INSTITUTO FEDERAL
Farroupilha
Campus Uruguaiana

CURSO TÉCNICO EM INFORMÁTICA INTEGRADO AO ENSINO MÉDIO

LORENZO NARDON FRIGO

CUNNING RACE:

UM PROJETO QUE RELEMBRA OS CLÁSSICOS JOGOS DE PLATAFORMA

URUGUAIANA

2025

LORENZO NARDON FRIGO

CUNNING RACE:
UM PROJETO QUE RELEMBRA OS CLÁSSICOS JOGOS DE PLATAFORMA

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

Anderson Mendes Rocha

URUGUAIANA

2025

Frigo, Lorenzo Nardon.

Cunning Race : Um jogo no estilo plataforma que relembra os clássicos de nossas infâncias/Lorenzo Nardon Frigo. — 2025.

49 f.

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

1.jogo eletrônico. 2.Godot Engine. 3. arcade aventureiro. 4. desenvolvimento de jogos. 5.plataforma 2D. Cunning Race.

CDD [número da CDD].

LORENZO NARDON FRIGO

CUNNING RACE:

UM PROJETO QUE RELEMBRA OS CLÁSSICOS JOGOS DE PLATAFORMA

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.

Este trabalho foi defendido e aprovado pela banca em 07/01/2026.

BANCA EXAMINADORA

_____

Prof. Me. Eduardo Henrique Spies
Orientador

_____

Prof. Me. Anderson Mendes Rocha
Co-Orientador

_____

Prof. Me. Leandro Martins Dallanora
Avaliador

_____

Prof. Me. Natthan Ruschel Soares
Avaliador

Dedico este trabalho para todos os
amantes de jogos, vamos juntos elevar
esse universo para outro patamar.

AGRADECIMENTOS

Agradeço ao apoio dos meus pais e da minha família por sempre demonstrarem apoio e auxílio para mim durante toda minha construção educacional. Também agradeço aos meus orientadores por me guiarem nesse projeto e desenvolvê-lo com o máximo de eficácia. Agradeço também aos meus amigos lendários denominados como “Os Bola” por me fazerem companhia e me distrair em momentos necessários.

Erre ontem, aprenda hoje, e rateie de novo amanhã…

GURI DE URUGUAIANA

RESUMO

O presente trabalho tem como objetivo o desenvolvimento de um protótipo de jogo eletrônico no estilo arcade aventureiro, utilizando a *Godot Engine* e a linguagem GDScript, com foco em proporcionar ao jogador uma experiência divertida, fluida e desafiadora. O projeto foi motivado pelo interesse pessoal e profissional do autor no universo dos jogos, bem como pela ausência de disciplinas que abordam diretamente o desenvolvimento de games no curso de formação técnica. A proposta envolve a criação de um jogo 2D com mecânicas clássicas de plataforma, incluindo movimentação, salto, sistema de vida, checkpoints e elementos anacrônicos como inimigos fantasiosos, inseridos em um universo visualmente coeso e interativo. A metodologia adotada inclui pesquisa aplicada, modelagem, implementação técnica, testes de jogabilidade e refinamento baseado em feedback de usuários. O jogo também se propõe como estudo de caso sobre o ciclo completo de produção de um game, abordando desde o planejamento até a documentação final. Os resultados obtidos demonstram o aprendizado significativo em programação orientada a objetos, lógica de jogo e design de níveis, além da viabilidade de criação de projetos completos utilizando ferramentas *open-source*.

Palavras-chave: jogo eletrônico; Godot Engine; arcade aventureiro; desenvolvimento de jogos; plataforma 2D.

ABSTRACT

This work aims to develop a prototype of an electronic game in the adventurous arcade style, using the Godot Engine and the GDScript language, focusing on providing the player with a fun, fluid, and challenging experience. The project was motivated by the author's personal and professional interest in the gaming field, as well as by the absence of disciplines that directly address game development in the technical training program. The proposal involves creating a 2D game with classic platform mechanics, including movement, jumping, health system, checkpoints, and anachronistic elements such as fantasy enemies, all inserted in a visually cohesive and interactive universe. The adopted methodology includes applied research, modeling, technical implementation, gameplay testing, and refinement based on user feedback. The game also serves as a case study of the complete game development cycle, addressing planning, implementation, and final documentation. The results obtained demonstrate significant learning in object-oriented programming, game logic, and level design, as well as the feasibility of creating complete projects using open-source tools.

Keywords: electronic game; Godot Engine; adventurous arcade; game development; 2D platform.

LISTA DE ILUSTRAÇÕES

LISTA DE ABREVIATURAS E SIGLAS

2D	Duas Dimensões Espaciais
3D	Três Dimensões Espaciais
ABNT	Associação Brasileira de Normas Técnicas
API	Application Programming Interface
CC0	Creative Commons Zero
CDD	Classificação Decimal de Dewey
HUD	Heads-Up Display
IFFar	Instituto Federal de Educação, Ciência e Tecnologia Farroupilha
RF	Requisito Funcional
TCC	Trabalho de Conclusão de Curso

SUMÁRIO

1. INTRODUÇÃO

O desenvolvimento de jogos eletrônicos consolidou-se como uma das áreas mais dinâmicas da indústria de software, envolvendo desafios de engenharia, design e experiência do usuário. Em particular, os jogos de plataforma 2D continuam populares por sua simplicidade aparente aliada a profundos mecanismos de interação, gerando interesse tanto no meio acadêmico quanto na comunidade de desenvolvedores independentes. Esses títulos demonstram que, mesmo com restrições técnicas menores do que em projetos 3D, é possível criar experiências ricas de narrativa, feedback imediato e progressão de dificuldade que cativam públicos de todas as idades.

Apesar desse potencial, os currículos de muitos cursos de Computação ainda destinam pouco espaço ao estudo sistemático de *game design* e das especificidades das *engines* 2D. Enquanto disciplinas tradicionais enfatizam algoritmos, estruturas de dados e sistemas distribuídos, aspectos como física de plataforma, transições de cena, gestão de pontos de verificação (*checkpoints*) e design iterativo de níveis raramente são abordados de forma integrada. A carência de atividades práticas voltadas a jogos no ambiente acadêmico pode limitar a capacidade dos estudantes de aplicar conceitos de arquitetura de software em contextos de tempo real, com requisitos de performance e usabilidade mais restritos.

As *engines* de código aberto, como a *Godot Engine*, têm ganhado destaque justamente por oferecerem um ambiente acessível e flexível para prototipação rápida. Seu modelo baseado em *nodes* (estruturas funcionais que compõem a hierarquia de uma cena) e *signals* (mecanismo de emissão e escuta de eventos), aliado a uma linguagem de script intuitiva (*GDScript*), permite a construção de sistemas complexos com menor curva de aprendizagem. Além disso, a comunidade ativa em torno da ferramenta disponibiliza uma ampla gama de recursos, desde tutoriais até bibliotecas de terceiros, fomentando a troca de conhecimento e acelerando o ciclo de desenvolvimento.

Diante desse cenário, este trabalho propõe a criação de um protótipo de jogo eletrônico no estilo arcade aventureiro, desenvolvido na *Godot Engine* com *GDScript*, como um estudo de caso completo do ciclo de produção de um jogo 2D. O projeto contempla a implementação de mecânicas centrais, como movimentação, pulo, detecção de colisões, sistema de vida e *checkpoints*, além da integração de

ativos visuais e sonoros que reforcem a coesão temática. Por meio de uma abordagem iterativa de prototipação e depuração, espera-se demonstrar tanto a viabilidade técnica quanto o valor pedagógico de documentar cada etapa do processo, servindo como referência prática para futuros desenvolvedores interessados em projetos de jogos 2D em ambiente *open-source*.

1.1. JUSTIFICATIVA

O desenvolvimento de um jogo também se configura como um problema clássico de engenharia de software, pois exige lógica de programação, estruturas de dados para gerenciar objetos de jogo, arquiteturas de cena que organizem níveis e técnicas de otimização de desempenho. Ao optar pela *Godot Engine* e por *GDScript*, busco consolidar meus conhecimentos em linguagens de script e em sistemas baseados em *nodes* (componentes estruturais que organizam a hierarquia de uma cena), explorando uma ferramenta *open-source* que permite rápida prototipagem e fácil extensibilidade — competências essenciais para o desenvolvedor moderno.

Este projeto representa ainda a minha baixa experiência prévia em criação de jogos. Cada etapa do desenvolvimento, desde a configuração inicial da *engine* até a implementação de sistemas de pulo e *checkpoints*, passando pela integração de arte e som, foi documentada como parte de um processo pedagógico intencional, no qual erros e iterações se transformam em lições para quem deseja trilhar esse caminho.

Por fim, percebi ao longo do curso que as disciplinas técnicas enfatizam conteúdos como algoritmos, redes e bancos de dados, mas raramente tratam do desenvolvimento de jogos ou de aspectos de ludificação e experiência do usuário. Este TCC, portanto, visa preencher essa lacuna, servindo como referência prática para futuros estudantes interessados em *game design*, programação e produção de jogos 2D utilizando a *Godot Engine*.

1.2. OBJETIVOS

1.2.1. Objetivo Geral

Desenvolver, implementar e validar um protótipo de jogo eletrônico no estilo *arcade aventureiro*, utilizando a *Godot Engine* e *GDScript*, que sirva como estudo de caso completo sobre todo o ciclo de produção de um jogo 2D.

1.2.2. Objetivos Específicos

- Pesquisar fundamentos de *design de jogos* de plataforma e metodologias de desenvolvimento de software para embasar teoricamente as escolhas de mecânicas e arquitetura do protótipo.
- Elaborar o documento de requisitos e modelagem do sistema, incluindo explicações da lógica aplicada no jogo, diagrama de cenas e planejamento de níveis.
- Implementar as principais mecânicas do jogo — movimentação, pulo, física de colisões, lógica de mortes e sistema de progressão — utilizando *Godot Engine* e *GDScript*.
- Criar e integrar ativos visuais (sprites, *tilesets* e animações) e sonoros (trilha e efeitos) que reforcem a temática *arcade aventureira* e melhorem a imersão do jogador.

1.3. METODOLOGIA

A metodologia adotada neste trabalho seguiu uma sequência lógica de etapas, cada uma com objetivos e entregáveis específicos, visando garantir a coerência entre pesquisa, implementação prática e documentação final.

● Revisão Conceitual sobre Jogos Digitais e Mecânicas de Plataforma

A primeira etapa consistiu na investigação teórica de conceitos fundamentais relacionados ao desenvolvimento de jogos digitais, com ênfase em títulos de plataforma 2D. Foram estudados princípios de jogabilidade, elementos de design,

estrutura de fases e fundamentos da interação em ambientes bidimensionais. Essa revisão forneceu o embasamento necessário para orientar as decisões iniciais de design e delimitar o escopo funcional do protótipo.

● **Familiarização Técnica com a Godot Engine**

Na etapa seguinte, desenvolveu-se um período de exploração prática da *Godot Engine* e de sua linguagem nativa, *GDScript*. Foram realizados experimentos de pequeno porte para compreender o sistema de nós (nodes), o fluxo de cenas, o modelo de sinais (signals), o comportamento da física 2D e o uso de animações. Essa fase permitiu estabelecer domínio operacional sobre a ferramenta e definiu as abordagens técnicas empregadas no desenvolvimento.

● **Implementação Inicial das Mecânicas Essenciais**

Com a base conceitual e técnica estabelecida, iniciou-se a implementação das funcionalidades centrais do jogo. Essa etapa abrangeu a criação do sistema de movimentação do personagem, lógica de colisões, coleta de itens, funcionamento dos sistemas de pontuação, implementação de *checkpoints*, comportamento básico de inimigos e transições entre cenas. O processo ocorreu de forma iterativa, alternando testes internos, ajustes de parâmetros e correções de bugs até alcançar estabilidade funcional.

● **Produção e Estruturação da Documentação do Trabalho**

Após a consolidação das mecânicas principais, iniciou-se a elaboração formal da documentação do TCC. Essa etapa envolveu a organização das seções, levantamento dos requisitos funcionais presentes no jogo, descrição técnica dos sistemas desenvolvidos, registro dos procedimentos adotados e articulação entre fundamentação teórica, metodologia e desenvolvimento. Esse processo garantiu clareza e alinhamento entre a prática realizada e os objetivos acadêmicos do projeto.

● **Refinamento Final e Planejamento Estrutural dos Níveis**

Por fim, procedeu-se ao polimento geral do protótipo. Esse refinamento incluiu ajustes visuais, reposicionamento de obstáculos e entidades, equilíbrio da dificuldade, definição do fluxo das fases e integração final dos elementos temáticos,

como as Frutas Cósmicas. Essa fase assegurou a coerência da experiência de jogo e preparou o protótipo para demonstração e análise.

Essa metodologia integrada, que combina pesquisa teórica, prática de prototipação, testes e documentação sistemática, assegura a transparência do processo e a replicabilidade por outras pessoas interessadas em produzir jogos 2D em ambiente *open-source*.

2. REFERENCIAL TEÓRICO

O referencial teórico apresenta os conceitos fundamentais que sustentam o desenvolvimento deste projeto, fornecendo a base conceitual necessária para compreender suas escolhas técnicas e metodológicas. São discutidos temas relacionados ao design de jogos 2D, às engines de desenvolvimento, às características da Godot Engine e ao processo de prototipação aplicado ao ciclo de criação do protótipo.

2.1. JOGOS 2D E PRINCÍPIOS DE GAME DESIGN

Os jogos de plataforma 2D permanecem relevantes na indústria por combinarem simplicidade estrutural com profundidade mecânica. Schell (2019) destaca que jogos bidimensionais favorecem respostas rápidas, precisão e clareza visual, características que facilitam o engajamento do jogador. Salen e Zimmerman (2004) afirmam que elementos fundamentais do *game design* — como desafio, interação, regras e feedback — constituem a base para a criação de experiências consistentes. Esses princípios sustentam mecânicas clássicas de jogos 2D, como movimentação lateral, saltos e progressão por fases.

2.2. ENGINES PARA DESENVOLVIMENTO DE JOGOS

Engines de jogos são plataformas que fornecem ferramentas integradas para renderização, áudio, física, animação e scripts, acelerando o processo de desenvolvimento. Novak (2012) explica que esse tipo de ambiente reduz custos técnicos ao disponibilizar estruturas prontas que podem ser adaptadas ao projeto.

Entre as engines contemporâneas, a *Godot Engine* se destaca por ser *open-source* e permitir grande flexibilidade. De acordo com a documentação oficial da Godot Engine (2024), seu sistema baseado em cenas modulares e hierarquias de *nodes* favorece organização, reuso e rápida prototipagem.

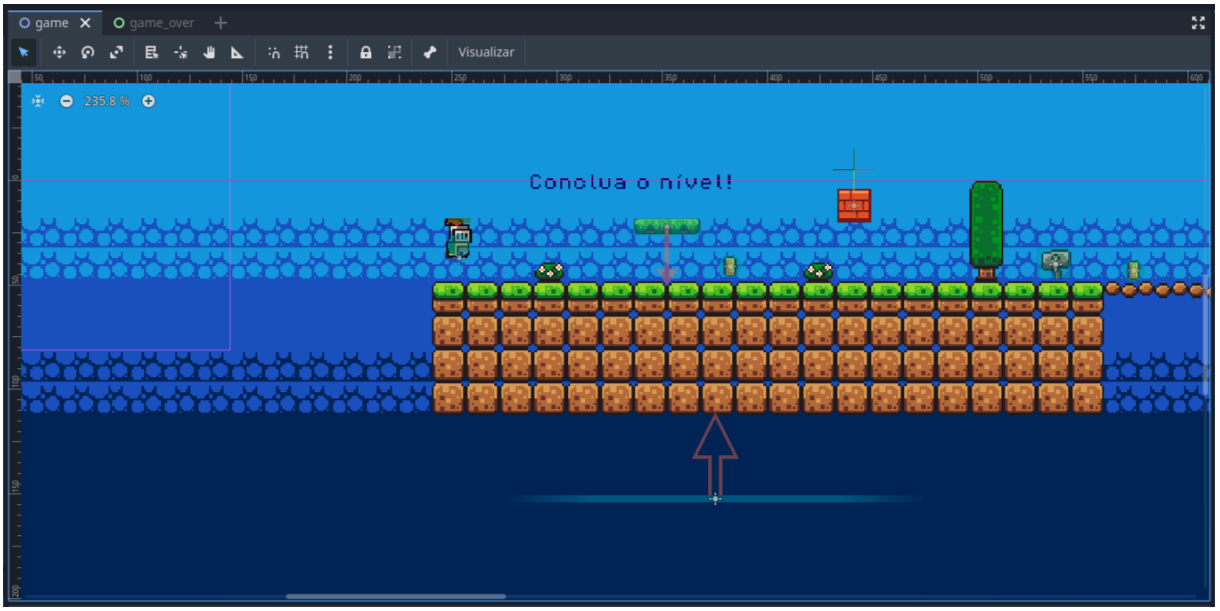
2.3. APRESENTANDO A GODOT ENGINE

Para a construção deste protótipo, optou-se pela *Godot Engine*, um motor de desenvolvimento de jogos 2D e 3D de código-aberto, leve e altamente extensível. Sua arquitetura baseada em cena possibilita organizar elementos do jogo de forma hierárquica e reaproveitável, enquanto a linguagem *GDScript* — inspirada em *Python* — oferece sintaxe simples e integração nativa com o editor. Além disso, a comunidade ativa e a documentação abrangente facilitam a adoção de boas práticas e a prototipação rápida. Esses fatores tornaram a *Godot* a escolha ideal para explorar mecânicas de plataforma, gerenciamento de estados de jogo e iteração ágil no ciclo de produção. A interface da *Godot Engine* pode ser dividida em quatro áreas principais:

- **Área de Visualização (*Viewport*)**

  É o painel central onde você monta e edita a cena propriamente dita. Nele você posiciona, redimensiona e organiza todos os elementos do seu nível em tempo real, vendo imediatamente como ficará o jogo em execução.

Figura 1 – Área de Visualização (*Viewport*)



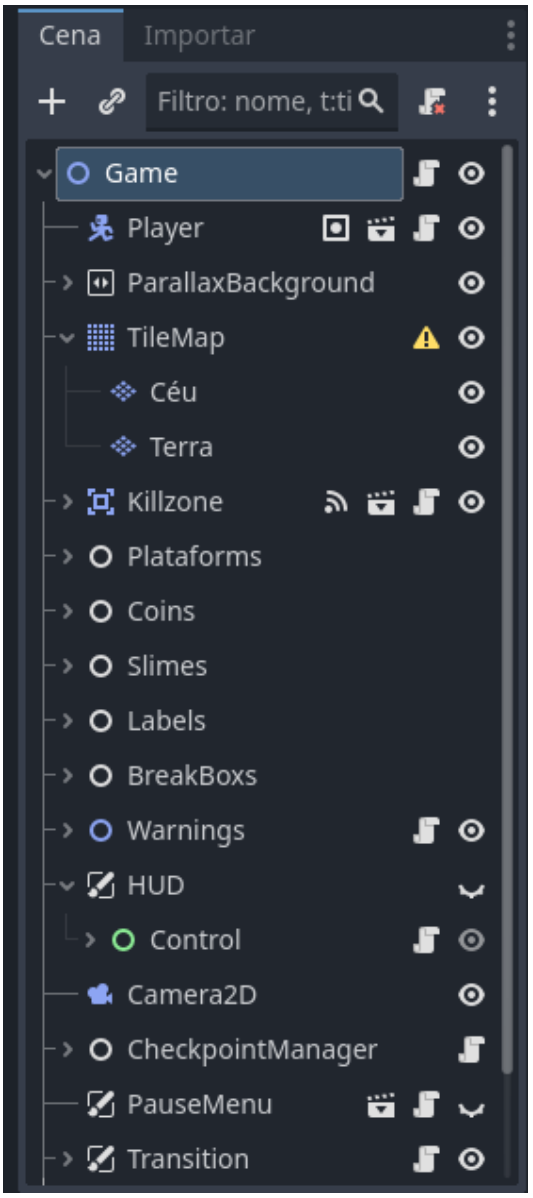
Fonte: Autoria própria, 2025.

- **Árvore de Cena (*Scene Tree*)**

  Localizada no canto superior esquerdo, exibe hierarquicamente todos os nós

que compõem a cena atual. A partir daí você cria, aninha ou remove elementos, definindo a estrutura de seu jogo de forma organizada e modular.

Figura 2 – Árvore de Cena (*Scene Tree*)

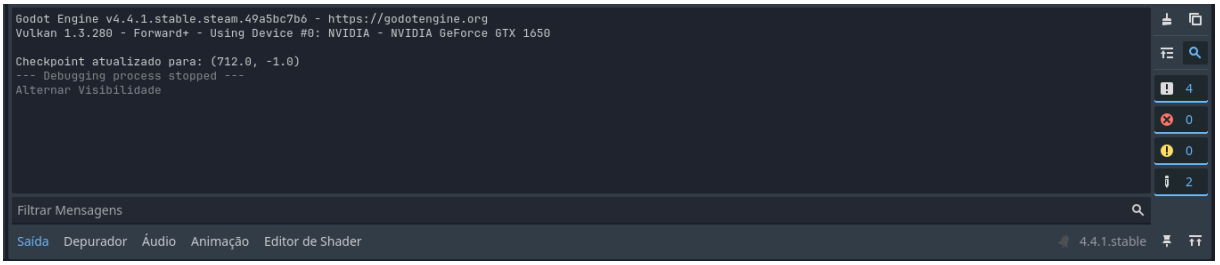


Fonte: Autoria própria, 2025.

- **Painel de Saída e Depuração (*Output/Debugger*)**

  Na parte inferior, exibe mensagens , erros, avisos e informações de depuração. Aqui pode conferir o resultado de erros em scripts, monitorar o desempenho, inspecionar variáveis em tempo real e usar as ferramentas de *profiling* para otimizar o jogo. Além de poder criar e alterar propriedades de animação e áudio.

Figura 3 – Painel de Saída e Depuração (*Output/Debugger*)

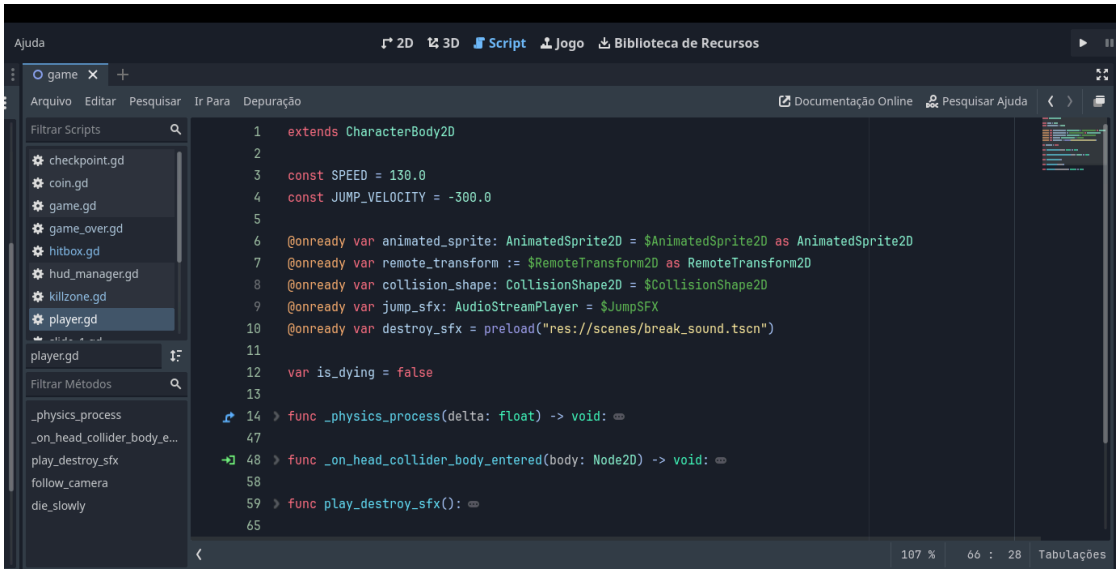


Fonte: Autoria própria, 2025.

- **Sessão de programação de scripts**

  É nesse local que são desenvolvidos os códigos cruciais para o funcionamento do projeto. São esses scripts que garantem o acontecimento de eventos necessários de acordo com o avanço do jogo.

Figura 4 – Sessão de programação de scripts



Fonte: Autoria própria, 2025.

2.4. PROTOTIPAÇÃO E PROCESSO ITERATIVO

O processo de prototipação é central no desenvolvimento de jogos porque permite testar ideias rapidamente e corrigir problemas antes da implementação definitiva. Fullerton (2018) descreve o desenvolvimento iterativo como uma prática composta por ciclos curtos de criação, teste, avaliação e refinamento, garantindo que cada mecânica atenda aos requisitos de jogabilidade. Essa abordagem foi

aplicada neste projeto na construção de sistemas como movimentação, coleta de itens, checkpoints e lógica de inimigos.

3. DESENVOLVIMENTO

Esta seção apresenta o processo do desenvolvimento do sistema proposto neste trabalho. Ela está dividida em 3 partes: documentação de requisitos, lógica do jogo e interfaces do sistema.

3.1. DOCUMENTAÇÃO DE REQUISITOS

Esta seção especifica os requisitos do sistema, que fornecem ao desenvolvedor as informações necessárias para a implementação do sistema.

3.1.1. Convenções, termos e abreviações

Por convenção, a referência a requisitos é feita através do nome da subseção onde eles estão descritos, seguidos do identificador do requisito, de acordo com a especificação a seguir:

[identificador do requisito – nome do requisito]

Os requisitos devem ser identificados com um identificador único. A numeração inicia com o identificador [RF01] e prossegue sendo incrementada à medida que forem surgindo novos requisitos.

3.1.2. Prioridades dos requisitos

Para estabelecer a prioridade dos requisitos foram adotadas as denominações “essencial”, “importante” e “desejável”.

● Essencial: é o requisito sem o qual o sistema não entra em funcionamento. São requisitos imprescindíveis, que devem ser implementados impreterivelmente.

● Importante: é o requisito sem o qual o sistema entra em funcionamento, mas de forma não satisfatória. Requisitos importantes devem ser implementados, mas, se não forem, o sistema poderá ser implantado e usado mesmo assim.

● Desejável: é o requisito que não compromete as funcionalidades básicas do sistema, isto é, o sistema pode funcionar de forma satisfatória sem ele. Requisitos desejáveis podem ser deixados para versões posteriores do sistema, caso não haja tempo hábil para implementá-los na versão que está sendo especificada.

3.1.3. Requisitos funcionais do sistema

Os requisitos funcionais do sistema descrevem, de forma objetiva, os comportamentos que o protótipo deve executar para garantir a jogabilidade e o funcionamento das mecânicas do jogo. Todos os requisitos apresentados foram elaborados integralmente pelo autor deste trabalho, a partir das necessidades identificadas durante o planejamento e a experimentação prática no desenvolvimento do jogo. Abaixo são listados os requisitos funcionais previstos para o sistema.

[RF001] Movimentar Personagem	
Descrição:	Controlar o jogador para andar à esquerda, à direita e parar, acionando animações correspondentes.
Prioridade:	■ Essencial □ Importante □ Desejável

[RF002] Pular com Física de Plataforma	
Descrição:	Executar o salto apenas em contato com o chão, aplicando gravidade e limites de altura/duração do pulo.
Prioridade:	■ Essencial □ Importante □ Desejável

[RF003] Detectar Colisões	
Descrição:	Identificar colisões entre personagem, cenário, inimigos e itens, gerando parada, dano ou coleta conforme o caso.
Prioridade:	■ Essencial □ Importante □ Desejável

[RF004] Gerenciar Vida do Jogador	
Descrição:	Exibir total de vezes que o jogador for morto em tempo real, sempre atualizando a cada nova morte.
Prioridade:	■ Essencial □ Importante □ Desejável

[RF005] Definir Checkpoint e Respawn	
Descrição:	Salvar posição e estado do jogador em pontos de verificação; retornar ao último ponto atingido após a sua morte.
Prioridade:	■ Essencial □ Importante □ Desejável

[RF006] Patrulhar Inimigos	
Descrição:	Fazer inimigos percorrerem um trecho do cenário, invertendo sua direção ao detectar bordas ou paredes.
Prioridade:	□ Essencial ■ Importante □ Desejável

[RF007] Eliminar Inimigo por Pulo	
Descrição:	Eliminar inimigo após o jogador pular em sua cabeça, gerando efeitos visuais e pontuação.
Prioridade:	□ Essencial ■ Importante □ Desejável

[RF008] Coletar Itens	
Descrição:	Permitir coleta de moedas que incrementam pontuação ou alteram atributos temporários.
Prioridade:	□ Essencial ■ Importante □ Desejável

[RF009] Mover e Quebrar Plataformas	
Descrição:	Suportar plataformas estáticas, plataformas móveis e plataformas quebráveis mediante impacto do jogador.
Prioridade:	□ Essencial ■ Importante □ Desejável

[RF010] Exibir Menu e Navegação	
Descrição:	Mostrar tela inicial com opções (Iniciar, Sair); permitir pausar (ESC), retomar e voltar ao menu principal.
Prioridade:	□ Essencial ■ Importante □ Desejável

[RF011] Mostrar HUD em Tempo Real	
Descrição:	Exibir total de mortes, moedas coletadas, inimigos derrotados e tempo restante durante o jogo.
Prioridade:	□ Essencial ■ Importante □ Desejável

[RF012] Transicionar entre Fases	
Descrição:	Carregar próxima cena ao coletar item de conclusão de fase, preservando estado do jogador.
Prioridade:	□ Essencial ■ Importante □ Desejável

[RF013] Reproduzir Áudio e Efeitos	
Descrição:	Tocar trilha de fundo contínua e efeitos sonoros sincronizados a eventos de pulo, coleta, dano, vitória e game over.
Prioridade:	□ Essencial ■ Importante □ Desejável

[RF014] Exibir Tela de Game Over	
Descrição:	Apresentar opções de reiniciar fase ou voltar ao menu quando o tempo de conclusão de fase se esgotar.
Prioridade:	□ Essencial ■ Importante □ Desejável

[RF015] Aplicar Efeitos de Transição	
Descrição:	Implementar camadas de transição com o intuito de dar suavidade ao jogo e aumentar a capacidade de imersão do usuário.
Prioridade:	□ Essencial □ Importante ■ Desejável

[RF016] Ajustar Nível Selecionado	
Descrição:	Permitir que o usuário acesse um menu que seja possível selecionar determinada fase de acordo com sua preferência.
Prioridade:	□ Essencial □ Importante ■ Desejável

3.2. LÓGICA DO JOGO

Nesta seção, descrevem-se os principais elementos que compõem o protótipo do jogo eletrônico em estilo *arcade aventureiro*. A explicação será acompanhada por figuras que complementam o texto, apresentando visualmente os componentes e suas funcionalidades. Cada imagem será referenciada diretamente no corpo do texto, garantindo a integração entre descrição e ilustração. O objetivo é apresentar, de forma clara e organizada, como cada elemento foi concebido, estruturado e implementado, evidenciando as escolhas de design e programação que contribuíram para o funcionamento do jogo.

3.2.1. Definição e licenciamento de *assets*

A definição dos *assets* foi a primeira etapa do desenvolvimento do protótipo. Esses elementos correspondem a todos os recursos visuais e sonoros presentes no jogo, como *sprites*, *tilesets*, animações, fontes, efeitos sonoros e trilha musical. O objetivo inicial foi assegurar que cada recurso atendesse ao estilo visual planejado e pudesse ser integrado à *Godot Engine* sem dificuldades técnicas.

Para evitar problemas relacionados a direitos autorais, optou-se pela utilização exclusiva de recursos licenciados sob *CC0 (Creative Commons Zero)*. Essa licença funciona como uma espécie de domínio público moderno, permitindo que os arquivos sejam utilizados, alterados e distribuídos livremente, sem a obrigação de atribuir crédito ao criador original. Essa escolha é particularmente importante em trabalhos acadêmicos, pois garante transparência legal e possibilita a reutilização futura do projeto sem restrições.

No contexto do jogo, *sprites* são imagens individuais que representam personagens, objetos, inimigos ou quadros de animação. Já os *tilesets* consistem em coleções organizadas de pequenos blocos gráficos utilizados para montar cenários de forma modular, como plataformas, pisos e paredes. A definição clara desses elementos foi fundamental para organizar a *pipeline* de arte, garantindo consistência visual e facilitando sua reutilização durante a construção das fases.

Por fim, a adoção de *assets* gratuitos e padronizados permitiu que o foco do trabalho permanecesse nas mecânicas e na implementação técnica, sem exigir produção artística autoral. Com isso, o processo tornou-se mais ágil e alinhado ao objetivo do TCC: demonstrar o ciclo completo de desenvolvimento de um jogo 2D, valorizando práticas de engenharia e documentação. Um exemplo desses recursos visuais podem ser observados na Figura 5.

Figura 5 – Assets das Frutas Cósmicas

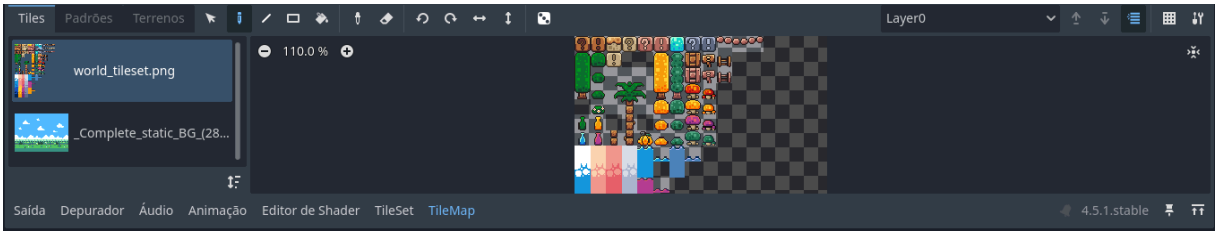
Fonte: Autoria própria, 2025.

3.2.2. Construção do cenário

A construção do cenário teve início com a organização dos *tilesets*, que são coleções de pequenos blocos gráficos utilizados para compor ambientes de forma modular. Esses elementos permitem criar pisos, plataformas e paredes mantendo consistência estética entre todas as partes do nível. Após a seleção dos *assets*, configurou-se o *tileset* na *Godot Engine*, definindo propriedades como colisões, regiões de navegação e áreas interativas, garantindo compatibilidade entre arte e mecânicas.

Em seguida, iniciou-se a montagem das fases utilizando o nó *TileMap*, que possibilita “desenhar” os cenários diretamente no editor. Essa ferramenta funciona como um mosaico, permitindo organizar rapidamente as estruturas principais do ambiente e facilitando alterações durante o processo. A modularidade do *TileMap* foi essencial para ajustar plataformas, corrigir alinhamentos e adaptar o design do nível conforme as necessidades do protótipo. O processo de pintura e organização visual do cenário no editor pode ser observado na Figura 6.

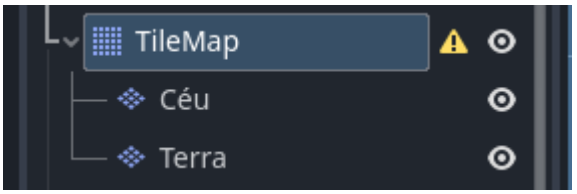
Figura 6 – Seleção de pintura do cenário



Fonte: Autoria própria, 2025.

Optou-se por utilizar um único *TileMap* estruturado em duas camadas internas: uma camada superior destinada ao céu, exclusivamente visual e sem interação física, e uma camada inferior destinada ao terreno, composta por blocos sólidos com *CollisionShape2D* para suportar a jogabilidade. Nesta camada de terreno, também foram inseridos elementos decorativos que não possuem colisão — permitindo enriquecer a ambientação sem interferir na navegação do player. Essa organização em camadas dentro do mesmo *TileMap* simplificou a gestão da cena, mantendo o controle sobre quais elementos participam da física do jogo e quais são meramente estéticos. Essa estrutura pode ser visualizada na hierarquia de nós apresentada na Figura 7.

Figura 7 – Árvore de nós do TileMap



Fonte: Autoria própria, 2025.

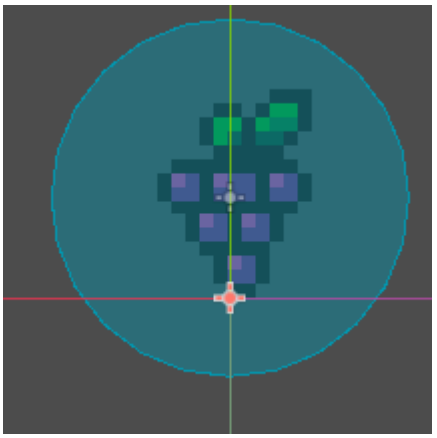
3.2.3. Sistema de colisões

O sistema de colisões do protótipo foi construído com base na infraestrutura padrão da *Godot Engine*, que utiliza *CollisionObject2D* como base para todos os elementos capazes de detectar ou responder a contatos físicos. No jogo, esse sistema é essencial para garantir que o jogador, plataformas, inimigos e elementos do cenário interajam de maneira consistente e previsível.

A ferramenta central para definir áreas colidíveis é o nó *CollisionShape2D*, responsável por atribuir uma forma geométrica simples — como retângulo ou

cápsula — a objetos que precisam participar do sistema de colisões. A escolha por formas simples deve-se ao fato de que elas oferecem melhor desempenho e são mais fáceis de ajustar visualmente, permitindo que cada área colidível cubra com precisão o *sprite* correspondente. Durante o desenvolvimento, cada *CollisionShape2D* foi definido manualmente para evitar espaços vazios ou sobreposições exageradas que pudessem comprometer a detecção da física. Entretanto, algumas situações fogem desse padrão, elementos podem ter áreas de colisão maiores a fim de ajudar o usuário, permitindo coletá-los com maior facilidade, como exemplificado na Figura 8.

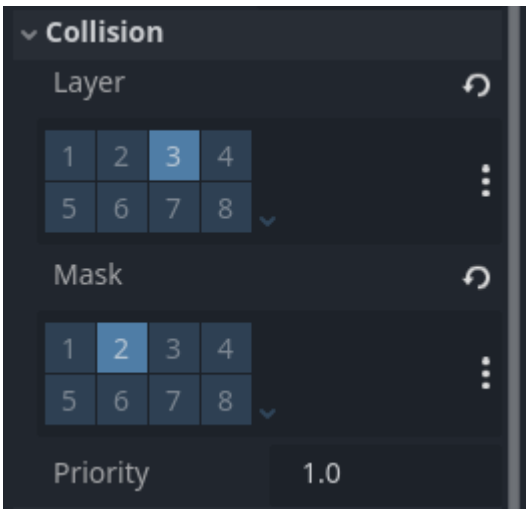
Figura 8 – Forma da colisão da Fruta Cósmica



Fonte: Autoria própria, 2025.

Complementando essa estrutura, o sistema de *layer* e *mask* foi configurado para organizar e controlar as interações físicas entre os elementos do jogo. Os *layers* definem a qual categoria cada objeto pertence, enquanto as *masks* determinam com quais categorias ele deve detectar colisão. Essa separação permite, por exemplo, que os inimigos colidam apenas com blocos e com o player, ignorando interações com moedas, pontos de *checkpoint* ou acessórios que poderiam gerar detecções desnecessárias. A configuração lógica dessas relações pode ser observada no painel apresentado na Figura 9, que centraliza o controle das camadas e máscaras de colisão.

Figura 9 – Painel de configuração lógica das colisões



Fonte: Autoria própria, 2025.

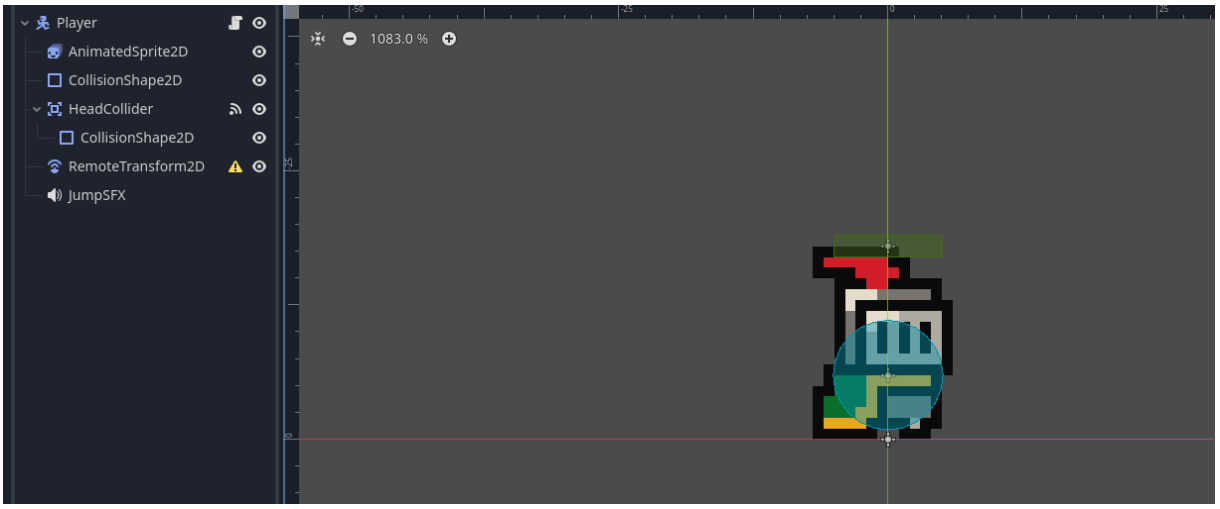
Ao longo da implementação, ajustes finos foram necessários para garantir a precisão do sistema. Pequenos erros no posicionamento de uma forma de colisão ou na configuração de *layer* e *mask* podem resultar em comportamentos inesperados, como o jogador “grudar” em quinas ou atravessar bordas inclinadas. Sendo assim, grande parte do refinamento consistiu em testar repetidamente diferentes valores, reposicionar formas e observar o comportamento em tempo real por meio da ferramenta de visualização de colisões da *Godot Engine*, que permite exibir contornos colidíveis durante a execução. Esse processo gradual assegurou um sistema robusto, coerente e adequado às mecânicas propostas.

3.2.4. Implementação do player

O personagem principal do jogo foi criado utilizando o nó *CharacterBody2D*, um tipo de objeto da *Godot Engine* voltado para personagens que necessitam de movimentação e física básica integradas. Dentro dessa estrutura foram adicionados componentes responsáveis por definir suas funções específicas, incluindo um *AnimatedSprite2D*, encarregado de exibir as animações de andar, pular e permanecer parado; um *CollisionShape2D*, que delimita a área física de interação do personagem com o cenário; um *Area2D* com uma colisão adicional posicionada acima da cabeça, utilizada para detectar interações com caixas acionadas por baixo; um *RemoteTransform2D*, responsável por sincronizar automaticamente a câmera com a posição do personagem; e um *AudioStreamPlayer*, utilizado para a

reprodução do som de pulo. A organização desses nós pode ser observada na Figura 10, que apresenta a cena do player e sua respectiva árvore de nós.

Figura 10 – Cena do player com a sua árvore de nós

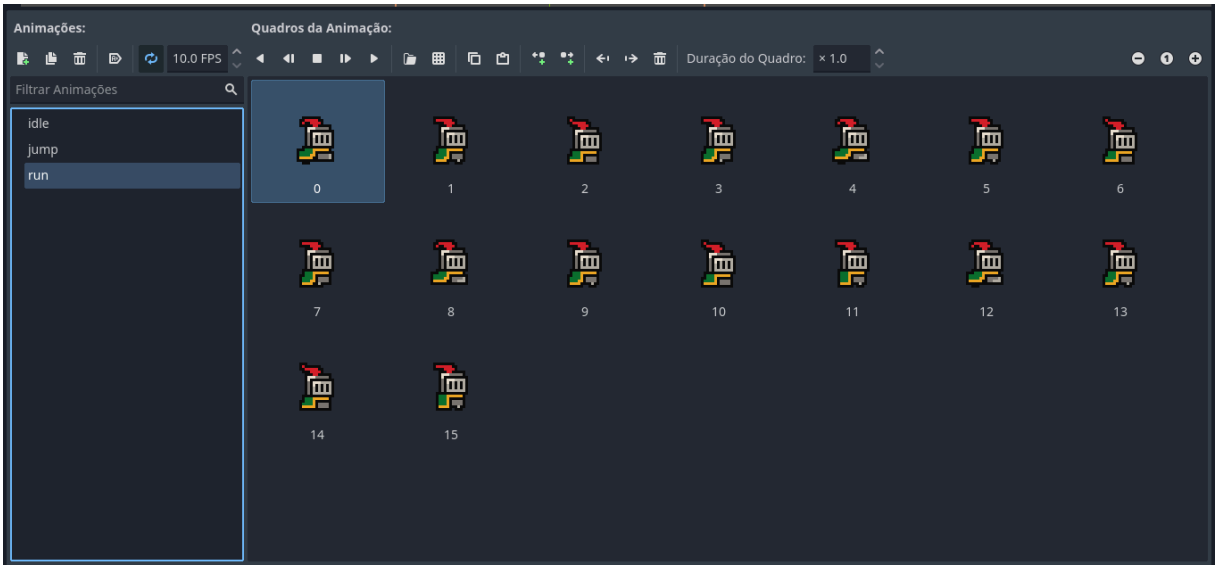


Fonte: Autoria própria, 2025.

A lógica de movimentação foi implementada no script principal do *CharacterBody2D* por meio da função *_physics_process()*, responsável por atualizar a física do jogador a cada ciclo de processamento. A movimentação horizontal é controlada por uma variável de velocidade que assume valores positivos ou negativos conforme o *input* do jogador. O pulo, por sua vez, depende de uma verificação de contato com o solo, assegurando que o salto seja executado apenas quando o personagem estiver apoiado. A gravidade é aplicada continuamente enquanto o personagem se encontra no ar, garantindo um comportamento previsível e natural durante o deslocamento.

Além do controle físico, o script também gerencia a troca de animações do personagem. A partir da análise da direção e da intensidade do movimento, o sistema alterna automaticamente entre as animações de deslocamento, salto ou estado ocioso. Esse comportamento visual é controlado diretamente pelo *AnimatedSprite2D*, cujo funcionamento pode ser visualizado na Figura 11, que demonstra o sistema de animações associado às ações do jogador.

Figura 11 – Sistema de animações do player



Fonte: Autoria própria, 2025.

Outro aspecto relevante é a detecção de colisões com objetos interativos por meio do *Area2D* posicionado acima da cabeça do personagem. Quando essa área entra em contato com caixas específicas do cenário, uma função é acionada, resultando na quebra do objeto e na liberação de moedas. Esse mecanismo garante maior precisão nas interações, evitando ativações indevidas por contato lateral. Por fim, o *RemoteTransform2D* mantém a câmera sincronizada ao jogador em tempo real, inclusive após eventos de *respawn*, assegurando continuidade visual, melhor leitura do cenário e fluidez na *gameplay*.

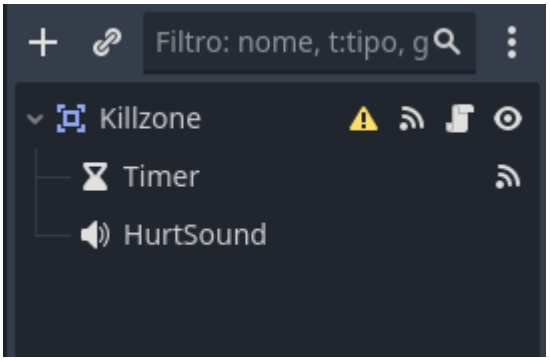
3.2.5. Padronização da Killzone

A opção por implementar uma *killzone* foi tomada com o intuito de centralizar e reaproveitar a lógica de eliminação do jogador em diferentes situações do jogo, como quedas fora do mapa ou contatos fatais causados por inimigos. A *killzone* consiste em uma área letal única, definida como um nó *Area2D* com um *CollisionShape2D* cobrindo a região desejada; sempre que um corpo entra nessa área, o sistema executa a rotina de morte associada. Essa padronização facilita a manutenção do projeto, pois diversos elementos (por exemplo, armadilhas ou comportamentos hostis) podem simplesmente emitir um sinal ou conferir colisão contra a mesma *killzone* em vez de duplicar a lógica em vários pontos.

Tecnicamente, a detecção é realizada por meio do sinal *body_entered* do *Area2D* (sinal que notifica quando um *CollisionObject2D* entra na área). Ao receber esse evento, a rotina de gerenciamento de morte executa ações como tocar efeitos sonoros, contabilizar o total de mortes e remover a instância do jogador da cena através de chamadas de limpeza (por exemplo, a função de liberação de nós). Essa abordagem assegura que a resposta seja imediata e consistente em todas as ocorrências, além de permitir tratamentos específicos *downstream*(qualquer ação que acontece após um evento principal) antes do reposicionamento.

No que diz respeito à integração com outros sistemas do jogo, os inimigos utilizam a mesma lógica de colisão letal. Quando o jogador entra em contato com a *killzone*, ele é removido da cena e, em seguida, o sistema realiza o *respawn* no último *checkpoint* registrado. A estrutura hierárquica e a organização dos nós responsáveis por essa funcionalidade podem ser observadas na Figura 12.

Figura 12 – Árvore de nós da Killzone



Fonte: Autoria própria, 2025.

Por fim, recomenda-se utilizar a visualização de colisões da Godot durante a fase de testes para validar a cobertura do *CollisionShape2D* e evitar falsos positivos ou pontos cegos. A centralização da lógica em uma *killzone* única simplifica testes, padroniza o tratamento de falhas e contribui para um código mais limpo e reutilizável, além de facilitar alterações futuras no comportamento letal sem necessidade de ajustar múltiplos objetos espalhados pela cena.

3.2.6. Criação dos inimigos

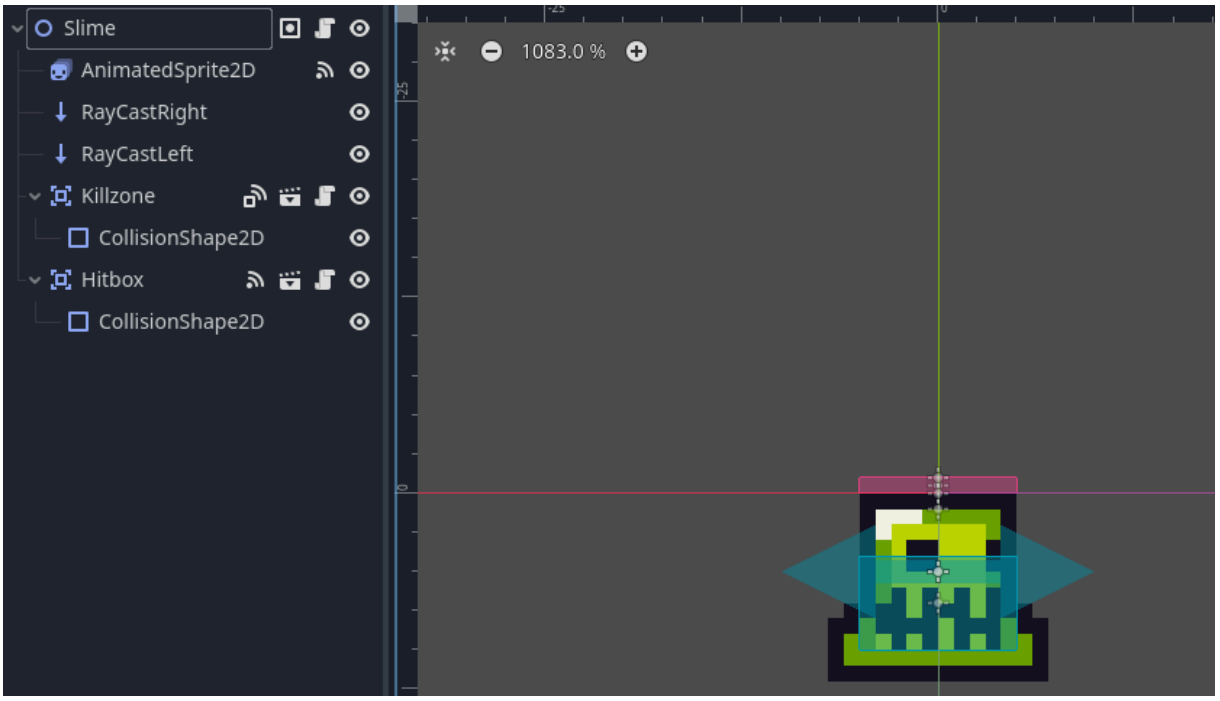
Os inimigos do protótipo foram construídos utilizando um nó *Node2D* como estrutura principal, ao qual foram adicionados os componentes necessários para comportamento, animação e detecção de colisões. O nó contém um *AnimatedSprite2D*, responsável por exibir a animação do inimigo em movimento, e dois *RayCast2D*, posicionados um à esquerda e outro à direita, utilizados para detectar quando o inimigo deve mudar de direção. Essa abordagem substitui o uso de física completa e permite criar um movimento simples, leve e eficiente.

A lógica de comportamento do inimigo é controlada por meio de um script que verifica constantemente se algum dos *RayCast2D* está em contato com um obstáculo. Quando o *RayCast* voltado para a direita detecta uma colisão, o inimigo inverte sua direção para a esquerda; quando o *RayCast* da esquerda colide, ele passa a se deslocar para a direita. Além disso, o sprite é invertido horizontalmente para manter a orientação visual coerente com o sentido do movimento. O deslocamento é aplicado diretamente no eixo X, resultando em um padrão contínuo de patrulhamento automático.

Para lidar com a interação com o jogador, o inimigo utiliza duas áreas de colisão distintas: uma *killzone*, localizada em seu corpo, e uma *hitbox* posicionada na região superior da cabeça. Caso o jogador entre em contato com a *killzone*, ele é removido da cena e retorna ao último *checkpoint* ativado, conforme descrito na seção dedicada ao sistema de eliminação. Por outro lado, ao atingir a *hitbox* superior, o inimigo é eliminado. O principal desafio dessa implementação consistiu em evitar a ativação simultânea das duas áreas, o que resultaria na eliminação do jogador e do inimigo ao mesmo tempo. Para solucionar esse problema, foram realizados diversos ajustes no tamanho e no posicionamento das colisões, garantindo que apenas a área correta fosse acionada em cada situação.

Após a finalização da lógica do inimigo, sua reutilização tornou-se simples dentro das fases, permitindo a inserção de múltiplas instâncias sem necessidade de reconfiguração adicional. A organização dos nós, bem como a separação clara entre animação, detecção de obstáculos e áreas de colisão, pode ser observada na Figura 13, que ilustra a estrutura completa da cena do inimigo.

Figura 13 – Cena do inimigo com a sua árvore de nós



Fonte: Autoria própria, 2025.

3.2.7. Itens coletáveis

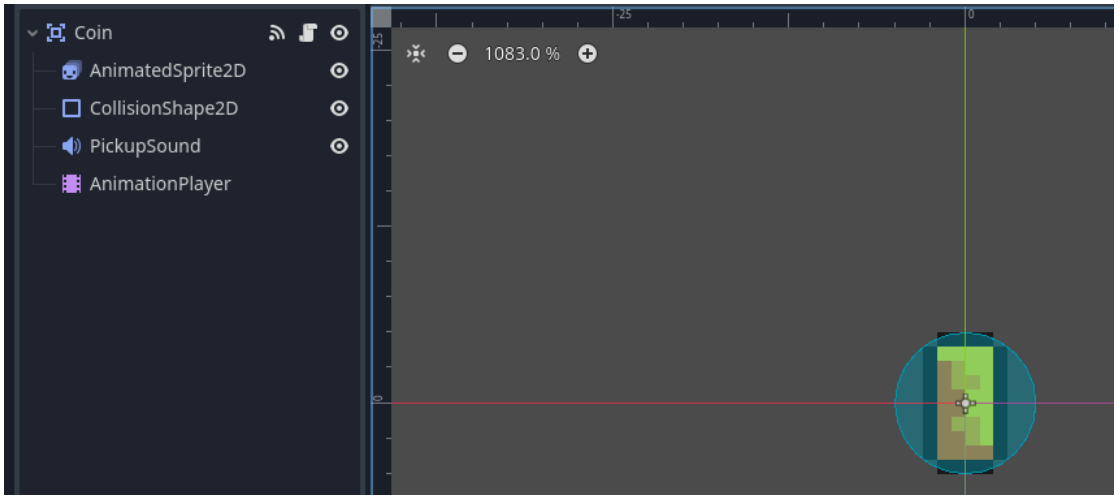
Os itens coletáveis desempenham um papel central no protótipo, servindo tanto como elemento de progressão quanto de recompensa ao jogador. Dois tipos principais foram implementados: as moedas, que funcionam como colecionáveis secundários, e as Frutas Cósmicas, que representam o objetivo principal de cada fase. Ambos foram estruturados utilizando o nó *Area2D*, que permite detectar quando o jogador entra em contato com eles por meio do sistema de sinais (*signals*).

As moedas foram projetadas para reforçar a sensação de exploração e oferecer um feedback constante ao jogador. Cada moeda é composta por um *Area2D* com um *AnimatedSprite2D*, responsável por exibir uma animação contínua de rotação, tornando o item mais visível e atrativo no cenário. O item possui também um *CollisionShape2D*, que define a área de coleta, além de um *AudioStreamPlayer* e um *AnimationPlayer*. Este último contém duas animações: a animação padrão, que mantém a moeda ativa e visível na fase, e a animação de coleta, acionada quando o jogador entra em contato com o item.

No momento da coleta, um *signal* é emitido, a área de colisão é removida, o sprite torna-se invisível e o efeito sonoro é reproduzido, indicando claramente a

interação com o jogador. Simultaneamente, o contador de moedas é incrementado, reforçando a função do item como recompensa. A organização desses componentes e a hierarquia de nós utilizada para implementar o comportamento da moeda podem ser observadas na Figura 14.

Figura 14 – Cena da moeda com a sua árvore de nós

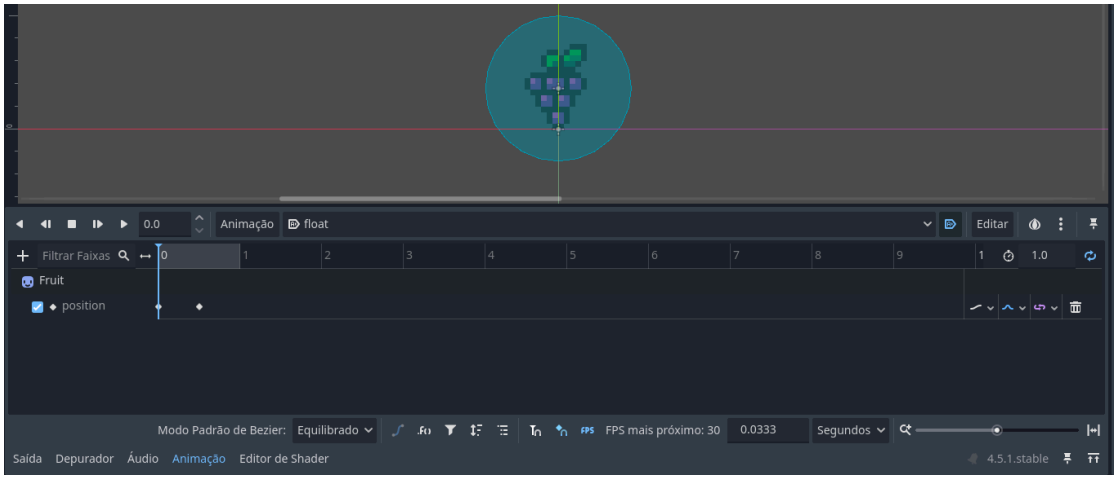


Fonte: Autoria própria, 2025.

As frutas cósmicas representam o item mais importante do jogo, uma vez que são responsáveis por permitir a progressão entre as fases. Assim como as moedas, elas foram implementadas como um *Area2D*, porém com uma estrutura mais simples, composta por um *Sprite2D*, um *CollisionShape2D* e um *AnimationPlayer*. A animação criada para a fruta possui caráter puramente decorativo, simulando um movimento suave de flutuação, o que contribui para destacá-la visualmente no cenário.

Quando o jogador entra em contato com a fruta, um *signal* é disparado e uma função de mudança de cena (*change_scene_to_file()*) é executada, direcionando o jogador a um menu de vitória. Nesse menu, o jogador pode optar por retornar ao menu principal ou avançar para a próxima fase, reforçando o papel da Fruta Cósmica como o principal objetivo de cada nível. A utilização do *AnimationPlayer* para controlar a animação decorativa e a organização desse componente dentro da cena da fruta podem ser observadas na Figura 15.

Figura 15 – Manipulação de animações via *AnimationPlayer*



Fonte: Autoria própria, 2025.

3.2.8. Informações e estatísticas

A *HUD* (Head-Up Display) foi implementada utilizando um nó base do tipo *Control*, que serve como contêiner principal para todos os elementos visuais exibidos permanentemente na tela. Dentro desse nó foram adicionados quatro *HBoxContainer*, responsáveis por organizar cada grupo de informações de forma horizontal. O uso de *HBoxContainer* é fundamental porque permite alinhar ícones e textos de maneira estruturada, mantendo espaçamentos proporcionais, evitando sobreposições e garantindo que a interface permaneça legível em diferentes resoluções. Cada *HBoxContainer* funciona como um agrupamento isolado, facilitando ajustes individuais, estilização e manutenção da interface.
As quatro estatísticas exibidas pela *HUD* são:

- **Quantidade de moedas coletadas:** mostrada por um *Label* acompanhado do ícone correspondente. Sempre que o jogador coleta uma moeda, essa moeda emite um *signal* para a *HUD*, que incrementa o contador e atualiza o texto do *Label* imediatamente. Esse sistema evita verificações constantes e mantém o código desacoplado.

- **Tempo restante da fase:** o tempo é decrementado a partir de um *Timer* ou contador interno e exibido em tempo real em um *Label* específico. A cada segundo, o valor é atualizado. Caso o cronômetro chegue a zero, a fase é

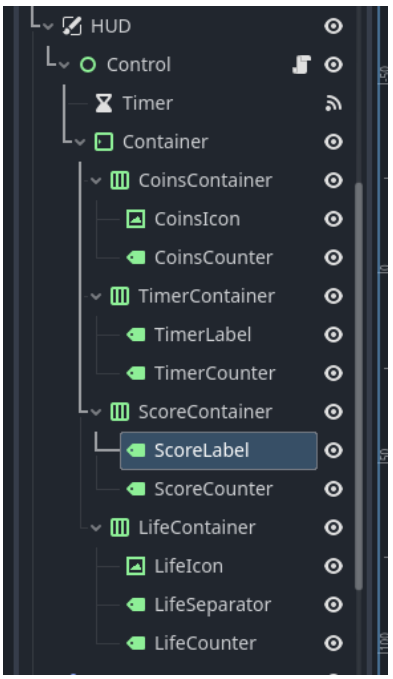
encerrada automaticamente e o jogador é redirecionado para uma tela de *game over*, caracterizando falha na missão.

- **Pontuação (*score*) do jogador:** essa estatística aumenta conforme inimigos são derrotados. Cada inimigo, ao ser eliminado, emite um *signal* contendo o valor de pontos a ser adicionado. A *HUD* recebe esse valor e soma ao total, atualizando o *Label* correspondente.

- **Quantidade de mortes do jogador durante a fase:** o contador é incrementado toda vez que o jogador é derrotado e retorna ao último ponto de controle. A lógica de respawn emite um *signal* indicando a morte, e a *HUD* atualiza o valor exibido.

Dentro de cada *HBoxContainer* também foram adicionados elementos decorativos, como ícones e separadores, que auxiliam o jogador na rápida identificação do tipo das informações. Além disso, pequenas animações ou efeitos podem ser incluídos para reforçar momentos específicos, como o aumento no número de moedas. A árvore de nós da *HUD* é exibida na figura 16.

Figura 16 – Árvore de nós da *HUD*



Fonte: Autoria própria, 2025.

3.3. INTERFACES

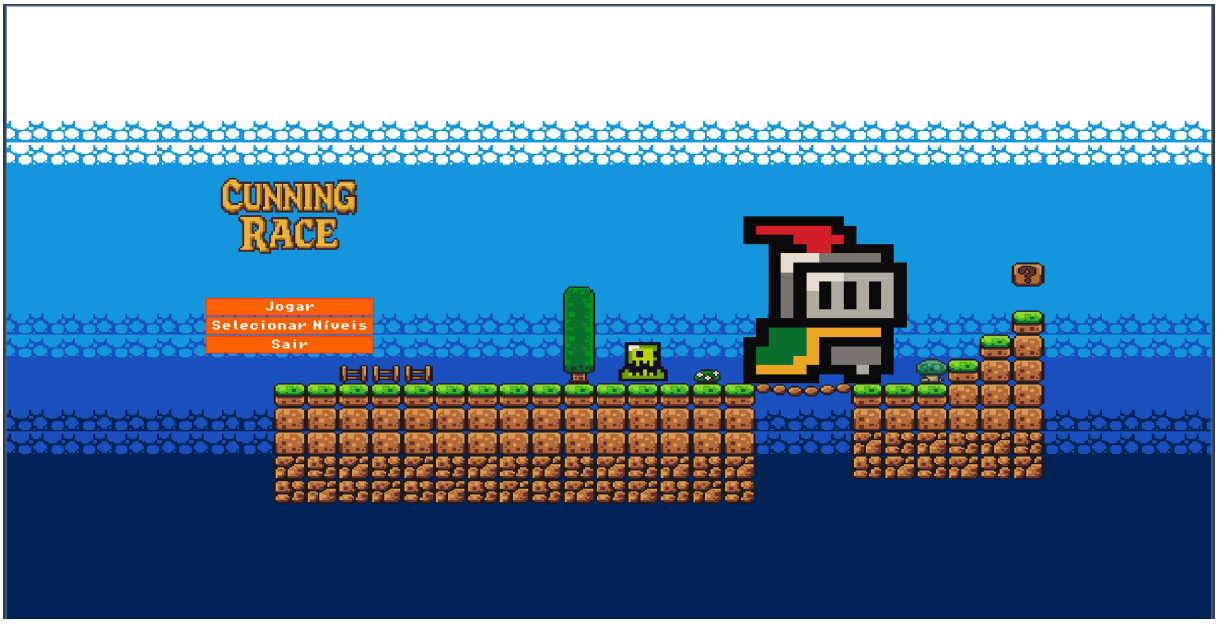
Para complementar a explicação textual do funcionamento do jogo, este trabalho apresenta imagens capturadas em momentos relevantes da execução do protótipo, com foco nas interfaces e menus que compõem cada seção do sistema. Essas imagens representam telas do jogo, evidenciando visualmente as opções disponíveis ao jogador e os estados fundamentais da aplicação.

As capturas foram selecionadas de forma intencional para documentar os principais menus — sendo a tela inicial, seleção de fases, pausa, vitória e game over, além de uma imagem da jogatina. Dessa forma, as imagens permitem identificar com clareza a estrutura das interfaces e o papel de cada elemento exibido na tela.

A utilização dessas representações visuais contribui para uma compreensão mais objetiva do sistema desenvolvido, uma vez que estabelece uma correspondência direta entre a descrição textual e a interface apresentada ao usuário. Assim, as imagens funcionam como suporte documental, reforçando a organização, a navegação e a lógica das seções do jogo descritas ao longo deste capítulo.

A Figura 17 exibe a tela inicial do jogo, composta pelos botões “Jogar”, “Selecionar Fases” e “Sair”. O botão “Jogar” carrega diretamente a Fase 1 com todos os indicadores reiniciados. O botão “Selecionar Fases” direciona ao menu de fases, enquanto “Sair” encerra a aplicação. O plano de fundo contém a arte estática principal do projeto.

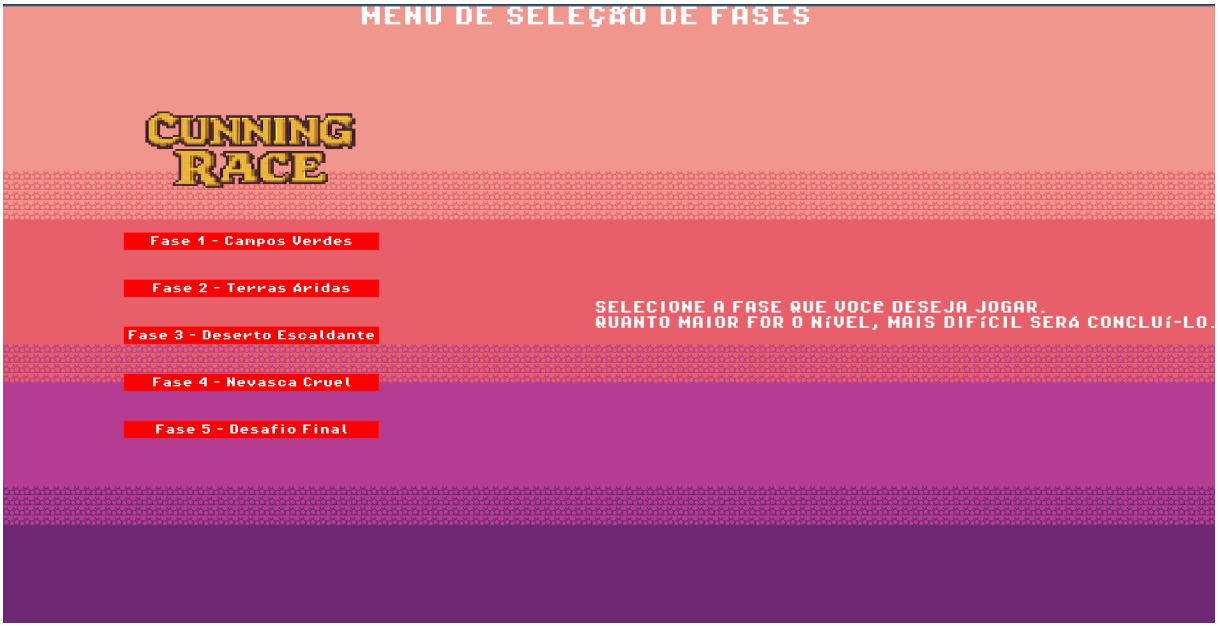
Figura 17 – Menu inicial do jogo



Fonte: Autoria própria, 2025.

A Figura 18 ilustra a interface de seleção de fases, que reúne cinco botões organizados verticalmente: “Fase 1 – Campos Verdes”, “Fase 2 – Terras Áridas”, “Fase 3 – Deserto Escaldante”, “Fase 4 – Nevasca Cruel” e “Fase 5 – Desafio Final”. Cada botão carrega sua respectiva cena e reinicia os parâmetros do jogo. A arte visual segue o padrão do menu inicial.

Figura 18 – Menu de seleção de fases



Fonte: Autoria própria, 2025.

A Figura 19 demonstra o menu de pausa ativado durante a jogabilidade. O painel apresenta os botões “Continuar”, que retoma a partida; “Voltar ao Menu”, que retorna ao menu inicial; e “Sair”, que encerra o jogo. O cenário permanece visível ao fundo e toda a cena do jogo junto do tempo são pausados.

Figura 19 – Menu de pausa

Fonte: Autoria própria, 2025.

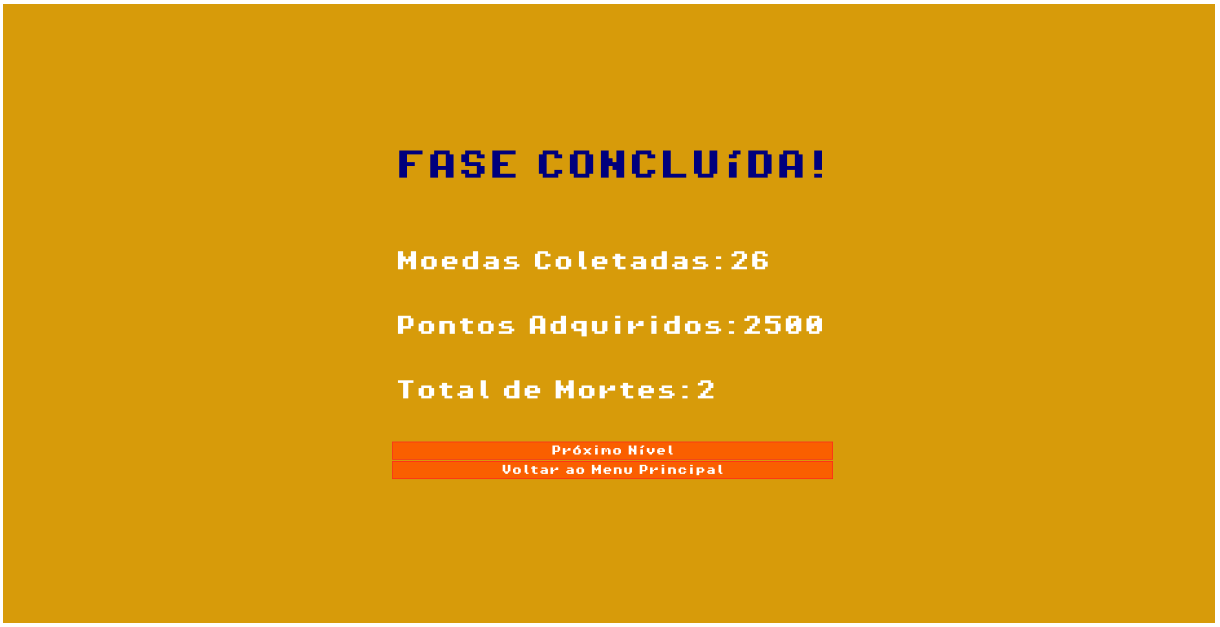
A Figura 20 apresenta um momento típico da jogabilidade, exibindo o personagem principal em deslocamento pelo cenário da fase. Na tela, observa-se o HUD completo: contador de moedas, tempo restante, pontuação acumulada e número de mortes registradas. O ambiente contém plataformas, inimigos e elementos interativos que compõem a progressão do nível. Os inimigos executam seus comportamentos programados de patrulha e ataque, enquanto o jogador navega pelo espaço por meio das mecânicas de movimento, pulo e interação. A imagem representa o funcionamento integrado dos sistemas de colisão, física, animações e lógica geral do jogo durante uma partida ativa.

Figura 20 – Gameplay de *Cunning Race*

Fonte: Autoria própria, 2025.

A Figura 21 descreve a tela exibida após a coleta da Fruta Cósmica. O painel central apresenta a mensagem “Fase Concluída” e mostra os indicadores finais de moedas, pontuação e total de mortes. Os botões disponíveis permitem avançar para a próxima fase ou retornar ao menu principal.

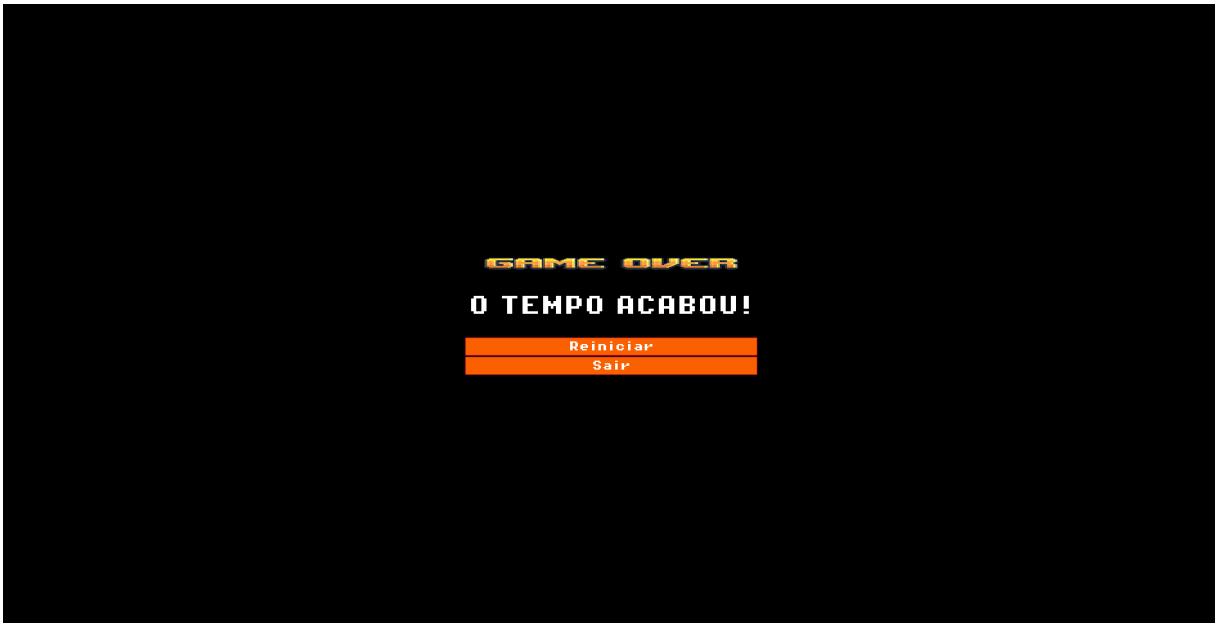
Figura 21 – Tela de conclusão da fase



Fonte: Autoria própria, 2025.

A Figura 22 mostra a tela exibida quando o tempo para a conclusão da fase se encerra. O painel central contém a mensagem “Game Over” acompanhada dos botões “Reiniciar”, que leva o usuário para o menu principal, e “Sair”, que encerra o jogo. A cena possui um fundo completamente escuro.

Figura 22 – Tela de game over



Fonte: Autoria própria, 2025.

4. CONSIDERAÇÕES FINAIS

O desenvolvimento do protótipo de jogo eletrônico no estilo arcade aventureiro permitiu consolidar, na prática, conhecimentos fundamentais de programação, arquitetura de software, design de jogos e uso de ferramentas profissionais de desenvolvimento. A escolha da *Godot Engine* e da linguagem *GDScrip*t mostrou-se acertada, pois possibilitou a criação de um fluxo de trabalho acessível, modular e eficiente, adequado tanto para iniciantes quanto para desenvolvedores que desejam compreender, de forma mais aprofundada, o funcionamento de engines baseadas em nós.

Ao longo do processo, foi possível compreender de maneira concreta como diferentes áreas da Computação se integram na produção de um jogo digital. Conceitos tradicionalmente vistos de forma isolada — como lógica de programação, gerenciamento de estados, estruturação de cenas, manipulação de colisões e uso de sinais — passaram a atuar de maneira conjunta para compor um sistema interativo completo. Além disso, desafios como depuração de erros, ajuste de parâmetros, organização de scripts e implementação de mecânicas responsivas fortaleceram habilidades essenciais do desenvolvimento de software, incluindo pensamento iterativo, resolução de problemas e documentação do processo.

Outro aspecto relevante foi o caráter pedagógico do projeto. A ausência de experiência prévia com criação de jogos transformou todo o percurso em uma oportunidade de aprendizagem ativa, na qual cada obstáculo exigiu pesquisa, experimentação e refinamento. Esse ciclo contínuo de tentativa e correção contribuiu para uma compreensão mais profunda de como jogos 2D são estruturados em engines modernas, permitindo o domínio gradual de conceitos como física de plataforma, checkpoints, *HUD*, sistemas de coleta e comportamento de inimigos.

O produto final — embora simples enquanto protótipo — cumpre seu papel como estudo de caso e demonstração das etapas fundamentais necessárias para a construção de um jogo funcional. O projeto evidencia que ferramentas open-source, como a *Godot Engine*, oferecem um ambiente rico e acessível para estudantes que desejam explorar o desenvolvimento de jogos, mesmo sem experiência anterior.

Como trabalhos futuros, destaca-se a possibilidade de expandir o protótipo para um jogo completo, incluindo novas fases, inimigos variados, menus adicionais, efeitos sonoros mais elaborados, narrativa e maior polimento visual. Também seria

possível investigar técnicas de otimização, implementação de física avançada e integração com bancos de dados ou *APIs* externas. Por fim, recomenda-se que futuros estudos aprofundem a relação entre ludificação e ensino, explorando como jogos simples podem auxiliar na aprendizagem de conceitos computacionais, matemáticos e lógicos.

Dessa forma, este trabalho contribui tanto para o desenvolvimento acadêmico do autor quanto para a formação de outros estudantes interessados na área, servindo como um registro prático e didático de todo o processo de criação de um jogo 2D em ambiente *open-source.*

REFERÊNCIAS

FULLERTON, Tracy. *Game Design Workshop: A Playcentric Approach to Creating Innovative Games*. 4. ed. Boca Raton: CRC Press, 2018.

GODOT ENGINE. *Documentation – Node and Scene System*. 2024. Disponível em: https://docs.godotengine.org. Acesso em: 29 nov. 2025.

HEARN, John. *Godot Engine Game Development Projects*. Birmingham: Packt Publishing, 2021.

NOVAK, Jeannie. *Game Development Essentials: An Introduction*. 3. ed. Clifton Park: Delmar Cengage Learning, 2012.

SALEN, Katie; ZIMMERMAN, Eric. *Rules of Play: Game Design Fundamentals*. Cambridge: MIT Press, 2004.

SCHELL, Jesse. *The Art of Game Design: A Book of Lenses*. 3. ed. Boca Raton: CRC Press, 2019.