**ESTUDO DE CASO NO DESENVOLVIMENTO DE UM JOGO RPG DE TURNO NA ENGINE GODOT**

Yune Uzuki

Graduanda em Engenharia de Software – Uni-FACEF

[yune.uzuki@gmail.com](mailto:yune.uzuki@gmail.com)

Márcio Funes

aaa – Uni-FACEF

email[@gmail.com](mailto:yune.uzuki@gmail.com)

**Resumo**

placeholder

**Palavras-chave**: palavra-chave1. palavra-chave2. palavra-chave3.

**Abstract**

*placeholder*

**Keywords**: *keyword1. keyword2, keyword3.*

1. **Introdução**

paragrafo1

paragrafo2

paragrafo3

paragrafo4

paragrafo5

1. **Referencial Teórico**

Para que seja claro o artigo, e para que seu objetivo de analisar a viabilidade de desenvolvimento de um jogo RPG utilizando da engine Godot, é necessário um breve entendimento de alguns assuntos diretamente relacionados. Para isso, um breve resumo da história das engines, quando, quais e o motivo de terem sido criadas, sua distribuição, alguns exemplos e tipos. Além disso, para que seja possível analisar a viabilidade da Engine Godot especificamente para o desenvolvimento de um jogo do gênero RPG, é necessário também uma definição de um jogo de tal gênero, seus aspectos principais e comuns entre vários destes.

* 1. **Um pouco da história dos jogos e o efeito de hardware e software sobre o desenvolvimento**

Inicialmente breve experimentos para testar os limites e capacidades de computadores, a história dos jogos pode ser argumentada ter começado com o algoritmo criado por Alan Turing para que um computador fosse capaz de jogar xadrez contra um humano. Embora o algoritmo não seja um jogo em si, demonstra ideias iniciais relacionadas com o futuro desenvolvimento de jogos. A partir de um ponto em sua história, entretanto, jogos passaram a ser sobre a criatividade e expressão artística de seus desenvolvedores e pessoas envolvidas. A história dos jogos é uma história de pessoas e arte. (DONOVAN, Tristan. **Replay**: The History of Videogames, 2010). Esta história possui outros pontos auxiliares, entretanto, detalhes que são necessários para uma visão completa da história dos jogos. Detalhes como Hardware e as ferramentas utilizadas no desenvolvimento, sejam estas o que chamamos de Engines, ou outras ferramentas, sejam estas relacionadas com arte, música, programação, entre outros.

Hardware sempre foi o fator limitante, não o que trouxe a inovação, não o que fez com que certos jogos fossem criados, mas o que permitiu estes. O console Nintendo 64, um dos primeiros consoles com a capacidade de renderização 3D em tempo real satisfatória, não foi o motivo do jogo Mario 64 ser criado, mas sim o que permitiu as ideias do jogo serem realizadas. Conforme o hardware evolui e introduz novas funcionalidades e capacidades, desenvolvedores de jogos ganham maior liberdade sobre as ideias que podem executar (DONOVAN, Tristan. **Replay**: The History of Videogames, 2010).

Software relacionado com o desenvolvimento de jogos, diferente de hardware, não traz um fator limitante, de forma geral, a ausência de software não traz limitações, porém, a existência de software criado para auxílio, e não exclusivamente ao desenvolvimento de jogos, traz um apoio e melhoria na eficiência do desenvolvimento. Se tudo precisasse ser feito via linha de código, o desenvolvimento de jogos seria extremamente inacessível para a maioria. Imagine criar toda a arte, a música, os modelos 3D, a física de cada jogo, tudo via código.

* 1. **O que são as engines e alguns exemplos**

Game engines e sua definição podem variar bastante. Diversos artigos tentam definir game engines, e cada um acaba tendo uma definição própria, pois engines são ferramentas que diferem muito umas das outras, podendo oferecer funcionalidade únicas. Para contexto deste projeto, a definição escolhida, por questão de ser a que melhor se aplica a̍s necessidades do mesmo, foi a definição de que, uma engine precisa de: um sistema que auxilia no desenvolvimento de jogos, oferecendo, no mínimo, um sistema de visualização, um sistema de áudio ou acústico e um sistema de controle. Esta definição é baseada nos sistemas principais de grande parte dos jogos conforme definido em “SOBOTA, Branislav; PIETRIKOVÁ, Emília. The role of game engines in game development and teaching. In: **Computer Science for Game Development and Game Development for Computer Science**. IntechOpen, 2023”. Esta definição abrange praticamente todas as engines modernas, e não traz limitações como considerar apenas engines 3D ou 2D. Além disso, consegue abranger não apenas engines de propósito geral, mas também engines específicas e focadas em um gênero em específico.

Engines de propósito geral são engines feitas com o propósito de permitir a criação de qualquer gênero de jogo. Estas são mais complexas, trazem mais ferramentas, mas também podem trazer menor eficiência no desenvolvimento, tendo em vista que estas tem menos sistemas já prontos, porém oferecem maior liberdade e permitem mais criatividade por parte do desenvolvedor. Enquanto isso, engines focadas tentam trazer um processo de desenvolvimento mais eficiente para gêneros específicos, por exemplo, RPG Maker para criação de RPGs ou Ren’Py para criação de *Visual Novels*. Estes são apenas dois exemplos, mas existem infinitas engines focadas, algumas em gêneros inteiros de jogos, outras feitas específicamente para um jogo, embora raro, existem. Também temos engines comerciais como Unity e Unreal, outras open source e gratuitas, como Godot, e engines prioritárias criadas por empresas para uso interno do desenvolvimento de seus próprios jogos, como a RE Engine da desenvolvedora Capcom, atualmente sendo utilizada em todos os seus jogos modernos.

Para o propósito deste projeto, as principais engines serão Godot e Unity. Tendo em vista a importância das mesmas para o projeto, julgo relevante uma visão um pouco mais detalhada sobre cada uma destas, nos sub-tópicos a seguir. A escolha de engines foi feita por dois fatores principais – a engine Godot, sendo uma engine de propósito geral, compete diretamente com a Unity, mesmo que seja open source e não comercial. Por questão de escopo, a Unity foi escolhida por ser similar a Godot e por ser a, atualmente, mais popular na área. Devido a engine Godot ser a principal engine de estudo neste projeto, estou dando maior foco na mesma, trazendo mais detalhes sobre suas funcionalidades e diferenciais. Muita da informação relacionada com game design vem da comunidade e sites oficiais, portanto, para os próximos tópicos, estas serão as principais fontes de informação.

**2.2.1 Godot**

A engine Godot é uma engine multi-plataforma de propósito geral, open source e gratuita, que suporta o desenvolvimento de jogos 2D e 3D, atualmente, na data de escrita deste artigo, em sua versão 4.4, com betas da 4.5 já lançadas para serem testadas. A engine suporta as linguagens GDScript, C# e C++, além de permitir que o usuário crie ou use uma extensão pronta para qualquer outra linguagem que prefira programar (GODOT Engine - Free and open source 2D and 3D game engine. [*S. l.*], 202-?. Disponível em: https://godotengine.org/. Acesso em: 21 mar. 2025.). A engine teve seu primeiro lançamento em 2014, desenvolvida inicialmente por Juan Linietsky e Ariel Manzur (First Public Release! - Godot Engine. [*S. l.*], 2014. Disponível em: https://godotengine.org/article/first-public-release/. Acesso em: 21 mar. 2025.), e atualmente é mantida pela comunidade e pela *Godot Foundation*, que aceita doações de fundos para auxiliar no desenvolvimento e manutenção da engine, além de oferecer suporte oficial e gerenciar e moderar canais da comunidade.

**2.2.2 Unity**

Publicada em Junho de 2005, também uma engine de propósito geral, a Unity foi criada, inicialmente, com o objetivo de democratizar o desenvolvimento de jogos, eventualmente se tornando a game engine mais popular tanto entre grandes empresas quanto desenvolvedores pequenos. Se trata de uma engine primariamente 3D, mas que permite o desenvolvimento de jogos 2D. Se trata de uma engine prioritária de código fechado, atualmente mantida pela empresa *Unity Technologies* (UNITY Engine. [*S. l.*], 202-?. Disponível em: https://unity.com/. Acesso em: 21 mar. 2025.).

* 1. **Sobre a importância das engines no desenvolvimento**

Como já mencionado no primeiro tópico do referencial, com o tempo jogos foram se tornando mais complexos, com times maiores trabalhando nos mesmos, mecânicas e funcionalidade únicas, gráficos com sistemas de renderização mais complicados, entre diversas outras formas onde jogos se tornaram mais complexos de se desenvolver e menos acessíveis.

Para solucionar estes problemas, vieram as engines, servindo como um nível de abstração fundamental para a criação de jogos, principalmente em tempos mais atuais (Anderson, E. F., Engel, S., Comninos, P., & McLoughlin, L. **The case for research in game engine architecture**, 2008). As engines para os jogos são como as linguagens de programação para a máquina. Quando a engine lida com a renderização dos ambientes, o desenvolvedor não precisa entender exatamente como a renderização funciona, e muito menos implementar o sistema por conta própria, e isso se aplica a qualquer funcionalidade que a engine oferecer.

Outro fator importante das engines são em sua capacidade de unificar as múltiplas faces do desenvolvimento de jogos e as pessoas as quais estão atreladas com cada função. Utilizando os dados e ideias do artigo “Toftedahl, Marcus, and Henrik Engström. **A taxonomy of game engines and the tools that drive the industry.** *2019”,* o qual faz um estudo no time de desenvolvimento do jogo *Assassin’s Creed: Odyssey*, trouxe que o jogo teve um total de 692 funções diferentes relacionadas diretamente com o desenvolvimento, e uma engine precisa tratar de todas estas, e unificar as mesmas, oferecendo suporte e funcionalidade para que todos no time possam contribuir com o desenvolvimento, independente de sua experiência e função dentro do mesmo.

Por fim, engines padronizam e trazem uma forma de reutilizar implementações já prontas de funcionalidades para jogos (Anderson, E. F., Engel, S., Comninos, P., & McLoughlin, L. **The case for research in game engine architecture**, 2008). Muitas vem com sistemas já prontos de física, que o desenvolvedor precisa apenas alterar números para customizar, sistemas de áudio e gráficos que precisam apenas dos assets e uma breve programação para que a engine saiba quando tocar tais áudios ou renderizar cada gráfico, entre diversas outras funcionalidades que engines podem ou não incluir para facilitar o desenvolvimento.

* 1. **Jogos RPG de turno, fatores comuns e franquias relevantes**

Os RPGs de turno tem suas raízes nos jogos de RPG de mesa, o exemplo mais conhecido e de origem sendo *D&D (Dungeons and Dragons).* RPGs de mesa são jogos onde cada jogador cria personagens, e um “mestre” cria um mundo e descreve acontecimentos e ações. Cada jogador tomando controle de seu próprio personagem. O propósito dos RPGs de mesa é proporcionar aos jogadores uma história, permitir que cada um utilize de sua imaginação para interpretar os ocorridos, se sentir como se fosse o personagem o qual criou, durante o tempo de jogo. Ações são tomadas em turno, todas baseadas nas habilidades do seu personagem, e muitas baseadas em roladas de dados, mais conhecido como D20, por ter 20 faces. Ações podem ser de combate, negociação, diálogo, investigação, o único limite sendo a criatividade dos jogadores. (WIZARDS RPG TEAM; WYATT, James; SCHWALB, Robert J; CORDELL, Bruce R. **Dungeons & Dragons: Player‘s Handbook**. [*S. l.*: *s. n.*], 2014.).

Com relação a combate em turno, como o nome já indica, se trata de combate onde jogador e inimigos tomam turnos para executar suas ações. O combate em turno, por normalmente oferecer ao jogador tempo ilimitado para executar suas ações e pensar nas mesmas, traz complexidade estratégica, onde o desafio do jogo está no jogador se preparar bem para uma batalha ou situação, e fazer bom uso das habilidades de seus personagens contra as habilidades de seu oponente. Algumas ações comuns em diversos jogos do gênero são um ataque normal, habilidades únicas de cada personagem, utilização de magia, utilização de itens para auxiliar, ações de cura, entre outras (AMEREH, Fatemeh. **A Study and Implementation of Turn-Based Combat Systems in Role-Playing Games**. 18 dez. 2024.). Outro fator em comum são recursos que precisam ser gerenciados além da saúde de cada personagem, geralmente chamados de *MP (Mana Points)*, e utilizados para executar ações mais fortes ou com efeitos mais impactantes, geralmente ataques mais fortes ou únicos de um personagem. Por fim, é importante considerar os valores, ou estatísticas, que definem cada personagem, estatísticas como força, magia, defesa, agilidade, destreza, entre outros, que afetam o combate e definem o que cada personagem consegue executar com maior eficiência, além de seus pontos fracos.

Alguns exemplos de franquias são *Final Fantasy,* definitivamente a mais conhecida, com Final Fantasy VII sendo, até hoje, um dos RPGs mais vendidos na história, *Dragon Quest* da mesma empresa desenvolvedora (*Square Enix*)*,* *The Legend of Heroes*, uma franquia menos popular, mas de grande relevância para fãs do gênero atualmente, *Chained Echoes,* *Sea of Stars* e *Undertale* são recentes exemplos de RPGs de turno criados por pequenos times, ou até mesmo uma única pessoa, além de diversos outros exemplos que poderia citar.

1. **Materiais e Métodos**

**3.1 Metodologia**

Para este projeto, será adotada uma metodologia onde, os problemas gerados pelo desenvolvimento de um jogo RPG de turno serão levantados e tratados como requisitos, gerando um “GDD”, *Game Design Document*, baseado nos mesmos. Este documento estará descrevendo como cada funcionalidade do jogo deve funcionar, considerando não apenas a funcionalidade mas também descrevendo qualquer feedback em forma de ou visuais que devam ocorrer. Além disso, neste documento também será descrito o ambiente, os personagens, os efeitos sonoros, entre quaisquer outros recursos que serão necessários para desenvolvimento do jogo. Voltando às funcionalidades e mecânicas do jogo, estas terão um olhar focado na engenharia de software, ou seja, como mencionado antes, serão tratadas como requisitos, e para cada funcionalidade será gerado um documento 2W1H, descrevendo os fatores o quê, por quê e como da implementação desta funcionalidade. Estes documentos estarão presentes no GDD. Outro detalhe importante é que todas as soluções apresentadas devem ser ferramentas e funcionalidades oferecidas pela engine Godot, de forma a demonstrar que esta oferece ferramentas e métodos para solucionar os problemas comuns do desenvolvimento de um jogo RPG de Turno. Codificação será necessária para algumas das soluções e, embora a engine suporte a linguagem C# e até C++ com plugins, a linguagem utilizada será a GDScript, uma linguagem da própria engine, que acarreta em menos linhas de código por ser bem integrada e já incluir diversos facilitadores no desenvolvimento. Mesmo que possa ser feito uso da codificação, será apenas caso necessário, sempre que possível serão utilizadas apenas ferramentas prontas da engine.

Os problemas elencados para solução devem ser problemas exclusivos ou absolutamente necessários aos RPGs de Turno, então todos devem ter relação com a funcionalidade das batalhas de turno, sendo o fator que define o gênero. Então, por exemplo, funcionalidades relacionadas com exploração de masmorras não serão incluídas neste trabalho, como movimentação do personagem, pois exploração, embora comum, não é um fator que define o gênero ou que necessita estar presente para um jogo ser definido como RPG de Turno.

Por fim, após a elaboração dos documentos, será feito o desenvolvimento de um simples protótipo, de pequeno escopo, para demonstrar os resultados da análise de viabilidade e do desenvolvimento. Este protótipo será completamente baseado no GDD criado, incluindo todas as funcionalidades descritas no mesmo.

Todos os artefatos gerados por este projeto estarão disponíveis em um repositório, utilizado para versionamento do código fonte e dos documentos utilizados no desenvolvimento do projeto. Por fim, após todo o processo de desenvolvimento, as funcionalidades utilizadas, capacidades da engine serão avaliadas em um caso de uso. O foco deste caso de uso será em apresentar os diferenciais da engine, onde a mesma auxilia no desenvolvimento de um RPG de Turno e onde a mesma traz alguns desafios, então não irá cobrir todas as funcionalidades utilizadas no processo de desenvolvimento, tendo em vista que diversas são funcionalidades comuns em praticamente qualquer engine 3D, como renderização de modelos 3D, texturas, luzes, gerenciadores de efeito sonoro ou música, entre outros.

**3.2 Ferramentas**

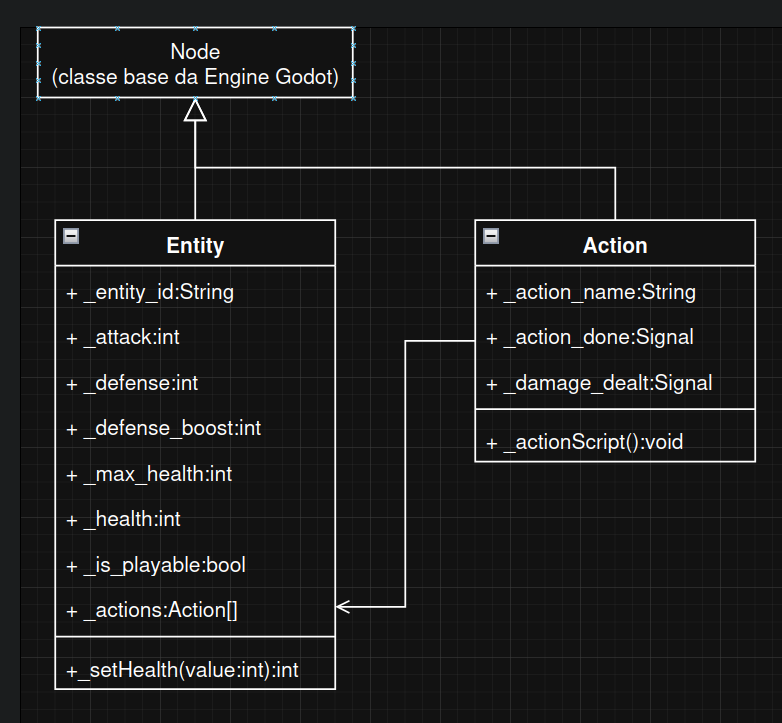
Para documentação, estarei utilizando a ferramenta Obsidian, um editor de texto, primariamente focado em anotações, mas que permite a criação e edição de arquivos do tipo MD (*Markdown*) e até mesmo de diagramas, além disso será utilizado [draw.io](http://draw.io) para a criação dos diagramas de classe. A documentação estará em um arquivo do tipo MD. Para o desenvolvimento, conforme o foco de todo o artigo, será a engine Godot, com a linguagem de programação GDScript, conforme já mencionado. As ferramentas dentro da engine Godot serão detalhadas durante o tópico de desenvolvimento, pois são diversas e mais simples de se explicar por meio de exemplificação. O versionamento do trabalho será feito via Git com Github. O repositório irá conter uma pasta para a documentação, uma pasta para o projeto Godot, ou seja, o código fonte e todos os *assets* do jogo, um arquivo README auxiliando a navegação pelo repositório e um arquivo LICENSE detalhando a licança para o projeto.

1. **Desenvolvimento e Prototipação**

Este tópico será separado em dois sub-tópicos, um sobre a documentação do projeto, um sobre o desenvolvimento do jogo em si. O sub-tópico da documentação será breve, falando apenas sobre o que considerei mais importante incluir na documentação e como foi formatada, além de um anexo do diagrama de classes utilizado no projeto. Já o sub-tópico do desenvolvimento será mais detalhado, trazendo informações sobre cada ferramenta utilizada, brevemente explicando a mesma, um pouco de seus usos e o por quê de ter utilizado a mesma.

**4.1 Elaboração do GDD e Diagramas de Classe**

Para o GDD, separei o documento em 4 tópicos com diversos sub-tópicos. Proposta e Escopo, Mecânicas, Visuais e Áudio. O mais importante para o estudo será o tópico de Mecânicas. Neste tópico, inclui os sub-tópicos Estatísticas, Ações, Turnos e Estados de Vitória ou Falha. Cada um destes sub-tópicos representa um dos “problemas” levantados relacionados ao desenvolvimento de um RPG de Turno, as mecânicas que definem o gênero. Para cada um destes sub-tópicos, o GDD descreve o quê este é, dando uma breve descrição da mecânica, o porque da mecânica e, por fim, o como será implementada a mecânica tendo em vista as funcionalidades da engine Godot. As funcionalidades de estatísticas e ações precisam de uma classe, conforme detalhado na documentação, para sua implementação, então, segue, na figura abaixo, o diagrama de classe elaborado:



subtexto imagem

**4.2 Desenvolvimento do Jogo**

O desenvolvimento se dá em diversas partes, pois são vários fatores que precisam ser feitos em um jogo. Estarei seguindo na ordem a qual desenvolvi. O primeiro passo no desenvolvimento foi a criação dos visuais do jogo. Cenário, personagens e animações. Os anexos para esta seção estarão no repositório, devido ao alto volume de anexos.

1. **Estudo de Caso**

Com todo o desenvolvimento e todos os estudos necessários feitos, chegou a hora de discutir sobre a engine, o processo de desenvolvimento e como a mesma auxiliou ou talvez dificultou o desenvolvimento. Nesta seção, primeiro irei tratar diretamente sobre o processo de desenvolvimento, mencionando as ferramentas utilizadas e proporcionadas pela engine no desenvolvimento do jogo criado neste artigo. Irei discutir um pouco também sobre fatores externos ao desenvolvimento, que acabam sendo relevantes para os desenvolvedores, por exemplo, as questões de licenciamento e o fato da engine ser open source. Considerando também o escopo pequeno do jogo desenvolvido, irei tocar em algumas ferramentas que não precisei utilizar no meu desenvolvimento, mas que seriam relevantes para a criação de um RPG de Turno mais completo, que inclue, como mencionado em capítulos anteriores, exploração de masmorras, diálogos entre personagens, sistemas de equipamento e estatísticas mais robustos, entre outras mecânicas comuns no gênero.

Procuro usar o máximo de métricas e fatos embasados em outros estudos o possível para determinar a viabilidade da engine, e esta seção do trabalho se trata dos argumentos a favor e contra o uso da Godot no desenvolvimento de um RPG de Turno na Engine Godot, porém, de certa forma, esta definição acaba sendo levemente opinativa, pois não consigo definir um ponto exato onde, por exemplo, tenhno provas suficientes para dizer que a engine é objetivamente viável, então embora esteja tentando ser o mais objetiva possível, ainda ocorrerá um pouco de opinião nesta seção.

**5.1 Composição**

Em uma discussão sobre a engine Godot, se torna praticamente obrigatório argumentar sobre a metodologia de composição, como funciona, os benefícios e possíveis dificuldades que a mesma pode trazer para o projeto que será feito na mesma, por isso, será o primeiro tópico nesta seção, onde irei discutir sobre seu impacto no desenvolvimento do combate em turno e sobre a modularidade que a mesma insere no processo de desenvolvimento do jogo, me baseando no meu próprio desenvolvimento e em exemplos que posso relacionar com o tema.

**5.1.1 Benefícios no desenvolvimento do combate em turno**

Como mencionado anteriormente, a engine Godot, em prática, funciona por meio de composição. O usuário adiciona um nó, dentro deste nó, mais nós, e assim compõe um objeto para seu jogo. Por exemplo, em um jogo de plataforma 3D, para criar o personagem principal, uma organização utilizada por diversos desenvolvedores é de ter um nó principal, como filhos deste, um nó que lida com o visual, que possue como filhos o modelo 3D do personagem e um nó para lidar com animações, além do nó que lida com o visual, o nó princípal também tem como filho todos os nós que lidam com funcionalidades, estados, física, entre outros.

Durante o desenvolvimento de meu jogo para este artigo, a composição foi usada durante todo o processo, pois é o básico de como a engine funciona, mas o interessante é observar o impacto da composição durante o desenvolvimento do sistema de combate, mais especificamente, das ações que cada personagem pode executar. Cada nó pode ter um script próprio, e scripts em diferentes nós podem ser interconectados por meio de referências. Por exemplo, um nó pai pode buscar seus nós filhos e adicionar uma referência aos mesmos em uma variável, e, a partir desta variável, o nó pai possui acesso as funções definidas dentro do script destes nós. Utilizando da composição, consegue-se facilmente criar máquinas de estado. Todo o combate do jogo se trata de uma grande máquina de estado, onde cada personagem é um estado diferente. Desta forma, para criar um personagem que consegue utilizar magia, cura e distração, tudo que precisaria fazer seria adicionar um novo nó da classe entidade e, abaixo deste nó, adicionar os nós referentes a estas funcionalidades, compondo um novo personagem com as capacidades definidas, utilizando o sistema de nós e cenas da engine para criar nós que guardam toda a funcionalidade de uma habilidade e que podem ser aplicados a qualquer entidade de combate.

Isso se provou uma metodologia extremamente benéfica para o desenvolvimento do jogo deste projeto, mas também seria útil para qualquer RPG de Turno, tendo em vista que acelera o processo de criação de novos personagens e inimigos. Um RPG de Turno em escala maior precisa de diversos inimigos e personagens, e, por meio da composição, o processo de criar a funcionalidade para entidades novas se torna praticamente um processo de arrastar nós com funcionalidades prontas para a entidade, assim como foi feito no desenvolvimento deste projeto. Claro, isso toma uma quantia de codificação, o desenvolvedor ainda precisa codificar a máquina de estado e como a mesma funcionará, ainda precisa codificar cada funcionalidade dos nós que serão utilizados para compor os personagens, mas após criar este nó, é apenas uma questão de re-utilizar estes nós onde precisar de funcionalidade similar.

Embora tenha falado apenas de personagens, esta metodologia pode ser levada além de apenas personagens. Um caso comum em jogos RPG são personagens com diversas habilidades que podem executar mais de uma função, por exemplo, um personagem pode ter uma habilidade que ataca um inimigo 3 vezes e depois soma o dano destes ataques e cura o personagem que atacou com base no dano que o alvo recebeu, com uma boa estrutura de código, uma habilidade assim poderia ser composta por 3 nós com funcionalidade de ataque e um nó com uma funcionalidade de cura. Novamente preciso dizer, isso necessita que o desenvolvedor primeiro crie um script que irá criar referências aos nós filhos e executar suas funcionalidades, mas por meio de algumas funcionalidades da engine, como a função *get\_children*, que retorna, em formato de vetor, uma lista de todos os filhos do nó em que for executada, e em conjunto com a funcionalidade *class\_name*, que permite a criação de classes, pode-se, por exemplo, criar uma classe chamada *funcionalidade*, esta classe possuirá a função *executar\_funcionalidade* e seria utilizada para criar os nós que irão compor uma habilidade, já no script do nó da habilidade, executa-se um script que usa a função *get\_children* para gerar uma lista de todos os nós de funcionalidade desta habilidade, e sempre que a habilidade é ativada, o script passa por toda essa lista executando a função *executar\_funcionalidade* de cada filho.

Em conclusão, a metodologia de composição se prova extremamente útil no desenvolvimento de um RPG de Turno pois, em natureza, é um gênero que envolve muitas habilidades, personagens e inimigos que compartilham de certas mecânicas e funções, onde esta metodologia consegue auxiliar e trazer o aspecto de re-utilização dessas funções, com menos codificação e reduzindo os grandes diagramas de classes e tempo gasto com todo um planejamento da estrutura do código para conseguir alcançar o mesmo nível de re-utilização no mesmo. Estas vantagens, somado com as funcionalidades que a engine propõe para facilitar o uso da metodologia resultam em uma redução da quantidade de linhas de código escritas, consequentemente reduzindo custos e tempo, além de permitir que desenvolvedores menos familiares com o aspecto de programação consigam contribuir com a funcionalidade do jogo, isso se mostra um fator importante pois diversos contribuidores em um time de desenvolvimento de jogos se trata de designers, não de programadores.

**5.1.2 A modularidade da composição**

Assim como mencionado no tópico anterior, ao exemplificar o uso da composição na criação de um personagem em um jogo de plataforma 3D, a engine Godot, com o sistema de nós, cenas e a metodologia de composição, proporciona ao desenvolvimento um alto nível de modularidade. Esta modularidade traz diversos benefícios a criação de um jogo, de forma geral, mas também na criação de um RPG de Turno, como discutido no tópico anterior. Embora já tenha tratado deste benefício para, especificamente, o sistema de combate em turno, gostaria também de discutir, de forma um pouco mais breve, os benefícios gerais da modularidade provida pela metodologia de composição.

Quando um desenvolvedor separa e deixa modular as características de um objeto, como separar um personagem entre visuais, áudio e funcionalidades, a facilidade de manutenção e alterações a este objeto é maior. Suponha que, para criação das primeiras versões de um jogo, os desenvolvedores queiram focar na funcionalidade, então os modelos do personagem e animações do mesmo são feitas de forma rápida, mas com o intuito de que estes aspectos sejam melhorados futuramente. Em um ambiente onde se aplica modularização, os artistas do time, além de terem seu trabalho facilitado, pois não vão precisar encostar no código e ̍̍apenas interagir com interfaces mais amigáveis para um artista, também podem fazer estas alterações apenas no módulo visual, de forma que outros módulos não precisem ser alterados para acomodar ao que foi feito no visual, desde que tudo tenha sido planejado e integrado corretamente.

Além de melhor manutenibilidade, um sistema modular ajuda na organização da base de código. Começando pela estrutura de pai e filho, que funciona praticamente como a indentação em um código, basta o desenvolvedor olhar para a lista de nós que irá conseguir distinguir quais nós são filhos de cada pai, além de permitir o desenvolvedor a “fechar” nós, escondendo todos os seus filhos da lista, assim, o desenvolvedor do projeto pode esconder tudo que não estará trabalhando sobre. Em conjunto com a estrutura de pai e filho, para ainda melhor organização, a engine permite a criação de nós vazios, sem scripts ou recursos, que podem ser nomeados e se tornarem pais de diversos nós filhos sem consequências para o funcionamento do jogo, permitindo uma customização em como um desenvolvedor prefere organizar os elementos.

**5.2 Open Source e licenciamento**

Um dos diferenciais da Godot sobre seus principais competidores é o fato da mesma ser uma engine Open Source. Hoje em dia existem diversas opções de game engines Open Source, algumas até mesmo em forma de bibliotecas para linguagens, Pygame, por exemplo, que se trata de uma biblioteca composta de diversos módulos focados em auxiliar o desenvolvimento de jogos se utilizando da linguagem Python (ABOUT - pygame wiki [*S. I.*] 20–. Disponível em: https://www.pygame.org/wiki/about. Acesso em: 31 out. 2025), com a Godot sendo uma das mais populares entre estas, e uma das poucas engines Open Source a ser uma engine de propósito geral e, por fim, sendo a mais próxima deste grupo a competir diretamente com a Unity, a atual engine mais popular para desenvolvimento de jogos. Neste tópico, estarei tratando sobre as vantagens e desvantagens do fator Open Source e o impacto deste no licenciamento do software.

Tendo em vista o possível uso da engine para desenvolvimento de um RPG de Turno comercial, este tópico se torna de grande importância para qualquer desenvolvedor, seja alguém que desenvolva por hobby mas com a intenção de vender seu jogo, pequenos times de desenvolvimento e, até mesmo, grandes empresas.

**5.2.1 Licenciamento da engine**

A engine Godot está sobre uma licença MIT, uma licença extremamente permissiva, que, conforme descrito na seção de licença do website, o usuário pode utilizar da engine Godot para qualquer propósito, estudar como a mesma funciona e modificá-la e, até mesmo, modificar a mesma e distribuir, para propósitos comerciais ou não, esta versão modificada da engine, em qualquer outro tipo de licença. Também importante para o desenvolvedor é a licença sobre o jogo que o mesmo desenvolver com a engine e, neste caso, um jogo desenvolvido na engine Godot não possui restrições sobre licenciamento ou uso comercial do mesmo, o desenvolvedor decide qual licença será a melhor para seu jogo. Um detalhe importante, entretanto, é que, ao distribuir o jogo, o desenvolvedor também está distribuindo um binário da engine Godot, então o mesmo precisa incluir, nos créditos, documentação ou onde preferir, uma notificação de Copyright indicando que a engine foi utilizada no desenvolvimento e, ao mínimo, um link a página de licenciamento da engine. Todas estas informações foram retiradas da mesma. (LICENSE - Godot Engine [*S. I*], 20–. Disponível em: https://godotengine.org/license/. Acesso em: 30 ago. 2025.).

A licença MIT, em resumo, redefine a relação entre o desenvolvedor e a ferramenta, permitindo modificações e até mesmo redistribuições. A possibilidade de modificações no código da engine é um fator de extrema relevância, tornando a engine adaptável para as necessidades do desenvolvedor. Além disso, devido a forma com que a engine é licenciada, o desenvolvedor mantém completa posse sobre a propriedade intelectual que criar pela mesma, sem necessidade de *royalties* ou taxas relacionadas com o uso da engine, sobre o seu jogo. Por fim, outra vantagem sobre esse modelo de licenciamento para o desenvolvedor é a simplicidade, tudo que o desenvolvedor precisa é de baixar a engine e incluir, de alguma forma, a indicação de que seu jogo foi feito com a engine Godot e seu licenciamento original (MIT), sem precisar se preocupar com preços e licenciamento mais complexo, o que cria um caminho mais direto e transparente para desenvolvedores que desejam inserir seu jogo no mercado.

Este modelo de licenciamento traz um grande contraste sobre engines proprietárias, como a Unity, por exemplo, quais costumam trazer maiores complicações e preocupações para um desenvolvedor. Como exemplo, atualmente a Unity funciona por um modelo de inscrição, o qual depende da renda do desenvolvedor ou empresa que está utilizando a engine. Este modelo de inscrição pode ter preços ou detalhes de seu contrato e licença de uso alterados a qualquer momento. Com isso, além do desenvolvedor ter um caminho menos direto e transparente caso queira inserir seu jogo no mercado, também poderá passar por complicações financeiras devido ao alto custo e valores variáveis da engine a partir de certas rendas.

**5.2.2 Modificações na engine e comunidade**

Outra vantagem do modelo Open Source, mencionada até mesmo no tópico anterior, é a possibilidade de modificações na engine. Por ser código aberto, toda a comunidade é capaz de não apenas contribuir com o código, constantemente melhorando a engine, mas também criar plugins e modificações, adicionando funcionalidades que um desenvolvedor pode utilizar para facilitar seu desenvolvimento. Além disso, tendo acesso ao código da engine, e sobre uma engine permissiva, o desenvolvedor tem total controle da engine e possui a capacidade de criar suas próprias modificações para facilitar no desenvolvimento.

Um grande exemplo dessa vantagem da engine se mostra pela *Godot Asset Library*, um website da comunidade onde plugins e modificações criadas para a engine podem ser compartilhadas e facilmente integradas na engine. Essa biblioteca de plugins é suportada pelos principais desenvolvedores da engine, tanto que a mesma pode ser acessada diretamente pela engine Godot para fácil acesso e uso. A *Asset Library* proporciona para o desenvolvedor uma forma de encontrar soluções prontas para um problema que possa ter encontrado durante o desenvolvimento de seu jogo. Existem até mesmo plugins prontos para combate em turno que, caso possuam todas as funcionalidades que o desenvolvedor precise, já seria suficiente para cobrir toda a parte da codificação do combate em turno, permitindo que o desenvolvedor trabalhe apenas com a integração e parte artística.

Além da *asset library*, como já mencionado, a possibilidade de modificar a engine proporciona ao desenvolvedor a liberdade de adaptar a mesma para suas necessidades. A engine inteira pode ser modificada, até mesmo física e biblioteca de renderização. Isso se torna um fator mais importante ainda quando se considera que alguns jogos precisam de engines customizadas para funcionamento, devido a falta de suporte de funcionalidades extremamente únicas ao jogo, levando alguns desenvolvedores a passarem por todo trabalho de criar sua própria engine. A Godot pode servir como uma base para a criação de uma engine nova em forma da modificação da mesma, proporcionando uma solução pronta mas adaptável para o desenvolvedor.

**5.2.3 Os desafios do código aberto**

O modelo Open Source traz diversas vantagens, mas também possui alguns desafios que os desenvolvedores e usuários da engine precisam enfrentar. O primeiro destes é a demora de integração e desenvolvimento lento. Embora a engine possua uma comunidade inteira por trás da mesma, contribuindo para seu desenvolvimento, existe uma maior lentidão ao integrar tecnologias novas e manter a engine atualizada ao mesmo nível de uma engine comercial. Um exemplo simples que pode ser utilizado aqui são as capacidades sobre inteligência artificial e machine learning. A engine Godot possui integrações ainda limitadas quando se fala de IA, principalmente generativa, e machine learning. Para projetos que envolvem estas tecnologias, uma engine como a Unity está bem a frente, afinal, é uma engine comercial e proprietária, possuindo um grande time por trás do constante desenvolvimento e atualização da mesma, além de maior poder monetário. Embora poucos jogos RPG de Turno utilizem de ferramentas mais novas, um desenvolvedor pode ter a ambição de criar, por exemplo, um RPG de Turno onde os inimigos possuem sistemas complexos de machine learning, que aprendem com o jogador e ficam constantemente melhores em batalhar contra o mesmo. Em um caso destes, e outros casos mais específicos, a engine Godot pode não ser uma boa escolha.

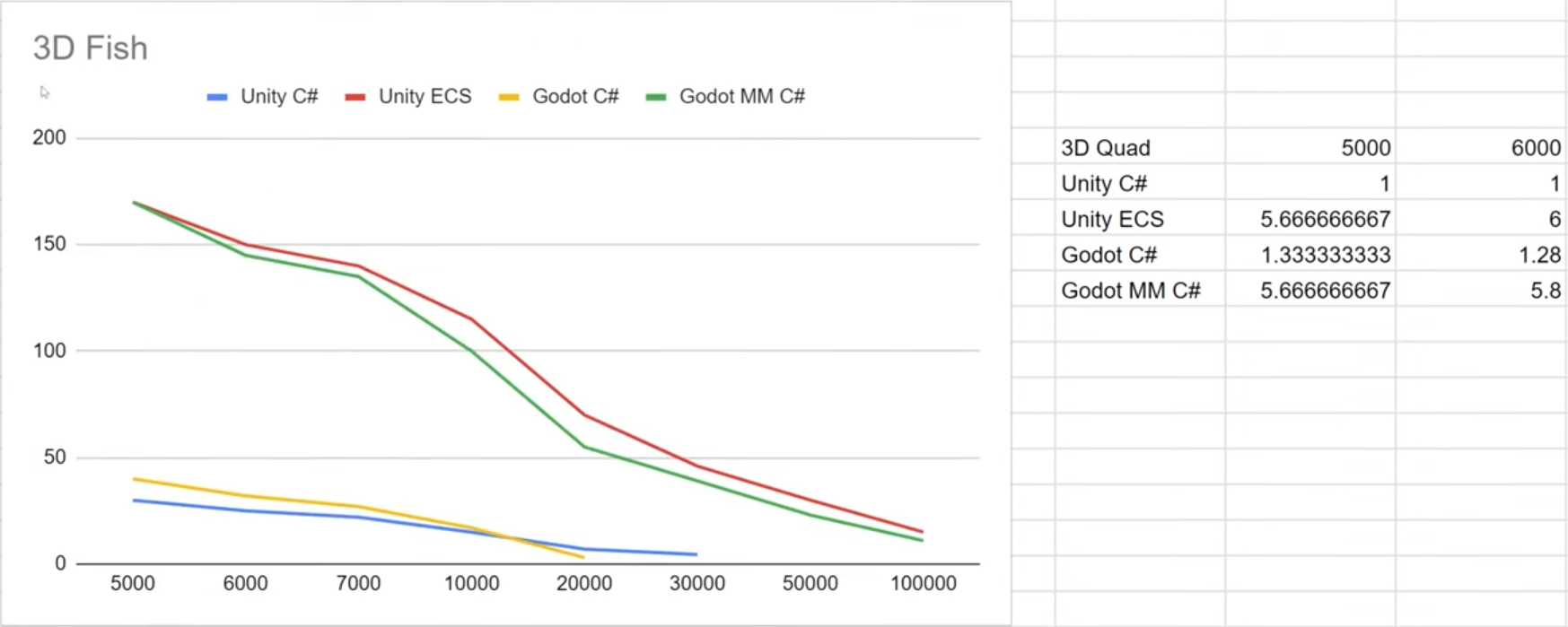
A lentidão no desenvolvimento também pode trazer problemas como funcionalidades incompletas ou instáveis, que precisam de mais atenção para que possam estar em um estado onde utilizar tal funcionalidade não acaba sendo difícil e trabalhoso. Um exemplo seria o nó *GridMap*, este nó, embora completamente funcional, possui algumas inconveniências e dificuldades em seu uso, como o fato de que os mesmos botões no teclado que são utilizados para rotacionar e trocar entre funcionalidades do GridMap, são botões utilizados para mover a câmera na engine e, não é possível travar a movimentação da câmera ou dos GridMap, causando inconveniências na interação com o mesmo. O motivo desta funcionalidade ainda estar assim é a mesma ser pouco utilizada. GridMaps podem ser utilizados para criação de mapas em 3D mas, diferente de TileMaps, que são extremamente comuns e populares em 2D, desenvolvimento de cenários 3D geralmente necessita de mais detalhes e é algo mais manual, fazendo GridMaps serem menos populares. Como poucos usam, existem menos contribuintes para a ferramenta.

Outro desafio que o modelo de código aberto e o licenciamento MIT trazem é a falta de suporte oficial para plataformas fechadas, como, por exemplo, consoles. Para que um desenvolvedor possa lançar seu jogo em um console, o mesmo precisará falar e trabalhar diretamente com uma empresa de terceiro, publicadoras e empresas especialistas em portes, representando um desafio para o desenvolvedor e reduzindo algumas das liberdades que o código aberto e licenciamento MIT proporcionam ao mesmo, além de aumentar custos no desenvolvimento e tornar publicadoras uma necessidade para um desenvolvedor que deseja publicar em consoles, não apenas em computadores.

**5.3 Performance**

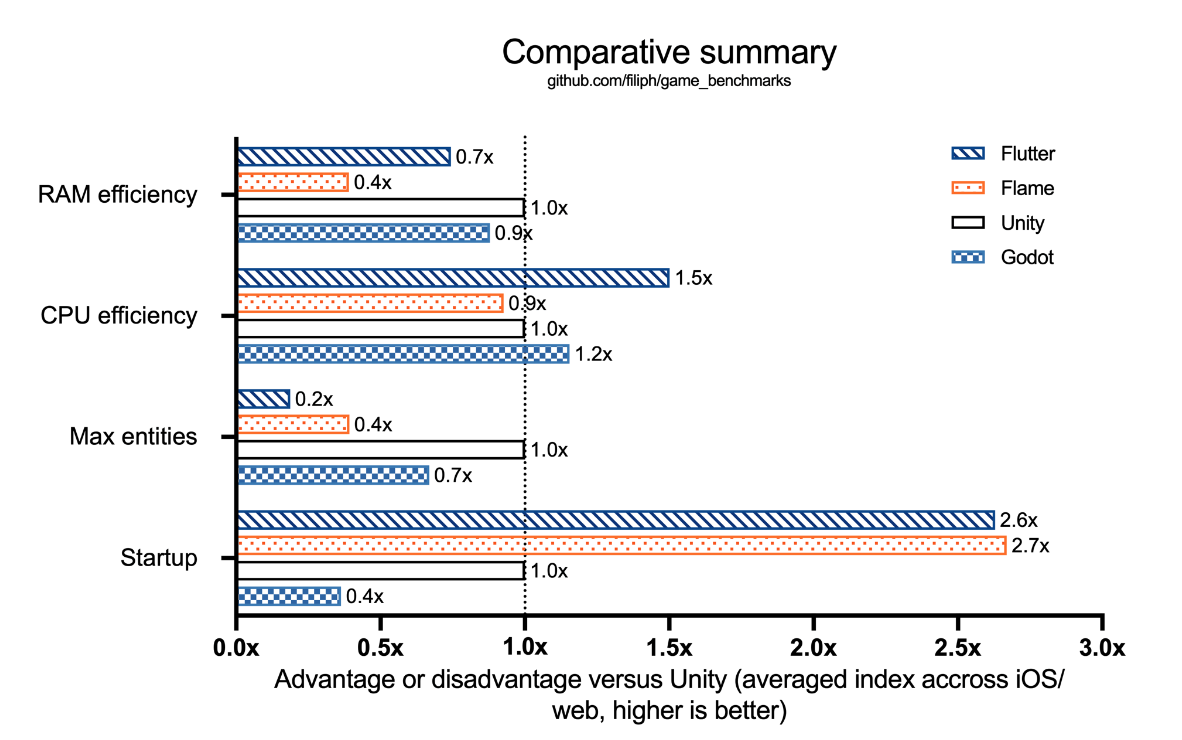
Quando se trata de performance, a principal métrica em jogos são instâncias, no caso da engine Godot, nós. Essa métrica é importante e relevante pois, dependendo da configuração feita, considera o quão eficiente a engine é na renderização de gráficos e também a eficiência na questão de script e funcionalidade, tendo em vista que instanciar novos objetos envolve também qualquer funcionalidade atrelada com o objeto, além de seus visuais. Também é importante verificar métricas de eficiência de uso de recursos como CPU e memória RAM da máquina. Neste tópico, gostaria de trazer dois estudos que encontrei, focando na comparação entre Godot e Unity dos mesmos. O primeiro se trata de uma comparação entre Godot e Unity ao instanciar objetos 2D ou objetos 3D, com os objetos 3D sendo separados entre *quads* - um plano com uma textura -e um modelo mais complexo, no caso deste estudo, um peixe. O outro estudo se trata da coleta de diversas métricas, focando mais no 2D, meu foco sendo as métricas de CPU e RAM.

Como mencionado anteriormente, o primeiro estudo compara as engines Godot e Unity, mas adiciona um bom nível de nuance nessa comparação, incluindo múltiplas linguagens e/ou técnicas que possam impactar as comparações. Do lado da Unity, são consideradas as tecnologias C# e ECS, enquanto do lado da Godot, são consideradas as linguagens GDScript, C# e também o nó *MultiMesh* - o nó *MultiMesh*, que possui variações em 2D e em 3D, se trata de um nó criado com o propósito de facilitar o processo de instanciar diversos objetos idênticos. Considere uma floresta que o desenvolvedor queira popular com árvores, utilizando o *MultiMesh*, o desenvolvedor apenas aponta o nó da árvore original, o nó do ambiente onde a floresta ficará, e assim, o *MultiMesh* popula o ambiente. Este nó também apresenta melhoria de performance para cenários específicos que envolvem diversos nós idênticos (USING MultiMeshInstance3D - Godot Engine (4.4) documentation in English. [*S. I.*], 202-?. Disponível em: https://docs.godotengine.org/en/4.4/tutorials/3d/using\_multi\_mesh\_instance.html). A conclusão encontrada pelo estudo, conforme anxo a seguir, é que, entre as 3 tecnologias, neste cenário específico, *MultiMesh* apresentou a melhor performance do lado da Godot e, de forma geral, no 2D a engine Godot apresenta melhoria de performance perceptível sobre a Unity, enquanto em 3D, tanto para *quads* quanto modelos 3D completos, a diferença de performance é próxima a imperceptível, mas com a Unity levemente sobre a engine Godot. Incluí apenas anexo ao gráfico de *3D Fish*, 3D completo. Os outros 2 gráficos estão disponíveis no repositório do trabalho (COMPARING Game Engine Performance Godot MultiMesh vs Unity ECS - Youtube. [*S. I.*], 2023. Disponível em: https://www.youtube.com/watch?v=6BVoy7kOFxY).



subtexto imagem

Já sobre o segundo estudo, em sua completude compara as tecnologias Flutter, Flame, Unity e Godot para o desenvolvimento de jogos, primariamente, para plataformas Web e Mobile. Embora estas plataformas não sejam o foco deste artigo, a comparação ainda se demonstra relevante, pois o uso de memória e CPU, embora não seja idêntico, se mostra similar entre diferentes plataformas. Focando na Unity e na Godot, os resultados concluem que, em questões de uso de CPU, a engine Godot é levemente mais eficiente, enquanto, para memória RAM, a mesma apresenta um uso maior. Segue, na figura abaixo, o gráfico dos resultados gerados. A Unity foi utilizada como base e seus dados convertidos para o valor 1.0, com os dados de outras tecnologias sendo convertidos com relação a este valor (BENCHMARKING Flutter, Flame, Unity and Godot. [*S. I.*], 2024. Disponível em: https://filiph.net/text/benchmarking-flutter-flame-unity-godot.html).



subtexto imagem

Em conclusão, os estudos mostram resultados satisfatórios. Com relação a 2D, a engine Godot apresenta performance até melhor que a Unity, enquanto, em 3D, com as tecnologias corretas sendo utilizadas, a performance se demonstra próxima a equivalente. Vale constatar, também, que os estudos foram executados em 2023 e 2024, respectivamente, enquanto a engine Godot recebe atualizações constantes, então a performance pode já estar melhor.

1. **Conclusão**

texto

texto

**Referências**

ABOUT - pygame wiki [*S. I.*] 20–. Disponível em: https://www.pygame.org/wiki/about. Acesso em: 31 ago. 2025

AMEREH, Fatemeh. A Study and Implementation of Turn-Based Combat Systems in Role-Playing Games. **Aalto University School of Science**, [*s. l.*], 18 dez. 2024.

ANDERSON, E. F., ENGEL, S., Comninos, P., & McLoughlin, L. **The case for research in game engine architecture**, 2008.

DONOVAN, Tristan. **Replay**: The History of Videogames. [*S. l.*: *s. n.*], 2010.

Features - Godot Engine. [*S. l.*], 202-?. Disponível em: https://godotengine.org/features/. Acesso em: 21 mar. 2025

First Public Release! - Godot Engine. [*S. l.*], 2014. Disponível em: https://godotengine.org/article/first-public-release/. Acesso em: 21 mar. 2025.

GAME Engines - how do they work?. [*S. l.*], 2019?. Disponível em: https://web.archive.org/web/20190720012557/https://unity3d.com/what-is-a-game-engine. Acesso em: 21 mar. 2025.

GODOT Docs - 4.4. [*S. l.*], 2025. Disponível em: https://docs.godotengine.org/en/stable/. Acesso em: 21 mar. 2025.

GODOT Engine - Free and open source 2D and 3D game engine. [*S. l.*], 202-?. Disponível em: https://godotengine.org/. Acesso em: 21 mar. 2025.

LICENSE - Godot Engine [*S. I*], 20–. Disponível em: https://godotengine.org/license/. Acesso em: 30 ago. 2025.

SOBOTA, Branislav; PIETRIKOVÁ, Emília. The role of game engines in game development and teaching. In: **Computer Science for Game Development and Game Development for Computer Science**. IntechOpen, 2023

Toftedahl, Marcus, and Henrik Engström. **A taxonomy of game engines and the tools that drive the industry.** *2019*

UNITY Engine. [*S. l.*], 202-?. Disponível em: https://unity.com/. Acesso em: 21 mar. 2025.

WIZARDS RPG TEAM; WYATT, James; SCHWALB, Robert J; CORDELL, Bruce R. **Dungeons & Dragons: Player‘s Handbook**. [*S. l.*: *s. n.*], 2014.

USING MultiMeshInstance3D - Godot Engine (4.4) documentation in English. [*S. I.*], 202-?. Disponível em: https://docs.godotengine.org/en/4.4/tutorials/3d/using\_multi\_mesh\_instance.html

COMPARING Game Engine Performance Godot MultiMesh vs Unity ECS - Youtube. [*S. I.*], 2023. Disponível em: <https://www.youtube.com/watch?v=6BVoy7kOFxY>

BENCHMARKING Flutter, Flame, Unity and Godot. [*S. I.*], 2024. Disponível em: https://filiph.net/text/benchmarking-flutter-flame-unity-godot.html