ANÁLISE COMPARATIVA DE PERFORMANCE EM SIMULAÇÕES DE COLISÃO 2D: UM ESTUDO DE CASO ENTRE GODOT E UNITY

Gabriel Rodrigues Bittencourt[1], Giácomo Antônio Althoff Bolan[2]

Resumo: A detecção de colisões em tempo real se apresenta como um desafio computacional no desenvolvimento de jogos eletrônicos devido ao seu potencial crescimento quadrático. Este trabalho apresenta uma análise comparativa de performance em simulações 2D entre os motores de jogos Godot e Unity. A metodologia consistiu no desenvolvimento de um protótipo experimental submetido a um estresse controlado de carga progressiva linear, monitorando a taxa de quadros por segundo, o consumo de memória RAM e métricas de microarquitetura da CPU por meio do Intel VTune Profiler. Foram avaliadas três abordagens, sendo elas a Godot Engine, Unity DOTS e a Unity utilizando o modelo tradicional orientado a objetos. Os resultados mostram que, embora as arquiteturas tradicionais orientadas a objetos apresentem maior taxa de quadros em cenário de baixa densidade de entidades em cena, essas mesmas arquiteturas sofrem uma queda drástica em cenários de alta densidade. A análise por meio do VTune Profiler mostra que essa queda se dá, em parte, por causa do gargalo no barramento de memória. Em contrapartida, o ecossistema Unity DOTS demonstrou maior estabilidade na taxa de quadros por segundo devido à redução no gargalo da DRAM, além do maior aproveitamento de múltiplos núcleos de CPU.

Palavras-chave: motor de jogos; análise de performance; detecção de colisões; arquitetura de software.

[1]Curso de Ciência da Computação, Universidade do Extremo Sul Catarinense (Unesc), Criciúma - Santa Catarina - Brasil. vegagabrielvega@unesc.net

[2]Orientador, Curso de Ciência da Computação, Universidade do Extremo Sul Catarinense (Unesc), Criciúma - Santa Catarina - Brasil. kinhobolan@live.com

ABSTRACT: Real-time collision detection presents a significant computational challenge in video game development due to its potential quadratic growth. This work presents a comparative performance analysis of 2D simulations between the Godot and Unity game engines. The methodology consisted of developing experimental prototypes subjected to a controlled stress test of linear progressive load, monitoring the frame rate per second, RAM consumption, and CPU microarchitecture metrics through the Intel® VTune™ Profiler. Three approaches were evaluated: Godot Engine, Unity DOTS, and Unity using the traditional object-oriented model. The results show that while traditional object-oriented architectures present a higher frame rate in low-density scenarios, these same structures suffer a drastic drop in high-density contexts. The analysis through the VTune Profiler shows that this drop occurs, in part, due to a bottleneck in the memory bus. In contrast, the Unity DOTS ecosystem demonstrated greater stability in frames per second due to a reduction in the DRAM bottleneck, as well as better utilization of multiple CPU cores.

Keywords: game engine; performance analysis; collision detection; software architecture.

1 INTRODUÇÃO

O mercado de jogos se encontra em constante crescimento, saindo de 137,9 bilhões de dólares em 2018 (Wijman, 2023a) para 182,9 bilhões de dólares em 2022, com expectativas de chegar até 206,4 bilhões de dólares no ano de 2025 (Wijman, 2023b). Esse mercado aquecido atrai um número crescente de desenvolvedores que buscam criar e distribuir seus produtos. Para auxiliar nesse processo, os motores de jogos são utilizados, pois abstraem tarefas de baixo nível, como renderização gráfica, gerenciamento de recursos e simulação física (Almeida, 2016).

A arquitetura de um motor de jogos é composta por diferentes subsistemas que precisam operar de forma coordenada para manter a estabilidade da aplicação. Entre esses subsistemas, a simulação física exerce papel relevante, pois processa forças, movimentos, contatos e colisões entre entidades do jogo. Segundo Park e Baek (2020), o módulo de física costuma ser organizado de forma separada da renderização, pois a simulação precisa ocorrer em intervalos fixos para reduzir os efeitos da variação da taxa de quadros sobre o comportamento dos objetos.

Tendo em vista que a detecção de colisões é uma operação computacionalmente custosa dentro de uma simulação física, devido ao seu

potencial crescimento quadrático (Jiménez; Thomas; Torras, 2001) e à necessidade de precisão na resposta, torna-se relevante compreender como motores distintos se comportam sob condições equivalentes de estresse. Comparativos técnicos desse tipo podem auxiliar desenvolvedores na tomada de decisões fundamentadas ao escolher uma plataforma para seus projetos.

Porém, cada motor apresenta modos diferentes de realizar a mesma tarefa, e esses modos diferentes impactam a performance e o consumo de recursos do jogo. A escolha de um motor se torna, portanto, uma tarefa que impacta diretamente o produto final que será disponibilizado para o consumidor.

Embora motores de jogos ofereçam recursos semelhantes para o desenvolvimento de aplicações interativas, cada motor utiliza estruturas internas, políticas de atualização e estratégias de processamento distintas. Dessa forma, a escolha de um motor ou de uma arquitetura de implementação não afeta apenas o fluxo de desenvolvimento, mas também a escalabilidade, o consumo de recursos e a taxa de quadros obtida em cenários de maior densidade de entidades. Para avaliar esse impacto, uma das métricas mais utilizadas é a quantidade de quadros por segundo (FPS), pois ela se relaciona diretamente à percepção de fluidez pelo usuário (Claypool; Claypool, 2009; Koulaxidis; Xinogalos, 2022).

A relevância da análise do desempenho dos motores se dá na própria natureza do sistema. De acordo com Li et al. (2022), os jogos se diferenciam de programas tradicionais por serem muito mais interativos, dinâmicos e não-determinísticos, e sua operação é gerida diretamente pelo motor de jogos. Por causa disso, possíveis degradações na performance podem impactar diretamente na experiência do jogador.

Para avaliar o impacto de diferentes abordagens de um jogo, uma das métricas principais a ser avaliada por Koulaxidis e Xinogalos (2022) é a quantidade de quadros por segundo (FPS). Essa métrica se mostra muito relevante, pois Claypool e Claypool (2009) mostra como a experiência do usuário está muito mais ligada a essa métrica do que a outras como, por exemplo, resolução da tela.

Diante desse cenário, este trabalho propõe uma análise comparativa de performance entre os motores Godot e Unity em simulações de colisão 2D, avaliando métricas como uso de CPU, memória e taxa de quadros por segundo em contextos de estresse controlado. O estudo busca oferecer evidências empíricas sobre as capacidades e limites de cada mo-

tor, contribuindo para a comunidade acadêmica e profissional com dados objetivos que auxiliem na compreensão do impacto de suas diferentes arquiteturas de física.

Como objetivos específicos, se apresenta descrever os recursos utilizados pelos motores para o cenário de testes de colisão; modelar testes em cenários de colisão em ambos os motores; e identificar os diferenciais nos testes, apresentando resultados comparativos de performance entre os motores. Esses objetivos orientam a construção dos protótipos, a coleta dos dados e a análise das diferenças observadas entre as arquiteturas avaliadas.

O estudo de motores de jogos e sua performance não é algo inédito na literatura científica e nesta seção, são apresentadas pesquisas que abordaram a comparação entre motores de jogos e avaliação de desenvolvimento e características técnicas.

O estudo técnico de Ezhil (2025) realiza uma análise das características técnicas, modelos de licenciamento e adequação para o desenvolvimento moderno de jogos. O foco do estudo é estudar como diferentes arquiteturas de motores influenciam o desempenho em hardware moderno, com uma seção dedicada à comparação dos sistemas de física, incluindo o Box2D e o PhysX.

A pesquisa compara a implementação técnica dos subsistemas de cada motor. No caso da Godot é analisada a integração do seu motor de física 3D próprio e o uso de bibliotecas leves para 2D, em comparação com a abordagem da Unity que suporta o pacote *Data-Oriented Technology Stack* (DOTS) para simulações de alta performance. O estudo avalia a documentação técnica e a eficiência teórica dos algoritmos de colisão utilizados por cada plataforma.

A análise aponta que a Godot se beneficia de sua natureza de código aberto e leve que oferece boa performance em hardware modesto, especialmente em 2D. Por outro lado, a Unity, através do DOTS e do Physics Package, consegue atingir contagens de objetos massivamente maiores em simulações complexas, mas ao custo de uma complexidade de código mais elevada para o desenvolvedor.

Os autores concluem que a Godot está fechando a lacuna tecnológica em relação aos motores proprietários. Para desenvolvedores independentes e estúdios focados em 2D, a sobrecarga de performance da Unity pode não justificar seus recursos extras, porém sua acessibilidade e suporte multiplataforma se mostram como pontos positivos. O artigo sugere

que a escolha deve basear-se na necessidade de cada projeto e também como as licenças podem afetar o rumo do jogo.

O artigo de Barczak e Woźniak (2019) foca na comparação direta de desempenho entre os motores de jogos Unity, Unreal Engine 4 e CryEngine 5.5 para jogos 3D, com o objetivo de analisar as capacidades técnicas e os fatores que influenciam sua popularidade, além de mostrar a diferença entre cada ferramenta, como seus prós e contras.

Para a análise experimental, foram desenvolvidos protótipos de teste que estressavam componentes específicos dos motores, como o sistema de partículas e a renderização de múltiplos objetos. A coleta de dados focou na quantidade de quadros gerados por cada jogo criado no seu específico motor, fazendo a variação da qualidade gráfica, quantidade de objetos em cena e resolução.

O estudo mostra que a Unity tende a apresentar uma taxa de quadros média superior aos outros motores, porém com a desvantagem de apresentar gráficos inferiores comparados com a Unreal Engine e a CryEngine. Por fim, por ser um trabalho focado em motores para jogos 3D, fica a lacuna no quesito de jogos 2D, que o presente trabalho procura preencher.

No artugi de Bayliss (2022), se fala sobre a mudança de paradigma no desenvolvimento de alta performance, focando no Design Orientado a Dados (DOD). O trabalho mostra que o DOD surge da necessidade de otimização em hardwares mais complexos e a necessidade de execução consistente em tempo real.

A autora utiliza exemplos teóricos, como a simulação *Autofarmers* e problemas de transformação de imagem, para ilustrar como a organização dos dados impacta a eficiência. Bayliss (2022) apresenta a implementação de DOD através do DOTS pela Unity. A análise técnica foca em como componentes do DOTS permitem que desenvolvedores utilizem melhor o cache da CPU e o processamento de vários núcleos.

Os resultados apresentados no artigo demonstram que a adoção do DOD, exemplificada pelo Unity DOTS, facilita o paralelismo de dados e elimina condições de corrida comuns em sistemas orientados a objetos tradicionais. O estudo aponta que ao estruturar o software com foco nos dados e na hierarquia de memória, é possível que simulações complexas rodem com taxas de quadros elevadas.

O presente artigo está estruturado em quatro seções. A introdução apresenta a contextualização e definição do problema, além da justificativa da pesquisa. Após, na seção de materiais e métodos são apre-

sentados os procedimentos que levaram ao desenvolvimento dos casos de testes para a coleta dos dados de performance. Em seguida, a seção de resultados e discussão apresenta os resultados obtidos. Por fim, na seção de conclusão são discutidos os resultados e são propostos possíveis caminhos para o desenvolvimento de futuros estudos.

2 MATERIAIS E MÉTODOS

O presente trabalho consistiu no desenvolvimento de três programas de teste, com o objetivo de comparar o desempenho de dois motores de jogos e suas respectivas implementações de motores físicos. Os programas foram feitos utilizando Godot Engine, a Unity utilizando o motor Box2D, padrão para jogos 2D, e a Unity utilizando a arquitetura *Data-Oriented Technology Stack* (DOTS).

A arquitetura de um motor de jogos requer uma divisão específica de tarefas para evitar a degradação da experiência do jogador. Segundo Park e Baek (2020), o módulo de física, que é responsável por processar as forças nas entidades e pela detecção de colisão, é executado de forma separada do módulo principal de renderização visual. Para que haja estabilidade na simulação, essas operações físicas ocorrem em intervalos de tempo fixos predeterminados, diminuindo o possível impacto que a flutuação de FPS possa causar.

Os protótipos foram desenvolvidos e testados em um ambiente controlado e buscando executar processamento equivalente entre cada uma das implementações para que o que fosse considerada seja apenas a diferença entre como cada um dos motores implementa as simulações físicas de colisão.

2.1 AMBIENTE DE DESENVOLVIMENTO E TESTES

O ambiente de testes foi configurado de forma a diminuir a interferência de variáveis externas e garantir que os experimentos possam ser reproduzidos futuramente em ambientes similares. O computador utilizado para o desenvolvimento dos programas e a execução dos testes, bem como as ferramentas de software utilizadas estão disponíveis, respectivamente, na Tabela 1 e Tabela 2.

Para garantir consistência dos dados, todas as aplicações foram executadas uma vez sem a obtenção dos dados para que a temperatura do processador se estabilizasse e se mantivesse constante durante toda a execução dos testes e não houvesse viés positivo nos primeiros testes.

Tabela 1 - Especificações do Hardware Utilizado

Componente	Especificação Técnica
Processador (CPU)	Intel Core i5-8400
Memória RAM	16GB DDR4
Placa de vídeo (GPU)	NVidia GeForce RTX 2060
Armazenamento	SSD Sata 2TB
Sistema Operacional	Linux Mint 22.2

Fonte: Elaborado pelo autor.

Tabela 2 - Ferramentas e Softwares Utilizados

Ferramenta	Versão	Finalidade
Godot Engine	4.4	Desenvolvimento do protótipo
Unity	2022.3.62f3	Desenvolvimento dos protótipos 2D Tradicional e DOTS
Intel VTune Profiler	2024.2	Monitoramento e amostragem do uso da CPU
Editor de Código	Visual Studio Code	Ambiente de desenvolvimento

Fonte: Elaborado pelo autor.

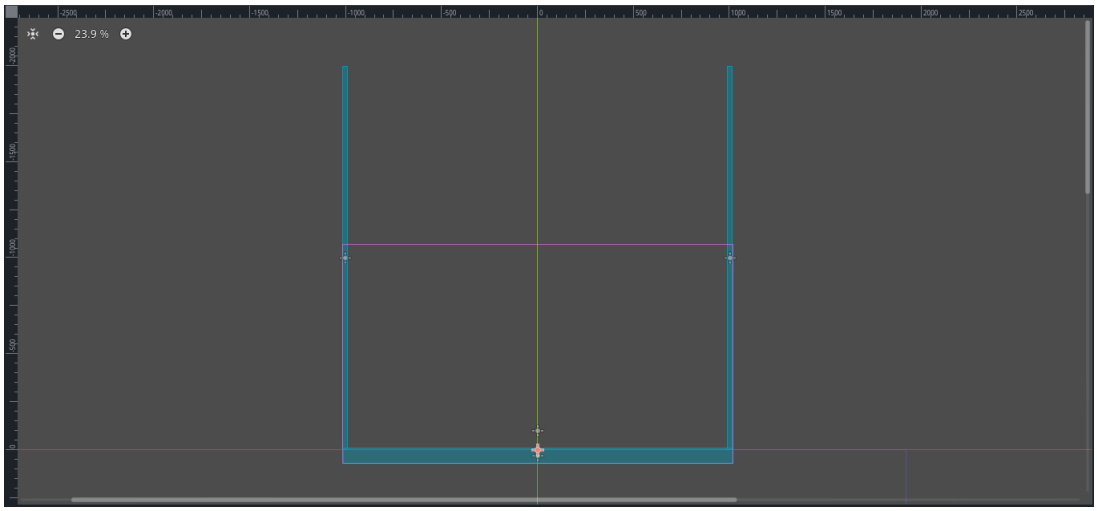
2.2 ESTRUTURA DOS PROTÓTIPOS E IMPLEMENTAÇÃO

Nos três protótipos desenvolvidos para a análise foi implementado um cenário de carga de processamento progressivo idêntico. Cada protótipo possui dois algoritmos. Um onde o algoritmo projetado instancia dez entidades simultaneamente e espera quatro segundos para dar tempo de que as entidades interajam com o ambiente e realizar os cálculos necessários antes de instanciar mais dez entidades, até o número de 5000 entidades, e outro algoritmo que faz o uso do *profiler* de cada motor para a aquisição dos dados estatísticos.

Para a Godot foi utilizada diretamente a API de baixo nível *PhysicsServer2D* para construir somente o necessário do objeto para a simulação física. Com a Unity 2D foi utilizado o padrão clássico de orientação a objetos, enquanto com o protótipo usando DOTS, foram utilizados os pacotes *Entities*, *Burst Compiler* e *Job System*.

Em todos os protótipos foi utilizado um *spawner*, que instancia as entidades físicas e três colisores que formam uma caixa com o topo aberto para que as entidades tenham um ambiente onde possam interagir entre si, porém, como há somente implementação 3D para o DOTS na Unity, foi necessário criar mais um script que faz a trava do eixo Z para que as entidades dessa implementação se comportassem de maneira similar às entidades presentes na implementação da Godot e da Unity utilizando o padrão 2D.

Figura 1 - Exemplo do ambiente de teste na Godot



Fonte: Elaborado pelo autor.

2.3 MÉTRICAS DE DESEMPENHO E METODOLOGIA DE COLETA

A comparação entre as abordagens se baseou na coleta de quatro métricas de desempenho do computador. A extração dos dados foi dividida entre um script que recebia a telemetria providenciada já pelo próprio motor e uma ferramenta de *profiling* do processador.

Para os testes, cada programa foi compilado para Linux e inicializado usando o Intel VTune Profiler, programa desenvolvido pela própria Intel, para o monitoramento do processador. Cada programa foi rodado por, em média, trinta minutos, e o script de telemetria armazenava os dados em um arquivo CSV a cada meio segundo. Esse processo foi repetido cinco vezes para cada abordagem e foi utilizada uma média aritmética dos resultados obtidos para que haja a representação mais fiel possível dos resultados.

3 RESULTADOS E DISCUSSÃO

Os resultados dessa pesquisa se originaram da análise comparativa entre os protótipos desenvolvidos na Godot Engine e na Unity, utilizando tanto o modelo tradicional 2D orientado a objeto, quanto o ecossistema DOTS, que utiliza *Entity Component System* (ECS) fazendo uso de componentes de dados contínuos em memória. O processo foi avaliado com uma carga linear de 0 a 5.000 entidades, gerando dados a cada meio segundo, mostrando as diferenças entre as arquiteturas utilizadas.

3.1 ANÁLISE DA TAXA DE QUADROS

A taxa de quadros por segundo (FPS) foi monitorada para verificar o possível conforto visual de cada abordagem em cenários de maior densidade de elementos em cena. A Tabela 3 mostra as médias aritméticas de FPS separadas em faixas de quantidades de objetos.

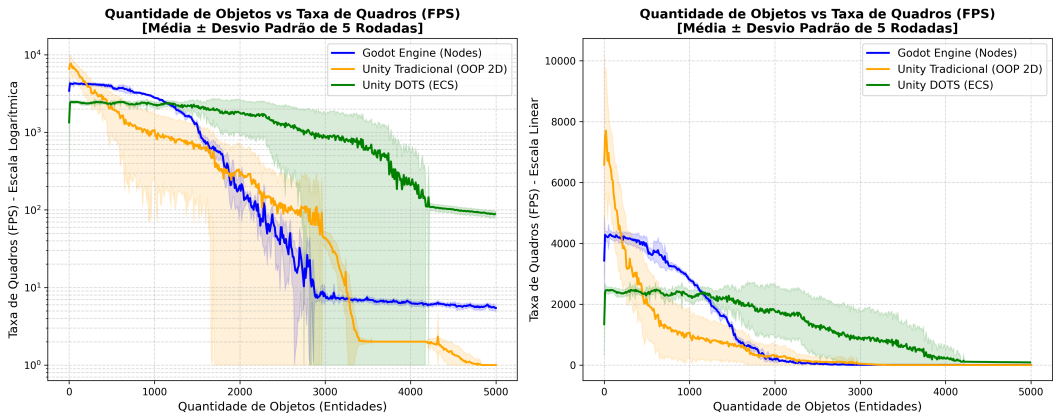
Tabela 3 - Média de FPS por Faixa de Quantidade de Objetos

Abordagem / Motor	0 a 100	101 a 1.000	1.001 a 2.000	2.001 a 3.000	3.001 a 4.000	4.001 a 5.000
Unity Tradicional (OOP 2D)	7.792,72	2.658,24	126,41	64,12	5,05	1,32
Godot Engine (Nodes)	4.157,71	3.651,33	1.233,94	54,29	6,74	6,06
Unity DOTS (ECS)	2.408,53	2.427,95	2.378,98	2.344,92	2.247,20	2.118,17

Fonte: Elaborado pelo autor.

Os resultados apresentam comportamentos distintos conforme a quantidade de objetos no cenário aumenta. Em cenários de baixa densidade de elementos, entre 0 e 100 objetos, as abordagens clássicas de 2D, tanto na Unity quanto na Godot apresentaram uma média maior de FPS, o que é de se esperar, considerando que ambientes 2D apresentam uma menor complexidade em relação a ambientes 3D.

Figura 2 - Relação entre a quantidade de objetos e a taxa de quadros



Fonte: Elaborado pelo autor.

Porém, como visto na Figura 2, as atuais soluções 2D apresentam uma queda de performance muito mais aparente em comparação ao DOTS da Unity em cenários com maior densidade de elementos. A partir de 1000 entidades em cena, ambas as implementações 2D obtiveram resultados similares à implementação 3D e a partir de 3.000 já apresentavam uma ordem de magnitude a menos FPS.

Apesar desses dados, as aplicações feitas em Unity apresentam maior estabilidade de FPS, mesmo com números inferiores aos da Godot. Isso corrobora com o que é mostrado em Barczak e Woźniak (2019) onde

a aplicação, mesmo estando em 3D ao invés de 2D, demonstra uma menor queda na taxa de FPS em comparação com outros motores como a Unreal Engine e a CryEngine.

Possíveis explicações para esse comportamento, como demonstrado na tabela 4, são que a arquitetura DOTS é capaz de utilizar de forma mais eficiente as tarefas em paralelo, além de saturar a largura de banda de leitura da memória RAM com maior sucesso e ter uma capacidade maior de fazer uso do *Hardware Prefetcher*, que antecipa o comportamento do programa e traz para o cachê do processador os próximos dados da memória RAM a serem acessados.

Tabela 4 - Métricas Top-Down do VTune por abordagem

Métrica	Unity Tradicional (OOP 2D)	Godot Engine	Unity DOTS (ECS)
Retiring (%)	30,07 ± 0,19	21,89 ± 0,18	31,69 ± 0,13
Front-End Bound (%)	16,97 ± 0,20	22,49 ± 0,19	27,40 ± 0,12
Bad Speculation (%)	3,20 ± 0,06	18,29 ± 0,19	9,44 ± 0,08
Back-End Bound (%)	49,80 ± 0,29	37,39 ± 0,50	31,44 ± 0,14
Memory Bound (%)	28,99 ± 0,32	19,31 ± 0,47	12,16 ± 0,08
DRAM Bound (%)	19,80 ± 0,58	8,24 ± 0,97	4,01 ± 0,15
Largura de banda da memória (%)	24,57 ± 0,39	16,10 ± 0,90	6,79 ± 0,16
Latência da memória (%)	28,01 ± 0,16	23,93 ± 0,29	20,59 ± 0,20
Core Bound (%)	20,83 ± 0,10	18,07 ± 0,11	19,29 ± 0,07

Fonte: Elaborado pelo autor.

No monitoramento da memória RAM se mostra que o ecossistema da Godot utiliza uma quantidade menor de memória como um todo, como demonstrado na tabela 5, em comparação com as aplicações feitas na Unity, porém foi o que teve o maior aumento relativo com 53,56% de aumento em comparação com os 30,88% da Unity 2D e 12,29% da Unity com DOTS.

Tabela 5 - Média de Consumo de RAM (MB) por Faixa de Objetos

Abordagem / Motor	0 a 100	101 a 1.000	1.001 a 2.000	2.001 a 3.000	3.001 a 4.000	4.001 a 5.000
Unity Tradicional (OOP 2D)	162,94	168,90	179,71	190,49	201,26	213,26
Godot Engine	40,92	42,41	46,66	52,01	57,19	62,84
Unity DOTS	349,26	363,58	368,59	375,72	387,50	392,20

Fonte: Elaborado pelo autor.

Com os dados extraídos pelo VTune Profiler foi possível obter uma visão mais detalhada das operações que acontecem no processador. O modelo tradicional da Unity apresentou o maior índice de estancamento no *back-end* do processador, com 49,80% de *Back-End Bound*. Isso se deve, em grande parte, por causa do impacto da *Memory Bound* de 28,99%, impulsionado por um gargalo de *DRAM Bound* de 19,80%. Isso

mostra que o processador está gastando ciclos de *clock* ociosos enquanto busca os dados diretamente da memória RAM.

Na Godot o que foi apresentado foi a alta porcentagem de *Bad Speculation* com 18,29%, que são previsões incorretas do processador. Essas previsões incorretas também acabam desperdiçando ciclos do processador. Os resultados da Tabela 6 mostram que a Godot, assim como a Unity 2D, apresentam baixo índice de paralelismo, com o uso de apenas 1,08 núcleos lógicos na Godot e 1,46 núcleos lógicos na Unity 2D.

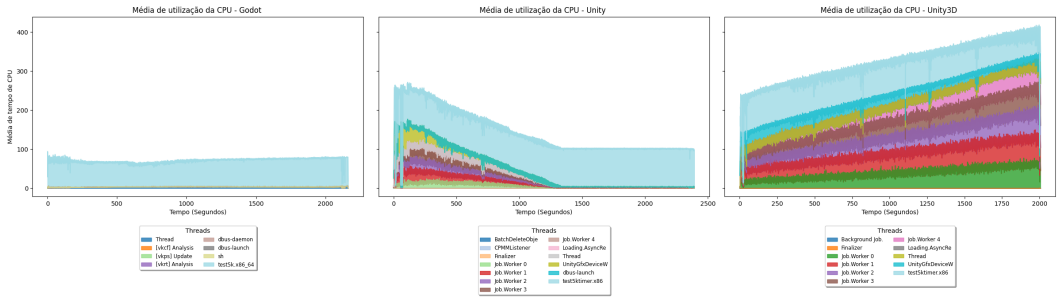
Tabela 6 - Métricas gerais do VTune por abordagem

Métrica	Unity Tradicional (OOP 2D)	Godot Engine	Unity DOTS (ECS)
Tempo decorrido (s)	2.374,21 ± 16,38	2.131,53 ± 17,47	2.003,20 ± 2,36
CPUs lógicos utilizados	1,46 ± 0,01	1,08 ± 0,00	3,22 ± 0,04
Utilização efetiva da CPU (%)	24,27 ± 0,21	17,96 ± 0,05	53,69 ± 0,69
CPI	0,829 ± 0,006	1,152 ± 0,009	0,833 ± 0,003
Instruções aposentadas ($\times 10^{12}$)	15,85 ± 0,17	7,58 ± 0,01	29,44 ± 0,29
Clockticks ($\times 10^{12}$)	13,13 ± 0,06	8,73 ± 0,07	24,52 ± 0,31
Total de threads	47,4 ± 0,5	30,4 ± 2,1	50,7 ± 0,5

Fonte: Elaborado pelo autor.

A Unity com a arquitetura DOTS foi a que apresentou melhor aproveitamento da CPU, com o maior índice de Retiring com 31,69%, que é o trabalho útil feito pelo processador, além de ter o menor índice de *DRAM Bound* com 4,01%. Como apresentado no trabalho de (Asri et al., 2021), aplicações que operam em regime de alto *Memory Bound* sofrem penalidades de desempenho devido à latência de transferência entre a CPU e a DRAM externa, fazendo com que o barramento físico seja o principal teto limitador da aplicação. Também é possível ver que a aplicação com a arquitetura DOTS alcançou uma utilização efetiva da CPU de 53,69%, utilizando em média 3,22 núcleos lógicos concorrentes.

Figura 3 - Média do uso de threads em cada motor



Fonte: Elaborado pelo autor.

Esses achados corroboram o que é mostrado em Bayliss (2022), que fala que *Data-Oriented Design* (DOD), que é a base do DOTS, utiliza

Structure of Array que é capaz de usar mais eficientemente o cachê do processador, além de fazer uso de *job systems* que distribuem o trabalho em várias *threads*, como mostrado na Figura 3, em comparação com os modelos mais tradicionais.

Esses resultados também corroboram com a análise de Ezhil (2025), que aponta teoricamente que a Godot se mostra como um motor mais leve e otimizado. Essa característica é validada empiricamente por meio do monitoramento da memória RAM e CPU, onde o ecossistema Godot apresentou uma demanda de recursos mais baixa se comparado com a Unity tradicional e com o ecossistema Unity DOTS quando colocados sob as mesmas condições.

4 CONCLUSÃO

O presente trabalho teve como objetivo comparar a escalabilidade e eficiência de motores de jogos Godot e Unity, que usam diferentes arquiteturas para seus motores de física, buscando avaliar o comportamento do *hardware* e *software* com foco nas simulações físicas de colisão em ambientes 2D. A pesquisa contemplou o levantamento de dados a partir de ferramentas nativas dos motores e também da ferramenta Intel VTune Profiler.

O resultado observado mostra que existe uma grande diferença entre, não necessariamente os motores de jogos em si, mas nos paradigmas utilizados nos processamentos dos dados. Tanto na Godot quanto na Unity é utilizada a orientação a objeto, que apresentam uma performance visual de até três ordens de magnitude a menos em comparação com a arquitetura DOTS, que é orientada a dados. Isso mostra que a queda na performance não é causada pelo algoritmo de detecção de colisão em si, mas sim pela forma como diferentes arquiteturas interagem com o processador e sua microarquitetura.

De modo geral, os resultados mostram que a escolha da arquitetura tem impacto direto na escalabilidade da simulação física. Embora abordagens orientadas a objeto tenham apresentado desempenhos que seriam considerados satisfatórios em cenários com baixa quantidade de entidades, a abordagem voltada a dados, utilizada pelo DOTS, se mostrou mais estável ao longo dos testes, mostrando uma queda menor na taxa de quadros e melhor aproveitamento dos recursos da CPU, apesar do maior uso de memória RAM.

Esses resultados foram possíveis graças ao uso da ferramenta

Intel VTune Profiler, que permitiu preencher uma lacuna na literatura de jogos 2D. O estudo mostra empiricamente a causa da diminuição de performance em abordagens orientadas a objeto, mostrando como o sistema é obrigado a trabalhar em regime de alto *memory bound*, que força a CPU a desperdiçar ciclos ociosos enquanto aguarda a busca de dados em RAM, além de mostrar a superioridade da orientação a dados e como o DOTS é capaz de utilizar de forma mais eficiente os diferentes núcleos da CPU.

Apesar dos resultados, as conclusões obtidas são apresentadas de maneira isolada e não representam a realidade de um ambiente de produção onde o motor físico compete com sistemas complexos pelos recursos da CPU, como inteligência artificial de personagens, renderização de interface e gerenciamento de áudio.

Para trabalhos futuros, recomenda-se a realização dos testes em diferentes máquinas e a avaliação de como a mudança de *hardwares* específicos, como a memória RAM com diferentes larguras de banda e processadores com diferentes números de núcleos, em específico para a arquitetura DOTS. Sugere-se ainda a expansão dessa pesquisa para outros motores de jogos ou de ampliar para ambientes em 3D. Como mencionado anteriormente, testes em cenários não isolados também são uma sugestão para trabalhos futuros.

DECLARAÇÃO DE USO DE INTELIGÊNCIA ARTIFICIAL

Os autores declaram que ferramentas de Inteligência Artificial generativa foram utilizadas exclusivamente como apoio auxiliar à elaboração deste trabalho, incluindo assistência na revisão linguística, estruturação textual e organização de ideias. Nenhuma ferramenta de IA foi utilizada para substituição da autoria intelectual, análise científica, interpretação dos resultados ou tomada de decisões acadêmicas. A responsabilidade integral pelo conteúdo, originalidade, veracidade das informações e conformidade com os princípios de integridade científica permanece integralmente atribuída aos autores.

REFERÊNCIAS

ALMEIDA, M. S. O. **Design de jogos baseado em componentes**. Tese (Doutorado) — Universidade de São Paulo, 2016.

ASRI, M. et al. **Hardware accelerator integration tradeoffs for high-performance computing: A case study of gemm acceleration in**

n-body methods. IEEE Transactions on Parallel and Distributed Systems, IEEE, v. 32, n. 8, p. 2035–2048. 2021.

BARCZAK, A.; WOŹNIAK, H. **Comparative study on game engines**. Studia Informatica. System and information technology, v. 23, n. 1-2, p. 5–24. 2019.

BAYLISS, J. D. **The data-oriented design process for game development**. Computer, IEEE, v. 55, n. 5, p. 31–38. 2022.

CLAYPOOL, M.; CLAYPOOL, K. **Perspectives, frame rates and resolutions: it's all in the game**. In: **Proceedings of the 4th International Conference on Foundations of Digital Games**. [S.l.: s.n.], 2009. 2009. p. 42–49.

EZHIL, S. L. **A comparative analysis of unity, unreal engine, godot, and cryengine: Technical features, licensing models, and suitability for modern game development**. Journal of Game Theory. 2025.

JIMÉNEZ, P.; THOMAS, F.; TORRAS, C. **3d collision detection: a survey**. Computers & Graphics, Elsevier, v. 25, n. 2, p. 269–285. 2001.

KOULAXIDIS, G.; XINOGALOS, S. **Improving mobile game performance with basic optimization techniques in unity**. Modelling, MDPI, v. 3, n. 2, p. 201–223. 2022.

LI, Z. et al. **Gbgallery: A benchmark and framework for game testing**. Empirical Software Engineering, Springer, v. 27, n. 6, p. 140. 2022.

PARK, H.; BAEK, N. **Developing an open-source lightweight game engine with dnn support**. Electronics, MDPI, v. 9, n. 9, p. 1421. 2020.

WIJMAN, T. **Newzoo's 2018 Report: Insights Into the $137.9 Billion Global Games Market | Newzoo**. 2023. Disponível em: <https://newzoo.com/resources/blog/newzoos-2018-report-insights-into-the-137-9-billion-global-games-market>. Acesso em: 30 de out. de 2025.

WIJMAN, T. **Newzoo's video games market size estimates and forecasts for 2022**. 2023. Disponível em: <https://newzoo.com/resources/blog/the-latest-games-market-size-estimates-and-forecasts>. Acesso em: 30 de out. de 2025.