UNIVERSIDADE FEDERAL DO SUL E SUDESTE DO PARÁ
INSTITUTO DE GEOCIÊNCIAS E ENGENHARIAS
Faculdade de Engenharia da Computação
Bacharelado em Engenharia da Computação

Projeto Final de Curso

JOGOS 3D NO SÉCULO XX - UMA JORNADA DE ANÁLISE E DESENVOLVIMENTO EM UNITY ATRAVÉS DOS CLÁSSICOS: STAR FOX, DOOM E SUPER MARIO 64

Lucas Antonio da Silva Lima

Marabá-PA
2024

Lucas Antonio da Silva Lima

JOGOS 3D NO SÉCULO XX - UMA JORNADA DE ANÁLISE E DESENVOLVIMENTO EM UNITY ATRAVÉS DOS CLÁSSICOS: STAR FOX, DOOM E SUPER MARIO 64

Projeto Final de Curso, apresentado à Faculdade de Engenharia da Computação do Instituto de Geociências e Engenharias da Universidade Federal do Sul e Sudeste do Pará, como parte dos requisitos necessários para obtenção do Título de Bacharel em Engenharia da Computação.

Orientador:
Profº. Dr. Manoel Ribeiro Filho

Marabá-PA
2024

Dados Internacionais de Catalogação-na-Publicação (CIP)
Universidade Federal do Sul e Sudeste do Pará
Biblioteca Setorial II da UNIFESSPA

L732j Lima, Lucas Antonio da Silva

Jogos 3D no século XX - uma jornada de análise e desenvolvimento em Unity através dos Clássicos: Star Fox, Doom e Super Mario 64 / Lucas Antonio da Silva Lima.— 2024.

130 f.; (algumas color).

Orientador(a): Manoel Ribeiro Filho

Trabalho de Conclusão de Curso (Graduação) - Universidade Federal do Sul e Sudeste do Pará, Campus Universitário de Marabá, Instituto de Geociencias e Engenharias, Faculdade deComputação e Engenharia Elétrica, Curso de Engenharia da Computação, Marabá, 2024.

1. Engenharia de computação. 2. Vídeo games - Jogos. 3. Jogos para computador - Programação. 4. Realidade virtual. I. Ribeiro Filho, Manoel, orient. II. Título.

CDD: 22. ed. 006.8

Elaborada por Marcelo da Silva Gomes – CRB2/1208

Lucas Antonio da Silva Lima

JOGOS 3D NO SÉCULO XX - UMA JORNADA DE ANÁLISE E DESENVOLVIMENTO EM UNITY ATRAVÉS DOS CLÁSSICOS: STAR FOX, DOOM E SUPER MARIO 64

Projeto Final de Curso, apresentado à Universidade Federal do Sul e Sudeste do Pará, como parte dos requisitos necessários para obtenção do Título de Bacharel em Engenharia da Computação.

Marabá: 18 de Setembro de 2024

BANCA QUALIFICADORA:

Profº. Dr. Manoel Ribeiro Filho
(Orientador - UNIFESSPA)

Profº. Dr. Joao Victor Costa Carmona
(Membro da Banca - UNIFESSPA)

Profª. Dra. Cindy Stella Fernandes
(Membro da Banca - UNIFESSPA)

Marabá-PA

2024

Dedico este trabalho à minha família, especialmente à minha mãe, pelo apoio incondicional e amor que sempre me deram. Dedico também aos meus amigos e colegas de classe, que estiveram ao meu lado em cada passo desta jornada, e a todos os professores, cujas orientações foram essenciais para o meu crescimento.

AGRADECIMENTOS

A realização deste trabalho não seria possível sem o apoio e a colaboração de muitas pessoas que me acompanharam ao longo desta jornada.

Primeiramente, agradeço à minha família, em especial à minha mãe, Rejane, pelo amor incondicional, paciência e encorajamento em todos os momentos, fornecendo-me a base emocional necessária para seguir em frente. Não posso deixar de mencionar alguns familiares que mais me influenciaram, apoiaram e cobraram, ajudando-me a manter o foco nos estudos desde antes de ingressar na universidade: Catiana Lima, Antonio Marruaz, Pedro Lima e Kacio Junior.

Agradeço aos meus professores, cujos ensinamentos foram fundamentais para o desenvolvimento deste trabalho. Em particular, expresso minha gratidão ao meu orientador, Manoel Ribeiro, pelas orientações, paciência e apoio em um projeto incomum na nossa universidade. Um agradecimento especial também à Profª Leslye Estefania, ao Profº. Diego Kasuo, ao Profº. Warley Muricy, à Profª Cindy Stella, ao Profº. João Victor, ao Profº Haroldo Gomes e ao Profº. Cláudio Maciel, que compartilharam seu conhecimento com maestria, forneceram grande apoio e tiveram um impacto significativo na minha formação acadêmica.

Aos amigos que conheci durante a universidade e que levarei para a vida, Lucas Leite, Messias Barros, Arthur Martins, Cristina Leite, Thiago Eleuterio, Henrique Viana e Felipe Coelho, que compartilharam comigo tantas experiências ao longo da graduação e me ofereceram apoio nos momentos de dúvida e cansaço. Vocês tornaram essa caminhada mais leve e divertida.

Aos meus amigos mais próximos, Rodolfo Carvalho, Ronald Carvalho e Antonio Pedro, que me proporcionaram apoio incondicional e momentos de alegria e descanso durante toda a jornada.

A todos, o meu sincero agradecimento!

A escalada pode ser longa, mas a vista vale a pena!

(Taric)

RESUMO

Este trabalho apresenta a análise de três aclamados jogos 3D da década de 1990: Star Fox, Doom e Super Mario 64. Contando um pouco da história dos jogos 3D e sua ascensão durante da década de 1990. O objetivo principal do trabalho é realizar uma análise de como foi o desenvolvimento desses jogos clássicos focado na parte gráfica, buscando identificar seus elementos mais marcantes e compreender suas contribuições para a história dos videogames. A partir dos resultados da análise, foi iniciado o desenvolvimento de um novo jogo 3D com uma reprodução aproximada do estilo visual desses jogos na unity engine. Cada fase do jogo é inspirada em um dos jogos analisados, preservando o estilo visual e algumas mecânicas originais. A intenção não é criar uma reprodução exata dos jogos originais, mas sim uma homenagem que captura a essência desses títulos. Para o desenvolvimento do projeto, foi utilizada a Unity Engine, um motor gráfico poderoso e versátil que permite a criação de jogos 3D de alta qualidade. A Unity oferece diversas ferramentas e recursos que facilitam o processo de desenvolvimento, desde a modelagem 3D até a programação e implementação de mecânicas de jogo. Em conjunto com a Unity Engine, o Blender foi utilizado para a criação dos modelos 3D e animações do jogo. A utilização do Blender foi fundamental para recriar o estilo visual "retro"dos jogos originais, preservando a estética clássica dos anos 90. Este trabalho contribui para a preservação da memória dos videogames e para a compreensão da evolução dos jogos 3D ao longo do tempo. A análise dos jogos clássicos Star Fox, Doom e Super Mario 64 permite identificar as características que os tornaram tão populares e influentes na indústria de videogames. O desenvolvimento do novo jogo, por sua vez, demonstra a capacidade de reimaginar e recriar jogos clássicos em um ambiente moderno, preservando sua essência.

Palavras-chave: Desenvolvimento de Jogos 3D, Vídeo Games, Star Fox, Doom, Super Mario 64.

ABSTRACT

This paper presents an analysis of three acclaimed 3D games from the 1990s: Star Fox, Doom and Super Mario 64. It tells a little about the history of 3D games and their rise during the 1990s. The main objective of the paper is to analyze how these classic games were developed, focusing on the graphical part, seeking to identify their most striking elements and understand their contributions to the history of video games. Based on the results of the analysis, the development of a new 3D game was started with an approximate reproduction of the visual style of these games in the unity engine. Each level of the game is inspired by one of the games analyzed, preserving the visual style and some original mechanics. The intention is not to create an exact reproduction of the original games, but rather a tribute that captures the essence of these titles. The Unity Engine was used to develop the project, a powerful and versatile graphics engine that allows the creation of high-quality 3D games. Unity offers several tools and resources that facilitate the development process, from 3D modeling to programming and implementation of game mechanics. In conjunction with the Unity Engine, Blender was used to create the game's 3D models and animations. The use of Blender was essential to recreate the "retro"visual style of the original games, while preserving the classic aesthetics of the 90s. This work contributes to preserving the memory of video games and to understanding the evolution of 3D games over time. Analyzing the classic games Star Fox, Doom and Super Mario 64 allows us to identify the characteristics that made them so popular and influential in the video game industry. The development of the new game, in turn, demonstrates the ability to reimagine and recreate classic games in a modern environment, while preserving their essence.

Keywords: 3D Game Development, Video Games, Star Fox, Doom, Super Mario 64.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

2D	*Duas Dimensões*
3D	*Três Dimensões*
C#	*C Sharp*
FPS	*First Person Shooter*
GDD	*Game Design Document*
SGDD	*Short Game Design Document*
SNES	*Super Nintendo Entertainment System*

SUMÁRIO

1 INTRODUÇÃO

Os jogos tridimensionais (3D) têm grande popularidade na atualidade, uma tendência que foi impulsionada pela evolução contínua das placas gráficas e das ferramentas de desenvolvimento de jogos. A origem dessa trajetória não se situa no presente, tudo isso se iniciou através de um jogo pioneiro que desempenhou um papel crucial na popularização deste estilo gráfico: o BattleZone(1983) (LEITE, 2006). Este jogo apresentou a um vasto público a experiência de um ambiente virtual tridimensional interativo. A história dos jogos 3D, de forma mais detalhada, será objeto de discussão mais adiante neste trabalho.

Antes de 1992, predominavam os jogos bidimensionais ou aqueles que simulavam um ambiente tridimensional por meio da renderização de estruturas aramadas simples em pixels na tela. Existiam poucas exceções que utilizavam colorização plana, como, por exemplo, o jogo Driller(1987)(PLUS, 2010). Em determinados casos, recorria-se ao uso de imagens previamente renderizadas para criar uma ilusão de tridimensionalidade, apesar de o jogo em si não incorporar qualquer tipo de código para cálculos 3D (SANTOS, 2004).

Uma das principais barreiras ao desenvolvimento de jogos tridimensionais (3D) antes de 1992, residia na necessidade de criar tanto o software quanto o hardware específicos para cada jogo "3D". Por exemplo, o BattleZone exigiu a implementação de um display de geração de vetores e a utilização de três microprocessadores (WALLICH, 2022).

O ano de 1992 marcou uma significativa transformação no universo dos jogos com o lançamento de Ultima Underworld e em seguida o lançamento do jogo Wolfenstein 3D. Ambos jogos com uma movimentação mais livre e mundo texturizado e que serão mais abordados no tópico relacionado a história dos jogos 3D. (SANTOS, 2004).

Na continuação dos anos 90, houve um aumento progressivo no desenvolvimento e lançamento de jogos tridimensionais (3D). Dentre esses, destacam-se clássicos ainda lembrados hoje, como StarFox e Doom, ambos de 1993 e o Super Mario 64. Este trabalho se concentrará na análise desses três jogos emblemáticos, buscando compreender o processo de produção, as dificuldades superadas e as limitações tecnológicas da época que influenciaram a sua criação e lançamento. Essa escolha é devida a popularidade dos mesmos e o impacto que cada um deles trouxe na industria (NINTENDO-BLAST, 2015)(LOWOOD, 2024)(GERARDI, 2016).

Nos dias atuais, as ferramentas e técnicas de produção de jogos evoluíram significativamente, facilitando a produção de jogos independentes. Além disso, conforme previsto no artigo "Evolution of 3D games on mobile phones" (CHEHIMI; COULTON; EDWARDS, 2005), os jogos 3D em dispositivos móveis superaram a maioria dos seus desafios técnicos, tornando o acesso a jogos 3D mais fácil do que nunca. Esse foi mais um dos fatores que tornou esse tipo de jogo extremamente relevante para o mercado de jogos, com muitas empresas focando no desenvolvimento de jogos 3D e na criação de motores de jogos e

outras ferramentas para facilitar a produção desses jogos.

Os “motores de jogos”, comumente referidos pelo termo em inglês “game engines”, são um dos principais fatores que possibilitam a produção de uma vasta quantidade de jogos, tanto por grandes equipes quanto por pequenas, especialmente jogos 3D. Esses motores constituem a base sobre a qual o jogo é construído e executado. Eles compreendem um conjunto de módulos de código de simulação que não são específicos para as lógicas do mundo do jogo. Além disso, incluem módulos de renderização, física e colisões, sistema de animações, sistema de áudio, entre outros (LEWIS, 2002)(GREGORY, 2018).

O motor de jogos Unity será a principal ferramenta do projeto, combinado com outras ferramentas para o desenvolvimento. Atualmente, Unity é um dos motores de jogos mais renomados e populares do mercado, sendo empregado não apenas para o desenvolvimento de jogos 3D, mas também para jogos 2D, simulações para computadores, realidade virtual, consoles e dispositivos móveis (MOHD *et al.*, 2023). Ao utilizar a Unity, o objetivo não é explorar todo o potencial da ferramenta, mas sim utilizá-la como base para entender como seria o processo de reprodução do visual de cada um dos 3 clássicos jogos 3D da década de 90 que foram selecionados para análise desse trabalho, dentro do ambiente moderno de desenvolvimento de jogos. Esta ferramenta poderosa e moderna será capaz de nos fornecer suporte para realizar o processo de construção, focando principalmente no visual, mas também possibilitando a reprodução de algumas de suas mecânicas na linguagem C#, que é a linguagem padrão da *Unity Engine*.

Espera-se que este projeto contribua para o entendimento do desenvolvimento em Unity com jogos 3D. Além disso, espera-se que ele auxilie na preservação da memória desses jogos clássicos, que tiveram um papel fundamental na indústria de jogos como um todo. Mais do que isso, o projeto visa proporcionar uma compreensão mais profunda de como se deu o início dessa tecnologia de mundos virtuais tridimensionais, que hoje é de suma importância não apenas na área de jogos.

1.1 Justificativa

Jogos representam sistemas de extrema complexidade, que frequentemente demandam a colaboração de uma equipe multidisciplinar para o desenvolvimento de um único título. A facilitação do acesso e do entendimento acerca da produção dessa forma de mídia tem contribuído para que cada vez mais pessoas reconheçam o valor inerente a essa indústria em constante crescimento. De fato, a indústria de jogos já ultrapassou as indústrias da música e do cinema em termos de receita, conforme evidenciado por uma pesquisa realizada em 2021, que mensurou o mercado de jogos como sendo maior que as indústrias do cinema e da música combinadas (WAKKA, 2021).

Nesse contexto, torna-se cada vez mais relevante a realização de trabalhos focados

na pesquisa e desenvolvimento de jogos em nosso país. Conforme indicado pela Pesquisa Game Brasil (PGB) de 2023 (GOGAMERS; SXGROUP, 2023), um estudo de campo relacionado aos jogadores de videogames no Brasil, foi revelado que os jogos estão presentes no cotidiano de aproximadamente 70,1% da população brasileira. Isso evidencia que o Brasil possui um grande público consumidor de jogos. No entanto, quando se trata de desenvolvimento de jogos, o país ainda tem uma presença modesta nesse mercado, com apenas mil e quarenta e duas (1042) empresas de desenvolvimento de jogos, conforme apontado pela pesquisa da Associação Brasileira das Desenvolvedoras de Games (Abra Games) de 2023 (CONSULTING, 2023). Esse número é pequeno quando comparado à China e aos Estados Unidos, que lideram o setor com vinte e duas mil (22.000) e dez mil (10.000) empresas de desenvolvimento de jogos, respectivamente, segundo dados da International Game Developers Association (IGDA) ou "Associação Internacional de Desenvolvedores de Jogos" (SIMMONS, 2024).

Estudos voltados para as tecnologias de desenvolvimento de jogos são frequentes, uma vez que novas ferramentas e métodos de produção de jogos são continuamente desenvolvidos para facilitar o processo de desenvolvimento e melhorar a qualidade dos jogos. No entanto, a proposta deste trabalho é diametralmente oposta, com um foco na história e na compreensão do desenvolvimento dos jogos tridimensionais pioneiros e seu processo de evolução. A difusão desse tipo de pesquisa é fundamental para a preservação e homenagem à história dos jogos, que hoje têm um impacto significativo no mercado de entretenimento e são importantes para milhões de pessoas.

Conforme mencionado anteriormente, o processo de desenvolvimento de jogos é extremamente complexo. Para que um indivíduo possa desenvolver um jogo de forma autônoma, é necessário um estudo aprofundado em diversas áreas, que vão desde arte, programação e modelagem até conceitos de engenharia de software e música. Isso se deve ao fato de que um jogo representa a integração de trabalhos provenientes de diferentes áreas (MIRANDA; STADZISZ, 2018).

Nesse contexto, a Unity Engine é destacada como um componente essencial deste projeto. A Unity é um motor de jogos gratuito que simplifica muitos processos no desenvolvimento de um jogo. Ela oferece uma vasta gama de conteúdos e diversas ferramentas de suporte, permitindo-nos compreender e desenvolver grande parte do que é necessário para o funcionamento de um jogo dentro do próprio software.

Diante deste cenário, temos como foco proporcionar uma compreensão e conhecimento aprofundados da história dos jogos selecionados. Paralelamente, o estudo demonstrará as facilidades do desenvolvimento de um jogo tridimensional na atualidade, utilizando a plataforma Unity, uma plataforma moderna e uma das mais populares globalmente.

A escolha da Unity como plataforma de desenvolvimento é complementada pelo uso do Blender, outro software de grande relevância para este projeto. O Blender se destaca

por ser um software gratuito e de código aberto, com uma vasta gama de ferramentas integradas. O Blender permite realizar modelagem 3D, texturização, animação, entre outras tarefas necessárias para a produção de objetos, cenários e personagens do jogo. Apesar de ser de código aberto, o Blender é um software robusto no mercado de arte 2D e 3D, prático e constantemente atualizado graças à comunidade e ao investimento de grandes empresas como AMD, Nvidia e Apple (AUGUSTO, 2021).

Ademais, é importante ressaltar que a pesquisa e o desenvolvimento deste trabalho estão focados em jogos 3D de baixa complexidade visual, não representando, portanto, o potencial total das ferramentas que serão utilizadas. Modelos, animações e texturas serão quase que integralmente elaborados com base na análise dos três jogos da década de noventa selecionados como referência para nossas pesquisas.

1.2 Objetivo Geral

Este trabalho tem como objetivo entender a história e o desenvolvimento de jogos 3D antes do século XXI e desenvolver um jogo 3D na unity engine baseado no visual e mecânicas de três jogos populares da década de noventa(90). Através desse processo, busca-se aprimorar os conhecimentos adquiridos nas disciplinas voltadas à modelagem 3D e ao desenvolvimento de jogos durante a graduação.

1.3 Objetivos Específicos

- Conduzir uma pesquisa exploratória com o propósito de compreender a história dos jogos 3D.
- Entender as complexidades enfrentadas pelas empresas ao desenvolver jogos 3D para o público, com base em uma pesquisa detalhada sobre três jogos populares da década de 90.
- Planejar e desenvolver um jogo 3D na Unity Engine, composto por três fases. Cada fase será desenvolvida com base no visual e nas mecânicas de cada um dos jogos selecionados.
- Compreender o processo subjacente ao desenvolvimento de um jogo 3D na Unity.
- Realizar uma análise e comparação final entre cada um dos jogos e suas respectivas fases, bem como as dificuldades encontradas no desenvolvimento desses visuais e mecânicas dentro da Unity Engine.

1.4 Organização do Trabalho

Além dessa introdução, esse trabalho possui mais seis capítulos, resumidos a seguir.

- Capítulo 2: Neste capítulo, apresentaremos uma revisão dos trabalhos relacionados à área de desenvolvimento de jogos 3D. Abordaremos o surgimento desses jogos, as melhorias ocorridas ao longo dos anos e realizaremos uma análise aprofundada dos três principais jogos que constituem o foco deste trabalho;

- Capítulo 3: Neste capítulo, abordaremos as técnicas e ferramentas utilizadas no desenvolvimento do jogo criado. Discutiremos sobre a função de cada ferramenta para a criação de jogos na atualidade. Além disso, exploraremos as etapas que ocorreram na prática, nas quais foram essenciais para o processo de desenvolvimento;

- Capítulo 4: Neste capítulo, apresentaremos em detalhes o projeto do jogo que foi desenvolvido. Cada fase será minuciosamente explicada, abordando os objetivos relacionados ao aspecto visual e à programação das mecânicas. Esses objetivos estão diretamente associados aos três jogos que foram analisados nos tópicos anteriores.

- Capítulo 5: Este capítulo apresenta a implementação do jogo, modelagem 3D, prioritariamente usando o Blender, e outras ferramentas relacionadas ao visual do jogo. E a criação do jogo na Game Engine(Motor de Jogo) Unity onde serão abordados os processos organizacionais para a execução das tarefas a serem realizadas como a programação dos personagens, sistemas, menus do jogo e outras...

- Capítulo 6: Este capítulo apresenta a análise dos resultados obtidos por meio da pesquisa realizada sobre jogos 3D na década de noventa e o desenvolvimento de um jogo 3D na atualidade, bem como os testes e feedbacks relacionados ao projeto desenvolvido.

- Capítulo 7: Finalmente, apresentamos nossas considerações finais sobre o trabalho realizado. Neste capítulo, discutiremos as conclusões relacionadas à evolução dos jogos 3D e, principalmente, abordaremos o processo de desenvolvimento de jogos 3D em geral, com foco especial nos de pequeno porte na atualidade. Levaremos em consideração toda a pesquisa e o desenvolvimento do nosso próprio jogo.

2 HISTÓRIA DOS JOGOS 3D, REVISÃO DE LITERATURA E TRABALHOS RELACIONADOS

Este capítulo inicialmente apresenta um histórico dos jogos 3D de 1973 até o inicio da década de 1990. Em seguida apresenta em detalhes os 3 jogos escolhidos para estudo basico neste trabalho: Star Fox, doom e Super Mario 64. Finaliza com a apresentação de outros trabalhos que produziram jogos com a Unity.

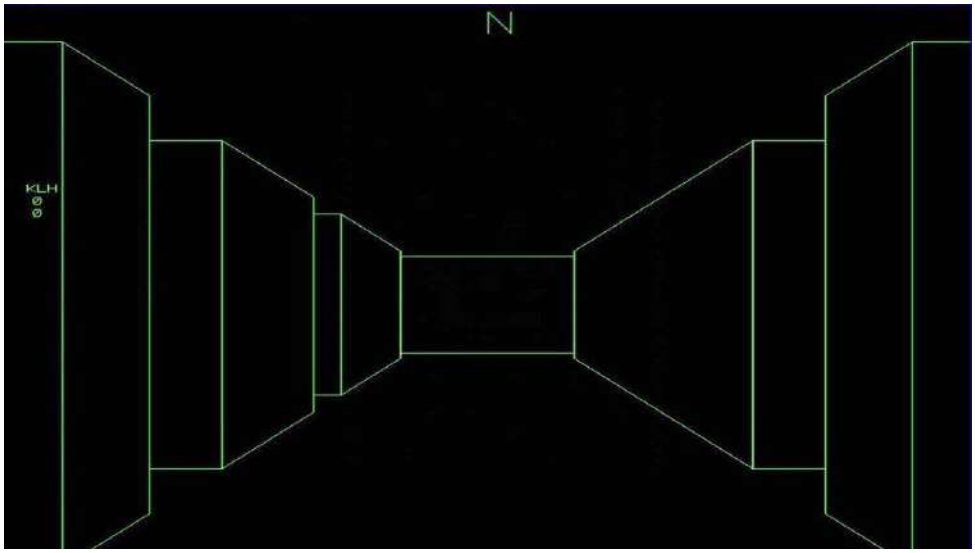
2.1 Jogos 3D

Quando se fala em 'jogos 3D', estamos nos referindo a um tipo específico de jogos de computador que utilizam imagens tridimensionais em todas as dimensões (altura, largura e profundidade). A ideia é que esses jogos proporcionem uma experiência mais imersiva para o jogador, dando vida a cenários mais estilizados ou até mesmo mais realistas (KAUFMAN, 2023).

2.1.1 História (1973-1996)

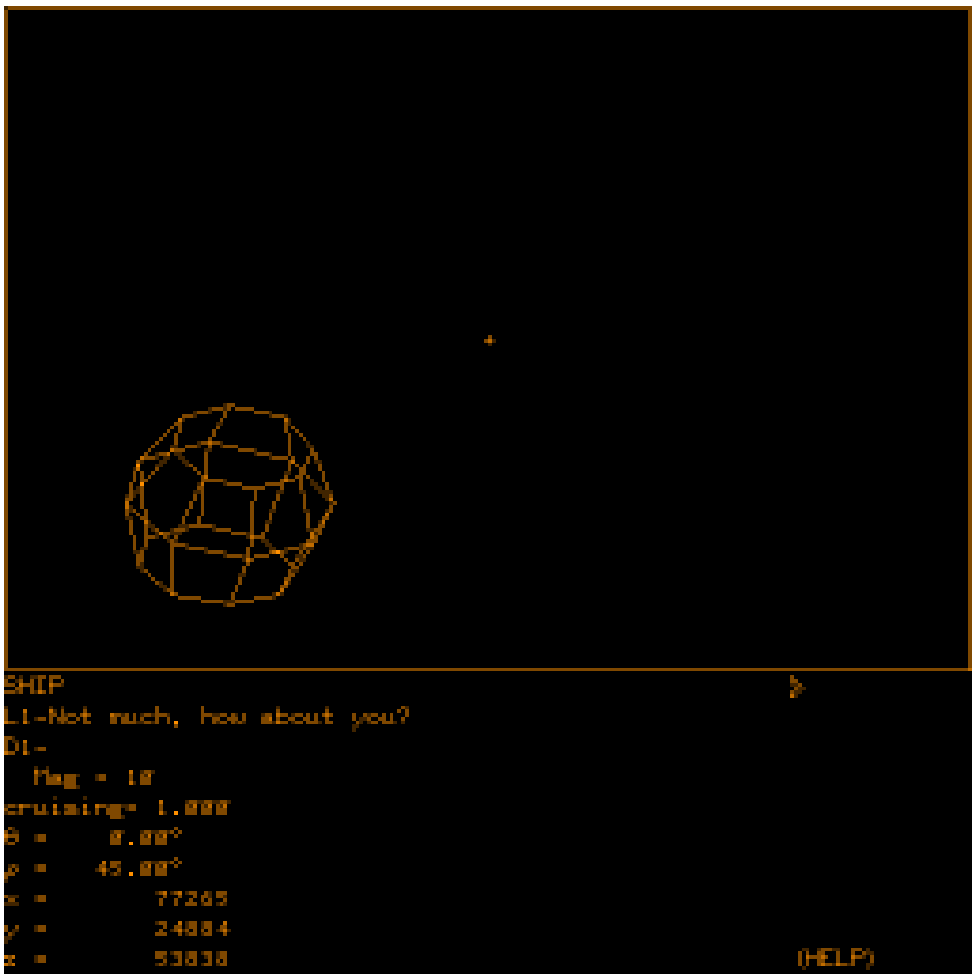
A linha do tempo dos jogos 3D não é bem definida até os dias de hoje, pois muitas grandes empresas estiveram envolvidas e determinadas a desenvolver seus diversos tipos de jogos e plataformas. No entanto, um dos mais aceitos como pioneiro nessa área foi o Maze War, considerado o primeiro jogo em um ambiente tridimensional e também o primeiro a ter jogabilidade em primeira pessoa no mundo (Figura 1). Ele foi desenvolvido no centro de pesquisas da NASA em 1973, em um sistema Imlac PDS-1. Outro jogo que também surgiu algum tempo depois foi o Spasim, criado por Bowery em 1974, com um pequeno espaço simulado, feito no sistema PLATO (Figura 2) (VOORHEES; CALL; WHITLOCK, 2012).

Figura 1 – Maze War (1973)



Fonte: Sonymaster (2021)

Figura 2 – Spasim (1974)

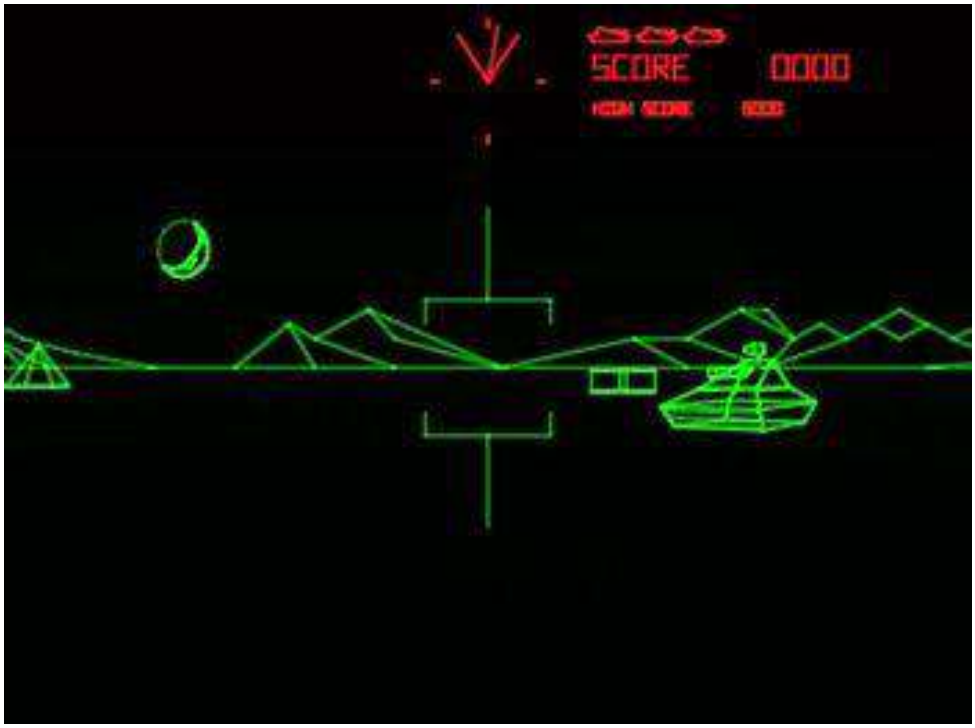


Fonte: Tropes (2022)

O pioneiro dos jogos 3D a alcançar sucesso com o público foi o ‘BattleZone’,

desenvolvido pela Atari. Lançado em 1980, este jogo de temática tanques destacou-se pelo uso de gráficos vetoriais, que compunham uma representação tridimensional e bicolor. ‘BattleZone’ demandou avanços tecnológicos significativos para sua execução. Para sua implementação, foi necessário um display de geração de vetores e três microprocessadores distintos: um 6502 para controle da jogabilidade, um processador customizado para o display e outro, baseado em processadores bit-slice 2901, dedicado aos cálculos matemáticos. O sistema operava com uma tela geradora e os microprocessadores funcionando tanto em paralelo quanto em sequência (Figura 3) (VOORHEES; CALL; WHITLOCK, 2012) (LEITE, 2006) (WALLICH, 2022).

Figura 3 – BattleZone (1980)



Fonte: Tropes (2023)

Em 1987, o jogo ‘Driller’ foi lançado, marcando um avanço significativo no mercado de jogos. Desenvolvido com o motor ‘Freescape’, que era inovador na época, ‘Driller’ foi o primeiro título a utilizar essa tecnologia. O desenvolvimento foi um processo complexo, consumindo 14 meses de trabalho de uma pequena equipe interna da Incentive Software, conhecida como Major Developments. Diferentemente do ‘BattleZone’, ‘Driller’ apresentou um mundo completamente tridimensional, ilustrado por polígonos preenchidos (Figura 4). Embora sua jogabilidade de quebra-cabeças e baixíssimas taxas de quadros não tenham conquistado grande popularidade entre o público, ‘Driller’ estabeleceu um marco importante, aumentando as expectativas quanto à viabilidade de jogos 3D no futuro (FAHS, 2012) (MALLO, 2018).

Figura 4 – Driller (1987)

Fonte: Fahs (2012)

Como já citado no começo deste trabalho o ano de 1992 marcou uma significativa transformação no universo dos jogos. Anteriormente, os ambientes eram representados em 3D vetorizado com movimentação restrita. No entanto, com o lançamento de Ultima Underworld pela Looking Glass Studios, os jogadores tiveram a oportunidade, pela primeira vez, de explorar um mundo tridimensional completamente texturizado (Figura 5). Logo após o lançamento de Ultima Underworld, a id Software, uma empresa então emergente, introduziu um título que inauguraria um novo gênero de jogo: o tiro em primeira pessoa. Este jogo, Wolfenstein 3D, priorizava a ação rápida, permitindo ao jogador mover-se livremente em uma perspectiva de primeira pessoa com alta taxa de quadros (Figura 6), tendo como objetivo atirar em tudo que se movesse (SANTOS, 2004).

Figura 5 – Ultima Underworld (1992)

Fonte: Fandom (2023)

Figura 6 – Wolfenstein 3D (1992)

Fonte: Id Software (2024)

Por fim, chegamos a 1993, ano em que grandes jogos 3D começaram a ser lançados, destacando-se 'Doom' para computador e 'Star Fox' para super nintendo. 'Doom', desenvolvido pela mesma empresa responsável por 'Wolfenstein 3D', manteve a premissa original, mas introduziu significativas melhorias gráficas e na construção do mundo tridimensional do jogo, aspectos que serão explorados mais detalhadamente em seções subsequentes. Embora o console Nintendo 64, um hardware que viria a ter um impacto considerável no mercado, ainda não tivesse sido lançado, foi a partir desse ano que se observou a ascensão dos jogos 3D. Em 1996, o Nintendo 64 foi lançado no Japão, acompanhado de títulos emblemáticos, incluindo 'Super Mario 64' (SANTOS, 2004)(NINTENDO, 2024b).

2.2 Star Fox (1993)

Star Fox, conhecido como Starwing na Europa, é um jogo eletrônico do gênero 'rail shooter', no qual o jogador controla a nave espacial do protagonista, Fox, em um ambiente tridimensional e enfrenta diversos tipos de inimigos ao longo da jornada. Lançado em 1993, foi desenvolvido e publicado pela Nintendo, com colaboração da Argonaut Software, para o Super Nintendo Entertainment System (SNES) (IGDB, 2024). Esta seção abordará a história de 'Star Fox', os desafios encontrados durante seu desenvolvimento, além de aspectos visuais e mecânicos do jogo.

2.2.1 Nascimento

A história do Star Fox se inicia com envolvimento da Argonaut Software, um pequeno estúdio do Reino Unido com grandes ideias, famoso por seu impressionante título para computador doméstico Starglider(1986). O trajeto até a Argonautse juntar com a gigante Nintendo foi feito de forma controversa quebrar a segurança de um dos consoles da época, Game Boy, para que pudessem colocar seu jogo e apresentar para alguém importante da empresa. Se utilizando de uma descoberta que para inicializar seu jogo no console da Nintendo, eles só precisariam que o logo "Nintendo" que está no topo da tela e que viesse para o meio. O maior problema estava em não poder utilizar a marca em seu jogo, então descobriram uma forma de burlar a proteção se utilizando de um resistor e um capacitor, fazendo o sistema ler a palavra 'nintendo' e inicializar o seu jogo.

Após conseguirem executar sua demo(demonstração) no console, Jez San, fundador da Argonaut, correu para a CES(Consumer Eletronic Show) daquele ano para encontrar Don James, um dos mais experientes desenvolvedores da Nintendo que ficou totalmente surpreendido com o o jogo e também com a forma que conseguiram rodar a demo dentro do console. Outros integrantes da empresa viram a demo do jogo de San e ele fez a oferta para que trabalhassem juntos, pois ele e sua equipe em Londres eram bons com jogos 3D, algo que a Nintendo relutava em explorar até então. Com essa demo ele conseguiu impressionar alguns desenvolvedores, assim conseguindo o interesse da gigante do mercado dos jogos. Ao retornar ao Reino Unido, San recebeu um convite de reuniões de grandes oficiais da nintendo, San explica."Eles me levaram de avião, me acomodaram no Kyoto Royal Hotel e eu me encontrei com o cara grande. Eles me disseram que queriam fazer três jogos conosco e explicaram seu desejo de que ensinássemos nossa tecnologia 3D. Passei muito tempo com a gerência da Nintendo imaginando o que poderíamos fazer. O lado comercial e de relacionamento deu muito trabalho e não pode ser subestimado; a Nintendo queria se sentir confortável trabalhando com uma empresa externa e estrangeira. Eu tinha que mostrar a eles que não só poderíamos entregar os produtos, mas que eles também poderiam confiar em nós."

A missão que foi lhes dada, foi criar um jogo 3D para SNES. O desafio era grande: o SNES não foi originalmente projetado para lidar com gráficos 3D. Para superar essa limitação, a Argonaut Games desenvolveu o chip Super FX, um processador RISC customizado que possibilitava a renderização de gráficos 3D complexos no console. Ela não só desenvolveu o hardware que seria responsável para a parte 3D, como também o software e quase toda a codificação do jogo que veio a ser desenvolvido, Star Fox.

A colaboração entre a Argonaut Games e a Nintendo foi essencial para o sucesso do Star Fox. A Nintendo juntamente do Shigeru Miyamoto, forneceu os conceitos, personagens e história para o jogo, enquanto a Argonaut Games foi responsável pela tecnologia e pela programação.

A equipe da Argonaut Games se mudou para o Japão para trabalhar em estreita colaboração com a equipe da Nintendo durante o desenvolvimento do jogo. Essa imersão na cultura japonesa foi fundamental para entender a visão da Nintendo para o Star Fox e para criar um jogo que ressoasse com o público japonês.

Star Fox foi lançado em 1993 e foi um grande sucesso comercial e crítico. O jogo foi elogiado por seus gráficos inovadores, jogabilidade emocionante e design de personagens memoráveis. Star Fox vendeu mais de 4 milhões de cópias em todo o mundo e se tornou um dos jogos mais populares do SNES (MCFERRAN, 2014).

2.2.2 Visual 3D

Conforme mencionado anteriormente, a concepção de 'Star Fox' foi viabilizada pelo desenvolvimento do chip Super FX , também conhecido pelo acrônimo 'MARIO', que representa 'Mathematical, Argonaut, Rotation, & Input/Output'. O Super FX é um processador RISC (Reduced Instruction Set Computer) de 16 bits, projetado especificamente para modelar e renderizar polígonos. Esse processador funcionava como um acelerador gráfico, responsável por modelar e renderizar polígonos, sprites e efeitos 2D em um buffer de quadros na RAM. Posteriormente, esses dados eram enviados para a arquitetura central do SNES para serem exibidos (TACHYON ARURU-SAN, 2023).

Mesmo com essa grande evolução, a quantidade de polígonos não poderia ser alta, tendo em vista a quantidade limitada de memória e capacidade de processamento. Observando o jogo, é notável que todos os cenários e personagens possuem uma quantidade baixa de polígonos, utilizando apenas o mínimo necessário para criar uma identidade visual agradável e possível de ser processada, além de empregar cores simples, sem texturas mais complexas, na Figura 7 é mostrada uma cena da primeira fase e na Figura 8 uma cena da última fase.

Figura 7 – Star Fox - Primeira fase

Fonte: Garrett (2006)

Figura 8 – Star Fox - Última fase

Fonte: Dude984 (2024)

2.2.3 Jogabilidade

Semelhante a outros jogos de “scrolling shooters” da época, o jogador era limitado a controlar a aeronave apenas na área permitida da tela, seguindo um caminho pré-determinado de forma linear. Apesar disso, em Star Fox, o jogador tem pleno controle da aeronave, podendo desviar de inimigos e obstáculos, controlando-a enquanto mira em outros inimigos; disparar lasers e bombas; e executar habilidades para fugir de inimigos, como impulso, frenagem e rotação em torno do próprio eixo. Embora a maior parte do jogo seja em terceira pessoa, em alguns trechos o jogador tem a experiência de jogar em primeira pessoa, conseguindo visualizar o painel da aeronave (PLAMZDOOM SIRKINSELLA98, 2023).

2.3 Doom 1993

Doom é um jogo de tiro em primeira pessoa (first-person shooter, FPS) desenvolvido pela Id Software e lançado em 1993. Seu lançamento teve grande impacto nos jogos para computadores pessoais (personal computers, PCs), mudando a direção de muitos aspectos que esses jogos possuíam na época (LOWOOD, 2024).

2.3.1 Nascimento

A história de Doom se inicia algum tempo antes da criação da empresa que o desenvolveu. A Id software nasceu de um grupo de programadores liderado por John Romero e John Carmack, formado no Texas para criar jogos mensais como funcionários da revista Softdisk. Durante esse tempo da Softdisk o grupo desenvolveu alguns jogos de plataforma viciantes e com base no sucesso destes jogos, o grupo formou a id Software em fevereiro de 1991 (LOWOOD, 2024).

Desde o seu ínicio a Id software focou-se em gráficos superiores, tendo Carmack já demonstrado a capacidade dos computadores pessoais de rivalizar com os consoles através de uma versão de Super Mario brothers 3 da Nintendo, que ele mesmo desenvolveu. Logo ele voltou sua atenção para jogos 3D, assim desenvolvendo uma engine gráfica para o Wolfenstein 3D (Figura 6), que como já citado anteriormente um jogo de FPS com alta taxa de quadros. Este jogo foi o precursor de Doom, preparando a Id software para o seu próximo grande passo neste gênero de jogos (LOWOOD, 2024).

Durante a produção de uma versão paralela de Wolfenstein, que estava sendo desenvolvida no motor do Wolfenstein original, Carmack teve a oportunidade de trabalhar em novas melhorias para seu motor. Grandes aprimoramentos foram implementados em uma outra versão do motor, que passou a se chamar Shadowcaster, com novidades notáveis: iluminação reduzida, portas e tetos mapeados com texturas e alturas variáveis para as

paredes. No desenvolvimento de Doom, o cenário foi uma das principais problemáticas, havendo discordância sobre como deveria ser a progressão dentro do jogo. Carmack desejava que o jogo fosse linear e sem sistema de níveis, enquanto Tom, o game designer da equipe, acreditava que o sistema de níveis proporcionava uma melhor experiência para o jogador. Tom, ao longo de dois meses, escreveu um documento de design, que nomeou como "Bíblia Doom", baseado na ideia inicial de Carmack. Porém, naquele ponto, a equipe já havia mudado de ideia, e o jogo não seguiria uma progressão linear, tornando a "Bíblia Doom" inútil.

Focando nos personagens, os monstros implementados no jogo vieram diretamente das visões aterrorizantes dos desenvolvedores, sendo elas as criaturas mais aterradoras e demoníacas que eles podiam imaginar. Desta vez, os monstros pareciam mais reais, pois utilizaram uma nova técnica de animação e outros recursos.

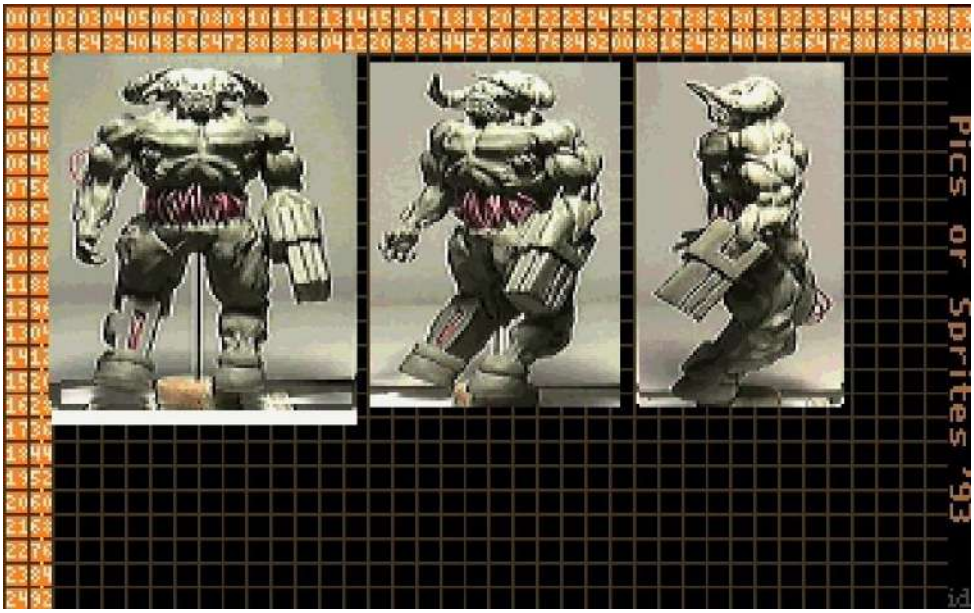
No decorrer do processo de desenvolvimento do jogo, os cenários continuavam sendo um desafio. Embora já tivessem um conceito elaborado por Tom, este era muito focado em arquitetura militar e estava muito próximo do que havia sido estabelecido na "Bíblia Doom", já descartada pela equipe. Após muitos problemas, Romero, outro desenvolvedor da equipe, conseguiu criar um conceito mais próximo do que a equipe necessitava, com cenários maiores, estranhos e abstratos. Um dos últimos passos para a conclusão do jogo foi a contratação de um novo game designer, Sandy, que ficou responsável por definir vários aspectos voltados à experiência do jogador e também a experiência dentro de alguns níveis.

Muito tempo foi investido no desenvolvimento das armas que seriam utilizadas no jogo, e ao fim do desenvolvimento todas já estavam prontas. Sob orientação de Romero, Bobby criou uma trilha sonora no estilo techno metal, e para os efeitos sonoros dos monstros, ele utilizou uma mistura de sons de vários animais. Após alguns meses de trabalho de divulgação, em dezembro de 1993, Doom foi lançado ao mundo com sua tecnologia e gráficos revolucionários (KUSHNER, 2004).

2.3.2 Visual 3D

O desenvolvimento do visual de Doom foi um pouco diferente. Em vez de desenhar quadro a quadro cada personagem, como foi feito em Wolfenstein, os desenvolvedores esculpiram os personagens em argila e, depois, os filmaram em diferentes posições. A partir dessas filmagens, eles obtiveram os quadros necessários, colorindo e programando as imagens com um programa desenvolvido pelo próprio John Carmack. Dessa forma, os personagens se moviam no jogo com uma aparência mais realista e menos cartunesca (Figura 9). De maneira semelhante, foram implementadas as mãos do jogador, capturando imagens das mãos de um dos desenvolvedores em um fundo azul, que foram aplicadas no jogo juntamente com as armas (KUSHNER, 2004).

Figura 9 – Fotos de um personagem modelado em argila



Fonte: Id Software (2024a)

O cenário do Doom foi produzido em um editor de mapas, onde os desenvolvedores tinham a possibilidade de "desenhar" o mapa, criando as paredes nos formatos que achassem necessários e também texturizando-as. A quantidade de polígonos não seria grande, mas já era uma grande melhoria se comparado ao jogo anterior, onde só era possível criar paredes com ângulos de 90º (KUSHNER, 2004).

Outra informação muito importante a respeito do visual de Doom é que ele era muito pesado para os computadores da época. Para que o jogo pudesse ser renderizado e ainda assim manter uma boa taxa de quadros, foi utilizado um processo de programação chamado Binary Space Partitioning (BSP, particionamento de espaço binário). Esse processo ajuda a renderizar modelos dividindo-os em seções maiores ou folhas de dados, em vez de desenhar lentamente cada pequeno polígono. No caso de Doom, o BSP foi usado não apenas para um modelo, mas para todo o mundo virtual do jogo, possibilitando que, mesmo com uma quantidade tão grande de elementos, o jogo funcionasse sem problemas de desempenho (KUSHNER, 2004).

No final, o visual de Doom foi uma mistura de 2D e 3D. Embora fosse um mundo virtual tridimensional, todos os personagens e objetos nos cenários eram 2D, sendo sprites desenhados ou fotos que foram tratadas e incorporadas no jogo como sprites (Figura 10).

Figura 10 – Visual do Doom

Fonte: Id Software (2024b)

2.3.3 Jogabilidade

A jogabilidade de Doom, apesar de simples, oferece muita variedade. Com a fusão de horror e ficção científica, o jogo apresenta um FPS com uma ampla gama de armas que o jogador pode alternar usando as teclas, além de inimigos variados ao longo das fases, onde se explora uma base em uma lua de Marte. Durante o jogo, é possível percorrer as salas, coletar itens ao passar por cima deles e aumentar os status que estão visíveis na interface da tela (Figura 10). Uma das limitações é a movimentação da câmera, permitindo apenas movimentos horizontais; não é possível olhar para cima ou para baixo.

2.4 Super Mario 64 (1996)

Super Mario 64 é um jogo eletrônico do gênero plataforma em 3D, lançado em 1996, sendo o primeiro jogo 3D da franquia Mario. Diferente dos jogos anteriormente discutidos neste trabalho, que marcaram o início de suas respectivas franquias, este jogo coloca o jogador no controle de Mario em um mundo tridimensional, combinando visual e mecânicas inovadoras, mas também aplicando a essência dos jogos antigos da franquia. Esse jogo foi um dos grandes marcos na história dos videogames. Assim como todos os outros títulos, foi desenvolvido e publicado pela Nintendo, para o console Nintendo 64 (N64) (ODDO, 2021). Esta seção abordará fatos sobre seu processo de desenvolvimento, além de aspectos visuais e mecânicos do jogo.

2.4.1 Nascimento

Shigeru Miyamoto, criador da franquia Mario entre outros jogos da Nintendo, teve a ideia de criar um Mario em 3D durante o desenvolvimento de Star Fox, no ano de 1990. Sua ideia inicial era utilizar o chip SUPER FX para desenvolver o jogo no SNES, mas eventualmente ele mudou de ideia e decidiu desenvolver o jogo para o Nintendo 64, já que esse novo console teria um controle com mais botões e um "stick analógico" no centro (figura 11), permitindo um giro fluido dentro do jogo.

Figura 11 – Controle do Nintendo 64

Fonte: Nintendo (2024a)

Desde o início do desenvolvimento de Super Mario 64, o foco de Miyamoto foi a movimentação de Mario, visando utilizar da melhor forma o stick analógico. Para isso, desenvolveram um programa com uma sala simples com blocos, onde poderiam testar as mecânicas de movimentação, como correr, pular e escalar. Eles sabiam que alcançar um controle suave seria um grande avanço no desenvolvimento do jogo. A movimentação era um dos pontos-chave do jogo, então durante a produção, os designers de Miyamoto criaram cerca de 250 padrões de animação para Mario, dos quais apenas 193 foram utilizados no jogo.

Uma das tarefas dos desenvolvedores foi estabelecer as regras para um jogo de plataforma 3D, já que esse conceito era inovador. Uma das primeiras ideias de Miyamoto foi posicionar a câmera atrás do jogador. Além disso, ele decidiu que a câmera seria controlada por um personagem chamado de "Lakitu", que só seria visível em uma sala espelhada dentro do jogo.

Os mapas que seriam os níveis do jogo, foram construídos de conceitos e rascunhos facilitando o design dos estágios do jogo. Miyamoto explica: "Foi como esculpir um diorama em argila. Primeiro você faz uma forma bem geral. Por exemplo, com o estágio King Bob-omb, para nosso design inicial e geral, teríamos aquele rio no meio do mapa, que você

atravessa para chegar à área do chefe, que fica no topo de uma grande colina que você tem que gire e gire conforme você sobe. Mas digamos que colocamos Mario naquele mapa e o movemos, percebemos que o rio corre muito rápido e o leva embora (risos), e isso é muito difícil para os jogadores, então trocamos aquele rio por um vale deserto, como a Morte. Vale. Portanto, a forma permanece a mesma, mas gradualmente adicionamos mais e mais ideias, mudando o mapa à medida que avançamos."

Muito se aprendeu durante o desenvolvimento de Super Mario 64. Uma das crenças de Miyamoto era que, se algo não pudesse ser vencido, não seria divertido. No entanto, durante a fase de testes, quando crianças jogaram o jogo, elas passaram horas explorando e tentando fazer coisas diferentes. Ao receber feedback, muitas crianças expressaram o desejo de voltar e jogar mais. Isso mudou a percepção de Miyamoto sobre como a dificuldade de um jogo afeta sua diversão.

Em junho de 1996, junto com o lançamento do Nintendo 64 no Japão, Super Mario 64 foi lançado e se tornou um dos jogos mais aclamados pela crítica especializada e pelo público. Até hoje, ele é frequentemente incluído em listas dos melhores jogos de todos os tempos (GOODALL, 2021).

2.4.2 Visual 3D

O visual de Super Mario 64 é o mais avançado entre os três jogos discutidos, graças à cooperação entre a Nintendo e a Silicon Graphics (SGI), uma antiga fabricante de soluções computacionais de alto desempenho, especializada em processamento de imagens tridimensionais. Com essa união, eles conseguiram integrar um sistema gráfico SGI em um pequeno console, que viria a se tornar o Nintendo 64 (HALL, 2024)(PEDDIE, 2020). Apesar disso, a memória ainda era um problema. Portanto, é fácil observar a baixa quantidade de polígonos no jogo, caracterizando-o como um jogo "low poly" (poucos polígonos) e com o uso de texturas de baixa resolução. Além disso, outra técnica utilizada no jogo são os sprites, que são posicionados de forma a estarem sempre virados para a direção da câmera, dando ao jogador a sensação de tridimensionalidade (figura 12).

Figura 12 – Super Mario 64

Fonte: Nintendo (2024c)

2.4.3 Jogabilidade

Na parte de jogabilidade, Super Mario 64 é surpreendente. O jogo apresenta uma vasta quantidade de mecânicas, desde diversos tipos de pulos até natação, corrida, pisar em inimigos para derrotá-los, socos, cambalhotas e itens especiais que conferem poderes de curta duração. Além de todas essas habilidades do personagem, o jogo também inclui diversas mecânicas dentro dos mapas, como deslizar em pistas arriscadas e superar situações difíceis, como pedras que se movimentam e locais acessíveis apenas nadando, consumindo o fôlego do personagem (NINTENDO,). Com todos esses recursos, o jogo busca proporcionar uma jogabilidade suave e dinâmica.

2.5 Trabalhos relacionados

Nesta sessão, vamos abordar alguns trabalhos correlatos ao nosso, focado em pesquisa e desenvolvimento de games dentro da Unity Engine, que é nessa principal ferramenta.

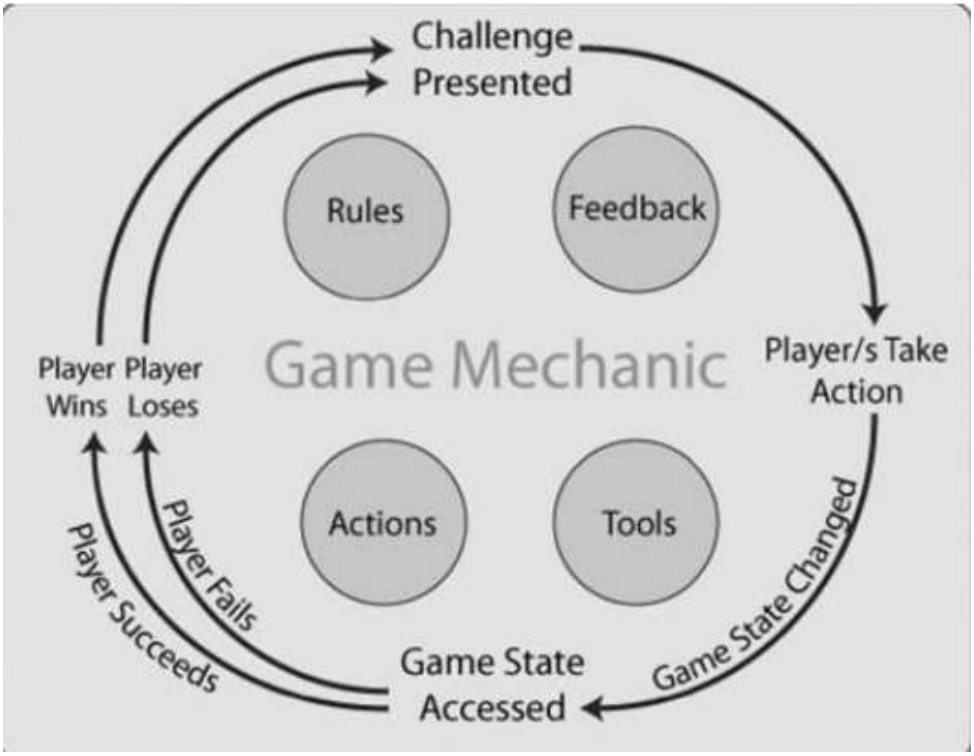
2.5.1 Desenvolvimento de um jogo FPS 3D com Unity

O trabalho de (XIA, 2014) é um estudo de caso sobre o desenvolvimento de um jogo 3D na Unity 4, especificamente um jogo do gênero FPS. Como resultado, foi criado um jogo 3D dentro do gênero proposto.

Como este trabalho é um estudo de caso, ele apresenta uma base teórica sólida e detalhada sobre a Unity e suas ferramentas utilizadas no desenvolvimento do jogo. Além

disso, há uma seção focada na teoria do círculo de mecânicas, que explica um ciclo de funcionamento de jogabilidade que deve ser criado dentro do jogo, sendo esse uma das bases de construção do jogo (Figura 13).

Figura 13 – O circulo de mecânica do jogo

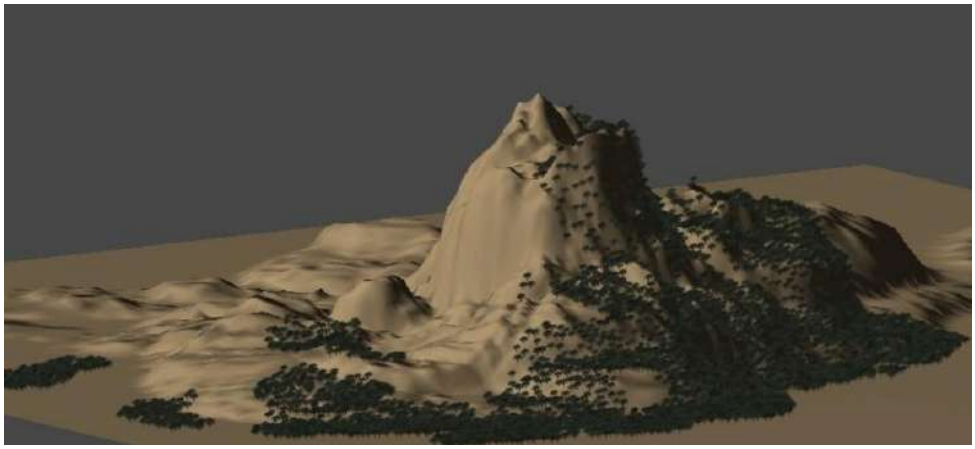


Fonte: Xia (2014)

Um ponto importante a ser mencionado é que, neste trabalho, foram utilizados vários pacotes de recursos da própria loja da Unity. Entre os recursos obtidos, estavam pacotes de controlador de personagem, terreno, texturas para o céu e criadores de árvores. Quando se desenvolve um jogo sozinho, há muitas funções a serem realizadas que demandam muita habilidade. Para poupar tempo e conseguir focar no essencial, muitos desenvolvedores optam por usar recursos disponíveis na internet e modificá-los para seus respectivos jogos.

Mesmo com o uso de recursos que aceleraram o processo de desenvolvimento do jogo, ainda havia muito trabalho a ser feito. Foi utilizada uma ferramenta de criação de terreno da própria Unity para desenvolver o nível onde o jogo se passaria, adicionando topografia, pintura de texturas coesas com o terreno e árvores (Figura 14). Além dos elementos do terreno, foram adicionadas caixas para o jogador se curar durante os combates e os próprios inimigos.

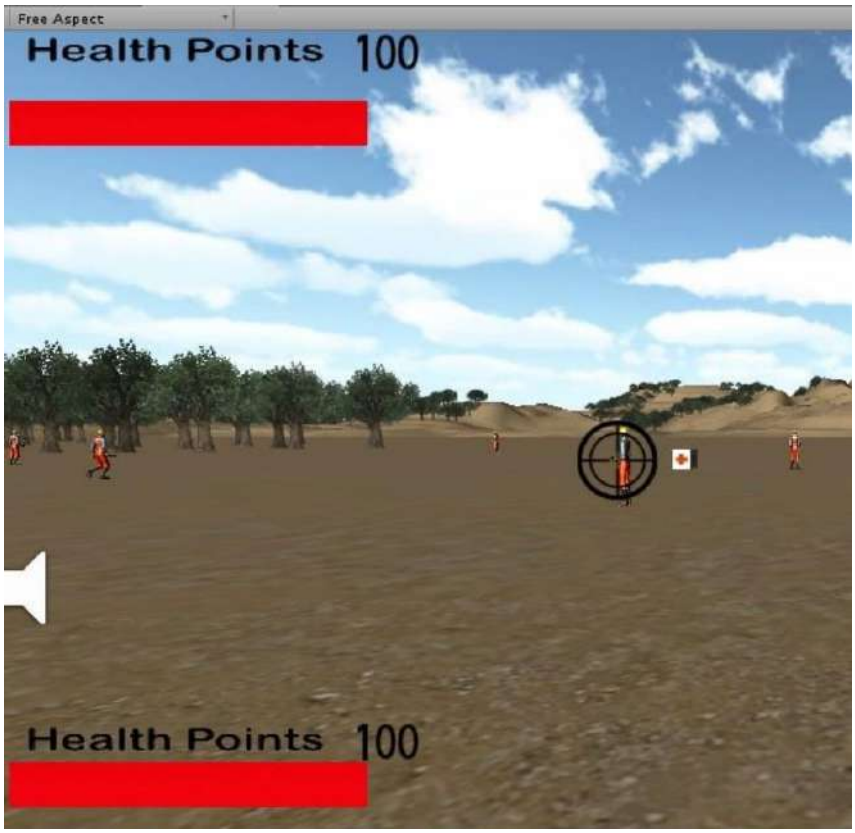
Figura 14 – O contorno do terreno



Fonte: Xia (2014)

Em relação à visão do jogador, era algo bem definido, pois o jogo era em primeira pessoa. Além disso, este era um jogo em primeira pessoa onde o jogador atuava como um sniper, portanto, havia um ponto de mira na tela para que, com o mouse, fosse possível mirar e atirar nos inimigos. Também havia uma barra de vida na interface (Figura 15).

Figura 15 – O efeito da mira e a barra de vida dentro de jogo



Fonte: Xia (2014)

Além disso, toda a programação dos sistemas foi realizada através dos scripts da Unity, escritos em C#.Isso inclui o comportamento dos inimigos, que atacavam o jogador

quando ele se aproximava do alcance deles ou quando atirava neles. O sistema de danos permitia que o jogador sofresse dano dos inimigos ou causasse dano a eles, removendo um valor numérico da barra de vida (Figura 16). Outro sistema importante para o jogo eram as caixas de cura, que recuperavam alguns pontos de vida do jogador ao serem tocadas. No entanto, essas caixas eram destruídas após o uso, criando um equilíbrio de dificuldade no jogo, onde o jogador tinha uma quantidade limitada de vida para vencer todos os inimigos.

Figura 16 – Barra de vida afetada por ataques de inimigos

Fonte: Xia (2014)

Podemos observar que o trabalho de desenvolvimento de um FPS na Unity possui relações com o nosso próprio estudo, já que ambos são estudos de caso sobre o desenvolvimento de um jogo 3D na Unity. Embora o projeto mencionado seja totalmente focado na jogabilidade FPS, enquanto o nosso projeto é focado nos aspectos dos jogos 3D da década de 90, incluímos em nosso escopo um nível voltado ao tiro em primeira pessoa, o que estabelece ainda mais conexões entre os trabalhos.

2.5.2 Desenvolvimento de jogos com Unity

O trabalho de (DANSIE, 2013) tem como objetivo principal explorar o design de diferentes gêneros de jogos, entender a funcionalidade de diversas mecânicas de jogo utilizando a Unity Engine como plataforma de desenvolvimento e examinar os efeitos dos jogos em adultos e crianças.

Dentro deste projeto, são abordados vários aspectos sobre o desenvolvimento de jogos na Unity, desde características do próprio software Unity até aspectos de lógica de programação necessários para o desenvolvimento de jogos. Ao todo, foram desenvolvidos seis jogos de pequeno porte para explorar as diferentes formas de se criar um jogo na Unity Engine.

O primeiro jogo foi o “Brick Breaker”(Quebra-Tijolos). Este jogo é uma adaptação do clássico jogo de quebrar blocos, onde o jogador controla uma barra para rebater uma bola que deve destruir os blocos. No entanto, nesta versão, o jogo é construído em um cenário 3D e utiliza um robô para lançar a bola (Figura 17). O jogo utiliza a física para determinar as colisões entre a bola e os blocos, bem como para o movimento da bola.

Figura 17 – Captura de tela de um nível do jogo Brick Breaker

Fonte: Dansie (2013)

O segundo jogo se chama “Flips”. Trata-se de um jogo de memória básico, focado na mecânica de virar cartas para tentar encontrar os pares iguais. Neste jogo, não foram utilizados muitos recursos visuais; o maior responsável pelo funcionamento é o próprio código que controla a mecânica, já que é um jogo com visual 2D simples (Figura 18).

Figura 18 – Captura de tela do Flips mostrando duas cartas não correspondentes viradas

Fonte: Dansie (2013)

O terceiro jogo foi “Mouse in a Maze”(Rato em um labirinto). Este é um jogo

simples no qual o jogador controla um rato dentro de um labirinto. O objetivo é encontrar o caminho através do labirinto utilizando o mouse ou o acelerômetro, se estiver jogando em um dispositivo móvel. A mecânica principal envolve navegação e resolução de puzzles, onde o jogador deve evitar armadilhas e encontrar o caminho mais eficiente para a saída. Assim como o jogo anterior, este também é um jogo 2D simples, que utiliza imagens em 2D para criar o ambiente do jogo (Figura 19).

Figura 19 – Captura de tela do nível 1 do jogo Mouse in a Maze

Fonte: Dansie (2013)

O próximo é o “Flight Game”(Jogo de Voo), um clássico jogo de tiro com rolagem lateral, onde o jogador controla um dragão enquanto voa e atira nos inimigos. O aspecto especial deste jogo é a implementação de um método de entrada por toque, mais focado em dispositivos móveis. Nesse método, no canto inferior esquerdo da tela, há um pequeno círculo sólido contido em um círculo maior. O círculo menor é arrastado do centro na direção que o jogador deseja ir. Em alguns jogos, quanto mais distante do centro o círculo estiver, mais rápido o personagem se moverá. Basicamente, trata-se de um joystick digital na tela. Em relação ao visual, o jogo mantém o padrão, sendo totalmente 2D e utilizando sprites (Figura 20).

Figura 20 – Captura de tela do joystick digital usado para controlar o dragão

Fonte: Dansie (2013)

Além desses quatro jogos apresentados, o projeto inclui mais dois jogos que seguem o mesmo padrão de 2D, envolvendo mecânicas de colisão, porém abordando gêneros diferentes. Cada um desses jogos foi desenvolvido com o objetivo de explorar diferentes aspectos dos jogos em computadores e, principalmente, em dispositivos móveis, como entrada por toque e uso de acelerômetros. O projeto também visa fornecer uma base de aprendizado sobre as mecânicas e técnicas envolvidas na criação de jogos utilizando a Unity Engine.

O trabalho apresenta o desenvolvimento de jogos de gêneros diferentes, similar ao que é proposto neste documento, desde a ideia de criar jogos variados, cada um explorando diferentes mecânicas e funcionalidades, como controle de toque, física de objetos e IA básica. Além disso a escolha da Unity, justificada por sua capacidade de desenvolvimento, permitindo a implementação e teste dos jogos de formas variadas e multiplataforma.

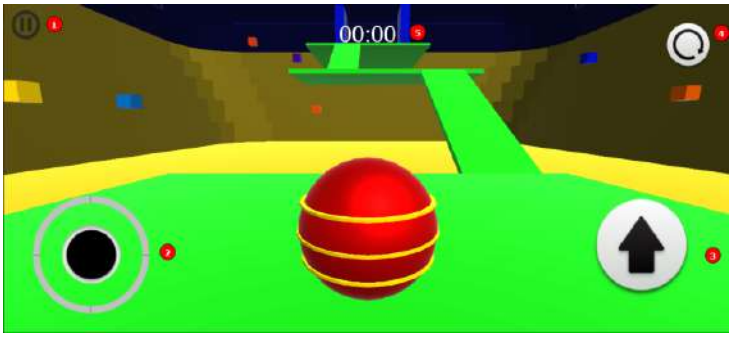
2.5.3 Análise e Desenvolvimento de um jogo Digital com Unity

O trabalho de (MARCONI, 2023) trata da análise e construção de um jogo 3D desenvolvido na Unity Engine. A análise é um ponto relevante dentro do trabalho, com o objetivo de compreender o processo de desenvolvimento de um jogo nesse motor. Assim como os trabalhos anteriores mencionados neste texto, este foca em explicar ferramentas e componentes da Unity Engine. Além disso, aborda o Blender e questões didáticas como a história dos jogos digitais e as classificações dadas aos jogos.

O jogo desenvolvido durante a pesquisa, segundo o autor, foi denominado BallRide. Trata-se de um jogo para dispositivos móveis com mecânicas simples, onde o jogador controla uma bolinha ao longo da fase até chegar ao final. Todo o controle da esfera é

realizado através dos controles na tela do dispositivo, incluindo movimentação, pulo e outras funcionalidades não diretamente relacionadas ao jogo (Figura 21).

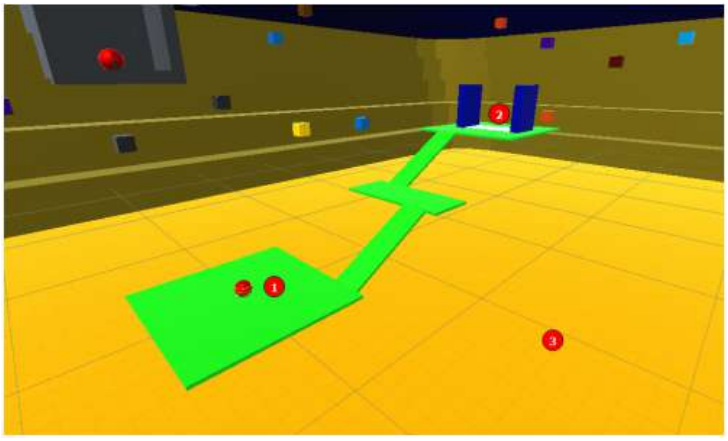
Figura 21 – Captura de tela do player jogando o primeiro nível



Fonte: Marconi (2023)

O cenário e as fases do jogo foram desenvolvidos utilizando o Blender, uma ferramenta de modelagem de objetos 3D que será abordada mais adiante neste documento. A modelagem no Blender envolveu a criação de objetos 3D utilizando formas primitivas como cubos e esferas, que foram combinadas para compor os elementos do jogo (Figura 22). No cenário, elementos como obstáculos e plataformas foram desenvolvidos com detalhes visuais simplificados, mas funcionais, para garantir um desempenho eficiente e uma estética coerente.

Figura 22 – Imagem do primeiro nível



Fonte: Marconi (2023)

Podemos notar a similaridade deste trabalho com o trabalho deste documento por conta de sua proposta de pesquisar e compreender o processo de desenvolvimento de um jogo 3D dentro da Unity. O trabalho também aborda brevemente a história dos jogos, que é um dos pontos principais neste documento. Além disso, o trabalho foca em explicar pontos relevantes do desenvolvimento do BallRide, algo que também será muito relevante mais adiante dentro deste documento para demonstrar o processo de construção de nosso jogo dentro da Unity.

3 MATERIAIS E MÉTODOS

Neste capítulo, apresentaremos todas as ferramentas e metodologias utilizadas no desenvolvimento do nosso jogo, incluindo um detalhamento maior da própria Unity. Descreveremos cada aspecto com a quantidade de detalhes necessária para garantir a compreensão completa do processo de desenvolvimento, desde o planejamento e as ferramentas utilizadas até os recursos auxiliares.

3.1 GDD

Uma das principais ferramentas organizacionais para o desenvolvimento de jogos é o Game Design Document (GDD, Documento de Design do Jogo), um documento de texto produzido por um designer que pode registrar tudo (ou quase tudo) sobre o jogo em questão. Ele faz uma descrição de vários elementos, como estética, narrativa, mecânicas, entre outros (MOTTA, 2013). Todo esse processo é necessário para comunicar a todos os envolvidos no projeto o mesmo caminho a ser seguido no desenvolvimento do jogo.

Embora o Documento de Design do Jogo (GDD) tenha grande importância no desenvolvimento de jogos, principalmente os grandes, não existe um modelo padrão a ser seguido (HALTSONEN, 2015). Isso porque tudo depende do escopo do jogo e de quanto tempo e detalhes serão necessários para o seu desenvolvimento. Com base nesse argumento, para este projeto, nos baseamos em uma versão do GDD apropriada para o projeto de um TCC: o "Short Game Design Document" (SGDD, Documento curto de design de jogo), que é uma versão mais curta e focada em descrever de forma mais direta o jogo a ser desenvolvido.

O SGDD é uma ferramenta textual que busca descrever o jogo de forma linear, descrevendo a história, personagens, mecânicas e condições para ganhar ou perder no jogo em um texto corrido (MOTTA, 2013). Essa escolha torna mais fácil a compreensão do jogo como um todo, até mesmo para pessoas que não são da área de desenvolvimento de jogos.

3.2 Metodologia Ágil

Essa metodologia funciona da seguinte forma: realiza entregas mais rápidas e menos espaçadas, criando um ciclo de desenvolvimento ativo em que pequenas versões do projeto são criadas, possibilitando a visualização da qualidade atual do software e um vislumbre do seu resultado final (MATOS, 2022). Inspirando-se no funcionamento da metodologia ágil, estabeleceu-se a meta de focar em cada parte específica, realizar análises sobre o que estava satisfatório ou não, e, por fim, executar testes em cada etapa do processo.

Dessa forma, foi possível basear o desenvolvimento do jogo nessa metodologia, concentrando-se em uma fase de cada vez e apresentando pequenas versões de cada nível nas reuniões de orientação.

3.3 Ferramentas de desenvolvimento

- Unity: A Unity Engine é uma plataforma líder na criação de conteúdo 3D interativo em tempo real, segundo o site da própria empresa. Esta ferramenta poderosa, conhecida como game engine (motor de jogo), foi criada pela Unity Technologies (UNITY, 2024c). Apesar de ser popular em diversos ramos diferentes, seu principal mercado é o desenvolvimento de jogos 2D e 3D. A Unity é uma plataforma muito ampla, capaz de desenvolver para consoles como Xbox e PlayStation, além de computadores e dispositivos móveis. Outro fator que torna esta engine muito popular é a liberdade que oferece aos desenvolvedores, permitindo a programação de scripts em C# o trabalho com diversos componentes dentro da engine, a criação ou utilização de plugins e a adição de diversos outros recursos externos. Neste projeto, a Unity Engine é a base para a construção do nosso jogo, onde utilizaremos uma grande quantidade de ferramentas que estão dentro e fora da plataforma.

- Visual Studio (2022): Visual Studio é um ambiente de desenvolvimento integrado (IDE) robusto, disponível na versão desktop, que permite aos desenvolvedores trabalharem na criação de seus projetos. O Visual Studio oferece uma ampla gama de ferramentas para codificação, depuração, teste e implantação de aplicativos. Entre seus recursos estão suporte para diversas linguagens de programação, integração com sistemas de controle de versão, edição de código com autocompletar e refatoração, design visual de interfaces, além de extensões criadas pela comunidade que podem ser integradas na plataforma (MICROSOFT, 2024). Neste projeto, o Visual Studio foi essencial para o desenvolvimento e depuração do código, bem como para a integração e implantação dos scripts dentro da Unity Engine.

- Blender: Blender é um software de criação 3D gratuito e de código aberto, licenciado sob a GNU GPL. Esta é uma das melhores e mais desenvolvidas ferramentas de criação 3D, que atende a todas as necessidades do 3D: texturização, animação, modelagem, rigging, simulação, renderização, composição e muitas outras atividades que podem ser realizadas com ele (BLENDER, 2024a). O Blender é amplamente utilizado no mercado de desenvolvimento de games por ser gratuito, versátil e de alta qualidade. No nosso projeto, o Blender será o responsável pela modelagem 3D de alguns personagens, animação e criação de imagens através de composição.

- Adobe Photoshop: Adobe Photoshop é um software líder em edição de imagens e design gráfico, amplamente reconhecido por sua versatilidade e capacidade de

criar gráficos de alta qualidade (ADOBE, 2024). Este software poderoso, lançado pela Adobe Systems, tem sido uma ferramenta essencial para designers, fotógrafos e artistas digitais desde sua criação. Seu principal mercado inclui a edição de fotografias, criação de ilustrações digitais e design gráfico para impressão e web. Photoshop é uma plataforma extremamente abrangente, oferecendo uma vasta gama de ferramentas para manipulação de imagens, como camadas, máscaras, filtros e ajustes de cor. Além disso, o software permite a criação de gráficos vetoriais e bitmap, o que o torna uma escolha ideal para uma ampla variedade de projetos de design.

- Pixilart: Pixilart é uma plataforma focada na criação, compartilhamento e compra de arte pixel, oferecendo tanto uma versão gratuita quanto uma paga (PIXILART, 2024). Pixel arts são artes totalmente baseadas em pixels, e a plataforma disponibiliza ferramentas específicas que permitem aos usuários desenhar, editar e animar suas criações pixel a pixel. O site é ideal para artistas digitais, tanto iniciantes quanto experientes, que desejam criar arte em estilo retro, reminiscentes dos gráficos dos jogos clássicos dos anos 80 e 90. Apesar de breve, o uso desse site dentro deste projeto foi relevante para a criação da interface do jogador de uma das fases do jogo.

- Figma: Figma é uma ferramenta de design poderosa, disponível tanto em versão web quanto em programa para desktop, que permite aos usuários trabalharem em conjunto na criação de seus designs. O Figma oferece diversas formas de trabalhar seus designs, com ferramentas para criar e modificar elementos, alinhamento de elementos, padronização de estilos, trabalho com camadas, vetores, personalização de fontes e diversos recursos criados pela comunidade e disponibilizados dentro da plataforma (FIGMA, 2024). Neste projeto, o Figma foi essencial para a criação e prototipação de elementos de interface, bem como a criação dos protótipos dos menus do jogo.

- Mixamo: Esta é uma incrível aplicação online desenvolvida pela Adobe. Com uma enorme biblioteca de animações pré-prontas, essa aplicação é capaz de animar qualquer objeto 3D se configurado da forma correta. Além disso, também realiza o processo de "rigging", que será explicado em um tópico futuro. Mixamo é uma das plataformas mais populares para animar personagens de forma fácil e prática, já que é gratuita, muito intuitiva e possui a capacidade de exportar personagens animados para uso em filmes, jogos e outras aplicações (MIXAMO, 2024).

- GitHub: Esta é uma plataforma que utiliza armazenamento em nuvem para que as pessoas possam criar códigos, armazenar, compartilhar e trabalhar em conjunto. Além disso, uma das principais utilidades do GitHub é ser uma plataforma de versionamento de projetos. Para isso, ele trabalha em conjunto com outro sistema, o Git. O Git é um sistema de controle de versão inteligente que detecta as mudanças

realizadas nos arquivos, permitindo que você crie variações do mesmo arquivo, salve-as no seu computador e, em seguida, envie para a plataforma de sua preferência, que, no nosso caso, é o GitHub (GITHUB, 2024). A principal função do GitHub neste projeto é o armazenamento e versionamento de todo o jogo, para que, caso existam problemas, seja possível recuperar facilmente versões anteriores que sejam funcionais.

- Trello: Trello é uma popular ferramenta de gerenciamento de projetos baseada na web que permite aos usuários organizar tarefas e colaborar em projetos de maneira visual e intuitiva. Desenvolvido pela Atlassian, Trello utiliza um sistema de cartões e quadros que facilita o acompanhamento do progresso das tarefas, tornando-o uma escolha ideal para equipes que buscam uma abordagem simplificada para a gestão de projetos (TRELLO, 2024). Para este projeto, o Trello foi relevante no papel de organização do desenvolvimento do jogo, ajudando a definir o que deveria ser feito e em qual ordem.

3.4 Técnicas de Modelagem, Texturização e Composição

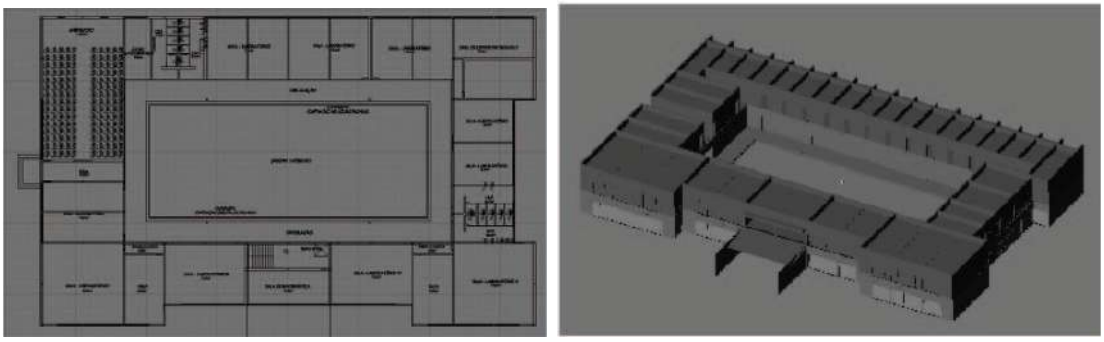
Segundo (MUSSE, 2017), "modelagem (em Computação Gráfica) consiste em todo o processo de descrever um modelo, objeto ou cena, de forma que se possa desenhá-lo". Isso é possível utilizando alguns softwares voltados à modelagem, como 3DS Max, Maya, Blender, entre outros. O desenvolvimento de modelos tridimensionais é feito através de um conjunto de técnicas de modelagem, como extrusão, imagem de referência e outras. Além disso, o uso de texturização e composição, técnicas que serão apresentadas e explicadas nesta seção.

3.4.1 Modelagem Arquitetônica

A técnica de modelagem arquitetônica consiste em criar modelos de edifícios, residências, salas, entre outros tipos de estruturas, a partir de plantas elaboradas em 2D por softwares de engenharia, como o AutoCAD (SILVA, 2023).

Esta é uma técnica muito popular no ramo da construção e projetos de engenharia, pois permite a visualização prévia do modelo em um ambiente virtual tridimensional, facilitando a detecção de problemas e o planejamento completo da obra. Com o uso de uma planta baixa, é possível criar um modelo 3D detalhado, incluindo portas, paredes e simulações de iluminação (Figura 23).

Figura 23 – Exemplo de modelagem arquitetônica



Fonte: Reis (2013)

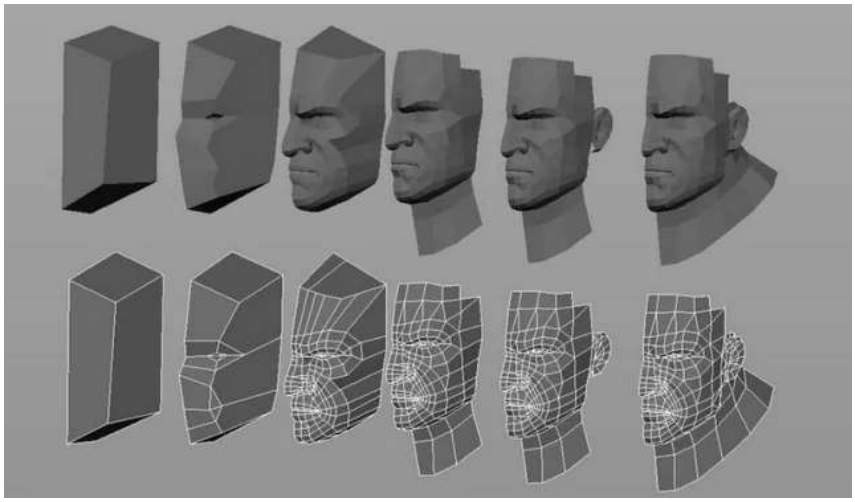
No contexto do desenvolvimento do jogo deste projeto, esta técnica é importante, pois foi com base nela que dois dos cenários utilizados foram criados. Foi utilizado o processo de desenho da planta baixa e a modelagem 3D baseado no desenho 2D criado.

3.4.2 Box Modelling

O processo de modelagem de personagens e objetos envolve o uso de diversas técnicas para alcançar o resultado desejado de forma eficiente. Dentre essas técnicas, algumas são mais comumente utilizadas e ensinadas devido à sua praticidade, como o box modeling. Esta técnica envolve o uso de formas primitivas (cubo, esfera, cilindro) como base para a modelagem do modelo final.

Esta técnica de modelagem, assim como outras, requer o uso de um software que suporte modelagem 3D. O processo, realizado com essa técnica, envolve a adição de detalhes e modificações na forma primitiva escolhida, que na maioria das vezes é um cubo. Isso inclui o uso de diversas ferramentas básicas de softwares de modelagem, como extrusão, conexão de vértices e criação de ciclos de arestas ao redor do objeto. Assim, utilizando as ferramentas do software, o modelo vai se aproximando gradualmente da forma desejada (Figura 24) (ARSALAN, 2023).

Figura 24 – Exemplo de modelagem com box modelling

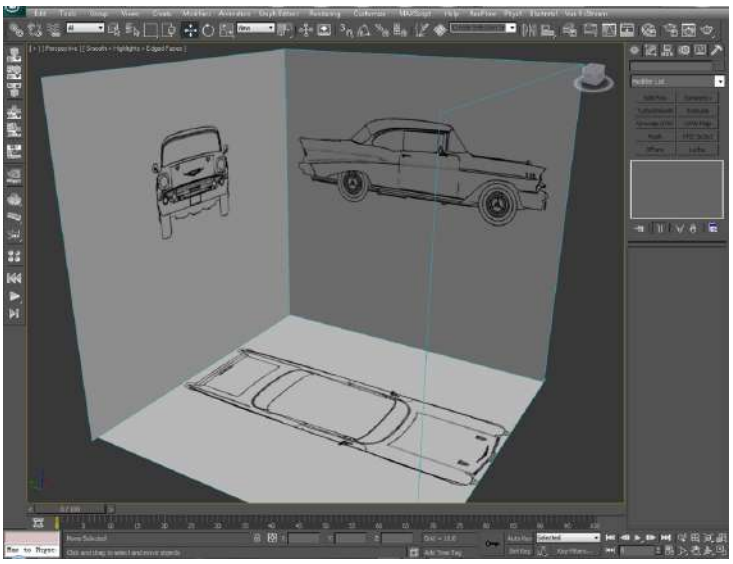


Fonte: Arsalan (2023)

3.4.3 Imagem de Referência

A técnica de uso de imagens de referência envolve a utilização de fotografias, desenhos, esboços, entre outros, como guia visual para a criação de modelos 3D. Essas imagens são fundamentais para criar um modelo base que é posteriormente ajustado e refinado, resultando em uma representação precisa do objeto. Este método é amplamente empregado nas indústrias de desenvolvimento de jogos, filmes e arquitetura, especialmente na criação de ambientes 3D que necessitam de uma reprodução fiel de objetos ou personagens (SILVA, 2023). Além disso, essas imagens podem ser posicionadas diretamente dentro do software, facilitando o seguimento preciso da modelagem (Figura 25), ou usadas fora do software, servindo apenas como guia para a modelagem.

Figura 25 – Exemplo de posicionamento de imagens de referencia dentro do 3D max 2010



Fonte: Leles (2010)

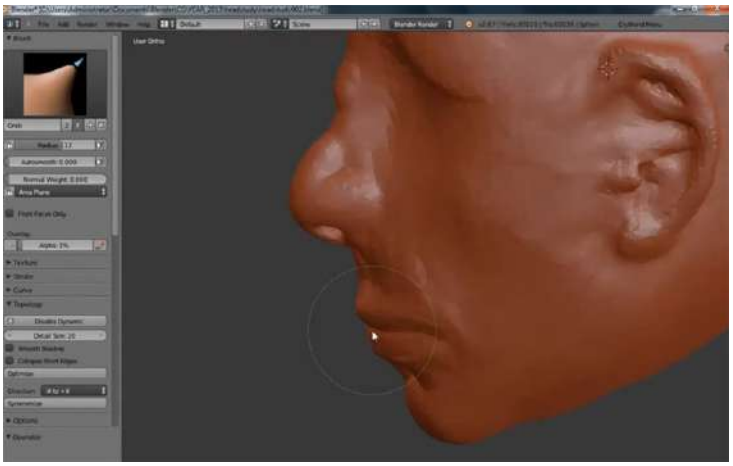
Esta técnica foi utilizada na modelagem de alguns personagens que foram utilizados dentro de todos os níveis do jogo em conjunto de algumas outras técnicas. A imagem de referência foi aplicada tanto dentro do software, como linha guia de referência fora do mesmo como forma de experimentação.

3.4.4 Escultura

A escultura digital 3D, ou escultura digital, é um método no qual um artista cria um objeto tridimensional usando software que simula a manipulação de argila digital. Esses programas oferecem diversas ferramentas e pincéis que permitem ao artista empurrar, puxar, beliscar e suavizar a geometria do modelo, criando esculturas detalhadas que replicam texturas e formas da vida real. O artista pode começar sua criação a partir de um modelo base ou do zero, utilizando cálculos complexos para gerar malhas poligonais detalhadas que se comportam como argila real. O tempo necessário para a finalização da escultura depende da complexidade do modelo e da habilidade do artista.

O processo de escultura digital é realizado em várias camadas, de forma similar à escultura tradicional com argila. Os artistas iniciam com um modelo básico ou uma forma simples, manipulando a geometria para definir características gerais, como o formato de um nariz ou a curvatura de um músculo, na etapa conhecida como “blocking”. Conforme o artista aprimora a forma básica, a geometria é subdividida para adicionar mais detalhes. Cada subdivisão aumenta a demanda de processamento do computador, tornando o projeto mais lento. Assim, a escultura digital avança camada por camada, permitindo a adição de detalhes cada vez mais refinados, como imperfeições na pele e texturas realistas (Figura 26). Na subdivisão final, com o nível mais alto de detalhes, o artista pode adicionar texturas menores, como poros, utilizando pincéis personalizados para criar superfícies detalhadas e realistas que enriquecem a escultura final (HEGINBOTHAM, 2024).

Figura 26 – Exemplo de escultura de cabeça sendo realizada no Blender 2.67



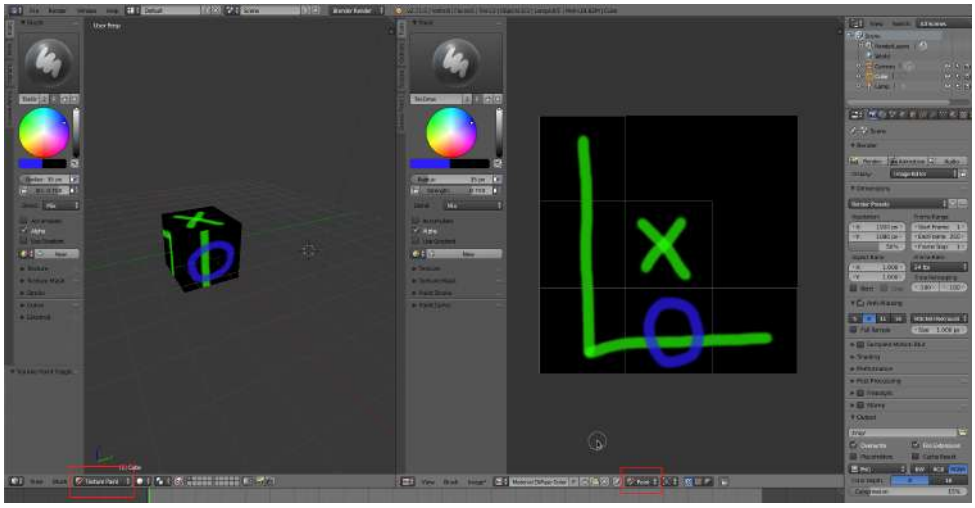
Fonte: Heginbotham (2024)

A escultura foi uma técnica pouco utilizada no nosso projeto, já que a maioria dos modelos não exigia uma alta quantidade de polígonos. Ela foi utilizada apenas em um personagem para o nível 2 do jogo.

3.4.5 Pintura de Textura e Materiais

Estas foram as técnicas utilizadas para colorir os objetos 3D modelados para o jogo dentro do Blender. A pintura de textura é uma forma de realizar pinturas manualmente com um pincel digital na malha de um modelo 3D no Blender através de um mapa UV. O mapa UV é uma representação bidimensional da malha 3D, necessária para a correta aplicação das texturas. Sem ele, a pintura de textura não seria possível. Assim, a partir desse mapa UV, o artista tem a opção de pintar diretamente o mapa UV 2D dentro do próprio Blender, pintar diretamente o modelo através da visualização 3D do software ou importar uma imagem externa e aplicar no mapa UV, sendo possível visualizar nas janelas do software a pintura realizada (Figura 27).

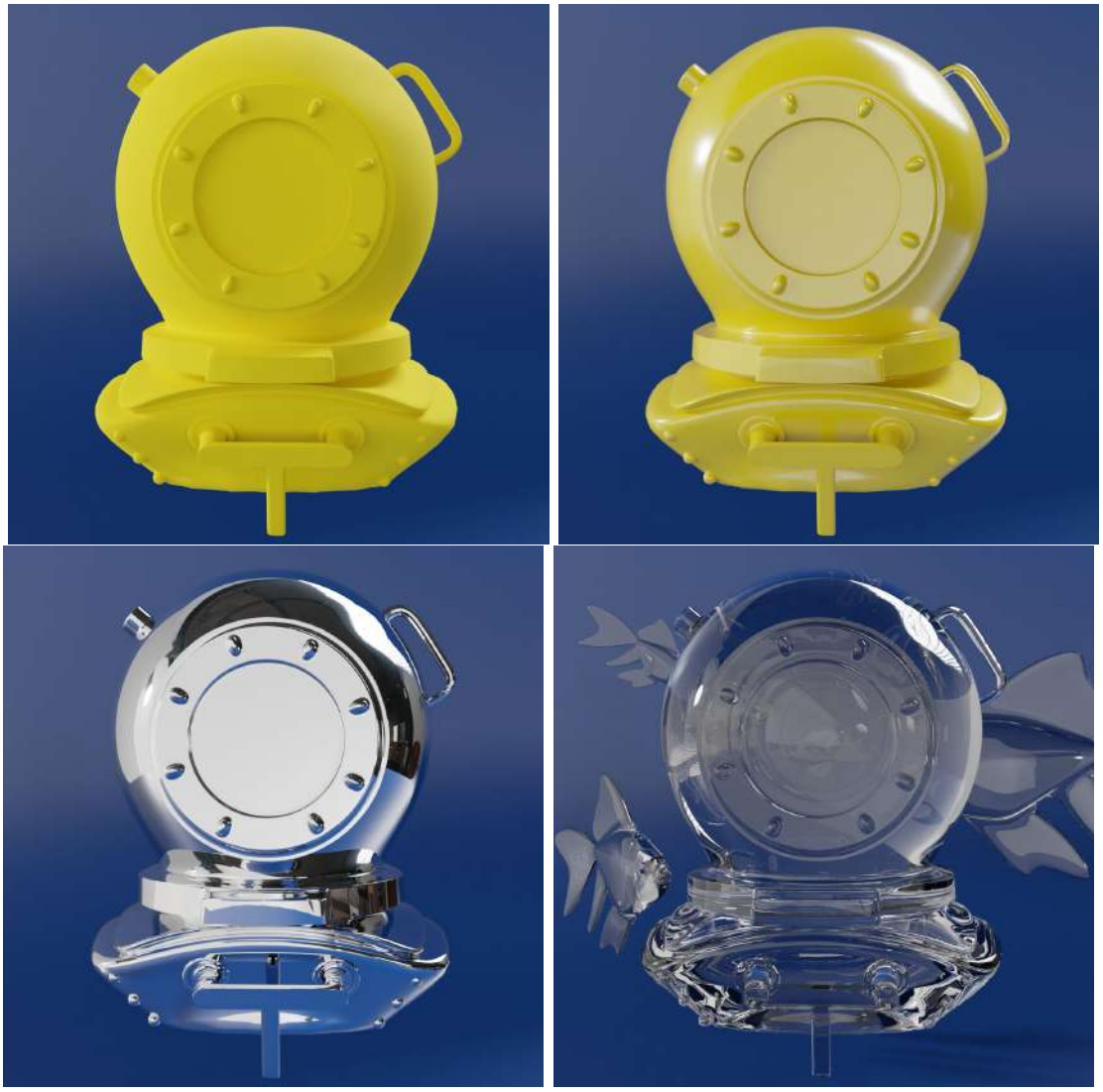
Figura 27 – Exemplo de pintura de textura em um cubo utilizando Blender 2.71



Fonte: Gonçalves (2014)

Já os materiais são basicamente elementos aplicados em uma camada sobre o objeto 3D. Os materiais contêm informações que têm um grande impacto em como o modelo final será visualizado, como, por exemplo, cores, brilho, reflexo, transparência, entre outras características (Figura 28) (SANTOS, 2022). Estes materiais podem ser aplicados a faces de uma malha de forma individual ou até mesmo em toda a malha.

Figura 28 – Materiais com diferentes características: Cor, Brilho, Reflexo e Transparência



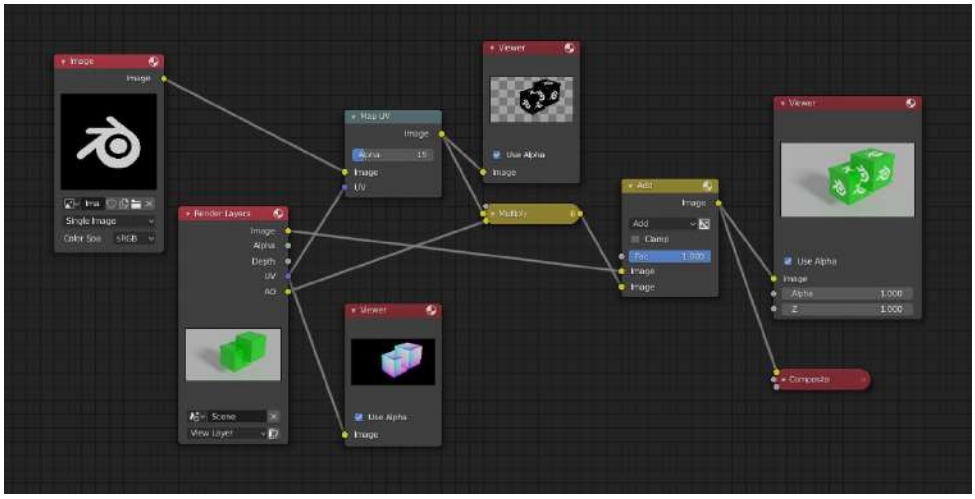
Fonte: Santos (2022)

Na maioria dos modelos do projeto, foram utilizados apenas materiais com cores e configurações específicas, pois a maioria dos modelos se utilizava de cores simples, sem grandes detalhamentos ou texturas.

3.4.6 Composição

Composição é um processo que permite aos usuários montar e melhorar uma imagem (ou filme). Aqui, utilizamos o Blender para realizar esse processo. Utilizando os nós de composição, por exemplo, você pode colar duas imagens ou filmagens e colorizar toda a sequência em um único passo (Figura 29). Desta maneira, você pode usar os nós de composição tanto para montar clipes de filme quanto para melhorá-los (BLENDER, 2024b).

Figura 29 – Exemplo de Nós de Composição de uma Imagem Com um Modelo 3D



Fonte: Blender (2024b)

Nesse contexto, você pode aplicar filtros, ajustar cores e criar efeitos visuais diretamente no Blender, sem a necessidade de usar programas externos (Figura 30). Esses efeitos podem ser úteis para diversos usos.

Figura 30 – Exemplo de correção de cores utilizando Composição

Fonte: Blender (2024b)

Dentro do projeto, o processo de composição foi útil para a criação de efeitos de pixelização em modelos 3D e, posteriormente, para a exportação desses modelos como imagens para serem utilizadas como sprites nos níveis 1 e 2.

3.5 Unity: Interface

A Unity, sendo a principal ferramenta de desenvolvimento deste projeto, merece um foco maior. Nesta seção, explicaremos de forma superficial a função de cada parte da interface principal da Unity Engine, utilizando como base a própria documentação do software (UNITY, 2024d). Na Figura 31 a seguir, cada letra representa um ponto da interface, que será explicado, indicando o que cada parte contém e o seu propósito. Vale ressaltar que essa organização da interface é mutável, e cada usuário pode organizá-la da forma que preferir.

Figura 31 – Interface da Unity

Fonte: Unity (2024d)

- (A) Barra de Ferramentas: Fornece acesso à sua conta Unity e aos serviços Unity Cloud. Ela também contém controles para o modo Play, onde o usuário pode iniciar, pausar e avançar frames (quadros) durante a execução do projeto. Além disso, inclui o histórico de desfazer, a pesquisa na Unity, um menu de visibilidade de layers (camadas) — uma ferramenta que permite separar GameObjects(Objetos de jogo) em suas cenas para diversos propósitos — e o menu de layout do Editor.

- (B) Janela de Hierarquia: É uma representação de texto hierárquica de cada GameObject dentro da cena(item D desta lista). Cada item na cena possui uma entrada na hierarquia, portanto as duas janelas estão inerentemente vinculadas. A hierarquia revela a estrutura de como os GameObjects se ligam uns aos outros.

- (C) Visualização do Jogo: Simula como está o seu jogo corrente renderizado através de suas câmeras de cena . Ao clicar no botão Play, a simulação se inicia.

- (D) Visualização de Cena: Permite que você navegue e edite visualmente sua cena. A visualização da cena pode exibir uma perspectiva 3D ou 2D, dependendo do tipo de projeto em que você está trabalhando.

- (E) Overlays(Sobreposições): Contém as ferramentas básicas para manipular a visualização da cena e os GameObjects dentro dela. Você também pode adicionar sobreposições personalizadas para melhorar seu fluxo de trabalho.

- (F) Janela de Inspeção: Permite visualizar e editar todas as propriedades do GameObject atualmente selecionado. Como diferentes tipos de GameObjects possuem diferentes conjuntos de propriedades, o layout e o conteúdo da Janela de Inspeção muda cada vez que você seleciona um GameObject diferente.

- (G) Janela de Projeto: Exibe sua biblioteca de ativos que estão disponíveis para uso em seu projeto. Quando você importa ativos para o seu projeto, eles aparecem aqui.

- (H) Barra de Status: Fornece notificações sobre vários processos do Unity e acesso rápido a ferramentas e configurações relacionadas.

3.6 Unity: Componentes e Recursos

A Unity oferece uma ampla gama de recursos e componentes que podem ser utilizados para criar um jogo. Esses componentes se combinam dentro dos objetos, permitindo a realização de diversas funções, como iluminação, movimentação de objetos e visualização de interfaces. Dentre esses recursos e componentes, alguns são essenciais para a construção de qualquer jogo, enquanto outros são mais específicos para cada projeto desenvolvido na Unity Engine.

Durante o desenvolvimento deste jogo, diversos recursos foram utilizados para a criação do visual e da mecânica dos níveis. Abaixo, apresentamos uma lista de componentes básicos e específicos que são importantes para o desenvolvimento do jogo, além de alguns recursos fundamentais.

- **Transform**: Cada objeto em uma cena possui um componente Transform, que é usado para armazenar e manipular a posição, rotação e escala do objeto. O Transform permite que esses atributos sejam ajustados de maneira individual para cada objeto. Além disso, cada Transform pode ter um pai, o que possibilita a aplicação hierárquica de posição, rotação e escala.

- **Colliders**: Um Collider é o componente responsável pela detecção de colisões do objeto com outros objetos na cena. Os Colliders definem a forma de um GameObject (objeto de jogo) para fins de colisões físicas. Além disso, são invisíveis e não precisam ter o mesmo formato da malha do GameObject.

- **Rigidbody**: Esse componente controla a posição de um objeto através da simulação física. Adicionar um componente Rigidbody a um objeto coloca seu movimento sob

o controle do motor de física da Unity. Mesmo sem adicionar nenhum código, um objeto com Rigidbody cairá sob a ação da aceleração da gravidade e reagirá a colisões com objetos próximos, desde que o componente Collider adequado também esteja presente.

- **Mesh Renderer**: Mesh Renderer é um componente responsável por renderizar a malha do seu objeto dentro do ambiente virtual 3D da Unity e tornar tudo visível aos olhos do usuário. Ele aplica texturas e materiais ao objeto, permitindo a interação com luz, sombras e outros efeitos visuais.

- **Particle System**: O Particle System (Sistema de Partículas) simula e renderiza várias pequenas imagens ou malhas, conhecidas como partículas, para criar um efeito visual. Cada partícula no sistema representa um elemento gráfico individual dentro do efeito. O sistema simula todas as partículas de forma coletiva, gerando a impressão do efeito completo. Sistemas de partículas são muito úteis para criar efeitos dinâmicos como fogo, fumaça ou líquidos, que são difíceis de representar com malhas (3D) ou sprites (2D).

- **Visual Effect Graph**: O Visual Effect Graph é um pacote que você pode usar para criar efeitos visuais em larga escala para seu projeto Unity. Ele simula o comportamento das partículas na GPU, permitindo a simulação de muito mais partículas do que o Particle System padrão da Unity.

- **Character Controller**: O Character Controller (Controlador de Personagem) é um componente específico dentro da Unity, voltado principalmente para o controle de jogadores em terceira ou primeira pessoa, eliminando a necessidade de um Rigidbody. Existem diversas vantagens em utilizar este componente, pois ele mantém o personagem equilibrado e utiliza vários parâmetros pré-definidos, como subir degraus, ângulo máximo de escalada e outros parâmetros que podem ser facilmente ajustados na interface da Unity.

- **Cinemachine**: Cinemachine é um conjunto de módulos para o controle da câmera na Unity. Esse recurso disponível no motor resolve a complexa matemática e lógica de rastreamento de alvos, composição, transição e corte entre tomadas. Foi projetado para reduzir significativamente o número de manipulações manuais demoradas e revisões de script que ocorrem durante o desenvolvimento.

- **ProBuilder**: Este é mais um dos recursos disponíveis dentro da Unity Engine. ProBuilder é um pacote que oferece diversas ferramentas para criar, editar e texturizar geometrias diretamente na engine. Isso significa que você pode realizar modelagem 3D de forma simplificada utilizando este pacote. Algumas das suas principais utilidades incluem: construção de design de níveis, prototipação, testes de jogos e malhas de colisão.

- **Animator**: Animator é um componente associado a objetos para aplicar animações na Unity. Para que esse componente funcione, ele precisa de um "Animator Controller" (controlador de animação), que é responsável por escolher quais clipes serão utilizados e como serão misturados e realizadas as transições entre eles.

- **MonoBehaviour**: Diferente dos componentes já apresentados neste trabalho, o MonoBehaviour é uma classe essencial e uma base para o uso dos scripts dentro da Unity. Ele fornece a estrutura necessária para que um script possa ser anexado a algum GameObject no Editor da Unity, além de oferecer eventos de extrema importância, como Start e Update, que serão abordados no capítulo de desenvolvimento.

- **Canvas**: O Canvas é um caso especial dentre todos os outros componentes, pois não é apenas um componente, mas sim uma área onde todos os elementos de interface devem estar. Trata-se de um GameObject Canvas com um componente Canvas atribuído. Dentro da cena, o Canvas é representado por uma área retangular que define a área total da tela que o jogador verá, facilitando o processo de posicionamento de elementos durante o desenvolvimento.

- **NavMesh**: O NavMesh é um sistema de descoberta de caminhos (pathfinding) na Unity que permite que agentes, como personagens controlados por IA, naveguem pelo ambiente do jogo de forma eficiente. Ele cria uma malha de navegação (NavMesh) que representa as superfícies navegáveis da cena. Com componentes como o NavMesh Agent, que controla o movimento dos agentes, e o NavMesh Obstacle, que define áreas a serem evitadas, o NavMesh facilita a movimentação dos agentes, encontrando o caminho mais eficiente e evitando obstáculos automaticamente.

4 PROJETO

Este capítulo visa abordar toda a idealização do jogo de forma objetiva, história, níveis, mecânicas, visual e etc.

4.1 Resumo

Este projeto será um jogo 3D desenvolvido em Unity com aspectos de jogos 3D da década de 90. Sendo ele um jogo com 3 níveis no total, cada nível sendo baseado em um jogo de visual e gênero completamente diferente do anterior, alterando os comandos, mecânicas e visuais a cada nova fase.

4.2 Objetivo

O jogador deve adentrar e vencer todas as três fases impostas pelo vilão principal do jogo, esforçando-se para chegar ao final, derrotar o chefe final e escapar dessa dimensão controlada por ele.

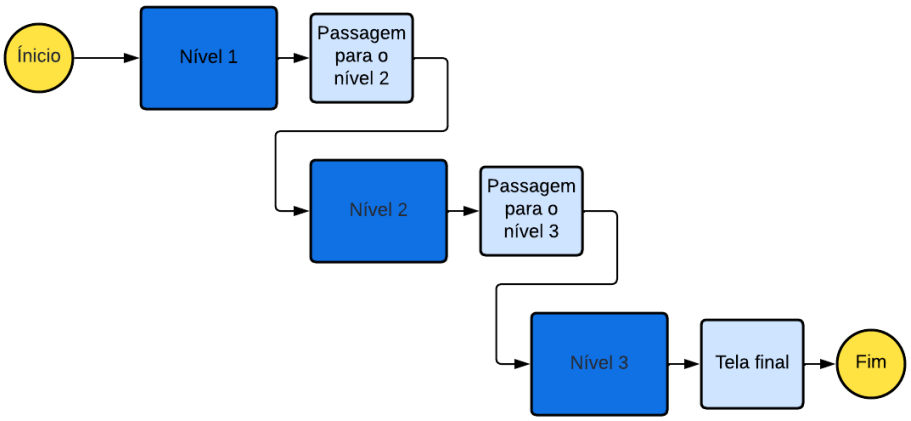
4.3 História

Um jovem rapaz chamado Relp é preso em um mundo paralelo após entrar em contato com um jogo antigo. O responsável por realizar esse sequestro é um fantasma sádico que gosta de brincar com as pessoas dentro deste mundo que ele controla. A única alternativa para fugir é derrotar o vilão dentro do jogo. Agora, preso neste mundo alternativo, o protagonista tem apenas uma opção: Vencer todos os desafios criados pelo vilão e derrotá-lo para conseguir escapar.

4.4 Visão Geral

O objetivo desta seção é abordar mais detalhadamente alguns aspectos importantes de cada nível do jogo, tanto nos conceitos visuais quanto nas mecânicas de jogo planejadas. Além disso, será fornecida uma explicação mais detalhada sobre a inspiração de cada nível, complementando a introdução geral já apresentada. No diagrama da Figura 32, é possível visualizar a progressão esperada do jogo.

Figura 32 – Representação do Fluxo de Progressão do Jogo

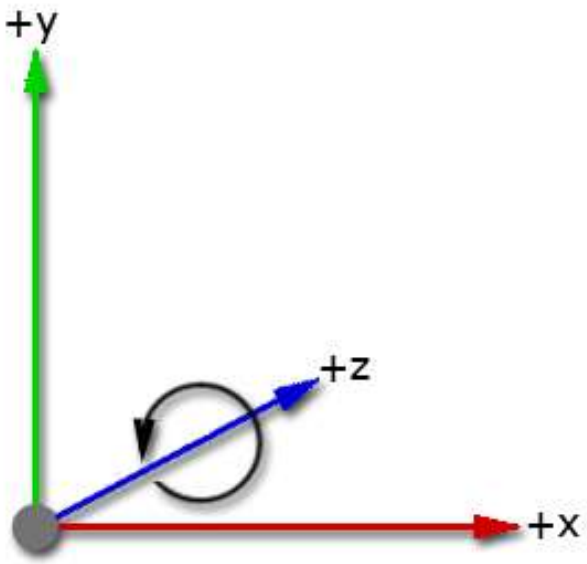


Fonte: elaborado pelo autor (2024)

4.4.1 Nível 1

Como mencionado na introdução, este nível é totalmente inspirado em Star Fox de 1993, um jogo do gênero rail shooter. O objetivo principal deste nível é a sobrevivência. O jogador é representado por uma nave espacial, que pode ser controlada para navegar no espaço ao longo da tela nos eixos X e Y, enquanto o jogo guia automaticamente o movimento da nave no eixo Z a uma velocidade constante, seguindo a orientação dos eixos mostrados na Figura 33. Com essa nave, o jogador pode se movimentar, realizar disparos e utilizar outras habilidades que serão discutidas posteriormente.

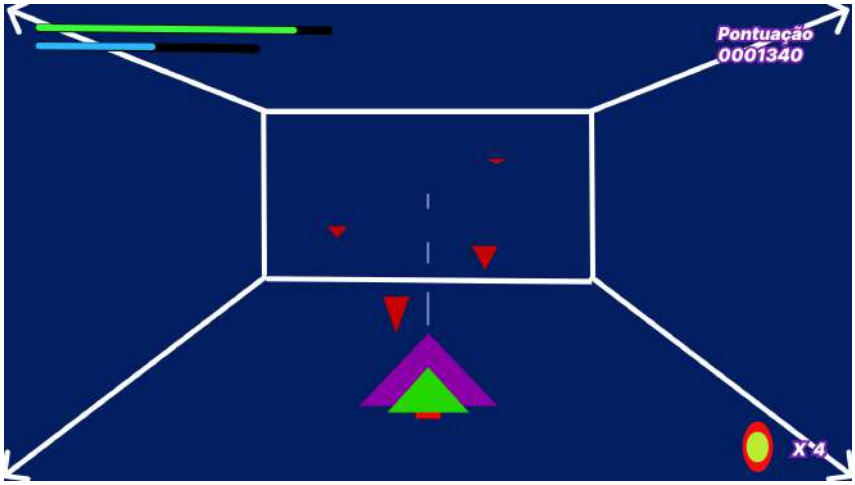
Figura 33 – Orientação padrão dos eixos X, Y e Z dentro da Unity Engine



Fonte: Unity (2024b)

Na interface, Figura 34, o jogador irá se deparar com 4 elementos , barra de vida(verde), barra de energia(azul), contador de bombas(canto inferior direito) e o contador de pontos(canto superior direito). A vida é o principal, já que se chegar a zero o jogador perde e tem que reiniciar o nível. Já a barra de energia é algo exclusivo para as habilidades de impulso e desaceleração, que consomem a energia da nave, que se recupera gradativamente e depois de esgotada só poderá ser utilizada novamente após ter se recuperado totalmente. E também temos o contador de bombas, que representa quantas bombas ainda estão disponíveis para disparo.

Figura 34 – Representação ilustrativa do nível 1



Fonte: elaborado pelo autor (2024)

Ao visualizar o portal vermelho durante a navegação espacial, o jogador perceberá que chegou ao final do nível. Ao entrar nesse portal, o nível será finalizado e o jogador será levado para a tela de transição, permitindo o avanço para o próximo nível.

4.4.1.1 Criação dos Conceitos

A criação de conceitos é uma parte fundamental do desenvolvimento de diversas mídias do entretenimento e com o desenvolvimento de jogos não é diferente. Sendo que aqui não teremos só a a criação de arte conceituação, mas também conceitos de quais mecânicas esta fase comportaria.

Os primeiros conceitos elaborados foram as mecânicas de jogo. É essencial definir no papel tanto as mecânicas que serão utilizadas quanto a jogabilidade desta fase, pois isso leva diretamente à fase de prototipação, abordada no próximo tópico, uma das primeiras etapas no desenvolvimento de um jogo, conforme discutido no artigo “The Video Game Development Essentials Guide” (TYLER, 2023).

De modo geral, optamos por recriar as mecânicas do jogador presentes em Star Fox, conforme mencionado no capítulo 04. A seguir, listamos as mecânicas e sistemas

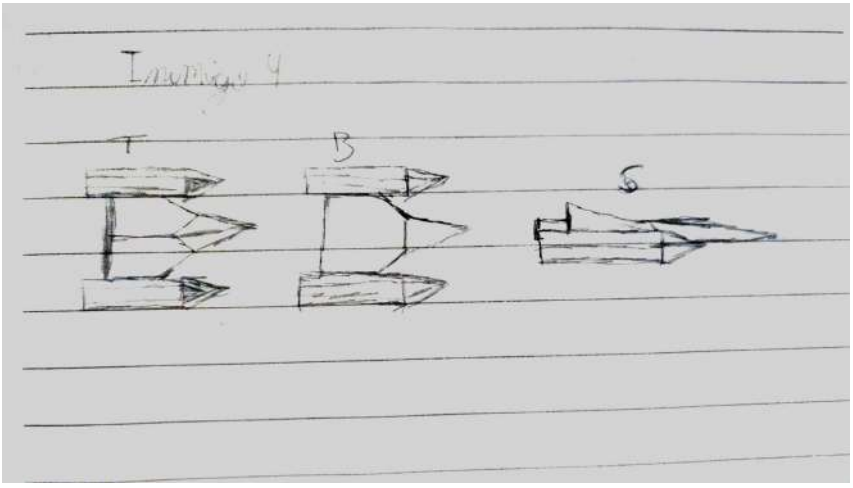
planejados para o jogador neste nível:

- Movimentação do Jogador: O jogador ter a possibilidade de se movimentar livremente na tela durante o percurso realizado no jogo utilizando as teclas **WASD**.
- Ações do Jogador: Disparos de alta frequência, disparo de bombas e rotações em volta do próprio eixo, aumento de velocidade e redução de velocidade.
- Sistemas: Barra de vida, barra de energia, contador de pontos e contador de bombas.

Os próximos conceitos a serem decididos foram os visuais dos personagens. Após um processo de pesquisa e análise do visual dos personagens e do próprio jogo Star Fox (1993), foram desenvolvidas as primeiras ideias de como os personagens deste nível deveriam ser. Esses conceitos iniciais consistiam em pequenos desenhos simples, que serviriam como base para a modelagem 3D a ser realizada no Blender.

Durante a pesquisa para criação desses desenhos iniciais uma das coisas mais observadas foi o visual geométrico de com poucos polígonos que todos os personagens do Star Fox tinham, além disso outro ponto notável na nossa análise foi as cores mais chapadas e sem textura nos modelos 3D. Sendo assim, todos os desenhos foram feitos em uma folha de papel e totalmente baseados nestas características, como é possível observar na Figura 35 e também no Apêndice A

Figura 35 – Conceitos Iniciais para os personagens do Nível 1



Fonte: elaborado pelo autor (2024)

Além desses conceitos visuais elaborados para os inimigos, também foram definidos alguns aspectos de comportamento para cada um deles:

- Inimigo 1: Nave com maior durabilidade que, ao ser destruída, libera pequenos inimigos que perseguem o jogador.

- Inimigo 2: Inimigo rápido que dispara em linha reta.
- Inimigo 3: Inimigo simples que se movimenta girando em torno de seu próprio eixo e causa dano ao colidir com o jogador.
- Inimigo 4: Inimigo simples que, ao colidir com o jogador, causa dano.

4.4.2 Nível 2

Baseado em Doom (1993), este nível é focado no gênero de tiro em primeira pessoa, onde o jogador pode explorar salas com uma arma e atirar em inimigos, conforme ilustrado na Figura 10. A principal ideia deste nível é fazer com que o jogador colete recursos distribuídos em pontos específicos do mapa para avançar e eliminar os monstros usando a arma encontrada na primeira sala.

Nesta parte do jogo, o jogador deverá gerenciar pontuações que representam diferentes status na interface: Vida, Escudo, Munição e Chaves (Figura 36). A Vida é o mais importante, pois, ao chegar a zero, o jogador perde e precisa reiniciar o nível, assim como em todos os outros níveis. O Escudo oferece proteção ao jogador, mas possui um valor máximo menor que o da Vida, sendo uma grande ajuda para se proteger dos disparos inimigos. O status de Munição indica quantos disparos o jogador ainda pode realizar; se a munição acabar, o jogador terá que encontrar mais munições no mapa ou tentar avançar sem eliminar os monstros. Por fim, o status de Chave indica quando o jogador adquiriu uma chave para abrir uma porta; ao abrir a porta, a chave é utilizada e o contador de chaves é zerado.

Figura 36 – Representação ilustrativa da interface do nível 2

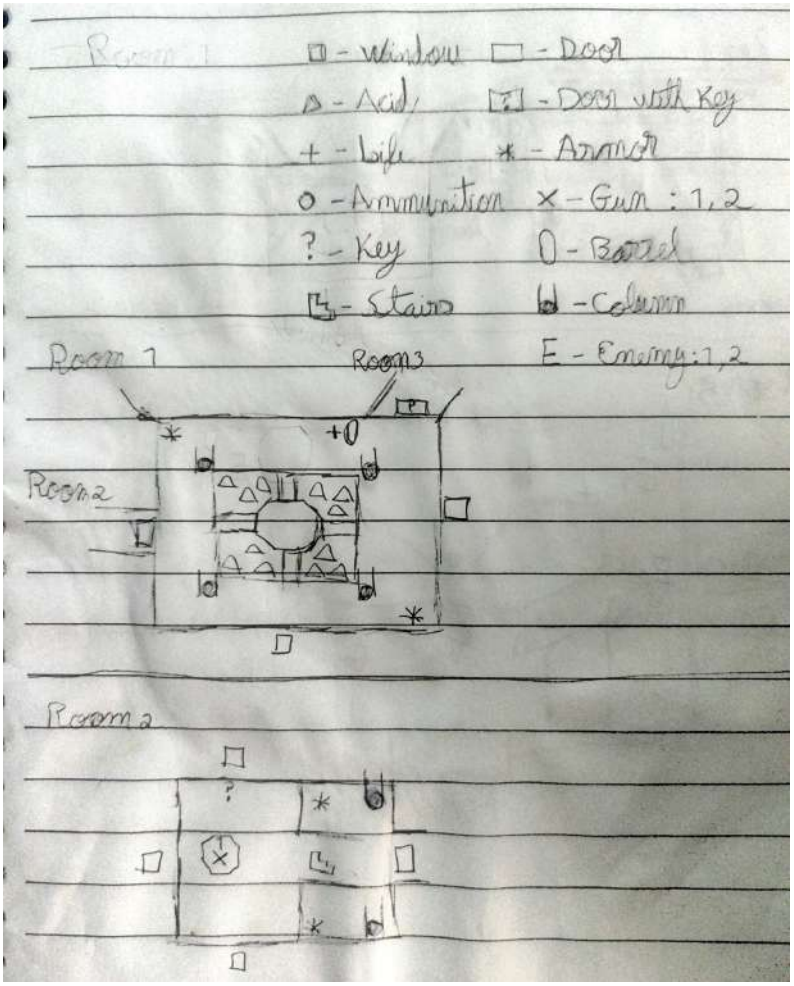
Fonte: elaborado pelo autor (2024)

Ao final do nível o jogador deve encontrar o próximo portal que faz com que ele finalize este desafio e avance para a tela de transição do terceiro e último nível deste jogo.

4.4.2.1 Criação dos Conceitos

Para este nível, a criação de conceitos foi inicialmente mais complexa, exigindo um planejamento detalhado do mapa. Diferente do nível anterior, este está situado em um local físico, onde o jogador irá progredir. Com isso em mente, o primeiro passo foi definir os elementos presentes nas salas deste nível, bem como a "planta" de cada sala que o jogador precisará atravessar (Figura 37 e Apêndice B). Cada uma das salas foi criada com o primeiro nível de Doom (1993) em mente, onde o foco é o sentimento de salas fechadas, que o jogador deve explorar e eliminar os inimigos.

Figura 37 – Planta de Mapa do Nível 2



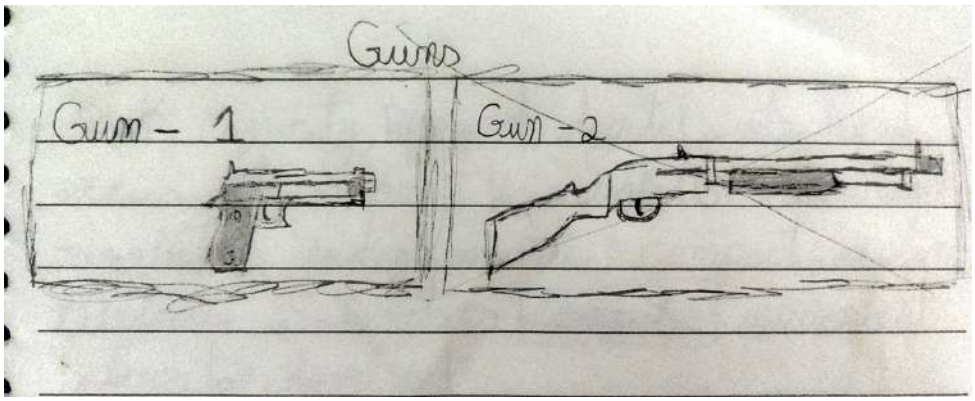
Fonte: elaborado pelo autor (2024)

Após a criação do design de todo o nível, o próximo passo foi criar os inimigos. Inicialmente, a ideia era criar dois inimigos diferentes, mas devido à complexidade do processo de modelagem, texturização e animação de personagens, decidimos manter apenas

um inimigo (Apêndice B), inspirado nos demônios do próprio Doom, que serviu como base de inspiração para este nível.

O último conceito em arte a ser definido foi o armamento do personagem. Inicialmente, planejamos ter duas armas (Figura 38), cada uma destinada a um tipo específico de inimigo. No entanto, com a decisão de manter apenas um tipo de inimigo, optamos por utilizar apenas uma arma, evitando a criação de recursos excessivos que teriam pouco uso.

Figura 38 – Ilustração conceitual das armas do nível 2



Fonte: elaborado pelo autor (2024)

No que diz respeito aos conceitos das mecânicas de jogo, o jogador tem menos opções em comparação com o nível 1, mas há mecânicas importantes relacionadas ao mapa. As seguintes mecânicas foram planejadas para o jogador:

- Movimentação: Movimentação padrão em 3 dimensões, utilizando as teclas **WASD**.
- Disparo: Disparo de arma de fogo utilizando a tecla **CTRL**.
- Abertura de Portas: Abertura de portas ao pressionar a tecla **J**, quando houver chaves no inventário.
- Barris Explosivos: Barris que explodem em área quando atingidos por disparos, causando dano a tudo que estiver próximo.
- Itens Coletáveis: Inclui recuperação de vida, aumento de escudo, aumento de munição e coleta de chaves, que constituem todos os sistemas envolvendo o jogador.

Já para o inimigo, foram definidas mecânicas de comportamento simples e de fácil compreensão:

- Sinalização: Ao enxergar o jogador, o inimigo emite um efeito sonoro.
- Movimentação: O inimigo geralmente permanece parado, mas ao estabelecer contato visual com o jogador, pode começar a se mover dependendo da distância do jogador.

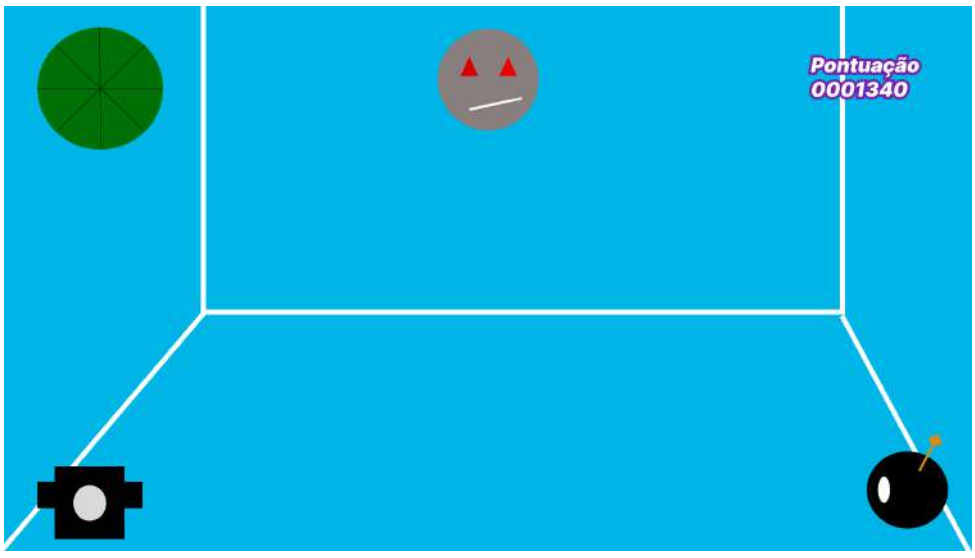
- Disparos: Quando o inimigo estabelece contato visual direto com o jogador, ele inicia disparos na direção do mesmo se estiver na distância necessária.

4.4.3 Nível 3

Baseado no Super Mario 64, este nível é focado no gênero de plataforma 3D, onde o jogador tem uma visão em terceira pessoa e utiliza a movimentação e os pulos do personagem para superar tanto os inimigos quanto o mapa (Figura 12), que é um dos pontos mais importantes de um jogo de plataforma. Além disso, este é o último nível do jogo. Ao chegar ao final, o jogador enfrentará o chefe responsável pela criação das criaturas. Para derrotá-lo, será necessário usar as próprias criaturas contra ele, enquanto se esconde da sua visão, já que, ao ver o jogador, o chefe se torna transparente e intangível, tornando-se impossível causar dano enquanto estiver nesse estado.

Na interface, o jogador poderá visualizar alguns elementos fixos na tela: o círculo de vida, a representação de pontuação(ambos na parte de cima da tela) e o modo de câmera no canto em baixo, que pode ser automático ou controlado manualmente pelo jogador. Além desses, haverá elementos situacionais, como o símbolo de bomba, indicando que o jogador está segurando um inimigo bomba, e o símbolo do rosto do chefe no topo da tela, que mudará de expressão conforme a vida dele diminui, ajudando o jogador a entender o quanto ele já sofreu de dano e quão próximo está de ser derrotado (Figura 39).

Figura 39 – Representação ilustrativa da interface do nível 3



Fonte: elaborado pelo autor (2024)

Ao final do nível, o jogador ativará o último portal do jogo. Ao entrar nele, o jogo entrará na sua tela final e após isso os créditos aparecerão na tela.

4.4.3.1 Criação dos Conceitos

Diferente dos níveis anteriores, este último nível exige a construção completa de um personagem humano, pois o jogo será em terceira pessoa, onde o jogador controlará o protagonista diretamente. Sendo assim, o primeiro passo na definição de conceitos foi elaborar o visual do protagonista para este nível. Optamos por um visual com roupas simples, um corpo pequeno e uma cabeça grande (Figura 40), inspirado na própria anatomia do Mario em "Super Mario 64".

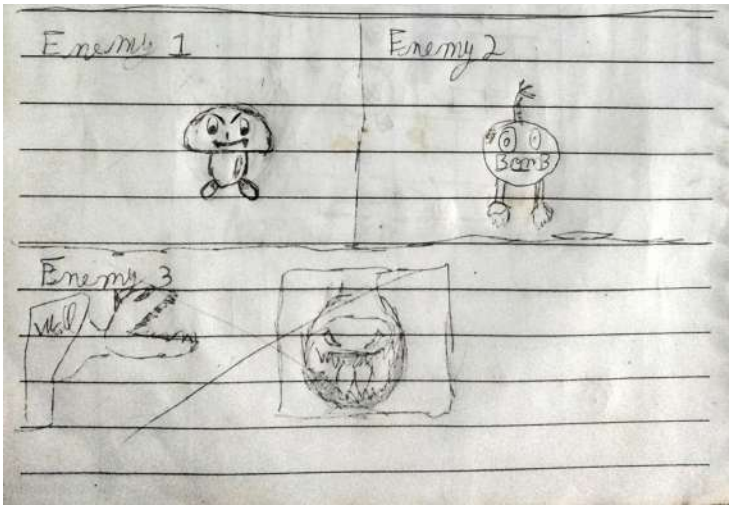
Figura 40 – Conceito Inicial do Protagonista

Fonte: elaborado pelo autor (2024)

Em relação ao mapa deste nível, assim como no nível anterior, ele desempenha um papel crucial no design. Por isso, a criação da planta do mapa foi uma das primeiras etapas a serem definidas para este nível. O planejamento da estrutura do mapa foi inspirado em alguns dos primeiros mapas de Super Mario 64, jogo que serve como referência para este nível. O mapa segue uma estrutura de escalada em espiral, conduzindo o jogador até o topo, onde estará o vilão principal do jogo, como é possível ver na planta do cenário no Apêndice C. Além disso, o mapa contará com uma ponte giratória que dará acesso à torre onde o chefe está localizado no topo. Esta torre incluirá pedras que se movem para fora e para dentro, exigindo que o jogador pule no momento certo em cada pedra para alcançar o topo.

Logo em seguida, o conceito dos personagens inimigos foi desenvolvido. Eles foram baseados em personagens populares do jogo que serviu de inspiração para este nível. Inicialmente, foram planejados três inimigos básicos, mas apenas dois foram implementados na versão final, como pode ser visto na Figura 41.

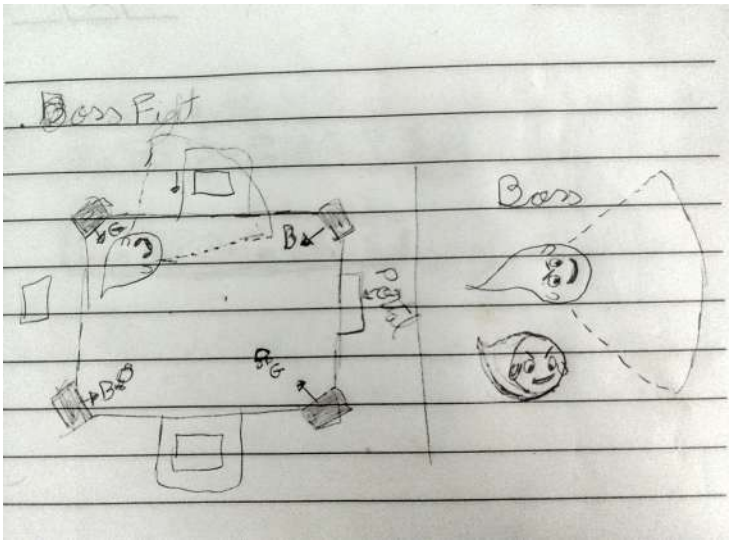
Figura 41 – Conceitos Iniciais dos Inimigos do Nível 3



Fonte: elaborado pelo autor (2024)

Por fim, o último conceito visual a ser definido foi a arena onde o chefe final do jogo se encontra, local onde o jogador poderá lutar e encontrar o portal final para sair do jogo. A luta é simples: enquanto o fantasma estiver vivo, monstros surgirão nas extremidades da arena, e o fantasma irá rondar, tentando localizar o jogador, como mostrado na Figura 42.

Figura 42 – Conceitos Iniciais Da Arena e Chefe Final do Jogo



Fonte: elaborado pelo autor (2024)

Em relação às mecânicas de jogo, o **personagem principal** deste nível oferece mais opções ao jogador em comparação com o nível anterior, permitindo as seguintes ações:

- Movimentação: O personagem pode se mover em todas as direções utilizando as teclas **WASD**.

- Pulo: Ao pressionar a barra de **ESPAÇO**, o personagem pula. Ao pressioná-la duas vezes consecutivas, o segundo pulo será mais alto que o primeiro.
- Esmagar: Ao pular sobre os "inimigos cogumelo", conseguirá esmaga-los e recuperar uma pequena quantidade de vida.
- Pegar e Arremessar: O jogador pode pegar os "inimigos bomba" e arremessá-los para causar dano em área.
- Controle de Câmera: A câmera estará em modo automático durante o jogo, mas o jogador poderá ajustá-la manualmente para a esquerda e direita utilizando duas teclas no teclado.
- Sistemas: Sistema de vida e sistema de pontuação.

Para os **inimigos comuns** deste nível, a ideia principal foi atribuir uma utilidade adicional a cada um deles. Além de serem obstáculos para o jogador, eles também podem ser úteis se o jogador utilizar suas habilidades de maneira estratégica. A seguir, são apresentadas as características de cada um:

- Inimigo Cogumelo: Este inimigo se move em ciclos e, ao avistar o jogador, irá persegui-lo. Se entrar em contato, causará dano ao jogador. O inimigo pode ser esmagado pelo jogador.
- Inimigo Bomba: Este inimigo também se move em ciclos e, ao avistar o jogador, o perseguirá até explodir, causando dano em área e arremessando o jogador a uma curta distância. O jogador pode pegar o inimigo e arremessá-lo.

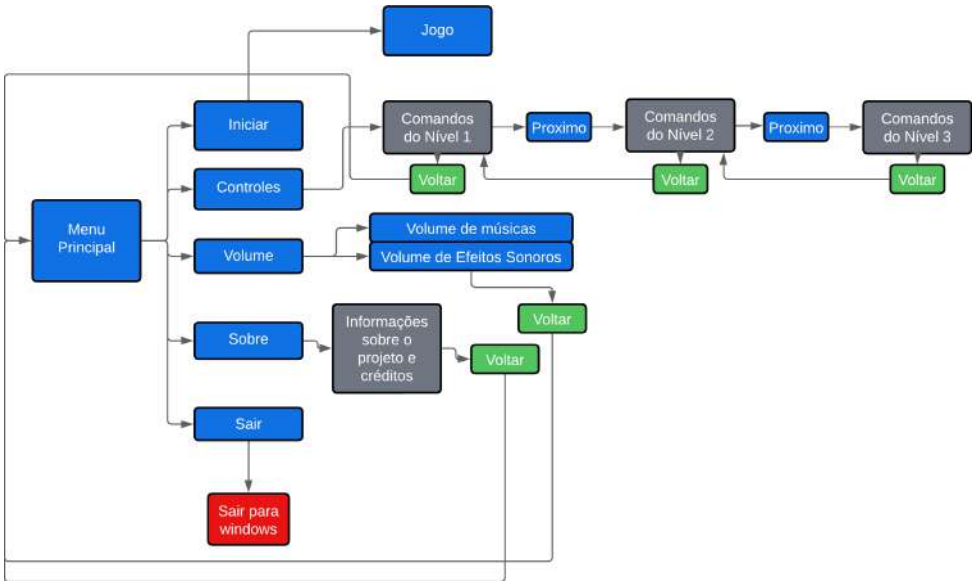
Por último, o **chefe** do jogo foi projetado para ser um desafio único. Embora ele não cause dano diretamente, o ponto central desse inimigo final é que o jogador precisa descobrir, durante a batalha, uma maneira de feri-lo. Quando o fantasma avista o jogador, ele se torna intangível, o que impede o jogador de causar dano. Nesse estado, o jogador deve se esconder ou sair do campo de visão do fantasma. Para que o fantasma se torne vulnerável, o jogador precisa encontrar uma forma de interromper a intangibilidade, criando assim uma oportunidade para atacá-lo. A única maneira de causar dano ao fantasma é usando os inimigos bomba no cenário, pegando-os e arremessando-os contra ele.

4.4.4 Menus do jogo

Os menus são uma parte crucial dos jogos, pois é por meio deles que podemos iniciar o jogo, pausar e acessar outras opções. Neste projeto, optamos por menus simples que oferecem as opções e informações relevantes para a navegação no jogo.

No menu principal, teremos algumas opções padrão, como Iniciar, Controles (explicação de controles de todos os níveis), Volume, Sobre(explicação sobre o projeto e créditos) e Sair (Figura 43). Além disso, nosso menu permitirá uma prévia dos níveis, mostrando uma visualização rápida de cada um ao passar o indicador do mouse sobre as imagens no canto superior direito da tela. Durante o jogo, haverá uma tela de pausa com opções como Volume, Controles (referentes ao nível atual) e a alternativa de voltar ao menu principal.

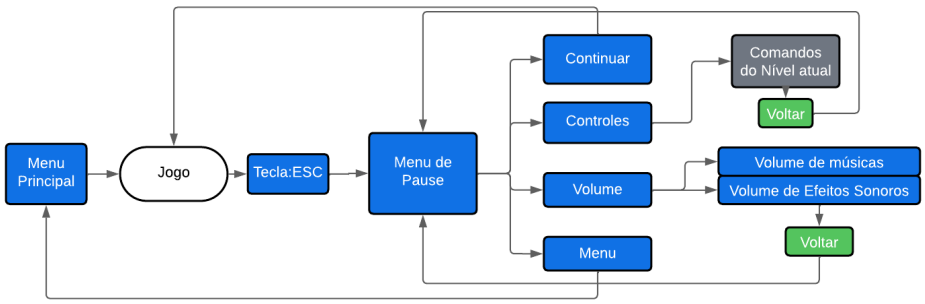
Figura 43 – Diagrama de Funcionamento do Menu Principal



Fonte: elaborado pelo autor (2024)

Além do menu principal e do menu de pausa (Figura 44), que permite ao jogador acessar opções como os comandos do nível atual, controle de volume e a opção de voltar ao menu principal, o jogo também contará com caixas de diálogo em cada nível para explicar a história, os controles e as mecânicas.

Figura 44 – Diagrama de Funcionamento Menu de Pause

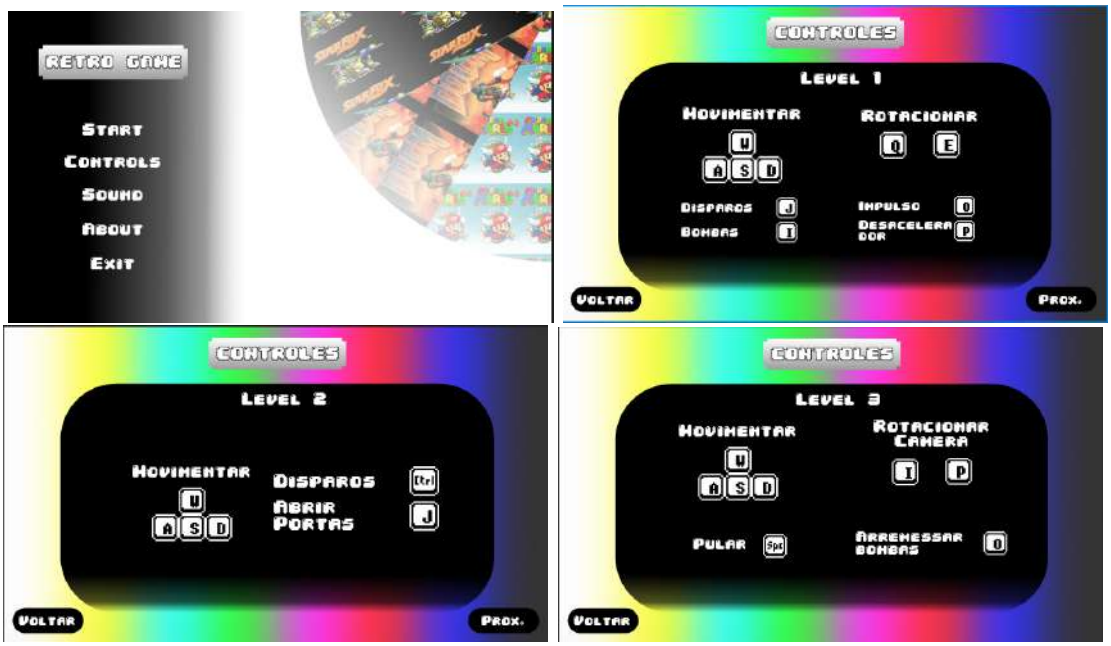


Fonte: elaborado pelo autor (2024)

4.4.4.1 Designs das Telas dos Menus

Para o desenvolvimento das telas dos menus do jogo, utilizou-se o software Figma, conforme apresentado no capítulo 3. Com ele, foram criados os visuais de todas as telas de menus que estariam presentes no jogo. Inicialmente, foram elaboradas a tela do menu principal e as telas que exibem as teclas de comando para cada nível, conforme ilustrado na Figura 45.

Figura 45 – Design de Telas no Figma: Menu e Teclas de Comandos de Cada Nível



Fonte: elaborado pelo autor (2024)

Em seguida, foram desenvolvidas as telas mais complexas para implementação na Unity Engine, como a tela de “Volumes”, que permite o controle individual dos volumes de músicas e efeitos sonoros, e a tela de “Sobre”, que disponibiliza informações adicionais sobre o projeto e os créditos de todos os recursos utilizados que não foram criados pelo autor. A última tela criada no Figma foi a tela de “Pause”. Essas telas podem ser visualizadas no Apêndice D.

5 IMPLEMENTAÇÃO

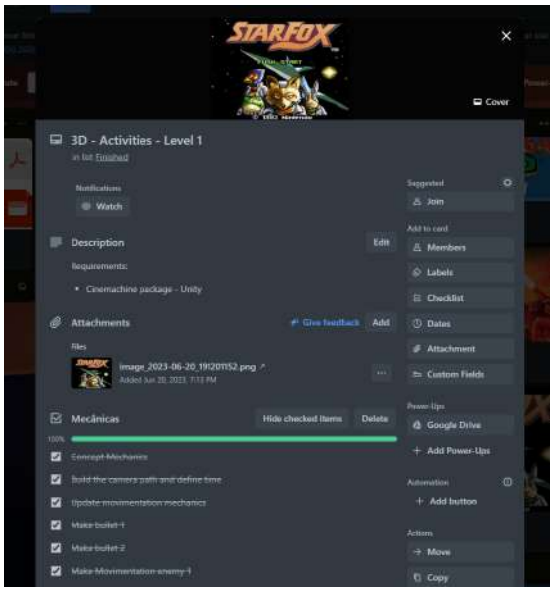
Neste capítulo, apresentaremos toda a etapa de implementação do jogo , abordando a organização de tarefas, modelagem, texturização, programação, além de outros sistemas do jogo, conforme estabelecido no projeto. As seções a seguir detalharão cada etapa do desenvolvimento, mas as dividiremos por níveis, já que cada um foi desenvolvido de maneira distinta.

5.1 Metodologia de Desenvolvimento e Trello

Com todos os pontos principais do que deveria ser desenvolvido documentados, nosso próximo passo foi organizar o desenvolvimento. Para isso, foi decidido basear o processo de desenvolvimento e entrega de etapas do jogo com base no funcionamento da "Metodologia Ágil", já apresentada anteriormente, cuja principal ideia era fazer pequenos progressos no desenvolvimento do jogo e realizar reuniões frequentes para apresentar e discutir o que já havia sido feito.

Para facilitar a visualização do processo de desenvolvimento e a entrega de cada parte, optamos por utilizar o Trello, uma excelente ferramenta para a organização de tarefas. No Trello, criamos cartões para cada nível, contendo uma lista de atividades a serem desenvolvidas para aquele nível. À medida que as tarefas eram concluídas, podíamos marcá-las como realizadas, o que nos ajudou a manter o controle do progresso (Figura 46). Essa lista funcionava para controlar o que ainda precisava ser feito, mas poderia ser alterada conforme o desenvolvimento do jogo avançasse.

Figura 46 – Cartão de Tarefas do Nível 1 no Trello



Fonte: elaborado pelo autor (2024)

5.2 Nível 1

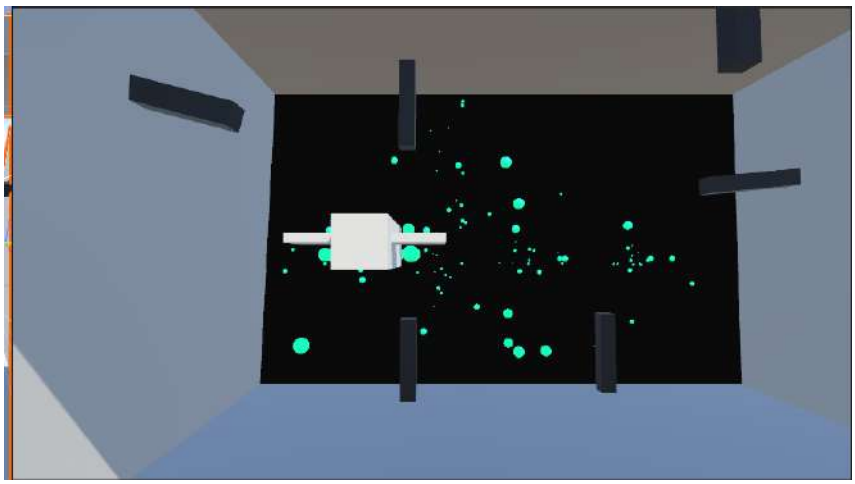
Nesta seção, abordaremos os principais pontos do processo de desenvolvimento do Nível 1 do jogo, desde a prototipação de mecânicas e modelagem até a programação.

5.2.1 Prototipação das Mecânicas do Nível

O processo de prototipação de mecânicas de um jogo é essencial para definir o que funcionará e o que não funcionará, permitindo que, a partir desses testes iniciais, sejam criados o visual e a programação final do jogo. A prototipação deste nível foi relativamente desafiadora, pois o gênero *Rail Shooter* (Tiro Sobre Trilhos) pertence a um nicho de jogos atualmente pouco explorado na área de desenvolvimento. Compreender os detalhes da jogabilidade e os aspectos de desenvolvimento de um jogo nesse gênero revelou-se mais complexo do que em outros tipos de jogos, que serão abordados nos próximos níveis.

Após alguns dias de tentativa e pesquisa para desenvolver um protótipo funcional, decidimos procurar projetos já desenvolvidos e de código aberto que pudessem nos auxiliar na compreensão do desenvolvimento de movimentação e controle de câmera dentro de uma área limitada. Durante essa busca, encontramos o projeto de (Mix and Jam, 2019) disponível no GitHub, que incluía a recriação de mecânicas do próprio Star Fox. Com base nisso, prosseguimos com a análise do código e adaptamos o que ainda não havíamos implementado em nosso projeto, resultando no primeiro protótipo das mecânicas do jogador (Figura 47).

Figura 47 – Primeiro Protótipo do Nível 1

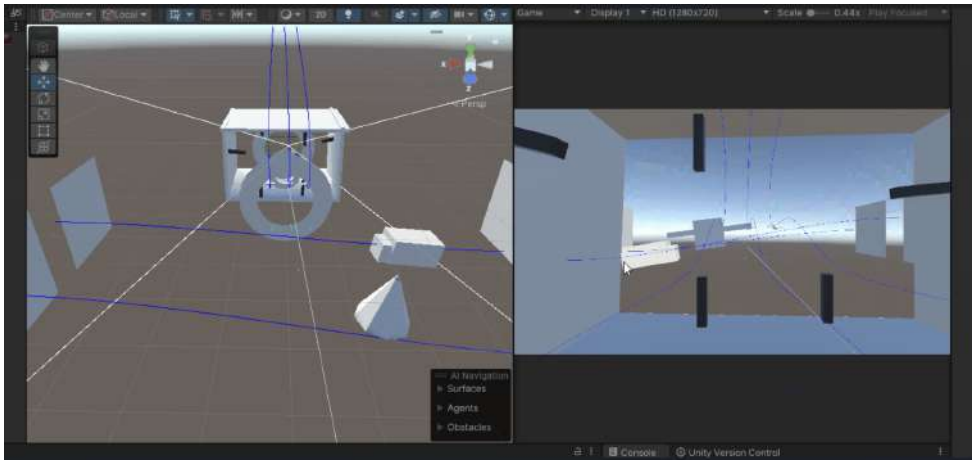


Fonte: elaborado pelo autor (2024)

Com a prototipação de movimentação jogador concluída, o próximo passo foi desenvolver a prototipação da movimentação dos inimigos e definir o comportamento de

cada um deles ao longo do trajeto do jogador. Esse processo será detalhado na subseção de programação deste nível.

Figura 48 – Primeiro Protótipo do Nível 1 com Inimigos



Fonte: elaborado pelo autor (2024)

5.2.2 Modelagem 3D

Conforme mencionado no Capítulo 4, este nível ocorre no espaço, eliminando a necessidade de modelagem de cenários complexos, exceto por um túnel inicial por onde o jogador adentra o espaço. Assim, o foco concentrou-se inteiramente na modelagem das entidades envolvidas neste nível.

5.2.2.1 Túnel de Saída Inicial

No início do nível, foi modelada uma saída simples pela qual o jogador passará por um túnel que apresenta um efeito de "sem sinal", semelhante ao exibido por alguns aparelhos de televisão antigos. Para a modelagem desse túnel, utilizou-se o recurso "ProBuilder" da Unity. Com ele, foram criados quatro cubos que foram unidos e ajustados para formar o túnel. Posteriormente, o túnel foi texturizado diretamente na Unity, utilizando uma imagem disponível em (FRANCFORT, 2024) como mostrado na Figura 49.

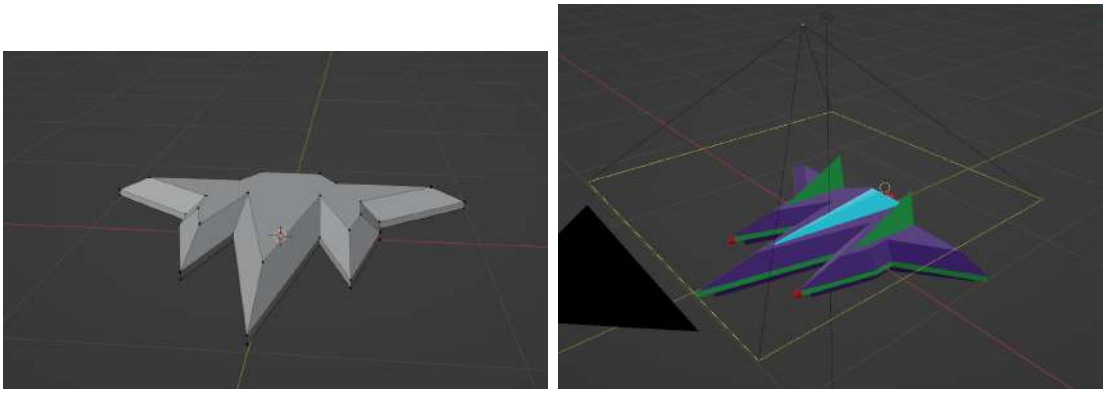
Figura 49 – Modelo 3D do Túnel

Fonte: elaborado pelo autor (2024)

5.2.2.2 Nave do Jogador

O primeiro modelo a ser criado foi a nave espacial do jogador, conforme mostra a Figura 50.

Figura 50 – Modelo 3D da Nave do Jogador



Fonte: elaborado pelo autor (2024)

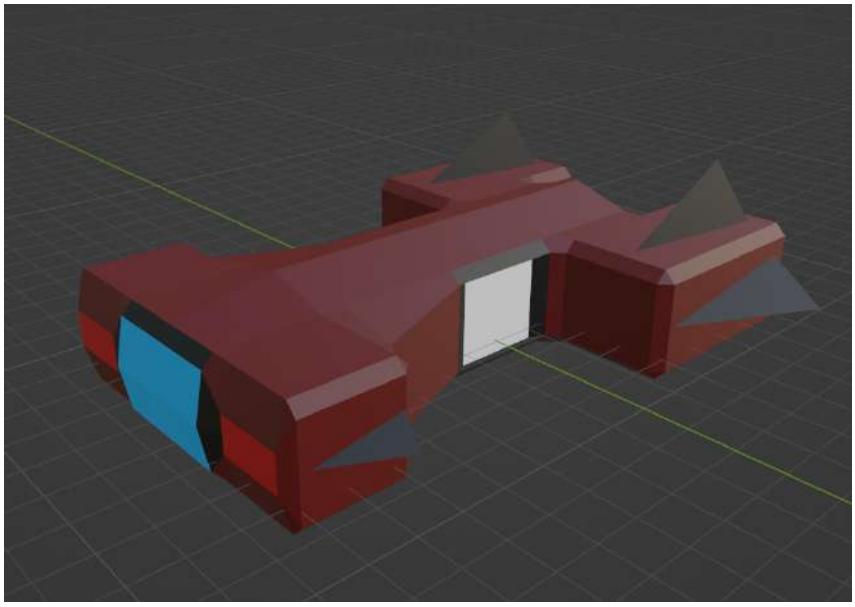
Na modelagem deste personagem, utilizou-se a técnica de *Box Modelling*, criando a nave espacial do protagonista a partir de um cubo. Foram utilizadas ferramentas básicas do Blender para gerar novos vértices e arestas, o que aumentou as possibilidades de modificação na malha do objeto, mantendo, contudo, uma baixa quantidade de polígonos para preservar a fidelidade ao visual de 3D antigo.

Para a colorização da nave, foram atribuídos quatro materiais com diferentes configurações (verde, roxo, vermelho e azul) às faces do objeto, aplicados separadamente.

5.2.2.3 Inimigos

Na modelagem do primeiro inimigo, utilizou-se a mesma estratégia de modelagem a partir de um cubo, aplicada para criar a "nave mãe". Esta nave é significativamente mais retangular e maior do que todas as outras naves inimigas, como ilustra a Figura 51.

Figura 51 – Modelo 3D do Inimigo 1 "Nave Mãe"



Fonte: elaborado pelo autor (2024)

Para a modelagem das naves "filhos", que estarão dentro deste inimigo 1, utilizamos como forma base um cone, porém removemos algumas arestas para deixar o objeto menos circular e com mais aspecto low poly (baixa quantidade de polígonos).

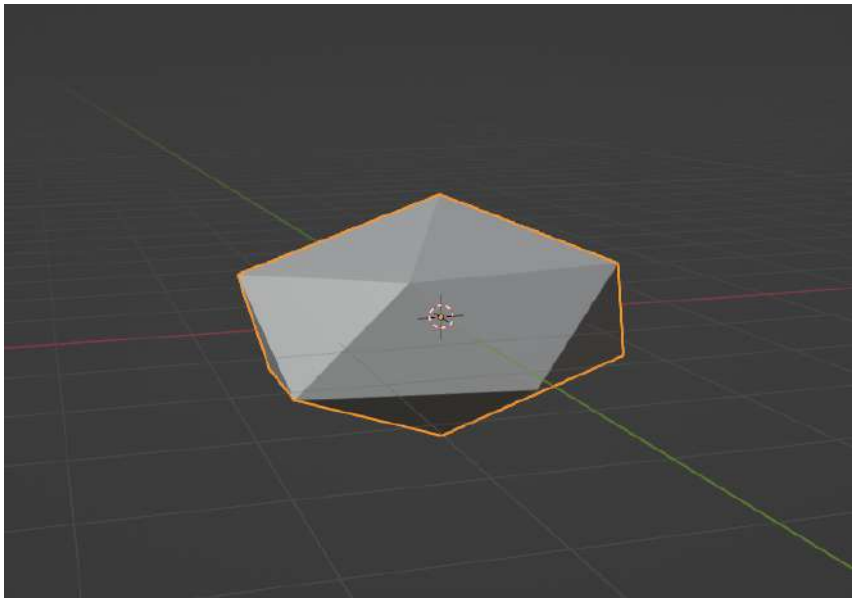
A coloração de ambos os modelos 3D foi realizada através da aplicação direta de materiais nas faces dos modelos, utilizando preto e vermelho escuro como cores base para todos os personagens inimigos deste nível.

Os modelos 3D dos demais personagens inimigos seguiram um processo de modelagem semelhante, mantendo uma coerência visual entre eles. Todos esses outros modelos podem ser visualizados no Apêndice E.

5.2.2.4 Asteroide

Além dos personagens apresentados nos conceitos iniciais do nível 1, decidiu-se também adicionar alguns asteroides no nível para aumentar a imersão no ambiente espacial e tornar a navegação um pouco mais desafiadora. Para a modelagem deste objeto, utilizamos uma "esfera icosaédrica", uma forma geométrica básica disponível no Blender, a partir da qual modificamos a quantidade de vértices e o formato para se assemelhar a um asteroide, mantendo uma baixa contagem de polígonos (Figura 52).

Figura 52 – Modelo 3D do Asteroide



Fonte: elaborado pelo autor (2024)

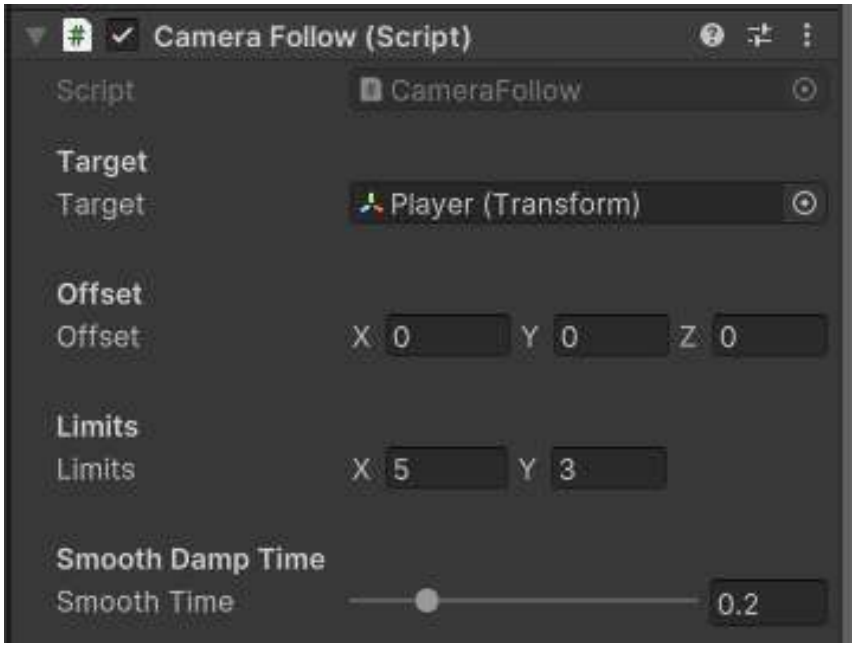
5.2.3 Programação

O foco desta subseção é abordar os principais pontos da programação e aplicação dos códigos criados em C# relacionada ao nível 1. Também incluindo a configuração e o uso de ferramentas da própria Unity Engine para criar a jogabilidade do mesmo.

5.2.3.1 Movimentação da Câmera

Um dos principais desafios ao programar este nível foi o comportamento da câmera ao longo do trajeto. Era necessário que a câmera seguisse o caminho pré-definido pelo qual o jogador passaria, mantendo, ao mesmo tempo, um limite máximo para seus movimentos em torno do ponto central. Para implementar essas regras de comportamento da câmera, utilizamos o sistema de câmera virtual do pacote *Cinemachine*. Além disso, contamos com o suporte de um dos códigos do projeto de (Mix and Jam, 2019) atribuído ao objeto da câmera virtual, que requer alguns parâmetros importantes para a definição do comportamento da câmera, como ilustrado na Figura 53.

Figura 53 – Parâmetros do Código “CameraFollow”



Fonte: elaborado pelo autor (2024)

Neste código, há quatro parâmetros relevantes a serem definidos: *Target*, que recebe a posição atual do jogador; *Offset*, que estabelece a diferença de posição entre a câmera e o jogador; *Limits*, que determina a distância máxima que a câmera pode se mover nos eixos X e Y; e *Smooth Damp Time*, que controla o tempo necessário para a câmera se ajustar ao movimento do alvo.

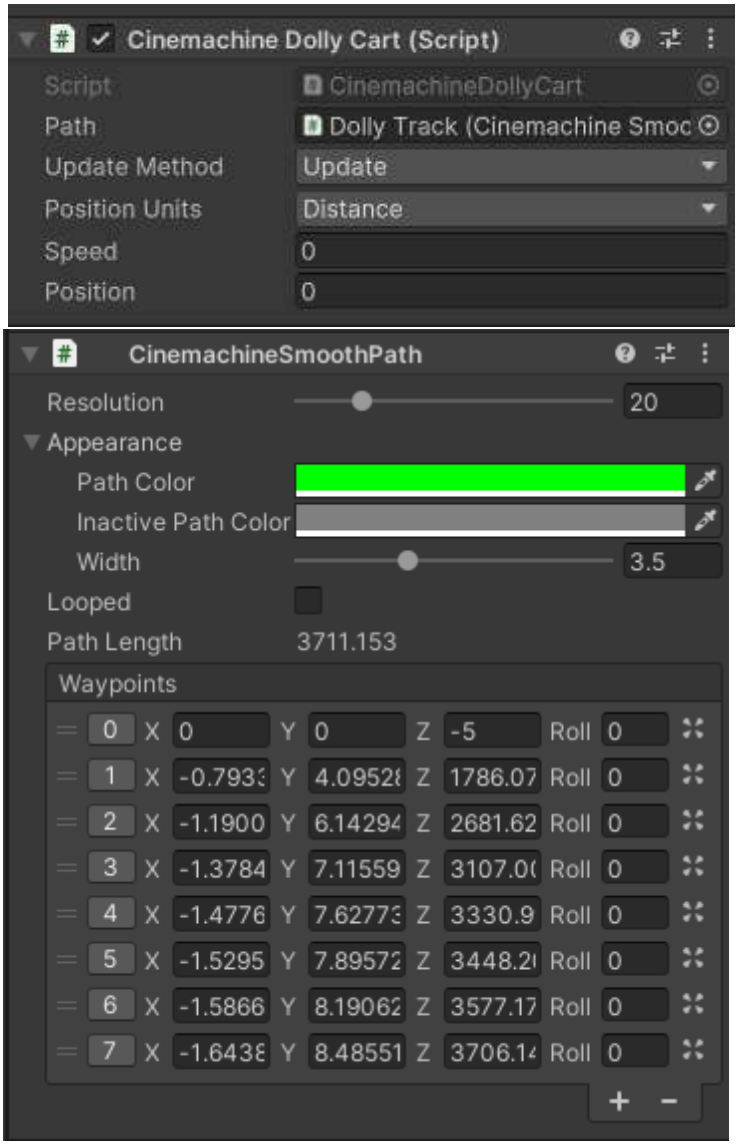
No código *CameraFollow* (Apêndice F), há três métodos principais responsáveis pelo controle de movimentação da câmera:

- **Update**: Método padrão da biblioteca da Unity, que é chamado uma vez a cada quadro (frame). Utilizamos esse método para redefinir a posição inicial da câmera quando o jogo não está em execução e, principalmente, para chamar o método FollowTarget a cada quadro.
- **LateUpdate**: Método padrão que é executado uma vez a cada quadro, mas só ocorre após todas as chamadas de *Update*. Aqui, utilizamos ele para limitar a posição da câmera dentro dos valores definidos em *limits*, evitando que a câmera saia dos limites desejados.
- **FollowTarget**: Calcula a nova posição desejada da câmera com base na posição do alvo, aplicando o *offset* e a suavização.

5.2.3.2 Movimentação do Jogador

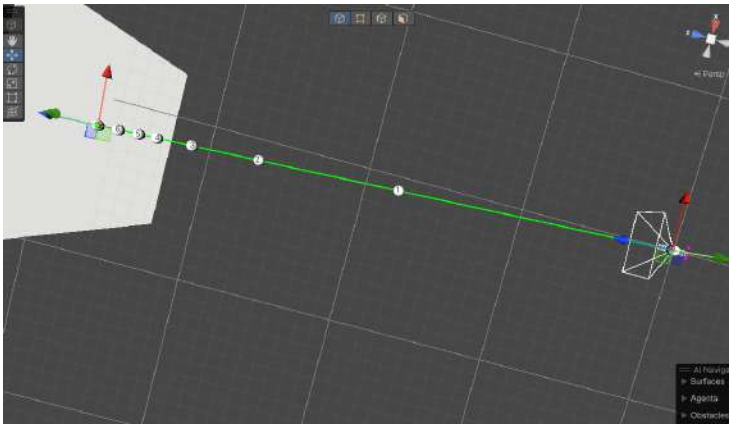
O primeiro ponto a ser definido sobre a movimentação do jogador foi como estabelecer uma rota pela qual ele seria guiado, acompanhado pela câmera. Para isso, utilizamos o componente *Cinemachine Dolly Cart*, que restringe a localização do *GameObject* a um *Cinemachine Path* ou *Cinemachine Smooth Path*. Esse componente é ideal para manter um objeto em um caminho predefinido, servindo como alvo de acompanhamento para câmeras virtuais (UNITY, 2024e). Em conjunto com o *Dolly Cart*, criamos um *Cinemachine Smooth Path* e adicionamos todos os pontos, através da janela de visualização de cena, que formam o caminho por onde o jogador navegará. Na Figura 54, é possível visualizar as definições dos parâmetros de ambos os componentes, e na Figura 55, o caminho formado na visualização de cena.

Figura 54 – Componentes Cinemachine Dolly Cart e Cinemachine Smooth Path



Fonte: elaborado pelo autor (2024)

Figura 55 – Visualização do Caminho Dentro da Cena



Fonte: elaborado pelo autor (2024)

Após a criação dos elementos essenciais, o próximo passo foi implementar a movimentação da nave do jogador, não apenas para movê-la na tela, mas também para garantir que o modelo 3D rotacionasse corretamente nas direções vertical e horizontal conforme indicado pelo jogador. Para isso, foi desenvolvido o script "PlayerMovement2", que contém todos os métodos de ações do jogador, exceto os relacionados ao disparo de projéteis. Aqui, focaremos apenas nos métodos e parâmetros dedicados à movimentação.

Os métodos utilizados para controlar os movimentos foram:

- **SetSpeed**: Define a velocidade de movimento do personagem através de uma referência ao *Cinemachine Dolly Cart* criado anteriormente, permitindo o controle direto da velocidade ao longo do caminho.
- **Update**: No método *Update* desta classe, é realizada a captura das entradas do jogador nos eixos horizontal e vertical, com a inversão do eixo vertical. Além disso, o método restringe a rotação atual para que permaneça dentro dos limites definidos pelo parâmetro *maxRotation* (rotação máxima), aplicando a rotação ajustada ao personagem.
- **LocalMove**: Move o personagem localmente com base nas entradas do jogador e no parâmetro de velocidade horizontal e vertical (*xyspeed*), considerando o tempo para manter a movimentação suave e dependente da taxa de quadros.
- **ClampPosition**: Restringe a posição do personagem dentro dos limites da tela. O limite é determinado pelo alcance da câmera; se os limites de movimentação da câmera forem ampliados, os limites da nave também se expandem.
- **RotationLookPath**: Ajusta a rotação do personagem para que ele olhe na direção desejada, usando um alvo posicionado na cena como mira para suavizar a rotação. A velocidade desse ajuste de rotação é definida pelo parâmetro *lookSpeed*.

- **HorizontalLean**: Inclina o modelo do personagem lateralmente com base nas entradas de movimento, simulando uma inclinação em curvas, o que aumenta o realismo do movimento.

5.2.3.3 Habilidades do Jogador

Após programada a movimentação básica do jogador, o próximo passo foi criar o código responsável pelas outras ações do jogador. Ainda dentro do script "PlayerMovement2", há alguns métodos importantes que são responsáveis pelas ações que envolvem a mudança no comportamento da nave, sendo elas: rotação em torno do próprio eixo, impulsionamento e desaceleração.

- **QuickSpin**: Faz a nave realizar um giro rápido no eixo Z, utilizando uma rotação local com a biblioteca *DOTween* para suavizar a animação. Esse método realiza essa rotação para ambos os lados, dependendo apenas do parâmetro *dir*, que passa o valor de 1 ou -1 para definir o lado.
- **Boost**: Controla o estado de impulso(boost) da nave, ajustando várias propriedades visuais como distorção de lente da câmera, aberração cromática, campo de visão, zoom, e a própria velocidade da nave, para tornar mais imersivo o momento em que o impulso é acionado.
- **Break**: Controla o estado de redução de velocidade(break) da nave, ajustando a velocidade e o zoom da câmera.

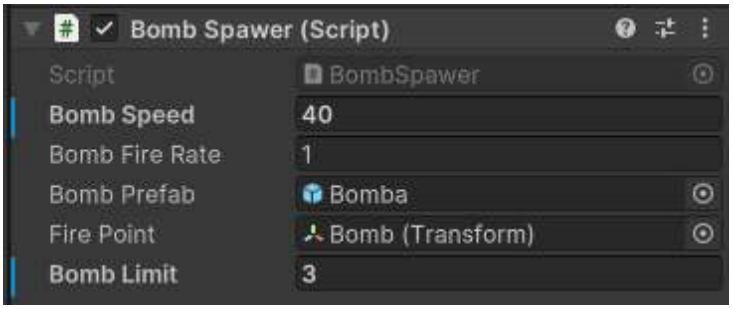
Para as outras habilidades do jogador, foi criado um script para cada uma. O primeiro script, denominado "ShootForward", contém dois métodos: um *Update*, que verifica quando o jogador clica para disparar os lasers da nave, e o método *FireLaser*, que realiza a instanciação de um Prefab (um tipo de ativo que permite armazenar um GameObject completo com componentes e propriedades) em uma variável GameObject "laser" e atribui velocidade a ele utilizando as propriedades físicas do componente *RigidBody* existente no objeto "laser" instanciado anteriormente, como é possível ver no código 5.1.

Listing 5.1 – Método "FireLaser"

```
1 void FireLaser()
2 {
3     GameObject laser = Instantiate(laserPrefab, firePoint.position,
          firePoint.rotation) as GameObject;
4     laser.GetComponent<Rigidbody>().velocity = firePoint.forward *
          laserSpeed;
5     laserSfx.Play();
6 }
```

O segundo script se chama “BombSpawer”, apresentando uma lógica semelhante à do script “ShootForward”. Ele realiza a instanciação das bombas e adiciona velocidade para o lançamento da bomba. No entanto, toda vez que uma bomba é instanciada, o contador de disparos é reduzido em 1. É possível definir a quantidade de bombas que o jogador poderá utilizar através dos parâmetros que estarão na interface do componente dentro da Unity, como é possível ver na Figura 56.

Figura 56 – Parâmetros do Script “Bomb Spawer”

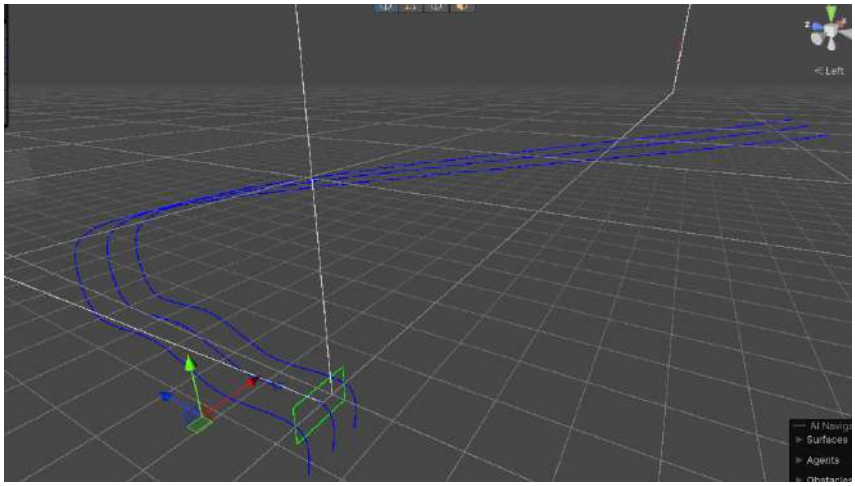


Fonte: elaborado pelo autor (2024)

5.2.3.4 Movimentação dos Inimigos

Um grande desafio para este nível, foi criar a movimentação dos inimigos. Pois ter uma movimentação mais aleatória no voo dos inimigos, seria o ideal para não tornar totalmente previsível a jogabilidade deste nível. Então através de pesquisas descobrimos que seria possível utilizar Splines para realizar esse objetivo, que é um pacote que trabalha com curvas e caminhos, pode criar comportamentos ao longo de caminhos criando trajetórias e desenhando formas (UNITY, 2024a), como é possível ver na Figura 57.

Figura 57 – Splines Criados Dentro de Cena



Fonte: elaborado pelo autor (2024)

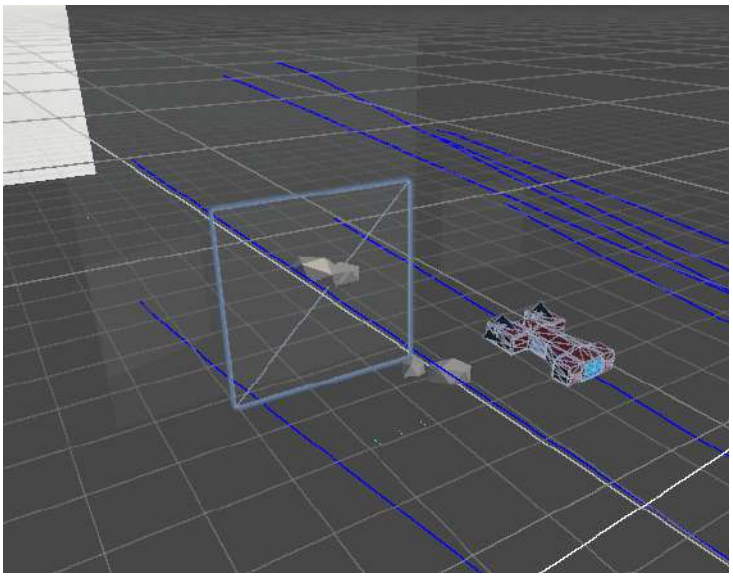
O projeto de (ADAMMYHRE, 2023) foi de extrema importância nesse estágio

do desenvolvimento, pois nos permitiu compreender que era possível gerar caminhos com *splines* de forma aleatória. Apresentamos, então, um código de uma classe estática chamada *FlightPathFactory*, responsável por gerar caminhos de voo. A classe contém dois métodos principais: *GenerateFlightPath* e *CreateFlightPath*, que trabalham juntos para criar e configurar um caminho de voo com base em pontos aleatórios gerados a partir de uma lista de quadrados fornecidos, definidos numericamente nos parâmetros do script.

- **GenerateFlightPath**: Gera um caminho de voo com base em pontos aleatórios. Cada ponto é gerado a partir de uma posição dentro de cada quadrado da lista fornecida.
- **CreateFlightPath**: Cria uma spline vazia e define sua forma usando os pontos já gerados em cada quadrado, retornando uma spline configurada ao final do processo.

Em nosso projeto, utilizamos apenas dois quadrados grandes para cada gerador de spline. Assim, um ponto aleatório era criado em cada quadrado, gerando splines mais lineares, mas com grandes descidas e subidas, dependendo da geração conforme mostra a Figura 58.

Figura 58 – Imagens de Splines em Cena: Inicio e Fim do Trajeto



Fonte: elaborado pelo autor (2024)

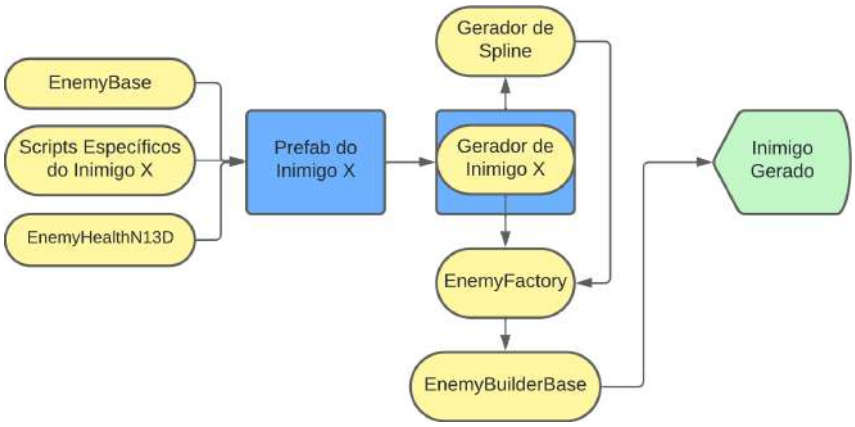
5.2.3.5 Geração de Inimigos

Ainda com base no projeto de (ADAMMYHRE, 2023), utilizamos o conceito de "Fábrica (Factory)". Utilizando esse padrão de desenvolvimento, temos quatro códigos principais que serão responsáveis pela geração dos inimigos no nosso projeto: a base, o gerador, a fábrica e o construtor. Esses scripts são:

- **EnemyBase**: Esse script é a **base**, responsável por atribuir algumas características padrão a todos os inimigos, como a capacidade de armazenar o seu spline por onde irá percorrer e informações sobre quando o objeto deve ser destruído.
- **EnemySpawner**: Esse script é o **gerador** e, na verdade, são vários scripts diferentes, mas que ainda assim serão geradores. Cada *EnemySpawner*, apesar de conter os mesmos métodos, terá pequenas modificações para cada tipo de inimigo que irá gerar.
- **EnemyFactory**: Esse script é a **fábrica**, responsável por reunir todas as informações para a construção do inimigo, como o Prefab, spline e local de geração.
- **EnemyBuilder**: Esse script é o **construtor**, responsável por receber todas as informações, configurar o inimigo e instanciar o mesmo dentro da cena.

Ao final, os objetos que contêm os scripts "*EnemySpawner*" foram posicionados à frente do jogador na cena, com o início à frente da visão do jogador e o final um pouco atrás dele. Além desses códigos para geração de inimigos, os prefabs dos inimigos também devem conter obrigatoriamente o script "**EnemyHealthN13D**", responsável pelo sistema de vida dos inimigos e pela pontuação que eles valem para o sistema de pontos após serem eliminados pelo jogador. Além disso, alguns inimigos possuem códigos específicos, pois têm interações diferentes. Por exemplo, o *inimigo 1*, apresentado nesta seção, ao morrer, libera pequenas naves que perseguem o jogador por um curto período de tempo. Todo esse processo de geração de inimigos pode ser melhor compreendido através do diagrama na Figura 59.

Figura 59 – Diagrama Representativo do Processo de Geração de Inimigos



Fonte: elaborado pelo autor (2024)

5.2.3.6 Sistemas: Vida, Energia, Pontos e Contagem de Bombas

Esses sistemas estão diretamente conectados aos elementos visíveis na interface durante o nível, pois são informações codificadas importantes que devem ser expostas ao

jogador. Cada um desses sistemas está em um script diferente.

O sistema de energia está presente no script já apresentado, *PlayerMovement2*, pois está diretamente relacionado à movimentação do jogador e é responsável por permitir o uso da habilidade de impulso. Enquanto o jogador estiver pressionando o botão de impulso, a energia é consumida através da subtração de uma variável que armazena o valor que representa a quantidade de energia disponível. Ao soltar o botão, essa variável recebe uma soma constante de pequenos valores, recuperando a energia até o máximo de 100%. De maneira semelhante, o sistema de contagem de bombas, previamente abordado no script *Bomb Spawer*, simplesmente resgata a quantidade de bombas e a apresenta na interface.

O sistema de vida possui um script próprio, denominado *PlayerHealth*, que contém três métodos. O método responsável pela redução do valor total da vida do jogador chama-se *TakeDamage*. Esse método público, sempre que chamado, causa uma redução na vida total do jogador, como apresentado no código 5.2.

Listing 5.2 – Método "TakeDamage"

```
1  public void TakeDamage(int damage)
2  {
3      currentHealth -= damage; // Subtrai da vida do jogador o valor
           recebido em "damage"
4      cameraShake.TriggerShake();
5      if (currentHealth <= 0) // Jogador morre se a vida chegar a menor ou
           igual a zero
6      {
7          Die();
8          menus.PlayerDied();
9      }
10 }
```

Assim como o sistema de vida, o sistema de pontos também possui um script próprio, denominado *PlayerStats*. Esse script contém o método público *AddScore*, responsável por armazenar todos os pontos obtidos durante o nível em uma "chave" denominada *Score*, utilizando a classe *PlayerPrefs*, que armazena as preferências do jogador entre diferentes sessões de jogo, como apresentado no código 5.3.

Listing 5.3 – Método "AddScore"

```
1  public void AddScore(int points)
2  {
3      score += points;
4      PlayerPrefs.SetInt("Score", score); // Armazena a pontuacao com a
           chave "Score"
5      PlayerPrefs.Save(); // Salva as alteracoes
6  }
```

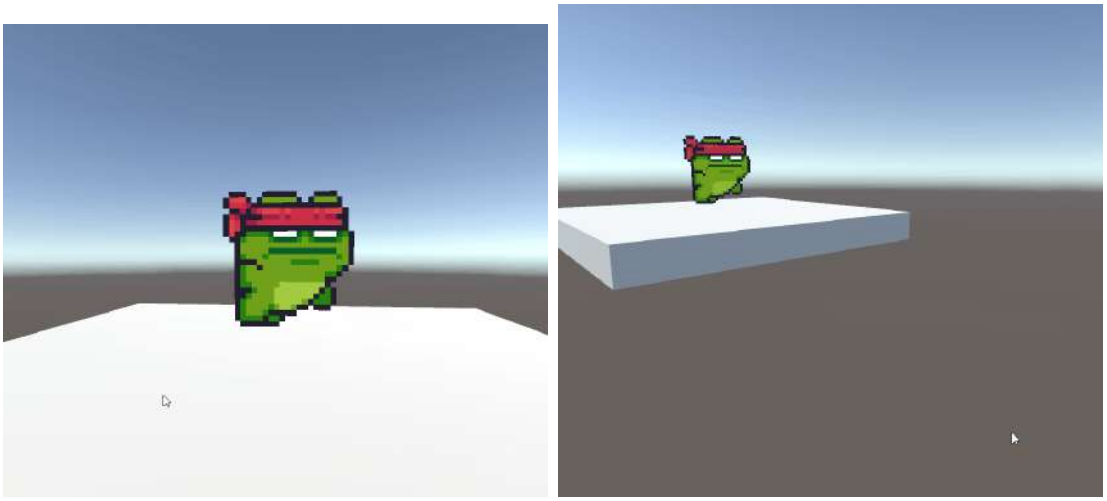
5.3 Nível 2

Nesta seção, abordaremos os principais pontos do processo de desenvolvimento do Nível 2 do jogo, desde a prototipação de mecânicas e modelagem até a programação.

5.3.1 Prototipação das Mecânicas do Nível

O primeiro desafio na prototipação deste nível foi o visual, pois a base visual é inspirada no *Doom* (1993). No *Doom*, os inimigos e elementos do cenário são sprites posicionados no ambiente tridimensional; porém, esses sprites sempre estão voltados na direção do jogador. Portanto, o primeiro passo na prototipação foi criar essa mecânica, como ilustrado na Figura 60.

Figura 60 – Protótipo de Sprites Com Orientação Para Câmera



Fonte: elaborado pelo autor (2024)

O segundo desafio na prototipação foi criar uma nova movimentação para o jogador, que agora está em uma perspectiva em primeira pessoa, porém não terá a capacidade de pular. Portanto, o jogador precisa ser capaz de andar pelo cenário e subir pequenas elevações. Durante o processo de prototipação e pesquisa, chegamos ao componente *Character Controller*, que foi ideal para criar uma movimentação básica inicial e validar o planejamento do mapa do jogo, que inclui poços com ácido onde o jogador poderia cair.

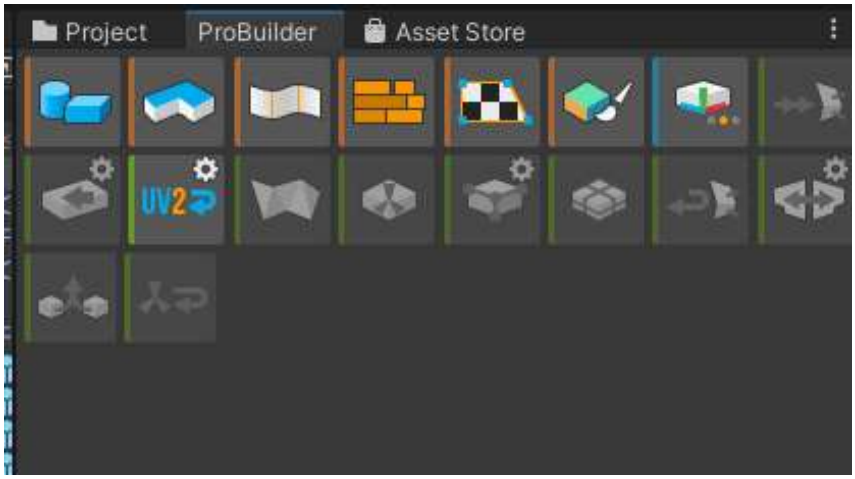
5.3.2 Modelagem 3D e Sprites

Diferente do Nível 1, este nível se passa dentro de uma base subterrânea com vários quartos. Portanto, foram necessárias a modelagem do mapa e do monstro que estará presente neste nível, além do uso de diversos sprites que representarão os itens do jogo.

5.3.2.1 Mapa do Nível 2

A modelagem 3D do mapa deste nível foi realizada utilizando uma ferramenta da *Unity* chamada *ProBuilder*, que já foi apresentada anteriormente. O *ProBuilder* possui uma interface com algumas opções básicas para a modelagem de objetos 3D, como mostrado na Figura 61.

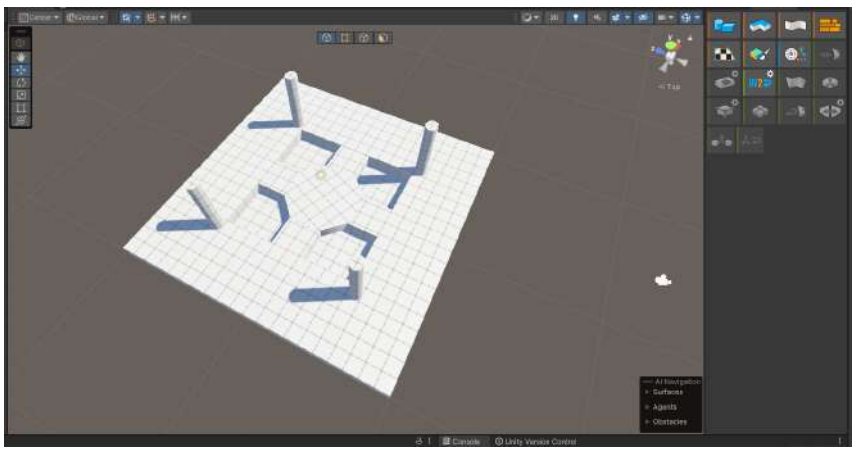
Figura 61 – Interface do ProBuilder Dentro da Unity



Fonte: elaborado pelo autor (2024)

Sendo o *ProBuilder* uma ferramenta amplamente utilizada no processo de criação de mapas 3D na *Unity*, foi relativamente fácil explorar a documentação e entender como utilizar suas ferramentas para manipulação de vértices, arestas e faces. Baseando-nos no processo de **modelagem arquitetônica**, optamos por desenhar a planta do nosso mapa à mão, em vez de utilizar um software de design (Figura 37). Com essa base desenhada e as ferramentas facilitadoras da *Unity*, foi possível iniciar rapidamente a modelagem deste nível (Figura 62).

Figura 62 – Modelagem inicial do Mapa do Nível 2



Fonte: elaborado pelo autor (2024)

Após compreender como seria realizada a modelagem do cenário com o *ProBuilder*, foi necessário encontrar as texturas que comporiam todo o cenário, incluindo paredes, pilares, portas e o ácido presente em alguns trechos das salas. Optamos por utilizar sprites feitas em **pixel art**, pois isso tornaria o visual mais próximo do objetivo de parecer um jogo 3D antigo.

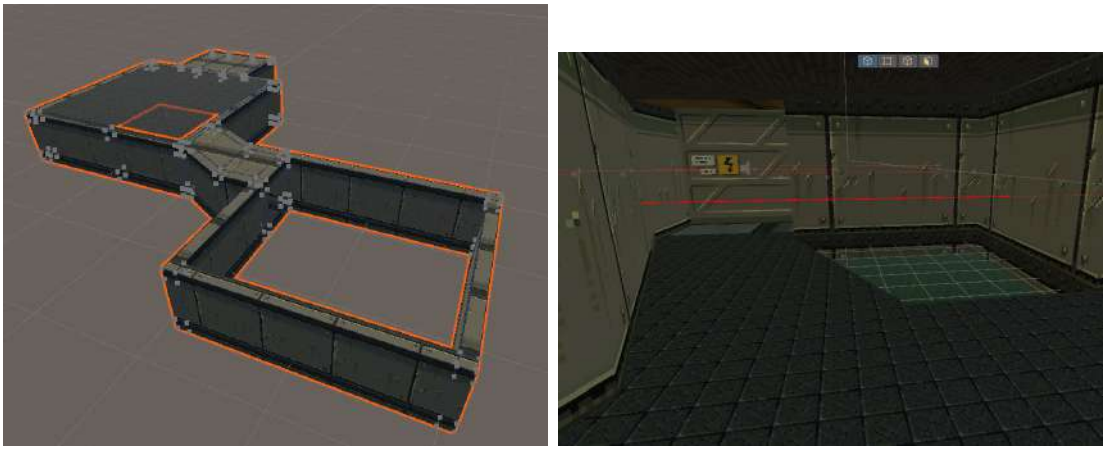
Seguindo essa ideia de texturas em pixel art, decidimos utilizar um pacote de sprites criado por (MARTIAN, 2022), disponibilizado em um site que oferece recursos para desenvolvimento de jogos, tanto de forma gratuita quanto paga. Dentro desse pacote, que contém diversas pixel arts retro, utilizamos apenas algumas delas para compor o nosso cenário (Figura 63).

Figura 63 – Sprites Utilizados para Texturizar o Mapa do Nível 2

Fonte: Martian (2022)

Por fim, com as texturas adquiridas, foi possível realizar o processo de modelagem do mapa e, em paralelo, aplicar as texturas na estrutura, como pode ser visto na Figura 64.

Figura 64 – Modelagem do Mapa com Texturas Aplicadas



Fonte: elaborado pelo autor (2024)

5.3.2.2 Jogador

Apesar do nível 2 ser em primeira pessoa e não precisar da modelagem de um personagem, o jogador pode ser representando com alguma parte do corpo ou dispositivo aparecendo na esquerda ou na direita da tela, assim como o próprio jogo base deste nível fez. Então aqui a ideia foi também criar essa visualização, onde o jogador pudesse visualizar sua mão com uma arma enquanto explora e derrota os monstros deste nível.

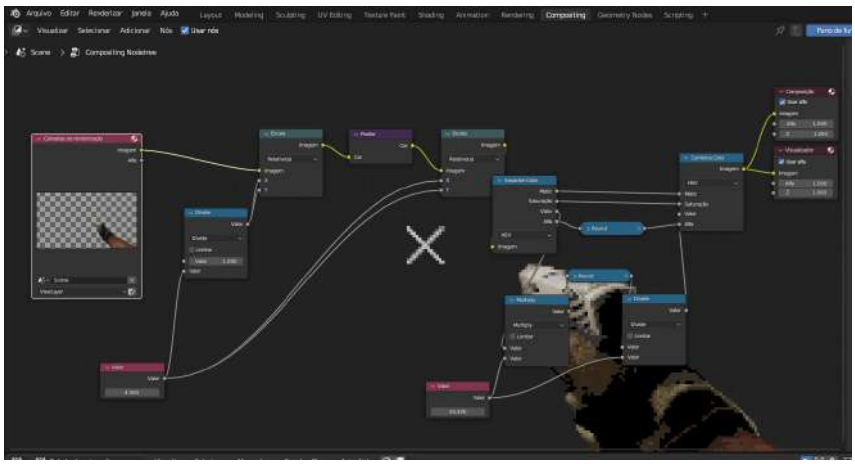
Para realizar isso de forma mais prática, em vez de modelar um braço e uma arma realista para aplicar dentro do nível, pesquisamos por modelos disponíveis para download e encontramos o modelo 3D de (DANIIL, 2023), que continha um braço com uma pistola (Figura 65), exatamente o que era necessário para nosso projeto.

Figura 65 – Modelo 3D do Braço com Arma

Fonte: Daniil (2023)

Após obter este modelo 3D, que já continha algumas animações, foi necessário apenas aplicar as texturas corretamente e realizar o processo de composição no Blender (Figura 66) para criar os sprites em baixa resolução.

Figura 66 – Composição para Renderizar os Sprites do Jogador



Fonte: elaborado pelo autor (2024)

Por fim, o sprite foi aplicado dentro do jogo no GameObject do jogador, utilizando o sistema de animação da Unity. O sprite foi posicionado no lado direito da tela, como pode ser visto na Figura 67.

Figura 67 – Sprite do Jogador Dentro de Jogo

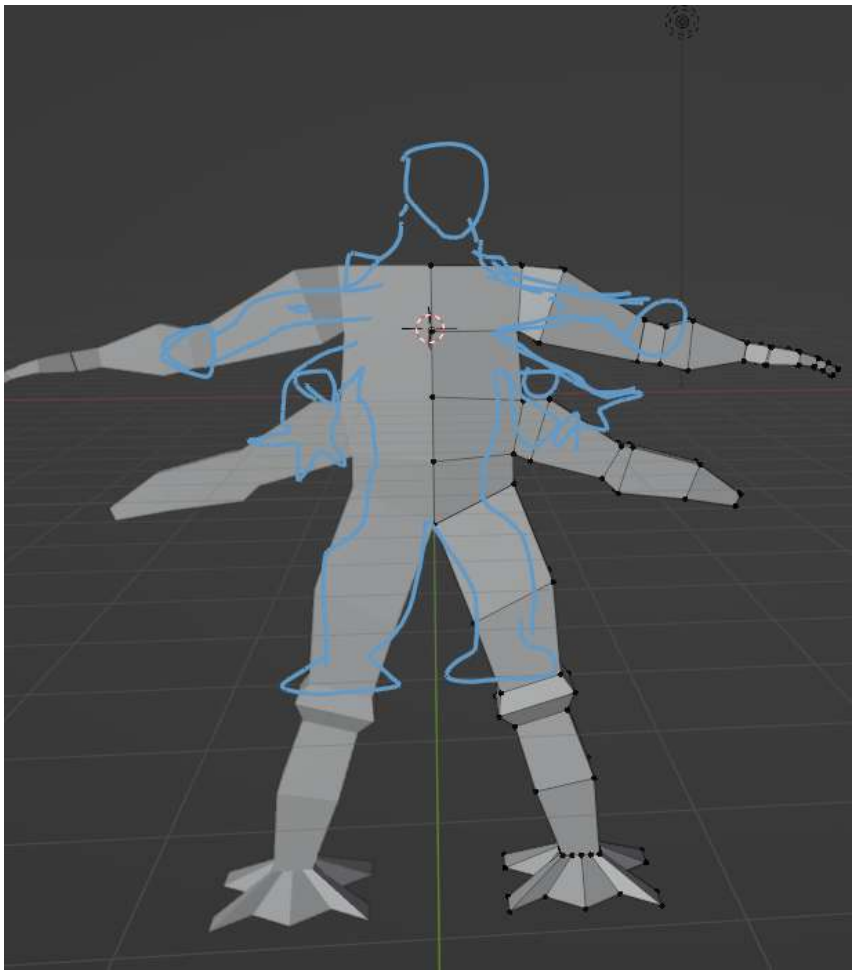
Fonte: elaborado pelo autor (2024)

5.3.2.3 Inimigo

O último trabalho com modelo 3D neste nível foi a criação do inimigo, o mais complexo até então. Durante o processo de desenvolvimento, decidimos criar nosso próprio modelo 3D. Além da modelagem, foram realizados os processos de pintura, colocação de ossos no modelo (**rigging**) e animação.

Durante o processo de modelagem deste personagem, utilizamos diversas técnicas, como **Box Modeling** e **Imagem de Referência**, para criar a base do corpo no Blender, como mostrado na Figura 68.

Figura 68 – Começo da Modelagem do Personagem Inimigo



Fonte: elaborado pelo autor (2024)

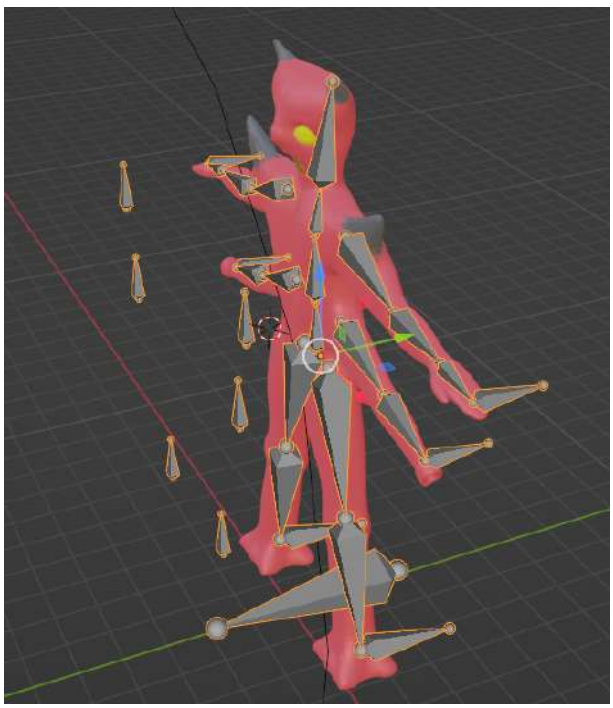
Após a modelagem do personagem, utilizamos a técnica de **escultura** para criar alguns detalhes no corpo, como abdômen, chifres e detalhes na cabeça. Além disso, usamos a técnica de **pintura** para colorir a boca, dentes, olhos e espinhos do corpo, chegando à versão final, como mostra a Figura 69.

Figura 69 – Versão final do Modelo 3D do Personagem Inimigo

Fonte: elaborado pelo autor (2024)

O último passo foi realizar o **rigging** e a animação do modelo, sendo o processo de **rigging** o mais trabalhoso de todos até agora. A aplicação de ossos na malha e a definição do movimento de cada osso requerem bastante prática e tempo. Apesar dos desafios, conseguimos concluir o rigging, as animações necessárias e, por fim, a renderização dos sprites. Na Figura 70, é possível ver os "ossos" do modelo e sua versão em sprite.

Figura 70 – Modelo 3D com 'ossos'



Fonte: elaborado pelo autor (2024)

5.3.2.4 Itens de Cenário

Itens de cenário serão todos representados por sprites, utilizando a mecânica previamente apresentada no protótipo, que assegura que os sprites mantenham sua orientação sempre voltada para a câmera do jogador. Esses sprites representarão diversos objetos, como barris, kits de primeiros socorros, escudos, entre outros, cada qual com uma função específica e representação visual dentro do jogo.

5.3.3 Programação

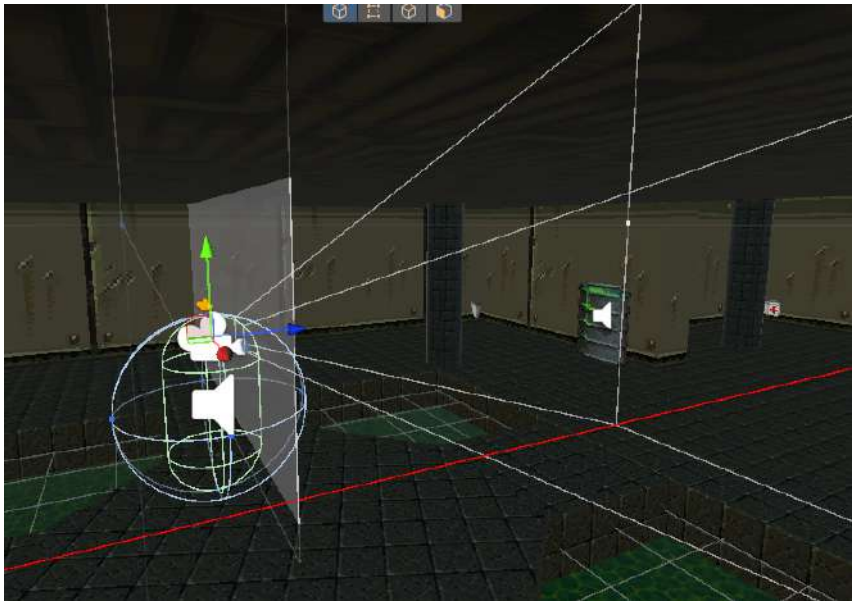
O foco desta subseção é abordar os principais pontos de programação e aplicação dos códigos criados em C# relacionados ao nível 2. Diferentemente do nível anterior, este apresenta uma complexidade de programação reduzida.

5.3.3.1 Câmera e Movimentação

O script da câmera deste nível (*CameraController*) já inclui toda a movimentação do jogador, uma vez que, por ser em primeira pessoa, a câmera está embutida no corpo do personagem dentro da cena (Figura 71). O grande diferencial da movimentação em primeira pessoa aqui, assim como em *Doom* (1993), é que o jogador pode utilizar as teclas W e S para se movimentar no cenário, enquanto as teclas A e D são usadas apenas para rotacionar a câmera para a direita e a esquerda. Todas as informações de movimentação são enviadas ao componente **Character Controller**, responsável pelo controle do personagem neste nível (Código 5.4).

O controle de câmera mais limitado aplicado a este nível pode parecer estranho para os dias atuais, mas está de acordo com nosso objetivo, que é criar um aspecto de jogo 3D antigo.

Figura 71 – Câmera no “Corpo” do Jogador Dentro de Cena



Fonte: elaborado pelo autor (2024)

Listing 5.4 – Método “HandleMovement”

```
private void HandleMovement()
{
    float translation = Input.GetAxis("Vertical") * speed;
    float rotation = Input.GetAxis("Horizontal") * rotationSpeed;

    // Movimentacao para frente e para tras
    translation *= Time.deltaTime;

    // Rotacao esquerda e direita
    rotation *= Time.deltaTime;

    Vector3 forwardMovement = transform.forward * translation;

    // Envia os comandos de movimento para o Character Controller
    characterController.SimpleMove(forwardMovement);

    transform.Rotate(0, rotation, 0);
}
```

5.3.3.2 Ações do Jogador

O jogador, neste nível, terá apenas dois tipos de ações: disparos e abertura de portas, ambas controladas pelo script *PlayerBase*. O **disparo** é implementado de forma simples, gerando a instância da bala no local atribuído nos parâmetros do script e, em seguida, adicionando força ao disparo por meio do componente **Rigidbody** do objeto

instanciado. Já a **abertura de portas** é realizada por meio de um método que lança um raio para detectar uma porta. Se o jogador possuir a chave e o raio colidir com uma porta, o método chama a função responsável por abrir a porta e remove a chave, como mostrado no código 5.5.

Listing 5.5 – Método "OpenDoor"

```
public void OpenDoor()
{
    RaycastHit hit;
    // Use Physics.Raycast com a LayerMask
    if (Physics.Raycast(transform.position, transform.forward, out hit,
        Mathf.Infinity, groundLayer) && key)
    {
        DoorController door = hit.transform.GetComponent<Level2.
            DoorController>();
        if (door != null)
        {
            door.Open(); // Chame o metodo Open para abrir a porta
            key = false; // Defina a chave como false
        }
    }
}
```

5.3.3.3 Sistemas: Vida, Escudo, Munições e Chaves

Os sistemas deste nível foram desenvolvidos em forma de status que o jogador possui durante a fase, sendo cada um deles implementado por meio de um método simples de adição, chamado conforme necessário. Por exemplo, ao coletar itens de escudo, o método *AddShield*, presente no script *PlayerBase*, é acionado. Esse método soma o valor coletado ao valor atual de escudo e, em seguida, verifica se o valor de escudo excede o limite permitido. Se isso ocorrer, o valor é ajustado para o máximo permitido (Código 5.6).

Listing 5.6 – Método "AddShield"

```
public void AddShield(float bonusShield)
{
    shield += bonusShield;

    if (shield > 50) //Se o Shield maior que 50
    {
        shield = 50; // Torna Shield igual a 50
    }
}
```

Essa lógica de adição e verificação do limite máximo é aplicada apenas ao sistema de vida e escudo do jogador. O sistema de munições não possui essa limitação, enquanto o

sistema de chaves permite um limite máximo de uma chave por vez, pois não é possível obter duas chaves simultaneamente.

5.3.3.4 Coleta de Itens

Os itens no cenário que adicionam valores aos sistemas do jogador possuem um script chamado *LifeShieldAmmo*, que realiza uma verificação. O *GameObject* presente na cena possui uma marcação (*tag*) que indica qual tipo de item ele é. Ao entrar em contato com o jogador, o script verifica a *tag* do item e concede o bônus correspondente, sendo essa a forma pela qual o item é coletado.

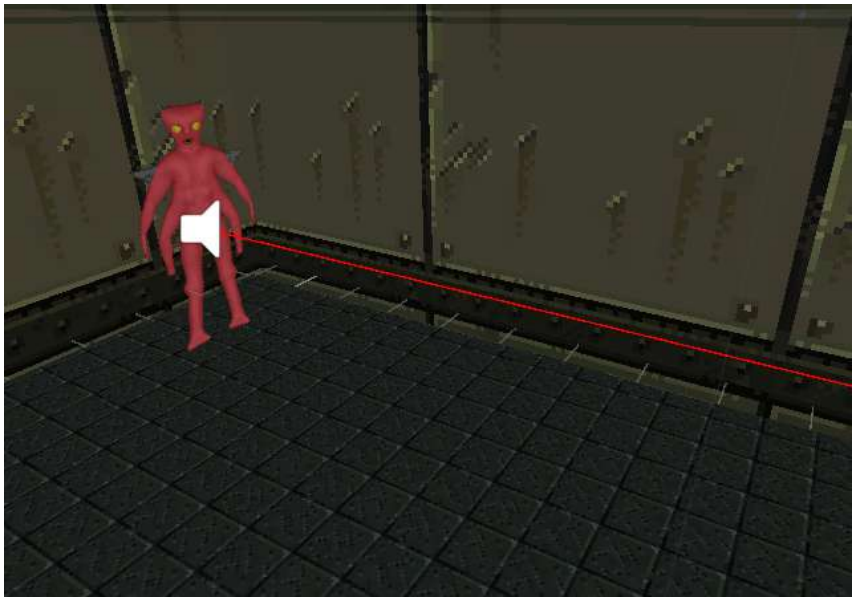
5.3.3.5 Inimigo

A programação das ações do inimigo neste nível consiste em um script chamado *Enemy1*, que herda da classe *EnemyStats* e possui dois métodos principais. Diferente do nível anterior, onde os inimigos eram gerados por uma fábrica para terem movimentação aleatória, neste nível, os inimigos já estão posicionados no mapa. Os métodos são:

- **MoveTowardsPlayer**: Este método faz o inimigo se mover em direção ao jogador (target). Se o jogador não estiver ao alcance, a animação de caminhada é ativada e o estado do inimigo é ajustado para "andando". Se o jogador estiver ao alcance, a animação de caminhada é desativada e o inimigo para de se mover.
- **Attack**: Este método controla o ataque do inimigo. Se o inimigo não estiver atacando e o jogador estiver dentro do alcance, a animação de ataque é iniciada, projéteis são disparados e o estado do inimigo é marcado como "atacando". Se o inimigo já estiver atacando e o tempo de ataque exceder o intervalo de disparo, a animação de ataque é interrompida e o estado de ataque é desativado.

O que permite ao inimigo detectar o jogador e ativar os métodos apresentados são dois elementos principais em sua programação. O primeiro é um colisor em formato de esfera, que define o alcance máximo no qual ele pode atirar. O segundo elemento é o *Raycast*, que lança um raio em linha reta, responsável por verificar se há obstáculos entre o inimigo e o jogador (Figura 72). Caso existam obstáculos, o inimigo não consegue "ver" o jogador naquele momento.

Figura 72 – Raio de Verificação de Obstáculos



Fonte: elaborado pelo autor (2024)

5.4 Nível 3

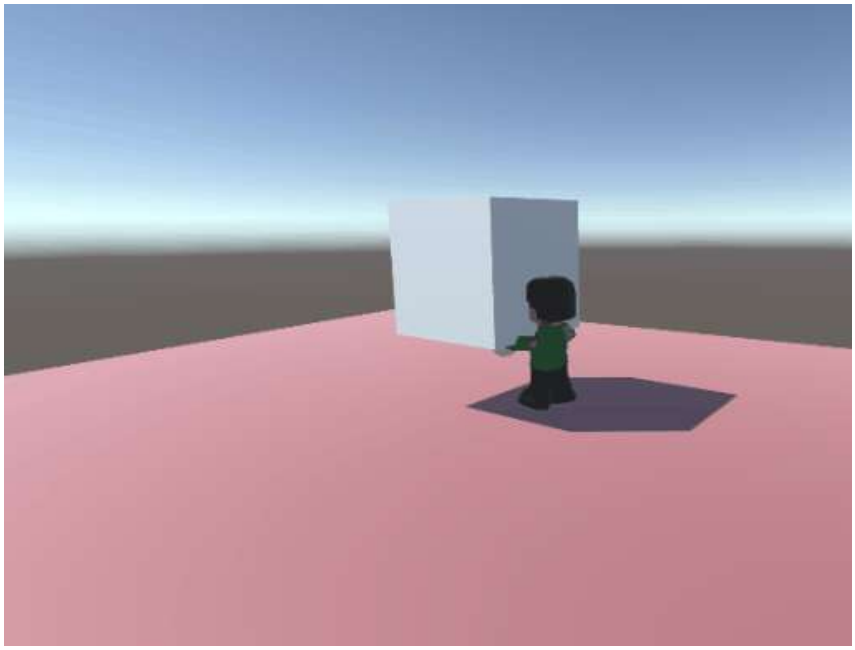
Nesta seção, abordaremos os principais pontos do processo de desenvolvimento do Nível 3, o último nível deste jogo, desde a prototipação de mecânicas e modelagem até a programação, assim como nos níveis anteriores.

5.4.1 Prototipação das Mecânicas do Nível

A prototipação inicial deste nível envolveu dois principais elementos: as ações do jogador com o sistema de animação e a *inteligência artificial* dos inimigos. A prototipação das mecânicas do jogador foi realizada de maneira um pouco diferente em relação aos níveis anteriores, pois, antes de desenvolver suas mecânicas, foi decidido que seria importante ter o modelo 3D do jogador com as animações das ações já prontas. Isso foi necessário porque, neste nível, as animações foram feitas no **Blender**, diferentemente dos níveis anteriores, que utilizaram o controle direto do corpo dos personagens e "sprites", respectivamente.

A primeira prototipação das mecânicas do jogador foi realizada com o modelo 3D e as animações já prontas, como mostrado na Figura 73.

Figura 73 – Prototipagem da Mecânica de Agarrar Inimigos



Fonte: elaborado pelo autor (2024)

O protótipo de comportamento dos inimigos foi desenvolvido em sequência. Diferentemente dos outros níveis, neste, os inimigos precisariam se movimentar de forma autônoma e consistente em um terreno. Para isso, foi necessário utilizar as ferramentas de inteligência artificial da Unity, que serão abordadas de forma mais detalhada na seção de programação, conforme ilustrado na Figura 74, onde um personagem caminha em direção a um ponto alvo.

Figura 74 – Prototipagem da Inteligencia Artificial dos Inimigos

Fonte: elaborado pelo autor (2024)

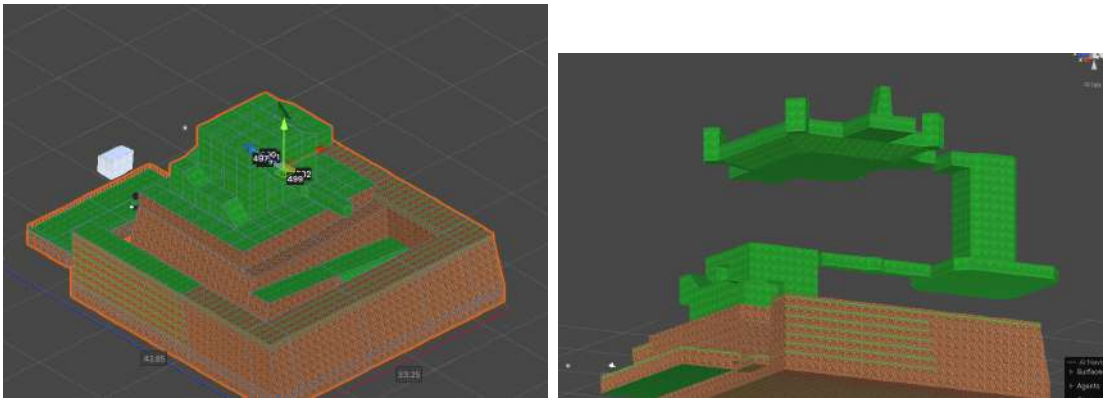
5.4.2 Modelagem 3D

Nesta subseção, abordaremos os processos de modelagem do mapa, assim como no nível anterior, bem como a modelagem do personagem principal e de outros modelos 3D de terceiros, que foram ajustados e animados para uso neste estudo.

5.4.2.1 Mapa do Nível 3

Assim como no nível anterior, a modelagem do mapa deste nível foi realizada com a ferramenta **ProBuilder**, utilizando as mesmas técnicas já abordadas no mapa do nível 2 e os próprios recursos de manipulação de **vértices**, **arestas** e **faces**. O modelo 3D final pode ser visualizado na Figura 79.

Figura 75 – Modelo 3D do Mapa do Nível 3 Feito com ProBuilder



Fonte: elaborado pelo autor (2024)

Dentro desse processo de modelagem, foi necessária a busca por texturas que fossem compatíveis com a proposta de um 3D mais antigo. Porém agora, um pouco mais avançado, assim como o próprio *Super Mario 64*. Dentre essa busca de texturas, foi encontrado um site chamado “models-resource”. Esse site (RESOURCE, 2024) tem o objetivo de armazenar recursos utilizados na construção de jogos de diversas plataformas e disponibilizá-los para uso das comunidades, incluindo o próprio jogo *Super Mario 64*. Com recursos disponíveis dentro da plataforma, optamos por utilizar algumas das texturas para dar esse aspecto mais velho ao nosso mapa.

5.4.2.2 Jogador

O processo de criação do modelo 3D do personagem principal neste nível foi um dos pontos mais complexos dentro deste nível, pois havia a necessidade de ter um modelo mais detalhado e com uma variedade de animações a serem criadas, o que até então não havia sido necessário neste projeto.

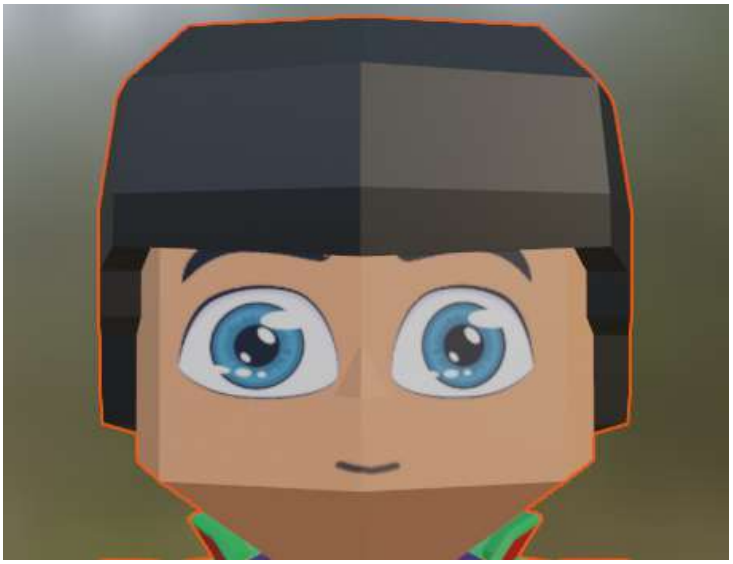
Para a modelagem inicial do personagem, foram utilizadas várias técnicas, porém uma das mais relevantes foi a Imagem de Referência (Figura 76), sendo a imagem o próprio **Mario** do *Super Mario 64*, com a intenção de alcançar uma proximidade maior com a estatura do Mario, que é nossa base.

Figura 76 – Modelagem do Personagem Principal com Imagem de Referência

Fonte: elaborado pelo autor (2024)

Todo o corpo do personagem foi colorido com o uso de materiais aplicados nas faces do modelo 3D. Já o rosto do personagem foi ilustrado com um pincel para pintura da boca e sobrancelhas e a aplicação de uma imagem de olhos no rosto do personagem (Figura 77).

Figura 77 – Ilustração do Rosto do Personagem



Fonte: elaborado pelo autor (2024)

A última etapa na produção desse modelo foi a realização do processo de *rigging*. No entanto, diferentemente do modelo do inimigo do nível 2, não foi realizada uma tentativa de criar todo o esqueleto de animação manualmente no *Blender*, a fim de evitar problemas com a malha do personagem e na animação.

Com o objetivo de evitar esses problemas, foi decidido utilizar o site *Mixamo*. Apesar da principal característica do site ser um grande acervo de animações gratuitas, uma outra função muito interessante é a capacidade de realizar o processo de *rigging* semi-automaticamente em personagens humanoides, permitindo que o modelo utilize as animações da plataforma. Assim, basicamente, aplicamos o modelo na plataforma e, em seguida, baixamos o modelo já com o esqueleto pronto para criar as animações.

5.4.2.3 Inimigos

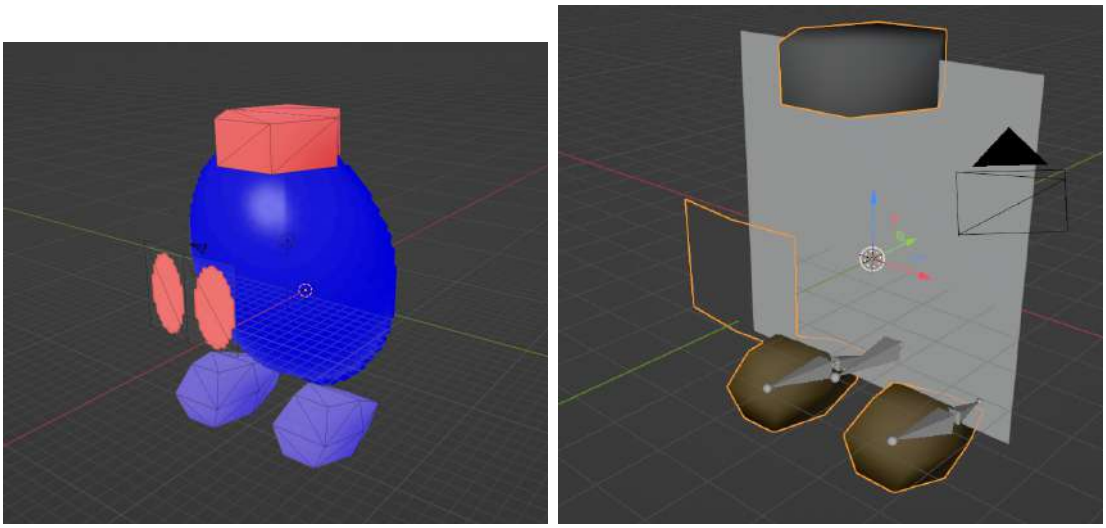
Para os modelos 3D dos inimigos deste nível, foi decidido, durante o processo de modelagem, que seriam utilizados modelos disponibilizados pelo site já citado anteriormente, "*models-resource*" (RESOURCE, 2024). Optamos por utilizar três modelos 3D: dois para inimigos comuns e um para o chefe final do jogo (Figura 78).

Figura 78 – Modelos 3D do Super Mario 64 Utilizados como Inimigos

Fonte: Resource (2024)

Apesar de baixar os modelos 3D ter ajudado bastante no processo, ainda foi necessário realizar o processo de aplicar as texturas que vieram nos arquivos, separar as partes que precisariam ser movimentadas e animar os mesmos (Figura 79), todo esse processo realizado através do *Blender*. Após esse ajuste nos modelos, cada um foi exportado para a *Unity*.

Figura 79 – Realizando Ajustes no Modelo 3D Através do Blender



Fonte: elaborado pelo autor (2024)

5.4.3 Programação

O foco desta subseção é abordar as partes mais relevantes da programação deste nível, a aplicação de códigos e o uso de ferramentas da *Unity* para criar a jogabilidade do Nível 3.

5.4.3.1 Movimentação com Orientação da Câmera

Um dos principais pontos para um jogo de plataforma 3D é a movimentação do personagem. No entanto, em jogos antigos, a câmera, na grande maioria dos casos, tinha uma movimentação automatizada, permitindo ao jogador controlar apenas o personagem.

O protagonista, neste nível, tem uma movimentação programada de forma simples. O maior diferencial de sua movimentação é que a orientação da câmera afeta totalmente as direções em que o personagem se desloca. Ou seja, dentro do *script* **PlayerControl**, no método **Move()**, a direção do movimento é calculada com base na entrada do jogador (horizontal e vertical), combinando a direção da câmera (frente e direita). O resultado é um vetor que aponta na direção que o jogador deve se mover, e ele é normalizado para garantir que a velocidade de movimento seja uniforme em todas as direções (Código 5.7).

Listing 5.7 – Linhas do Código de Movimentação no Método "Move()"

```
void Move()
{
    // Recebe as entradas do jogador
    float horizontalInput = Input.GetAxis("Horizontal");
    float verticalInput = Input.GetAxis("Vertical");
    if (horizontalInput != 0 || verticalInput != 0)
    {
        // Calcule as direcoes para frente e para a direita com base na
           rotacao da camera
        forwardDirection = Camera.main.transform.forward;
        rightDirection = Camera.main.transform.right;

        // Remova qualquer movimento vertical das direcoes
        forwardDirection.y = 0;
        rightDirection.y = 0;
        forwardDirection.Normalize();
        rightDirection.Normalize();

        // Calcule a direcao do movimento com base na entrada
        Vector3 movementDirection = (forwardDirection * verticalInput) +
            (rightDirection * horizontalInput);
        movementDirection.Normalize();
    }
}
```

Resumidamente, isso significa que, se o jogador apertar a mesma tecla com a câmera em posições diferentes, o personagem se moverá em direções diferentes.

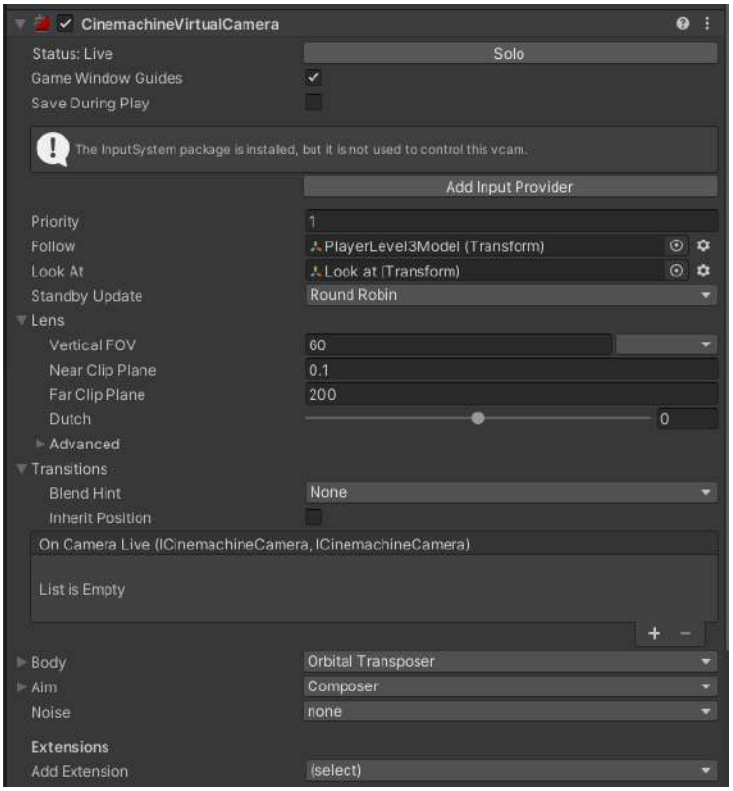
5.4.3.2 Câmera Principal

Inicialmente, o controle da câmera principal foi feito através de código. No entanto, a complexidade das variações no comportamento da câmera tornou a programação muito difícil de gerenciar. Diante disso, após um estudo mais aprofundado do funcionamento do *Cinemachine*, recurso citado anteriormente na Seção 5.2, optou-se por utilizar essa ferramenta para automatizar a movimentação da câmera.

Dentro do *Cinemachine*, foram necessárias algumas configurações para que a câmera se movesse de forma automática (Figura 80):

1. Definir o personagem do jogador como o alvo a ser seguido.
2. Designar um ponto no pescoço do personagem como local para onde a câmera deve sempre olhar.
3. Utilizar o *Orbital Transposer* como algoritmo do **Body** da câmera virtual, responsável por mover a câmera em relação ao alvo de acompanhamento, que é o personagem.
4. Definir o *Composer* no **Aim**, que ajusta automaticamente a rotação da câmera para garantir que o alvo esteja sempre visível na tela.

Figura 80 – Configurações da Câmera Virtual do Jogador



Fonte: elaborado pelo autor (2024)

Além dessas configurações definidas dentro do componente *CinemachineVirtual-Machine*, foi necessária a adição do componente *Cinemachine Collider* e o script "Camera-Rotation". O *Cinemachine Collider* foi utilizado para impedir que a câmera atravesse as paredes do mapa durante a movimentação do jogador, garantindo que ela se comporte de maneira natural, sem invadir objetos sólidos. Já o script "CameraRotation" possibilitou a codificação de um controle manual da câmera, permitindo que o jogador ajuste sua posição para a direita ou esquerda, alterando os valores do eixo X do componente *Orbital Transposer* com as teclas "I" e "P", respectivamente.

5.4.3.3 Câmeras Adicionais

Além da câmera que acompanha o jogador durante a maior parte do jogo, haverá outras câmeras em pontos específicos, nas quais o jogador observará uma alternância de perspectivas para facilitar escaladas ou destacar momentos importantes. Um exemplo disso é a escalada para a batalha final, onde uma câmera ganha aumento de **prioridade** para se tornar a principal durante a escalada em torno da torre, seguindo o caminho pré-estabelecido com um *Dolly Track* posicionado ao redor (Figura 82).

Essa **prioridade** (*Priority*) é um parâmetro dentro das câmeras virtuais que pode ser alterado tanto nos ajustes do componente *CinemachineVirtualCamera* (Figura 80) quanto via código. A câmera com maior valor de prioridade se torna a principal. Isso possibilitou a criação de transições entre câmeras ao longo do Nível 3, conforme programado em nosso projeto.

Figura 81 – “Dolly Track” Criado para Movimentação da Câmera de Escalada

Fonte: elaborado pelo autor (2024)

5.4.3.4 Mecânicas do Personagem Principal

Neste nível, as mecânicas do jogador foram implementadas no script “**PlayerControl**” e são divididas entre *ativas* e *passivas*. As mecânicas *ativas* referem-se às ações que o jogador executa diretamente, como pular, agarrar inimigos-bomba e arremessá-los. Já as mecânicas *passivas* são aquelas que ocorrem automaticamente, como receber dano ao cair de uma certa altura ou ser empurrado ao receber dano. A seguir, detalharemos cada uma delas.

Mecânicas Ativas:

- Pulo e Pulo Alto: A programação desta mecânica foi implementada dentro do método Jump(), utilizando uma lógica relativamente simples, onde foi necessário adicionar uma força no eixo Y do CharacterController do personagem. Para criar o pulo alto, foi desenvolvida uma lógica que detecta se o jogador realiza um segundo pulo dentro de um pequeno intervalo de tempo. Caso isso ocorra, o valor da força aplicada ao pulo será um pouco maior, permitindo saltos mais altos.

- Pegar e Arremessar: A mecânica de pegar e arremessar foi implementada no método Take_Object(). Quando o jogador pressiona a tecla “O”, o código verifica se há um objeto pegável por meio de um *Raycast* — neste caso, os inimigos-bomba — à frente do jogador. Se o jogador ainda não estiver segurando um inimigo e um inimigo com a tag “*Catchable*” for detectado, ele será posicionado na mão do jogador e sua

física será desativada. Ao pressionar "O" novamente, o inimigo é solto e arremessado à frente, aplicando-se uma força física, permitindo sua interação dinâmica com o ambiente.

Mecânicas passivas:

- Dano de Queda: O método responsável por essa mecânica se chama **FallDamage**. O código utiliza duas variáveis para armazenar o valor da posição vertical do personagem: *previousYPosition* e *currentYPosition*. A primeira guarda o valor da posição vertical anterior, e a segunda guarda o valor atual. Quando o personagem está no chão, ele atualiza a posição vertical anterior. Se o jogador cai de uma altura maior que o valor limite definido e está no chão, o método aplica dano ao jogador.
- Recuo ao Receber Dano: Essa mecânica está diretamente ligada ao método **GetDamage()**, responsável por aplicar dano à vida do jogador. Quando o personagem recebe dano que não é de queda, o método **Knockback()** é chamado para aplicar um efeito de recuo ao jogador na direção contrária à origem do dano. A cada quadro, o vetor de movimento é calculado e aplicado ao *CharacterController*, e a distância percorrida é incrementada até alcançar a distância máxima permitida.

5.4.3.5 Geração de Inimigos

Este nível contém uma geração de inimigos, assim como no nível 1, porém com uma complexidade de programação bem mais baixa, pois aqui utilizamos o NavMesh da Unity, o que minimiza a complexidade de criar áreas terrestres pelas quais os personagens podem caminhar. Cada tipo de inimigo recebeu um script com a mesma estrutura, mas que geraria aquele inimigo específico. Por exemplo, o inimigo bomba é gerado pelo script BlackBombSpawner. Este script necessita apenas de quatro parâmetros:

- **enemyPrefab**: Prefab do personagem que será gerado;
- **pathPoints**: Objetos Transform que definem os pontos para onde esse personagem gerado deve caminhar;
- **spawnedEnemy**: Parâmetro que guarda a referência do último personagem gerado;
- **respawnDelay**: Responsável por definir o intervalo de tempo da geração de inimigos.

Ele verifica se os parâmetros necessários (como o prefab do inimigo e os pontos de caminho que o inimigo deve seguir) estão configurados corretamente e, em seguida, gera o inimigo em intervalos regulares ou quando o inimigo anterior morre. O spawner atribui aos inimigos os pontos de caminho pelos quais eles devem se movimentar e mantém o controle

do inimigo ativo para garantir que ele seja reaparecido, caso esteja morto ou inexistente, seguindo um ciclo contínuo de geração e reaparecimento de inimigos.

5.4.3.6 Movimentação dos Inimigos

Depois que os inimigos são gerados e posicionados na cena, cada um deles possui em seu próprio código um método chamado *moveToNextPoint*, responsável por gerenciar o próximo ponto para o qual o inimigo deve se mover (Código 5.8). Para que os personagens possam navegar pelo terreno do jogo, foi necessário utilizar o **NavMesh**. Através da janela de Navegação da Unity, em poucos segundos é possível criar uma malha de forma automática que define por onde os personagens poderão andar. Na Figura 82, a área demarcada em azul claro mostra os locais onde os personagens gerados conseguem se movimentar.

Listing 5.8 – Método "moveToNextPoint"

```
1  void moveToNextPoint()
2  {
3      if (pathPoints.Count > 0)
4      {
5          // Calcula a distancia entre o ponto de caminho atual e a
               posicao do objeto
6          float distance = Vector3.Distance(pathPoints[currentPathIndex].
               position, transform.position);
7
8          // Define o destino do agente como o ponto de caminho atual
9          agent.destination = pathPoints[currentPathIndex].position;
10
11         if (distance <= 4f)
12         {
13             currentPathIndex++;
14             // Garante que o indice do ponto de caminho volte ao inicio
                   se ultrapassar o numero total de pontos
15             currentPathIndex %= pathPoints.Count;
16         }
17     }
18 }
```

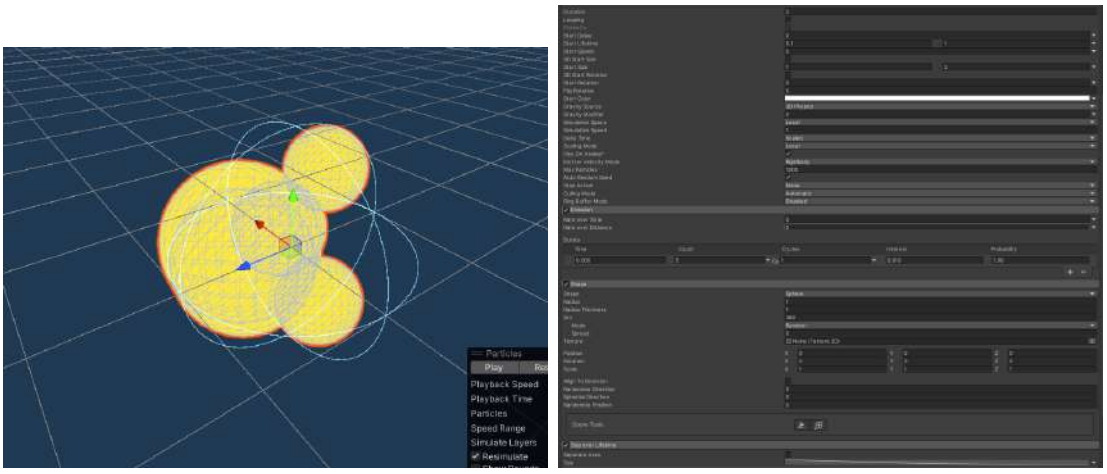
Figura 82 – Mapa com a Malha de Navegação Visível Sobre o Mapa

Fonte: elaborado pelo autor (2024)

5.5 Efeitos visuais

Todos os níveis dentro do projeto envolvem algum tipo de efeito visual, criado e utilizado com ferramentas da própria Unity ou recursos da loja da Unity. No nível 1, por exemplo, os efeitos de explosão foram criados com o Sistema de Partículas (*Particle System*) da Unity, realizando apenas alterações nos parâmetros de geração de partículas, tempo de surgimento e desaparecimento, entre outros (Figura 83). Já no nível 3, um efeito de explosão foi obtido através da Loja de Recursos da Unity e aplicado diretamente no nível, com apenas alterações na escala.

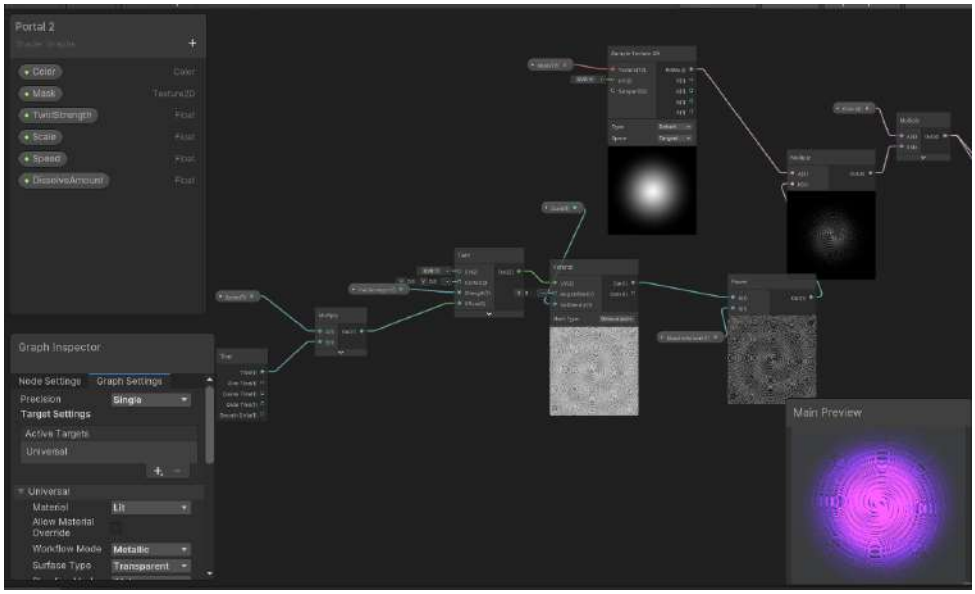
Figura 83 – Criação de Efeito de Explosão com Particle System



Fonte: elaborado pelo autor (2024)

Além dos efeitos visuais de explosões, foram criados portais que são fundamentais para a progressão nos níveis do jogo. Para desenvolver os efeitos visuais de cada portal, foi utilizada a ferramenta *Visual Effect Graph* (Gráfico de Efeitos Visuais). Ela utiliza uma **lógica visual** baseada em nós, permitindo a criação e manipulação de efeitos de partículas de maneira fácil e flexível, como mostrado na Figura 84.

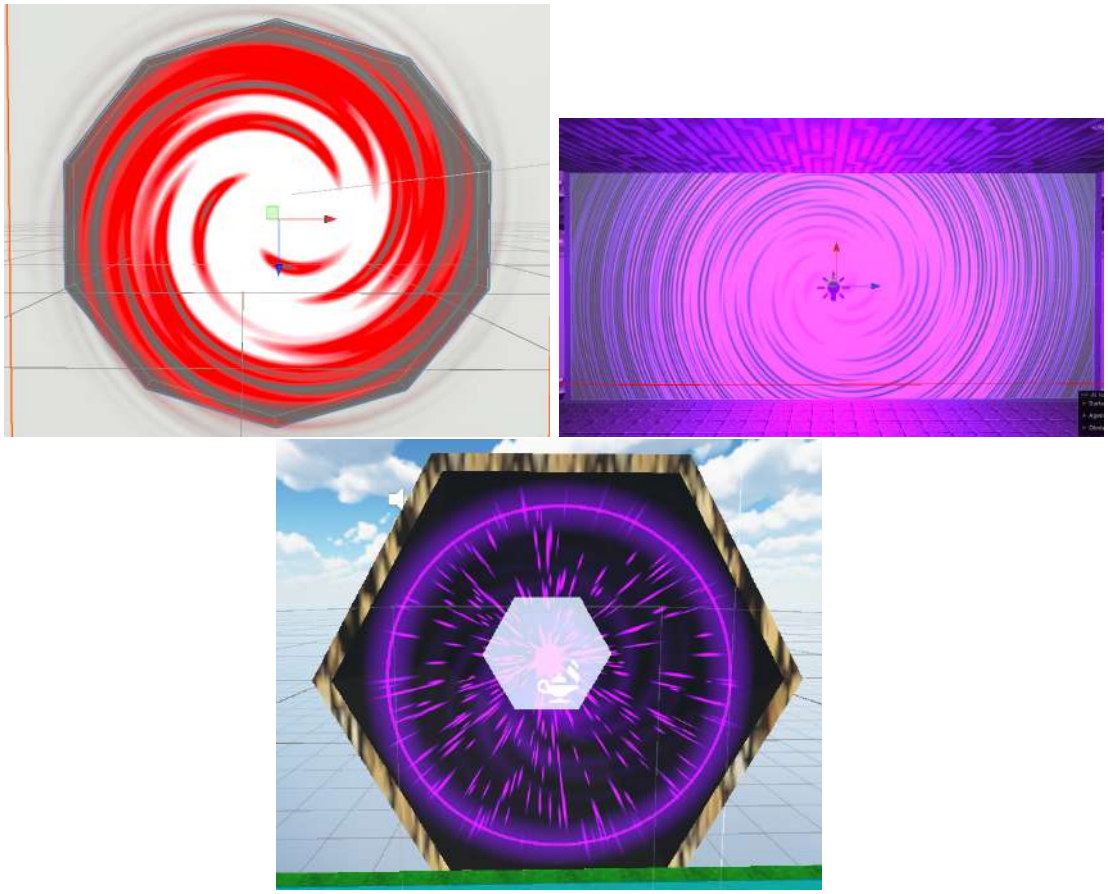
Figura 84 – Programação do Efeitos Através de Lógica Visual com Nós



Fonte: elaborado pelo autor (2024)

Com essa ferramenta da Unity foi possível criar três portais diferentes, 1 para cada nível, como mostra a Figura 85.

Figura 85 – Portais Criados Utilizando Visual Effect Graph



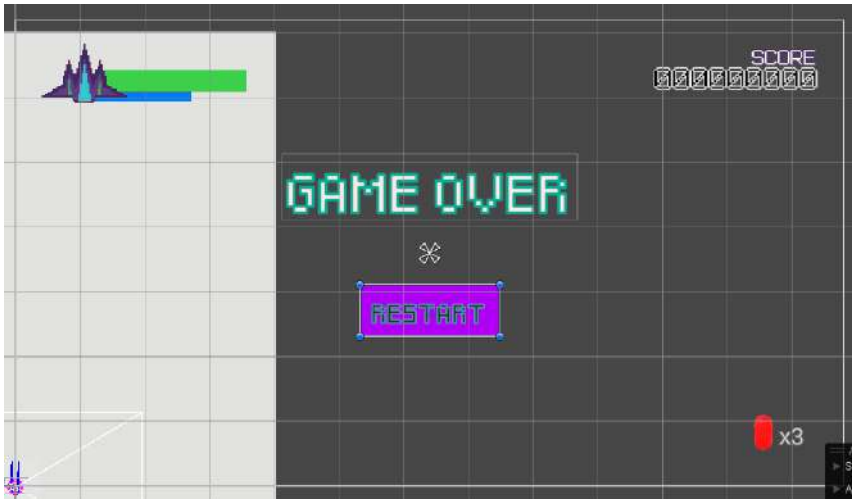
Fonte: elaborado pelo autor (2024)

5.6 Criação de Interfaces na Unity

A criação da interfaces dos níveis foi realizada utilizando o Canvas e recursos de interface fornecidos pela Unity para os desenvolvedores, além de também apoio de ferramentas externas como o site Pixilart, utilizado para desenhar alguns sprites de interface e o Figma para gerar algumas imagens com formatos especificos.

Para trabalhar com os elementos da interface, a Unity permite que o desenvolvedor utilize a perspectiva em duas dimensões, possibilitando a movimentação e o redimensionamento dos elementos sem a preocupação com os três eixos, como ilustrado na Figura 86.

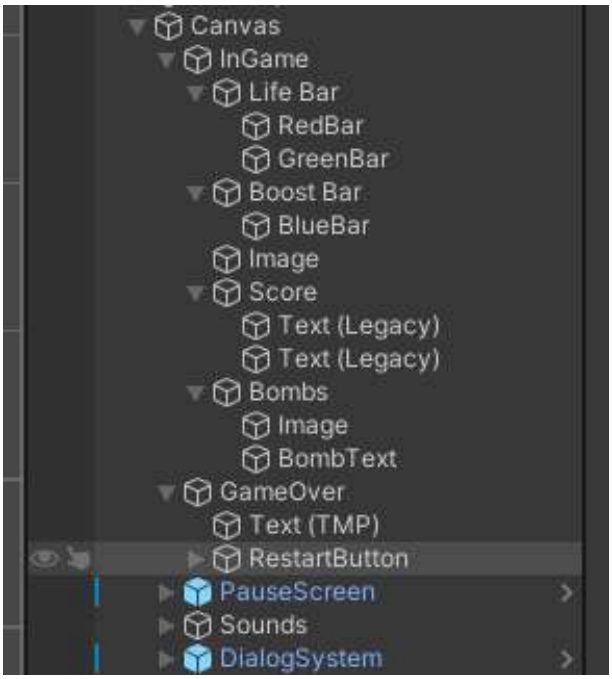
Figura 86 – Unity em Modo 2D para Manipulação de Elementos do Canvas



Fonte: elaborado pelo autor (2024)

No processo de personalização da interface, utilizamos botões, barras, imagens, textos e outros elementos do *Canvas*. Todos esses elementos podem ser organizados na janela de Hierarquia (Figura 87) e configurados tanto por meio de códigos quanto por componentes na Janela de Inspeção.

Figura 87 – Elementos do Canvas na Janela de Hierarquia

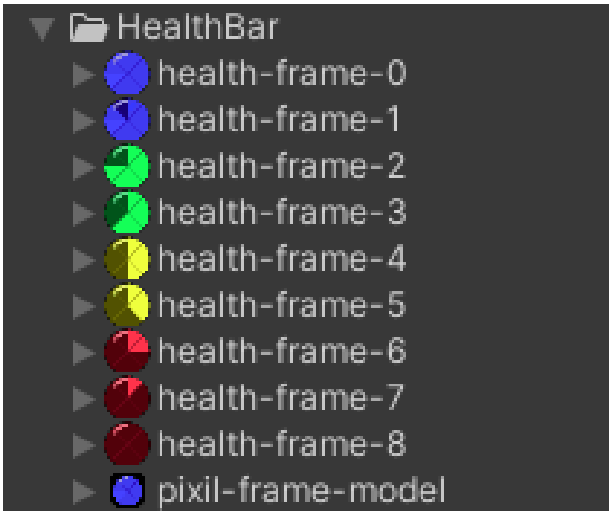


Fonte: elaborado pelo autor (2024)

Cada nível do jogo pedia por alguns elementos específicos para a criação de suas interfaces. No nível 1, utilizamos barras para ilustrar o valor da vida e energia do jogador. No nível 2, a interface foi mais simples, utilizando apenas textos conectados ao código. Já

no nível 3, a maior parte dos elementos foi composta por imagens posicionadas na tela, mas a representação da vida do jogador, por exemplo, é um conjunto de sprites desenhados em pixel art (Figura 88) e que se alternam conforme o valor numérico da vida do personagem aumentar ou diminuir.

Figura 88 – Sprites da Vida do Personagem Principal do Nível 3

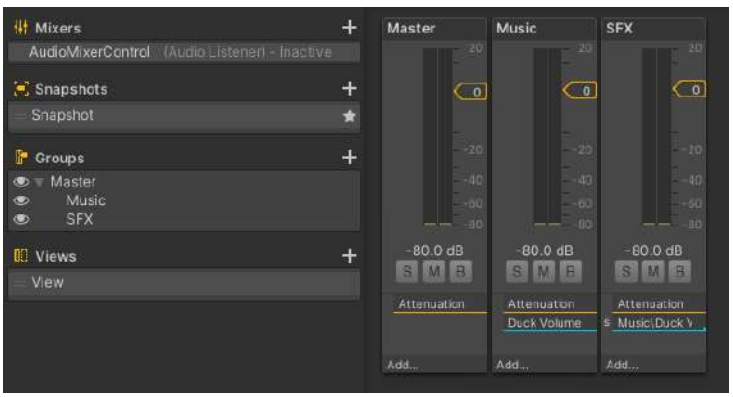


Fonte: elaborado pelo autor (2024)

5.7 Implementação de Sons

A implementação de sons é, em geral, um dos últimos aspectos trabalhados no desenvolvimento do projeto. Para isso, a Unity oferece os componentes **Audio Source** e **Audio Listener**, que são responsáveis por reproduzir e captar sons dentro da cena. Além disso, foi necessário utilizar a ferramenta **Audio Mixer**, que permite controlar múltiplas faixas de áudio simultaneamente (Figura 89), como no menu, onde o jogador pode ajustar o volume das músicas e dos efeitos sonoros do jogo.

Figura 89 – Janela de Gerenciamento do Audio Mixer



Fonte: elaborado pelo autor (2024)

6 DESAFIOS E ANÁLISE DE RESULTADOS

Este capítulo descreve os principais desafios e resultados obtidos ao final da etapa de desenvolvimento do jogo na Unity Engine, discutindo o resultado final do projeto, bem como o feedback das pessoas que testaram os protótipos de cada nível. Além disso, aborda os pontos em que o projeto poderia se aproximar mais de seus objetivos e apresenta ideias que poderiam ser implementadas em uma versão futura.

6.1 Principais desafios

Desde o início, este projeto apresentou um grande desafio. Embora seja baseado em jogos antigos, cuja reprodução pode ser facilitada pelas tecnologias da Unity Engine, esses jogos são resultados de grandes produções, com equipes multidisciplinares responsáveis por cada aspecto de sua criação. Assim, para um desenvolvedor iniciante, criar algo que se aproxime da qualidade desses jogos se mostrou um desafio significativo.

Outro fator relevante foi a inclusão de três tipos diferentes de jogos no projeto, o que aumentou consideravelmente sua complexidade. A cada novo nível desenvolvido, era necessário iniciar um novo processo de pesquisa e aprendizado de ferramentas e técnicas específicas. Além disso, a reutilização de código foi limitada, com exceção de algumas partes relacionadas à interface e menus, o que demandou mais tempo e esforço no desenvolvimento.

6.2 Resultados

Apesar da complexidade, o projeto alcançou resultados próximos das expectativas. O nível 1 foi o que mais se distanciou, pois, embora seguisse o estilo gráfico dos modelos 3D e de jogabilidade, alguns aspectos, como resolução e efeitos de pós-processamento, não foram aplicados de forma eficaz para aproximá-lo de um jogo 3D típico dos anos 90. Diferentemente dos níveis 2 e 3, que, tanto em visual quanto em jogabilidade, conseguiram atingir um alto nível de fidelidade em relação aos jogos que serviram como inspiração para o projeto.

Outro ponto importante alcançado foi criar um jogo mais coeso(seus menus, tutoriais e níveis, todos conectados criando um jogo completo e uniforme), criando uma experiência de níveis conectados, com sua própria narrativa, de forma que o jogador pudesse enxergá-lo mais como uma obra de entretenimento do que como um simples objeto de estudo. Foram inseridas telas de transição entre os níveis, diálogos que introduzem a história no início de cada fase e também menus principais e de pausa, contendo configurações gerais e informações sobre o projeto.

Cada nível foi desenvolvido com base em objetivos visuais e de jogabilidade

inspirados por análises dos jogos Star Fox, Doom e Super Mario 64. A implementação dessas características foi bem-sucedida, e os resultados podem ser vistos na Tabela 1, que lista as principais características implementadas em cada nível.

	Nível 1	**Nível 2**	**Nível 3**
Visual	Modelos 3D geométricos com poucos polígonos e com poucas cores.	Ambiente 3D texturizado, porém com personagens e itens aplicados como sprite dentro do cenário.	Cenário e personagens em 3D mesclados com sprites e texturizados.
Jogabilidade	Controle da Nave no eixo X e Y, Combate espacial, câmera que acompanha o jogador e segue um caminho pré-determinado.	Câmera em primeira pessoa, Combate com arma de fogo, exploração de cenários e coleta de itens.	Câmera em terceira pessoa, movimentação livre, câmera com movimentação automática e controle manual, inimigos com movimentação autônoma em terreno e cenário com desafios.

Tabela 1 – Características derivadas da análise implementadas no projeto

Por fim, foi possível criar interfaces que melhoraram o acesso às informações sobre os comandos do jogo, créditos dos recursos utilizados que não foram produzidos internamente, além das telas de transição, que mostram ao jogador a pontuação alcançada em cada nível. Esses detalhes contribuíram para tornar o jogo mais completo e uniforme.

6.3 Testes

Uma das abordagens adotadas durante o desenvolvimento do jogo foi a realização de testes controlados entre os níveis. À medida que cada nível se aproximava de sua versão final, uma versão jogável era construída para que algumas pessoas pudessem testar.

Os testes envolveram principalmente alunos do curso de engenharia da computação e amigos próximos, com cada nível sendo testado por 2 a 3 pessoas, totalizando 6 participantes. O objetivo desses testes era avaliar a jogabilidade de cada nível e garantir que outras pessoas, além do autor, conseguissem completar o nível testado, validando a funcionalidade dos níveis propostos.

De modo geral, os feedbacks foram positivos, embora também houvesse sugestões de melhorias. No nível 1, por exemplo, os comandos foram considerados um pouco confusos. Já no nível 2, houve debate sobre a possibilidade de o jogador controlar a câmera também na vertical. No entanto, como o objetivo era aproximar-se da jogabilidade de Doom (1993),

decidiu-se manter o controle como estava. O nível 3 foi, certamente, o que recebeu mais feedbacks negativos inicialmente, principalmente devido aos problemas na câmera, que, por ser automatizada, atrapalhava a experiência do jogador. Além disso, muitos jogadores relataram dificuldades em entender o que fazer na batalha final. Após esses feedbacks, foram realizadas correções que melhoraram consideravelmente o nível 3.

6.4 Trabalhos Futuros

Algumas melhorias podem ser implementadas no projeto para aprimorar a experiência de jogo. Entre elas, destaca-se a correção de alguns bugs relacionados à movimentação em certos pontos do jogo, ajustes no nível de dificuldade, e a adição de mais personagens inimigos, principalmente nos últimos níveis, que atualmente apresentam uma quantidade pequena de adversários.

Outra melhoria considerada durante o desenvolvimento do projeto é a criação de uma maior conexão entre os níveis. Um exemplo seria a implementação de um sistema no qual a pontuação obtida no nível anterior oferece bônus no nível seguinte, incentivando o jogador a eliminar inimigos e aumentando o desafio.

Realização de testes (planejados), para um público maior, buscando entender quais públicos seriam mais propícios a jogar esse estilo de jogo quer foi abordado neste projeto e entender possíveis problemas e melhorias para o jogo por completo.

Por fim, melhorias utilizando o sistema de pós-processamento de imagem da Unity. Aumentando a imersão durante o jogo, reforçando o aspecto visual de “jogo 3D antigo”, que foi um dos objetivos ao longo do desenvolvimento deste projeto na Unity Engine.

7 CONCLUSÃO

Este projeto apresentou um estudo aprofundado sobre o desenvolvimento de jogos 3D na década de 1990, com ênfase nos títulos icônicos Star Fox, Doom e Super Mario 64. Além de realizar uma análise detalhada dos aspectos visuais e de jogabilidade desses jogos, o trabalho propôs a criação de um jogo 3D inspirado nesses clássicos, utilizando a Unity Engine. Essa abordagem permitiu a exploração prática de diversas técnicas de modelagem, texturização e programação, com o objetivo de reproduzir o estilo característico dos jogos estudados. A recriação desses elementos se revelou o principal desafio deste trabalho.

A escolha da Unity Engine como principal ferramenta para o desenvolvimento do jogo mostrou-se uma decisão acertada. A plataforma oferece uma ampla gama de recursos, especialmente no trabalho com elementos 3D, além de possuir excelente suporte para elementos 2D, que foram amplamente utilizados nos jogos 3D da década de 1990. Isso permitiu recriar ambientes tridimensionais complexos de maneira eficiente e com menor exigência de desempenho.

Além da Unity, várias outras ferramentas foram indispensáveis ao longo do desenvolvimento. Seria praticamente inviável criar um jogo com essas características sem o auxílio de ferramentas externas, que não apenas viabilizaram o desenvolvimento do projeto, mas também auxiliaram na superação dos desafios enfrentados e nas soluções encontradas ao longo do processo dentro da Unity.

Embora o projeto tenha atingido muitos de seus objetivos, como a criação de um jogo coeso com elementos visuais e jogabilidade inspirada nos três jogos analisados, ainda há melhorias que podem ser implementadas. Entre elas, destaca-se o aprimoramento dos efeitos visuais e do pós-processamento, além da inclusão de uma maior variedade de personagens e elementos, conforme discutido no capítulo anterior.

Em suma, este projeto, além de contribuir para a preservação da memória de jogos clássicos, demonstra como é possível reproduzir aspectos de jogos 3D antigos dentro da Unity Engine. Ele também reforça a complexidade do desenvolvimento de jogos nos anos 90, ao analisar as técnicas utilizadas na criação dos títulos abordados, evidenciando a criatividade dos desenvolvedores mesmo diante das limitações tecnológicas da época. O jogo desenvolvido serve tanto como um tributo aos jogos da década de 1990 quanto como um exemplo prático das capacidades do desenvolvimento moderno.

Além das contribuições citadas, o projeto teve extrema relevância para o aprofundamento dos conhecimentos adquiridos em disciplinas da graduação, como Computação Gráfica e Projetos de Engenharia II. Essas disciplinas foram essenciais para o desenvolvimento do jogo, uma vez que abordam, respectivamente, conceitos de modelagem 3D e os processos de criação e desenvolvimento de jogos.

Por fim, espera-se que este estudo demonstre a capacidade de ferramentas como Unity e Blender, mesmo para pessoas com pouca experiência em desenvolvimento de jogos, e inspire futuros desenvolvedores. Ao mesmo tempo, ele abre espaço para a inovação e experimentação de novas ideias dentro do contexto acadêmico no ramo de desenvolvimento de jogos.

O código-fonte do projeto será disponibilizado de forma aberta através de um repositório público disponibilizado na plataforma *GitHub*(clique ou acesse: https://github.com/LucasAntonioC-137/RetroGame3D.git).

REFERÊNCIAS

ADAMMYHRE. **Unity 3D Rail Shooter**. 2023. Disponível em: <https://github.com/adammyhre/Rail-Shooter/tree/master>.

ADOBE. **Adobe Photoshop**. 2024. Disponível em: <https://www.adobe.com/br/products/photoshop.html>. Acesso em: 20 de julho 2024.

ARSALAN. **What Is Box Modeling? - ITS**. 2023. Disponível em: <https://it-s.com/what-is-box-modeling/>.

AUGUSTO, C. **Apple agora faz parte do Blender Development Fund e auxiliará no desenvolvimento do software**. 2021. Disponível em: <https://diolinux.com.br/softwares/apple-apoia-blender.html>.

BLENDER. **Blender**. 2024. Disponível em: <https://www.blender.org/about/>. Acesso em: 18 de julho 2024.

BLENDER. **Composição - Introdução**. 2024. Disponível em: <https://docs.blender.org/manual/pt/dev/compositing/introduction.html>. Acesso em: 26 de julho 2024.

CHEHIMI, F.; COULTON, P.; EDWARDS, R. Evolution of 3d games on mobile phones. In: **International Conference on Mobile Business (ICMB'05)**. [S.l.: s.n.], 2005. p. 173–179.

CONSULTING. **Indústria Brasileira de Games 2023**. 2023. Disponível em: <https://www.abragames.org/uploads/5/6/8/0/56805537/2023_fact_sheet_v1.6.2_-_ptbr_version.pdf>.

DANIIL, V. **Pistol 43 Tactical**. 2023. Disponível em: <https://sketchfab.com/3d-models/pistol-43-tactical-fps-animations-1a4c9d15818a4522aedbeb70919a17da>.

DANSIE, J. Bachelor's Thesis, **Game Development in Unity**. Finlândia: [s.n.], 2013. 34 p.

DUDE984. **Star Fox (game)/Gallery**. 2024. Disponível em: <https://starfox.fandom.com/wiki/Star_Fox_(game)/Gallery?file=Andross_NES_em.gif>. Acesso em: 18 de junho 2024.

FAHS, T. **Exploring the Freescape**. 2012. Disponível em: <https://www.ign.com/articles/2008/10/22/exploring-the-freescape>.

FANDOM. **Ultima Underworld: The Stygian Abyss**. 2023. Disponível em: <https://ultima.fandom.com/wiki/Ultima_Underworld:_The_Stygian_Abyss>. Acesso em: 11 de junho 2024.

FIGMA. **Figma Design**. 2024. Disponível em: <https://www.figma.com/pt-br/design/>. Acesso em: 30 de julho 2024.

FRANCFORT, E. **1973**. 2024. Disponível em: <https://www.museudatv.com.br/cronologias/a-tv-no-brasil/1973/>.

GARRETT, K. **Star Fox**. 2006. Disponível em: <https://www.mobygames.com/game/6629/star-fox/screenshots/snes/164494/>. Acesso em: 18 de junho 2024.

GERARDI, M. **How and why Super Mario 64 changed gaming culture**. 2016. Disponível em: <https://www.avclub.com/how-and-why-super-mario-64-changed-gaming-culture-1798252181>.

GITHUB. **About GitHub and Git**. 2024. Disponível em: <https://docs.github.com/en/get-started/start-your-journey/about-github-and-git>. Acesso em: 20 de julho 2024.

GOGAMERS; SXGROUP. **Pesquisa Game Brasil 2023**. 2023. Disponível em: <https://materiais.pesquisagamebrasil.com.br/2023-painel-gratuito-pgb10-anos>.

GONçALVES, N. **Texture painting (pintura de texturas)**. 2014. Disponível em: <https://nafergo.github.io/manual-livre-blender/texture_painting.html>.

GOODALL, R. **Concept to Console: a history of 'Super Mario 64'**. 2021. Disponível em: <https://theboar.org/2021/01/concept-to-console-super-mario-64/>.

GREGORY, J. In: _____. **Game engine Architecture**. [S.l.]: New York:CRC Press, 2018. p. 3.

HALL, M. **SGI**. 2024. Disponível em: <https://www.britannica.com/money/SGI>.

HALTSONEN, J. Bachelor's Thesis, **GUIDE TO WRITING A GAME DESIGN DOCUMENT**. 2015. 23 p.

HEGINBOTHAM, C. **What is 3D Digital Sculpting?** 2024. Disponível em: <https://conceptartempire.com/what-is-3d-sculpting/>.

IDSOFTWARE. **File:Cyberdemon model photo.jpg**. 2024. Disponível em: <https://doomwiki.org/wiki/File:Cyberdemon_model_photo.jpg>. Acesso em: 02 de julho 2024.

IDSOFTWARE. **Steam: Doom (1993)**. 2024. Disponível em: <https://store.steampowered.com/app/2280/DOOM_1993/>. Acesso em: 02 de julho 2024.

IGDB. **Star Fox**. 2024. Disponível em: <https://www.igdb.com/games/star-fox>. Acesso em: 15 de junho 2024.

KAUFMAN, F. **Visão geral de jogos 3D e recomendação de jogos para jogadores ávidos de jogos 3D**. 2023. Disponível em: <https://www.vidmore.com/pt/knowledge/3d-games/>.

KUSHNER david. **Masters of Doom: How Two Guys Created an Empire and Transformed Pop Culture**. [S.l.]: Random House Trade Paperbacks, 2004.

LEITE, L. C. **Jogos Eletrônicos Multi-plataforma: Compreendendo As Plataformas De Jogo E Seus Jogos Através De Uma Análise Em Design**. Rio de Janeiro: [s.n.], 2006. 271 p.

LELES, D. **Imagem referência Photoshop e Max - 3D1**. 2010. Disponível em: <https://3d1.com.br/tutoriais/outros-softs-3d/edicao-de-imagens/imagem-referncia-photoshop-e-max>.

LEWIS, J. J. M. Serious computationalresults are derived fromcomputer-based games. In: _____. **Game engines in scientific research**. [S.l.]: Communications of the ACM, 2002. v. 45, n. 1, p. 27–31.

LOWOOD, H. E. **Doom**. 2024. Disponível em: <https://www.britannica.com/topic/Doom>.

MALLO. **DRILLER, ZX SPECTRUM**. 2018. Disponível em: <https://thekingofgrabs.com/2018/07/24/driller-zx-spectrum/>.

MARCONI, G. S. **ANÁLISE E DESENVOLVIMENTO DE UM JOGO DIGITAL**. 2023. Disponível em: <https://www.adobe.com/br/products/photoshop.html>. Acesso em: 20 de julho 2024.

MARTIAN, L. **Retro Texture Pack**. 2022. Disponível em: <https://little-martian.itch.io/retro-texture-pack>.

MATOS Évilin. **Metodologia ágil: definição, benefícios e como implementar na sua empresa**. 2022. Disponível em: <https://blog.runrun.it/metodologia-agil/>.

MCFERRAN, D. **Born slippy: the making of Star Fox. From the archive: ex-Argonaut staff reveal the story behind the revolutionary SNES hit.** 2014. Disponível em: <https://www.eurogamer.net/born-slippy-the-making-of-star-fox>.

MICROSOFT. **What is Visual Studio?** 2024. Disponível em: <https://learn.microsoft.com/en-us/visualstudio/get-started/visual-studio-ide?view=vs-2022#get-started>. Acesso em: 30 de julho 2024.

MIRANDA, F.; STADZISZ, P. Jogo digital: definição do termo. 05 2018.

Mix and Jam. **Star Fox's Rail Movement**. 2019. Disponível em: <https://github.com/mixandjam/StarFox-RailMovement>.

MIXAMO. **Mixamo, Get animated.** 2024. Disponível em: <https://www.mixamo.com/#/>. Acesso em: 20 de julho 2024.

MOHD, T. K. et al. Analyzing strengths and weaknesses of modern game engines. **International Journal of Computer Theory and Engineering**, v. 15, p. 54–60, 01 2023.

MOTTA, J. T. J. R. L. Short game design document (sgdd). **SBC – Proceedings of SBGames**, 2013.

MUSSE, S. **Modelagem de Objetos**. 2017. Disponível em: <https://www.inf.pucrs.br/~smusse/CG/PDF_2017_1/ModelagemNew.pdf>.

NINTENDO. **Super Mario 64**. Disponível em: <https://www.nintendo.com/en-gb/Games/Nintendo-64/Super-Mario-64-269745.html#Overview>. Acesso em: 05 de julho 2024.

NINTENDO. **Nintendo 64**. 2024. Disponível em: <https://www.nintendo.com/en-gb/Hardware/Nintendo-History/Nintendo-64/Nintendo-64-625959.html>. Acesso em: 04 de julho 2024.

NINTENDO. **Nintendo History**. 2024. Disponível em: <https://www.nintendo.com/en-gb/Hardware/Nintendo-History/Nintendo-History-625945.html>. Acesso em: 11 de junho 2024.

NINTENDO. **Super Mario 64**. 2024. Disponível em: <https://www.nintendo.com/en-gb/Games/Nintendo-64/Super-Mario-64-269745.html#Gallery>. Acesso em: 05 de julho 2024.

NINTENDO-BLAST. **Star Fox (SNES): 22 anos da primeira jornada espacial que mudaria o mundo dos videogames**. 2015. Disponível em: <https://www.nintendoblast.com.br/2015/04/star-fox-snes-historia.html>.

ODDO, M. V. **25 Years Later, 'Super Mario 64' Is Still One of the Best 3D Platformers Ever**. 2021. Disponível em: <https://collider.com/why-super-mario-64-is-the-best-game-ever/>.

PEDDIE, J. **Famous Graphics Chips: Nintendo 64**. 2020. Disponível em: <https://www.computer.org/publications/tech-news/nintendo-64>.

PIXILART. **Pixilart - Sobre**. 2024. Disponível em: <https://www.pixilart.com/about>. Acesso em: 29 de julho 2024.

PLAMZDOOM SIRKINSELLA98, f. A.-s. A. r. H. j. M. **Star Fox**. 2023. Disponível em: <https://www.giantbomb.com/star-fox/3030-3984/>.

PLUS, P. **The evolution of 3D games**. 2010. Disponível em: <https://www.techradar.com/news/gaming/the-evolution-of-3d-games-700995>.

REIS, R. O. TCC (Graduação), **LABORATÓRIO VIRTUAL DE ELETRÔNICA**. 2013. 64 p.

RESOURCE, T. V. **Nintendo 64 - Super Mario 64 - The Models Resource**. 2024. Disponível em: <https://www.models-resource.com/nintendo_64/supermario64/>.

SANTOS, C. F. de A. **O que são materiais e texturas em um software 3D?** 2022. Disponível em: <https://www.alura.com.br/artigos/o-que-sao-materiais-texturas-software-3d>.

SANTOS, J. S. dos. **NUVENS VIRTUAIS COMO EXEMPLO DE TÉCNICAS DE JOGOS PARA GRÁFICOS TRIDIMENSIONAIS EM TEMPO REAL**. Florianópolis: [s.n.], 2004. 125 p.

SILVA, A. W. O. da. TCC (Graduação), **LABORATORIO VIRTUAL DE MICROBIOLOGIA**. 2023. 74 p.

SIMMONS, J. **The global landscape of game development: A quantitative analysis**. 2024. Disponível em: <https://stepofweb.com/game-developer-count-worldwide/>.

SOFTWARE, I. **Wolfenstein 3D**. 2024. Disponível em: <https://store.steampowered.com/app/2270/Wolfenstein_3D/>. Acesso em: 12 de junho 2024.

SONYMASTER. **Maze War - Jogo de 1973 foi o primeiro FPS de todos os tempos**. 2021. Disponível em: <https://www.sega-brasil.com.br/forum/viewtopic.php?t=6567>.

TACHYON ARURU-SAN, P. **Super FX chip**. 2023. Disponível em: <https://www.giantbomb.com/super-fx-chip/3015-7544/>.

TRELLO. **Trello - NOÇÕES BÁSICAS DO TRELLO**. 2024. Disponível em: <https://trello.com/pt-BR/>. Acesso em: 29 de julho 2024.

TROPES, T. **Video Game / Spasim**. 2022. Disponível em: <https://tvtropes.org/pmwiki/pmwiki.php/VideoGame/Spasim>.

TROPES, T. **Video Game / Battlezone (1980)**. 2023. Disponível em: <https://tvtropes.org/pmwiki/pmwiki.php/VideoGame/Battlezone1980>.

TYLER, D. **The Video Game Development Essentials Guide**. 2023. Disponível em: <https://www.gamedesigning.org/video-game-development/>.

UNITY. **About Splines**. 2024. Disponível em: <https://docs.unity3d.com/Packages/com.unity.splines@2.0/manual/index.html>.

UNITY. **Rotation and orientation in Unity**. 2024. Disponível em: <https://docs.unity3d.com/Manual/QuaternionAndEulerRotationsInUnity.html>.

UNITY. **Unity para iniciantes**. 2024. Disponível em: <https://unity.com/pt/learn/get-started>. Acesso em: 19 de julho 2024.

UNITY. **Unity's Interface**. 2024. Disponível em: <https://docs.unity3d.com/Manual/UsingTheEditor.html>. Acesso em: 21 de agosto 2024.

UNITY. **Using dolly paths**. 2024. Disponível em: <https://docs.unity3d.com/Packages/com.unity.cinemachine@2.8/manual/CinemachineDolly.html>.

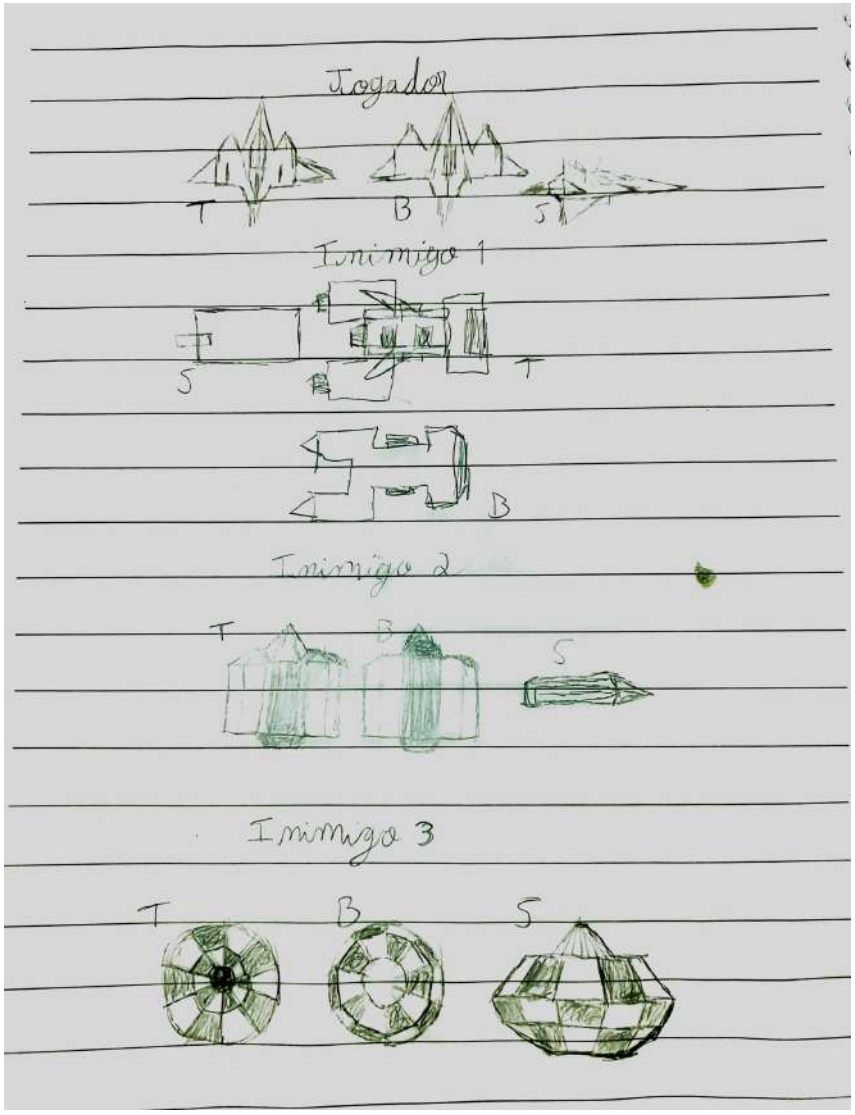
VOORHEES, G. A.; CALL, J.; WHITLOCK, K. **Guns, Grenades, and Grunts: First-Person Shooter Games**. New York, NY: Continuum, 2012. Uma análise abrangente dos jogos de tiro em primeira pessoa e seu impacto na cultura e sociedade.

WAKKA, V. **Mercado de games agora vale mais que indústrias de música e cinema juntas**. 2021. Disponível em: <https://canaltech.com.br/games/mercado-de-games-agora-vale-mais-que-industrias-de-musica-e-cinema-juntas-179455/#google_vignette>.

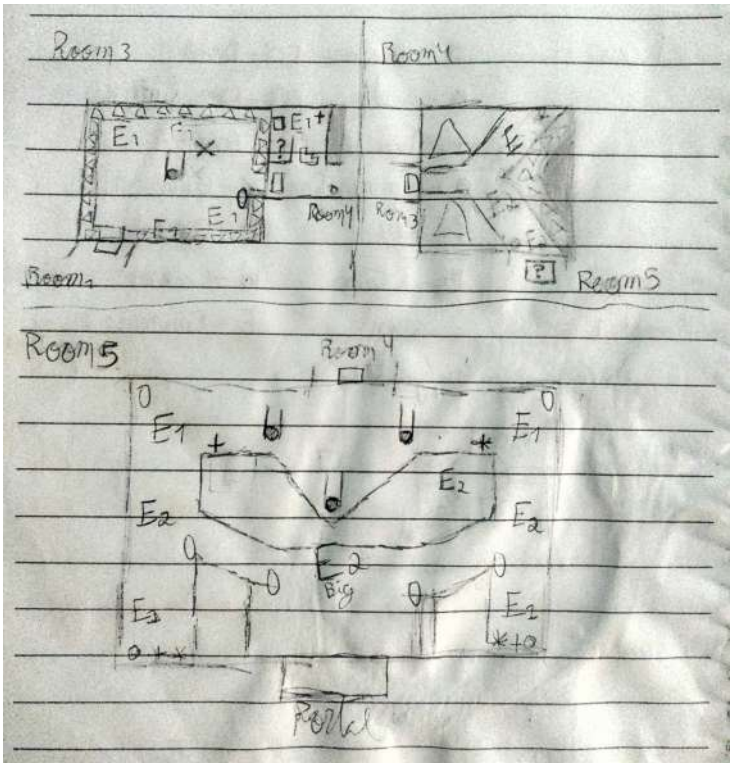
WALLICH, P. The invention of battlezone. **IEEE Sprectrum**, 2022.

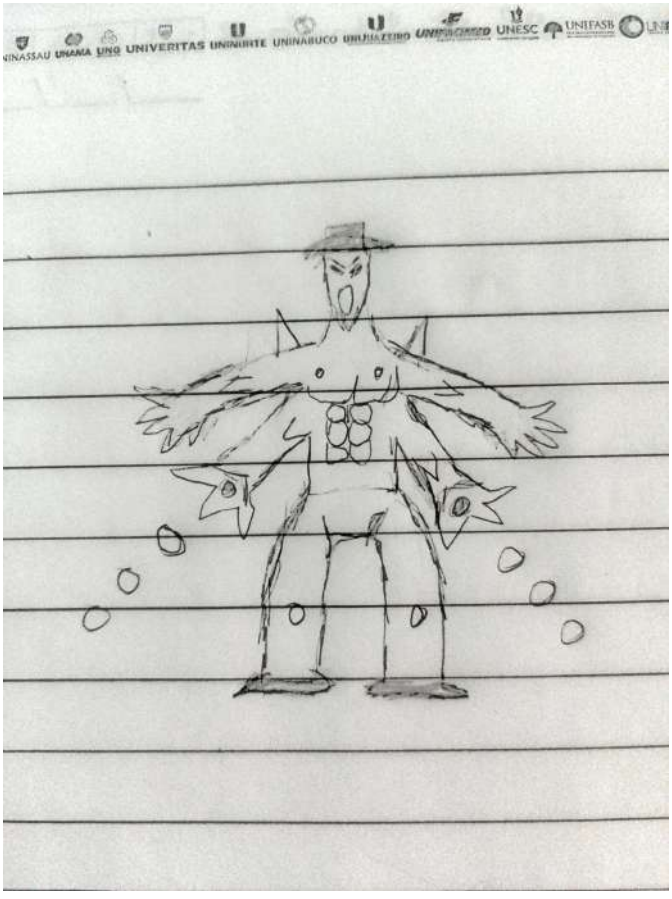
XIA, P. Bachelor's Thesis, **3D Game Development with Unity A Case Study: A First-Person Shooter (FPS) Game**. Finlândia: [s.n.], 2014. 58 p.

APÊNDICE A – DESIGNS INICIAIS DE PERSONAGENS FEITOS NO PAPEL PARA O NÍVEL 1

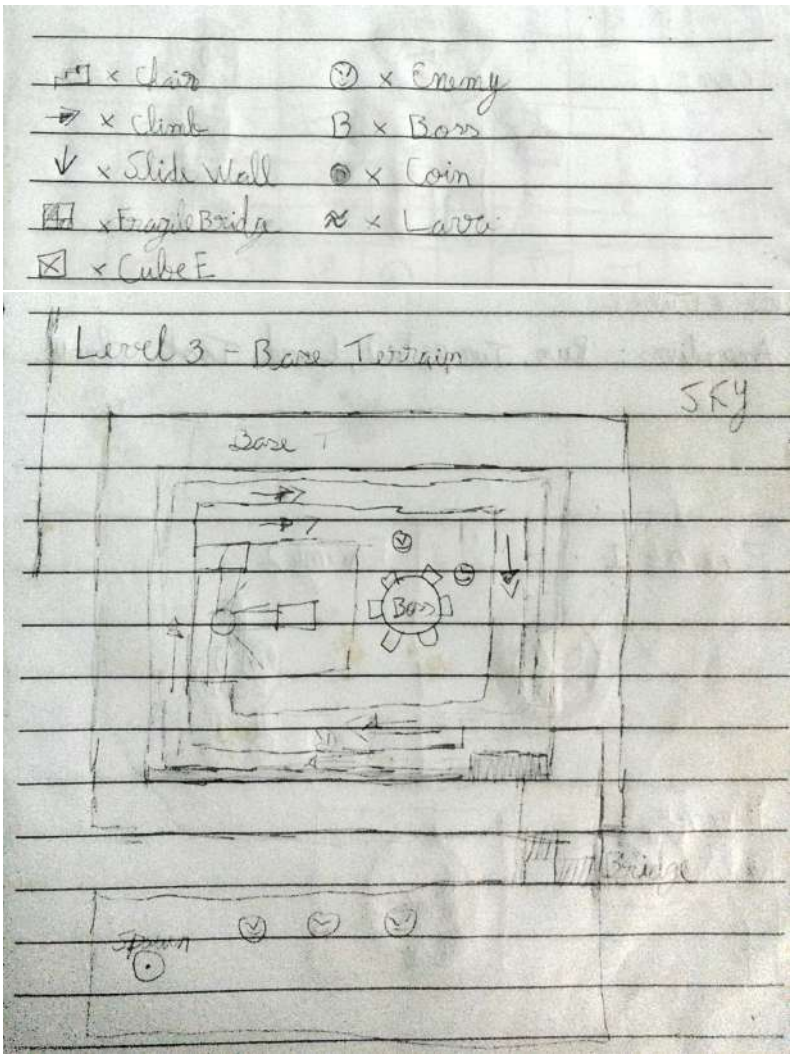


APÊNDICE B – PLANTAS DE TERRENOS E DESIGN DE INIMIGO DO NÍVEL 2 FEITOS NO PAPEL





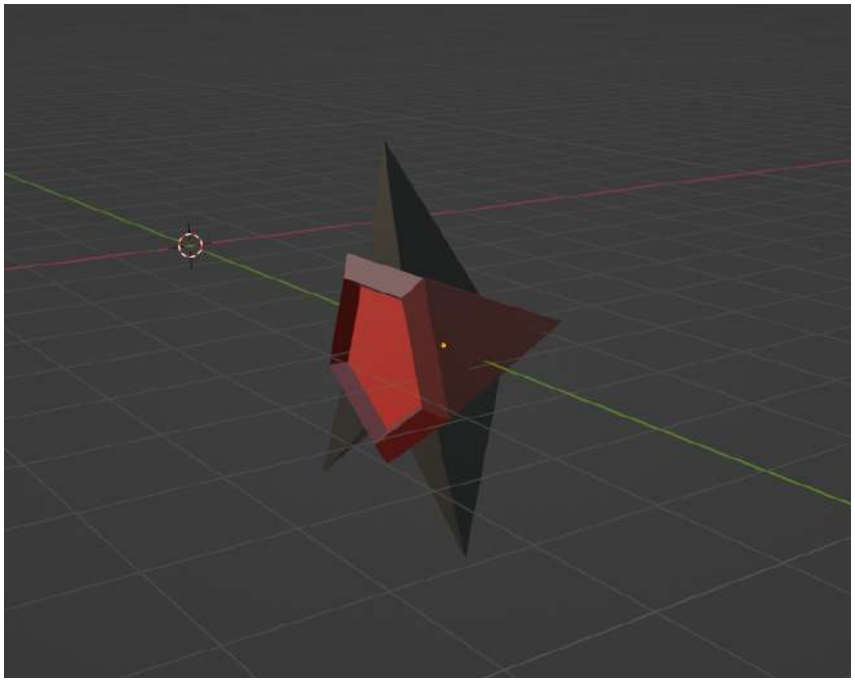
APÊNDICE C – PLANTA DO TERRENO DO NÍVEL 3



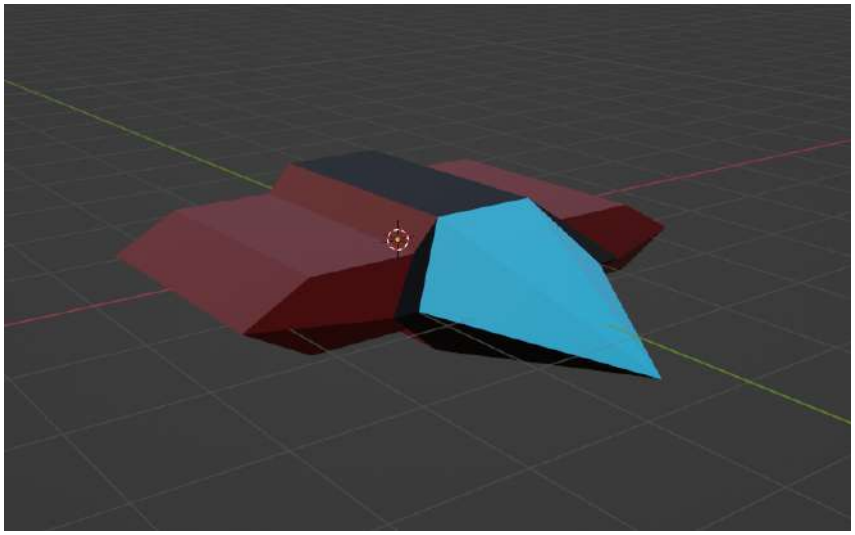
APÊNDICE D – TELAS CRIADAS NO FIGMA PARA O MENU PRINCIPAL

APÊNDICE E – MODELOS 3D DE PERSONAGENS INIMIGOS DO NÍVEL 1

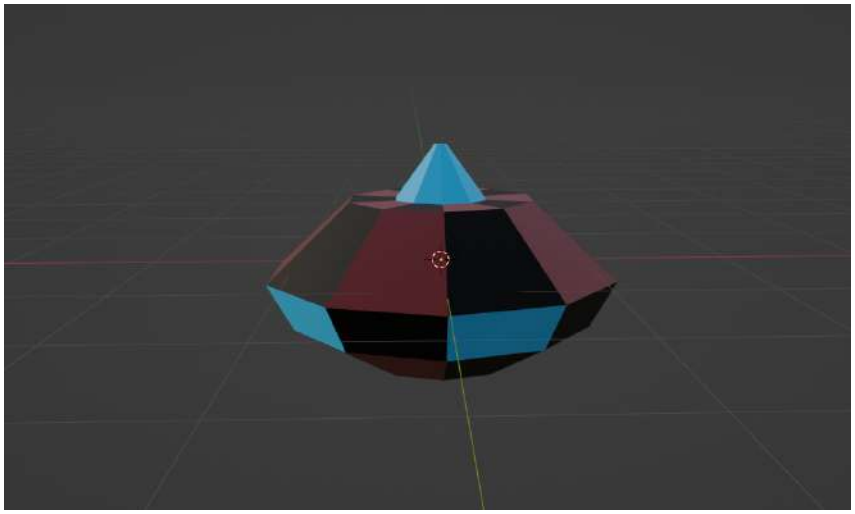
Modelo 3D do Inimigo 1 “Nave Filho”



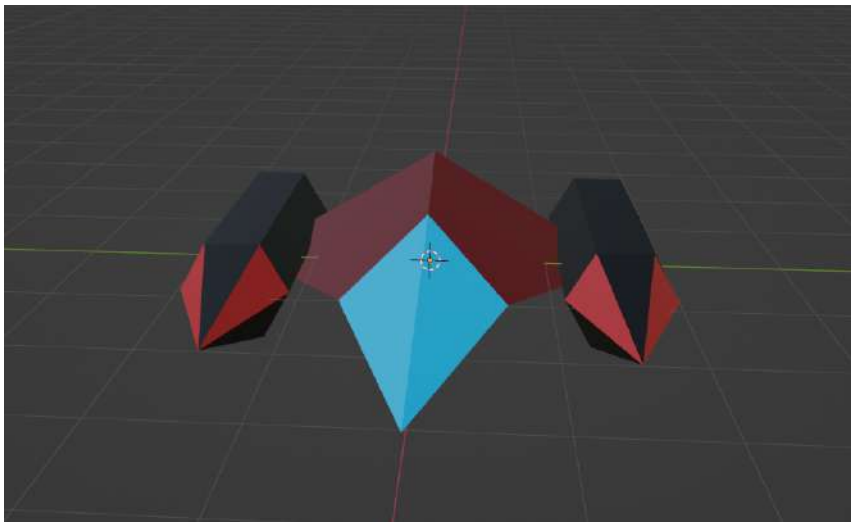
Modelo 3D do Inimigo 2



Modelo 3D do Inimigo 3



Modelo 3D do Inimigo 4



APÊNDICE F – MÉTODOS DO CÓDIGO “CAMERAFOLLOW”

Listing F.1 – Métodos do Código “CameraFollow”

```
1     void Update()
2     {
3         if (!Application.isPlaying)
4         {
5             transform.localPosition = offset;
6         }
7
8         FollowTarget(target);
9     }
10
11    void LateUpdate()
12    {
13        Vector3 localPos = transform.localPosition;
14
15        transform.localPosition = new Vector3(Mathf.Clamp(localPos.x, -
              limits.x, limits.x), Mathf.Clamp(localPos.y, -limits.y,
              limits.y), localPos.z);
16    }
17
18    public void FollowTarget(Transform t)
19    {
20        Vector3 localPos = transform.localPosition;
21        Vector3 targetLocalPos = t.transform.localPosition;
22        transform.localPosition = Vector3.SmoothDamp(localPos, new
              Vector3(targetLocalPos.x + offset.x, targetLocalPos.y +
              offset.y, localPos.z), ref velocity, smoothTime);
23    }
```