Theodoro Flôr da Rosa Satyro

Desenvolvimento do jogo "Pequenos Sobreviventes"

Vilhena - RO

2025

Theodoro Flôr da Rosa Satyro

Desenvolvimento do jogo "Pequenos Sobreviventes"

Trabalho de Conclusão de Curso apresentado ao Instituto Federal de Educação, Ciência e Tecnologia de Rondônia – campus Vilhena, realizado em cumprimento de requisito parcial para a obtenção do título de Tecnólogo em Análise e Desenvolvimento de Sistemas.

Instituto Federal de Educação, Ciência e Tecnologia de Rondônia – IFRO

Campus Vilhena

Curso Superior em Tecnologia em Análise e Desenvolvimento de Sistemas

Orientador: Prof.Esp.Erick Leonardo Weil

Vilhena - RO

2025

Ficha catalográfica elaborada pelo Sistema Gerador de Ficha Catalográfica do IFRO, com dados informados pelo(a) próprio(a) autor(a).

Satyro, Theodoro Flôr da Rosa.
Desenvolvimento do jogo "Pequenos Sobreviventes" / Theodoro Flôr da Rosa Satyro, Vilhena-RO, 2025.
79 f.

Orientador(a): Prof.Esp Erick Leonardo Weil.

Trabalho de Conclusão de Curso (Superior de Tecnologia em Análise e Desenvolvimento de Sistemas) – Instituto Federal de Educação, Ciência e Tecnologia de Rondônia - IFRO, Vilhena-RO, 2025.

1. Desenvolvimento de Jogos. 2. Unity 3D. 3. Survivors-like. I. Weil, Erick Leonardo (orient.). II. Instituto Federal de Educação, Ciência e Tecnologia de Rondônia - IFRO. III. Título.

Bibliotecário(a) Responsável: Rosilene Maria do Couto Marques, CRB-11/321 (Campus Vilhena)

INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA DE RONDÔNIA
Vilhena - Código INEP: 11107804
Rodovia BR 174, KM 3, CEP 76982-270, Vilhena (RO)
CNPJ: 10.817.343/0003-69 - Telefone: 69 2101-0703

ATA DE DEFESA DE TRABALHO DE CONCLUSÃO DE CURSO

Na data 28/04/2025 realizou-se a sessão pública de defesa do Trabalho de Conclusão de Curso intitulada **Desenvolvimento do jogo "Pequenos Sobreviventes"** apresentada pelo aluno **Theodoro Flor da Rosa Satyro (2022103070005)** do Curso **Superior de Tecnologia em Análise e Desenvolvimento de Sistemas (Vilhena)**. Os trabalhos foram iniciados às **16:00** pelo Professor **Erick Leonardo Weil** presidente da banca examinadora, constituída pelos seguintes membros:

- **Erick Leonardo Weil** (Orientador)
- **Marco Antonio Augusto de Andrade** (Examinador Interno)
- **Jose Lucas Brandao Montes** (Examinador Interno)

A banca examinadora, tendo terminado a apresentação do conteúdo do Trabalho de Conclusão de Curso, passou à arguição do candidato. Em seguida, os examinadores reuniram-se para avaliação e deram o parecer final sobre o trabalho apresentado pelo aluno, tendo sido atribuído o seguinte resultado:

[X] APROVADO **Nota: 96**

Proclamados os resultados pelo presidente da banca examinadora, foram encerrados os trabalhos e, para constar, eu **Erick Leonardo Weil** lavrei a presente ata que assino juntamente com os demais membros da banca examinadora.

VILHENA / RO, 28/04/2025

Documento assinado eletronicamente por **Theodoro Flor da Rosa Satyro**, Discente, em 28/04/2025, às 17:03, conforme horário oficial de Rondônia, com fundamento no art. 6º, § 1º, do Decreto nº 8.539, de 8 de outubro de 2015.

Documento assinado eletronicamente por **Erick Leonardo Weil**, Orientador, em 28/04/2025, às 17:02, conforme horário oficial de Rondônia, com fundamento no art. 6º, § 1º, do Decreto nº 8.539, de 8 de outubro de 2015.

Documento assinado eletronicamente por **Marco Antonio Augusto de Andrade**, Examinador Interno, em 28/04/2025, às 17:07, conforme horário oficial de Rondônia, com fundamento no art. 6º, § 1º, do Decreto nº 8.539, de 8 de outubro de 2015.

Documento assinado eletronicamente por **Jose Lucas Brandao Montes**, Examinador Interno, em 28/04/2025, às 17:04, conforme horário oficial de Rondônia, com fundamento no art. 6º, § 1º, do Decreto nº 8.539, de 8 de outubro de 2015.

Dedico este trabalho a todos que sonham e se dedicam diariamente para transformar seus sonhos em realidade, seguindo o caminho da esperança, da perseverança e do esforço, "Ebenézer, até aqui nos ajudou o Senhor".

Agradecimentos

Primeiramente, agradeço a Deus, que me deu força, sabedoria e paciência ao longo de toda essa jornada.

À minha família, pelo apoio incondicional e por acreditarem em mim, mesmo nos momentos de incerteza. Sou grato por me proporcionarem as condições para seguir meus sonhos e por me ensinarem o valor da perseverança.

Aos meus amigos, pelo apoio, pelas ideias compartilhadas e pela motivação constante.

Ao meu orientador, cuja dedicação e compromisso foram fundamentais para a realização deste trabalho.

A todos que contribuíram, direta ou indiretamente, para a conclusão deste TCC, o meu sincero agradecimento.

RESUMO

Este trabalho apresenta o desenvolvimento de um jogo eletrônico no estilo *Survivors-like* e *Roguelite*, utilizando o motor de jogo *Unity* 3D e a linguagem de programação C#. O objetivo do projeto é explorar as mecânicas desse gênero, proporcionando uma experiência dinâmica e acessível a um público amplo. A metodologia adotada inclui pesquisa e análise da área de desenvolvimento de jogos, com foco nas tendências do mercado e nas características dos jogos *Survivors-like*, sendo seguida por um desenvolvimento incremental com a implementação progressiva de funcionalidades e testes contínuos para avaliar a jogabilidade e o desempenho. O projeto foi desenvolvido para dispositivos móveis, adotando uma estética em pixel art e mecânicas que combinam progressão, combate automático e desafios característicos de *roguelites*. Como resultado, o projeto entrega um jogo funcional, destacando os desafios enfrentados ao longo do desenvolvimento, como a otimização para plataformas móveis, o balanceamento de mecânicas e a criação de uma experiência envolvente. Conclui-se que o desenvolvimento desse jogo não apenas reforça conhecimentos técnicos adquiridos ao longo da graduação, mas também demonstra a capacidade de adaptação a novas tecnologias e metodologias, contribuindo para a formação profissional e incentivando a exploração do mercado de desenvolvimento de jogos.

Palavras-chave: Desenvolvimento de Jogos; *Unity* 3D; C#;*Survivors-like*; *Roguelite*.

Abstract

This work presents the development of an electronic game in the Survivors-like and Roguelite style, using the Unity 3D game engine and the C# programming language. The project's goal is to explore the mechanics of this genre, providing a dynamic and accessible experience for a broad audience. The methodology adopted includes research and analysis of the game development field, focusing on market trends and the characteristics of Survivors-like games, followed by incremental development with the progressive implementation of features and continuous testing to assess gameplay and performance. The project was developed for mobile devices, adopting a pixel art aesthetic and mechanics that combine progression, automatic combat, and challenges typical of roguelites. As a result, the project delivers a functional game, highlighting the challenges faced throughout development, such as optimization for mobile platforms, balancing mechanics, and creating an engaging experience. It is concluded that the development of this game not only reinforces technical knowledge acquired during the undergraduate program but also demonstrates the ability to adapt to new technologies and methodologies, contributing to professional growth and encouraging exploration in the game development market.

Keywords: Game Development; Unity 3D; C#; Survivors-like; Roguelite.

Lista de ilustrações

Lista de tabelas

Lista de abreviaturas e siglas

TI	Tecnologia da Informação.
ASCII	American Standard Code for Information Interchange.
GDD	Game Design Document, Documento de design de jogo.
VS	Visual Studio.
CI/CD	integração contínua e entrega contínua.
ECS	Entity-Component-System em português: Sistema de Componente e Entidade.
CPU	Unidade Central de Processamento.
UI	Interface do Usuário.
FPS	Frames per second","quadros por segundo".
SFX	"Sound effects"(efeitos sonoros).
Licença MIT	Licença de software criada pela Massachusetts Institute of Technology.

Sumário

1 Introdução

1.1 Contexto e problema

A palavra jogo, derivado do vocábulo latino "*ludus*", refere-se à diversão e brincadeira.(Alves; Bianchin, 2010) Os jogos são um conceito amplo e diversificado, podendo envolver diferentes tipos de atividades, desde aquelas baseadas em sorte ou habilidade até aquelas que exigem estratégia e conhecimento. Entre os diversos tipos de jogos, encontramos jogos de tabuleiro, jogos de cartas, jogos eletrônicos, esportivos, entre outros.

A definição do que é um jogo e o que não é, pode ser bem abrangente. Contudo, de maneira geral, jogos podem ser entendidos como uma forma de simulação interativa com objetivos claros, como podemos ver na seguinte definição:

> Jogos são atividades sociais e culturais voluntárias, significativas, fortemente absorventes, não produtivas, que se utilizam de um mundo abstrato, com efeitos negociados no mundo real, e cujo desenvolvimento e resultado final são incertos, onde um ou mais jogadores, ou equipes de jogadores, modificam interativamente e de forma quantificável o estado de um sistema artificial, possivelmente em busca de objetivos conflitantes, por meio de decisões e ações, algumas com a capacidade de atrapalhar o adversário, sendo todo o processo regulado, orientado e limitado, por regras aceitas, e obtendo, com isso, uma recompensa psicológica, normalmente na forma de diversão, entretenimento, ou sensação de vitória sobre um adversário ou desafio.(Xexéo, 2013, p. 4)

Portanto, esse trabalho irá abordar os jogos eletrônicos. Os jogos eletrônicos destacam-se como uma forma fascinante de entretenimento e comunicação. Atualmente, a indústria dos jogos eletrônicos lidera o setor de entretenimento em termos de receita, consolidando-se como a mais lucrativa globalmente. Além de seu impressionante desempenho econômico, o alcance e o impacto cultural dos jogos estão em contínua expansão, refletindo sua crescente influência no cenário mundial. (Coelho, 2022)

Portanto, este trabalho enfatiza os jogos *mobile*. Os jogos *mobile*, desenvolvidos especificamente para dispositivos móveis como *smartphones* e *tablets*, representam atualmente o maior segmento da indústria de jogos eletrônicos. Sua ascensão em popularidade pode ser atribuída a diversos fatores, como a evolução dos dispositivos móveis, que agora possuem maior potência e permitem o desenvolvimento de jogos mais sofisticados.(Coelho, 2022; Pereira, 2024)

Outro aspecto importante é que os jogos mobile oferecem uma experiência de entretenimento mais acessível e conveniente para jogadores de todas as idades e realidades. A simplicidade dos controles e a ampla variedade de gêneros disponíveis contribuem para

sua popularidade, tornando-os uma opção atrativa tanto para jogadores casuais quanto para os mais dedicados. Além disso, esse fator desempenha um papel fundamental na democratização do acesso aos jogos, permitindo que um público cada vez maior tenha contato com esse tipo de entretenimento.(Pereira, 2024)

Como vimos anteriormente, o mercado de jogos mobile é uma área promissora na indústria de tecnologia, combinando inovação, criatividade e alto potencial lucrativo. No entanto, apesar de sua crescente relevância, o desenvolvimento de jogos ainda é pouco abordado nos cursos de Tecnologia da Informação (TI). Muitas graduações priorizam áreas tradicionais como, por exemplo, desenvolvimento web, redes, segurança da informação, dispositivos móveis, etc., negligenciando e deixando de explorar outras áreas da tecnologia que podem representar uma excelente oportunidade de carreira para os futuros profissionais da área.(Andrade; Dantas, 2018)

Neste contexto, o projeto também tem o intuito de incentivar a exploração de novas áreas da tecnologia, despertar o interesse pelo desenvolvimento de jogos, suas aplicações e seu mercado, além de aprimorar diversas habilidades e contribuir para melhor formação acadêmica e profissional.

A questão central é: como aplicar os conhecimentos, habilidades e aprendizados adquiridos ao longo da graduação no desenvolvimento de um jogo para dispositivos móveis, gratuito e de código aberto, como parte do Trabalho de Conclusão de Curso? Esse é o objetivo proposto.

1.2 Objetivos

1.2.1 Objetivo geral

O objetivo deste trabalho é desenvolver um jogo bidimensional para dispositivos móveis com sistema operacional *Android*, utilizando a *Unity*3D, um motor de jogo popular e amplamente adotado na indústria. O projeto adotará uma estética visual em *pixel art*, cujos gráficos são formados pelas menores unidades de uma imagem digital, representando personagens, cenários e objetos com esse estilo artístico característico. O gênero escolhido é o *Survivors-like* um subgênero de *Roguelite* que são jogos conhecidos por sua progressão desafiadora e elevada dificuldade, características que proporcionam uma experiência de jogo intensa e de alta rejogabilidade.

1.2.2 Objetivos específicos

Os objetivos específicos deste trabalho são:

- Estudar as principais etapas do desenvolvimento de jogos, desde o planejamento até

a execução;

- Aprender a utilizar o motor de jogos *Unity* 3D, explorando suas funcionalidades e recursos.
- Desenvolver o produto final, aplicando as técnicas e ferramentas aprendidas durante o processo.
- Publicar o jogo em uma plataforma de distribuição de aplicativos digitais, tornando-o acessível ao público.

1.3 Justificativa

Desenvolver um jogo como Trabalho de Conclusão de Curso representa uma oportunidade única de aplicar, de maneira prática e criativa, os conhecimentos adquiridos ao longo da graduação. Esse processo não apenas reforça habilidades técnicas, mas também evidencia a capacidade de adaptação a novas tecnologias e possibilita a exploração de áreas não abordadas no curso, como o desenvolvimento de jogos. Dessa forma, permite o aprimoramento de competências essenciais, como programação, design e resolução de problemas, preparando-me melhor para os desafios do mercado de trabalho.

Além de consolidar o aprendizado acadêmico e a formação profissional, este trabalho também busca despertar o interesse pelo desenvolvimento de jogos e suas aplicações, incentivando sua integração e exploração dentro de um curso de graduação, destacando os benefícios que essa área pode trazer e ampliando as possibilidades de estudo e atuação no setor.

2 Fundamentação teórica

Este trabalho explora o universo dos jogos digitais. Antes de prosseguir, é importante entender alguns conceitos básicos, que serão fundamentais para o bom entendimento do conteúdo abordado ao longo do trabalho.

2.1 Jogos 2D

Jogos 2D (bidimensionais) são aqueles cujo ambiente e espaço são representados em um plano com duas dimensões: altura e largura, sem profundidade. Eles utilizam sprites (imagens ou animações) para representar personagens, cenários e objetos, diferentemente dos jogos 3D, que utilizam modelos tridimensionais para adicionar profundidade aos cenários e personagens.Devido à sua menor complexidade de desenvolvimento, os jogos 2D costumam ser mais acessíveis para desenvolvedores independentes.(Alves, G. B., 2021)

2.2 *Pixel Art*

O *Pixel Art* é uma forma de arte digital que utiliza pequenos quadrados de cor, os píxeis, para criar imagens e animações. Essa técnica possui uma estética característica, com píxeis bem aparentes e definidos. O estilo é frequentemente associado aos primeiros videogames, quando as limitações de hardware impediam o uso de gráficos mais sofisticados. Mesmo com o avanço da tecnologia, o *Pixel Art* continua sendo uma escolha estética popular, devido ao seu charme nostálgico, simplicidade visual e acessibilidade. Para desenvolvedores de jogos independentes, o *Pixel Art* é uma opção viável, pois não exige poderosas ferramentas de gráficos 3D, tornando o processo de criação mais acessível. (Bolívar Torres, 2023)

2.3 Game design

Game design é um termo amplo, mas pode ser definido como a área responsável pelo planejamento, idealização e organização da proposta de um jogo. O *game designer* tem a responsabilidade de equilibrar os diversos elementos que compõem o jogo, com o objetivo de proporcionar uma experiência significativa e envolvente para o jogador.(Alura, 2023a)

O processo de *game design* envolve a estruturação e o desenvolvimento de regras, mecânicas, comandos, narrativa, personagens, aspectos estéticos, objetivos, desafios, recompensas e progressão, entre outros componentes. Em outras palavras, trata-se de um

conjunto de dinâmicas e sistemas que se interconectam de forma coesa, proporcionando uma experiência interativa e divertida.(Davi Baptista, 2023)

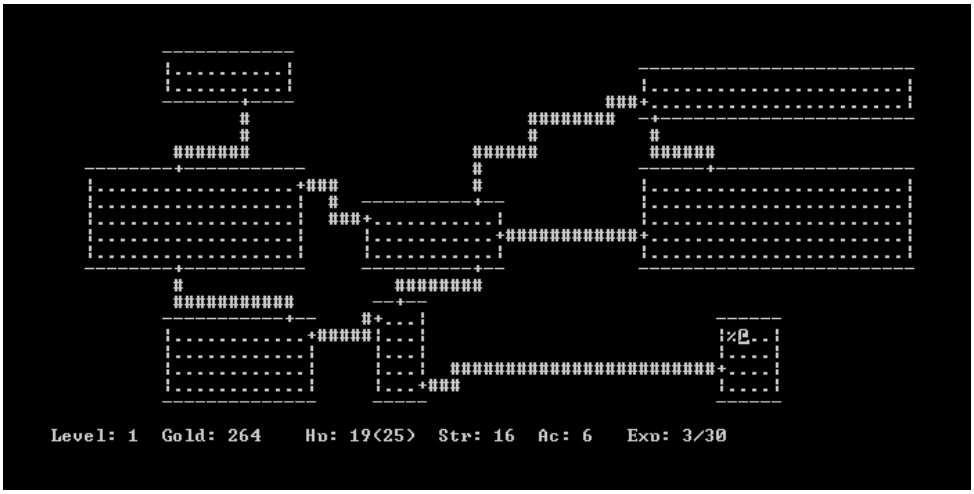
Game design é tanto uma arte quanto uma ciência, pois vai além dos aspectos visuais ou da programação. Trata-se de um campo multidisciplinar que combina diversas áreas do conhecimento, como psicologia, para entender como o jogo afeta as emoções e decisões do jogador, narrativa interativa, *design* de níveis (*level design*), *design* gráfico, trilha sonora, interface do usuário (*UI/UX*) e usabilidade. O *game designer* atua de forma colaborativa com a equipe de desenvolvimento, promovendo o diálogo, gerando ideias, incorporando sugestões e refinando continuamente a experiência de jogo.(Carpenedo, 2017a)

2.4 Gênero

2.4.1 *Rogue*

"*Rogue*"(1980) é um jogo que se tornou um marco no gênero dungeon crawl, (explorar masmorras).(Baramallo, 2023)

Figura 1 – *Uma partida de Rogue*



Fonte:https://spillhistorie.no/2024/07/14/the-story-of-rogue/,2025

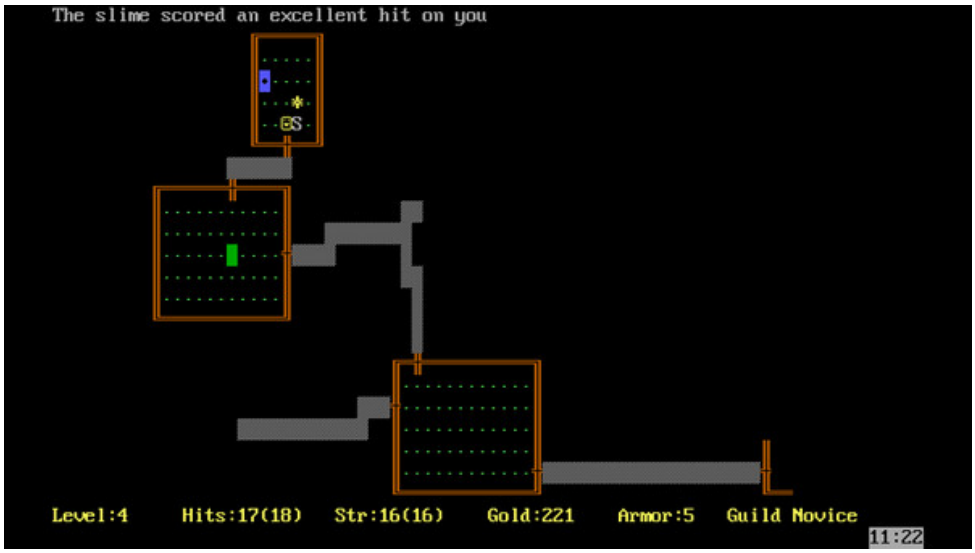
Suas principais características são:

- *Permadeath* (Morte Permanente): Se o jogador morre, perde todo o progresso e deve começar do zero.
- Gráficos ASCII: O jogo usa caracteres para representar o mundo, com símbolos como @ para o personagem e outras letras para monstros e itens.

- Geração Procedural: Cada nova partida cria um mapa único, tornando cada jogada diferente.
- Combate Baseado em Turnos: O jogo avança apenas quando o jogador faz uma ação, exigindo estratégia.
- Exploração e Recompensas: O jogador encontra armas, poções e pergaminhos que podem ajudar (ou prejudicar) sua jornada.

Em 1985, uma versão de Rogue foi desenvolvida pela Epyx. Posteriormente, essa versão foi adaptada e lançada na *Steam*.

Figura 2 – *Partida de Rogue(Epyx)*



Fonte:https://store.steampowered.com/app/1443430/Rogue/,2025

2.4.2 *Roguelike* (assim como *Rogue*)

O gênero *Roguelike* é um estilo de jogo que surgiu como uma referência ao jogo *Rogue* (1980), e, com o passar do tempo, esse estilo de jogo evoluiu, incorporando novas mecânicas e características que o tornaram distinto do jogo original. A essência do gênero permaneceu, no entanto, à medida que o gênero se expandia, as diferenças entre os jogos se tornaram mais evidentes, e o termo *Roguelike* passou a englobar uma gama mais ampla de experiências.

Em 2008, durante a Conferência Internacional de Desenvolvimento *Roguelike*, realizada em Berlim, na Alemanha, foram estabelecidas as diretrizes da *Berlin Interpretation*. Esse conjunto de critérios tinha como objetivo padronizar e definir o que caracteriza um jogo *Roguelike*, proporcionando uma base comum para os desenvolvedores e a comunidade de jogadores.(Baramallo, 2023)

2.4.3 *Roguelite* (*Roguelike* leve)

O gênero *Roguelite* é uma variação mais acessível do gênero *Roguelike*, preservando algumas de suas principais características, mas com mecânicas mais flexíveis e a incorporação de elementos de outros gêneros.

Um exemplo comum de mudança é o conceito de *Permadeath* (morte permanente), que nos jogos *Roguelite* foi adaptado para permitir que certos avanços, como o desbloqueio de habilidades, atualizações ou recursos, sejam mantidos entre as tentativas, oferecendo uma progressão mais contínua.(Baramallo, 2023)

2.4.4 *Survivor-like* (assim como *Vampire Survivors*)

Assim como *Rogue* deu origem ao gênero *Roguelike*, *Vampire Survivors* se tornou o precursor de um novo estilo de jogo. Ele popularizou uma abordagem inovadora de *gameplay*, caracterizada por alta rejogabilidade e pela combinação de elementos de ação, *Roguelike* e *Bullet Hell*. Nesse estilo, o jogador enfrenta hordas massivas de inimigos, coletando melhorias e evoluindo suas habilidades para sobreviver o máximo de tempo possível. (Epic Games, 2025)

2.5 Trabalhos similares

Os seguintes softwares proprietários, para jogos, foram encontrados durante esta pesquisa:

- ***Vampire Survivors***: é um jogo casual, gótico, de estilo *bullet hell* com elementos *roguelike*, onde suas escolhas podem fazer com que você cresça rapidamente e destrua milhares de monstros que surgem pelo caminho. Com uma jogabilidade minimalista, um visual chamativo e o desafio da progressão crescente, este jogo se tornou um verdadeiro fenômeno[1].

- ***Brotato***: é um jogo 2D de tiro em arena, no estilo *roguelite* e com visão aérea, onde você controla uma batata armada com até seis armas e enfrenta hordas de alienígenas. Escolha entre várias habilidades e itens para criar sua estratégia e sobreviver até o fim das hordas[2].

- ***20 Minutes Till Dawn***: é um jogo de sobrevivência *roguelite* onde você escolhe diferentes habilidades e suas melhorias para criar combos.Com uma seleção diversificada de personagens e armas, o jogo oferece uma experiência única a cada partida. O objetivo é sobreviver por 20 minutos, até o amanhecer.[3]

[1] Disponível em <https://store.steampowered.com/app/1794680/Vampire_Survivors/>
[2] Disponível em <https://store.steampowered.com/app/1942280/Brotato/>
[3] Disponível em <https://store.steampowered.com/app/1966900/20_Minutes_Till_Dawn/>

3 Materiais e métodos

3.1 Ciclo de vida e processos

3.1.1 Etapas do desenvolvimento de um jogo.

O desenvolvimento de um jogo pode variar conforme as especificidades do produto desejado, da equipe ou da empresa. No entanto, há etapas fundamentais que geralmente são seguidas, conforme destacado nos trabalhos (Peniche, 2024) e (Tokio, 2016).

Conceito/Concepção: Esta etapa envolve a formulação da ideia do jogo, abordando seus aspectos principais, como gênero, mecânicas básicas, enredo, público-alvo, entre outros. Com o conceito geral estabelecido, inicia-se a elaboração do GDD (*Game Design Document*), o documento de planejamento que guiará a equipe de desenvolvimento ao longo do projeto, conforme destacado nos trabalhos (Oliveira, 2020),(Naspolini, 2022a) e (Soares, 2021).

Pré-produção: Após a análise da viabilidade do jogo, começa a etapa da pré-produção, que consiste no planejamento detalhado de como o jogo será produzido. Nesta fase, o GDD é expandido com informações mais específicas sobre cada aspecto do jogo.(Farmando XP, 2023)

Protótipo: Criação de uma versão limitada e simplificada do jogo, para testar e validar as principais ideias e mecânicas. Este protótipo permite avaliar na prática se o jogo será divertido e envolvente antes de investir na produção completa. Conforme destacados nos trabalhos (Soares, 2020) e (Naspolini, 2022b), se o protótipo não atender aos requisitos esperados, o ciclo de desenvolvimento é reiniciado para refinar as mecânicas e aprimorar a experiência do jogo, garantindo uma base sólida para a produção completa.

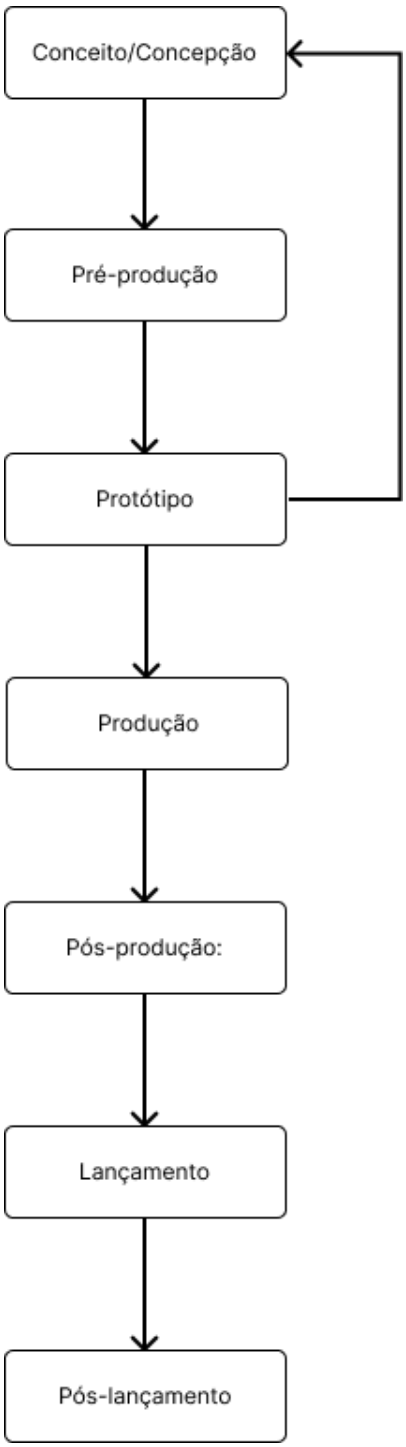
Produção: É a etapa em que o jogo começa a ser construído de fato. Isso inclui o desenvolvimento das mecânicas, mapas, personagens, interfaces, ambientes e suas interações, texturas, efeitos visuais e outros componentes essenciais.(Tokio, 2016)

Pós-produção: Com o jogo quase pronto, temos uma versão Alfa, onde testes são realizados em diversos aspectos, como desempenho, jogabilidade, gráficos, efeitos sonoros, nível de dificuldade, equilíbrio do jogo, etc., a fim de realizar melhorias e correções de erros, levando à criação de uma versão Beta, que é disponibilizada ao público para mais testes e refinamentos.(Farmando XP, 2023)

Lançamento: Após os testes de qualidade, o jogo atinge o estado chamado *Gold* (ouro), indicando que está pronto para ser lançado.(Tokio, 2016)

Pós-lançamento: Após o lançamento, o jogo pode receber atualizações de conteúdo ou correções de problemas. Dependendo do tipo de jogo, podem ser lançadas expansões, eventos ou novos recursos para manter o interesse dos jogadores e garantir a longevidade do produto.(Tokio, 2016)

Figura 3 – Etapas do desenvolvimento de um jogo



Fonte: elaborado pelo autor, 2025

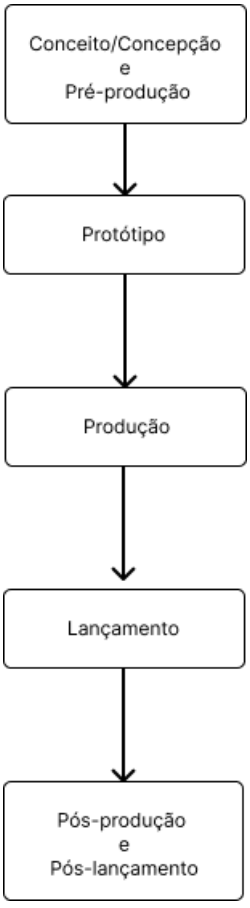
3.1.2 Etapas do desenvolvimento do produto.

No geral, as etapas fundamentais do desenvolvimento foram contempladas, embora a sequência adotada tenha divergido do fluxo tradicional. Ao examinar o processo de criação do produto, destacam-se os seguintes aspectos relevantes:

A fase de Conceito/Concepção e Pré-produção foi unificada e acelerada, visto que o projeto não se trata de um produto comercial, cujo resultado final não implicaria grandes consequências, como impactos financeiros. Dessa forma, o planejamento foi mais enxuto, iniciando com uma ideia central e um planejamento básico. Ao longo do desenvolvimento, o escopo foi ajustado conforme as limitações e desafios encontrados, permitindo que o conceito evoluísse de maneira prática e alinhada às necessidades do momento.

A Pós-produção foi realizada em paralelo com a fase de Pós-lançamento. O lançamento da versão Alfa ocorreu, seguido por testes públicos. Em seguida, várias versões Beta foram desenvolvidas e compartilhadas para testes adicionais e refinamento do jogo.

Figura 4 – Como foram as etapas do desenvolvimento do jogo.



Fonte: elaborado pelo autor, 2025

Em resumo, o processo adotou uma abordagem mais flexível, adaptando as etapas conforme as necessidades do projeto, sem seguir rigidamente a ordem tradicional.

3.2 Ferramentas e tecnologias utilizadas

3.2.1 *Game Engine*

Um Motor de jogo (ou *game engine*) é um *software* que abstrai e facilita o desenvolvimento de jogos. Esse tipo de programa oferece ferramentas e bibliotecas integradas . Ele gerencia aspectos essenciais como gráficos, física, áudio e entrada do usuário, permitindo que os desenvolvedores foquem na criação da jogabilidade sem precisar programar do zero. Essencialmente, ele funciona como um *framework*, fornecendo uma base estruturada para otimizar o processo de desenvolvimento e tornar a produção de jogos mais acessível e eficiente.(Carpenedo, 2017b)

Um motor de jogo é composto por diversos componentes que gerenciam aspectos essenciais do desenvolvimento (Valeri, 2023), como:

- Gerenciamento de entrada do usuário: processa toques, teclas e comandos de interação.
- Motor físico: simula gravidade, colisões e outras interações físicas.
- Motor gráfico: renderiza animações, efeitos visuais e elementos do jogo.
- Motor de áudio: gerencia sons e trilhas sonoras.

Com esses componentes prontos para serem usados e centralizados em um único *software*, é possível construir o jogo sem a necessidade de começar do zero. Não há necessidade de usar linguagens de baixo nível, como linguagem de máquina, para programar as interações com o *hardware*. O motor de jogo já abstrai e cuida dessa parte complexa, permitindo que o foco esteja no desenvolvimento do jogo.(Naspolini, 2024)

Grandes empresas de desenvolvimento de jogos costumam utilizar motores proprietários para otimizar o desempenho e personalizar funcionalidades de acordo com suas necessidades (Montenegro, 2023). Como podemos observar no trabalho de (Valeri, 2023), diversas empresas podem utilizar diferentes motores. A seguir, destacarei alguns exemplos:

- *Bethesda*:*Creation Engine*.
- *CD Projekt Red*: *REDengine*.
- *Sony Interactive Entertainment*: *PhyreEngine*.
- *Square Enix: Crystal Tools e Luminous Engine*.
- *Rockstar Games*: *Rockstar Advanced Game Engine* (RAGE).

Além dos motores proprietários, muitas empresas usam motores de jogo públicos e gratuitos:

- *Unity* 3D.
- *Unreal Engine*.
- *Godot*.
- *GameMaker*.

3.2.2 Qual motor de jogos usar?

O motor de jogos deve atender às especificidades do projeto e da equipe de desenvolvimento. Fatores importantes para a escolha são o tipo de jogo e a plataforma alvo. Por exemplo, um jogo grande com gráficos realistas deverá optar pela *Unreal Engine* [1], para jogos pequenos ou médios 3D e 2D as opções mais versáteis são *Unity* 3D[2] ou *Godot*[3]. Portanto, cada motor tem seus prós e contras; a escolha do motor depende diretamente do estilo do jogo, da empresa e de seus objetivos.(Felipe, 2024; Farmando XP, 2024)

3.2.2.1 Por que usar *Unity* 3D?

A escolha da Unity se deve ao fato de ser uma das ferramentas mais utilizadas no mercado de trabalho para o desenvolvimento de jogos, conforme apontado nos trabalhos (Junior, 2024; EBAC, 2023; Dio, 2023; Felipe, 2023; Brodi, 2023).

> "[...]para desenvolver os jogos, o ambiente mais utilizado é o Unity. Dentre os motivos citados para utiliza-lo estão: a praticidade, a facilidade em encontrar materiais de apoio, e o fato de ser multiplataforma."(Andrade; Dantas, 2018, p.30)

Outro fator determinante é o suporte e a comunidade ativa do ecossistema *Unity* 3D, possibilitando o acesso a tutoriais, fóruns e recursos que auxiliam na solução de problemas e na otimização do desenvolvimento do jogo.

3.2.3 Vs code

O Visual *Studio Code* (VS *Code*)[4] é um editor de código-fonte gratuito, desenvolvido pela *Microsoft*. Leve e altamente personalizável, ele oferece um ecossistema avançado de extensões, permitindo a adaptação para diversas necessidades de desenvolvimento.

[1] Disponível em <https://www.unrealengine.com/pt-BR>
[2] Disponível em <https://unity.com/pt>
[3] Disponível em <https://godotengine.org/>
[4] Disponível em <https://code.visualstudio.com/>

Sua extensibilidade possibilita a integração com o *Unity* 3D e suporte ao C#, tornando-o uma ferramenta eficiente para o desenvolvimento de jogos. (Microsoft, 2025a)

3.2.4 Linguagem de programação

3.2.4.1 C# (C-Sharp)

Como linguagem de programação, foi utilizado o C# (C-Sharp) devido à sua forte integração com o *Unity* 3D.

Além disso, o C# é uma linguagem orientada a objetos, o que favorece a organização e manutenção do código, permitindo reutilização de componentes e modularidade no desenvolvimento.

Como podemos ver na página "Um tour pela linguagem C#"da própria Microsoft

> "O C#(C-Sharp) é uma linguagem de uso geral multiplataforma que torna os desenvolvedores produtivos ao escrever um código de alto desempenho. Com milhões de desenvolvedores, o C# (C-Sharp) é a linguagem .NET mais popular. O C# (C-Sharp) tem amplo suporte no ecossistema e em todas as cargas de trabalho do .NET. Com base em princípios orientados a objetos, ele incorpora muitos recursos de outros paradigmas, especialmente a programação funcional. Recursos de baixo nível dão suporte a cenários de alta eficiência sem escrever código não seguro. A maioria dos runtimes e bibliotecas do .NET são escritos em C# (C-Sharp) e avanços no C# (C-Sharp) geralmente beneficiam todos os desenvolvedores do .NET."(Microsoft, 2025b)

3.2.5 Krita

O Krita[5] é um programa de pintura profissional gratuito e de código aberto. É criado por artistas que desejam ter ferramentas artísticas acessíveis para todos.Neste trabalho, o Krita foi utilizado para realizar pequenos ajustes nos recursos gráficos utilizados.

3.2.6 Figma

O Figma[6] é uma ferramenta de design de interface, usada para criar *layouts* de sites, aplicativos e prototipagem interativa. Ele permite colaboração em tempo real, facilitando o trabalho em equipe.Utilizado para organizar algumas ideias de maneira visual e criar elementos gráficos utilizados nesse trabalho e no protótipo. No contexto deste trabalho, o Figma foi utilizado para organizar visualmente algumas ideias e criar os elementos gráficos presentes tanto na elaboração do projeto quanto no protótipo.

[5] Disponível em <https://krita.org/pt-pt/>
[6] Disponível em <https://www.figma.com/pt-br/>

3.2.7 Gitlab

O *GitLab*[7] é uma plataforma de *DevOps* baseada em *Git*, que permite o versionamento de código, automação de testes e *deploys* (CI/CD), além de ferramentas para gerenciamento de projetos. Ele pode ser usado na nuvem ou em servidores próprios, oferecendo segurança e controle para equipes de desenvolvimento. Além disso, o GitLab é uma plataforma de *open source*(código aberto). A versão do GitLab[8] utilizada no IFRO é executada localmente.

3.2.8 Kanban

O Kanban é uma metodologia de gerenciamento de fluxo de trabalho que visa melhorar a eficiência e a produtividade. Ele utiliza um quadro visual (físico ou digital), com colunas que representam as diferentes etapas do processo. À medida que as tarefas avançam, elas são movidas de uma coluna para outra, refletindo seu progresso.(Alura, 2023b; TOTVS, 2023)

Originalmente desenvolvido pela Toyota na década de 1940 para aprimorar a produção industrial, o Kanban foi adaptado e hoje é amplamente utilizado em diversas áreas, como desenvolvimento de software e gestão de projetos.(Boeg, 2018)

O método ajuda a visualizar o andamento das tarefas, otimizar os processos e aumentar a eficiência. No painel Kanban, as tarefas são organizadas em colunas que representam diferentes estágios do processo (ex.: "A Fazer", "Em Progresso", "Concluído"). As tarefas, representadas por cartões, são movidas entre as colunas conforme avançam no fluxo de trabalho.(Lomelino, 2021)

3.2.9 Google forms

Google forms[9], ferramenta utilizada para criar formulários e receber *feedbacks* dos usuários, que foi fundamental para identificar problemas e propor melhorias no jogo.

3.3 Requisitos

Os requisitos funcionais e não-funcionais foram elaborados com base no GDD (Game Design Document) desenvolvido no início do processo de criação do projeto.

A seguir, são apresentados os requisitos funcionais, que descrevem as funcionalidades essenciais do jogo, e os requisitos não-funcionais, que tratam das especificações relativas à plataforma, qualidade e usabilidade.

[7] Disponível em <https://about.gitlab.com/>
[8] Disponível em <https://gitlab.fslab.dev/users/sign_in>
[9] Disponível em <https://workspace.google.com/intl/pt-BR/products/forms/>

3.3.1 Requisitos Funcionais (RF)

Jogabilidade	
RF01	O jogo deve permitir que o jogador controle um personagem em um ambiente 2D.
RF02	O personagem do jogador deve atacar automaticamente os inimigos sem a necessidade de comandos adicionais.
RF03	O jogo deve gerar inimigos de forma progressiva, aumentando a dificuldade ao longo do tempo.
RF04	O jogo deve incluir diferentes tipos de inimigos, cada um com padrões de dificuldades distintos.
RF05	O personagem do jogador deve ganhar experiência ao derrotar inimigos e poder evoluir ou desbloquear novas armas.
RF06	O jogo deve terminar quando o jogador perde toda a sua vida ou atinge um objetivo específico.
Interface e Experiência do Usuário	
RF07	O jogo deve exibir um HUD com informações como tempo de jogo, vida do personagem e nível atual.
RF08	Deve haver um sistema de pausa para interromper temporariamente a partida.
Progresso e Personalização	
RF09	Deve haver um sistema de progressão para permitir a melhoria de habilidades e equipamentos ao longo da partida.
RF10	O jogo deve salvar o progresso do jogador entre sessões.

Tabela 1 – Requisitos Funcionais do Jogo

3.3.2 Requisitos Não-Funcionais (RNF)

Plataformas e Tecnologias	
RNF01	O jogo será desenvolvido na engine Unity.
RNF02	Deve ser compatível com dispositivo móvel.
RNF03	Deve ser compatível com sistema operacional Android.
RNF04	O jogo deve suportar entrada via teclado e controle.
Qualidade e Manutenção	
RNF05	Deve seguir boas práticas de desenvolvimento, incluindo versionamento do código (ex.: Git).
Estética e Áudio	
RNF06	A trilha sonora e os efeitos sonoros devem se adequar ao ritmo do jogo.
RNF07	Os gráficos devem ser estilizados para manter uma identidade visual coerente com o tema do jogo.
Acessibilidade e Usabilidade	
RNF08	O jogo deve ter uma interface intuitiva e fácil de aprender.

Tabela 2 – Requisitos Não-Funcionais do Jogo

3.4 Arquitetura do *software*

A arquitetura de componentes na *Unity* 3D é baseada no padrão *Entity-Component-System* (ECS) em português: Sistema de Componente e Entidade, que divide os elementos do jogo em três partes essenciais: entidades (objetos no jogo, como personagens ou itens), componentes (dados que descrevem as entidades, como posição, velocidade, etc.) e sistemas (lógica que opera sobre esses componentes, como movimentação ou física).(Pavarini et al, 2024, p.24)

Exemplo: Se você tem um *GameObject* chamado "*Player*", ele pode ter os seguintes componentes:

- *Transform* (componente que lida com posição, rotação e escala).
- *Rigidbody* (para simulação de física e movimentação).
- *Collider* (para detecção de colisões).
- *Script* (um script que acessa e manipula outros componentes, definindo o comportamento do jogador, como mover-se ou pular).

Como explicado no trabalho (Pavarini et al, 2024, p.24), esses componentes são anexados ao *GameObject*, e a *Unity* 3D encarrega-se de gerenciar sua execução no ciclo de vida do jogo.

> A principal vantagem dessa arquitetura é a modularidade. Como os componentes são desacoplados, é possível adicionar ou modificar comportamentos sem alterar o restante do código, o que facilita a manutenção e o desenvolvimento de jogos de grande escala. O *Unity* permite implementar essa arquitetura utilizando C#, uma linguagem orientada a objetos que possibilita a criação de scripts que manipulam esses componentes de forma eficiente.(Pavarini et al, 2024, p.25)

Vantagens:

- Modularidade: Componentes são modulares e podem ser adicionados, removidos ou trocados facilmente.
- Reutilização: Comportamentos podem ser reutilizados em diferentes *GameObjects*.
- Simplicidade: Facilita o entendimento do código, pois cada componente tem uma função específica.
- Flexibilidade: Permite que um *GameObject* seja configurado de maneiras diferentes, apenas alterando os componentes que ele possui.

Foram utilizados vários padrões de projeto, isso ocorre porque, em um jogo, diferentes partes do sistema frequentemente têm requisitos distintos que não podem ser atendidos apenas por um único padrão de projeto. Em muitos casos, as partes do jogo exigem soluções específicas para atender a essas necessidades. Por isso, outros padrões de projeto são utilizados para resolver essas questões de maneira eficiente, garantindo que o jogo seja modular, escalável e fácil de manter.

Cito apenas alguns exemplos, pois as demais abordagens representam características muito pontuais que foram utilizadas, sem envolver a implementação completa e, portanto, correta, do padrão de projeto.

3.4.1 *ScriptableObjects*

Um *ScriptableObject* é um contêiner de dados que permite armazenar informações de forma eficiente e independente de objetos específicos na cena. Um dos principais casos de uso dos *ScriptableObjects* é a redução do uso de memória, permitindo que diferentes partes do projeto acessem os mesmos dados, evitando a criação de cópias desnecessárias.(Unity, s.d.b)

Figura 5 – Código do *ScriptableObject* da *Wave* (Onda de inimigos).

```
1  using System.Collections;
2  using System.Collections.Generic;
3  using UnityEngine;
4
5  [CreateAssetMenu(fileName = "Wave", menuName = "Wave", order = 1)]
6  public class Wave : ScriptableObject
7  {
8      public List<EnemyObject> enemies = new List<EnemyObject>();
9      public int n_enemies;
10
11     public float timeBetweenSpawns = 1f;
12
13     public float time_between_waves = 1f;
14
15 }
```

Fonte: elaborado pelo autor, 2025

Um exemplo pode ser visto na Figura 36.

3.4.2 *Singleton*

Singletons é um padrão de *design* utilizado para garantir que uma classe tenha apenas uma instância em todo o sistema e forneça um ponto de acesso global a essa instância,como explicado no trabalho de (Bento, 2020).

Uma precaução importante ao utilizar Singletons é que eles criam uma dependência global entre diferentes partes do código, o que pode aumentar o acoplamento. Isso adiciona uma camada extra de complexidade ao sistema, tornando-o mais difícil de compreender e testar, especialmente à medida que o projeto cresce. Além disso, pode levar a problemas de manutenção, já que alterações na implementação do Singleton podem afetar várias partes do código simultaneamente.

Figura 6 – Código Singleton

```
1  private void Awake()
2  {
3      //Se houver um instance, e não for eu, exclua-me.
4
5      if (Instance != null && Instance != this)
6      {
7          Destroy(this);
8      }
9      else
10     {
11         Instance = this;
12     }
13 }
```

Fonte: elaborado pelo autor, 2025

Um outro exemplo pode ser visto na Figura 14.A função *Awake* é a primeira a ser executada ao iniciar o objeto. Ela verifica se já existe uma instância ativa. Caso não haja, a instância atual é atribuída e configurada para não ser destruída ao carregar uma nova cena. Se uma instância já existir e não for a mesma que a atual, a nova instância será automaticamente destruída.

3.4.3 *Object pool pattern*

Object Pooling é um padrão de *design* que otimiza o desempenho ao evitar a criação e destruição repetida de objetos. Em vez disso, objetos são criados e desativados antecipadamente, sendo armazenados em um agrupamento (*pool*). Quando necessário, um objeto é retirado do *pool*, ativado, usado e, ao ser descartado, é desativado e devolvido ao

agrupamento (*pool*) para reutilização. Isso economiza recursos da *CPU* e melhora a gestão de memória, reduzindo a sobrecarga da coleta de lixo.

Estou utilizando esse padrão de *design* nos inimigos, pop de dano, experiência, ouro, armas e projéteis. (Unity, s.d.c; Unity, s.d.a)

Figura 7 – Código IReservaDeObjetos

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public interface IReservaDeObjetos
{
    void ColocarNaReserva(GameObject objeto);
    GameObject PegarDaReserva();
    bool TemObjetoNaReserva();
}

```

Fonte: elaborado pelo autor, 2025

A interface IReservaDeObjetos permite que outras classes interajam com o sistema de reserva sem depender da sua implementação específica, promovendo desacoplamento e reutilização.

Figura 8 – Código IReservavel

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public interface IReservavel
{
    void SetReserva(IReservaDeObjetos reserva);

    void AoEntrarNaReserva();

    void AoSairDaReserva();

}

```

Fonte: elaborado pelo autor, 2025

Um objeto que implementa a interface IReservavel pode ser notificado quando entra ou sai da reserva, além de manter uma referência à reserva à qual pertence.

Figura 9 – Código da Reserva

```
6  public class Reserva : MonoBehaviour, IReservaDeObjetos
7  {
8
9      [SerializeField]
10     public GameObject prefab;
11     [SerializeField]
12     public int quantidade;
13
14     //[SerializeField] public Stack<GameObject> reserva = new Stack<GameObject>();
15    public List<GameObject> reserva = new List<GameObject>();
16     public List<GameObject> pool = new List<GameObject>();
17
18     private void Awake()
19     {
20         //this.reserva = new Stack<GameObject>();
21         this.CriarTodosOsObjetos();
22     }
23
24     private void CriarTodosOsObjetos()
25     {
26         for (int i = 0; i < this.quantidade; i++)
27         {
28             this.CriarNovoObjeto();
29         }
30     }
31
32     private void CriarNovoObjeto()
33     {
34         var objeto = GameObject.Instantiate(this.prefab, this.transform);
35         var objetoReservavel = objeto.GetComponent<IReservavel>();
36         objetoReservavel.SetReserva(this);
37         pool.Add(objeto);
38         this.ColocarNaReserva(objeto);
39     }
40
41     public void ColocarNaReserva(GameObject objeto)
42     {
43         objeto.SetActive(false);
44         //this.reserva.Push(objeto);
45
46         if (!reserva.Contains(objeto)) // Evita adicionar duplicatas
47         {
48             reserva.Add(objeto);
49         }
50
51         var objetoReservavel = objeto.GetComponent<IReservavel>();
52         objetoReservavel.AoEntrarNaReserva();
53     }
```

Fonte: elaborado pelo autor, 2025

A classe da reserva cria no antes do início uma quantidade pré-definida de objetos, desativando e colocando-os na reserva, garantindo que não haja duplicatas. Ela também chama o método AoEntrarNaReserva() para cada objeto reservado.

Figura 10 – Código da Reserva

```
55      public GameObject PegarDaReserva()
56      {
57          if (this.reserva.Count <= 0)
58          {
59              this.CriarNovoObjeto();
60          }
61
62          var objeto = reserva[0]; // Pega o primeiro disponível
63          reserva.RemoveAt(0); // Remove da reserva
64
65          var objetoReservavel = objeto.GetComponent<IReservavel>();
66          objetoReservavel.AoSairDaReserva();
67
68
69      return objeto;
70      }
71
72      public bool TemObjetoNaReserva()
73      {
74          return this.reserva.Count > 0;
75      }
76
77      public void GarantirQuantidadeNaReserva()
78      {
79
80          if (reserva.Count <= quantidade)
81          {
82              this.CriarNovoObjeto();
83              GarantirQuantidadeNaReserva();
84          }
85
86          // while (reserva.Count < quantidade)
87          // {
88          //     CriarNovoObjeto();
89          // }
90      }
91
92  }
```

Fonte: elaborado pelo autor, 2025

Ao solicitar um objeto, o sistema retira o primeiro objeto da reserva, chama o método AoSairDaReserva() do objeto e retorna.

Figura 11 – Código da ReservaExtensivel

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class ReservaExtensivel : MonoBehaviour , IReservaDeObjetos
{
    [SerializeField]
    private GameObject prefab;

    private Stack<GameObject> reserva;

    private void Awake()
    {
        this.reserva = new Stack<GameObject>();

    }

    private void CriarNovoObjeto()
    {
        var objeto = GameObject.Instantiate(this.prefab, this.transform);
        var objetoReservavel = objeto.GetComponent<IReservavel>();
        objetoReservavel.SetReserva(this);
        this.ColocarNaReserva(objeto);
    }

    public void ColocarNaReserva(GameObject objeto)
    {
        objeto.SetActive(false);
        this.reserva.Push(objeto);
        var objetoReservavel = objeto.GetComponent<IReservavel>();
        objetoReservavel.AoEntrarNaReserva();
    }

    public GameObject PegarDaReserva()
    {
        if (this.reserva.Count <= 0)
        {
            this.CriarNovoObjeto();
        }
        var objeto = this.reserva.Pop();
        var objetoReservavel = objeto.GetComponent<IReservavel>();
        objetoReservavel.AoSairDaReserva();
        return objeto;
    }

    public bool TemObjetoNaReserva()
    {
        return true;
    }

}

```

Fonte: elaborado pelo autor, 2025

A implementação ReservaExtensivel também segue o contrato de IReservaDeObjetos, mas com a diferença de que os objetos são criados sob demanda, conforme necessário.

Figura 12 – Código do EnemyReservavel

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class EnemyReservavel : MonoBehaviour, IReservavel
{
    // Aplica o Enemy Scriptable Object no Inimigo
    private IReservaDeObjetos reserva;

    public void SetReserva(IReservaDeObjetos reserva)
    {
        this.reserva = reserva;
    }

    public void VoltarParaReserva()
    {
        this.reserva.ColocarNaReserva(this.gameObject);
    }

    public void AoEntrarNaReserva()
    {
        this.gameObject.SetActive(false);
    }

    public void AoSairDaReserva()
    {
        //this.gameObject.SetActive(true);
    }


}

```

Fonte: elaborado pelo autor, 2025

Já a classe EnemyReservavel implementa IReservavel e representa inimigos reutilizáveis, permitindo que eles retornem automaticamente à reserva quando forem derrotados.

3.4.4 Outras padroes de projeto

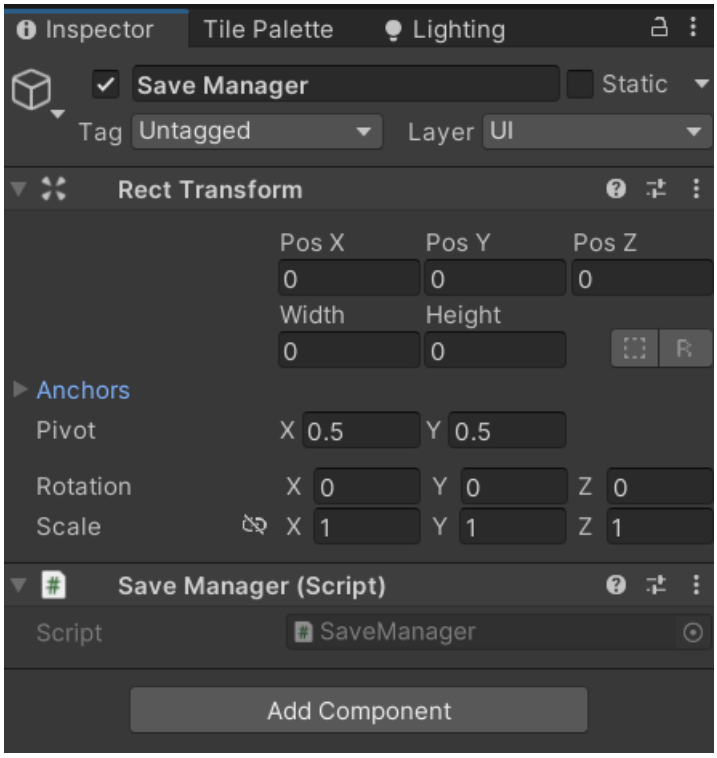
Alguns conceitos de outros padrões de projeto foram utilizados. Menções honrosas: *Factory Pattern*, *State Pattern*, Arquitetura orientada a eventos, *Observer Pattern*.

3.5 Persistência de dados

3.5.1 *SaveManager*

O *SaveManager* é o Gerenciador de Salvamento, ele é o responsável por criar,e acessar o arquivo .JSON.

Figura 13 – Janela *Inspetor* do *SaveManager*



Fonte: elaborado pelo autor, 2025

No objeto *SaveManager*, na janela *Inspector*, é possível visualizar seus componentes e verificar suas informações. Um dos componentes é o *script SaveManager*, anexado a esse objeto.

Figura 14 – Código do *SaveManager*

```
using System.IO;

public class SaveManager : MonoBehaviour
{
    public static SaveManager Instance;
    private static string fileName = "goldData.json";

    private void Awake()
    {

        if (Instance == null)
        {
            Instance = this;
            DontDestroyOnLoad(gameObject);
        }
        else
        {
            Destroy(gameObject);
        }

    }
    // Função para salvar a quantidade de ouro em um arquivo JSON
    public void SaveGold(int currentGold)
    {
        // PlayerPrefs.SetInt("GoldAmount", currentGold);
        // PlayerPrefs.Save();  // Força a gravação no disco

        GoldData goldData = new GoldData();
        goldData.goldAmount = currentGold;

        string json = JsonUtility.ToJson(goldData);

        // Caminho para o arquivo onde o ouro será salvo
        string filePath = Path.Combine(Application.persistentDataPath, fileName);
        File.WriteAllText(filePath, json);  // Salva o JSON no arquivo

        // Debug.Log("Gold Saved: " + filePath);
    }

```

Fonte: elaborado pelo autor, 2025

Esse *script* é um *singleton*, responsável por criar um objeto do tipo *GoldData*, convertê-lo em uma string JSON e salvar o arquivo em um diretório seguro.

Figura 15 – Código do *SaveManager*

```
// Função para carregar a quantidade de ouro em um arquivo JSON
    public int LoadGold()
    {
        //Retorna 0 caso não haja ouro salvo
        //return PlayerPrefs.GetInt("GoldAmount", 0);

        string filePath = Path.Combine(Application.persistentDataPath, fileName);

        if (File.Exists(filePath))
        {
            string json = File.ReadAllText(filePath); // Lê o conteúdo do arquivo JSON
            GoldData goldData = JsonUtility.FromJson<GoldData>(json); //Desserializa o JSON para o objeto GoldData
            return goldData.goldAmount;  // Retorna a quantidade de ouro salva
        }
        else
        {
            //Debug.Log("No saved gold data found.");
            return 0;  // Se não houver arquivo, retorna 0 (sem ouro salvo)
        }
    }
}

[System.Serializable]
public class GoldData
{
    public int goldAmount;
}
```

Fonte: elaborado pelo autor, 2025

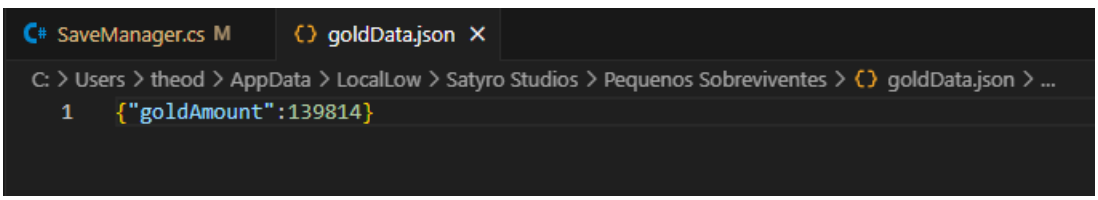
Verifica se o arquivo já existe no caminho especificado. Se existir, ele lê o conteúdo, converte o JSON de volta para um objeto *GoldData* e retorna a quantidade de ouro. Caso contrário, retorna zero.

Observe que há um código antigo comentado, que foi descontinuado, pois a abordagem de usar o *PlayerPrefs* (Preferências do Jogador) é destinada a armazenar preferências do jogo, como configurações de volume, tela, entre outras. Não é recomendado utilizá-lo para salvar dados grandes e importantes. Embora essa prática seja comum, ela é desaconselhada.

3.5.2 Arquivo goldData.json

Arquivo goldData.json, armazena a quantidade de ouro localmente no dispositivo.

Figura 16 – *goldData*.json

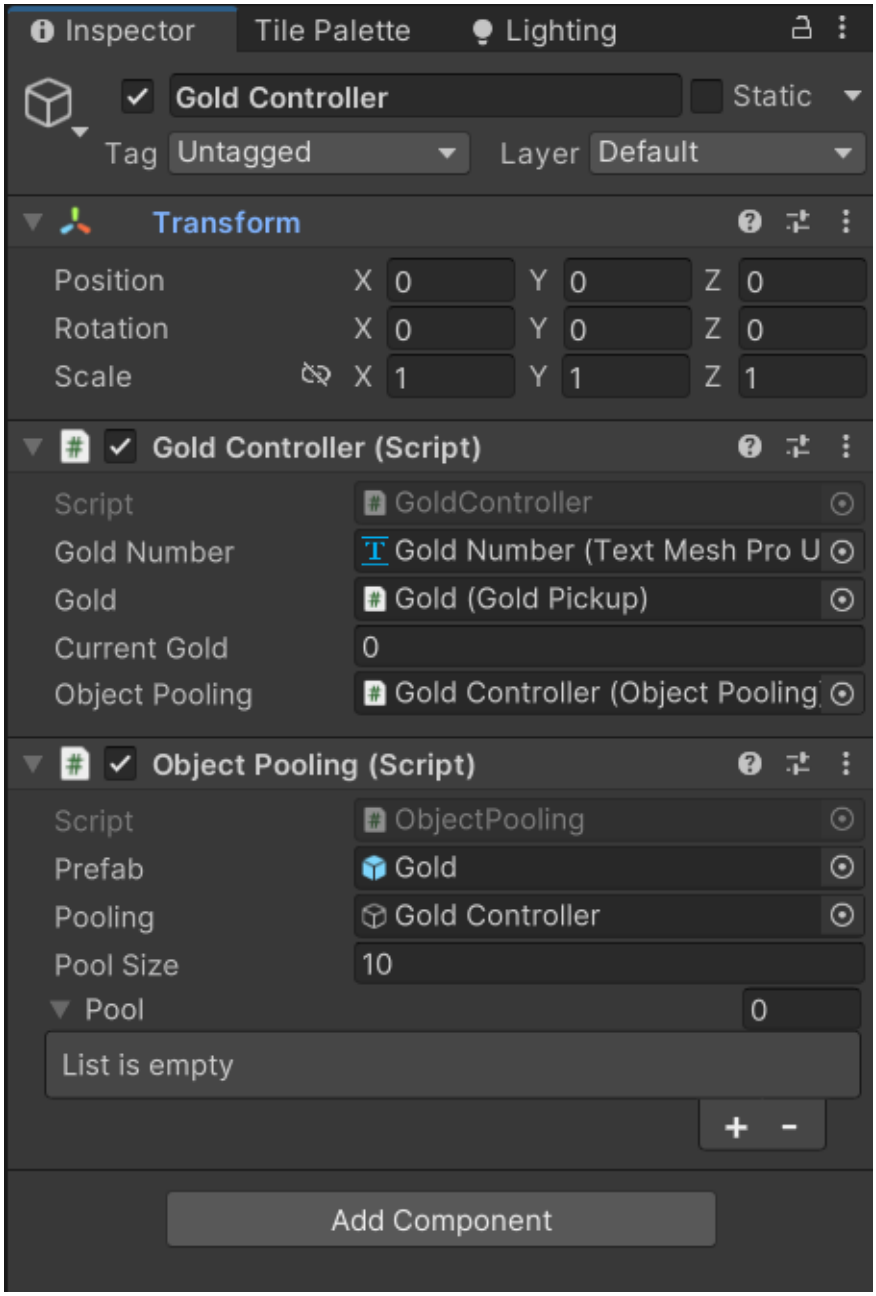


Fonte: elaborado pelo autor, 2025

3.5.3 *GoldController*

O *GoldController* controla a quantidade de ouro, utilizando a instância Singleton do SaveManager para carregar e alterar o valor do ouro. Ele também é responsável por atualizar o número de ouro na UI (Interface do Usuário) e disponibiliza funções para que outros códigos possam adicionar ou retirar a quantidade de ouro e colocar objetos de ouro na cena.

Figura 17 – Janela *Inspetor* do *GoldController*



Fonte: elaborado pelo autor, 2025

Figura 18 – Código *GoldController*

```
using System.Collections;
using System.Collections.Generic;
using TMPro;
using UnityEngine;

public class GoldController : MonoBehaviour
{
    public static GoldController Instance;
    public TMP_Text goldNumber;

    public GoldPickup gold;

    public int currentGold;

    public ObjectPooling objectPooling;

    private void Awake()
    {
        Instance = this;
    }

    void Start()
    {
        objectPooling = GetComponent<ObjectPooling>();
        objectPooling.prefab = gold.gameObject;

        // Carregar a quantidade de ouro salva
        currentGold = SaveManager.Instance.LoadGold();
        // UIController.Instance.UpdateCoins();
        UpdateCoins(); // Atualiza a UI com a quantidade carregada
    }



    public void AddCoins(int coinsToAdd)
    {
        currentGold += coinsToAdd;
        //UIController.Instance.UpdateCoins();
        UpdateCoins();
        SaveManager.Instance.SaveGold(currentGold);  // Salvar ouro depois de adicionar
        SFXManager.Instance.PlaySFXPitched(SFXManager.Instance.More, 1);
    }
    public void UpdateCoins()
    {
        goldNumber.text = currentGold.ToString();
    }
```

Fonte: elaborado pelo autor, 2025

No início acessa a instância do *SaveManager*, obtém a quantidade de ouro armazenada e atualiza a interface do usuário com o novo valor.

A função *addCoins* adiciona a quantidade de ouro recebida, altera o texto exibido para o usuário refletindo o novo valor, salva essa atualização utilizando o *SaveManager* e

emite o efeito sonoro correspondente por meio do *SFXManager* Figura 35.

Figura 19 – Código do *GoldController*

```
public void DropCoin(Vector3 position, int value)
    {
        // GoldPickup newGoldBag = Instantiate(gold, position + new Vector3(.2f, .1f, 0f), Quaternion.identity);
        // newGoldBag.coinAmount = value;
        //newGoldBag.gameObject.SetActive(true);

        GameObject GameObject_GoldBag;
        GameObject_GoldBag = objectPooling.GetPooledObject();
        GameObject_GoldBag.transform.position = position + new Vector3(0.2f, 0.1f, 0f);
        GameObject_GoldBag.transform.rotation = Quaternion.identity;

        GoldPickup newGoldBag = GameObject_GoldBag.GetComponent<GoldPickup>();
        newGoldBag.coinAmount = value;
        newGoldBag.gameObject.SetActive(true);

    }

    public void SpendCoins(int coinsToSpend)
    {
        currentGold -= coinsToSpend;
        //UIController.Instance.UpdateCoins();
        UpdateCoins();
        SaveManager.Instance.SaveGold(currentGold);  // Salvar ouro depois de gastar
    }
}
```

Fonte: elaborado pelo autor, 2025

A função *DropCoin* retira um objeto de ouro da reserva, altera sua posição, atribui o valor correspondente a esse objeto e, em seguida, ativa o objeto.

A função *SpendCoins* diminui a quantidade de ouro, atualiza a interface do usuário e realiza o salvamento.

3.6 Plano de testes

3.6.1 Teste em jogos

O objetivo dos testes em jogos é avaliar a estabilidade, jogabilidade e experiência do jogador, garantindo que todas as mecânicas funcionem corretamente e que a interação com o jogo seja fluida e satisfatória.

3.6.2 Tipos de testes

Serão realizados diferentes tipos de testes para abranger os principais aspectos do jogo:

- Testes de Jogabilidade: Avaliam se as mecânicas são intuitivas, equilibradas e proporcionam uma experiência divertida.

- Testes de Funcionalidade: Verificam se todas as interações e mecânicas do jogo operam conforme o esperado.

- Testes de Performance: Analisam o consumo de recursos, tempo de carregamento e taxa de FPS para garantir uma experiência otimizada.

- Testes de Compatibilidade: Avaliam o funcionamento do jogo em diferentes dispositivos, sistemas operacionais e resoluções de tela.

3.6.3 Metodologia dos testes

Os testes são conduzidos em diferentes fases do desenvolvimento:

- Testes Internos: A equipe de desenvolvimento jogará repetidamente para detectar falhas técnicas e problemas de design.

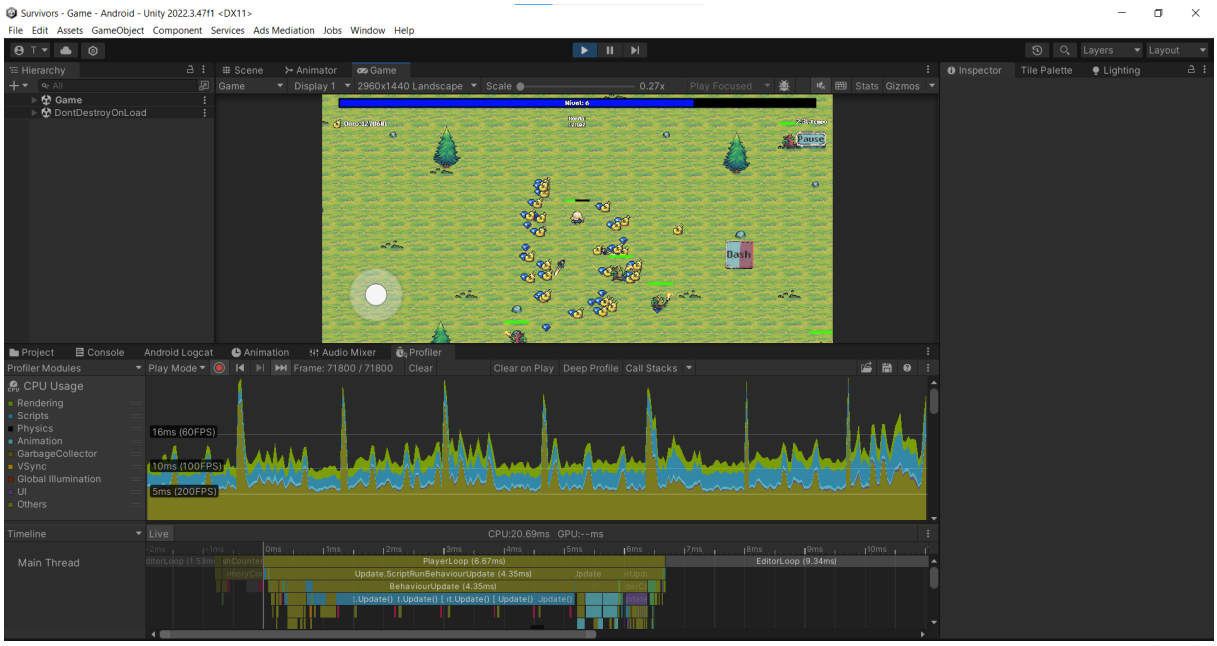
- Testes com Usuários: Jogadores externos testarão o jogo e fornecerão feedback por meio de formulários de avaliação.

- Testes com Cenários Extremos: Serão realizados testes tentando quebrar o jogo, explorando possíveis falhas.

3.6.4 Ferramentas e Recursos Utilizados

Os testes internos podem ser realizados utilizando o *Unity Profiler*, uma ferramenta da *Unity* 3D que permite monitorar o desempenho e o consumo de recursos do jogo, facilitando a identificação de problemas de performance e a avaliação de áreas que podem ser otimizadas para melhorar o desempenho geral.

Figura 20 – Janela do *Unity Profiler*



Fonte: elaborado pelo autor, 2025

Para os testes com usuários, pode-se utilizar fichas de feedback para receber opiniões e sugestões de melhorias. Nos testes de cenários extremos, o jogo será submetido a situações de estresse máximo. Em jogos *online*, por exemplo, os servidores serão testados para verificar sua capacidade de suportar um grande número de jogadores simultaneamente.

3.6.5 Critérios de Sucesso

O jogo será considerado pronto para lançamento se atender aos critérios estabelecidos pela equipe ou pela empresa. Possíveis critérios incluem o funcionamento correto das mecânicas principais, a ausência de erros críticos e uma performance estável, sem quedas significativas de FPS, entre outros aspectos essenciais para uma experiência satisfatória.(Naspolini, 2019; Naspolini, 2021a; Naspolini, 2021b)

3.7 Licença de uso

Este projeto é distribuído sob a Licença MIT (*Massachusetts Institute of Technology*), uma licença de software livre que permite ampla liberdade de uso, modificação e distribuição do código-fonte. A escolha dessa licença visa garantir que outros desenvolvedores possam utilizar e aprimorar o projeto, promovendo a colaboração e a evolução da comunidade de desenvolvimento de jogos.

4 Resultados e discussões

4.1 Gerenciamento de configuração e mudanças

Conforme mencionado no capítulo de Materiais e Métodos, utilizei o GitLab do IFRO como repositório e ferramenta de versionamento do projeto.

Para organizar e dividir as implementações, foram utilizadas *branches* específicas para diferentes funcionalidades do jogo. como por exemplo:

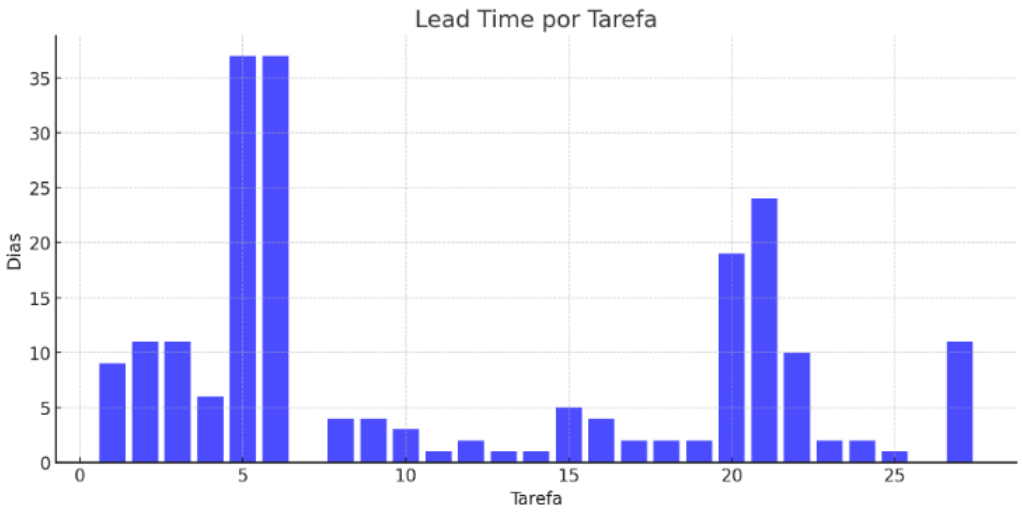
- Movimentação do personagem – Responsável pela implementação do sistema de controle do jogador.
- Mecânicas roguelite – Para desenvolver e testar as características do gênero, como progressão e upgrades das armas.
- Configuração para publicação – Ajustes para a disponibilização do jogo.

Repositório: https://gitlab.fslab.dev/Theodoro/survivors

4.2 Processo de desenvolvimento

Conforme mencionado no capítulo de Materiais e Métodos, utilizei o Kanban como metodologia ágil.

Figura 21 – LeadTime

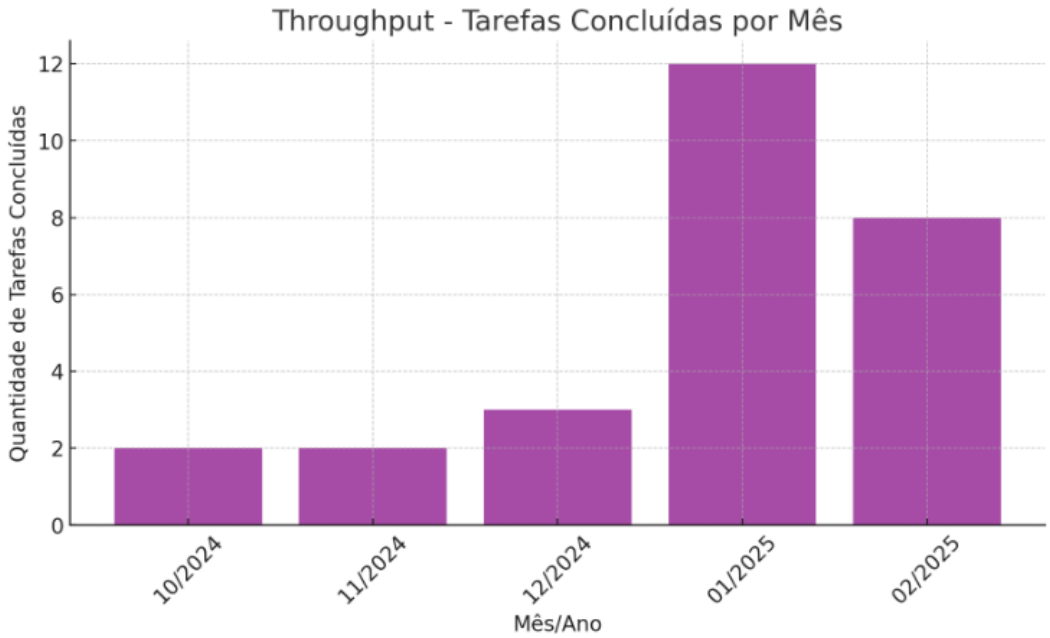


Fonte: Elaborado pelo autor, (2025)

O gráfico acima representa o *Lead Time* das tarefas realizadas, ou seja, o tempo total entre a criação e a conclusão de cada atividade. Ao todo, foram registradas 27 tarefas, que foram sendo criadas conforme as necessidades do projeto.

Podemos observar que há uma variação significativa nos tempos de conclusão das tarefas, o que indica que algumas atividades foram resolvidas rapidamente, enquanto outras demandaram um período maior. Essa variação está relacionada à complexidade das tarefas e ao tempo necessário para estudar e encontrar uma solução para cada uma delas.

Figura 22 – Throughput



Fonte: Elaborado pelo autor, (2025)

O gráfico acima mostra a quantidade de tarefas concluídas por mês. Observamos que, nos dois primeiros meses, o ritmo foi mais lento, pois o foco estava no desenvolvimento de um código mais genérico e abrangente, projetado para ser facilmente componentizado e exportado para reutilização em outros projetos. No entanto, ao perceber que essa abordagem seria inviável devido às restrições de tempo, optou-se por uma implementação mais específica e acoplada, resultando em um código desenvolvido para atender às particularidades deste projeto.

4.3 Relatório dos testes

4.3.1 Testes Internos

Durante todo o desenvolvimento do jogo, eram sempre realizados testes internos, como se confirma verificando o tempo em que algumas tarefas ficam na coluna de revisão,

e também pode ser observado nos nomes das tarefas registradas, como aquelas com o termo "Refazendo"ou que indicam toda a funcionalidade melhorada.

Por exemplo, na tarefa 19. Bugs, Balanceamento, foram feitas correções de bugs e um primeiro ajuste de balanceamento do jogo.

Na tarefa 24. Object Pool Pattern, foi identificada uma queda de desempenho após certo tempo de jogo. Através da ferramenta Unity Profiler, confirmou-se um alto consumo de processamento. Após análise, o problema foi identificado: os inimigos, ao serem derrotados, apenas eram desativados, mas seus *GameObjects* continuavam ocupando espaço na memória, acumulando-se com o tempo.

Uma solução simples seria destruir os *GameObjects* dos inimigos, mas optou-se pela implementação do Object Pool Pattern, melhorando o desempenho do jogo ao reutilizar os objetos ao invés de destruí-los e recriá-los constantemente.

Foram coletados *feedbacks* dos usuários,que ajudaram a identificar problemas e sugerir melhorias no jogo.

Por exemplo, na tarefa 25. Tempo de Invencibilidade, alguns jogadores relataram que, ao reviver, morriam quase imediatamente por estarem cercados por vários inimigos.Para solucionar esse problema, foi implementado um tempo de invencibilidade ao renascer, garantindo uma experiência mais justa.

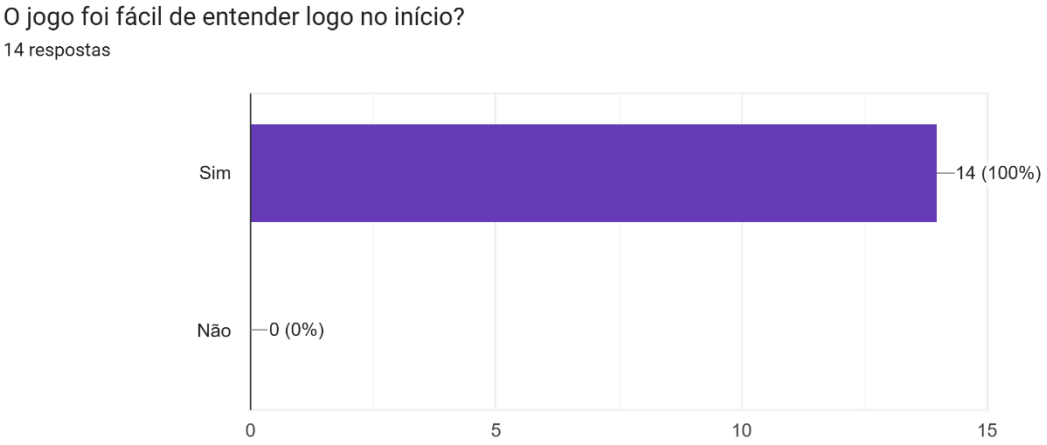
Outras sugestões incluíram ajustes na configuração padrão do joystick e melhorias no balanceamento do jogo, proporcionando uma experiência mais fluida e equilibrada.

Para a realização dos testes com usuários, também foi disponibilizada uma versão de teste do jogo, acompanhada de um formulário no *Google Forms*.

No total, 14 participantes contribuíram com a pesquisa, fornecendo *insights* valiosos sobre a experiência de jogo.

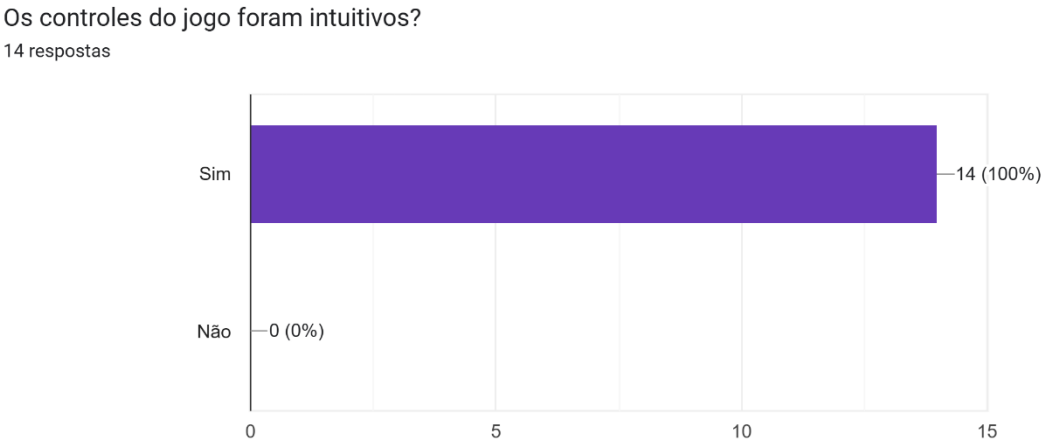
A seguir, alguns gráficos obtidos:

Figura 23 – O jogo foi fácil de entender logo no início?



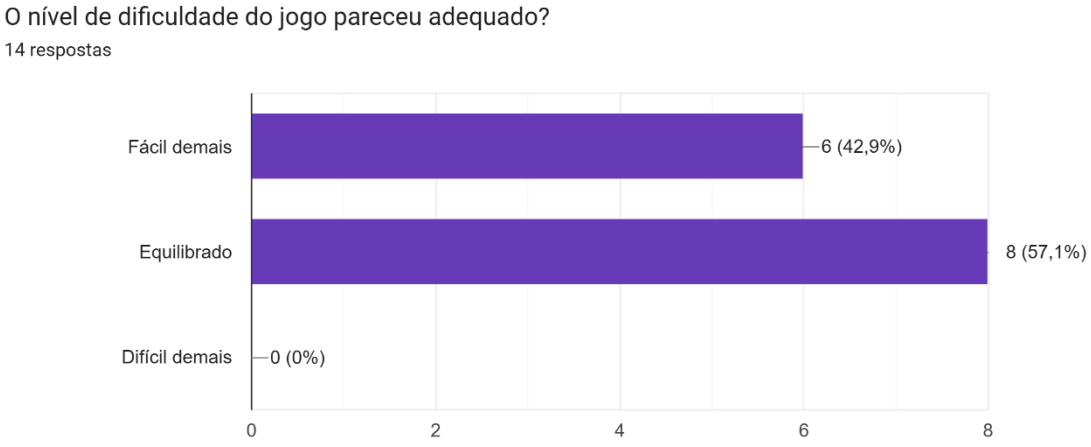
Fonte: Elaborado pelo autor, Google Forms, (2025)

Figura 24 – Os controles do jogo foram intuitivos?



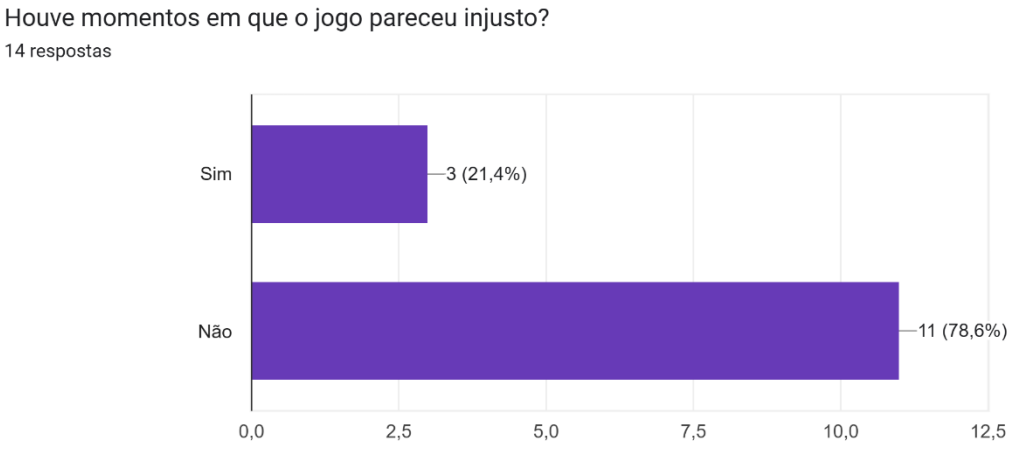
Fonte: Elaborado pelo autor, Google Forms, (2025)

Figura 25 – O nível de dificuldade do jogo pareceu adequado?



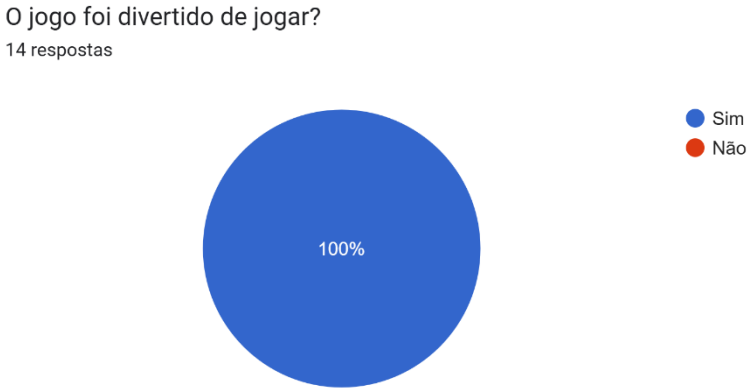
Fonte: Elaborado pelo autor, Google Forms, (2025)

Figura 26 – Houve momentos em que o jogo pareceu injusto?



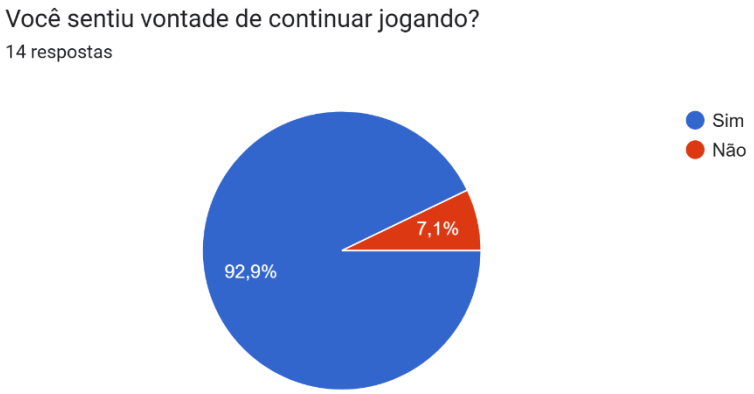
Fonte: Elaborado pelo autor, Google Forms, (2025)

Figura 27 – O jogo foi divertido de jogar?



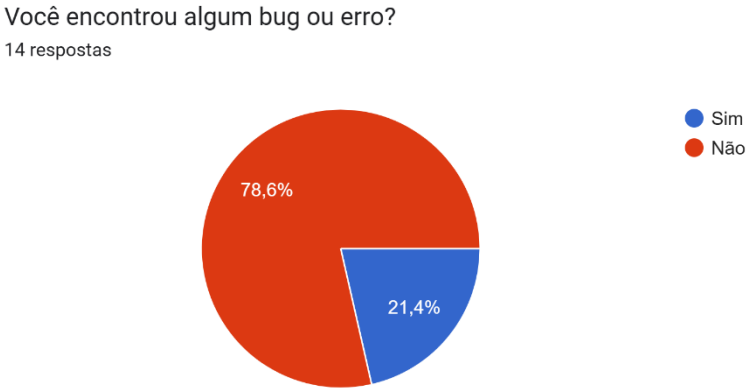
Fonte: Elaborado pelo autor, Google Forms, (2025)

Figura 28 – Você sentiu vontade de continuar jogando?



Fonte: Elaborado pelo autor, Google Forms, (2025)

Figura 29 – Você encontrou algum bug ou erro?



Fonte: Elaborado pelo autor, Google Forms, (2025)

Figura 30 – O jogo teve quedas de desempenho?

Fonte: Elaborado pelo autor, Google Forms, (2025)

As demais questões do formulário eram descritivas, permitindo que os usuários expressassem suas opiniões sobre o que estava bom, problemas encontrados, o que poderia ser melhorado, e quais aspectos que estavam insatisfatórios. Além disso, diversas sugestões interessantes para novos recursos foram apresentadas.

Com base no feedback recebido, foi realizada uma análise de viabilidade para avaliar a possibilidade de implementação das sugestões e possíveis alterações no jogo.

4.4 Documentação

4.4.1 Documentação para desenvolvedores

A explicação do comportamento do jogo pode ser bastante complexa, pois envolve muita lógica e a interação entre diversos componentes.

Por isso, apresentarei uma visão geral simplificada em uma linguagem mais acessível, destacando os principais elementos e componentes que considerei mais importantes para o entendimento geral do jogo.

Antes de prosseguir, é importante entender alguns conceitos:

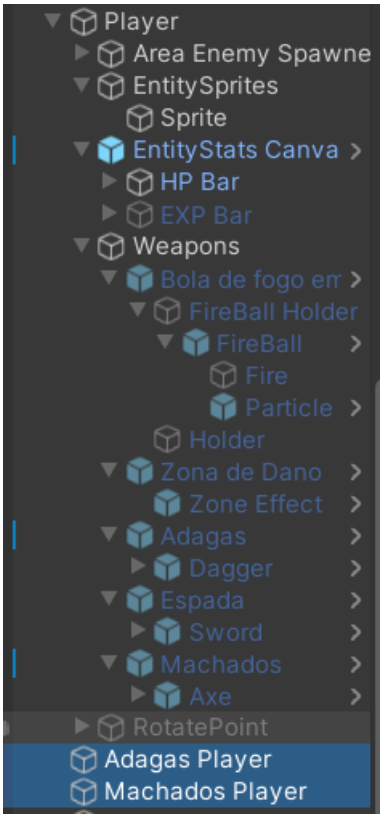
- *Prefab*(Pré-Fabricado) é um modelo de objeto que pode ser reutilizado no jogo. Ele funciona como a base para a criação de objetos dentro da cena, permitindo que você crie múltiplas instâncias desse objeto de maneira eficiente.

- Canvas é um *GameObject* específico para elementos de UI (Interface de Usuário). Ele é responsável por renderizar e organizar os componentes da interface, como botões, controles deslizantes e textos, de forma adequada na tela.

4.4.1.1 Player

Na janela de Hierarquia, o *GameObject Player* contém vários outros *GameObjects* dentro dele, chamados de filhos, pois estão organizados hierarquicamente dentro do *GameObject* do *Player*, permitindo um melhor gerenciamento e estruturação dos elementos relacionados ao personagem.

Figura 31 – Hierarquia do *Player*



Fonte: elaborado pelo autor, (2025)

4.4.1.1.1 EntityStats Canvas

O *EntityStats Canvas* é um *Prefab* do tipo *Canvas*, responsável por exibir um *slider*(controle deslizante) utilizado como indicador da quantidade de vida do *Player*.

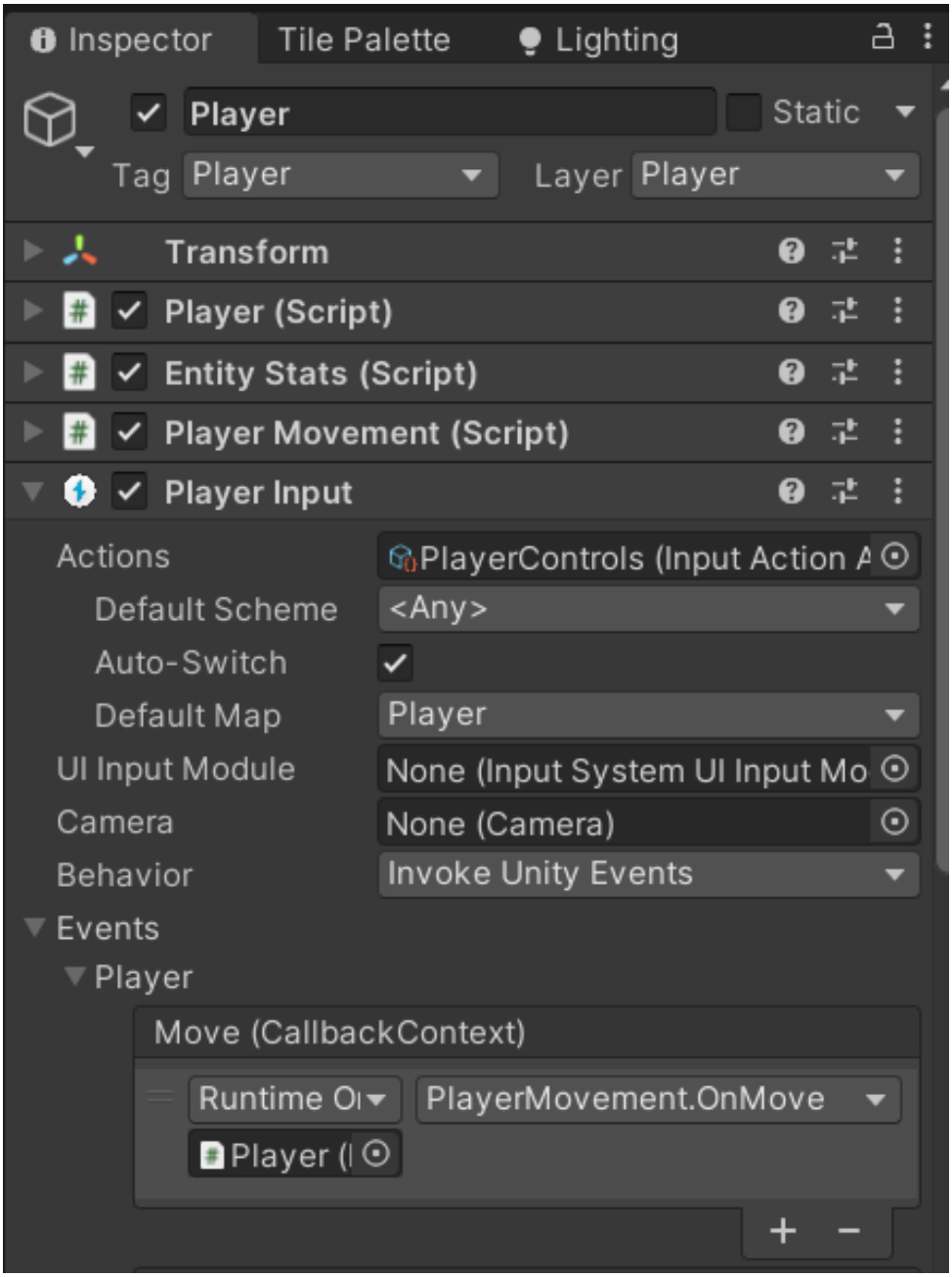
4.4.1.1.2 Weapons

O *GameObject Weapons* é um objeto vazio utilizado exclusivamente para organizar os *GameObjects* das armas dentro da hierarquia. Inicialmente, as armas estão desativadas e só serão ativadas quando forem desbloqueadas através do painel de melhoria.

É importante observar que os *GameObjects* Adagas *Player* e Machados *Player* estão fora da hierarquia do *Player*. Isso ocorre porque esses objetos precisam manter independência de posição no mundo do jogo. Se estivessem como filhos diretos do *Player*, eles se moveriam automaticamente junto com ele, o que não é o comportamento desejado.

4.4.1.1.3 Player Input

Figura 32 – Janela *Inspetor* do *Player*

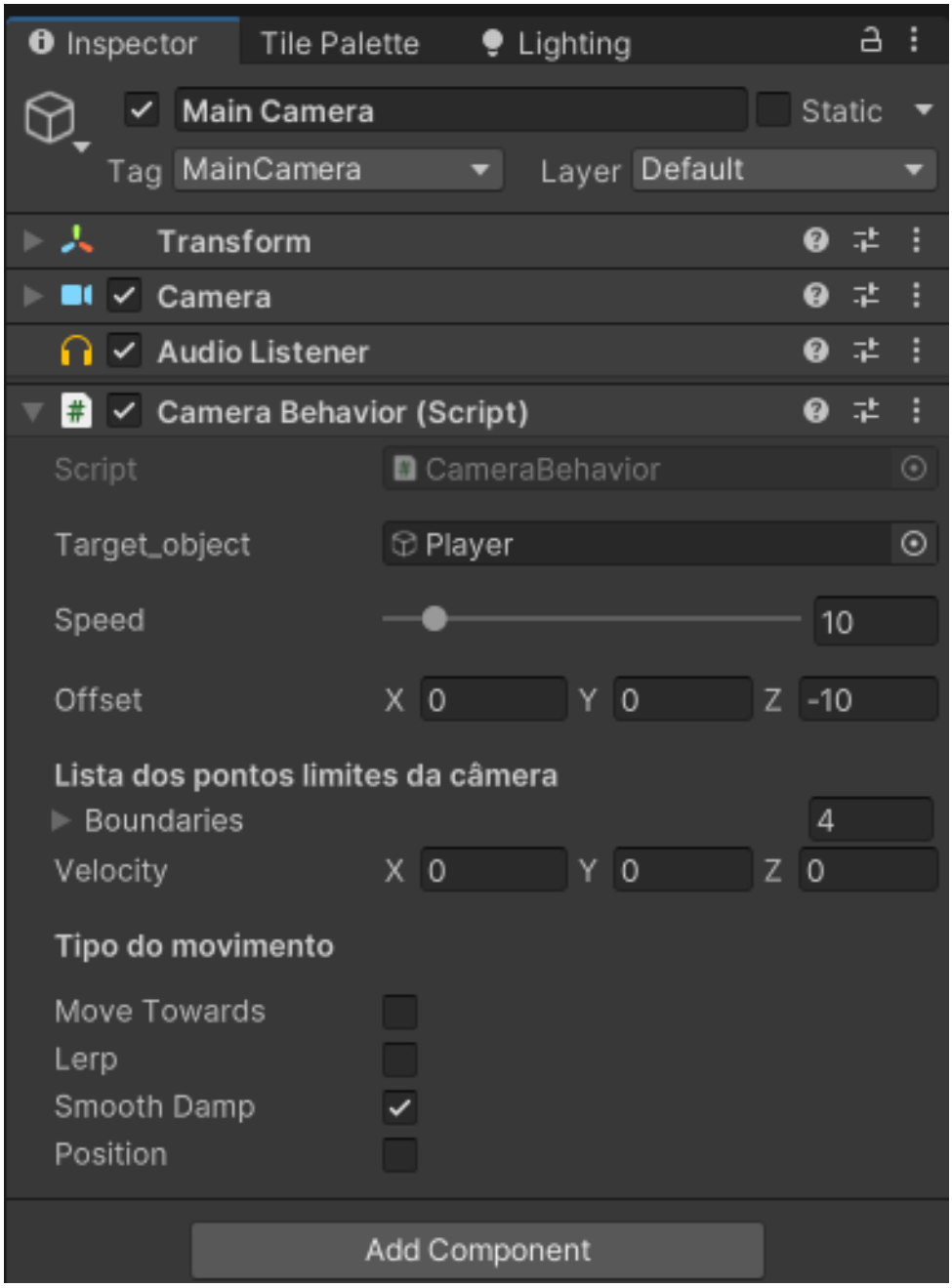


Fonte: elaborado pelo autor, (2025)

Na janela Inspetor do *GameObject Player*, encontramos o componente *Player Input*, parte do novo sistema de entradas da *Unity* 3D. Esse componente é responsável por interpretar comandos vindos de diferentes dispositivos, como teclado, mouse, controle, tela sensível ao toque ou sensores. Quando um comando é detectado, o *Player Input* dispara um evento configurado, e transmite os valores necessários para que o código execute a ação correspondente.

4.4.1.2 Camera

Figura 33 – Janela *Inspetor* do *Main Camera*



Fonte: elaborado pelo autor, (2025)

Na janela Inspetor do *GameObject Main Camera* encontra-se o *script Camera Behavior*.Ele faz a câmera seguir um objeto-alvo (por exemplo, o personagem do jogador ou outro objeto relevante no jogo).

Parâmetros Controlados:

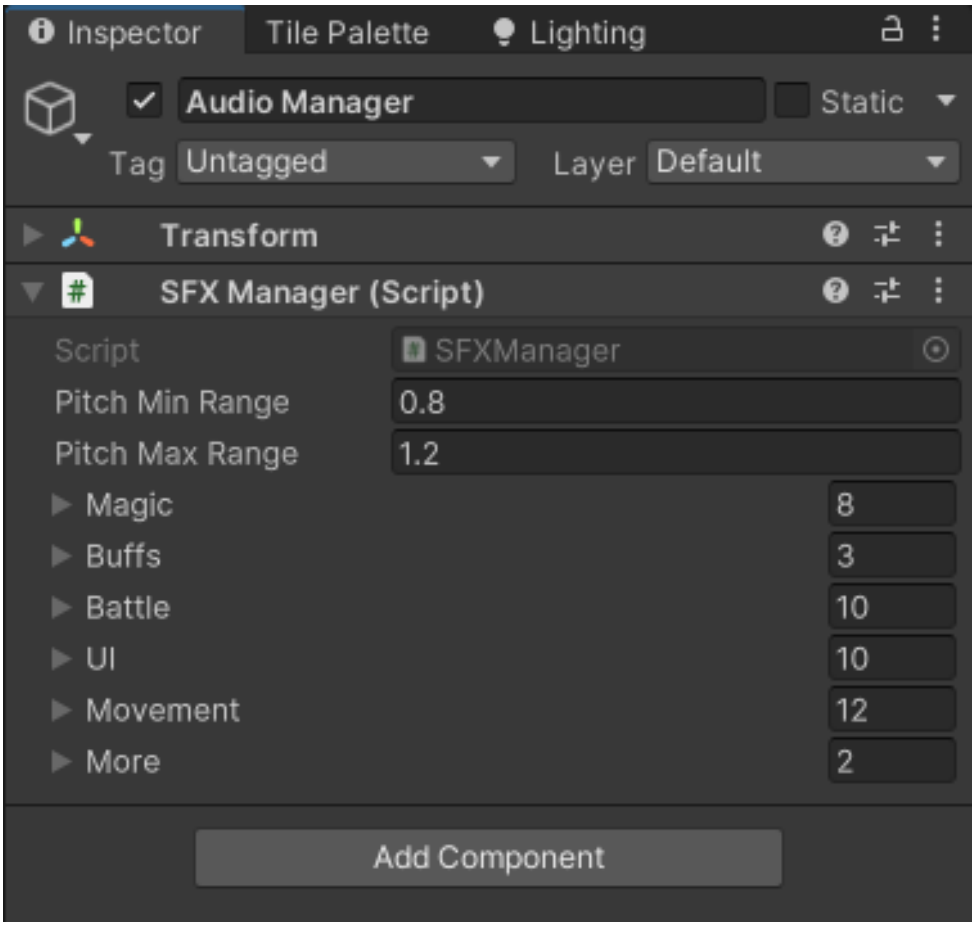
- Velocidade: A velocidade com que a câmera segue o alvo.

- Desvio: Um espaço adicional entre a câmera e o alvo.
- Tipo de Movimento: Pode definir o comportamento da câmera,por exemplo, suavização, movimentação rígida, ou algum outro tipo).

O *script* também define os limites para onde a câmera pode se mover. Isso é feito através de objetos de referência para o limite da câmera, garantindo que a câmera não ultrapasse as bordas do cenário.

4.4.1.3 Audio Manager

Figura 34 – Janela *Inspetor* do Audio *Manager*



Fonte: elaborado pelo autor, (2025)

Na janela Inspetor do *GameObject Audio Manager* encontra-se o *script* SFX *Manager* (*Sound effects Manger*) em português o Gerente de efeitos sonoros, ele é um *Singleton* tornando-o acessível de qualquer outro *script* que precise interagir com ele.

Esse *script* disponibiliza uma lista de áudios para que outros componentes do jogo possam selecionar o som que desejam emitir. Além disso, ele oferece duas funções para controle da reprodução dos áudios:

- Emissão de áudio padrão: Essa função reproduz o áudio selecionado.
- Emissão de áudio com variação de pitch:Essa função permite alterar o *pitch* (tom) do áudio de forma aleatória, entre dois valores definidos, o que adiciona uma variação interessante nos sons criando uma sensação de maior diversidade sonora.

Figura 35 – Código SFX*Manager*

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class SFXManager : MonoBehaviour
{
    public static SFXManager Instance;

    public float pitchMinRange = .8f;
    public float pitchMaxRange = 1.2f;


    public AudioSource[] magic;
    public AudioSource[] buffs;

    public AudioSource[] battle;

    public AudioSource[] UI;

    public AudioSource[] Movement;

    public AudioSource[] More;

    private void Awake()
    {
        Instance = this;

        // if (Instance == null)
        // {
        //     Instance = this;
        //     DontDestroyOnLoad(gameObject);
        // }
        // else
        // {
        //     Destroy(gameObject);
        // }
    }


    public void PlaySFX(AudioSource[] soundEffects, int sfxToPlay)
    {
        //soundEffects[sfxToPlay].Stop();
        soundEffects[sfxToPlay].Play();
    }

    public void PlaySFXPitched(AudioSource[] soundEffects, int sfxToPlay)
    {
        soundEffects[sfxToPlay].pitch = Random.Range(pitchMinRange, pitchMaxRange);

        PlaySFX(soundEffects, sfxToPlay);
    }

}

```

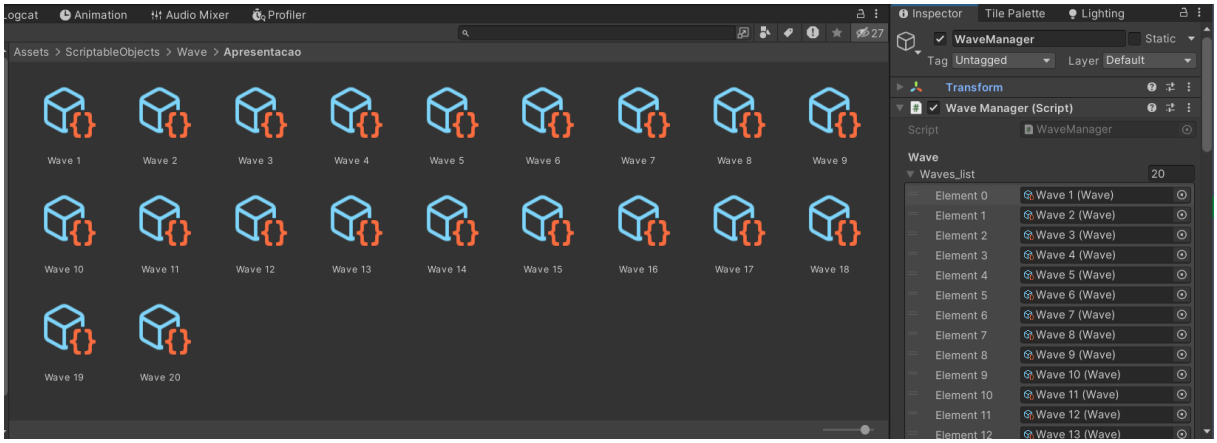
Fonte: elaborado pelo autor, (2025)

Um exemplo pode ser visto no *GoldController*, já mencionado neste trabalho na Figura 19, onde o efeito sonoro da coleta de ouro é emitido com uma variação no tom (*pitch*) a cada coleta, dando mais dinamismo e naturalidade ao som.

4.4.1.4 WaveManager

Na janela Inspetor do *GameObject WaveManager* encontra-se o *script WaveManager* em português, o Gerente de Ondas, que é responsável por gerar as hordas de inimigos durante o jogo.

Figura 36 – Janela *Inspetor* do *WaveManager*



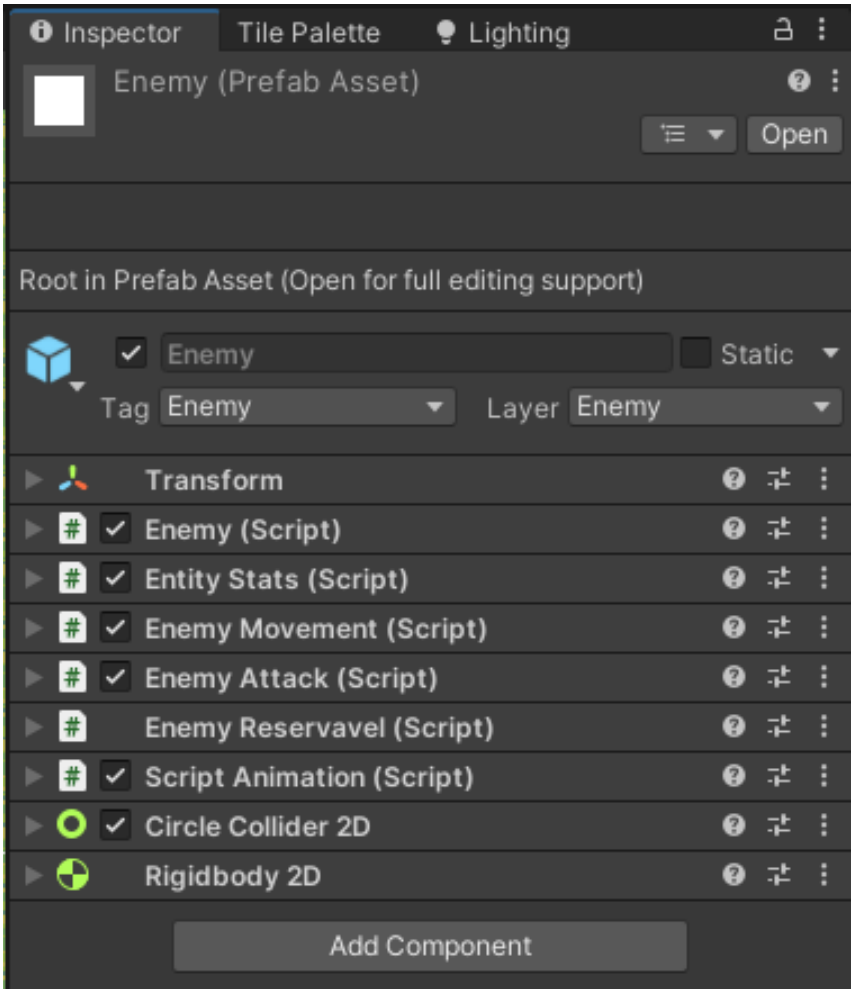
Fonte: elaborado pelo autor, (2025)

Dentro desse script, existe uma lista de ondas. Cada onda é representada por um *ScriptableObject* (figura 5) que contém as informações específicas sobre a horda de inimigos a ser gerada. Essas informações incluem, por exemplo, a quantidade de inimigos que serão gerados, além de uma lista de *ScriptableObjects* de inimigos, que define quais tipos de inimigos serão gerados aleatoriamente durante aquela onda.

4.4.1.5 Inimigo

O *script Enemy* é responsável por gerenciar o comportamento do inimigo, e ele recebe um *ScriptableObject* que contém informações configuráveis e determina como o inimigo será configurado.

Figura 37 – Janela *Inspetor* do *Prefab* do inimigo



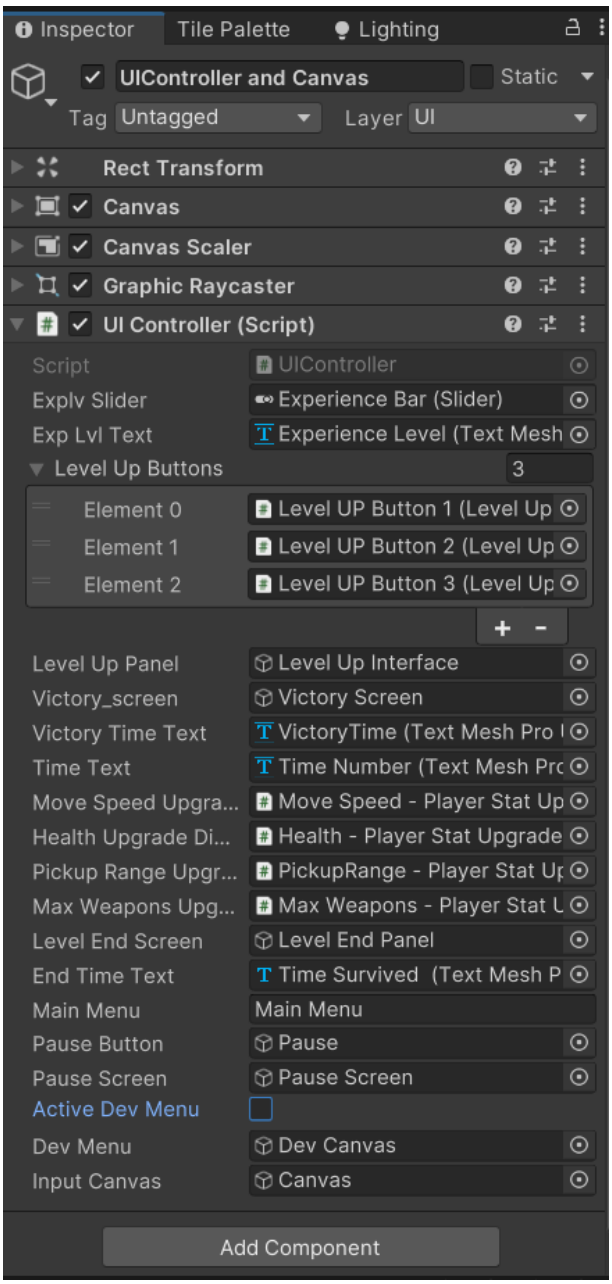
Fonte: elaborado pelo autor, (2025)

Com base nas informações fornecidas, o *script Enemy* ajusta e disponibiliza vários parâmetros que modificam outros *scripts* associados ao *GameObject* do inimigo, por exemplo a quantidade de vida, a velocidade de movimento, o dano de ataque, além de aspectos visuais, como a imagem que será utilizada e a animação do inimigo.

Esse sistema oferece grande flexibilidade e modularidade, permitindo que diferentes inimigos sejam configurados facilmente através do *ScriptableObject*, sem a necessidade de criar um novo *Prefab* para cada novo tipo de inimigo.

4.4.1.6 *UIController*

Figura 38 – Janela *Inspetor* do *UIController and Canvas*

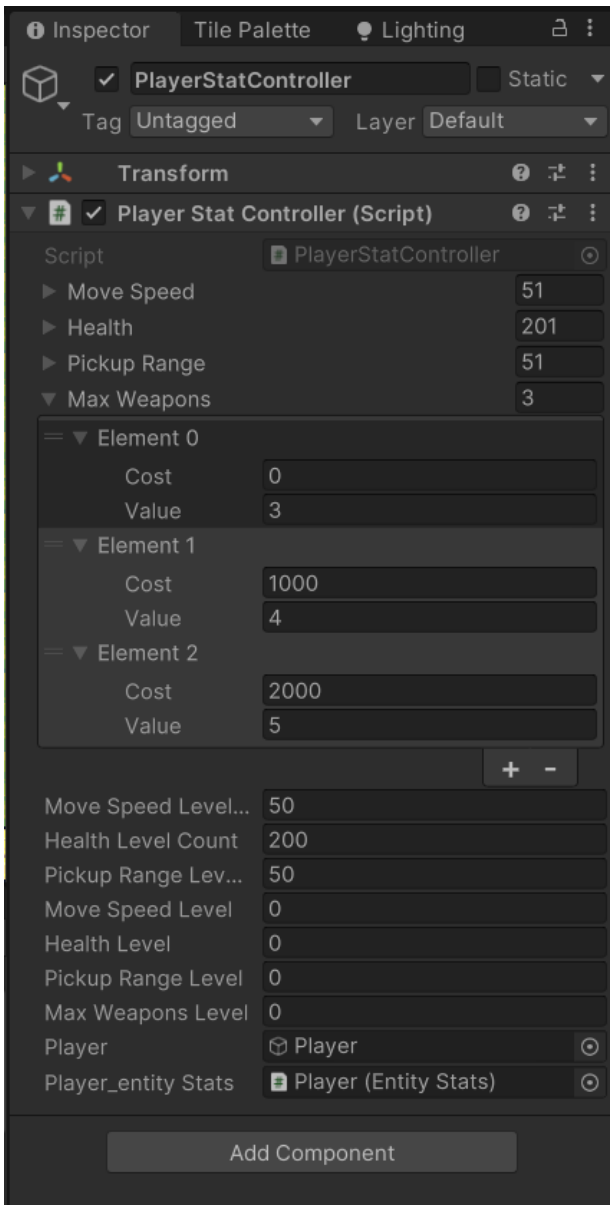


Fonte: elaborado pelo autor, (2025)

Na janela Inspetor do *UIController and Canvas* encontra-se *script Singleton UIController*(Controlador de Interface do Usuário). Como o nome sugere, esse *script* é responsável por gerenciar a interface do usuário, incluindo a tela de pausa, a tela de vitória e o painel de melhoria.

4.4.1.7 *PlayerStatController*

Figura 39 – Janela *Inspetor* do *PlayerStatController*



Fonte: elaborado pelo autor, (2025)

Na janela Inspetor do *PlayerStatController* encontra-se o *script Singleton Player Stat Controller*, esse *script* armazena as listas com os valores das melhorias e contém a lógica para as melhorias dos atributos do *Player*, além de gerenciar a exibição na tela de melhorias.

4.5 Implantação

Figura 40 – QR Code para Pequenos sobrevivientes na Google Play Store

Fonte: elaborado pelo autor, (2025)

Pequenos Sobreviventes: <https://play.google.com/store/apps/details?id=com.SatyroStudios.TinySurvivors>

Os recursos utilizados no jogo (arte e sons) podem ser encontrados em sites como:

- *itch.io*, uma plataforma online de distribuição de jogos independentes e outros tipos de conteúdo criativo, como arte, música e recursos para desenvolvedores de jogos.
- *Unity Asset Store*, a plataforma oficial da Unity. A loja oferece uma ampla gama de *assets*(recursos), ferramentas e extensões, que ajudam a acelerar o processo de desenvolvimento de jogos e aplicações, além de fornecer soluções prontas para uso.
- *OpenGameArt* é uma plataforma que disponibiliza recursos gráficos e sonoros gratuitos para jogos, permitindo que artistas compartilhem suas criações e fornecendo materiais acessíveis para desenvolvedores de jogos.
- *Mixkit* um site que oferece recursos como vídeos, músicas e efeitos sonoros gratuitos para uso em projetos.

Cada uma dessas plataformas tem um foco específico, oferecendo conteúdos gratuitos ou pagos para facilitar a criação de conteúdo digital, especialmente no desenvolvimento de jogos.

- Música de fundo: Driving Ambition, por Ahjay Stelino, baixado no site<https://mixkit.co/free-stock-music/>
- Efeitos sonoros:
  - Os efeitos sonoros estão diponiveis na AssetStore da Unity,<https://assetstore.unity.com/packages/audio/sound-fx/rpg-essentials-sound-effects-free-227708>,contendo várias amostras gratuitas.
  - O efeito sonoro da coleta do ouro está disponível no site <https://mixkit.co/free-sound-effects/coin/> com o nome Money bag drop.
  - O efeito sonoro da coleta de experiência está disponível no site OpengGameArt, <https://opengameart.org/content/item-collected>.
- Recursos gráficos:
  - O pacote de recursos gráficos Tiny Swords, <https://pixelfrog-assets.itch.io/tiny-swords>.
  - A imagem da experiência, <https://knobiek.itch.io/items-16x16-free>.

4.6 Demonstração do software

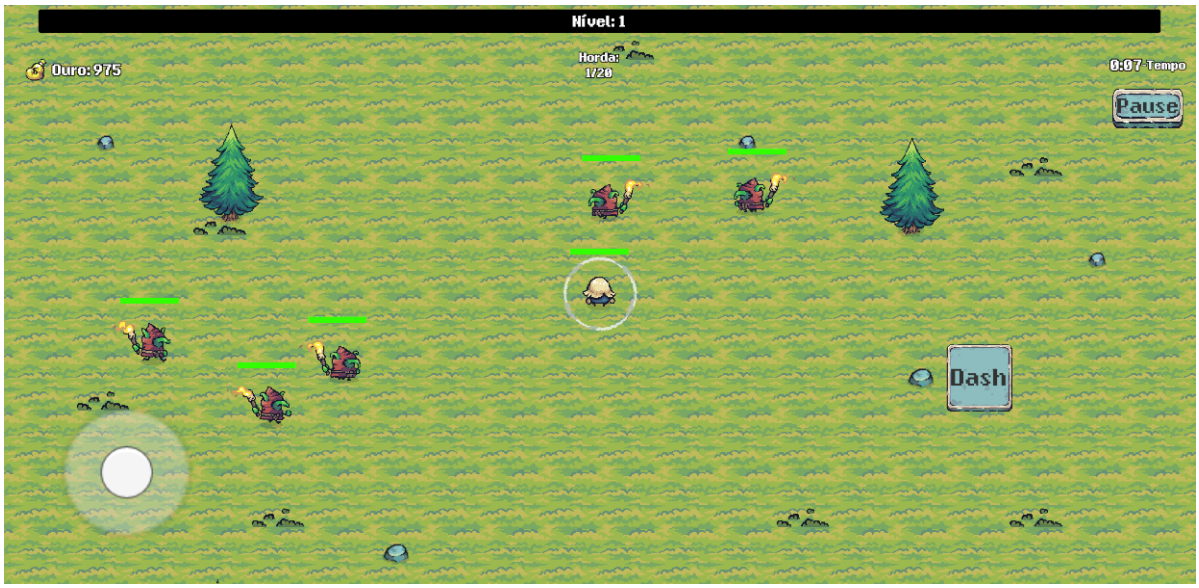
Esta é a tela inicial do jogo, onde se pode visualizar o título, a quantidade de ouro disponível e as opções para iniciar ou sair.

Figura 41 – Tela inicial

Fonte: elaborado pelo autor, (2025)

As imagens foram extraídas de uma versão de apresentação do jogo. O número de ondas de inimigos e o tempo necessário para alcançar a vitória não refletem os valores reais da versão final do jogo.

Figura 42 – *Gameplay* 1



Fonte: elaborado pelo autor, (2025)

O jogo inicia e as ondas de inimigos começam. O contato com eles causa dano, e se a vida do personagem chegar a zero, ele morre. O jogador inicia com uma arma aleatória, que causa dano aos inimigos ao entrar em contato. Quando a vida de um inimigo chega a zero, ele é derrotado, concedendo experiência e uma chance de soltar ouro. Ao coletar experiência e ouro, os contadores são atualizados, barra de experiência aumenta e o número de ouro também. Quando a barra de experiência é preenchida, um painel de melhorias é exibido.

Figura 43 – *Gameplay* 2

Fonte: elaborado pelo autor, (2025)

Neste painel, o jogador pode escolher entre três opções de melhorias ou desbloquear novas armas. Além disso, o ouro acumulado pode ser usado para comprar melhorias nas habilidades do personagem durante a partida. Ao evoluir todas as armas, as opções de melhoria podem aparecer vazias. Se isso acontecer, utilize o botão 'Pular Nível' para fechar o painel e continuar o jogo.

Figura 44 – *Gameplay* 3

Fonte: elaborado pelo autor, (2025)

As habilidades e poderes do personagem são aprimorados, permitindo que ele

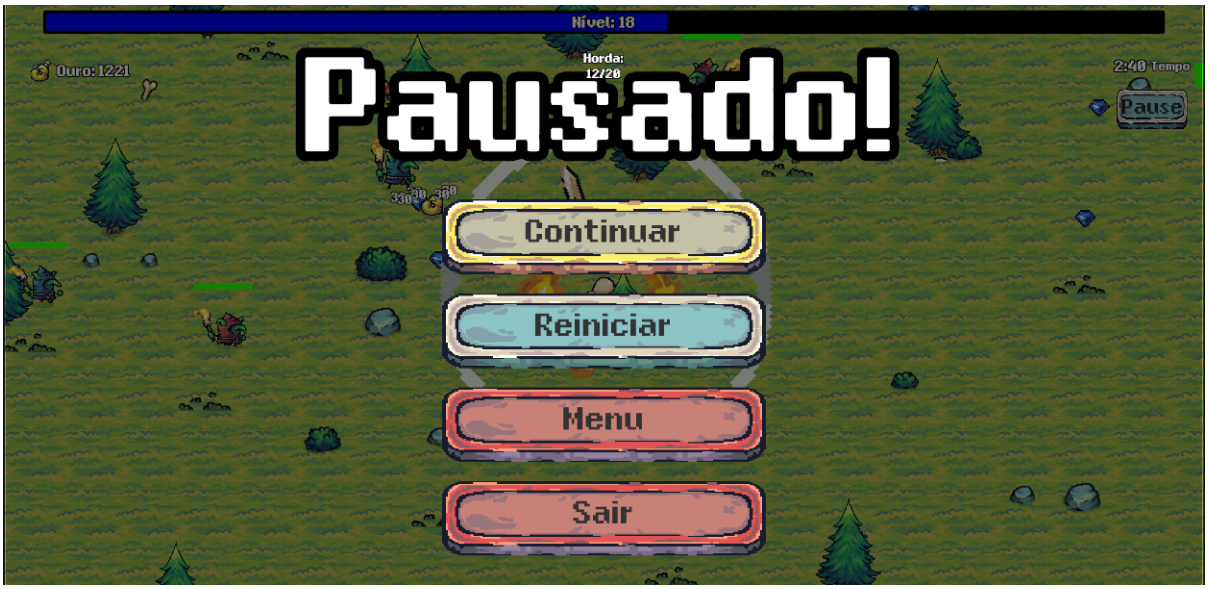
elimine os inimigos com mais facilidade. No entanto, o nível dos inimigos também aumenta, adicionando maior dificuldade ao jogo.As armas dos inimigos não evoluem; todas permanecem no nível 1.

Figura 45 – *Gameplay* 4

Fonte: elaborado pelo autor, (2025)

Com o aumento da dificuldade, os inimigos concedem mais experiência, mas a quantidade de experiência necessária para subir de nível também aumenta.

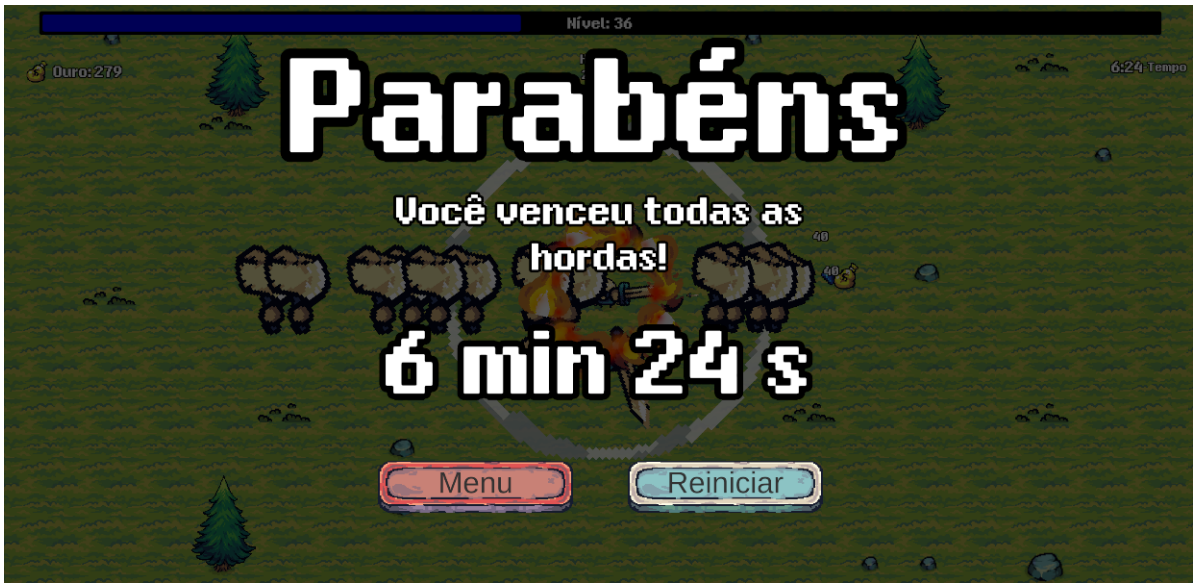
Figura 46 – Tela de Pause



Fonte: elaborado pelo autor, (2025)

O jogo inclui uma opção de pausa, que exibe um menu com as seguintes opções: continuar a partida, iniciar uma nova partida, retornar ao menu principal ou sair e fechar o jogo.

Figura 47 – *Gameplay* 5



Fonte: elaborado pelo autor, (2025)

O loop de gameplay consiste em eliminar inimigos, coletar experiência e ouro, adquirir melhorias e sobreviver às ondas de inimigos até o fim.

5 Considerações finais

Desenvolver um jogo já é um grande desafio por si só, sabia que seria uma tarefa complexa. No entanto, não esperava que fosse tão trabalhoso e exigente quanto se mostrou. Há inúmeros detalhes que influenciam diretamente a experiência do jogador, e cada um deles demanda tempo, estudo e dedicação.

Apesar das dificuldades, obtive sucesso no desenvolvimento da base do jogo. O projeto resultou em um jogo 2D utilizando a estética *pixel art*, adotando o estilo *roguelite*, mas com elementos do subgênero *survivors*. Busquei modularizar ao máximo a estrutura do jogo, tornando-o mais flexível e facilitando futuras expansões e otimizações.

Em relação às limitações do projeto, alguns aspectos poderiam ter sido melhor trabalhados, como a diversidade de mecânicas e o balanceamento de dificuldades. Além disso, como foi um projeto desenvolvido dentro de um tempo limitado, alguns detalhes foram simplificados para garantir a entrega do produto dentro do prazo estipulado.

No aspecto gerencial, o projeto reforçou a importância do planejamento e da organização no desenvolvimento de jogos. Lidar com prazos, prioridades e imprevistos fez parte do processo, e foi essencial aprender a gerenciar esses desafios de forma eficiente.

Os objetivos específicos estabelecidos inicialmente foram atendidos em grande parte. O foco na criação de um jogo funcional, utilizando princípios fundamentais de desenvolvimento e boas práticas, foi alcançado. Além disso, consegui aprofundar meu conhecimento em *Unity* 3D, programação e *design* de jogos, o que agregou muito ao meu aprendizado.

Por fim, esta jornada foi extremamente enriquecedora. Além dos conhecimentos técnicos adquiridos, a experiência de desenvolver um jogo utilizando a *Unity*3D e a escrita desta monografia foram fundamentais para o meu crescimento acadêmico e profissional.

5.1 Trabalhos futuros

Após o desenvolvimento da solução proposta, foram identificados os seguintes trabalhos futuros que podem ser realizados:

- Implementação de um sistema de progressão mais eficiente.
- Maior variedade de inimigos, com diferentes atributos e comportamentos.
- Expansão do arsenal de armas e habilidades disponíveis para o personagem.
- Inclusão de novos personagens jogáveis, cada um com atributos distintos.

- Desenvolvimento de um sistema de recompensas, como bônus diários.
- Integração com um sistema de conquistas, vinculado ao Google Play Games.
- Criação de novas fases e cenários.
- Implementação de uma fase de desafio infinito, acompanhada de um ranking público para registrar as melhores pontuações.

Esses aprimoramentos não apenas agregariam valor à experiência do jogador, como também tornariam o jogo mais dinâmico e atrativo para um público maior.

Referências

Alura. *Game Design: o que é, qual a sua importância e como se tornar um game designer*. 2023. Alura, artigo institucional. Disponível em: <https://www.alura.com.br/artigos/game-design>. Acesso em: 17 abril 2025. Citado na página 18.

Alura. *Método Kanban: O que é e como aplicar na sua rotina*. 2023. Disponível em: <https://www.alura.com.br/artigos/metodo-kanban>. Acesso em: 07 fevereiro 2025. Citado na página 28.

Alves; Bianchin. *O jogo como recurso de aprendizagem*. 2010. Disponível em: <https://www.revistapsicopedagogia.com.br/detalhes/210/o-jogo-como->. Acesso em: 13 fevereiro 2025. Citado na página 15.

Alves, G. B. *Aplicação de boas práticas na arquitetura de sistemas de jogos: estudo de caso com o padrão Entity-Component-System (ECS)*. 2021. Trabalho de Conclusão de Curso — Universidade Federal do Ceará. Disponível em: <http://repositorio.ufc.br/handle/riufc/58827>. Acesso em: 30 abril 2025. Citado na página 18.

Andrade; Dantas. *Da Teoria à Prática em Desenvolvimento de Jogos Digitais: Um Estudo Sobre os Modelos de Processo Utilizados no Mercado Paraibano*. 2018. Disponível em: <https://www.researchgate.net/profile/Jose-Raul-Andrade/publication/324532563_Da_Teoria_a_Pratica_em_Desenvolvimento_de_Jogos_Digitais_Um_Estudo_Sobre_os_Modelos_de_Processo_Utilizados_no_Mercado_Paraibano/links/5ad3b1c0a6fdcc29357ff370/Da-Teoria-a-Pratica-em-Desenvolvimento-de-Jogos-Digitais-Um-Estudo-Sobre-os-Modelos-de-Process pdf>. Acesso em: 15 dezembro 2024. Citado 2 vezes nas páginas 16 e 26.

Baramallo. *Qual é a Diferença entre Roguelike e Roguelite?* 2023. YouTube, Fazendo Nerdice, 19 dezembro 2023. 43min03s. Disponível em: <https://www.youtube.com/watch?v=fF6rqrhfZww>. Acesso em: 03 setembro 2024. Citado 3 vezes nas páginas 19, 20 e 21.

Bento. Refatoração do jogo bicho ufc rampage usando solid e padrões de projeto. 2020. Disponível em: <https://repositorio.ufc.br/bitstream/riufc/55596/1/2020_tcc_gjtrbento.pdf>. Acesso em: 16 fevereiro 2025. Citado na página 32.

Boeg. *Kanban: 10 Passos para Implementação*. 2018. Disponível em: <https://gianfratti.com/wp-content/uploads/2018/04/InfoQBrasil-Kanban10Passos.pdf>. Acesso em: 07 fevereiro 2025. Citado na página 28.

Bolívar Torres. *Pixel art: impulsionada pela nostalgia e pelos games, técnica de baixa resolução está em alta na cultura pop*. 2023. O Globo, 21 junho 2023. Disponível em: <https://oglobo.globo.com/cultura/noticia/2023/06/pixel-art-impulsionada-pela-nostalgia-e-pelos-games-tecnica-de-baixa-resolucao-esta-em-alta-na-cultu ghtml>. Acesso em: 30 abril 2025. Citado na página 18.

Brodi. *Se a Unity Engine é Tão Poderosa, Por que Quase Nenhum Jogo AAA Usa Ela??* 2023. YouTube, Game Brodis, 5 junho 2023. 6min32s. Disponível em:

<https://www.youtube.com/watch?v=LWM8AX_w2yI&ab_channel=GameBrodis>. Acesso em: 10 setembro 2024. Citado na página 26.

Carpenedo. *O QUE É GAME DESIGN?* 2017. YouTube,Carpenedo, 1 de maio de 2017,7min26s. Disponível em: <https://www.youtube.com/watch?v=k09eqlnnWA0>. Acesso em: 17 abril 2025. Citado na página 19.

Carpenedo. *O que é uma ENGINE? (MOTOR GRÁFICO)*. 2017. YouTube, Carpenedo, 24 fevereiro 2017. 5min33s. Disponível em: <https://www.youtube.com/watch?v=Zf4JKBG6_I4>. Acesso em: 15 dezembro 2024. Citado na página 25.

Coelho. *Indústria dos games: a mais lucrativa no mundo do entretenimento*. 2022. Disponível em: <https://gazzconecta.com.br/gazz-conecta/papo-raiz/industria-dos-games-mais-lucrativa-mundo-do-entretenimento/>. Acesso em: 07 fevereiro 2025. Citado na página 15.

Davi Baptista. *O que é um Game Designer? | com Davi Baptista*. 2023. YouTube, Full Sail Brazil Community, 17 julho 2023, 16min20s. Disponível em: <https://www.youtube.com/watch?v=tZwKZlx_KsE>. Acesso em: 17 abril 2025. Citado na página 19.

Dio. *Por que Aprender Unity3D: A Engine do Momento para Desenvolvimento de Games*. 2023. Disponível em: <https://www.dio.me/articles/por-que-aprender-unity3d-a-engine-do-momento-para-desenvolvimento-de-games>. Acesso em: 10 setembro 2024. Citado na página 26.

EBAC. *Unity X Unreal: Qual é o motor de Jogos Mais Adequado para um Projeto?* 2023. Disponível em: <https://ebaconline.com.br/blog/unity-unreal-qual-e-o-motor-de-jogos-mais-adequado-para-um-projeto#:~:text=Originalmente%20projetado%20para%20funcionar%20em,poderosa%20como%20o%20Unreal%20necessitaria>. Acesso em: 10 setembro 2024. Citado na página 26.

Epic Games. *Vampire Survivors: Como o jogo criou o gênero Auto-Shooter e Bullet Heaven*. 2025. Disponível em: <https://store.epicgames.com/pt-BR/news/vampire-survivors-built-genre-autoshooter-bullet-heaven>. Acesso em: 22 fevereiro 2025. Citado na página 21.

Farmando XP. *Explorando as Etapas de Produção de um Jogo [Do Conceito ao Lançamento]*. 2023. YouTube, Farmando XP, 2 agosto 2023. 23min34s. Disponível em: <https://www.youtube.com/watch?v=qRiMNlmTFqI>. Acesso em: 01 maio 2025. Citado na página 22.

Farmando XP. *AINDA VALE A PENA USAR UNITY ENGINE EM 2024?* 2024. YouTube, Farmando XP, 5 abril 2024. 11min50s. Disponível em: <https://www.youtube.com/watch?v=BfKAXGuf5L8>. Acesso em: 16 dezembro 2024. Citado na página 26.

Felipe. *Unity 3D: 11 Motivos para Usar, Será que Vale a Pena?* 2023. YouTube, Married Games, 1 dezembro 2023. 2min39s. Disponível em: <https://www.youtube.com/watch?v=cYxArOp_JQk&ab_channel=MarriedGames>. Acesso em: 10 setembro 2024. Citado na página 26.

Felipe. *Qual Game Engine Escolher? Guia Completo: Unity, Unreal, GameMaker, Godot!)*. 2024. YouTube, Game Dev com Juan, 25 abril 2024. 9min58s. Disponível em: <https://www.youtube.com/watch?v=8B_H7i5lHvI4>. Acesso em: 16 dezembro 2024. Citado na página 26.

Junior. *Por que Unity se Unreal É Melhor?* 2024. Disponível em: <https://www.mixmods.com.br/2024/03/por-que-unity-se-unreal-e-melhor/>. Acesso em: 10 setembro 2024. Citado na página 26.

Lomelino. *Kanban: O que é Kanban? Como funciona o Kanban? | Na Prática*. 2021. YouTube, Na Prática, 24 junho 2021. 5min03s. Disponível em: <https://www.youtube.com/watch?v=hU2NIHOU9aM>. Acesso em: 20 fevereiro 2025. Citado na página 28.

Microsoft. *Visual Studio e Visual Studio Code*. 2025. Disponível em: <https://visualstudio.microsoft.com/pt-br/#vscode-section>. Acesso em: 10 fevereiro 2025. Citado na página 27.

Microsoft. *Visão geral do C#*. 2025. Disponível em: <https://learn.microsoft.com/pt-br/dotnet/csharp/tour-of-csharp/overview>. Acesso em: 16 fevereiro 2025. Citado na página 27.

Montenegro. *O que é um desenvolvedor Unity e como se tornar um?* 2023. Disponível em: <https://ebaconline.com.br/blog/o-que-e-um-desenvolvedor-unity-e-como-se-tornar-um>. Acesso em: 16 dezembro 2024. Citado na página 25.

Naspolini. *Como Testar um Jogo: Dicas Rápidas para um Game Designer*. 2019. YouTube, Fábrica de Jogos | Fabiano Naspolini, 19 de setembro 2019. 8min25s. Disponível em: <https://youtu.be/5dPVx_Pa3eg?si=zvFAnhAaCBQ9Dnml>. Acesso em: 20 fevereiro 2025. Citado na página 45.

Naspolini. *4 Formas de Testar um Jogo e Ver se Está Bom e Divertido*. 2021. YouTube, Fábrica de Jogos | Fabiano Naspolini, 10 de junho 2021. 13min43s. Disponível em: <https://www.youtube.com/watch?v=yd-J6hfma-4>. Acesso em: 20 fevereiro 2025. Citado na página 45.

Naspolini. *Uma Forma Simples de Testar Jogos e Obter Informações Preciosas*. 2021. YouTube, Fábrica de Jogos | Fabiano Naspolini, 11 de fevereiro 2021. 6min23s. Disponível em: <https://www.youtube.com/watch?v=dbUl4D430yE>. Acesso em: 20 fevereiro 2025. Citado na página 45.

Naspolini. *O que é GDD (Game Design Document)? O que tem nele? Única forma de Planejar um Jogo?* 2022. YouTube, Fábrica de Jogos | Fabiano Naspolini, 10 de janeiro de 2020, 17min23s. Disponível em: <https://www.youtube.com/watch?v=_l1XNQ1ofjU>. Acesso em: 09 setembro 2024. Citado na página 22.

Naspolini. *O que é um Protótipo de um Jogo? Já fez um Bom Protótipo de Jogo? | Fábrica de Jogos*. 2022. YouTube, Fábrica de Jogos | Fabiano Naspolini, 9 março 2022. 22min21s. Disponível em: <https://www.youtube.com/watch?v=6_SJPeaYJW0>. Acesso em: 09 setembro 2024. Citado na página 22.

Naspolini. *O que é uma Game Engine? Quais as Mais Usadas no Brasil e Recomendações! | Fábrica de Jogos*. 2024. YouTube, Fábrica de Jogos | Fabiano Naspolini, 30 maio 2024. 10min54s. Disponível em: <https://www.youtube.com/watch?v=6VJ8Qmf920g>. Acesso em: 15 dezembro 2024. Citado na página 25.

Oliveira. *Game Design Document (GDD) Ainda É a Melhor Forma de Planejar seu jogo?* 2020. Disponível em: <https://www.fabricadejogos.net/posts/game-design-document-gdd-ainda-e-a-melhor-forma-de-planejar-seu-jogo/>. Acesso em: 09 setembro 2024. Citado na página 22.

Pavarini et al. Desenvolvimento de jogo retrô–leap quest: Além do asfalto. 041, 2024. Disponível em: <https://ric.cps.sp.gov.br/bitstream/123456789/28309/1/programa%c3%a7%c3%a3o_de_jogos_digitais_2024_desenvolvimento_de_jogo_retro_leap_quest_alem_do_asfalto_amanda_meneghin_pavarini.pdf>. Acesso em: 17 fevereiro 2025. Citado na página 30.

Peniche. *7 etapas no Desenvolvimento de Jogos*. 2024. Disponível em: <https://www.dio.me/articles/7-etapas-no-desenvolvimento-de-jogos>. Acesso em: 08 setembro de 2024. Citado na página 22.

Pereira. *Jogo mobile tem ganhado preferência do público e é aposta do mercado*. 2024. Disponível em: <https://www.correiobraziliense.com.br/diversao-e-arte/2024/04/6840471-jogo-mobile-tem-ganhado-preferencia-do-publico-e-e-aposta-do-mercado.html>. Acesso em: 07 fevereiro 2025. Citado 2 vezes nas páginas 15 e 16.

Soares. *COMO CRIAR UM BOM PROTÓTIPO?* 2020. YouTube, Crie Seus Jogos, 19 agosto 2020. 6min37s. Disponível em: <https://www.youtube.com/watch?v=AVJ3x83TB9E&ab_channel=CrieSeusJogos>. Acesso em: 09 setembro 2024. Citado na página 22.

Soares. *COMO CRIAR UM GDD (GAME DESIGN DOCUMENT) COM MODELO PARA DOWNLOAD*. 2021. YouTube, Crie Seus Jogos, 12 abril 2021. 14min44s. Disponível em: <https://www.youtube.com/watch?v=aqXwal3tRkg>. Acesso em: 27 agosto 2024. Citado na página 22.

Tokio. *8 etapas de Criação de games Para Auxiliar Os desenvolvedores iniciantes*. 2016. Disponível em: <https://dropsdejogos.uai.com.br/noticias/indie/8-etapas-de-criacao-de-games-para-auxiliar-os-desenvolvedores-iniciantes-por-jeannie-novak/>. Acesso em: 08 setembro de 2024. Citado 2 vezes nas páginas 22 e 23.

TOTVS. *Kanban: O que é e como aplicar no seu negócio*. 2023. Disponível em: <https://www.totvs.com/blog/negocios/kanban/>. Acesso em: 20 fevereiro 2025. Citado na página 28.

Unity. *Introdução ao Object Pooling*. s.d. Disponível em: <https://learn.unity.com/tutorial/introduction-to-object-pooling#>. Acesso em: 16 fevereiro 2025. Citado na página 33.

Unity. *ScriptableObject*. s.d. Disponível em: <https://docs.unity3d.com/Manual/class-ScriptableObject.html>. Acesso em: 16 fevereiro 2025. Citado na página 31.

Unity. *Tutorial Unity*. s.d. Disponível em: <https://learn.unity.com/tutorial/65df850fedbc2a082fb11029#>. Acesso em: 16 fevereiro 2025. Citado na página 33.

Valeri. *O que é um motor de jogo E um motor gráfico*. 2023. Disponível em: <https://www.terra.com.br/byte/o-que-e-um-motor-de-jogo-e-um-motor-grafico, bf61968b6b5fbd288cb9a5164e9ab27f56kgaha8.html#google_vignette>. Acesso em: 16 dezembro 2024. Citado na página 25.

Xexéo. *O que são jogos*. 2013. 1–30 p. Disponível em: <https://ludes.cos.ufrj.br/wp-content/uploads/2016/07/LJP1C01-O-que-sao-jogos-v2.pdf>. Acesso em: 8 outubro 2024. Citado na página 15.

Apêndices

Anexos

ANEXO A – Licença MIT