v7.n11.2026

REVISTA CIENTÍFICA - RECIMA21 ISSN 2675-6218

DESENVOLVIMENTO DE JOGOS UTILIZANDO A PLATAFORMA GODOT

GAME DEVELOPMENT USING THE GODOT PLATFORM

DESARROLLO DE VIDEOJUEGOS UTILIZANDO LA PLATAFORMA GODOT

Higor Robert Barbist[1], Felipe Diniz Dallilo[2]

e7119462
https://doi.org/10.47820/recima21.v7i11.9462

PUBLICADO: 11/2026

RESUMO

Este trabalho relata o desenvolvimento de um protótipo de jogo digital bidimensional (2D) com mecânicas de combate por turnos, cartas e rolagem de dados, utilizando a plataforma de código aberto Godot Engine, versão 4.3. O objetivo foi descrever o processo de criação e as funcionalidades estruturais da plataforma para discutir sua viabilidade, sua usabilidade e sua produtividade para desenvolvedores independentes e programadores iniciantes. A pesquisa, de caráter exploratório e qualitativo, adotou o estudo de caso e a pesquisa bibliográfica, com base na documentação oficial da plataforma e no registro do código-fonte e de testes funcionais. O protótipo foi organizado em módulos, por meio de cenas independentes e nós (*Nodes*), com a lógica programada em GDScript. Na experiência do desenvolvedor, a organização em cenas e nós favoreceu o reaproveitamento de componentes e a manutenção do projeto, e foi possível obter um protótipo funcional sem infraestrutura complexa. Por se tratar de um único caso, avaliado pelo próprio desenvolvedor, sem medições sistemáticas de desempenho nem testes com usuários iniciantes, os resultados não podem ser generalizados. Conclui-se que a Godot Engine se mostrou uma alternativa viável para a prototipagem de jogos 2D por desenvolvedores independentes, e sugere-se, como trabalhos futuros, a realização de testes com iniciantes e de comparações com outras *engines*.

PALAVRAS-CHAVE: Arquitetura de nós. Desenvolvimento de jogos. Desenvolvedor independente. GDScript. Godot Engine.

ABSTRACT

This paper reports the development of a two-dimensional (2D) digital game prototype featuring turn-based combat, card, and dice-rolling mechanics, using the open-source Godot Engine, version 4.3. The objective was to describe the creation process and the structural features of the platform in order to discuss its viability, usability, and productivity for independent developers and beginner programmers. The exploratory, qualitative study adopted a case-study approach and a literature review, based on the platform's official documentation and on records of the source code and functional tests. The prototype was organized into modules through independent scenes and nodes, with the game logic programmed in GDScript. In the developer's experience, the scene-and-node organization favored component reuse and project maintenance, and a functional prototype was obtained without complex infrastructure. Because the study covers a single case, evaluated by the developer who built it, without systematic performance measurements or tests with novice users, the results cannot be generalized. It is concluded that Godot Engine proved to be a viable alternative for prototyping 2D games by independent developers, and future work should include tests with beginners and comparisons with other engines.

[1] Graduando em Sistemas de Informação pela Universidade de Araraquara (UNIARA).
[2] Graduado em Processamento de Dados pela FATEC (2009). Mestre e Doutor em Engenharia de Software pela USP (2014 e 2025). Possui MBA em Gestão de Pessoas, MBA em Gestão de Projetos e MBA em Data Science & Analytics.

Este artigo é publicado em acesso aberto (Open Access) sob a licença Creative Commons Atribuição 4.0 Internacional (CC-BY), que permite uso, distribuição e reprodução irrestritos em qualquer meio, desde que o autor original e a fonte sejam creditados.

REVISTA CIENTÍFICA - RECIMA21 ISSN 2675-6218

KEYWORDS: *Node architecture. Game development. Independent developer. GDScript. Godot Engine.*

RESUMEN
Este trabajo presenta el desarrollo de un prototipo de videojuego digital bidimensional (2D) con mecánicas de combate por turnos, cartas y lanzamiento de dados, utilizando la plataforma de código abierto Godot Engine, versión 4.3. El objetivo fue describir el proceso de creación y las funcionalidades estructurales de la plataforma para discutir su viabilidad, su usabilidad y su productividad para desarrolladores independientes y programadores principiantes. La investigación, de carácter exploratorio y cualitativo, adoptó el estudio de caso y la revisión bibliográfica, con base en la documentación oficial de la plataforma y en el registro del código fuente y de pruebas funcionales. El prototipo se organizó en módulos, mediante escenas independientes y nodos (Nodes), con la lógica programada en GDScript. En la experiencia del desarrollador, la organización en escenas y nodos favoreció la reutilización de componentes y el mantenimiento del proyecto, y fue posible obtener un prototipo funcional sin infraestructura compleja. Por tratarse de un único caso, evaluado por el propio desarrollador y sin mediciones sistemáticas de rendimiento ni pruebas con usuarios principiantes, los resultados no pueden generalizarse. Se concluye que Godot Engine se mostró como una alternativa viable para la creación de prototipos de juegos 2D por desarrolladores independientes y, como trabajos futuros, se sugieren pruebas con principiantes y comparaciones con otros motores de juego.

PALABRAS CLAVE: *Arquitectura de nodos. Desarrollo de videojuegos. Desarrollador independiente. GDScript. Godot Engine.*

INTRODUÇÃO

O desenvolvimento de jogos consolidou-se como uma área em constante expansão, que vai além do domínio de grandes corporações. Esse cenário, associado à popularização dos motores de jogos (*game engines*), ampliou o espaço dos desenvolvedores independentes, que atualmente conseguem produzir, divulgar e comercializar seus títulos com recursos próprios.

Entre as plataformas disponíveis, a Godot ganha destaque por suas ferramentas e funcionalidades de alta usabilidade e produtividade e por ser de código aberto, o que favorece uma curva de aprendizado eficiente para desenvolvedores iniciantes e, ao mesmo tempo, comporta a complexidade de projetos maiores. Segundo Silva e Yepes (2018), o uso de ferramentas de alta usabilidade e produtividade é essencial para democratizar o acesso à tecnologia criativa, permitindo que o foco permaneça na inovação e na qualidade do produto desenvolvido.

A relevância deste trabalho está no fato de que o desenvolvimento de jogos atual permite criar projetos profissionais com ferramentas de baixo consumo de *hardware*, em contraposição ao senso comum de que a área exige produções de grande porte e gráficos ultrarrealistas. Essa diferença de escopo é evidente em estudos locais sobre desenvolvimento de jogos na Universidade de Araraquara, como o de Malara (2024), que detalha a alta exigência de *hardware*

Este artigo é publicado em acesso aberto (Open Access) sob a licença Creative Commons Atribuição 4.0 Internacional (CC-BY), que permite uso, distribuição e reprodução irrestritos em qualquer meio, desde que o autor original e a fonte sejam creditados.

e a complexidade técnica de projetos na Unreal Engine 5. Nesse contexto, o presente estudo ilustra uma aplicação prática de projeto independente na Godot, que se destaca por ser de código aberto (*open source*) e, por isso, mais acessível do que outras *engines* de grande porte, o que lhe confere relevância social para os desenvolvedores independentes brasileiros.

Um dos obstáculos do desenvolvimento de jogos é a percepção equivocada do que a área realmente exige. A ideia de que ela demanda alta complexidade, equipes numerosas e equipamentos avançados desestimula novos talentos e leva ao abandono de projetos com potencial de impacto pessoal e profissional. Diante disso, este trabalho busca responder à seguinte questão: de que maneira a arquitetura e as ferramentas da Godot Engine facilitam o ingresso de desenvolvedores iniciantes na criação de jogos 2D? A hipótese levantada é que a estrutura baseada em cenas e nós (*Nodes*), aliada à linguagem GDScript, reduz a barreira técnica de entrada e permite que um desenvolvedor solo ou iniciante entregue um produto coeso e funcional sem infraestrutura complexa.

O objetivo geral deste trabalho é desenvolver um jogo 2D utilizando a plataforma Godot, analisando o processo de criação e as principais funcionalidades da ferramenta para discutir sua viabilidade como ambiente amigável e produtivo para novos desenvolvedores independentes. Os objetivos específicos são: (a) descrever a arquitetura da Godot Engine, baseada em cenas e nós, e a linguagem GDScript; (b) implementar, em um protótipo, as mecânicas de combate por turnos, cartas e rolagem de dados; (c) registrar as etapas de desenvolvimento e as decisões técnicas adotadas; e (d) discutir as potencialidades e as limitações da plataforma com base na experiência obtida.

1. REFERENCIAL TEÓRICO

Esta seção apresenta os fundamentos teóricos que sustentam o desenvolvimento do protótipo, com base em obras da área e em projetos similares, de modo a conferir maior confiabilidade às ideias desenvolvidas.

1.1. Conceitos fundamentais do desenvolvimento de jogos

Para a compreensão do projeto, são definidos a seguir alguns termos importantes de desenvolvimento de jogos e de tecnologia, de modo a oferecer ao leitor uma base conceitual para a leitura do trabalho.

Este artigo é publicado em acesso aberto (Open Access) sob a licença Creative Commons Atribuição 4.0 Internacional (CC-BY), que permite uso, distribuição e reprodução irrestritos em qualquer meio, desde que o autor original e a fonte sejam creditados.

1.1.1. *Game engine* (motor de jogo)

O motor de jogo fornece ao programador elementos essenciais e ferramentas prontas, como a física de objetos e a iluminação, o que evita que ele recrie funcionalidades básicas presentes na maioria dos jogos. Dessa forma, o desenvolvedor concentra seus esforços na parte artística, como a música e os elementos visuais, e na lógica das regras e das mecânicas.

A principal vantagem dessa organização, comum nas *engines* modernas, é o reaproveitamento desses elementos (GREGORY, 2018), que reduz a complexidade técnica enfrentada pelo programador no início da jornada de criação e torna mais acessível o primeiro projeto de equipes novatas.

1.1.2. GDScript

O GDScript é a linguagem de programação padrão da *engine* Godot e interage diretamente com a arquitetura de nós e cenas criada pelo usuário. Sua base é fortemente inspirada na linguagem *Python*, da qual herda características como ser interpretada e ter tipagem dinâmica, com foco na clareza e em uma baixa curva de aprendizado para iniciantes (SILVA; YEPES, 2018).

Diferentemente de sistemas que utilizam somente código ou somente recursos visuais, o GDScript combina os dois caminhos: o código executa as ações dos nós e de seus filhos por meio de referências no próprio *script* (como *$NomeDoNó*), além de recursos como *@export*, que expõe variáveis no editor (painel *Inspector*), e *@onready*, que inicia processos assim que a cena é carregada. Essa integração confere flexibilidade ao código sem torná-lo repetitivo.

1.1.3. Código aberto (*open source*)

O código aberto, ou *open source*, significa que o código-fonte do motor de jogo está disponível para que qualquer pessoa o consulte, estude e modifique em projetos próprios. A obra de Raymond (1999) é uma das fundadoras desse conceito: o autor divide o desenvolvimento em dois modelos simbólicos, a “catedral”, centralizada em uma empresa, e o “bazar”, um ambiente aberto e colaborativo em que todos podem contribuir simultaneamente, o que favorece o surgimento de ideias e soluções mais criativas e rápidas.

A Godot é um exemplo desse tipo de plataforma, o que amplia sua acessibilidade e favorece a formação de uma comunidade próxima. Com diversas pessoas trabalhando ao mesmo tempo, a comunidade produz melhorias e versões próprias do software, sugere atualizações com mais frequência e elabora tutoriais e materiais de aprendizagem, o que garante maior versatilidade e engajamento dos usuários.

Este artigo é publicado em acesso aberto (Open Access) sob a licença Creative Commons Atribuição 4.0 Internacional (CC-BY), que permite uso, distribuição e reprodução irrestritos em qualquer meio, desde que o autor original e a fonte sejam creditados.

1.1.4. Desenvolvimento 2D

O desenvolvimento 2D é muito utilizado em jogos por seu estilo de arte e de animação e emprega os *sprites*, imagens planas que podem representar desde personagens até o cenário. Diferencia-se dos modelos tridimensionais, que possuem profundidade e exigem um estilo próprio de modelagem de personagens e de elementos de cenário.

Sua principal vantagem é utilizar apenas altura e largura como planos de medida, o que reduz consideravelmente a dificuldade técnica de criação de novos modelos e o consumo computacional. Silva (2024) destaca que, para equipes pequenas e desenvolvedores solo, o 2D é uma escolha recorrente, por superar limitações técnicas, acelerar o desenvolvimento e garantir identidade visual. Por demandar menos etapas de produção do que a modelagem 3D, é apontado como porta de entrada para novos desenvolvedores.

1.1.5. RPG (*Role-Playing Game*)

O RPG (*Role-Playing Game*) é um gênero de jogos eletrônicos centrado em um personagem controlado pelo jogador e em sua evolução ao longo do jogo. Essa evolução ocorre pela descoberta de novas habilidades, pela obtenção de itens de tipos variados, que alteram o estilo de jogo, e pelo progresso narrativo, que envolve roteiro, personagens e uma história capaz de motivar o jogador a continuar explorando, cumprindo missões e conhecendo o enredo do mundo.

Segundo Saldanha e Batista (2009), o gênero permite que o jogador saia do sistema linear de narrativa e construa a própria história ao longo da aventura, por meio de escolhas e motivações que orientam seu progresso. No projeto deste trabalho, esse tipo de sistema aparece nos recursos e nas cartas, que permitem escolhas e estratégias definidas pelo jogador.

1.1.6. *Roguelike*

O gênero *roguelike* teve origem no jogo *Rogue* (1980) e é marcado pela rejogabilidade, construída a partir das repetidas tentativas de que o jogador precisa para aprender e progredir. Em 2008, durante a *International Roguelike Development Conference*, foram estabelecidas regras para definir esse gênero, conhecidas como *Berlin Interpretation* (INTERNATIONAL ROGUELIKE DEVELOPMENT CONFERENCE, 2008).

Entre os elementos que definem o conceito estão a geração procedural de conteúdo e a morte permanente do personagem. Na prática, quando o jogador é derrotado, perde todo o progresso daquela tentativa, o que faz parte de seu processo de aprendizado: ele percebe padrões e meios de tentar novamente com êxito. A sensação de novidade decorre da geração

Este artigo é publicado em acesso aberto (Open Access) sob a licença Creative Commons Atribuição 4.0 Internacional (CC-BY), que permite uso, distribuição e reprodução irrestritos em qualquer meio, desde que o autor original e a fonte sejam creditados.

aleatória de caminhos, adversários e itens, que aumenta o valor de *replay* sem a necessidade de acrescentar novos conteúdos.

1.2. Fundamentos de *game design*

Antes de iniciar a produção de um jogo, é preciso garantir que ele seja coeso e bem planejado. O planejamento inicial deve considerar, principalmente, a experiência proporcionada ao jogador, definindo como os elementos do projeto se relacionam com a jogabilidade (SCHELL, 2014). O autor defende que as mecânicas estejam em harmonia e equilibradas para despertar maior interesse, independentemente da dimensão do título produzido.

De acordo com Schell (2014), é importante haver um “ciclo básico” de jogabilidade, isto é, um conjunto de ações que o jogador repete constantemente. Um *design* bem estruturado torna esse ciclo satisfatório, desafiador e divertido, pois o principal objetivo é manter o interesse do jogador, formando um aprendizado constante que recompense a dedicação ao longo de muitas horas e evite a monotonia.

Para o autor, a prototipagem rápida é uma etapa fundamental do projeto, pois permite demonstrar, de forma funcional, a ideia inicial do criador. Testar as mecânicas nos estágios iniciais ajuda o desenvolvedor a economizar tempo e recursos e assegura que as estruturas básicas estejam bem estabelecidas antes da etapa de polimento, evitando partir para algo muito avançado sem uma base sólida.

A construção de uma experiência imersiva depende da harmonia entre mecânicas, narrativa, estética e interação, o que garante maior coerência ao projeto (SCHELL, 2014). Assim, os elementos do jogo devem se complementar, mantendo uma narrativa coerente e ações do jogador que façam sentido, alinhadas à parte artística, de modo a gerar equilíbrio e uma experiência única e imersiva.

1.3. O cenário dos jogos independentes (*indies*)

Um jogo não precisa ser desenvolvido exclusivamente por grandes estúdios internacionais, e dessa constatação surgiu o movimento dos jogos independentes. Esse cenário é caracterizado pela liberdade criativa e pela autonomia dos desenvolvedores em todas as etapas da produção (LEMES, 2009): o próprio desenvolvedor conduz todo o processo, desde a ideia inicial até a distribuição final. Essa autonomia é ampliada por ferramentas gratuitas, que equalizam as oportunidades no mundo do desenvolvimento de jogos.

Antes, ter acesso a tecnologias e participar do mercado de jogos era algo incomum. Com o avanço da informação e da internet, contudo, o desenvolvedor *indie* passou a ter o mesmo

Este artigo é publicado em acesso aberto (Open Access) sob a licença Creative Commons Atribuição 4.0 Internacional (CC-BY), que permite uso, distribuição e reprodução irrestritos em qualquer meio, desde que o autor original e a fonte sejam creditados.

alcance global que uma grande corporação. Assim, o destaque de um título deixou de depender de um orçamento colossal e passou a depender de sua criatividade e originalidade, com mecânicas, narrativa e características que tornam esses jogos únicos.

Outro fator que favorece a consolidação de pequenos projetos são as lojas digitais, como a *Steam*, essenciais para expor e acessar diversos tipos de jogos e que eliminam a necessidade de uma publicadora tradicional ou de lojas físicas. Esse modelo permite que os jogadores encontrem de forma mais direta títulos dos gêneros de seu interesse, conheçam novas experiências e formem comunidades fortes, o que dá reconhecimento aos desenvolvedores e os estabiliza como empresas menores.

Além da distribuição, o mercado *indie* se destaca por sua ligação com a parte criativa dos desenvolvedores, enquanto empresas de maior porte se limitam a fórmulas e a análises de mercado para garantir retorno financeiro. Por não estarem presos a essas limitações, os desenvolvedores independentes têm maior capacidade de influenciar a indústria por meio de novas mecânicas e novos gêneros, que muitas vezes são absorvidos pelo mercado e ampliam seu reconhecimento.

1.4. A arquitetura da Godot Engine

A Godot se destaca de outros motores da área por sua organização estrutural. Conforme a Godot Engine (2024), a *engine* funciona por meio de uma hierarquia de *Nodes* (nós) que, agrupados, formam as cenas. Essa organização permite dividir o trabalho em módulos reutilizáveis em várias partes do projeto, o que torna a gestão do jogo mais acessível, principalmente para o programador independente.

A linguagem GDScript é um dos principais diferenciais da plataforma, sobretudo por possuir sintaxe inspirada em Python e integração direta com a *engine* (SILVA; YEPES, 2018), o que facilita a adoção por novos programadores. Essa integração permite ao desenvolvedor trabalhar com o código junto à interface geral da plataforma.

A Godot tem atraído também estúdios consolidados. Um caso recente foi o lançamento, em março de 2026, do acesso antecipado de *Slay the Spire 2*, sequência do *roguelike* desenvolvido pela Mega Crit, que optou por migrar da *engine* utilizada no jogo anterior para a Godot. O caso mostra que a plataforma suporta títulos de alta complexidade e sucesso comercial (GODOT ENGINE, 2026).

Este artigo é publicado em acesso aberto (Open Access) sob a licença Creative Commons Atribuição 4.0 Internacional (CC-BY), que permite uso, distribuição e reprodução irrestritos em qualquer meio, desde que o autor original e a fonte sejam creditados.

1.5. Usabilidade e democratização do desenvolvimento

Para um iniciante, é comum relacionar os jogos a algo realista, com gráficos avançados e jogabilidade complexa, o que pode causar frustração ou desencorajá-lo. Silva e Yepes (2018) abordam as ferramentas de alta usabilidade e produtividade, como a Godot, com as quais, mesmo com menor consumo de *hardware* e equipes menores, é possível entregar um produto de qualidade, o que amplia a democratização do uso da tecnologia para fins criativos.

O potencial de um uso mais democratizado é ilustrado por projetos como *Cassette Beasts*, um RPG de mundo aberto desenvolvido por uma equipe de apenas duas pessoas. Em entrevista ao *Godot Showcase*, os criadores afirmam que a Godot permitiu que uma equipe reduzida entregasse um jogo completo, o que indica que a plataforma pode reduzir os obstáculos da produção e auxiliar o desenvolvedor *indie* (GODOT ENGINE, 2023).

Portanto, a democratização proporcionada pelas *game engines* não apenas permite a criação de títulos bem-sucedidos, mas também auxilia a inovação acadêmica. Ao diminuir as dificuldades do caminho, o desenvolvedor pode concentrar-se mais na parte criativa do projeto, o que tende a gerar um produto mais viável e de maior qualidade. Dessa forma, o uso de tecnologias de código aberto e de alta usabilidade é importante para esse tipo de projeto, pois evita que parte dele seja sacrificada por barreiras tecnológicas ou financeiras.

2. METODOLOGIA

2.1. Caracterização da pesquisa

A pesquisa teve caráter exploratório e qualitativo, com foco na usabilidade e nos processos de criação. Adotou-se uma abordagem experimental, tendo a versão 4.3 da Godot Engine como estudo de caso. Como procedimento técnico, realizou-se pesquisa bibliográfica em obras de *game design*, em artigos científicos e na documentação oficial da plataforma, a fim de identificar a arquitetura do motor, sua hierarquia de *Nodes* e a implementação de mecânicas por meio do GDScript. O universo do estudo compreendeu os recursos tecnológicos do software e as artes digitais (*sprites*) aplicadas. Os dados, coletados por meio do registro do código-fonte e de testes funcionais, foram analisados de forma descritiva, de modo a discutir em que medida o uso da Godot se mostrou favorável ao programador.

A avaliação baseou-se na experiência do próprio desenvolvedor, sem testes sistemáticos com usuários e sem indicadores quantitativos de desempenho, limitação retomada na seção de resultados e discussão. Os procedimentos de desenvolvimento do protótipo são descritos nas subseções seguintes.

Este artigo é publicado em acesso aberto (Open Access) sob a licença Creative Commons Atribuição 4.0 Internacional (CC-BY), que permite uso, distribuição e reprodução irrestritos em qualquer meio, desde que o autor original e a fonte sejam creditados.

2.2. Planejamento inicial do projeto

O planejamento inicial definiu as principais características do jogo antes da implementação das mecânicas: um jogo 2D de combate por turnos, com cartas e aleatoriedade por meio de rolagens de dados, modelo escolhido para apresentar as escolhas do jogador de maneira prática e dinâmica. Foram utilizados conceitos de *game design* apresentados por Schell (2014), principalmente os relacionados ao planejamento da experiência do jogador e à construção de um ciclo básico de *gameplay*, segundo os quais jogos bem estruturados dependem de mecânicas que mantenham o jogador engajado por meio de desafios, recompensas e tomadas de decisão (SCHELL, 2014). Com base nesse autor, o projeto foi planejado para criar um fluxo contínuo de ações, mantendo o jogador sempre decidindo qual a melhor jogada.

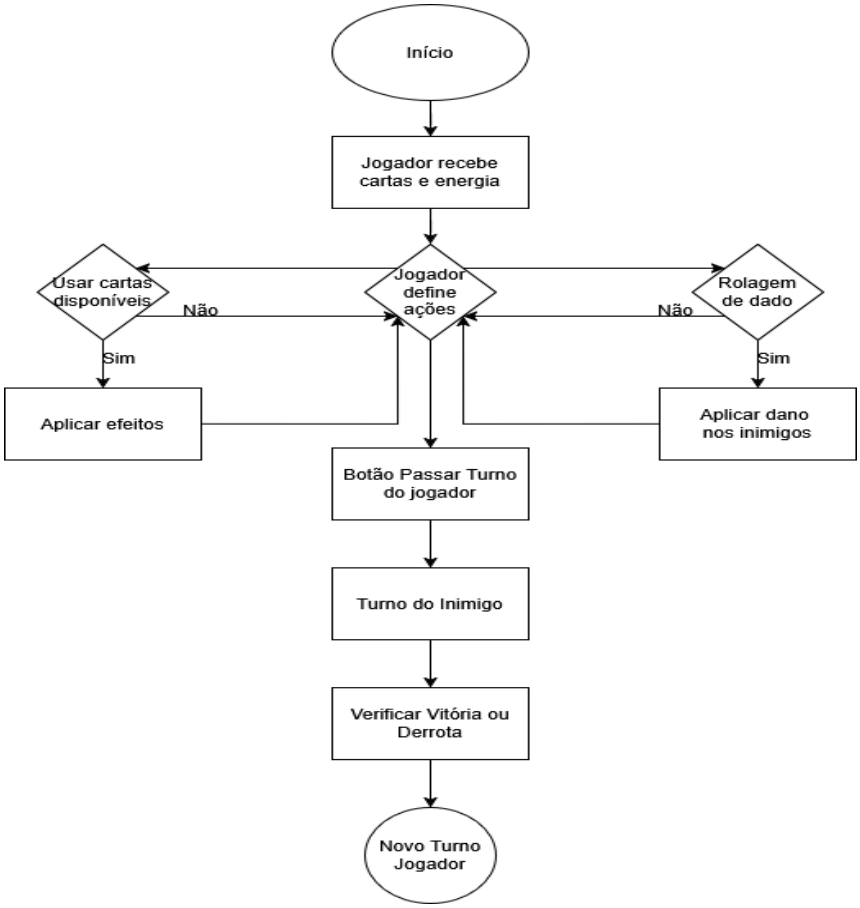
2.2.1. Fluxograma da jogabilidade

No fluxo básico da jogabilidade (Figura 1), o jogador inicia o combate e, no seu turno, recebe as cartas devidas e a energia (*mana*) para gastá-las. Em seguida, define sua ação: “Usar cartas disponíveis”, analisando as cartas em mãos conforme a vida própria e a do inimigo, o custo de energia e o efeito de cada carta; ou “Rolagem de dado”, que garante um dano mínimo ao inimigo por turno, sendo esse dano um valor aleatório que depende do tipo de dado utilizado (de 4 a 8 lados).

Depois de realizar suas escolhas, o jogador clica em “Passar Turno”, e o jogo avança para o turno do inimigo, que, controlado por alguns comandos programados, causa dano ao jogador ou aplica um efeito em si próprio. Somente depois disso o jogo verifica se o jogador perdeu toda a sua vida ou se derrotou todos os inimigos. Se nenhuma dessas condições ocorrer, um novo turno começa, até que se chegue a uma conclusão.

Este artigo é publicado em acesso aberto (Open Access) sob a licença Creative Commons Atribuição 4.0 Internacional (CC-BY), que permite uso, distribuição e reprodução irrestritos em qualquer meio, desde que o autor original e a fonte sejam creditados.

Figura 1. Fluxograma da jogabilidade



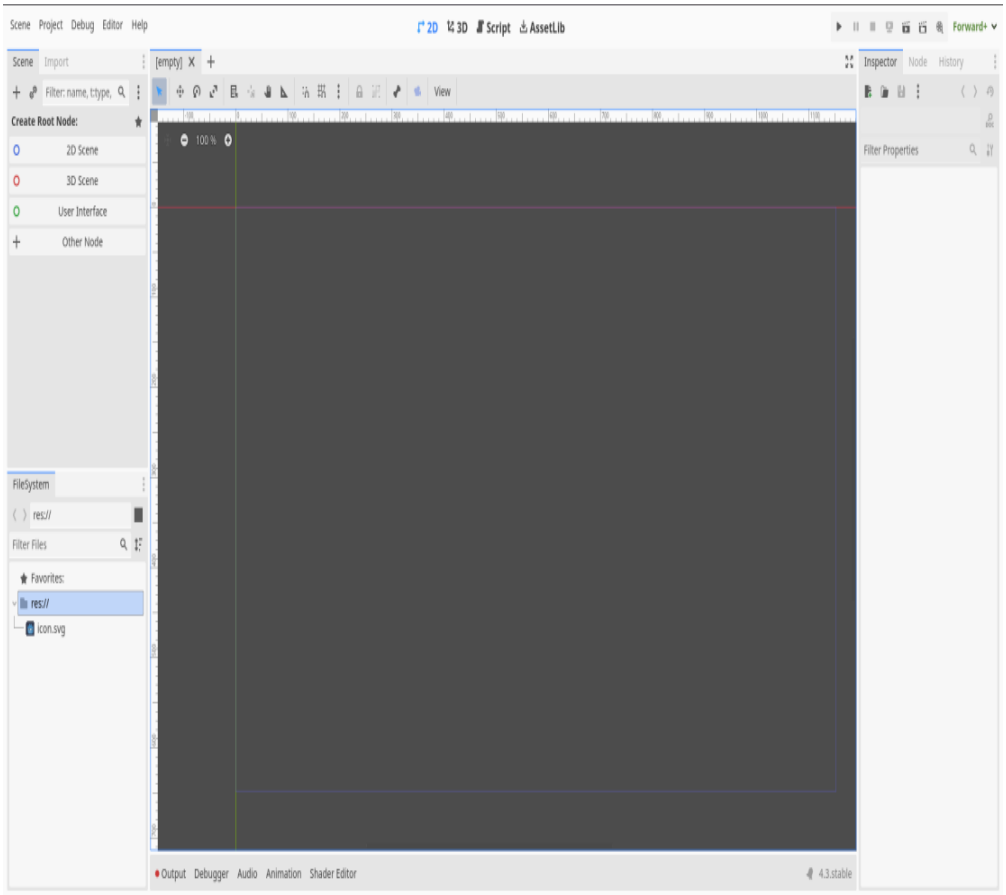
Fonte: Autoria própria.

2.3. Preparação do ambiente de desenvolvimento

Definidos o conceito e o fluxo de jogabilidade, preparou-se o ambiente de desenvolvimento. A plataforma selecionada foi a *game engine* Godot, versão 4.3, cujas ferramentas da própria interface facilitam a criação de novos projetos com poucos cliques. Inicialmente, abriu-se o gerenciador de projetos (*Project Manager*), no qual foi criado um projeto dedicado, e a interface gráfica padrão foi configurada diretamente para o modo de edição 2D (Figura 2).

Este artigo é publicado em acesso aberto (Open Access) sob a licença Creative Commons Atribuição 4.0 Internacional (CC-BY), que permite uso, distribuição e reprodução irrestritos em qualquer meio, desde que o autor original e a fonte sejam creditados.

Figura 2. Interface inicial da Godot



Fonte: Autoria própria.

2.4. Importação de *assets* e gerenciamento de *sprites*

Para este projeto foram elaboradas algumas artes conceituais, que definiram o estilo do jogo e serviram de base para que o jogador compreendesse cada elemento da tela. Como mostra a Figura 3, o projeto foi organizado em pastas dentro do sistema de arquivos da plataforma (caminho “res://”) e dividido em quatro diretórios principais: cenas, *resource*, *scripts* e *sprites*; este último foi subdividido em animação, *background*, cartas, dados, efeitos, fonte de texto e ícones. Essa distribuição evidencia a praticidade de dividir o projeto em módulos: código e *assets* visuais permanecem em locais determinados pelo desenvolvedor, sem afetar um ao outro, e podem ser utilizados em conjunto em cenas mais elaboradas, além de facilitar a manutenção à medida que o projeto cresce.

Este artigo é publicado em acesso aberto (Open Access) sob a licença Creative Commons Atribuição 4.0 Internacional (CC-BY), que permite uso, distribuição e reprodução irrestritos em qualquer meio, desde que o autor original e a fonte sejam creditados.

Figura 3. Estrutura de organização das pastas



Fonte: Autoria própria.

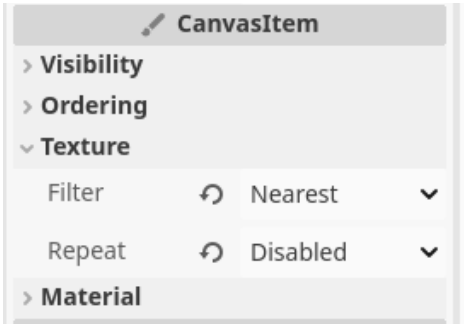
2.4.1. Configuração de elementos gráficos

Os componentes gráficos utilizados na interface gráfica do usuário (*UI*) e nas mecânicas de jogabilidade, como as artes das cartas, os dados e os indicadores de *status*, foram tratados individualmente como imagens estáticas no formato PNG. Esses arquivos foram importados para o projeto, sendo necessário configurá-los manualmente.

Por padrão, a Godot aplica filtros que suavizam os *pixels*, o que resulta em imagem borrada quando se trabalha com baixas resoluções. Para evitar esse problema e garantir a nitidez das imagens em *pixel art*, as texturas foram alteradas para o modo de filtragem *Nearest* (*Pixel Art*). Conforme a Figura 4, essa alteração foi feita no painel da Godot, no campo *Filter*, localizado em *CanvasItem*, no campo *Texture*. Além disso, a opção *Repeat* foi desabilitada, para evitar que a textura se repita nas bordas do *sprite* quando sua dimensão for alterada ou ele for esticado na cena em que é utilizado.

Este artigo é publicado em acesso aberto (Open Access) sob a licença Creative Commons Atribuição 4.0 Internacional (CC-BY), que permite uso, distribuição e reprodução irrestritos em qualquer meio, desde que o autor original e a fonte sejam creditados.

Figura 4. Configuração do filtro de textura *Nearest*



Fonte: Autoria própria.

2.4.2. Folhas de *sprites* (*Sprite Sheets*) e configuração de animações

Para as animações do jogador, do oponente e do efeito giratório dos dados, foi utilizada a técnica de *Sprite Sheets*, que agrupa vários quadros (*frames*) alinhados em sequência em uma única folha de imagem contínua. A execução das sequências ocorre por meio do recurso *SpriteFrames*, fornecido pelo nó *AnimatedSprite2D*. No editor de animação, as folhas foram organizadas conforme o tamanho em *pixels* de cada *frame*; a partir desse recorte, foram definidos os blocos de animação dos estados de repouso (*idle*) e da rotação das faces dos dados, e configurou-se a taxa de quadros por segundo (FPS) para garantir fluidez visual durante o combate (Figura 5).

Figura 5. Editor de *SpriteFrames* exibindo quadros recortados de uma animação de rolagem de dados



Fonte: Autoria própria.

2.5. Estruturação da arquitetura do jogo: cenas, *Nodes* e código

A arquitetura foi estruturada conforme a regra de modularidade da plataforma: os nós (*Nodes*), quando agrupados de modo coerente com sua funcionalidade, compõem as cenas (GODOT ENGINE, 2024). A árvore de elementos permite separar as partes visuais e de programação de cada componente, o que garante um gerenciamento de código limpo e

Este artigo é publicado em acesso aberto (Open Access) sob a licença Creative Commons Atribuição 4.0 Internacional (CC-BY), que permite uso, distribuição e reprodução irrestritos em qualquer meio, desde que o autor original e a fonte sejam creditados.

potencialmente reutilizável. Segundo a experiência do desenvolvedor, essa separação simplificou a manutenção dos componentes, o que é coerente com as premissas de alta produtividade referidas por Silva e Yepes (2018) para equipes de desenvolvedores independentes.

2.5.1. Implementação dos *Nodes* do jogador e do inimigo

O jogador e o inimigo foram construídos como cenas independentes entre si, ambas com um nó raiz do tipo *Node2D*, nomeadas *Jogador* e *Inimigo*, o que torna a manutenção mais prática e facilita a reutilização de funções específicas. Abaixo da raiz de cada cena, inseriu-se um nó *AnimatedSprite2D*, responsável por reproduzir as folhas de *sprites* importadas anteriormente, e, para exibir os pontos de vida (*HP*), foram acoplados nós de interface da classe *ProgressBar* e outros nós para os demais elementos de *gameplay* (Figura 6).

Figura 6. Cena com *sprite* e barra de vida do jogador



Fonte: Autoria própria.

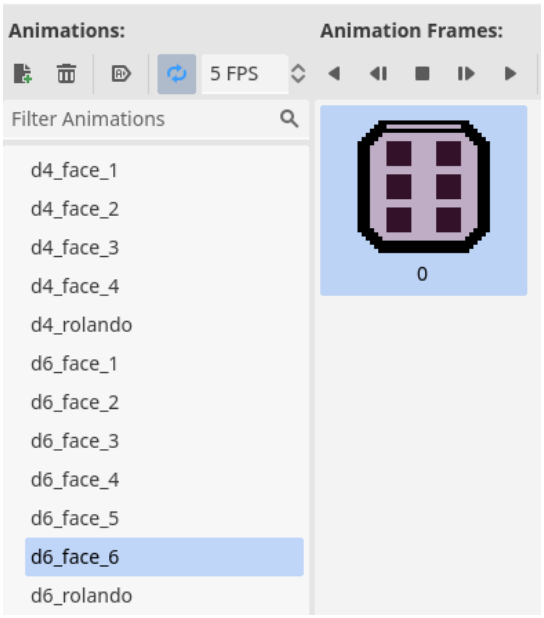
2.5.2. Implementação do *Node* de dados e do sistema de rolagem

A mecânica de rolar dados foi dividida em duas partes: a criação de um único dado e a do sistema completo que gerencia a jogada, o que ajuda a organizar o projeto e permite combinar peças simples para criar uma mecânica mais complexa. A primeira cena, *Dado*, trata apenas do comportamento de um dado individual, com raiz *Node2D* e um nó *AnimatedSprite2D* que armazena a animação do dado girando e as imagens fixas de cada face, conforme o tipo de dado (d4, d6, d8). Quando o código sorteia um número, o motor gráfico altera o visual do dado para mostrar o valor correto. A Figura 7 mostra o editor de *SpriteFrames* com as animações e os

Este artigo é publicado em acesso aberto (Open Access) sob a licença Creative Commons Atribuição 4.0 Internacional (CC-BY), que permite uso, distribuição e reprodução irrestritos em qualquer meio, desde que o autor original e a fonte sejam creditados.

sprites dos dados, organizados por tipo (*d4_face_1*, *d6_face_1*, *d6_face_6*), que definem a face resultante, e as animações de rolagem de cada tipo (*d4_rolando*, *d6_rolando*), que criam expectativa antes da exibição do valor final.

Figura 7. Lista de faces e animações dos dados

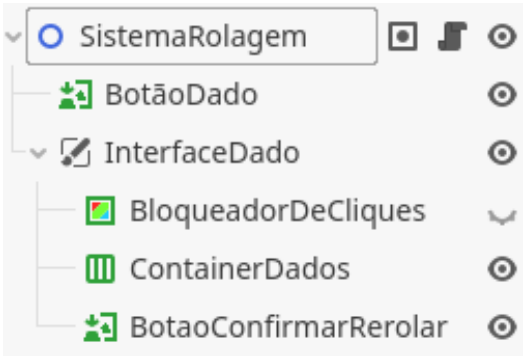


Fonte: Autoria própria.

A segunda cena, *RolagemDados*, também com raiz *Node2D*, faz o sistema de sorteio funcionar nas batalhas e reaproveita a cena do dado para qualquer tipo de dado. Nela foram adicionados nós de interface: um *Button*, que inicia o giro e o sorteio, e um *Container*, que organiza os valores da rolagem conforme a quantidade e o tipo de dado utilizado. A Figura 8 mostra a árvore completa desse sistema. O *BotãoDado* serve como acionador das ações da cena; entre os elementos de interface, o *BloqueadorDeCliques* impede que o usuário acione qualquer outro componente da tela, o *ContainerDados* organiza a exibição de um ou mais dados no monitor e o *BotaoConfirmarRerolar* permite que o jogador, depois de selecionar os dados no contêiner, confirme que deseja rolar novamente os dados selecionados.

Este artigo é publicado em acesso aberto (Open Access) sob a licença Creative Commons Atribuição 4.0 Internacional (CC-BY), que permite uso, distribuição e reprodução irrestritos em qualquer meio, desde que o autor original e a fonte sejam creditados.

Figura 8. Árvore de nós do sistema de rolagem



Fonte: Autoria própria.

2.5.3. Implementação do *Node* de carta

A criação das cartas também foi modular, separando a parte visual de uma carta individual do sistema que gerencia a mão do jogador. A primeira parte é a cena *CartaBase* (Figura 9), montada com nós de interface voltados ao posicionamento dos elementos visuais: nós *TextureRect* carregam as imagens do fundo e da arte que ilustra o poder, e nós *Label* exibem dinamicamente o custo de energia, o nome da habilidade e a descrição do efeito.

Figura 9. Carta base com nome, custo e descrição



Fonte: Autoria própria.

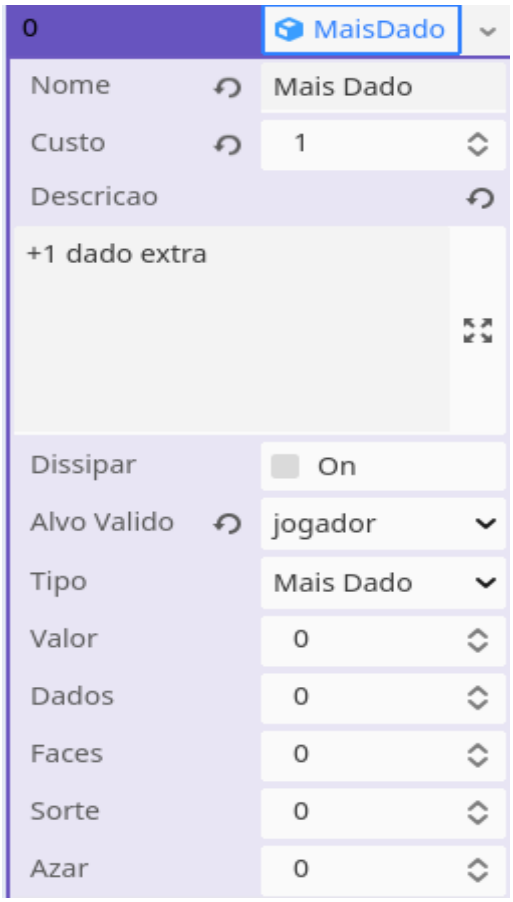
A segunda parte é a cena *MaoJogador*, o espaço da tela em que as cartas sorteadas ficam alinhadas para a escolha do jogador, que trabalha em conjunto com o gerenciador de baralho. O diferencial desse gerenciador é uma propriedade customizada, criada por código,

Este artigo é publicado em acesso aberto (Open Access) sob a licença Creative Commons Atribuição 4.0 Internacional (CC-BY), que permite uso, distribuição e reprodução irrestritos em qualquer meio, desde que o autor original e a fonte sejam creditados.

chamada “Baralho Inicial” e visível no painel *Inspector* da Godot: para adicionar novas cartas, basta criar o arquivo da carta e arrastá-lo para essa lista, sem alterar as linhas de programação principais do sistema de combate.

A Figura 10 exibe a primeira carta de forma expandida, identificada pelo nome *MaisDado.tres*, tipo de arquivo que organiza cada carta com a mesma estrutura da classe *Resource*, com campos editáveis no próprio *Inspector*. Entre eles estão *Nome*, *Custo* e *Descrição*, mostrados na imagem da carta, além de efeitos criados pelo desenvolvedor, como *Dissipar*, *Tipo*, *Sorte* e *Azar*. Essa abordagem permite reaproveitar a mesma estrutura de código de maneiras distintas e, caso seja necessário alterar algo, identificar com precisão onde está o problema e como resolvê-lo.

Figura 10. Exemplo de criação da carta



Fonte: Autoria própria.

Este artigo é publicado em acesso aberto (Open Access) sob a licença Creative Commons Atribuição 4.0 Internacional (CC-BY), que permite uso, distribuição e reprodução irrestritos em qualquer meio, desde que o autor original e a fonte sejam creditados.

2.5.4. Implementação da lógica de dano ao inimigo

Para exemplificar o uso do GDScript, a Figura 11 destaca a função *aplicar_dano()*, do nó do Inimigo, que processa o dano recebido de qualquer fonte e altera o valor atual da vida do adversário após algumas validações. A função possui dois parâmetros: *valor*, do tipo inteiro, que corresponde ao dano a ser aplicado, e *tipo*, uma *String* de valor padrão “NORMAL”, usada para determinar se o dano é um crítico ou uma falha. A vida é calculada com a função nativa *max()*, que mantém o mínimo em zero, e a interface é atualizada com *atualizar_vida()*.

Figura 11. Função *aplicar_dano*

```
func aplicar_dano(valor: int, tipo: String = "NORMAL"):
    vida = max(vida - valor, 0)
    atualizar_vida()

    status_alvo = "NONE"

    TextoDano.texto_flutuante(global_position, valor, tipo)

    if valor > 0:
        sprite_inimigo.modulate = Color.CRIMSON
        await get_tree().create_timer(0.1).timeout
        sprite_inimigo.modulate = Color.WHITE

    if vida <= 0:
        hide()
```

Fonte: Autoria própria.

Em seguida, a função *TextoDano.texto_flutuante()* faz aparecer um número sobre o *sprite* do inimigo quando ele recebe algum tipo de dano. Dentro do ponto de decisão (*if valor > 0:*), a imagem do inimigo pisca em vermelho para indicar que o dano foi recebido. Por fim, uma última validação verifica se a vida do inimigo é menor ou igual a zero; se for verdadeira, a função *hide()* é chamada para esconder a figura do inimigo, indicando que ele foi derrotado.

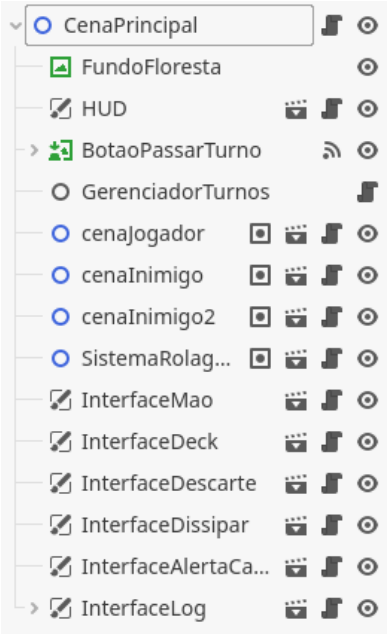
2.5.5. Configuração da cena principal (*Main Scene*)

A cena principal funciona como núcleo que centraliza as aplicações e como nó raiz responsável por conectar e coordenar todos os subsistemas do jogo. Estruturada a partir de um nó *Node2D* nomeado *CenaPrincipal*, ela gerencia o carregamento do cenário de combate e serve de base para a execução do ciclo de jogo mapeado no planejamento. Nessa hierarquia são instanciados os nós dos agentes (*Jogador* e *Inimigo*) e a representação visual de fundo (*Background*), e a interface gráfica e o gerenciador de turnos são estruturados em conjunto, sob

Este artigo é publicado em acesso aberto (Open Access) sob a licença Creative Commons Atribuição 4.0 Internacional (CC-BY), que permite uso, distribuição e reprodução irrestritos em qualquer meio, desde que o autor original e a fonte sejam creditados.

os nós de controle (*CanvasLayer* e *Control*), criando o ambiente integrado em que as mecânicas interagem (Figura 12).

Figura 12. Árvore de nós da cena principal



Fonte: Autoria própria.

3. RESULTADOS E DISCUSSÃO

3.1. Produto obtido

O produto final do desenvolvimento é um protótipo de jogo 2D de combate por turnos, com elementos de RPG e de *roguelike*; o uso da aleatoriedade, das cartas e da rolagem de dados é coerente com esses gêneros. O jogador inicia o combate recebendo cartas e energia (*mana*) e usa os valores obtidos nas rolagens de dados como dano causado aos inimigos. Quando o jogador passa o turno, o inimigo realiza uma ação predeterminada; em seguida, o jogo verifica se houve vitória ou segue para o turno seguinte.

O Quadro 1 sintetiza as principais cenas implementadas, os nós que as compõem e sua função no jogo, conforme descrito na metodologia.

Este artigo é publicado em acesso aberto (Open Access) sob a licença Creative Commons Atribuição 4.0 Internacional (CC-BY), que permite uso, distribuição e reprodução irrestritos em qualquer meio, desde que o autor original e a fonte sejam creditados.

Quadro 1. Principais cenas do protótipo, seus nós e suas funções

Cena	Nós principais	Função no jogo
Jogador	*Node2D*; *AnimatedSprite2D*; *ProgressBar*	Personagem controlado pelo usuário, com animações e barra de vida (*HP*)
Inimigo	*Node2D*; *AnimatedSprite2D*; *ProgressBar*	Oponente com animações, barra de vida e função *aplicar_dano()*
Dado	*Node2D*; *AnimatedSprite2D*	Comportamento de um dado individual (animação de giro e faces dos tipos d4, d6 e d8)
RolagemDados	*Node2D*; *Button*; *Container*	Sistema de rolagem; reaproveita a cena *Dado* para qualquer tipo de dado
CartaBase	*TextureRect*; *Label*	Parte visual de uma carta (fundo, arte, custo, nome e descrição)
MaoJogador	Gerenciador de baralho (propriedade “Baralho Inicial”)	Espaço em que as cartas sorteadas ficam alinhadas para a escolha do jogador
CenaPrincipal	*Node2D*; *CanvasLayer*; *Control*	Instancia *Jogador*, *Inimigo*, *Background*, interface e gerenciador de turnos

Fonte: Autoria própria.

A Figura 13 mostra a tela de combate do jogo em execução, no início da partida. No centro, o jogador, um mago de capuz, enfrenta dois inimigos em formato de castor, em um cenário de floresta em *pixel art*. Cada personagem possui barra de vida (30/30), e acima de cada inimigo aparece um ícone com a ação que ele pretende realizar no próximo turno, no caso, dano de +1. No topo, o contador indica “Turno: 1” e, no canto superior direito, o registro (*log*) informa o início da partida e do primeiro turno.

Figura 13. Protótipo do jogo



Fonte: Autoria própria.

Este artigo é publicado em acesso aberto (Open Access) sob a licença Creative Commons Atribuição 4.0 Internacional (CC-BY), que permite uso, distribuição e reprodução irrestritos em qualquer meio, desde que o autor original e a fonte sejam creditados.

Do lado esquerdo estão os elementos da mecânica de dados: o botão “Rolar Dado”, o tipo de dado, indicado no círculo superior (ícone de um d6), e os indicadores de bônus (+0 no início). Do lado direito ficam a energia do jogador (3/3), o botão “Passar Turno” e as pilhas de compra e de descarte, com a quantidade de cartas em cada uma. Na parte inferior está a mão do jogador, com as cartas “Cara ou Coroa”, “Força” e “Fúria”, que podem ser usadas conforme a energia disponível.

3.2. Discussão

Na experiência do desenvolvedor, as cenas e os nós foram importantes para a criação do projeto, pois permitiram reaproveitar componentes, como a cena do dado e as cartas, e facilitaram a manutenção, enquanto o GDScript permitiu criar a lógica de combate de forma direta. A integração desses elementos resultou em um protótipo com algumas cartas e inimigos de comportamento simples, porém funcionais. Esse resultado é coerente com o que a literatura consultada afirma sobre a organização modular da plataforma e sobre a facilidade de aprendizado do GDScript (GODOT ENGINE, 2024; SILVA; YEPES, 2018).

Cabe ressaltar, porém, os limites do estudo. A avaliação de usabilidade, produtividade e viabilidade baseou-se na experiência do próprio desenvolvedor, em um único projeto de escopo reduzido, sem testes sistemáticos, sem indicadores quantitativos (como tempo de desenvolvimento, desempenho de execução ou consumo de memória) e sem a participação de usuários iniciantes. Por isso, os resultados descrevem o que foi possível realizar com a plataforma neste caso e não permitem afirmar, de modo geral, que ela reduz a barreira de entrada para iniciantes, nem comparar seu desempenho com o de outras *engines*. A hipótese da pesquisa foi, portanto, explorada apenas de forma indicativa.

4. CONSIDERAÇÕES FINAIS

Este trabalho teve como objetivo desenvolver um jogo 2D de combate por turnos na Godot Engine 4.3, para discutir a plataforma como ambiente amigável e produtivo para desenvolvedores independentes e iniciantes. A entrega de um protótipo funcional, construído com a arquitetura baseada em cenas e nós e com a lógica programada em GDScript, mostra que, nesse caso, foi possível implementar as mecânicas de combate por turnos, cartas e rolagem de dados em uma estrutura modular e de fácil manutenção.

Na experiência do desenvolvedor, a Godot Engine mostrou-se uma ferramenta viável e acessível para a criação de um protótipo 2D, desenvolvido por uma única pessoa e sem

Este artigo é publicado em acesso aberto (Open Access) sob a licença Creative Commons Atribuição 4.0 Internacional (CC-BY), que permite uso, distribuição e reprodução irrestritos em qualquer meio, desde que o autor original e a fonte sejam creditados.

equipamento de alto desempenho. O fato de ser de código aberto e de adotar uma linguagem considerada de fácil aprendizado, segundo a literatura consultada, indica potencial para favorecer a entrada de iniciantes na criação de jogos. No entanto, como a avaliação se baseou em um único caso e na percepção do próprio desenvolvedor, sem testes sistemáticos nem indicadores quantitativos, essa conclusão não pode ser generalizada.

Como trabalhos futuros, sugere-se ampliar o conteúdo do jogo, medir indicadores objetivos (como tempo de desenvolvimento, desempenho de execução e dificuldades encontradas), comparar a Godot com outras *engines* e realizar testes com um grupo de iniciantes.

REFERÊNCIAS

GODOT ENGINE. **Godot Showcase - Cassette Beasts**. 2023. Disponível em: https://godotengine.org/article/godot-showcase-cassette-beasts/. Acesso em: 11 abr. 2026.

GODOT ENGINE. **Godot Engine documentation**. 2024. Disponível em: https://docs.godotengine.org/en/4.3/. Acesso em: 4 abr. 2026.

GODOT ENGINE. **Slay the Spire 2 - Showcase**. 2026. Disponível em: https://godotengine.org/showcase/slay-the-spire-2/. Acesso em: 11 abr. 2026.

GREGORY, Jason. **Game Engine Architecture**. 3. ed. Boca Raton: CRC Press, 2018.

INTERNATIONAL ROGUELIKE DEVELOPMENT CONFERENCE. **The Berlin Interpretation**. Berlim, 2008. Disponível em: http://www.roguebasin.com/index.php?title=Berlin_Interpretation. Acesso em: 15 abr. 2026.

LEMES, David de Oliveira. **Games independentes**: fundamentos metodológicos para criação, planejamento e desenvolvimento de jogos digitais. 2009. 158 f. Dissertação (Mestrado em Tecnologias da Inteligência e Design Digital) – Pontifícia Universidade Católica de São Paulo, São Paulo, 2009.

MALARA, Matheus Cacheta. **Desenvolvimento de humanoide dirigido por inteligência artificial usando Unreal Engine 5**. 2024. 30 f. Trabalho de Conclusão de Curso (Bacharelado em Sistemas de Informação) – Universidade de Araraquara (UNIARA), Araraquara, 2024.

RAYMOND, Eric S. **The Cathedral and the Bazaar**. Sebastopol: O'Reilly, 1999.

SALDANHA, Ana Alayde; BATISTA, José Roniere Morais. A concepção do role-playing game (RPG) em jogadores sistemáticos. **Psicologia: Ciência e Profissão**, v. 29, n. 4, p. 700-717, 2009.

SCHELL, Jesse. **A Arte de Game Design**: o livro de lentes. 2. ed. Rio de Janeiro: Elsevier, 2014.

SILVA, Laura Francine Araújo. **Design de jogos indies**: desenvolvimento de personagens e narrativa em jogos de pixel art no The Rogue Angler. 2024. Trabalho de Conclusão de Curso (Graduação) – Universidade Federal da Paraíba (UFPB), João Pessoa, 2024. Disponível em: https://repositorio.ufpb.br/jspui/handle/123456789/34693. Acesso em: 13 abr. 2026.

Este artigo é publicado em acesso aberto (Open Access) sob a licença Creative Commons Atribuição 4.0 Internacional (CC-BY), que permite uso, distribuição e reprodução irrestritos em qualquer meio, desde que o autor original e a fonte sejam creditados.

SILVA, Vitor Poltronieri da; YEPES, Igor. Desenvolvimento de jogos na plataforma Godot. In: ENCONTRO ANUAL DE TECNOLOGIA DA INFORMAÇÃO (EATI), 8., 2018, Frederico Westphalen. **Anais** [...]. Frederico Westphalen: IFFar, 2018. p. 102-109.

Este artigo é publicado em acesso aberto (Open Access) sob a licença Creative Commons Atribuição 4.0 Internacional (CC-BY), que permite uso, distribuição e reprodução irrestritos em qualquer meio, desde que o autor original e a fonte sejam creditados.