportam as suas funcionalidades.

3.6.1. Jogo Wild Soul

Nesta secção apresenta o jogo *Wild Soul*, um protótipo desenvolvido com o objetivo de oferecer uma experiência narrativa e interativa no universo nórdico. Inicialmente, são descritas as suas características gerais, abordando elementos como a jogabilidade, os objetivos principais e os elementos que compõem o ambiente de

jogo. Em seguida, é explorado o esquema navegacional, que representa a estrutura de navegação e interação do jogador com os diferentes elementos do jogo, como menus, mapas e eventos. Por fim, são apresentados exemplos de código fonte que evidenciam a implementação de funcionalidades fundamentais, permitindo compreender as bases técnicas e criativas do jogo.

3.6.1.1. Características Gerais

O jogo apresenta uma mecânica de movimentação que permite ao jogador deslocar-se em quatro direções: cima, baixo, esquerda e direita.

O ambiente inicial é apresenta uma pequena floresta que inclui a casa do protagonista. Este cenário foi desenvolvido para oferecer ao jogador um espaço limitado, mas funcional, para explorar e testar a movimentação antes de começar a campanha. Este cenário inclui um inimigo que pode atacar o personagem ao detetar o seu movimento.

A movimentação do personagem é fluida e responde imediatamente aos comandos do jogador, garantindo uma experiência consistente e intuitiva.

3.6.1.2. Navegação e cenários de jogo

Nesta secção apresenta-se de forma gráfica todos os conceitos do jogo como: o jogador, os inimigos e as estruturas.

A figura seguinte ilustra uma captura de ecrã do personagem principal, que se desloca em todas as direções e utiliza uma espada curta para atacar.

Figura 15 Personagem principal

A figura seguinte ilustra uma captura de ecrã do *slime*, uma criatura hostil que, ao detetar o jogador, se movimenta até ele para atacar.

Figura 16 Slime, criatura hostil

A figura seguinte ilustra uma captura de ecrã do exterior casa do jogador.

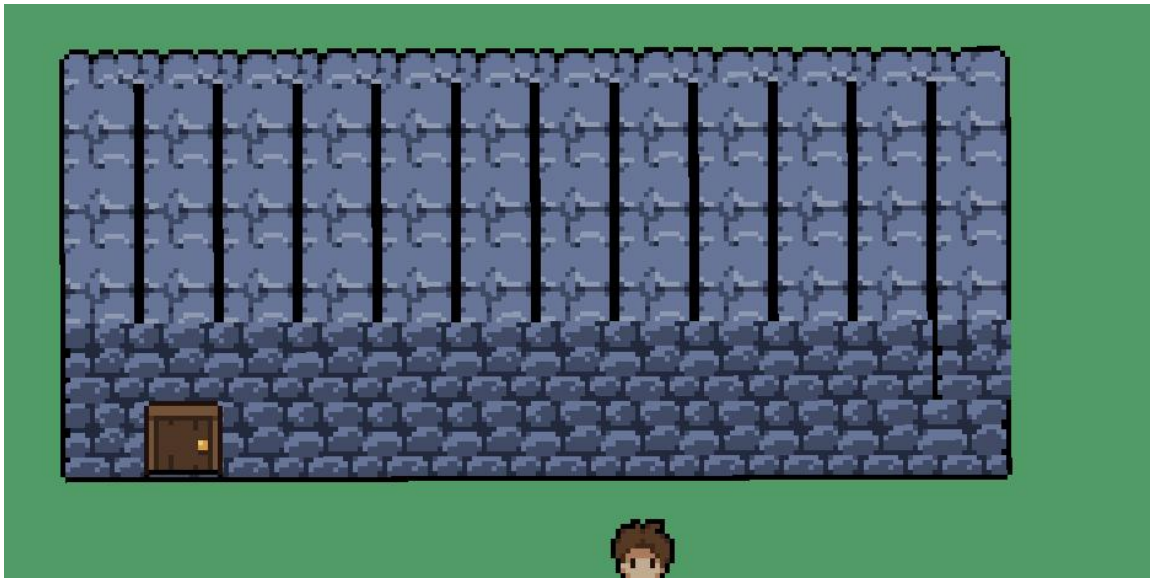

Figura 17 Exterior casa do jogador

A figura seguinte ilustra uma captura de ecrã do interior da casa do jogador, onde estão dispostos diversos móveis.

Figura 18 Interior da casa do jogador

A figura seguinte ilustra uma captura de ecrã da entrada para a caverna onde é possível encontrar o primeiro *Boss* Guardião, acompanhado por vários inimigos nas proximidades.

Figura 19 Entrada para caverna do 1º Guardião

A figura seguinte ilustra uma captura de ecrã da vila abandonada, onde é possível encontrar vários inimigos.

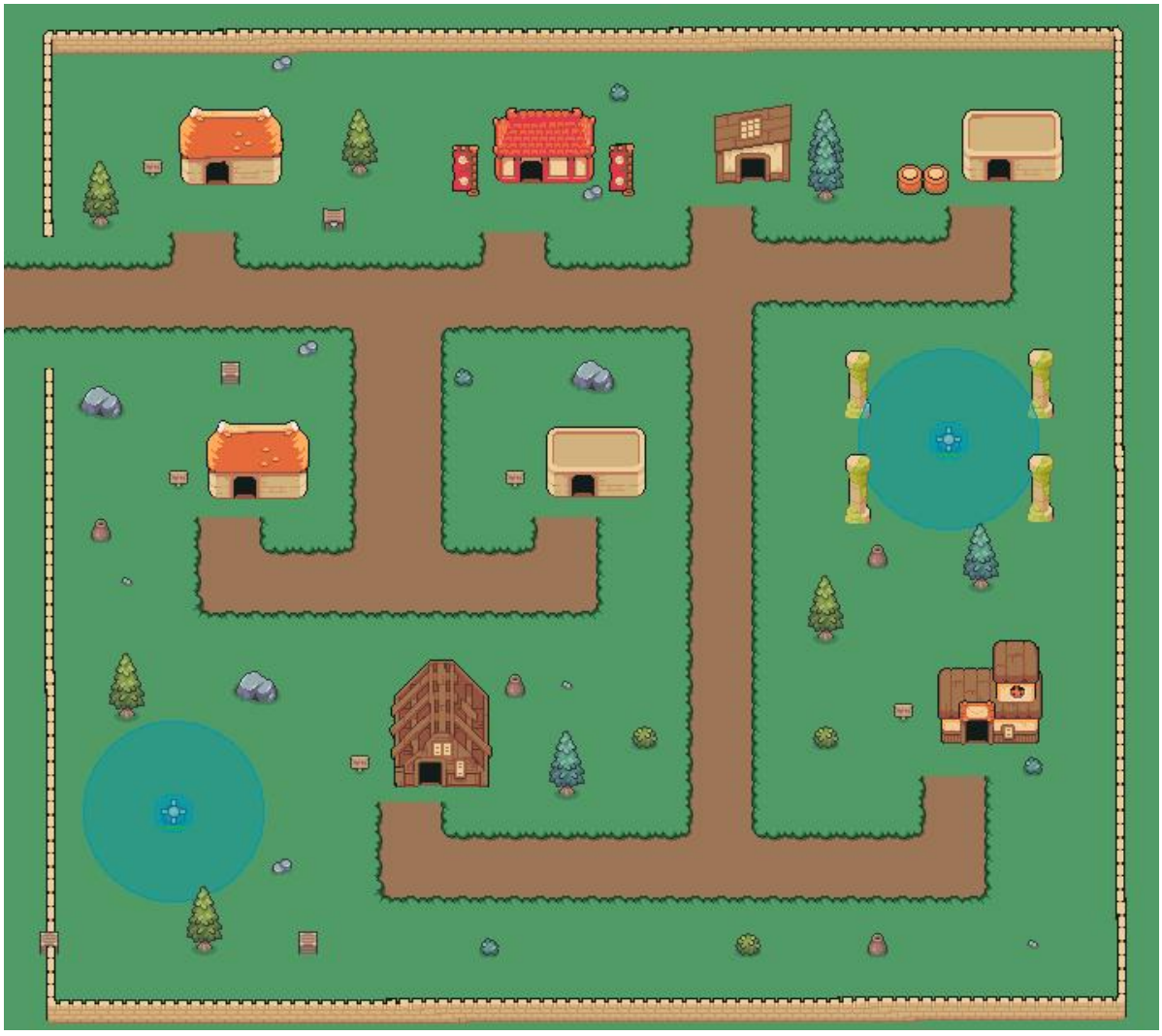

Figura 20 Vila abandonada

3.6.1.3. Exemplo de código-fonte

Está secção apresenta o código fonte tanto do personagem como do inimigo inicial.

O código seguinte implementa as variáveis, animações e a movimentação do jogador para o jogo, utilizando a classe *CharacterBody2D*. As funções principais incluem a definição da velocidade do personagem, a direção do movimento e a reprodução de animações com base no estado atual do jogador. Além disso, o código controla o estado de vida do personagem, garantindo que ele seja removido do jogo ao atingir 0 de vida. A lógica para atualizar a posição e o estado do jogador é processada a cada *frame* no método *_physics_process*.

```
extends CharacterBody2D


var enemy_inattack_range = false
var enemy_attack_cooldown = true
var vida = 100
var jogador_alive = true
var attack_ip = false
const SPEED = 120
var current_dir = "none"

func _ready():
	$AnimatedSprite2D.play("Idle")

func _physics_process(delta):
	player_movement(delta)
	enemy_attack()
	attack()
	update_vida()

	if vida <= 0:
		jogador_alive = false
		vida = 0
		print("jogador morreu")
		self.queue_free()

func player_movement(delta):

	if Input.is_action_pressed("move_right"):
		current_dir = "right"
		play_anim(1)
		velocity.x = SPEED
		velocity.y = 0
	elif Input.is_action_pressed("move_left"):
		current_dir = "left"
		play_anim(1)
		velocity.x = -SPEED
		velocity.y = 0
	elif Input.is_action_pressed("move_down"):
		current_dir = "down"
		play_anim(1)
```

```
                velocity.y = SPEED
                velocity.x = 0
        elif Input.is_action_pressed("move_up"):
                current_dir = "up"
                play_anim(1)
                velocity.y = -SPEED
                velocity.x = 0
        else:
                play_anim(0)
                velocity.x = 0
                velocity.y = 0

        move_and_slide()

func play_anim(movement):
        var dir = current_dir
        var anim = $AnimatedSprite2D

        if dir == "right":
                anim.flip_h = false
                if movement == 1:
                        anim.play("Run_side")
                elif movement == 0:
                        if attack_ip == false:
                                anim.play("Idle_side")
        if dir == "left":
                anim.flip_h = true
                if movement == 1:
                        anim.play("Run_side")
                elif movement == 0:
                        if attack_ip == false:
                                anim.play("Idle_side")
        if dir == "down":
                anim.flip_h = true
                if movement == 1:
                        anim.play("Run")
                elif movement == 0:
                        if attack_ip == false:
                                anim.play("Idle")
        if dir == "up":
```

```
                anim.flip_h = true
                if movement == 1:
                        anim.play("Run_back")
                elif movement == 0:
                        if attack_ip == false:
                                anim.play("Idle_back")
```

O código seguinte implementa o sistema de combate do jogo, incluindo a interação entre o jogador e os inimigos, além do controlo da barra de vida. Através de colisões, o jogo determina se o jogador está ao alcance do inimigo para ataques, aplicando cooldowns entre as ações ofensivas. Também permite que o jogador execute ataques, com animações específicas para cada direção. Adicionalmente, inclui uma lógica de regeneração da vida e atualização visual da barra de vida com base no estado atual do jogador.

```
func jogador():
        pass


func _on_hitbox_body_entered(body):
        if body.has_method("enemy"):
                enemy_inattack_range = true



func _on_hitbox_body_exited(body):
        if body.has_method("enemy"):
                enemy_inattack_range = false

func enemy_attack():
        if enemy_inattack_range and enemy_attack_cooldown == true:
                vida = vida - 5
                enemy_attack_cooldown = false
                $attack_cooldown.start()
                print(vida)



func _on_attack_cooldown_timeout():
        enemy_attack_cooldown = true

func attack():
        var dir = current_dir
```

```
        if Input.is_action_just_pressed("ataque"):
                global.jogador_current_attack = true
                attack_ip = true
                if dir == "right":
                        $AnimatedSprite2D.flip_h = false
                        $AnimatedSprite2D.play("Attack_side")
                        $deal_attack_timer.start()
                if dir == "left":
                        $AnimatedSprite2D.flip_h = true
                        $AnimatedSprite2D.play("Attack_side")
                        $deal_attack_timer.start()
                if dir == "down":
                        $AnimatedSprite2D.play("Attack")
                        $deal_attack_timer.start()
                if dir == "up":
                        $AnimatedSprite2D.play("Attack_back")
                        $deal_attack_timer.start()

func _on_deal_attack_timer_timeout():
        $deal_attack_timer.stop()
        global.jogador_current_attack = false
        attack_ip = false

func update_vida():
        var Barra_PV = $Barra_PV
        Barra_PV.value = vida
        if vida >= 200:
                Barra_PV.visible = false
        else:
                Barra_PV.visible = true

func _on_regin_timer_timeout():
        if vida < 200:
                vida = vida + 20
                if vida > 200:
                        vida = 200
        if vida <= 0:
                vida = 0
```

O código seguinte define as variáveis e a movimentação do inimigo do tipo *slime*. A movimentação do *slime* que é baseada no estado de perseguição ao jogador, movendo-se em direção ao mesmo quando ativada. Inclui animações específicas para o estado de movimento ou inatividade e ajusta a direção visual (espelhamento horizontal) com base na posição relativa ao jogador. O sistema também controla a vida do *slime*, determinando se ele pode sofrer dano e atualizando visualmente o seu estado.

```
extends CharacterBody2D


var SPEED = 40
var jogador_chase = false
var jogador = null
var vida = 100
var jogador_inattack_zone = false
var can_take_damage = true


func _physics_process(delta):
	deal_with_damage()
	update_vida()


	if jogador_chase:
		position += (jogador.position - position)/SPEED


		$AnimatedSprite2D.play("Run")


		if(jogador.position.x - position.x) <0:
			$AnimatedSprite2D.flip_h = true
		else:
			$AnimatedSprite2D.flip_h = false
	else:
		$AnimatedSprite2D.play("Idle")
```

O código seguinte complementa a lógica de combate e interação do inimigo *slime* com o jogador. A detecção de proximidade ativa ou desativa a perseguição ao jogador. Quando o jogador está na zona de ataque, o *slime* pode sofrer dano, respeitando um *cooldown* entre os danos consecutivos. A vida do *slime* é atualizada dinamicamente e refletida numa barra de vida, sendo o inimigo eliminado caso a sua vida chegue a 0. Este sistema também assegura que o *slime* apenas sofre dano quando permitido, com um controlo eficaz do tempo entre ataques.

```
func _on_detetar_body_entered(body):
	jogador = body
	jogador_chase = true

func _on_detetar_body_exited(body):
	jogador = null
	jogador_chase = false

func enemy():
	pass
func _on_hitbox_body_entered(body):
	if body.has_method("jogador"):
		jogador_inattack_zone = true

func _on_hitbox_body_exited(body):
	if body.has_method("jogador"):
		jogador_inattack_zone = false

func deal_with_damage():
	if jogador_inattack_zone and global.jogador_current_attack == true:
		if can_take_damage == true:
			vida = vida - 20
			$take_damage_cooldown.start()
			can_take_damage = false
			print("vida =", vida)
			if vida <= 0:
				self.queue_free()

func _on_take_damage_cooldown_timeout():
	can_take_damage = true

func update_vida():
	var Barra_PV = $Barra_PV

	Barra_PV.value = vida

	if vida >= 100:
		Barra_PV.visible = false
	else:
		Barra_PV.visible = true
```

3.6.2. Página de suporte aos jogadores

Nesta secção apresenta-se o *site* do jogo *Wild Soul*, uma plataforma *web* concebida para facultar informação, novidades e interação à comunidade de jogadores. Inicialmente, apresentam-se as suas funcionalidades principais, nomeadamente os menus *Início*, *Novidades*, *Comunidade* e *Instalação*, bem como as páginas de *Login* e *Registo*. Seguidamente, apresenta-se o esquema navegacional, que ilustra a interligação entre estas secções e permite entender o percurso do utilizador. Por fim, apresentam-se exemplos de fluxos de navegação que evidenciam a estrutura do *site* e a forma como o utilizador acede às suas funcionalidades.

3.6.2.1. Características Gerais

O *site* apresenta as seguintes páginas principais:

- Início;
- Novidades;
- Comunidade;
- Instalação.

Na página inicial, encontra-se um vídeo demonstrativo do jogo, acompanhado por um texto sobre o mesmo. Além disso, existe um menu de navegação que inclui as opções *Início, Novidades, Comunidade* e *Instalação*, que permite ao utilizador aceder às respetivas páginas e selecionar opções.

O *site* também disponibiliza dois botões para *login* e registo que serão descritos posteriormente.

As páginas de *Novidades* e Instalação encontram-se em desenvolvimento.

Na página da *Comunidade*, são apresentadas informações sobre a comunidade. Para aceder a esta página, o utilizador deve efetuar *login*. Se ainda não tiver iniciado sessão, será exibido um botão "Fazer *login* para aceder", que redirecionará para a página de autenticação. Caso a sessão já tenha sido iniciada, basta clicar em "Entrar" para ser direcionado à comunidade oficial no Discord.

3.6.2.2. Esquema Navegacional

O esquema navegacional demonstra a estrutura e a organização da navegação dentro de um sistema, *site* ou aplicação. Este representa visualmente como as diferentes páginas ou secções estão interligadas, que permite compreender o fluxo de interação do utilizador. Este esquema é essencial para garantir uma experiência de navegação intuitiva e eficiente, que facilita o acesso às informações e funcionalidades do sistema.

A figura seguinte ilustra o esquema navegacional, que contém um fluxo de navegação entre páginas do *site*.

Cada nó (de 1 a 8) será detalhado nesta secção.

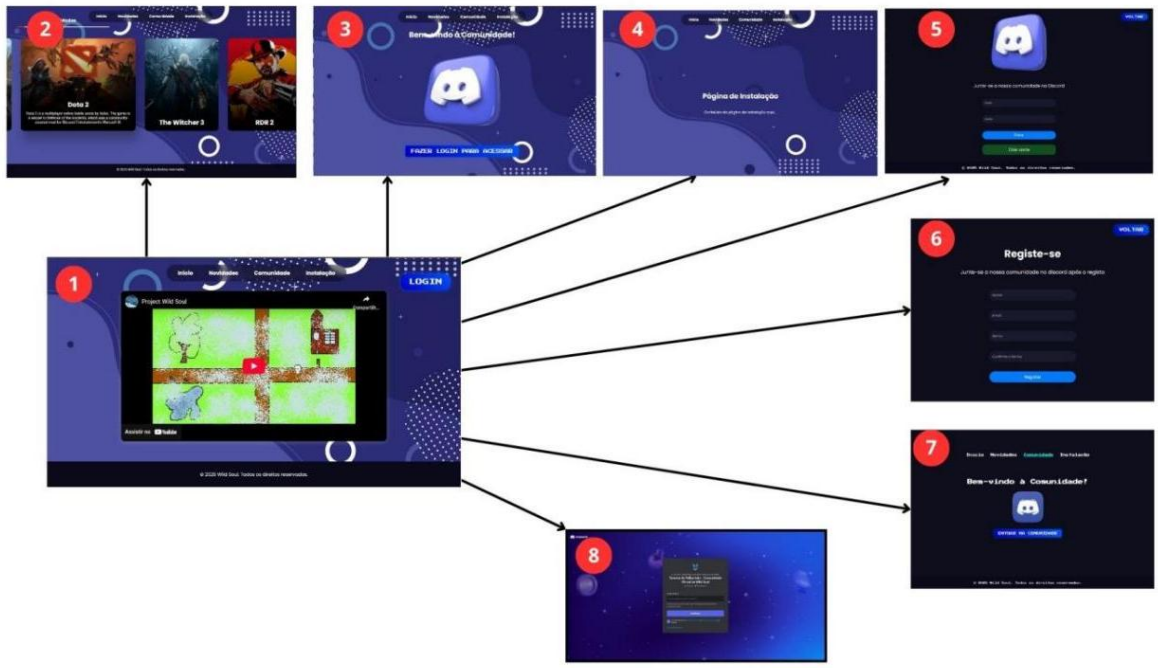

Figura 21 Esquema navegacional da página de suporte

1. A figura seguinte apresenta a página **Início** , que possui um menu navegacional, um vídeo demonstrativo do jogo e um botão para acesso ao ***login***.

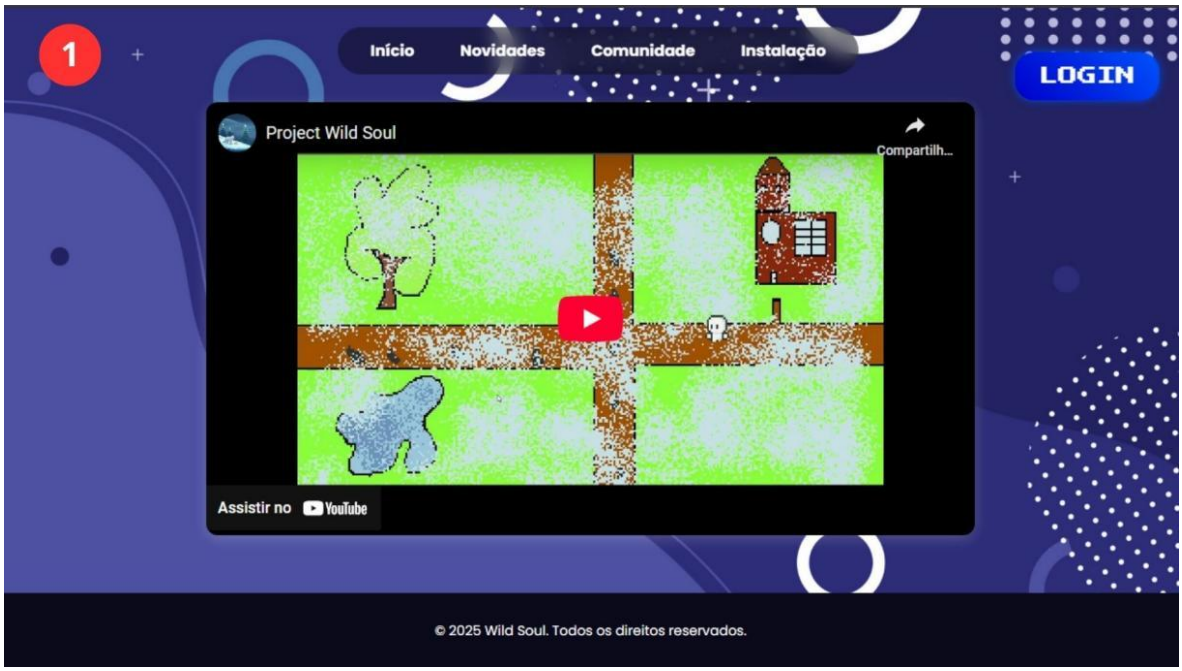

Figura 22 Página inicial

2. A figura seguinte apresenta a página **Novidades**, que apresenta ***cards*** com **imagens**, **títulos**, **descrições** e **datas de lançamento**.

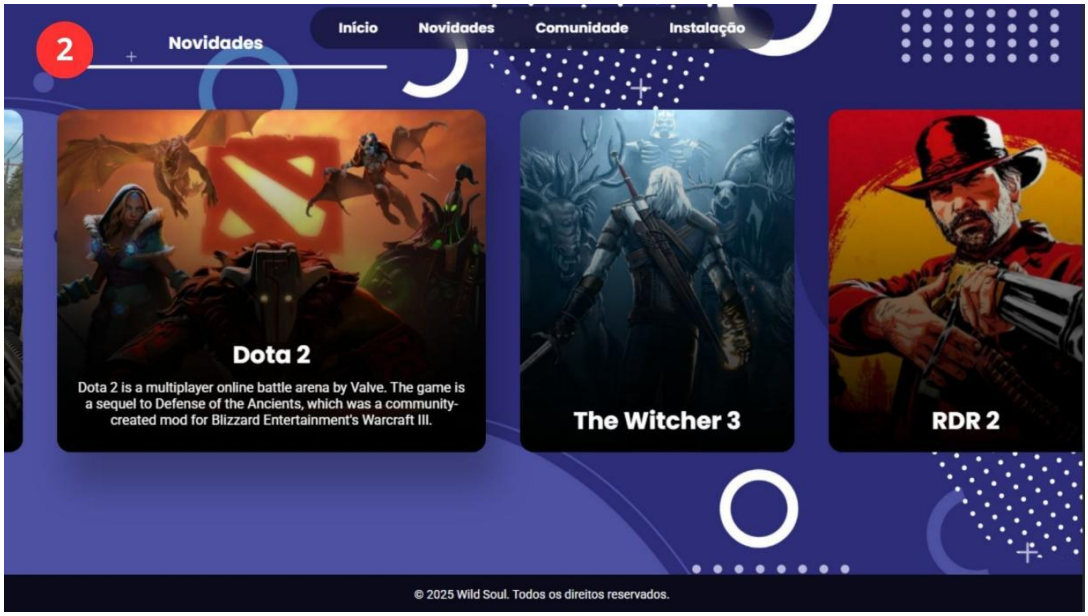

Figura 23 Página de novidades

3. A figura seguinte apresenta a página **Comunidade** do *site*. Esta página inclui texto de boas-vindas, acompanhado por uma imagem representativa da plataforma onde esta se encontra. Adicionalmente, dispõe de um botão que permite a autenticação do utilizador, que garante o acesso à comunidade .

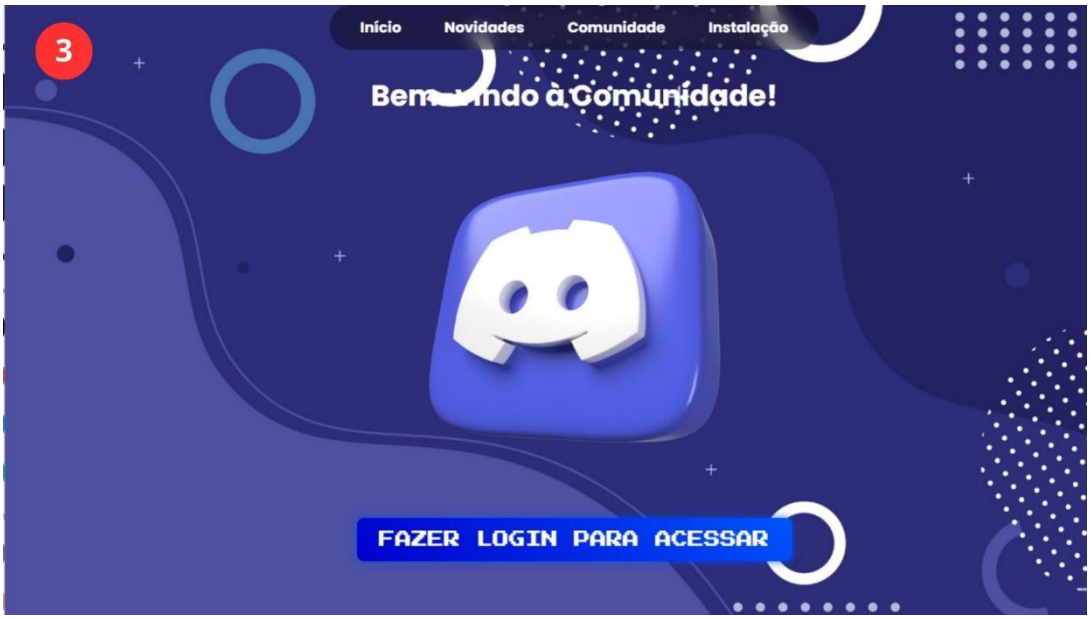

Figura 24 Página da comunidade

4. A figura seguinte apresenta a página de **Instalação** do jogo, que contém texto que descreve informações sobre o ficheiro do jogo e inclui um botão para a instalação do mesmo.

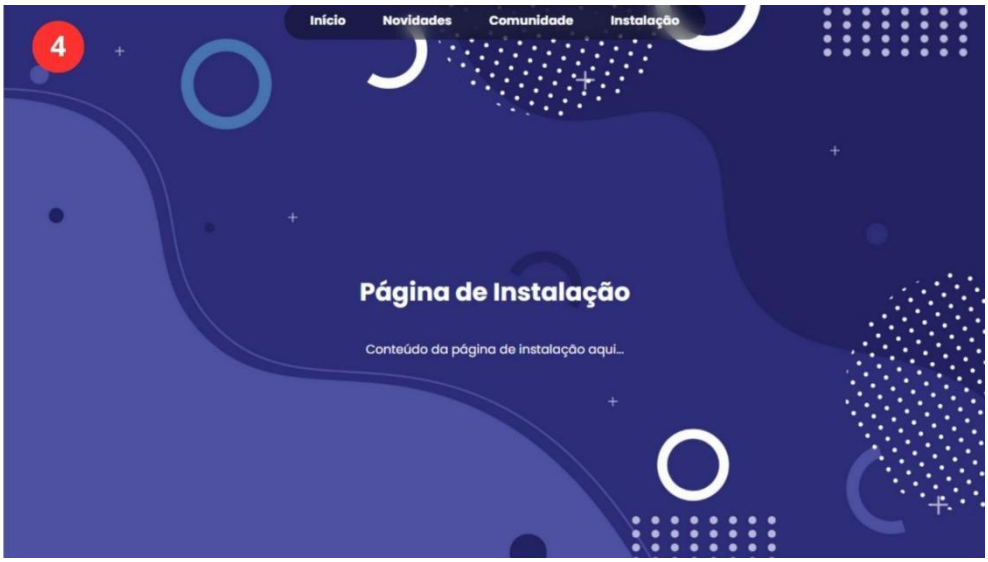

Figura 25 Página de instalação

5. A figura seguinte apresenta a página de ***login.*** A página contém a imagem referente à plataforma da comunidade, texto, zona de *inputs* para o utilizador introduzir suas credencias sendo elas *email* e *senha*, e por fim, possui botões, sendo eles o botão "*entrar*" para o utilizador autenticar suas credencias, o botão "*criar conta*" que redireciona o utilizador para página de registo, o botão "*voltar*" que redireciona o utilizador para à página de **início**.

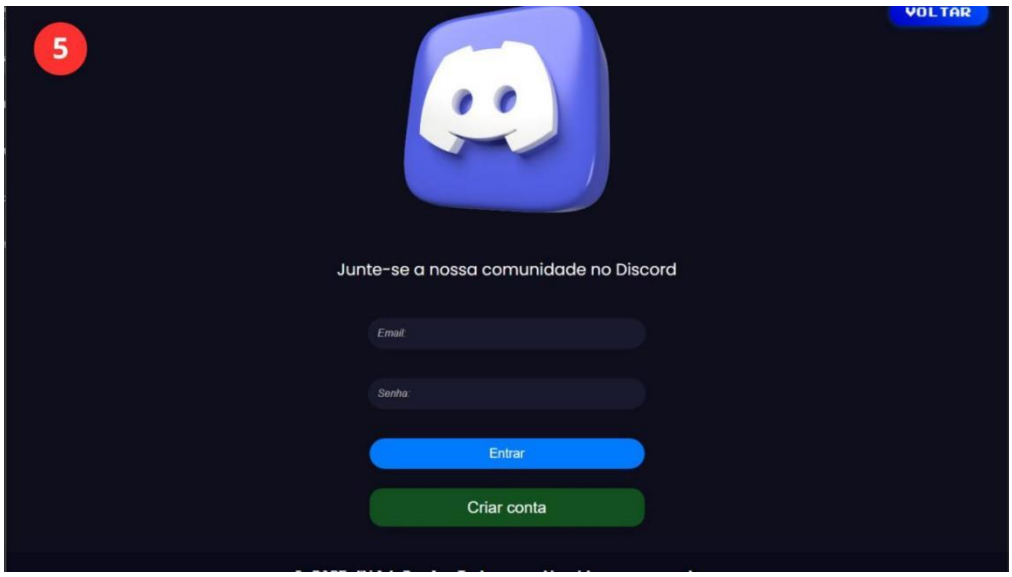

Figura 26 Página de login

6. A figura seguinte apresenta a página de **registo**. Nesta página, há texto e campos de entrada (*inputs*), nomeadamente *nome*, *email*, *senha* e confirmar senha. Além disso, contém botões como *"Voltar"* e *"Registar"*, sendo que este último executa a submissão das credenciais e regista o utilizador na base de dados.

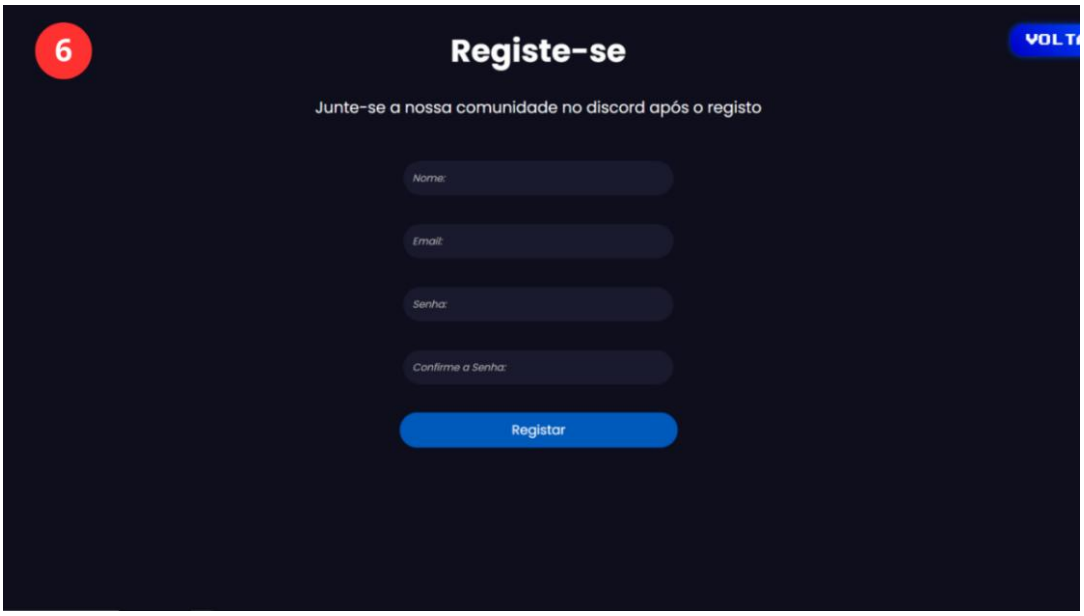

Figura 27 Página de registo

7. A figura seguinte apresenta a página Comunidade do *site*. Esta página apresenta texto de boas-vindas, bem como uma imagem que representa a plataforma. Adicionalmente, dispõe de um botão que permite o acesso à comunidade, funcionalidade disponível exclusivamente para utilizadores autenticados.

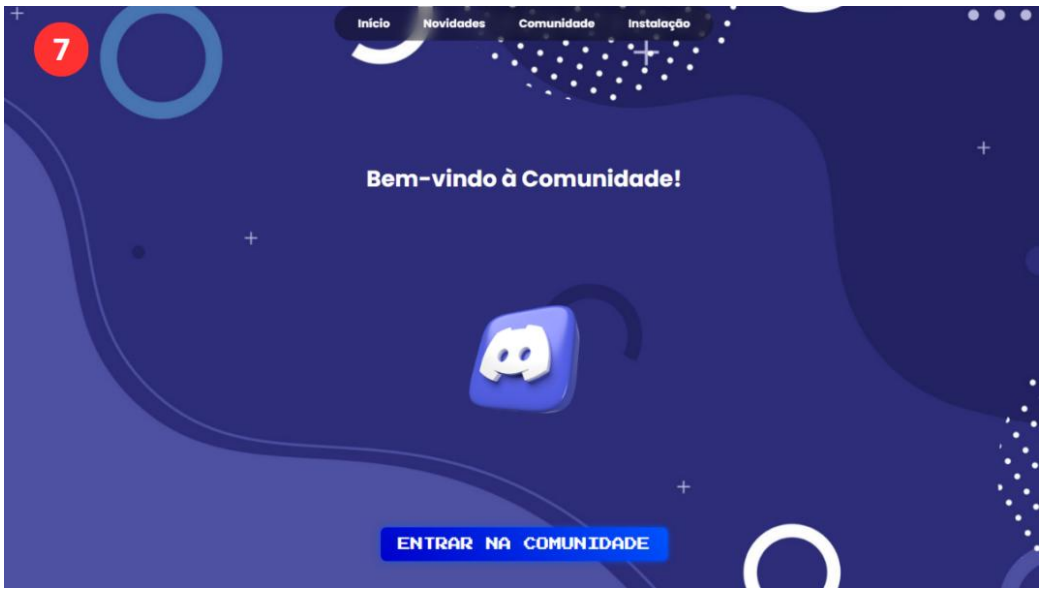

Figura 28 Página da comunidade

8. A figura seguinte apresenta a *interface* de convite para a **comunidade** oficial do projeto ***Wild Soul*** na plataforma Discord. No centro da página, exibe-se então uma caixa de convite contendo o nome da comunidade, ***Taverna de Valkarheim*** – *Comunidade Oficial de* ***Wild Soul***, juntamente com informações sobre o número de membros *online* e totais de utilizadores. Observa-se também um botão com a opção *"Aceitar convite"*, permitindo o acesso à comunidade. O fundo da página apresenta um *design* com tons de azul e elementos gráficos decorativos, característicos da identidade visual da plataforma.

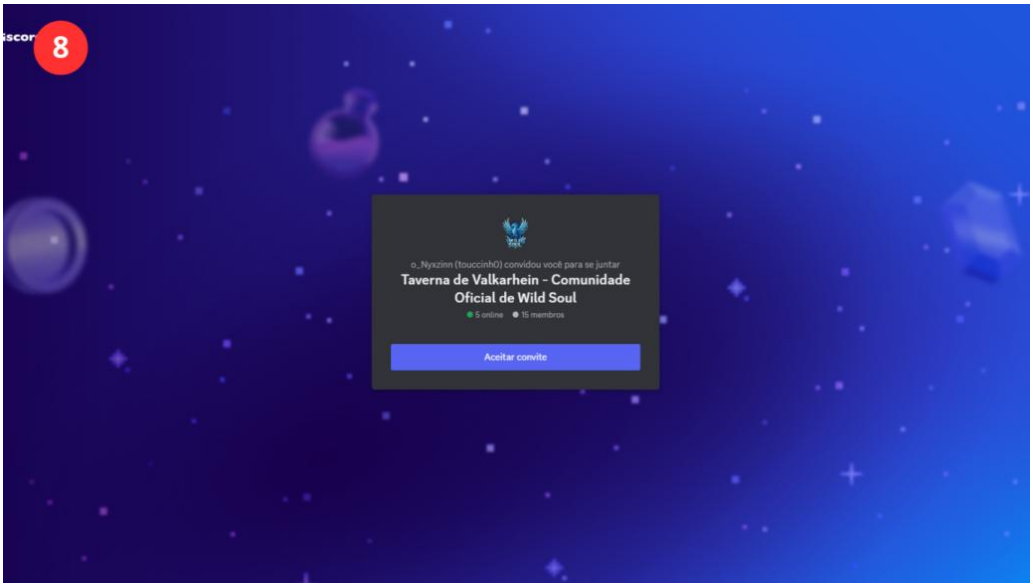

Figura 29 Página de introdução (Discord)

3.6.3. Exemplos de código-fonte

O código seguinte estabelece uma conexão com o banco de dados MySQL utilizando PHP. São definidas as configurações necessárias para a conexão, que incluem o endereço do servidor, o nome do banco de dados, o nome de utilizador e a senha. Em seguida, a conexão é estabelecida através da classe PDO (*PHP Data Objects*), que permite interagir com o banco de dados de forma segura.

```
<?php
// Configurações de conexão
$host = 'sql305.infinityfree.com'; // Endereço do servidor MySQL
$dbname = 'if0_38194502_db_wildsoul'; // Nome da base de dados
$User = 'if0_38194502'; // Usuário do MySQL
$pass = 'yhonO7yc4L'; // Senha do MySQL
// conexão usando PDO
try {
$pdo = new PDO("mysql:host=$host;dbname=$dbname;charset=utf8", $User, $pass);
$pdo->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
?>
```

O código seguinte define um menu de navegação dinâmico em PHP, estruturado em um ficheiro separado, que permite a reutilização em várias páginas sem necessidade de repetição de código. A variável *"$paginaAtual"* armazena o nome do ficheiro em execução, que permite a identificação da página ativa. Com base nessa verificação, a classe "*active*" é atribuída ao *link* correspondente, destacando visualmente a página que está a ser visualizada pelo utilizador.

```
<?php
  $paginaAtual = basename($_Server['Script_NAME']);
?>
<nav class="menu">
 <ul>
  <li><a href="/index.php" class="<?= ($paginaAtual == 'index.php') ? 'active' :'' ?>">Início</a></li>
  <li><a href="/public/novidade.php" class="<?= ($paginaAtual == 'novidade.php') ? 'active' :
'' ?>">Novidades</a></li>
  <li><a  href="/public/comunidade.php"  class="<?=  ($paginaAtual  ==  'comunidade.php')  ?  'active'  :
'' ?>">Comunidade</a></li>
  <li><a  href="/public/download.php"  class="<?=  ($paginaAtual  ==  'download.php')  ?  'active'  :
'' ?>">Instalação</a></li>
 </ul>
</nav>
```

O código seguinte incorpora um vídeo do *YouTube* na página, que permite que seja exibido sem sair do *site*. A *tag "<iframe>"* é usada para essa integração, que garante que o vídeo possa ser reproduzido diretamente. Além disso, a estrutura do código facilita ajustes visuais para melhor apresentação.

```
<div class="video-container">
  <div class="video-wrapper">
  <iframe
  src="https://www.youtube.com/embed/qNObAkPIIAM?si=PdsqGDrn8KktpK89"
  title="YouTube video player"
   allow="accelerometer;  autoplay;  clipboard-write;  encrypted-media;  gyroscope;  picture-in-picture;
web-share"
  referrerpolicy="strict-origin-when-cross-origin"
  allowfullscreen
  loading="lazy">
  </iframe>
  </div>
</div>
```

O código seguinte cria um formulário de *login* que permite aos utilizadores inserir o seu *email* e *senha* para autenticação. Através da *tag "<form>"*, os dados são enviados para um ficheiro PHP responsável pelo processamento. O botão *"Entrar"* submete o formulário, enquanto o botão *"Criar conta"* redireciona o utilizador para a página de registo.

```
<form action="/actions/Login_action.php" method="post">
<div class="Input-wrapper">
<Input type="Email" name="Email" placeholder="Email:" required> <br>
<Input type="Password" name="senha" placeholder="Senha:" required> <br>
        <button type="submit">Entrar</button>
        </div>
        <div class="criarconta">
        <button class="criar-conta" onClick="redirectToLogin()">Criar conta</button>
        <Script>
        function redirectToLogin() {
        window.location.href = "/public/regista.php";
        }
        </Script>
        </div>
</form>
```

O código seguinte processa o *login* de um utilizador verificando as credenciais inseridas. Primeiro, os dados do formulário são capturados e o *email* é sanitizado para maior segurança. Em seguida, o código procura o utilizador no banco de dados com base no *email* fornecido. Se o utilizador for encontrado e a senha corresponder à armazenada, a sessão é iniciada e o utilizador é redirecionado para a página comunidade. Caso contrário, ele é redirecionado para a página de *login* com uma mensagem de erro.

```
if ($_Server['REQUEST_METHOD'] == 'POST') {
    // Captura os dados do formulário
    $Email = filter_var($_POST['Email'], FILTER_SANITIZE_Email); // Sanitiza o Email
    $senha = $_POST['senha'];
    try {
        // Consulta para buscar o utilizador pelo Email
        $sql = "SELECT * FROM utilizador WHERE Email = :Email";
        $stmt = $pdo->prepare($sql);
        $stmt->execute(['Email' => $Email]);
        $utilizador = $stmt->fetch(PDO::FETCH_ASSOC);

        // Verifica se o utilizador existe e a senha está correta
```

```
        if ($utilizador && Password_verify($senha, $utilizador['senha'])) {
            // Login bem-sucedido: define a sessão corretamente
            $_SESSION['usuario_id'] = $utilizador['id']; // Armazena o ID do usuário
            $_SESSION['usuario_nome'] = $utilizador['nome']; // Armazena o nome do usuário

            // Redireciona para a página comunidade
            header("Location: /public/comunidade.php?mensagem=logado");
            exit;
        } else {
            // Falha no Login: redireciona com mensagem de erro
            header("Location: /public/Login.php?erro=credenciais_invalidas");
            exit;
            }
        } catch (PDOException $e) {
        // Trata erros do banco de dados
        echo "Erro ao fazer Login: " . $e->getMessage();
        exit;
        }
}
?>
```

O código seguinte cria um formulário de registo para novos utilizadores. Os campos solicitam o *nome*, *email*, *senha* e a confirmação da senha. Todos os campos são obrigatórios, garantindo que as informações necessárias sejam fornecidas antes do envio. O formulário envia os dados para o ficheiro PHP responsável pelo processamento da registo. O botão *"Registar"* permite que o utilizador submeta as informações inseridas.

```
<form action="/actions/registo_action.php" method="post" onsubmit="validarFormulario(event)">
<Input type="text" name="nome" id="nome" placeholder="Nome:" required><br>
<Input type="Email" name="Email" id="Email" placeholder="Email:" required><br>
<Input type="Password" name="senha" id="senha" placeholder="Senha:" required><br>
<Input type="Password" name="confirmar_senha" id="confirmar_senha" placeholder="Confirme a Senha:" required><br>
<button type="submit">Registar</button>
</form>
```

O código seguinte processa o registo de um novo utilizador ao validar e armazenar as informações fornecidas. Quando o formulário é submetido, os dados são capturados e sanitizados para segurança. O código verifica se o *email* é válido e se as senhas coincidem. Caso contrário, mensagens de erro apropriadas são geradas.

Se o *email* for válido e as senhas coincidirem, a senha é criptografada para garantir a segurança. O código então verifica se o *email* já está cadastrado no banco de dados. Se não estiver, os dados do utilizador são inseridos na tabela correspondente e o utilizador é redirecionado para a página de registo com uma mensagem de sucesso. Se o *email* já existir, uma mensagem de erro é gerada informando que o *email* já está registado.

```
if ($_Server['REQUEST_METHOD'] == 'POST') {
    // Captura e sanitiza os dados do formulário
    $nome = htmlspecialchars(trim($_POST['nome']));
    $Email = filter_var(trim($_POST['Email']), FILTER_SANITIZE_Email);
    // Verifica se o Email é válido
    if (!filter_var($Email, FILTER_VALIDATE_Email)) {
        $mensagem = "Erro: O Email fornecido não é válido.";
    } elseif ($_POST['senha'] !== $_POST['confirmar_senha']) { // Verifica se as senhas coincidem
        $mensagem = "Erro: As senhas não coincidem.";
    } else {
        $senha = Password_hash($_POST['senha'], Password_DEFAULT); // Criptografa a senha
        try {
            // Verifica se o Email já existe no banco de dados
            $sqlCheck = "SELECT COUNT(*) FROM utilizador WHERE Email = :Email";
            $stmtCheck = $pdo->prepare($sqlCheck);
            $stmtCheck->execute(['Email' => $Email]);
            $EmailExists = $stmtCheck->fetchColumn();
            if ($EmailExists) {
                $mensagem = "Erro: O Email já está cadastrado.";
            } else {
                // Insere os dados na tabela 'utilizador'
                $sql = "INSERT INTO utilizador (nome, Email, senha) VALUES (:nome, :Email, :senha)";
                $stmt = $pdo->prepare($sql);
                $stmt->execute([ ':nome' => $nome, ':Email' => $Email, ':senha' => $senha ]);
                // Redireciona de volta para a página do formulário com uma mensagem de sucesso
                header("Location: /public/registo.php?mensagem=" . urlencode("Cadastro realizado com
                sucesso!"));
                exit;
            }
```

O código seguinte exibe a página de boas-vindas para a comunidade. Se o utilizador estiver autenticado, surge um botão que permite entrar na comunidade do Discord. Se não estiver autenticado, é exibido um botão para fazer *login* e aceder à comunidade. A página também inclui um logótipo do Discord para reforçar a identidade visual.

```
  <div class="comunidade-container">

  <h1>Bem-vindo à Comunidade!</h1>

  <img src="/imagens/discordpx.png" alt="Discord Logo" class="discord-logo">


  <?php if (isset($_SESSION['usuario_id'])): ?>

  <!-- Se o usuário estiver logado, exibe o botão "Entrar" -->

  <a href="<?= $discordLink ?>" class="botao entrar">Entrar na Comunidade</a>

  <?php else: ?>

  <!-- Se não estiver logado, exibe o botão de Login -->

  <a href="/public/Login.php" class="botao Login">Fazer Login para Acessar</a>

  <?php endif; ?>
</div>
```

Página em branco

4. Conclusões

Neste capítulo apresenta-se a discussão relativa ao desenvolvimento do projeto e o trabalho futuro.

4.1. Discussão

Este trabalho permitiu consolidar conhecimentos adquiridos ao longo do curso bem como desenvolver novas competências no domínio de desenvolvimento de *Software*.

Considerando o contexto e motivação, perante o problema identificado:

"Apesar do crescimento e reconhecimento gradual da indústria de jogos indie, verifica-se uma escassez dos mesmos. Por esta razão, este projeto visa contribuir para a criação de mais um título", definiram-se sete objetivos principais:

- **O1** – Analisar o enquadramento teórico do projeto;
- **O2** - Definir os casos de uso do projeto, i.e. identificação dos principais atores, processos e fronteiras do sistema;
- **O3** - Elaborar o modelo de domínio que traduza as relações estáticas das principais classes do projeto;
- **O4** - Definir a arquitetura do sistema;
- **O5** - Definir os requisitos das aplicações;
- **O6** - Definir mecanismos de persistência de dados;
- **O7** - Realizar o protótipo aplicacional.

Assim, relativamente a:

- **O1** – Como se demonstrou no Capítulo 2, foram estudadas e incluídas no presente relatório: videojogos (plataformas para videojogos, videojogos online vs. offline, videojogos 2D, 3D e 2.5D, tipos de videojogos, E-sports e aspetos de competitividade dos videojogos), metodologias de análise (casos de uso, modelo de domínio, requisitos funcionais e não funcionais), tecnologias utilizadas (Godot, Godot 2D IDE, Godot 2D Runner, Godot 2D Editores de ativos, Libresprite, Auidacity, HTML5, CSS3, PHP, Java*Script*, MySQL, Visual Studio Code);
- **O2** - Como se demonstrou na secção 3.1, foram definidos os casos de uso do projeto, identificados os principais atores e processos associados, bem como as fronteiras do sistema;
- **O3** - Como se demonstrou na secção 3.2, foi elaborado o modelo de domínio que traduz as principais relações estáticas das classes do projeto;
- **O4** - Como se demonstrou no Capítulo 3, secção 3.3, foi definida a arquitetura do sistema para o jogo e para o *site* de suporte aos jogadores;
- **O5** - Como se demonstrou na secção 3.4, foram definidos os requisitos funcionais e não funcionais das aplicações realizadas;

- **O6** - Como se demonstrou na secção 3.5, foram definidos mecanismos de persistência dos dados ao nível do *site* (os visitantes da página podem-se registar e os seus dados são guardados numa base de dados);
- **O7** - Como se demonstrou na secção 3.6 foram realizados e testados dois protótipos aplicacionais (para o jogo e *site* respetivamente).

Conclui-se deste modo que os objetivos principais foram alcançados.

4.2. Trabalho futuro

Este projeto representou uma etapa inicial no nosso percurso como criadores nas áreas de *game design* e *web design*. Ao longo do desenvolvimento, enfrentámos diversos desafios técnicos e criativos que nos permitiram adquirir experiência prática e uma melhor compreensão do processo de criação de jogos e interfaces digitais.
A concretização deste protótipo serviu não apenas como uma prova de conceito do universo *Wild Soul*, mas também como uma base sólida para futuras iterações.

No contexto do jogo, foi possível implementar as principais funcionalidades relacionadas com o movimento, combate e interação com o ambiente. No entanto, reconhecemos que ainda existem várias áreas por desenvolver. Entre as funcionalidades que pretendemos introduzir futuramente estão a implementação de poderes especiais para o personagem principal, bem como um sistema de persistência de dados, que permitirá guardar o progresso do jogador entre sessões.

Relativamente ao *site*, o foco futuro será desenvolver conhecimentos avançados na área da *interface* e do *design* visual, com o objetivo de otimizar a experiência de navegação. De igual desenvolver mecanismos previstos para o sistema de doações.

Este projeto é apenas o início de uma visão maior. Como referido neste relatório, a história apresentada neste protótipo será expandida numa trilogia completa intitulada *Wild Soul*, que explorará mais a fundo o universo narrativo.

Para alcançar uma experiência mais imersiva e ambiciosa, planeamos migrar para tecnologia 3D, que permitirá um salto significativo na qualidade gráfica, animações, e liberdade de *design*.

Em suma, este trabalho serviu para o desenvolvimento de competências e como ponto de partida para projetos mais complexos, de acordo com a nossa motivação em continuar a aprender e a evoluir nestas áreas.

.

Referências

[AC,25] Audacity
https://www.audacityteam.org/,
[online] acedido em 23/01/2025

[CG, 4] ChatGPT, Versão 4

[DC, 25] Discord,
https://discord.com/,
[online] acedido em 09/5/2025

[EDIO, 24] What is a use case diagram?, Educative Inc,
https://www.educative.io/answers/what-is-a-use-case-diagram,
[online] acedido em 25/9/2024

[FV, 24] Valente, Francisco. Requisitos Funcionais/Não Funcionais. Relatório de PAP. E.S.Camões (2024)

[GDDC,25] *Godot Engine Documentation*,
https://docs.godot*Engine*.org,
[online] acedido em 06/02/2025.

[IF, 24] InfinityFree,
https://www.infinityfree.com/,
[online] acedido em 26/1/2025

[JC, 24] Corbelli, Johnny. Modelo de Negócio. Relatório de PAP. E.S. Camões (2024)

[MW, 24] Requisitos funcionais e não funcionais o que são?, Mestres da Web, artigo escrito por Fernando Cunha
https://mestresdaweb.com.br/tecnologias/requisitos-funcionais-e-nao-funcionais-o-que-sao
[online] acedido em 22/10/2024

[OC,25] Oracle. MySQL
https://www.oracle.com/mysql/what-is-mysql/
[online] acedido em 03/02/2025

[OI, 25] Orama Interactive,
https://www.oramainteractive.com/,
[online] acedido em 08/01/2025

[OS, 24] Sow, Oumar. Modelo de Domínio. Relatório de PAP. E.S.Camões (2024)

[PG, 22] What is AES-256 Encryption & How Does it Work?, Progress *Software* Corporation, escrito por Victor Kananda a 22/6/2022
https://www.progress.com/blogs/use-aes-256-encryption-secure-data
[online] acedido em 23/10/2024

[PHPGRP, 25] PHP Group, PHP,
https://www.php.net/,
[online] acedido em 22/01/2025

[VSM, 24] Microsoft, Visual Studio code
https://visualstudio.microsoft.com/ ,
[online] acedido em 10/01/2025

[W3CSS, 25] CSS Animations, W3Schools, Refsnes Data,
https://www.w3schools.com/w3css/,
[online] acedido em 10/01/2025

[W3HTML, 25] HTML Form Validation, W3Schools, Refsnes Data,
https://www.w3schools.com/html/,
[online] acedido em 10/01/2025

[W3JV, 25] CSS Animations, W3Schools, Refsnes Data,
https://www.w3schools.com/js/,
[online] acedido em 10/01/2025