REVISTA TÓPICOS
ISSN 2965-6672 | Qualis A2

# ARQUITETURA DE SISTEMAS EM JOGOS INDEPENDENTES UTILIZANDO A GODOT ENGINE

**SYSTEMS ARCHITECTURE IN INDIE GAMES USING THE GODOT ENGINE**

Engenharias • 26/11/2026

REGISTRO DOI: 10.70773/revistatopicos/795583127

Álvaro Ross de Paula[1]

Nilton Kazuo Gomes Suzuki[2]

**RESUMO**

Este trabalho analisa a implementação de uma estrutura lógica modular no desenvolvimento independente de jogos digitais utilizando a Godot Engine. Apresenta um estudo de caso focado no desenvolvimento do protótipo Ritebreaker, um jogo de combate contra hordas baseado na estética de horror cósmico. Adota uma abordagem experimental para validar o desacoplamento de sistemas por meio da comunicação via sinais e do design orientado a dados com o uso de recursos, integrando rotinas de otimização para o descarte dinâmico de instâncias. Demonstra a obtenção de um software funcional capaz de sustentar uma taxa estável de sessenta quadros por segundo sob alta densidade de elementos, desenvolvido em um ciclo iterativo de dez dias. Conclui que a arquitetura modular potencializa a autonomia do desenvolvedor ao permitir a construção de sistemas resilientes, organizados e de fácil expansão técnica.

**Palavras-chave:** desenvolvimento de jogos; modularidade; horror cósmico; desenvolvimento independente; godot engine.

**ABSTRACT**

This study analyzes the implementation of a modular logical structure in independent digital game development using the Godot Engine. It presents a case study focused on the development of the Ritebreaker prototype, a horde combat game featuring a cosmic horror aesthetic. An experimental approach was adopted to validate system decoupling through signal-based communication and data-driven design using resources. Additionally, the work integrates optimization routines for dynamic instance discarding. The results demonstrate a functional software capable of maintaining a stable sixty frames per second even under high screen density, developed within a ten-day iterative cycle. It concludes that

modular architecture is highly effective, enabling developers to build resilient, organized, and easily expandable systems.
**Keywords:** game development; modularity; cosmic horror; independent development; godot engine.

## 1. INTRODUÇÃO

O desenvolvimento de jogos digitais têm demonstrado um crescimento significativo nas últimas décadas, impulsionado pela expansão da indústria de entretenimento interativo e pela democratização das ferramentas de criação. Com o avanço das tecnologias digitais e a popularização de softwares acessíveis, o processo de desenvolvimento tornou-se viável para pequenos grupos e desenvolvedores independentes, descentralizando a produção que era restrita aos grandes estúdios corporativos. Diante deste cenário, os motores de jogos (game engines) assumem um papel fundamental ao fornecer a infraestrutura de software necessária para a implementação de sistemas complexos, mecânicas e interfaces.

Do ponto de vista conceitual, jogos digitais podem ser compreendidos como sistemas dinâmicos baseados em regras que estruturam a interação entre o usuário e o ambiente virtual. Segundo Adams e Dormans (2012), os jogos digitais constituem sistemas formais de regras interativas que operam por meio de mecânicas estruturadas, sendo a organização desses elementos determinante para a experiência final do jogador. De uma maneira semelhante, Salen e Zimmerman (2003) destacam que o design de jogos engloba a construção de sistemas significativos de interação, nos quais as ações feitas pelo jogador geram consequências perceptíveis dentro de um escopo preestabelecido. Desse modo, o

desenvolvimento de um software de entretenimento exige decisões arquiteturais rigorosas relacionadas à organização total do sistema de componentes lógicos, indo além da mera dimensão estética ou narrativa.

Paralelamente, é possível observar um fortalecimento do desenvolvimento independente de jogos, impulsionado pela disponibilidade e popularização das engines acessíveis e open-source. Tais ferramentas permitem que desenvolvedores com recursos limitados consigam produzir projetos com níveis mais elevados de complexidade técnica e criativa, ampliando a participação de pequenos estúdios no mercado (Holfeld, 2024).

Nesse segmento, a Godot Engine tem se destacado no cenário independente devido a sua natureza de código aberto (open-source), arquitetura modular e com forte apoio da comunidade. Essas características permitem a criação de sistemas estruturados por meio da composição de nós e cenas reutilizáveis, favorecendo uma organização modular dos projetos (Salmela, 2022). Estudos mais recentes indicam que o uso da engine teve um crescimento significativo em plataformas de distribuição digital voltadas a desenvolvimento independente, consolidando-a como alternativa viável frente às engines proprietárias tradicionais do mercado (Holfeld, 2024).

“A Godot tem apresentado um aumento significativo de uso em plataformas de desenvolvimento de jogos independentes” (Holfeld, 2024, p. 6, tradução nossa).

Porém, mesmo com a expansão do uso de tais ferramentas, diversos projetos de jogos digitais enfrentam desafios relacionados à

organização estrutural, escalabilidade do código ao longo do tempo e modularização adequada de seus sistemas. Projetos que iniciam com um escopo simplificado frequentemente sofrem degradação arquitetural à medida que novos recursos são integrados de forma acoplada, tornando o sistema frágil. Conforme aponta Schell (2014), o ciclo de design de jogos é integralmente iterativo, exigindo modificação e expansão constante, tornando-se fundamental a adoção de padrões organizacionais que protejam a integridade interna do software. Adicionalmente, é possível observar uma lacuna na literatura científica que tange à sistematização de boas práticas de engenharia de software aplicadas especificamente à arquitetura da Godot Engine (Holfeld, 2024; Salmela, 2022).

Diante dessa problemática, emerge a seguinte questão de pesquisa: de que forma o desenvolvimento de um jogo digital de escopo reduzido utilizando a engine Godot pode contribuir para demonstrar e documentar boas práticas de organização estrutural e modularização de sistemas no processo de desenvolvimento de jogos digitais?

Para responder a este problema, o objetivo geral deste trabalho consiste em desenvolver um jogo digital na Godot Engine para investigar como a organização estrutural e a modularização de sistemas influenciam a escalabilidade e a manutenção do projeto, validando empiricamente a importância de uma arquitetura limpa no domínio de jogos. Para tanto, definem-se os seguintes objetivos específicos: i) investigar padrões de desenvolvimento aplicáveis a jogos independentes que priorizem clareza, reuso e escalabilidade; ii) analisar o comportamento prático da arquitetura baseada em nós e cenas da ferramenta escolhida; iii) conceber e implementar um protótipo funcional de escopo delimitado denominado Ritebreaker;

e iv) avaliar qualitativamente e quantitativamente a contribuição da arquitetura modular na manutenção e estabilidade do software obtido.

A relevância deste artigo se justifica pela necessidade de mitigar o empirismo técnico recorrente em equipes iniciantes e desenvolvedores independentes. Mesmo que os jogos digitais operem como sistemas complexos, formados por múltiplos elementos independentes (Adams; Dormans, 2012; Salen; Zimmerman, 2003), o acesso facilitado às engines não garante a qualidade interna do código gerado. Assim, a criação de documentação técnica de um protótipo sob condições controladas serve como um guia prático aplicado, unindo fundamentos conceituais e implementação real para auxiliar na formação de desenvolvedores e garantir que futuros projetos escaláveis e sustentáveis sejam amplamente utilizados.

## 2. FUNDAMENTAÇÃO TEÓRICA

Fundamentalmente, o desenvolvimento de jogos digitais engloba a interação de múltiplos conceitos originados tanto da área de computação quanto do design de sistemas interativos. Diante disso, compreender os fundamentos teóricos relacionados à estruturação de sistemas, mecânicas de jogo e organização do desenvolvimento é fundamental para a construção de projetos consistentes. Segundo Game Mechanics: Advanced Game Design, as mecânicas de jogo podem ser interpretadas como estruturas formais que definem as interações e regras presentes no sistema, sendo assim responsáveis pela determinação do comportamento e dinâmica do jogo (Adams; Dormans, 2012, p. 3). Dessa maneira, a organização de tais mecânicas

é uma parte importante que determina o funcionamento adequado do sistema como um todo.

De maneira complementar a essa visão, Rules of Play: Game Design Fundamentals destaca que os jogos digitais são sistemas formados por regras que produzem experiências significativas a partir da interação do jogador com o jogo. Os autores afirmam que o design de jogos envolve a criação de sistemas em que as ações do jogador acabam resultando em consequências que são perceptíveis e integradas ao contexto do jogo (Salen; Zimmerman, 2003, p. 34). Portanto, o elemento central para garantir a coerência e a consistência da experiência do usuário acaba se tornando a organização estrutural do sistema.

Além da estrutura conceitual dos jogos, é fundamental considerar o processo de desenvolvimento como um fator que pode determinar de maneira importante a qualidade do produto final. Como citado em The Art of Game Design: A Book of Lenses, o desenvolvimento de jogos digitais é um processo iterativo, em que diferentes sistemas são ajustados constantemente e refinados ao longo do tempo (Schell, 2014, p. 89). Essa característica iterativa necessita de uma organização clara e bem estruturada, permitindo que seja possível realizar modificações sem comprometer a integridade de todo o projeto. Dessa forma, práticas de organização e documentação se tornam uma peça essencial para garantir que o jogo tenha uma evolução contínua.

**A Godot Engine e o Desenvolvimento Independente**

Através de uma visão técnica, as game engines desempenham um papel de extrema importância ao fornecer as ferramentas

necessárias para que seja possível implementar um jogo. Dentre as diversas ferramentas disponíveis, a engine Godot tem se destacado no cenário de desenvolvimento independente, graças a sua arquitetura baseada em cenas e nós, o que acaba permitindo a construção de sistemas modulares e reutilizáveis (Salmela, 2022, p. 15). Segundo o autor, tal abordagem acaba facilitando a organização do projeto, permitindo a separação de responsabilidades entre diferentes componentes do sistema.

A Godot Engine foi desenvolvida originalmente pelos programadores Juan Linietsky e Ariel Manzur, inicialmente para uso interno no OKAM Studio, na Argentina. Após vários anos de desenvolvimento proprietário, o projeto foi lançado ao público como uma ferramenta de código aberto sob a licença MIT em 2014. Desde seu lançamento, a engine tem tido grande destaque no cenário de desenvolvimento independente devido à sua arquitetura modular e ao forte suporte da comunidade (Salmela, 2022).

A ferramenta se diferencia de forma técnica por seu sistema, que é baseado na composição de nós e cenas reutilizáveis, o que permite uma organização sistêmica eficiente e a separação de responsabilidades dentro de um projeto (Salmela, 2022). Adicionalmente, estudos mais recentes demonstram que a adoção da Godot tem aumentado de forma significativa na indústria de jogos independentes, se consolidando como uma alternativa viável diante das engines proprietárias (Holfeld, 2024, p. 6). Tal crescimento destaca a relevância da engine no contexto atual e reforça a importância de compreender suas particularidades, principalmente as que dizem respeito à organização estrutural e ao desenvolvimento de projetos escaláveis.

Porém, mesmo com a disponibilidade de ferramentas e da evolução constante das engines, diversos projetos de jogos digitais ainda apresentam problemas que estão relacionados à sua organização estrutural e à ausência de documentação adequada. Problemas como estes se tornam mais evidentes em projetos desenvolvidos por iniciantes, nos quais a falta de estrutura e planejamento pode acabar dificultando a manutenção e a expansão do sistema. Diante desse contexto, a adoção de boas práticas de desenvolvimento bem definidas, tendo o auxílio de documentação adequada, pode contribuir de forma muito positiva para a melhoria da qualidade dos projetos.

Portanto, a literatura estudada e analisada demonstra a importância da organização estrutural, da modularização e da documentação no desenvolvimento de jogos digitais. A partir das contribuições de autores como Adams e Dormans (2012), Salen e Zimmerman (2003), Schell (2014), Salmela (2022) e Holfeld (2024), torna-se possível a compreensão de que a construção de sistemas interativos e eficazes não depende apenas da implementação de técnicas, mas também da forma como os elementos do jogo são organizados e documentados. Dessa forma, esta pesquisa fundamenta-se nos conceitos apresentados para propor a aplicação prática dessas abordagens no desenvolvimento de um jogo digital utilizando a engine Godot.

## 3. METODOLOGIA

### 3.1. Caracterização da Pesquisa

A presente pesquisa se classifica como aplicada e de abordagem qualitativa, uma vez que busca resolver um problema prático, a

organização de sistemas em desenvolvimento independente, por meio de fundamentação teórica e experimentação técnica, tendo como objetivo a avaliação de como a modularidade pode otimizar o fluxo de trabalho em motores de jogos de código aberto.

O método adotado na pesquisa é o experimental, fundamentado no desenvolvimento de um protótipo funcional. A partir desse processo, torna-se possível observar a interação entre componentes de software e avaliar a eficiência da arquitetura proposta em um cenário real de desenvolvimento sob restrição de tempo. O projeto foi desenvolvido em um ciclo iterativo e incremental com a duração de dez dias, unindo conceitos de game design à engenharia de software para validar uma arquitetura modular na Godot Engine.

### 3.2. Conceituação e Inspirações do Protótipo

O desenvolvimento teve início a partir da análise de pilares do gênero horde survival e roguelike, notoriamente caracterizados por alta densidade de elementos simultâneos e progressão incremental. Mecanicamente, o projeto sintetiza a dinâmica de combate contra hordas de Vampire Survivors (Poncle, 2021), o ritmo de progressão de 20 Minutes Till Dawn (Flanne, 2022) e a dificuldade baseada em tempo de Risk of Rain 2 (Hopoo Games, 2019). Esteticamente, a atmosfera e a narrativa do protótipo se fundamentam no horror cósmico[3] clássico, utilizando elementos baseados na obra O Rei de Amarelo (Chambers, 2020). O jogo adota uma perspectiva top-down, em que o jogador controla um personagem que deve sobreviver ao enfrentar hordas contínuas de inimigos.

### 3.3. Escolha da Arquitetura Modular e Aplicação do Processo RUP

A escolha da arquitetura modular se deu a partir da necessidade de garantir a manutenção e o isolamento de escopo dentro do ambiente de desenvolvimento ágil. Em sistemas complexos de jogos digitais, o acoplamento rígido gera falhas em cascata, já a modularidade permite que cada mecânica, como o sistema de hordas ou as melhorias de equipamento, funcione como um componente autônomo, facilitando testes e futuras expansões.

Para ter um planejamento adequado e delimitar os requisitos do software dentro do cronograma restrito de dez dias, foi utilizada uma adaptação simplificada do Rational Unified Process (RUP). O framework estruturou o fluxo de engenharia através de quatro fases fundamentais:

1. Concepção: definição do escopo macro do protótipo, levantamento dos pilares lógicos do gênero horde survival e especificação dos requisitos mínimos para a viabilidade do projeto.

2. Elaboração: modelagem inicial da arquitetura de nós e cenas na Godot Engine, mapeamento dos riscos técnicos relacionados à alta densidade de elementos na tela e especificação dos casos de uso principais do jogador.

3. Construção: implementação iterativa do código em GDScript, em que os módulos de combate, interface e gerenciamento de recursos foram desenvolvidos de forma desacoplada.

4. Transição: fase dedicada aos testes de estresse, refinamento da taxa de quadros por segundo (FPS) e homologação do protótipo funcional estável.

Para solucionar o problema de acoplamento rígido de código, um problema comum que prejudica a manutenção do software, foram adotados dois padrões de projeto fundamentais. O primeiro é a comunicação baseada em Sinais (Signals), na qual foi implementado o padrão Observer nativo da engine: em vez de nós filhos chamarem diretamente funções de nós pais ou adjacentes, o que causa uma dependência mútua, os nós emitem sinais baseados em eventos (ex.: on_gui_input) e os sistemas centrais conectam-se a esses sinais de forma isolada, garantindo o desacoplamento. O segundo é o Design Orientado a Dados (Data-Driven Design), no qual se utilizou a classe Resource para separar os dados brutos da lógica comportamental, com estatísticas de melhorias, tipos de inimigo e armas armazenados em arquivos de recursos independentes, o que permite modificar o comportamento do jogo sem alterar o código fonte.

### 3.4. Materiais e Ferramentas

A implementação técnica do projeto teve início com a obtenção da Godot Engine, realizada de forma direta por meio do seu site oficial (Figura 1), que disponibiliza versões executáveis que não necessitam de processos complexos de instalação, sendo apenas um arquivo executável, o que facilita a configuração imediata do ambiente de desenvolvimento (Godot Engine, 2026).

**Figura 1.** Página oficial da Godot Engine.




**Fonte:** https://godotengine.org (2026)

A engenharia do protótipo foi executada utilizando a Godot Engine 4.6.2, escolhida devido a sua arquitetura baseada em nós e cenas, que nativamente favorece a modularidade, com codificação em linguagem GDScript[4], devido a sua integração direta com a engine e agilidade na prototipagem. Para a criação dos elementos visuais e ambientação inspirada no horror cósmico, foi utilizado o software Adobe Photoshop.

A estruturação prévia do sistema de arquivos é uma etapa fundamental que deve preceder o desenvolvimento técnico do jogo. Ao organizar sistematicamente os diretórios de ativos (assets), códigos-fonte e recursos, estabelece-se um ambiente de trabalho que favorece a manutenção e a escalabilidade do projeto. Essa organização lógica não apenas facilita a localização de componentes durante o ciclo de desenvolvimento, mas também otimiza o fluxo de trabalho ao garantir que os arquivos estejam acessíveis conforme as demandas do sistema surgem.

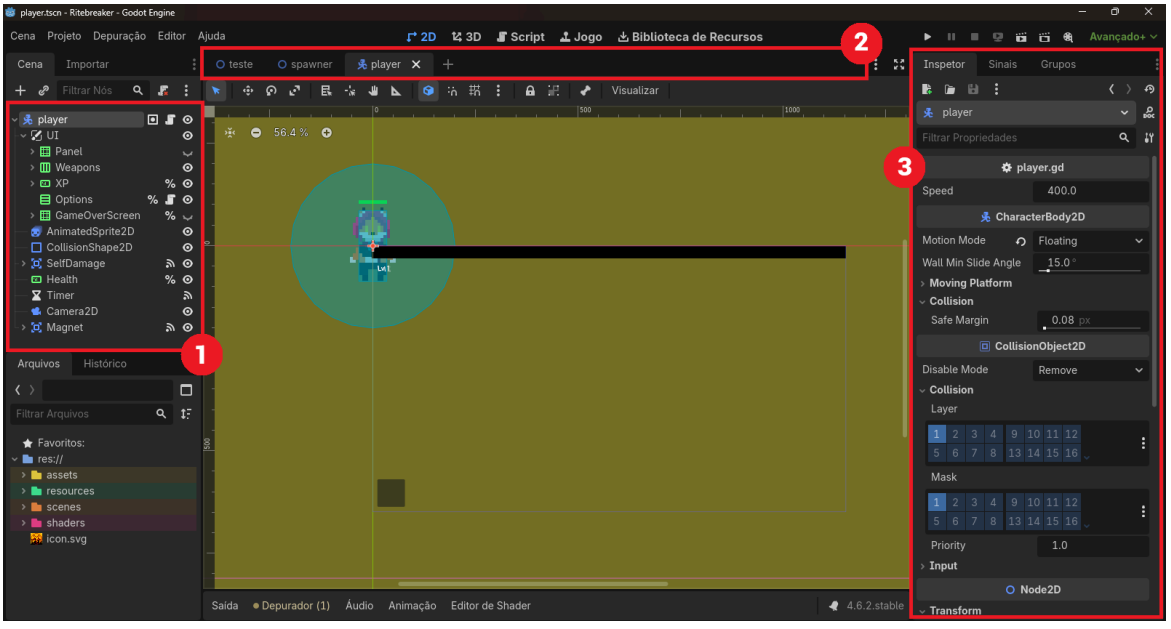
“Uma estrutura de projeto bem organizada e consistente é essencial para gerenciar a complexidade e facilitar o design modular promovido pela engine” (Salmela, 2022, p. 15, tradução nossa).

### 3.5. Arquitetura de Cenas e Composição de Nós

A arquitetura da Godot Engine fundamenta-se na fragmentação de sistemas complexos em unidades menores e gerenciáveis. Conforme aponta Salmela (2022, p. 15, tradução nossa), “a Godot utiliza um sistema baseado em nós que permite um desenvolvimento flexível e modular”. Essa estrutura permite que o desenvolvedor organize a lógica do jogo de forma hierárquica, garantindo que as mecânicas, definidas como “estruturas de regras destinadas a produzir uma experiência de jogabilidade” (Adams; Dormans, 2012, p. 3), sejam implementadas de maneira isolada e eficiente.

A interface da engine reflete essa filosofia de design através de três componentes principais, que permitem o controle total sobre a arquitetura do projeto (Figura 2):

**Figura 2.** Principais painéis da interface da Godot.




**Fonte:** Elaborada pelo autor (2026)

1. Nós (Nodes): constituem os blocos básicos de construção. Cada nó tem uma função específica, como processamento de física, exibição

de imagem ou execução de áudio, e opera em uma hierarquia de “pais e filhos”. Tal organização permite que propriedades e comportamentos sejam herdados ou compartilhados, facilitando a implementação de códigos que definem a lógica de cada elemento.

2. Cenas (Scenes): são coleções organizadas de nós que representam objetos ou sistemas completos. A maior vantagem dessa abordagem é a capacidade de conectar cenas dentro de outras, permitindo que um personagem ou cenário seja desenvolvido separadamente e integrado ao projeto final sem conflitos, promovendo o aproveitamento de elementos.

3. Inspetor (Inspector): atua como o painel de controle de propriedades, permitindo a edição direta de diversos valores e parâmetros de qualquer nó selecionado, o que possibilita ajustes rápidos e precisos em tempo real. Essa funcionalidade é essencial para o processo iterativo de design, permitindo que o desenvolvedor defina o comportamento dos elementos sem a necessidade de alterações constantes no código fonte.

### 3.6. Procedimentos de Avaliação e Validação Técnica

Para que a validação fosse realizada, foram monitoradas a taxa de quadros por segundo (FPS) e a contagem de nós ativos na árvore de cena, a fim de confirmar se o gerenciamento de instâncias e a comunicação via sinais conseguiam operar sem gargalos de processamento. Após isso, a modularidade foi validada pela facilidade de expansão do protótipo. Também foi possível testar a adição de novos tipos de inimigos e a alteração do sistema de interface de maneira isolada, comprovando que a arquitetura escolhida permite modificações isoladas.

### 3.7. Rotinas de Otimização

Devido à restrição de desempenho imposta pela alta densidade de elementos ativos, como vários inimigos simultâneos, foi desenvolvido um sistema personalizado de descarte dinâmico de instância fora da área de visão (culling lógico). Projéteis e inimigos que ultrapassam os limites da câmera são processados por uma rotina matemática de distância em relação ao jogador e, caso fiquem inativos ou distantes além do necessário, suas instâncias são liberadas da memória utilizando o comando “queue_free()”, para que não cause sobrecarga no loop de renderização da engine.

## 4. RESULTADOS E DISCUSSÕES

O processo iterativo de desenvolvimento acabou resultando na entrega do protótipo funcional estável de Ritebreaker ao final do ciclo de dez dias. A utilização conjunta das fases do RUP e das escolhas de arquitetura na Godot Engine 4 permitiram obter dados qualitativos e quantitativos sobre a estabilidade e a manutenção do software.

### 4.1. Implementação da Lógica de Combate e Hordas

A definição das mecânicas centrais de combate de Ritebreaker resultou de um processo de refinamento iterativo, no qual as referências iniciais foram decididas em relação à viabilidade técnica e ao escopo do projeto. Originalmente a jogabilidade apresentava forte inspiração nos títulos Risk of Rain 2 (Hopoo Games, 2019) e 20 Minutes Till Dawn (Flanne, 2022). Contudo, visando a simplificação da curva de aprendizado e a otimização do fluxo de desenvolvimento, o sistema de combate convergiu para uma estrutura próxima ao título Vampire Survivors (Poncle, 2021).

Na abordagem adotada, o desafio reside no cerco progressivo realizado por hordas de inimigos, o que exige do jogador a decisão constante entre o confronto e a movimentação estratégica de fuga. “Um jogo é um sistema no qual jogadores se envolvem em um conflito artificial, definido por regras, que resulta em um desfecho quantificável” (Salen; Zimmerman, 2003, p. 80, tradução nossa).

A arquitetura lógica desenvolvida para o sistema de combate foi submetida a sucessivas revisões e testes para garantir a estabilidade do software. Foi implementado um sistema de controle de instâncias que estabelece um limite máximo para a quantidade de inimigos ativos simultaneamente na cena. Essa medida de otimização é essencial para mitigar o risco de queda da performance ou instabilidade na engine do jogo durante picos de processamento, garantindo que a experiência de jogabilidade permaneça fluida mesmo sob condições de alta quantidade de elementos na tela (Figura 3).

**Figura 3.** Parte do código que faz a lógica da geração de hordas.

```
52      # Adiciona o inimigo fisicamente na cena do jogo
53      get_tree().current_scene.add_child(enemy_instance)
54
55  func get_random_position() -> Vector2:
56      # Retorna um ponto aleatório em um círculo distante ao redor do jogador
57      return player.position + distance * Vector2.RIGHT.rotated(randf_range(0, 2 * PI))
58
59  func amount(number : int = 1):
60      # Bloco de repetição (loop)
61      for i in range(number):
62          # Chama a função spawn a quantidade de vezes definida em 'number'
63          spawn(get_random_position())
64
65  func _on_timer_timeout():
66      # Executa a cada 1 segundo: aumenta o relógio e cria inimigos pequenos
67      second += 1
68      amount(second % 10)
```

**Fonte:** Elaborada pelo autor (2026)

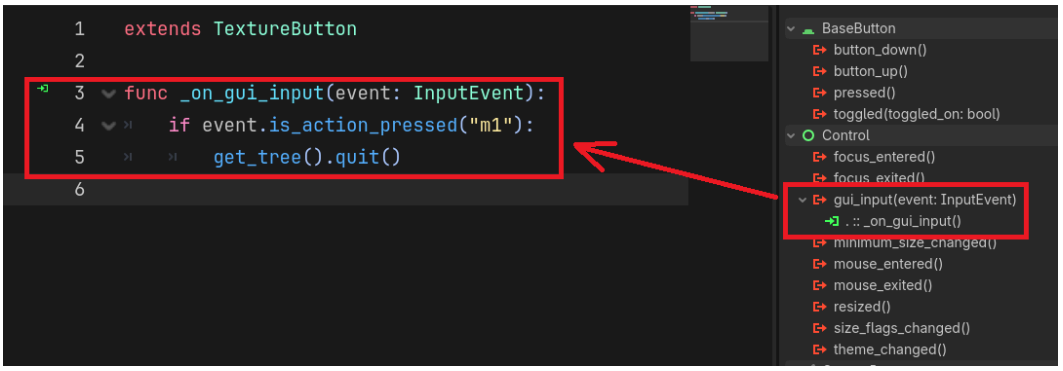
Além da estruturação lógica e das otimizações de performance feitas no projeto, a manutenção de uma documentação interna por meio de comentários detalhados em meio ao código fonte se mostrou uma prática indispensável para a viabilidade do projeto. A adição de anotações explicativas sobre a funcionalidade de cada bloco de comando facilita a identificação e correção de erros, evitando assim que o desenvolvimento tenha interrupções por dificuldade na interpretação de lógicas complexas feitas anteriormente. Essa clareza voltada para a parte técnica fortalece a perspectiva de Salmela (2022), ao destacar que a organização rigorosa e a consistência são elementos fundamentais para que o design modular seja sustentado.

### 4.2. Comunicação Entre Objetos Via Sinais

A manutenção de uma arquitetura modular necessita que os componentes do jogo sejam independentes, o que evita que alterações em um objeto causem erros em cadeia em outras partes do sistema. Na Godot Engine, essa comunicação é possibilitada através do sistema de Sinais (Signals). Esse mecanismo permite que um nó envie uma notificação sobre um evento específico sem a necessidade de referenciar diretamente o objeto que receberá a informação, promovendo o que se define como desacoplamento técnico (Salmela, 2022).

Durante o desenvolvimento do projeto, a integração via sinais foi fundamental para diversas funcionalidades lógicas e de interface. Um exemplo de uso do mecanismo foi a lógica de sair do jogo durante a tela de morte, em que o botão da interface de Game Over emite um sinal ao ser pressionado, disparando uma função que finaliza a execução do programa de forma segura (Figura 4).

**Figura 4.** Sinal de input integrado com o botão de sair do jogo.




**Fonte:** Elaborada pelo autor (2026)

### 4.3. Gerenciamento de Dados e Escalabilidade

Para assegurar a escalabilidade do projeto e mitigar o retrabalho técnico, foi aplicada a lógica de Recursos (Resources). Na Godot Engine, um recurso atua como um contêiner de dados persistentes que permite o desacoplamento entre as informações de configuração e a lógica de execução dos objetos. Essa abordagem é essencial para manter a organização sistêmica, permitindo que alterações em atributos de jogo sejam realizadas sem a necessidade de modificar o código fonte (Salmela, 2022).

A implementação prática deste conceito pode ser observada na estruturação da Classe de Inimigos (Figura 5). Ao invés de programar individualmente cada tipo de inimigo, foi criado um recurso base que armazena variáveis como título, modelo visual, vida e dano. Dessa maneira, para criar uma nova variante de inimigo, basta criar um novo arquivo de recurso e ajustar seu valor no inspetor. Essa técnica permite que o sistema de combate de Ritebreaker seja altamente flexível, facilitando o balanceamento das hordas e a expansão do catálogo de inimigos de forma ágil e segura.

**Figura 5.** Recurso da classe inimigo.

```
1   extends Resource
2   class_name  Enemy
3
4   @export var title : String
5   @export var texture : Texture2D
6   @export var health : float
7   @export var damage : float
8   @export var drops : Array[Pickups]
9
```

**Fonte:** Elaborada pelo autor (2026)

### 4.4. Interface, Gameplay e Identidade Visual

A identidade visual de Ritebreaker se consolidou na estética de horror cósmico inspirada na obra O Rei de Amarelo, de Robert W. Chambers. A interface do usuário (UI) foi projetada para exibir as informações críticas de sobrevivência de maneira clara e simples. A disposição dos elementos em tela pode ser visualizada na Figura 6.

**Figura 6.** Interface principal e cenário de jogo do protótipo Ritebreaker.



**Fonte:** Elaborada pelo autor (2026)

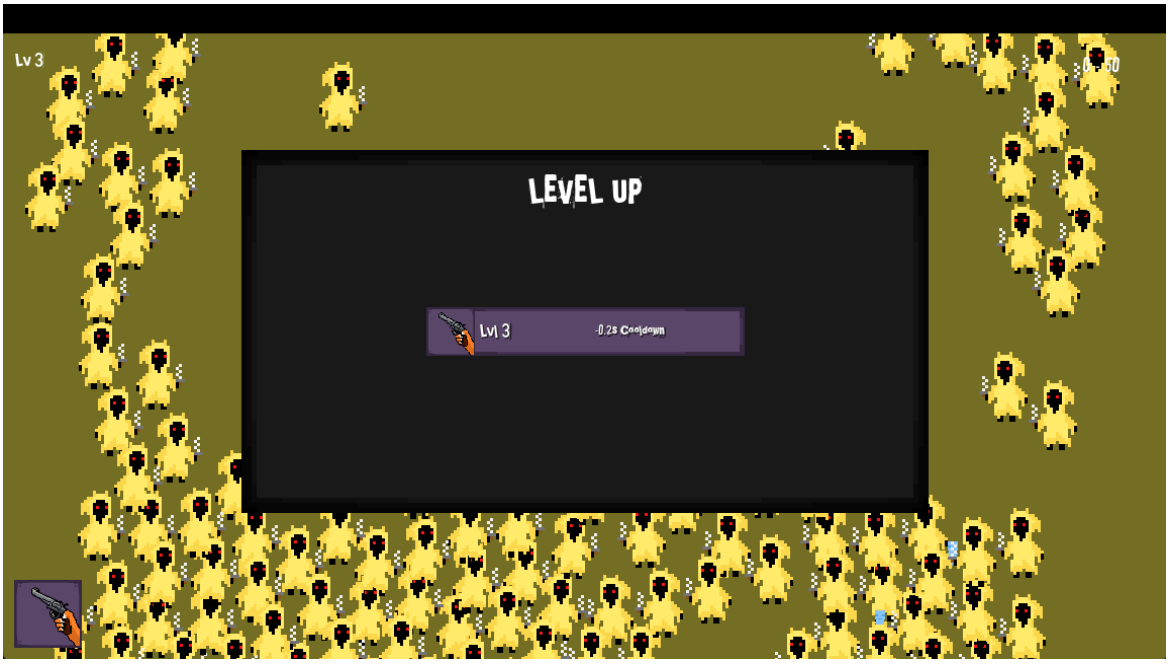
A mecânica principal do jogo se baseia na movimentação da personagem principal em um ambiente bidimensional com perspectiva top-down, enfrentando hordas crescentes de inimigos cujos comportamentos lógicos de perseguição operam de forma separada do nó do jogador.

O protótipo funcional de Ritebreaker apresenta um ciclo de jogo focado na sobrevivência e na progressão incremental. Devido ao escopo simples do projeto, a jogabilidade se concentra em três pilares fundamentais: o combate contra hordas de inimigos, a coleta de experiência (representada por almas azuis e vermelhas) para a evolução de nível e o desafio de sobreviver até o esgotamento dos pontos de vida da protagonista. Embora se trate apenas de um protótipo com escopo reduzido, a implementação dessas mecânicas básicas demonstra a viabilidade da arquitetura proposta. O desenvolvimento completo do sistema funcional em um período de dez dias evidencia a eficiência do fluxo de trabalho estabelecido, comprovando que a escolha por uma estrutura modular e o uso de recursos (resources) na Godot Engine permite a entrega de resultados satisfatórios e estáveis em curtos períodos de tempo.

A interface de Ritebreaker foi desenvolvida tendo como prioridade a clareza, oferecendo um HUD (Heads-Up Display) funcional e informativo, distribuindo os dados essenciais de forma estratégica para não obstruir a visão da área de combate. No canto superior esquerdo, o jogador encontra o indicador de nível atual da protagonista, servindo como marcador persistente de progressão. Já no canto superior direito exibe-se o tempo decorrido da sessão, um dado crucial para títulos de sobrevivência, em que a duração da partida é um dos principais indicadores de desempenho do jogador, sendo também o que define quais inimigos irão aparecer.

Para ter um feedback instantâneo sobre o estado da personagem, a barra de saúde foi posicionada de forma flutuante diretamente acima da cabeça da protagonista. Essa decisão de design permite que o jogador monitore sua integridade sem desviar a atenção do centro de ação. Da mesma forma, a interatividade e o efeito dos ataques são reforçados por meio de indicadores de dano numéricos que surgem sobre os inimigos atingidos. Para complementar o sistema de informações, a barra de progressão de experiência é posicionada ao topo da tela, fornecendo um feedback visual necessário sobre a coleta de “almas”. Ao atingir o limite de experiência, o fluxo do jogo é pausado para apresentar a interface de aprimoramento (Figura 7). Esse conjunto de elementos visuais só foi possível graças à coordenação da emissão de sinais (signals) entre a lógica de combate e os elementos de UI, permitindo uma experiência de usuário responsiva e dinâmica.

**Figura 7.** Tela de aprimoramento ao subir de nível.




**Fonte:** Elaborada pelo autor (2026)

A atmosfera de horror cósmico pretendida para Ritebreaker foi consolidada através da integração dos ativos visuais inspirados na

obra de Chambers (2020). A transição dos modelos desenvolvidos externamente para o ambiente da Godot Engine permitiu a materialização da identidade visual do projeto, com destaque para os cultistas e suas vestes amarelas, o que estabelece um tom de insanidade e mistério característicos da narrativa visual.

No que diz respeito à protagonista, foi utilizada uma abordagem que rompe o convencionalismo de personagens humanos, empregando uma figura antropomórfica, uma raposa. Trata-se de uma escolha estética que visa criar um contraste visual marcante com o ambiente sombrio do jogo, além de conferir uma identidade única ao título. Essa característica se estende também aos inimigos, que apresentam traços animalescos em suas formas, como exemplificado na Figura 8. Cabe ressaltar que a fase de implementação priorizou a estabilidade e a diversidade comportamental em detrimento da quantidade, o que garantiu que os inimigos incorporados ao protótipo apresentassem comportamentos distintos e estatísticas equilibradas, com parâmetros de resistência e dano únicos geridos através de recursos. O resultado final demonstra uma harmonia entre a estética animalística da protagonista e a natureza ameaçadora dos inimigos, cumprindo os requisitos de ambientação propostos na fase de concepção.

**Figura 8.** Demonstração da estética de Ritebreaker.



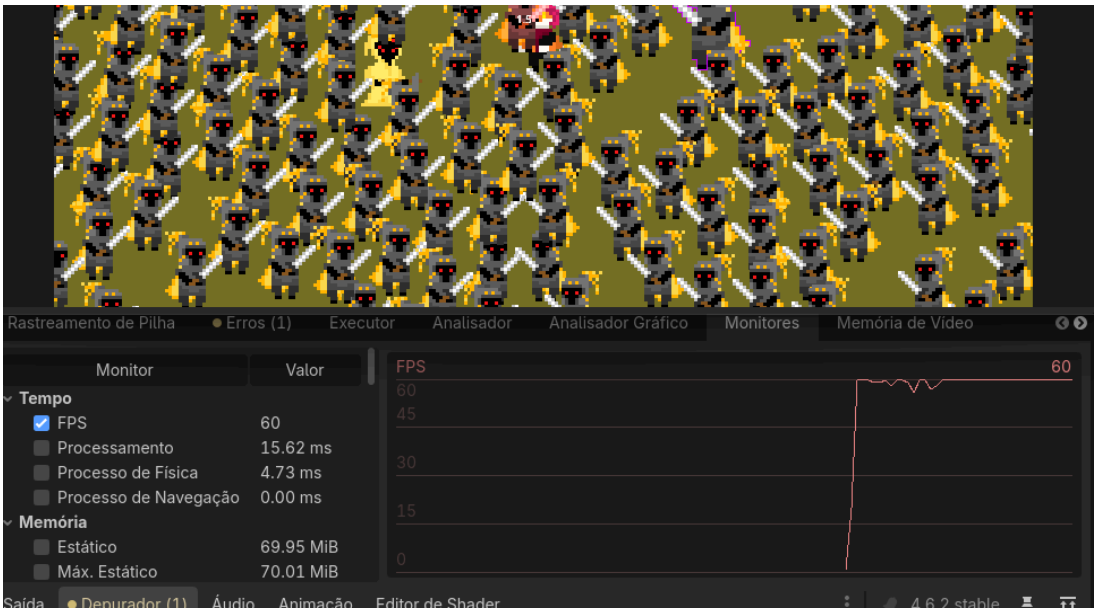

**Fonte:** Elaborada pelo autor (2026)

### 4.5. Otimização e Estabilidade Técnica

O maior desafio de engenharia identificado durante a fase de elaboração do RUP foi o risco de queda na taxa de quadros (framerate) devido à alta densidade de elementos presentes na tela. Para solucionar o gargalo de processamento, os testes de estresse validaram a eficácia do algoritmo de descarte dinâmico (culling lógico) implementado em GDScript.

Mesmo o software apresentando picos superiores a trezentas instâncias simultâneas processadas em tela, incluindo inimigos, projéteis e coletáveis, a taxa de atualização de quadros se manteve estável em 60 FPS. O descarte dinâmico de memória das entidades que estavam fora dos limites da câmera evitou acúmulo de processamento desnecessário.

**Figura 9.** Monitoramento de FPS do jogo em execução.




**Fonte:** Elaborada pelo autor (2026)

A estabilidade do protótipo foi um dos aspectos fundamentais durante a fase de implementação, visando assegurar uma experiência de jogo fluida e responsiva. Essa performance foi alcançada através de um sistema de gerenciamento dinâmico de instâncias que realiza o descarte sistemático de inimigos que estão fora do campo de visão da protagonista, reduzindo significativamente a carga de processamento e o consumo de memória. Para preservar a integridade do desafio estratégico, foi definida uma exceção lógica para os inimigos da variante “Elite”, que permanecem ativos independentemente da visibilidade em tela (Figura 9). Esse controle rigoroso sobre o ciclo de vida dos objetos demonstra a eficiência da estruturação lógica do projeto, mitigando riscos de queda de desempenho durante picos de atividade.

### 4.6. Avaliação da Arquitetura Modular

A separação de conceitos através do uso de Sinais e de Recursos provou-se altamente eficiente durante todas as fases do projeto, principalmente durante a construção do mesmo. Na Tabela 1 é apresentada uma análise comparativa entre os impactos da arquitetura modular adotada diante do modelo tradicional de

acoplamento rígido comumente observado em projetos independentes iniciantes.

**Tabela 1.** Comparativo de impacto arquitetural no desenvolvimento.

| **Critério Técnico** | **Abordagem Acoplada Tradicional** | **Abordagem Modular (Godot 4)** |
|---|---|---|
| Manutenção | Alterar um script pode acabar quebrando outras interações | Alteração isolada sem impacto colateral |
| Comunicação | Chamadas diretas de funções (get_node) | Emissão de Sinais nativos |
| Ajustes de balanceamento | Valores fixos no código (hardcoded) | Uso de arquivos Resource editáveis externos |
| Desempenho | Sobrecarga por acúmulo de instâncias | Otimização ativa por culling lógico |

**Fonte:** Elaborada pelo autor (2026).

O uso de Resources deu a possibilidade de alterar atributos de dano, velocidade e pontos de vida das hordas em tempo de design sem a necessidade de reescrever as funções lógicas no código. A comunicação totalmente baseada em Sinais fez com que os sistemas de interface (UI) e lógica do jogo funcionassem de forma completamente independente, mitigando bugs em cascata e assegurando a escalabilidade do código.

A capacidade de entregar um protótipo funcional em apenas dez dias corrobora a tese de Salmela (2022) sobre a flexibilidade que a Godot Engine proporciona. A estruturação baseada em cenas e nós deu a possibilidade de desenvolver a lógica de combate e a interface

de forma independente, sem causar conflitos de sistema. Como observado nos resultados, a modularidade não foi apenas uma escolha organizacional, mas o fator determinante para a viabilidade do projeto com um escopo simples e cronograma reduzido, permitindo que ajustes pontuais não comprometessem a integridade total do software.

A utilização de Recursos (Resources) para o gerenciamento das estatísticas dos inimigos e upgrades se provou uma estratégia eficaz para a escalabilidade. Ao comparar a prática com a teoria de Adams e Dormans (2012), é possível notar que a separação entre dados e código facilitou o balanceamento das mecânicas de horda. A estabilidade de 60 FPS alcançada, mesmo com o sistema de descarte dinâmico de inimigos, demonstra que a preocupação com a performance técnica, citada por Holfeld (2024) como um diferencial para desenvolvedores independentes, foi plenamente contemplada, o que garantiu uma experiência de usuário fluida.

### 4.7. Limitações e Desafios Durante o Desenvolvimento

Embora os resultados do desenvolvimento se mostrem satisfatórios, o projeto enfrentou limitações decorrentes do curto período de execução. A não implementação da total variedade de inimigos planejados e a ausência de sistema de áudio são pontos que evidenciam o desafio da gestão de escopo em projetos com apenas um desenvolvedor. Porém, o fato de o sistema central estar funcionando e ser facilmente expansível confirma que a estrutura lógica adotada é resiliente o suficiente para suportar futuros incrementos e adições sem a necessidade de refatoração do código base.

## 5. CONCLUSÃO

O desenvolvimento deste trabalho permitiu a validação empírica da eficácia da arquitetura modular no desenvolvimento de jogos digitais independentes, cumprindo integralmente os objetivos propostos. Por meio da criação do protótipo Ritebreaker, foi possível documentar como isolar o escopo técnico e como a aplicação prática de padrões de engenharia de software viabiliza produções ágeis e sustentáveis dentro de cronogramas restritos.

Partindo dos objetivos específicos, a investigação das boas práticas de desenvolvimento foi cumprida através de uma revisão bibliográfica que forneceu os fundamentos necessários para compreender a clareza estrutural e a reutilização de componentes em projetos independentes. Essa investigação foi de extrema importância para identificar que a modularidade e a documentação interna são pilares que fortalecem os sistemas, por mais simples que sejam, e facilitam a manutenção ao longo do tempo.

A análise da estrutura e do funcionamento da Godot Engine foi realizada a partir do estudo técnico de sua arquitetura baseada em nós e cenas, permitindo compreender como sua arquitetura nativa favorece a separação de responsabilidades. O desenvolvimento do jogo de escopo reduzido, o protótipo Ritebreaker, colocou em prática as abordagens arquiteturais estudadas: ao implementar mecânicas de horda e progressão em um ciclo de dez dias, ficou claro que, para um desenvolvedor independente, é possível criar um sistema funcional e otimizado em um prazo curto, desde que se siga um fluxo de trabalho bem estruturado.

A análise do comportamento da Godot Engine 4 demonstrou que a árvore de nós e cenas, quando associada à comunicação baseada em Sinais e ao Design Orientado a Dados com o uso de Resources, resolve com eficiência o problema do acoplamento rígido de código. Essa abordagem solucionou problemas em cascata e permitiu que alterações no balanceamento de jogo fossem feitas sem alterações nas funções lógicas do código fonte. Os testes quantitativos comprovaram que o algoritmo de descarte dinâmico foi um fator determinante para manter o desempenho do software estável em 60 FPS, resolvendo os gargalos de hardware mesmo sob condições extremas de densidade de instâncias.

Constatou-se, portanto, que a modularidade de sistemas e a adoção de processos estruturados como o RUP não apenas auxiliam a manter a integridade interna do software, mas também impulsionam a autonomia e a produtividade do desenvolvedor independente, resultando em um código limpo, organizado e resiliente. Futuramente, pretende-se dar continuidade a esta pesquisa ao explorar a implementação de algoritmos de geração procedural de cenários mais complexos e o aprofundamento na carga narrativa baseada no horror cósmico, visando uma possível publicação comercial do título em plataformas digitais de jogos.

## REFERÊNCIAS BIBLIOGRÁFICAS

ADAMS, Ernest; DORMANS, Joris. **Game mechanics:** advanced game design. Berkeley: New Riders, 2012.

CHAMBERS, Robert W. **O Rei de Amarelo.** Tradução de Ana Death S. Rio de Janeiro: DarkSide Books, 2020.

FLANNE. **20 Minutes Till Dawn.** [S. l.]: Erabit, 2022. Jogo eletrônico.

GODOT ENGINE. **Godot Engine:** free and open source 2D and 3D game engine. Disponível em: https://godotengine.org. Acesso em: 14 abr. 2026.

HOLFELD, Markus. **On the relevance of the Godot Engine in the indie game development industry.** 2024. Disponível em: https://arxiv.org/abs/2401.01909. Acesso em: 3 mar. 2026.

HOPOO GAMES. **Risk of Rain 2.** [S. l.]: Gearbox Publishing, 2019. Jogo eletrônico.

PONCLE. **Vampire Survivors.** [S. l.]: Poncle, 2021. Jogo eletrônico.

SALEN, Katie; ZIMMERMAN, Eric. **Rules of play:** game design fundamentals. Cambridge: MIT Press, 2003.

SALMELA, Tero. **Game development using the open-source Godot game engine.** Jyväskylä: JAMK University of Applied Sciences, 2022.

SCHELL, Jesse. **The art of game design:** a book of lenses. 2. ed. Boca Raton: CRC Press, 2014.

---

**AGRADECIMENTOS**

Dedico os agradecimentos aos meus pais, que sempre acreditaram no meu potencial e por todo o suporte necessário que eles me proporcionaram para eu chegar até aqui

Ao meu namorado, Roger Pinheiro, pelo companheirismo e por estar ao meu lado durante todo o desenvolvimento do jogo, sendo a pessoa que mais me incentivou.

Ao meu orientador Nilton Kazuo, pelos ensinamentos durante estes longos anos e sua dedicação ao acompanhar este trabalho.

E a todos os meus amigos, que foram meu suporte emocional e nunca desistiram de mim, me incentivando a continuar a cada etapa desta jornada.

[1] Discente do Curso Superior de Engenharia de Software do Instituto UnC *Campus* Curitibanos. E-mail: acesse o artigo original para visualizar o e-mail

[2] Docente do Curso Superior de Engenharia de Software do Instituto UnC *Campus* Curitibanos. E-mail: acesse o artigo original para visualizar o e-mail

[3] Horror cósmico é um subgênero do horror que enfatiza o medo diante da imensidão e da incompreensibilidade do universo, geralmente retratando forças ou entidades além da compreensão humana. O termo é frequentemente associado às obras de Robert W. Chambers e H. P. Lovecraft.

[4] GDScript: linguagem de programação utilizada na Godot Engine.