CURSO TÉCNICO EM INFORMÁTICA INTEGRADO AO ENSINO MÉDIO

ARTHUR CASTILHOS LEÃO

DESENVOLVIMENTO DE JOGOS DIGITAIS COM UNITY:

UMA INTRODUÇÃO

URUGUAIANA

2025

ARTHUR CASTILHOS LEÃO

DESENVOLVIMENTO DE JOGOS DIGITAIS COM UNITY:
UMA INTRODUÇÃO

Trabalho de Conclusão de Curso apresentado ao Curso Técnico em Informática Integrado ao Ensino Médio do Campus Uruguaiana do Instituto Federal de Educação, Ciência e Tecnologia Farroupilha como requisito parcial para a obtenção do título de Técnico em Informática.

Orientadores:

Eduardo Henrique Spies

URUGUAIANA

2025

Sobrenome do autor, nome do autor.

Título : Subtítulo(se houver) / Nome e sobrenome. — 2023.

[qtd. de folhas] f.

Trabalho de Conclusão de Curso Técnico – Instituto Federal de Educação, Ciência e Tecnologia Farroupilha, Uruguaiana, 2023.

1. palavra-chave. 2. [segunda entrada de assunto]. 3. [terceira entrada de assunto]. I. Título.

CDD [número da CDD].

ARTHUR CASTILHOS LEÃO

DESENVOLVIMENTO DE JOGOS DIGITAIS COM UNITY:
UMA INTRODUÇÃO

Trabalho de Conclusão de Curso apresentado ao Curso Técnico em Informática Integrado ao Ensino Médio do Campus Avançado Uruguaiana do Instituto Federal de Educação, Ciência e Tecnologia Farroupilha como requisito parcial para a obtenção do título de Técnico em Informática.

Este trabalho foi defendido e aprovado pela banca em DD/MM/AAAA.

BANCA EXAMINADORA

_____
Prof./Prof.ª Dr./Dr.ª Ms. Eduardo Henrique Spies

_____
Prof./Prof.ª Dr./Dr.ª Ms. Anderson Mendes Rocha

_____
Prof./Prof.ª Dr./Dr.ª Ms.
Guilherme Bardemaker Bernardi

Dedico este trabalho aos meus pais, que sempre me motivaram aos estudos acima de tudo, e aos amigos que me apoiaram nessa jornada.

AGRADECIMENTOS

Agradeço aos meus pais, que sempre lutaram para mim ter as melhores oportunidades que eles foram capazes de dar, aos meus amigos que sempre me ajudaram e apoiaram quando necessitei, aos professores por nunca desistirem de seus aluno, e a Steve Russell, por em 1962 ter criado *Spacewar!,* o primeiro videogame da história.

“O poder da inteligência é mais forte que o poder material”

DIÓGENES

"Nunca se esqueça de quem você é, porque é certo que o mundo não se esquecerá. Faça disso sua força. Assim, não poderá ser nunca a sua franqueza.”

GEORGE R. R. MARTIN, A GUERRA DOS TRONOS

RESUMO

Este Trabalho de Conclusão de Curso apresenta o desenvolvimento de um projeto com foco na introdução ao desenvolvimento de jogos digitais utilizando a engine Unity. O objetivo principal do trabalho é relatar a experiência de aprendizado de um estudante da área de Informática ao explorar os conceitos fundamentais do game design e da programação aplicada a jogos, mesmo não sendo um conteúdo previsto diretamente na ementa do curso. Para isso, foi desenvolvido um jogo teste em estilo 2D top-down, que contempla mecânicas básicas como movimentação do personagem, sistema de ataque, detecção de colisões, uso de animações e troca de cenas. O desenvolvimento do projeto permitiu a aplicação prática de conceitos como programação orientada a objetos em C#, uso do Input System, controle de animações por meio do Animator, gerenciamento de física 2D e depuração de erros. O trabalho evidencia que o desenvolvimento de jogos pode ser uma ferramenta eficaz para consolidar conhecimentos de programação e lógica computacional, além de estimular a autonomia, a resolução de problemas e o aprendizado prático de novas tecnologias.

Palavras-chave: Desenvolvimento de jogos. Unity. Programação. Game design. Aprendizagem prática.

ABSTRACT

This Final Course Project presents the development of a project focused on introducing digital game development using the Unity engine. The main objective of this work is to report the learning experience of an Information Technology student while exploring fundamental concepts of game design and game programming, even though this content is not directly included in the course curriculum. To achieve this goal, a test game was developed in a 2D top-down style, featuring basic mechanics such as character movement, attack system, collision detection, animation control, and scene transitions. The development process enabled the practical application of concepts such as object-oriented programming in C#, use of the Unity Input System, animation management through the Animator, 2D physics handling, and debugging techniques. The project demonstrates that game development can be an effective approach to strengthening programming skills, computational logic, problem-solving abilities, and autonomous learning through practical experimentation with new technologies.

Keywords: Game development. Unity. Programming. Game design. Practical learning.

LISTA DE ILUSTRAÇÕES

LISTA DE ABREVIATURAS E SIGLAS

IFFAR	Instituto Federal de Educação, Ciência e Tecnologia Farroupilha

SUMÁRIO

1 INTRODUÇÃO

O desenvolvimento de jogos digitais envolve a integração de múltiplas áreas do conhecimento, como programação, lógica computacional, design de sistemas interativos e experiência do usuário. Com o avanço das engines de desenvolvimento, como a Unity, tornou-se possível criar jogos e protótipos interativos de forma mais acessível, ao mesmo tempo em que se mantém contato com conceitos técnicos utilizados profissionalmente na indústria de jogos.

Embora o desenvolvimento de jogos digitais não faça parte da ementa curricular do curso de Informática, essa área se apresenta como um campo fértil para a aplicação prática de diversos conteúdos fundamentais da formação técnica. Nesse sentido, este Trabalho de Conclusão de Curso propõe uma jornada de aprendizado voltada à introdução ao desenvolvimento de jogos na engine Unity, com ênfase na exploração de conceitos de game design e programação, e não na criação de um jogo completo ou comercial.

O projeto consiste no desenvolvimento de um jogo-teste, concebido para demonstrar e aplicar conceitos essenciais do desenvolvimento de jogos. Ao longo dessa jornada, são exploradas mecânicas básicas como movimentação, animação de personagem, interação com inimigos, colisões e criação de cenários, escolhidas por seu valor didático e por representarem fundamentos recorrentes em diversos gêneros de jogos.

Dessa forma, o trabalho se caracteriza como um relato de experiência de aprendizado, no qual o processo de desenvolvimento assume papel central. São apresentados os conceitos estudados, as decisões de design adotadas, as dificuldades encontradas e as soluções implementadas, buscando demonstrar como a engine Unity pode ser utilizada como ferramenta introdutória para estudantes interessados em ingressar no universo do desenvolvimento de jogos digitais.

1.1 JUSTIFICATIVA

A escolha de desenvolver um projeto voltado à introdução ao desenvolvimento de jogos digitais se justifica pela capacidade dessa área de integrar, de forma prática, diversos conceitos essenciais da formação em Informática. Programação orientada a objetos, lógica computacional, organização de projetos, interfaces gráficas e resolução de problemas são conteúdos que encontram no desenvolvimento de jogos um ambiente propício para experimentação e aprendizagem.

Ao adotar a engine Unity como ferramenta principal, o projeto permite o contato direto com uma plataforma amplamente utilizada no mercado e no meio acadêmico, favorecendo a compreensão de conceitos como arquitetura baseada em componentes, gerenciamento de cenas, sistemas de entrada e controle de animações. Esse contato contribui para ampliar a visão do estudante sobre possibilidades de atuação profissional e áreas correlatas à sua formação.

Além disso, o foco do projeto no processo de aprendizado reforça seu caráter pedagógico. O jogo-teste desenvolvido funciona como um artefato didático, cuja principal finalidade é servir de demonstração dos conceitos de game design e desenvolvimento na Unity. Assim, o trabalho demonstra que o desenvolvimento de jogos pode ser utilizado como uma ferramenta complementar à formação técnica, mesmo quando não previsto diretamente na matriz curricular do curso.

Dessa forma, o projeto se justifica como uma experiência formativa que documenta o percurso de aprendizagem de um aluno de Informática no contato inicial com o desenvolvimento de jogos digitais, contribuindo para a reflexão sobre metodologias práticas de ensino e aprendizagem na área da computação.

1.2 OBJETIVOS

1.2.1 Objetivo Geral

Explorar mecânicas e documentar os conceitos fundamentais do game design e do desenvolvimento de jogos digitais na engine Unity.

1.2.2 Objetivos Específicos

- Estudar os fundamentos da engine Unity;
- Explorar os conceitos iniciais de game design e como aplicá-los a um projeto;
- Registrar e analisar as dificuldades, soluções e aprendizados obtidos durante o desenvolvimento;
- Compreender e utilizar princípios da Programação aprendida no curso no contexto do desenvolvimento de jogos.

1.3 METODOLOGIA

Definição o tema:

A escolha de desenvolver um jogo vem do interesse pessoal de trabalhar com o desenvolvimento de jogos. Também foram definidos o estilo e as mecânicas que seriam exploradas. Então, foi definido que o projeto se tratará de um pequeno jogo 2D em pixel art[1], com visão top-down[2] com mecânicas essenciais e que estão presentes na maioria dos videogames, como atacar, ser atacado por inimigos e balões de diálogo.

[1] Pixel art é uma forma de arte digital onde imagens são criadas e editadas pixel por pixel (os menores pontos de uma imagem digital), resultando em um visual característico, nostálgico e que remete aos primeiros videogames e computadores.

[2] Visão top-down em videogames é uma perspectiva onde a câmera fica posicionada acima do personagem, olhando para baixo, como um mapa aéreo, permitindo ao jogador ver o ambiente, inimigos e obstáculos de forma ampla.

Pesquisa de ferramentas de desenvolvimento:

Investigação sobre as ferramentas técnicas disponíveis. Para o desenvolvimento de uma engine[3] (Motor gráfico), e primeiramente, o Godot[4] era a engine escolhida, porém, devido a uma questão técnica, a Unity foi escolhida para o desenvolvimento deste projeto, pois, ao contrário do Godot, a Unity possui diversas sprites[5] e assets[6] já existentes disponíveis tanto na própria Unity, quanto online, o que facilita o processo da criação de elementos visuais do jogo.

Pesquisa de trabalhos relacionados:

Análise de TCCs e projetos anteriores do IFFAR e outras instituições com propostas semelhantes, a fim de compreender abordagens, estruturas e resultados obtidos.

Pesquisa de sistemas semelhantes:

Estudo de jogos comerciais e independentes com elementos parecidos, como narrativa textual, exploração e combate, para inspiração de mecânicas e estrutura de design.

Desenvolvimento do jogo:

Etapa prática de construção do jogo, que inclui a escolha dos sprites, criação do cenário, movimentação, ataque, colisões, animações e inimigos dentro do jogo.

---

[3] Uma engine é conjunto de componentes ou sistemas que executam uma tarefa específica dentro de um software ou aplicação

[4] Godot é um motor de jogo livre e de código aberto, utilizado para criar jogos 2D e 3D, além de outros tipos de aplicações

[5] Sprite é uma imagem ou animação bidimensional que é integrada a uma cena ou ambiente maior.

[6] Assets são todos os recursos e componentes (gráficos 2D/3D, sons, músicas) usados para construir um jogo, podendo ser criados por desenvolvedores ou comprados/baixados de lojas de assets, agilizando o desenvolvimento.

2 REFERENCIAL TEÓRICO

Neste capítulo, será apresentada a base conceitual utilizada no projeto, definindo os conceitos e teorias centrais presentes no trabalho.

2.1 TRABALHOS RELACIONADOS

2.1.1 Jogo educacional de Redes de Computador.

O trabalho desenvolvido por uma ex-aluna do IFFAR, propõe a criação de um jogo educativo voltado para o ensino dos fundamentos das redes de computadores. A proposta inclui a criação de um jogo em plataforma 2D, com foco em situações/problemas que exigem que o jogador aplique os conhecimentos sobre a disciplina de Redes de computadores. A abordagem do jogo é mais objetiva, com perguntas e respostas, e visa fortalecer os conteúdos ensinados em sala de aula.

A semelhança com o meu trabalho está na aplicação da tecnologia como ferramenta de apoio à aprendizagem. No entanto, o jogo de Bianca segue uma estrutura mais técnica e direta, com foco em memorização e resolução de problemas objetivos, enquanto o jogo aqui proposto explora os conceitos de game design e na utilização de mecânicas através da Unity.

2.1.2 Jogo educacional de Geografia: Perdido em Pindorama

O projeto, também desenvolvido por um ex-aluno do IFFAR, consiste no desenvolvimento de um jogo voltado para o ensino de conceitos de Geografia, como localização, relevo, clima e vegetação. O jogo é construído em estilo quiz interativo, com perguntas sobre mapas e imagens, e possui sistema de pontuação. A proposta apresenta um ambiente visual atrativo e estimula o aprendizado por meio da repetição e do reforço positivo.

2.2 ENGINES DE JOGOS DIGITAIS

As engines de jogos digitais são plataformas de desenvolvimento que reúnem um conjunto de ferramentas, bibliotecas e sistemas integrados com o objetivo de facilitar a criação de jogos eletrônicos. Essas ferramentas permitem que desenvolvedores concentrem seus esforços na lógica, na mecânica e no design do jogo, sem a necessidade de construir todos os componentes técnicos do zero, como sistemas de renderização gráfica, gerenciamento de física, áudio e entrada de dados.

De forma geral, uma engine de jogos fornece recursos essenciais como motor gráfico, sistema de física, gerenciamento de cenas, suporte a scripts, controle de animações, detecção de colisões e ferramentas para criação de interfaces gráficas. Além disso, muitas engines modernas oferecem ambientes visuais integrados, que permitem a edição de cenários, personagens e comportamentos de forma intuitiva, mesmo para desenvolvedores iniciantes.

Entre as principais engines utilizadas atualmente no desenvolvimento de jogos destacam-se a Unity, a Unreal Engine e a Godot. A Unity é amplamente utilizada tanto no meio acadêmico quanto no mercado profissional, devido à sua flexibilidade, ampla documentação e suporte a múltiplas plataformas. A Unreal Engine, por sua vez, é conhecida por seu alto nível gráfico e por utilizar a linguagem C++ juntamente com sistemas visuais de programação. Já a Godot é uma engine de código aberto que tem ganhado destaque por sua leveza e facilidade de uso, especialmente em projetos independentes.

O uso de engines de jogos democratizou o desenvolvimento de jogos digitais, permitindo que estudantes, pesquisadores e pequenos estúdios tenham acesso a ferramentas antes restritas a grandes empresas. Dessa forma, as engines tornam-se fundamentais não apenas para a produção de jogos comerciais, mas também como instrumentos pedagógicos no ensino de programação e lógica computacional.

2.3 INTRODUÇÃO À UNITY

A Unity é uma engine de desenvolvimento de jogos amplamente utilizada tanto no meio acadêmico quanto na indústria de jogos digitais, sendo reconhecida por sua versatilidade, facilidade de uso e suporte a múltiplas plataformas. A engine permite o desenvolvimento de jogos e aplicações interativas em duas e três dimensões (2D e 3D), oferecendo um ambiente integrado que reúne recursos gráficos, físicos e lógicos em um único sistema.

O funcionamento da Unity baseia-se em um modelo de desenvolvimento orientado a componentes. Nesse modelo, todos os elementos presentes no jogo são representados por objetos chamados GameObjects. Um GameObject, por si só, não possui comportamento ou funcionalidade, sendo apenas uma entidade vazia. As funcionalidades são adicionadas a esses objetos por meio de Componentes, que podem ser visuais, físicos ou lógicos. Exemplos de componentes incluem o Transform, responsável por posição, rotação e escala; o Renderer, que define como o objeto será exibido visualmente; e os Colliders, que permitem a detecção de colisões.

A lógica de funcionamento do jogo é implementada por meio de scripts, que também são componentes associados aos GameObjects. Esses scripts são desenvolvidos utilizando a linguagem de programação C#, uma linguagem moderna, orientada a objetos e amplamente utilizada no desenvolvimento de aplicações. A escolha do C# como linguagem principal da Unity permite a aplicação de conceitos fundamentais da Programação Orientada a Objetos, como classes, métodos, atributos, encapsulamento e reutilização de código.

A Unity utiliza um ciclo de execução baseado em eventos e métodos pré-definidos, conhecidos como métodos de ciclo de vida. Entre os principais métodos estão o Start(), executado uma única vez no início da execução do jogo; o Update(), chamado continuamente a cada quadro renderizado; e o FixedUpdate(), utilizado para cálculos relacionados à física. Esse modelo permite que o desenvolvedor controle de forma precisa o comportamento dos objetos ao longo do tempo, reagindo às ações do jogador e às mudanças no ambiente do jogo.

Outro aspecto fundamental da Unity é o gerenciamento de cenas. Uma cena representa um estado ou “fase” do jogo, podendo corresponder a um menu, um nível ou um momento específico . A organização do jogo em múltiplas cenas facilita a estruturação do projeto, o controle do fluxo da aplicação e a reutilização

de elementos. A troca entre cenas é realizada por meio do comando LoadScene, possibilitando a criação de transições e progressões dentro do jogo.

A engine também fornece um sistema robusto para criação de interfaces gráficas, conhecido como UI System. Esse sistema permite a construção de menus, caixas de diálogo, textos informativos e botões interativos, elementos essenciais para a comunicação entre o jogo e o jogador. A interface gráfica é organizada dentro de um objeto chamado Canvas, que funciona como a base para todos os elementos visuais da interface.

Além disso, a Unity conta com sistemas integrados para entrada de dados (Input System), permitindo a leitura de comandos do teclado, mouse e outros dispositivos. Esse sistema possibilita que o desenvolvedor associe ações do jogador, como cliques e movimentos, às respostas do jogo.

A engine também oferece suporte a animações por meio do Animator, um sistema que permite controlar transições entre diferentes estados de animação, como repouso e movimento. Esse recurso é fundamental para dar vida aos personagens e melhorar a imersão do jogador, mesmo em projetos de pequeno porte ou com foco educacional.

Por fim, a Unity se destaca por seu ambiente de desenvolvimento integrado, que fornece ferramentas visuais para edição de cenas, organização de objetos, inspeção de componentes e depuração de código. Esse conjunto de recursos torna a engine adequada para iniciantes, ao mesmo tempo em que oferece profundidade suficiente para projetos mais avançados. No contexto deste trabalho, a Unity foi escolhida por possibilitar a exploração prática dos conceitos fundamentais do desenvolvimento de jogos digitais, servindo como uma introdução eficaz à área para estudantes de informática.

2.4 C# (C SHARP)

A linguagem de programação C# (C Sharp) é uma linguagem moderna, orientada a objetos e de propósito geral, desenvolvida pela Microsoft como parte da plataforma .NET. Seu projeto foi concebido com o objetivo de oferecer uma sintaxe clara, segura e eficiente, tornando-se amplamente utilizada no desenvolvimento de aplicações desktop, web, mobile e, especialmente, no desenvolvimento de jogos digitais por meio da Unity.

Uma das principais características do C# é o suporte completo à programação orientada a objetos (POO). Esse paradigma organiza o código em estruturas chamadas classes, que representam entidades do sistema e encapsulam dados e comportamentos. No contexto do desenvolvimento de jogos, esse modelo permite representar personagens, inimigos, itens e sistemas do jogo como objetos independentes, facilitando a modularização, a reutilização de código e a manutenção do projeto.

O C# é uma linguagem fortemente tipada, o que significa que todas as variáveis devem ter seus tipos definidos explicitamente. Essa característica contribui para a prevenção de erros comuns durante a execução do programa, pois inconsistências de tipo são detectadas ainda em tempo de compilação. Além disso, a linguagem oferece suporte a diversos tipos de dados, como inteiros, números decimais, booleanos, vetores e estruturas personalizadas, amplamente utilizados no controle da lógica e do estado do jogo.

Outro aspecto relevante da linguagem é o uso de métodos e funções, que permitem dividir o código em blocos organizados e reutilizáveis. No desenvolvimento de jogos, métodos são utilizados para implementar ações específicas, como movimentar o personagem, executar ataques, detectar colisões e responder às interações do jogador. Essa abordagem contribui para a clareza do código e para a separação de responsabilidades dentro do sistema.

A linguagem C# também oferece suporte a estruturas de controle, como comandos condicionais e laços de repetição, fundamentais para a tomada de decisões e para o controle do fluxo de execução do programa. Essas estruturas são amplamente empregadas para verificar estados do jogo, como a vida do personagem, a presença de inimigos ou a ativação de eventos específicos.

No ambiente da Unity, o C# é utilizado por meio de scripts que herdam da classe MonoBehaviour, permitindo que o código interaja diretamente com os objetos da cena. A Unity fornece um conjunto de métodos pré-definidos, como os responsáveis pela inicialização e atualização dos objetos, possibilitando que o programador controle o comportamento do jogo ao longo do tempo.

Por fim, o uso do C# no desenvolvimento do projeto contribuiu para a compreensão prática de conceitos fundamentais de programação, como lógica, organização de código, depuração e interação entre sistemas. Dessa forma, a linguagem se mostrou adequada não apenas para a implementação técnica do

jogo, mas também como ferramenta de aprendizado para estudantes que estão dando os primeiros passos no desenvolvimento de jogos digitais.

3 DESENVOLVIMENTO

Esta seção especifica o processo de desenvolvimento do projeto, demonstrando como o projeto foi organizado, as principais mecânicas desenvolvidas, como foram desenvolvidas, e que ferramentas foram utilizadas para desenvolvê-las.

3.1 ESTRUTURA DO PROJETO E ELEMENTOS VISUAIS

A estrutura do projeto foi organizada de acordo com o modelo padrão da engine Unity, que utiliza uma abordagem baseada em componentes e organização por pastas. Essa estrutura facilita o gerenciamento dos recursos do jogo, a manutenção do código e a compreensão do funcionamento do projeto, especialmente para desenvolvedores iniciantes.

No diretório principal do projeto, foram criadas pastas específicas para separar os diferentes tipos de recursos utilizados, como Scripts, Sprites, Animações e Prefabs[7]. Essa organização contribuiu para uma melhor legibilidade do projeto e para a localização rápida dos arquivos durante o desenvolvimento, além de seguir boas práticas recomendadas pela própria Unity.

Para a criação visual do jogo, optou-se pelo uso de sprites prontos, obtidos a partir de um pacote de recursos gratuito chamado Mystic Woods[8], de onde foram retirados todos os elementos visuais do projeto. Essa escolha foi feita com

[7] Prefabs no Unity são modelos reutilizáveis de GameObjects (objetos do jogo) pré-configurados, salvos como ativos no projeto, que permitem criar múltiplas cópias (instâncias) com as mesmas propriedades, componentes e scripts, facilitando a organização e atualização de múltiplos objetos iguais.

[8] Adquira e faça o download do pacote: https://game-endeavor.itch.io/mystic-woods

o objetivo de concentrar o foco do projeto nos aspectos técnicos e conceituais do desenvolvimento de jogos, como programação, lógica e mecânicas, em vez da produção artística dos elementos gráficos. O uso de sprites prontos é uma prática comum em projetos educacionais, protótipos, ou em projetos independentes (jogos indie[9]), permitindo maior agilidade no desenvolvimento.

A montagem do cenário foi realizada em ambiente 2D com vista top-down, utilizando sprites individuais para representar o chão, relevos e demais elementos do ambiente. Esses sprites foram posicionados diretamente na cena por meio do editor visual da Unity, possibilitando a construção do mapa de forma intuitiva. Para cada elemento visual, foi utilizado o componente Sprite Renderer, responsável por exibir as imagens na tela e controlar propriedades como ordem de renderização e visibilidade.

A organização visual dos elementos do cenário levou em consideração a camada de renderização, garantindo que personagens e objetos interativos fossem exibidos corretamente sobre o ambiente. Para isso, foram utilizados parâmetros como a ordem de camadas e a separação por sorting layers, recurso que permite controlar a sobreposição dos sprites em cenas 2D.

Além dos elementos visuais, os objetos do cenário receberam componentes adicionais, como Colliders 2D, responsáveis por impedir que o personagem atravesse paredes ou objetos sólidos. Esses componentes são essenciais para a criação de limites físicos no jogo e para a detecção de interações entre o jogador e o ambiente. Os detalhes sobre o sistema de colisões serão abordados em breve neste capítulo.

A estrutura baseada em componentes da Unity permitiu que cada objeto da cena possuísse apenas os elementos necessários para seu funcionamento, como renderização, colisão e scripts de comportamento. Essa abordagem modular facilitou a implementação e o ajuste das mecânicas do jogo, além de contribuir para o entendimento prático do funcionamento da engine.

Dessa forma, a estrutura do projeto na Unity foi planejada para oferecer um ambiente organizado, funcional e adequado ao aprendizado dos principais conceitos envolvidos no desenvolvimento de jogos digitais, reforçando o caráter introdutório e educacional do projeto.

---

[9] Jogos indie (independentes) são games criados por desenvolvedores ou pequenas equipes sem o apoio financeiro de grandes publicadoras.

3.2 CRIAÇÃO DO PERSONAGEM JOGÁVEL

A criação do personagem jogável constitui uma das etapas centrais do desenvolvimento do jogo, pois envolve a implementação dos sistemas responsáveis pela representação visual, movimentação, interação e controle do jogador. Neste projeto, o personagem foi desenvolvido em um ambiente 2D com visão top-down, utilizando os recursos nativos da Unity para renderização, física e entrada de comandos.

3.2.1 Sprite e Renderização

A representação visual do personagem foi realizada por meio de sprites 2D, imagens que representam o personagem em diferentes estados, como parado e em movimento. Para exibir esses sprites na cena, foi utilizado o componente Sprite Renderer, responsável por renderizar a imagem associada ao objeto do jogador na tela.

O Sprite Renderer permite controlar propriedades como: Sprite exibido, Ordem de renderização, Camadas visuais (sorting layers) Espelhamento horizontal do sprite.

No contexto do jogo top-down, essa abordagem é essencial para garantir que o personagem seja exibido corretamente em relação ao cenário e aos demais objetos da cena. O espelhamento do sprite, por exemplo, foi utilizado para indicar visualmente a direção do movimento horizontal do personagem, contribuindo para uma melhor percepção de controle por parte do jogador.

3.2.2 Sistema de Movimento do Jogador

O sistema de movimentação do personagem foi implementado utilizando uma abordagem baseada em física 2D, fazendo uso dos componentes Rigidbody2D e Collider2D, em conjunto com o Input System da Unity.

O componente Rigidbody2D é responsável por permitir que o personagem interaja com o sistema de física da Unity. No script PlayerController, esse

componente é utilizado para mover o personagem de forma controlada por meiodo método MovePosition, que desloca o objeto respeitando colisões e a simulação física. Essa abordagem evita movimentos abruptos ou atravessamento de objetos sólidos, garantindo um deslocamento mais consistente e realista dentro do ambiente do jogo.

Já o Collider2D define a área física do personagem, permitindo a detecção de colisões com o cenário e outros objetos. Em conjunto com o Rigidbody2D, ele impede que o jogador atravesse paredes ou elementos do ambiente.

No script, o sistema de colisão é reforçado pelo uso do método Cast, que realiza uma verificação antecipada de possíveis colisões antes que o movimento seja efetivamente aplicado. Dessa forma, o personagem só se desloca caso não exista nenhum obstáculo no caminho.

A lógica de movimentação é executada no método FixedUpdate, que é indicado pela Unity para operações relacionadas à física. A cada atualização, o script verifica se existe algum valor de entrada de movimento e, caso exista, tenta mover o personagem na direção desejada. O processo ocorre da seguinte forma:

O vetor de movimento é recebido a partir do input do jogador e o script tenta mover o personagem na direção completa. Caso haja colisão, o movimento é testado separadamente nos eixos horizontal e vertical. Se nenhuma colisão for detectada, o movimento é aplicado ao Rigidbody2D.

Essa lógica garante maior fluidez na movimentação, mesmo em situações onde o personagem colide parcialmente com obstáculos, permitindo que ele deslize ao longo das paredes.

3.2.3 Input System

O controle do personagem foi implementado utilizando o Input System, sistema moderno de entrada da Unity que substitui o método tradicional baseado em teclas fixas. Esse sistema permite uma maior flexibilidade na configuração dos comandos e facilita a adaptação para diferentes dispositivos de entrada.

No script PlayerController, o método OnMove recebe um vetor bidimensional que representa a direção do movimento, baseado nas teclas

pressionadas pelo jogador. Esse vetor é então armazenado e utilizado pelo sistema de movimentação no FixedUpdate.

Essa separação entre captura de input e aplicação do movimento segue boas práticas de desenvolvimento, tornando o código mais organizado e fácil de manter.

O controle do personagem foi configurado para teclado, utilizando as teclas direcionais ou equivalentes (como WASD), mapeadas no Input System. O vetor de movimento resultante dessas entradas define a direção e velocidade do deslocamento do personagem no cenário. Além disso, o sistema de controle permite a expansão para outras ações, como ataques ou interações.

3.2.4 Animações do personagem

Embora o foco deste tópico seja a movimentação, o sistema de controle também está diretamente integrado ao sistema de animações. O script utiliza parâmetros do tipo booleano para informar ao Animator se o personagem está em movimento ou parado, permitindo a alternância entre animações de idle (personagem parado) e deslocamento de forma automática. Essa integração reforça a importância da comunicação entre lógica e apresentação visual no desenvolvimento de jogos. Detalhes sobre o funcionamento das animações serão abordados em breve neste capítulo

3.3 ANIMAÇÕES

O sistema de animações é responsável por dar vida ao personagem jogável, tornando suas ações visualmente compreensíveis e coerentes com os comandos do jogador. Neste projeto, o sistema de animações foi implementado utilizando o Animator Controller da Unity, que permite gerenciar diferentes estados de animação, suas transições e a comunicação com os scripts de controle.

3.3.1 Animator Controller

O Animator Controller é um recurso da Unity utilizado para organizar e controlar animações por meio de uma máquina de estados (state machine). Cada estado representa uma animação específica, enquanto as transições definem as condições necessárias para alternar entre esses estados. No projeto, o Animator Controller foi associado ao personagem jogável e configurado para controlar suas animações básicas de comportamento, como permanecer parado, se mover e atacar.

3.3.2 Estados de Animação

O Animator Controller foi estruturado com os seguintes estados principais:

Idle, que representa o estado em que o personagem está parado, sem receber comandos de movimentação. Essa animação é exibida sempre que o personagem não está se deslocando pelo cenário. O movimento, que corresponde à animação de deslocamento do personagem. Esse estado é ativado quando o jogador fornece comandos de movimento, indicando que o personagem está andando pelo ambiente. E por último, o Ataque, estado responsável pela animação de ataque com espada. Essa animação é executada quando o jogador aciona o comando de ataque, representando visualmente a ação ofensiva do personagem.

3.3.3 Parâmetros do Animator

O Animator Unity é a ferramenta dentro do motor da engine que gerencia o comportamento e as transições entre animações de personagens e objetos, atuando como um "cérebro" que decide qual animação (parado, correndo,atacando) deve ser executada com base em variáveis e condições definidas, criando uma experiência fluida e mais imersa para o jogador. A comunicação entre os scripts e o Animator ocorre por meio de parâmetros, que são valores utilizados para controlar as transições entre os estados de animação.

O parâmetro booleano é utilizado para indicar estados contínuos, como movimento. Por exemplo, quando o personagem está andando, o script define o valor do parâmetro isMoving como verdadeiro, fazendo com que o Animator transite do estado Idle para o estado Movimento.

Os parâmetros do tipo Trigger são utilizados para ações pontuais, como o ataque. Diferentemente dos parâmetros booleanos, o Trigger é ativado apenas uma vez e automaticamente retorna ao estado inativo após a transição.

No projeto, o Trigger swordAttack é acionado quando o jogador executa o comando de ataque, iniciando a animação correspondente.

3.3.4 Transições entre Estados

As transições definem como e quando o Animator muda de um estado para outro. Cada transição possui condições baseadas nos parâmetros configurados. No caso do personagem, A transição de Idle para Movimento ocorre quando o parâmetro isMoving é verdadeiro, e a transição de Movimento para Idle ocorre quando isMoving é falso. A transição para o estado de Ataque ocorre quando o Trigger swordAttack é ativado.

Essas transições garantem que as animações reflitam corretamente o comportamento do jogador em tempo real.

3.4.5 Animation Events

Os Animation Events são recursos da Unity que permitem chamar funções de scripts em momentos específicos de uma animação. Eles são fundamentais para sincronizar ações lógicas com a execução visual da animação.

No projeto, os Animation Events foram utilizados para controlar o sistema de ataque, garantindo que o dano seja aplicado apenas no momento correto da animação.

O evento swordAttack é acionado durante a animação de ataque, no exato momento em que o golpe é visualmente executado. Ele chama uma função no script do personagem responsável por detectar inimigos próximos e aplicar dano.

Já o EndSwordAttack é executado ao final da animação de ataque e tem como função encerrar o estado de ataque, permitindo que o personagem volte a se mover ou executar outras ações.

Essa abordagem evita que o ataque seja aplicado fora do tempo correto e garante maior precisão e sensação de impacto durante o combate.

3.4.6 Integração entre Animação e Código

A integração entre o Animator e os scripts é realizada por meio do componente Animator, acessado diretamente no código do personagem. Os scripts definem os valores dos parâmetros do Animator com base nas ações do jogador, enquanto o Animator controla a execução visual dessas ações. Essa separação de responsabilidades melhora a organização do projeto, permitindo que ajustes visuais nas animações sejam feitos sem a necessidade de modificar o código, e vice-versa.

3.5 ATAQUE E INIMIGOS

O sistema de ataque do jogo foi desenvolvido com o objetivo de introduzir conceitos fundamentais do desenvolvimento de jogos, como detecção de colisões, aplicação de dano, controle de estados e sincronização entre lógica e animação. A implementação adota uma abordagem simples e funcional, adequada para um projeto introdutório, mas que reflete práticas comuns em jogos 2D do gênero top-down.

3.5.1 Ataque do Jogador

O ataque do jogador é realizado por meio de uma ação direta, associada a uma animação de golpe com espada. Esse sistema é baseado na ativação temporária de uma área de colisão que detecta inimigos atingidos. A área de ataque é representada por um Collider2D posicionado na região da espada do personagem. Esse collider define a zona na qual inimigos podem ser atingidos durante a execução do ataque.

Durante o jogo, esse collider não permanece ativo continuamente, evitando que o jogador cause dano de forma indevida ao simplesmente se aproximar de um inimigo.

O collider da espada é ativado apenas durante o momento exato do golpe. Esse controle é feito por meio de métodos chamados a partir do sistema de animações, garantindo que o ataque ocorra apenas quando a animação estiver sendo executada. Quando o ataque começa, o collider é ativado, permitindo a detecção de colisões com inimigos. Ao final da animação, o collider é desativado, encerrando a janela de ataque.

Essa abordagem melhora a precisão do combate e evita comportamentos inesperados.

3.5.2 Sincronização com a Animação

A sincronização entre ataque e animação é realizada por meio de Animation Events configurados diretamente no Editor da Unity. Esses eventos chamam funções específicas no script do jogador durante a execução da animação de ataque. O evento swordAttack é acionado no momento em que o golpe é visualmente executado, enquanto o evento EndSwordAttack marca o término da ação. Essa técnica garante que o dano seja aplicado apenas no instante correto, reforçando a coerência entre a lógica do jogo e sua representação visual.

3.5.3 Inimigos

Os inimigos do jogo foram implementados com um comportamento básico, suficiente para demonstrar os principais conceitos envolvidos em sistemas de combate, como vida, recebimento de dano e destruição do objeto.

Cada inimigo possui um script próprio responsável por gerenciar seu estado e interações com o jogador. Esse script controla atributos como vida e define como o inimigo reage ao ser atingido. A separação da lógica do inimigo em um script dedicado contribui para a organização do projeto e facilita futuras expansões, como a adição de diferentes tipos de inimigos.

3.5.4 Vida

O sistema de vida do inimigo é baseado em um valor numérico que representa sua quantidade de pontos de vida. Ao receber dano, esse valor é reduzido de acordo com o dano de ataque do jogador. Quando a vida do inimigo atinge zero, ele é considerado derrotado, acionando o processo de destruição do objeto, e a animação de morte é acionada.

Após a vida do inimigo chegar a zero, o objeto é removido da cena utilizando o método de destruição da Unity. Esse processo encerra a interação com o inimigo e representa visualmente sua derrota no jogo. Essa etapa demonstra o ciclo completo de uma entidade no jogo: criação, interação, dano e remoção.

Esses conceitos formam a base para sistemas de combate mais complexos e são amplamente utilizados em grandes jogos comerciais.

3.6 COLISÕES E FÍSICA

A utilização do sistema de física 2D da Unity foi essencial para o funcionamento adequado da movimentação, do combate e das interações presentes no jogo. Esse sistema permite detectar colisões, impedir atravessamentos indevidos de objetos e garantir maior realismo e consistência às ações realizadas pelo jogador e pelos inimigos.

3.6.1 Colliders e Triggers

Na Unity, os Colliders 2D são componentes responsáveis por definir a forma física de um objeto no jogo, permitindo que o motor de física detecte interações entre diferentes elementos da cena.

Existem dois comportamentos principais associados aos colliders:

Collider comum: utilizado para impedir a passagem de objetos, como paredes, obstáculos do cenário e limites do mapa;

Trigger: utilizado para detectar a entrada ou saída de um objeto em determinada área, sem bloquear seu movimento.

No projeto, colliders comuns foram empregados para estruturar o cenário e evitar que o personagem atravessasse paredes, enquanto triggers foram utilizados para detectar eventos específicos, como ataques e interações.

3.6.2 OnTriggerEnter2D

A função OnTriggerEnter2D é chamada automaticamente pela Unity sempre que um objeto com Collider2D configurado como Trigger entra em contato com outro collider. Esse método foi utilizado principalmente no sistema de combate, permitindo detectar quando a área de ataque do jogador entra em contato com um inimigo. A partir dessa detecção, o sistema aplica o dano correspondente ao inimigo atingido.

3.6.3 Detecção de Obstáculos

A detecção de obstáculos foi implementada utilizando o método Cast do componente Rigidbody2D. Esse método realiza uma verificação antecipada na direção do movimento, identificando possíveis colisões antes que o deslocamento ocorra. Caso nenhuma colisão seja detectada, o movimento é permitido. Caso contrário, o deslocamento é bloqueado, impedindo que o personagem atravesse objetos sólidos.

Essa abordagem melhora a robustez do sistema de movimentação e evita falhas comuns em jogos 2D, como atravessar paredes ou objetos do cenário.

3.7 INPUT SYSTEM

O gerenciamento de entrada do jogador é um dos elementos fundamentais no desenvolvimento de jogos digitais, pois é por meio dele que o usuário interage com o sistema. Neste projeto, foi utilizado o novo Input System da Unity, uma solução moderna que substitui o sistema de entrada legado, oferecendo maior flexibilidade, organização e compatibilidade com diferentes dispositivos.

O Input System da Unity é um pacote oficial que permite gerenciar entradas de teclado, mouse, controle e outros dispositivos de forma unificada. Diferentemente do sistema antigo, que dependia de chamadas diretas como Input.GetAxis() e Input.GetKeyDown(), o novo sistema é baseado em ações, facilitando a manutenção e a expansão do projeto.

Esse sistema foi escolhido por representar o padrão atual recomendado pela Unity, além de permitir uma separação mais clara entre a lógica do jogo e os dispositivos de entrada.

3.7.1 Actions

As Actions representam intenções do jogador, como mover o personagem ou realizar um ataque. No projeto, foram criadas ações específicas, como Move, responsável pela movimentação do personagem, e Fire (Attack), responsável pelo ataque do jogador.

Cada ação pode ser vinculada a diferentes dispositivos e teclas, permitindo que o mesmo comando seja acionado de múltiplas formas sem alterar o código.

3.7.2 Mapeamento de Teclas

O mapeamento de teclas é realizado dentro de um Input Action Asset, onde cada ação recebe associações específicas, como Teclas W, A, S, D e setas direcionais para movimentação e Tecla Espaço para ataque. Esse modelo facilita futuras alterações, pois as mudanças podem ser feitas diretamente no asset de input, sem necessidade de modificar os scripts.

3.7.3 Eventos (OnMove e OnFire)

No novo Input System, as entradas são tratadas por meio de eventos, que disparam métodos automaticamente quando uma ação é executada.

No projeto, foram utilizados métodos OnMove, responsável por capturar a direção do movimento do jogador, e OnFire responsável por ativar o ataque do personagem

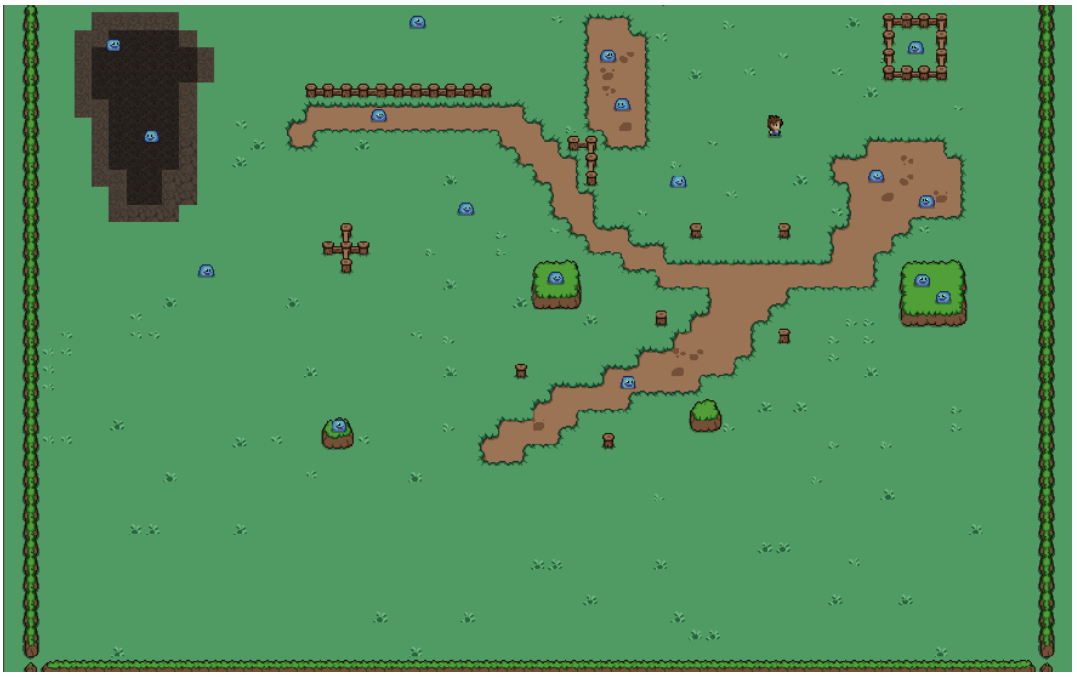
Esses métodos são chamados automaticamente quando o jogador executa a ação correspondente.

3.8 TELAS DO SISTEMA

Figura 1 – Personagem jogável

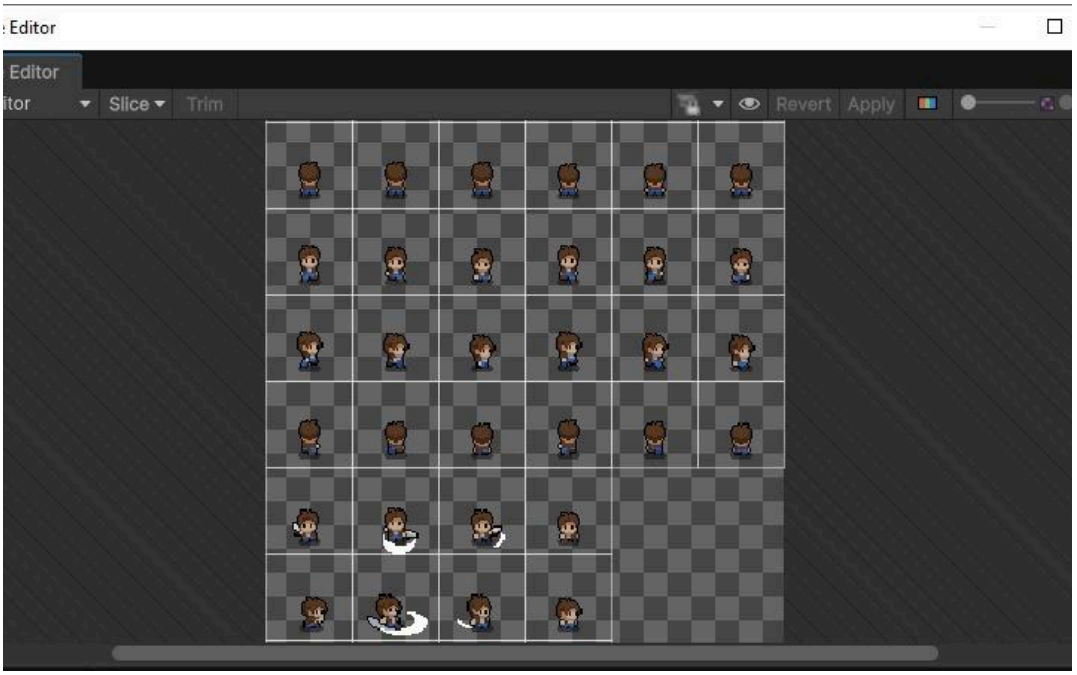
Fonte: autoria própria

Figura 2 – Cenário completo



Fonte: autoria própria

Figura 3 – Janela “Sprite editor” do personagem. Janela em que cada sprite da animação é separado individualmente.



Fonte: autoria própria

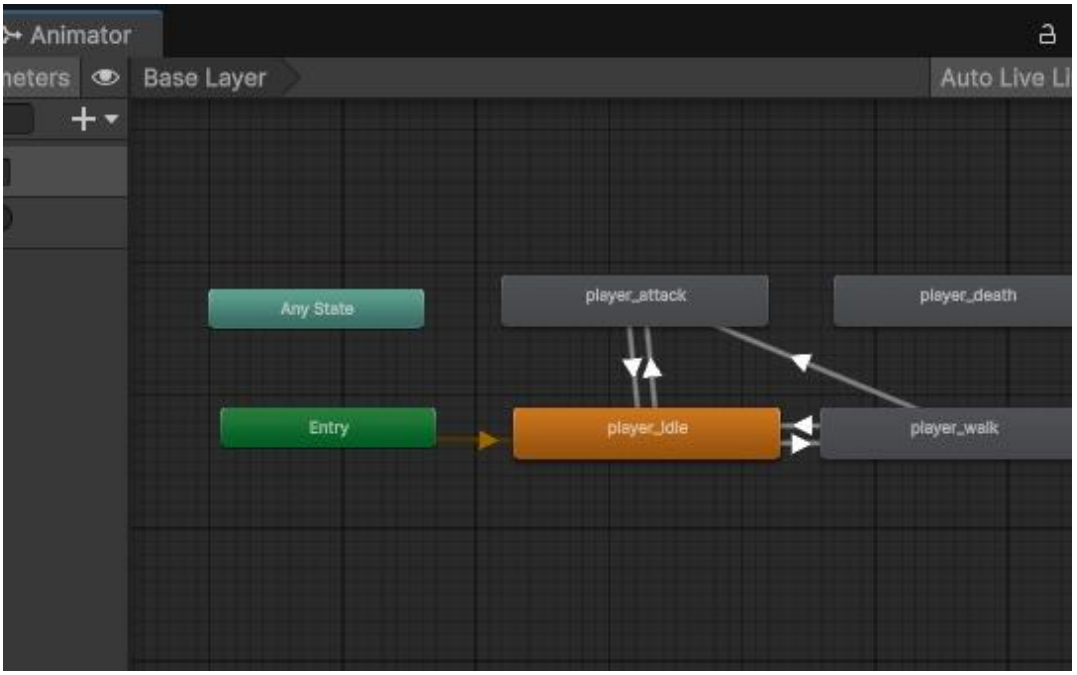
Figura 4 – Animação do Personagem se movendo

Fonte: autoria própria

Figura 5 – Animação do Personagem atacando

Fonte: autoria própria

Figura 6 – Janela “Animator” do personagem. Janela onde o fluxo de transições entre animações é criado/mostrado.

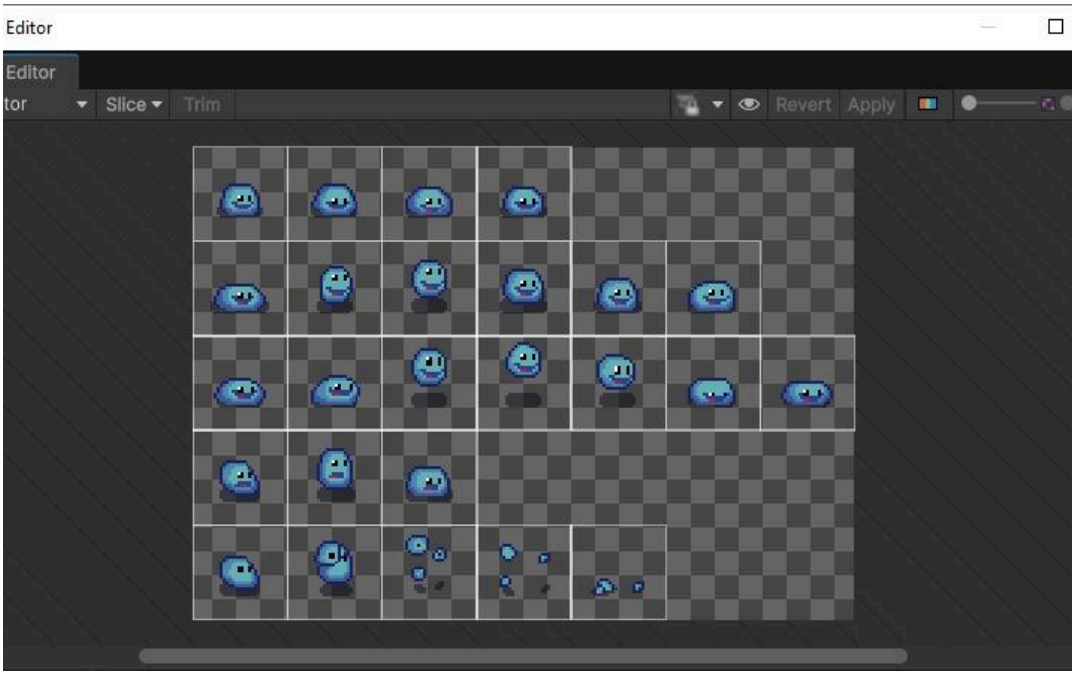


Fonte: autoria própria

Figura 7 – Inimigo padrão

Fonte: autoria própria

Figura 8 – Janela “Sprite editor” do Inimigo. Janela em que cada sprite da animação é separado individualmente.

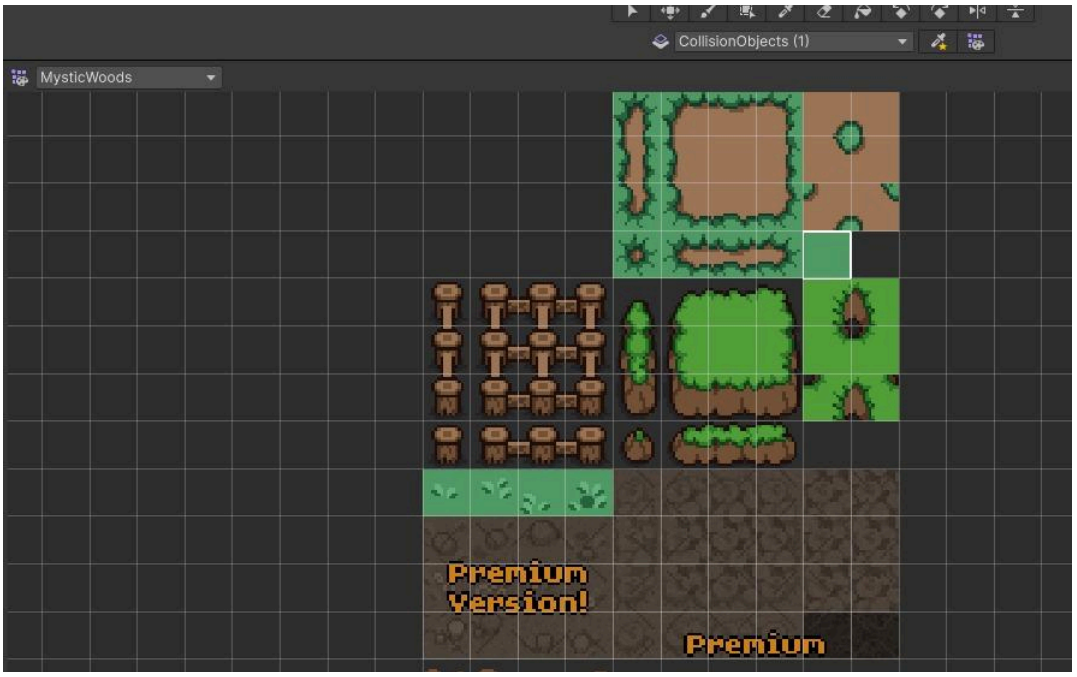


Fonte: autoria própria

Figura 9 – Inimigo sendo derrotado. O inimigo é derrotado após ser atacado.

Fonte: autoria própria

Figura 10 – Janela “Tile pallete” dos elementos do cenário. Janela onde os elementos que serão usados para montar o cenário são selecionados



Fonte: autoria própria

4 CONSIDERAÇÕES FINAIS

A realização deste Trabalho de Conclusão de Curso proporcionou uma experiência significativa de aprendizagem ao explorar o desenvolvimento de jogos digitais como uma área prática e interdisciplinar dentro da Informática. O projeto, reformulado com foco na introdução ao desenvolvimento de jogos utilizando a engine Unity, permitiu não apenas a criação de um jogo funcional, mas principalmente a compreensão dos conceitos fundamentais que envolvem o processo de criação de jogos digitais.

Ao longo do desenvolvimento, foi possível aplicar e aprofundar conhecimentos de programação em C#, lógica computacional, orientação a objetos, sistemas de entrada, animação, física 2D, colisões, gerenciamento de cenas e interação com o jogador. A utilização da Unity mostrou-se adequada para fins educacionais, pois oferece uma estrutura robusta, ferramentas visuais

integradas e ampla documentação, facilitando o aprendizado mesmo para desenvolvedores iniciantes.

O projeto evidenciou que o desenvolvimento de jogos vai além da criação de um produto final, envolvendo planejamento, organização de sistemas, testes constantes e resolução de problemas. Durante o processo, diversos desafios técnicos foram enfrentados, como a implementação correta do sistema de movimento, o gerenciamento de animações sincronizadas com ações do jogador, o tratamento de colisões e a adaptação ao novo Input System da Unity. A superação desses desafios contribuiu diretamente para o amadurecimento técnico e para o desenvolvimento do pensamento lógico e analítico.

Além disso, o projeto reforça a importância do aprendizado prático como ferramenta pedagógica, demonstrando que o desenvolvimento de jogos pode ser uma porta de entrada eficaz para o estudo de programação e engenharia de software. Mesmo não sendo uma área prevista diretamente na ementa do curso, o desenvolvimento de jogos se mostrou altamente relevante por integrar conceitos fundamentais da Informática em um contexto aplicado e motivador.

Por fim, conclui-se que este trabalho atingiu seus objetivos ao documentar o processo de aprendizado e desenvolvimento de um jogo na Unity, servindo como referência introdutória para estudantes que desejam iniciar seus estudos na área de desenvolvimento de jogos. Como trabalhos futuros, sugere-se a expansão do projeto com novas mecânicas, aprimoramento da interface, inclusão de sistemas mais avançados de inteligência artificial e a adaptação do jogo para diferentes plataformas, ampliando ainda mais o escopo educacional da proposta.

REFERÊNCIAS

RIBEIRO, Bianca Maia. **Jogo educacional de Redes de Computador**. 2021. 33 f. TCC (Graduação) - Curso de Informática, Instituto Federal Farroupilha Campus Uruguaiana, Uruguaiana, 2021. Disponível em: https://arandu.iffarroupilha.edu.br/bitstream/itemid/200/1/Bianca_jogo%20educacional%20de%20redes%20de%20computador.pdf. Acesso em: 25 jul. 2025.

RODRIGUES, João Victor dos Santos. **Jogo educacional de Geografia**: perdido em pindorama. 2021. 51 f. TCC (Doutorado) - Curso de Informática, Instituto Federal Farroupilha Campus Uruguaiana, Uruguaiana, 2023. Disponível em: https://arandu.iffarroupilha.edu.br/bitstream/itemid/200/1/Bianca_jogo%20educacional%20de%20redes%20de%20computador.pdf. Acesso em: 25 jul. 2025.

LOURENÇO, Patrícia. **Games e ensino de História**: potencialidades e desafios. 2016. 32 f. TCC (Graduação) - Curso de História, Universidade Federal do Tocantins Campus de Araguaína Curso de Licenciatura em História, Araguaína, 2016. Disponível em: https://drive.google.com/file/d/1gf7ewJhEZ-xJft8bZQgnaYm9O08VxHth/view?usp=sharing. Acesso em: 25 jul. 2025.