UNIVERSIDADE ESTADUAL DA PARAÍBA
CAMPUS VII - PATOS
CENTRO DE CIÊNCIAS EXATAS E SOCIAIS APLICADAS
CURSO DE GRADUAÇÃO EM BACHARELADO EM COMPUTAÇÃO

FÁBIO OLIVEIRA DA SILVA LIMA

TÉCNICAS DE ANIMAÇÃO PROCEDURAL EM JOGOS DIGITAIS 2D: REDUÇÃO DE CUSTOS E AUMENTO DA DINAMICIDADE

PATOS - PB
2025

FÁBIO OLIVEIRA DA SILVA LIMA

TÉCNICAS DE ANIMAÇÃO PROCEDURAL EM JOGOS DIGITAIS 2D: REDUÇÃO DE CUSTOS E AUMENTO DA DINAMICIDADE

Trabalho de Conclusão de Curso apresentado ao Curso de Bacharelado em Ciência da Computação do Centro de Ciências Exatas e Sociais Aplicadas da Universidade Estadual da Paraíba, em cumprimento à exigência para obtenção do grau de Bacharel em Ciência da Computação.

Orientador: Harllem Alves do Nascimento

PATOS - PB
2025

L732t Lima, Fábio Oliveira da Silva.

Técnicas de animação procedural em jogos digitais 2D [manuscrito] : redução de custos e aumento da dinamicidade / Fábio Oliveira da Silva Lima. - 2025.

98 f.

Digitado.

Trabalho de Conclusão de Curso (Graduação em Ciência da computação) - Universidade Estadual da Paraíba, Centro de Ciências Exatas e Sociais Aplicadas, 2025.

"Orientação : Prof. Grad. Harllem Alves do Nascimento, Coordenação do Curso de Computação - CCEA".

1. Animação procedural. 2. Animação tradicional. 3. Jogos de plataforma 2d. 4. Cinemática inversa. 5. Geração procedural de conteúdo. I. Título

21. ed. CDD 004

Elaborada por Kaliane Eveny Martins de Oliveira - CRB - 15/986 BSC7

FÁBIO OLIVEIRA DA SILVA LIMA

TÉCNICAS DE ANIMAÇÃO PROCEDURAL EM JOGOS DIGITAIS 2D: REDUÇÃO DE CUSTOS E AUMENTO DA DINAMICIDADE

Trabalho de Conclusão de Curso apresentado à Coordenação do Curso de Ciência da Computação da Universidade Estadual da Paraíba, como requisito parcial à obtenção do título de Bacharel em Ciência da Computação

Aprovada em: 05/06/2025.

BANCA EXAMINADORA

Documento assinado eletronicamente por:

- **Vinícius Augustus Alves Gomes** (***.754.334-**), em **13/06/2025 20:25:20** com chave **ac20811a48ad11f08eaf06adb0a3afce**.
- **Pablo Ribeiro Suárez** (***.806.354-**), em **15/06/2025 08:21:56** com chave **f23e23e049da11f088201a7cc27eb1f9**.
- **Harllem Alves do Nascimento** (***.796.924-**), em **13/06/2025 19:30:51** com chave **0ff9549448a611f0850206adb0a3afce**.

Documento emitido pelo SUAP. Para comprovar sua autenticidade, faça a leitura do QrCode ao lado ou acesse https://suap.uepb.edu.br/comum/autenticar_documento/ e informe os dados a seguir.
Tipo de Documento: Folha de Aprovação do Projeto Final
Data da Emissão: 15/06/2025
Código de Autenticação: cb2553

Dedico este trabalho aos meus pais, meus familiares, meus amigos, meus professores e a todos que estiveram ao meu lado nessa jornada. Serei eternamente grato a todos vocês.

AGRADECIMENTOS

Agradeço, em primeiro lugar, aos meus pais, pelo esforço incansável e pelo apoio incondicional ao longo de toda a minha trajetória. Sem o suporte e dedicação deles, não teria sido possível chegar até aqui.

Sou grato também a todos os familiares e amigos que estiveram ao meu lado. O incentivo, as palavras de apoio e a presença constante foram fundamentais para que eu encontrasse forças para continuar, me reerguer nos momentos difíceis e seguir em frente.

Aos meus professores, deixo meu sincero agradecimento pelas valiosas lições transmitidas. Levarei cada ensinamento comigo por toda a vida, com admiração e respeito, lembrando sempre da importância que cada um teve na minha formação.

Agradeço ao meu orientador Harllem por todo o empenho e dedicação, sempre me auxiliando com orientações valiosas e *feedbacks* construtivos que foram essenciais para o desenvolvimento deste trabalho.

Por fim, agradeço aos meus colegas de faculdade, que compartilharam comigo essa jornada, colaborando, apoiando e tornando cada etapa mais leve e significativa. Obrigado por estarem ao meu lado.

“Quando o mundo te empurra, você só tem que se levantar e empurrar de volta. Ninguém vai te salvar se você começar a murmurar desculpas."

Eiichiro Oda

RESUMO

A indústria de jogos eletrônicos tem apresentado crescimento contínuo nos últimos anos, com aumentos significativos na arrecadação de capital. Contudo, esse crescimento também eleva os custos de desenvolvimento, impulsionado pela alta competitividade do mercado e pelos exigentes critérios de qualidade que os jogos precisam atender para se destacarem. Nesse cenário, pequenos estúdios com recursos limitados enfrentam dificuldades para competir no mercado de jogos. Uma das estratégias utilizadas para contornar essas limitações é o uso de geração procedural de conteúdo, que possibilita a criação automatizada de elementos do jogo, reduzindo a necessidade de produção manual por parte dos desenvolvedores. Entre os conteúdos que podem ser gerados dessa forma estão as animações procedurais, que permitem movimentações dinâmicas e automatizadas para personagens e objetos do jogo, sem a necessidade de desenhar manualmente cada quadro. Apesar de seu grande potencial, esse tipo de animação ainda é pouco explorado em jogos 2D, sendo comum que desenvolvedores independentes optem por animações tradicionais. Este trabalho teve como objetivo apresentar e comparar técnicas de animação tradicional e procedural, por meio do desenvolvimento de elementos de jogo com ambas as abordagens. Foram criados, por exemplo, uma entidade inimiga (aranha) com movimentação dinâmica, e elementos ambientais como água, fogo e tecido. Em seguida, foram realizados testes de estresse para avaliar o uso de recursos computacionais em diversas máquinas e dispositivos *mobiles* de gerações variadas, além de uma análise comparativa quanto à fluidez visual e à carga de trabalho exigida dos desenvolvedores. Com os dados de desempenho coletados, foi realizada uma comparação direta utilizando a métrica de quadros por segundo para cada animação — tanto tradicional quanto procedural — aplicada a cada elemento, em cada um dos dispositivos testados. Os resultados mostraram que as animações procedurais apresentam bom desempenho em dispositivos com maior capacidade computacional e com quantidade de instâncias reduzidas, enquanto equipamentos mais antigos ou com *hardware* limitado enfrentam dificuldades para processá-las adequadamente. Por outro lado, as animações tradicionais, com menor funcionalidade e dinamismo, funcionaram de maneira satisfatória na maioria dos dispositivos, inclusive os mais obsoletos. Concluiu-se que, embora as animações procedurais sejam tecnicamente mais complexas de implementar, elas oferecem maior fluidez, dinamicidade e realismo, além de facilitarem ajustes finos e refatorações ao longo do desenvolvimento. No entanto, essas vantagens vêm acompanhadas de um maior consumo de recursos computacionais. Já as animações tradicionais, elas se mostraram mais leves para o sistema e com maior liberdade artística, mas demonstraram uma baixa faixa de dinamismo em ambientes de jogos no qual o terreno não é plano, além de demandar mais tempo para refatorações, comprometendo a agilidade no processo de desenvolvimento.

Palavras-chave: animação procedural; animação tradicional; jogos de plataforma 2d; cinemática inversa e geração procedural de conteúdo.

ABSTRACT

The video game industry has experienced continuous growth in recent years, with significant increases in revenue. However, this growth also raises development costs, driven by the highly competitive market and the demanding quality standards that games must meet to stand out. In this context, small studios with limited resources face challenges in keeping up. One of the strategies used to overcome these limitations is the use of procedural content generation, which enables the automated creation of game elements, reducing the need for manual production by developers. Among the elements that can be generated by this method are procedural animations, which allow dynamic and automated movements for characters and objects in the game, without the need to manually draw each frame. Despite their great potential, this type of animation is still underutilized in 2D games, with many independent developers opting for traditional animations instead. This work aimed to present and compare traditional and procedural animation techniques through the development of game elements using both approaches. For example, a dynamic enemy entity (a spider) was created, as well as environmental elements such as water, fire, and cloth. Stress tests were then carried out to evaluate the computational resource usage on various machines and mobile devices from different generations, along with a comparative analysis regarding visual smoothness and developer workload. With the collected performance data, a direct comparison was made using the frames-per-second metric for each type of animation — both traditional and procedural — applied to each element on each tested device. The results showed that procedural animations perform well on devices with higher computational capacity and with a reduced number of instances, while older equipment or hardware-limited systems struggle to process them adequately. On the other hand, traditional animations — with less functionality and dynamism — performed satisfactorily on most devices, including outdated ones. It was concluded that although procedural animations are technically more complex to implement, they offer greater fluidity, dynamism, and realism, and they facilitate fine-tuning and refactoring throughout development. However, these advantages come at the cost of higher computational resource consumption. Traditional animations, while lighter on the system and allowing for greater artistic freedom, showed a lower degree of dynamism in game environments with non-flat terrain and required more time for refactoring, compromising development agility.

Keywords: procedural animation; traditional animation; 2d platformer games; inverse kinematics; procedural content generation.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

PCG	*Procedural Content Generation*
2D	Duas Dimensões
3D	Três Dimensões
GDD	*Game Design Document*
CPU	*Central Processing Unit*
GPU	*Graphics Processing Unit*
IK	*Inverse Kinematics*
FK	*Forward Kinematics*
NPCs	*Non-Playable Characters*
FPS	*Frame Per Second*
IA	Inteligência Artificial
L1	Nível 1 de Estresse
L2	Nível 2 de Estresse
L3	Nível 3 de Estresse

SUMÁRIO

1 INTRODUÇÃO

A cada ano, os jogos eletrônicos (popularmente conhecidos como *video games*) têm se tornado uma parte cada vez mais presente na vida cotidiana das pessoas. Em busca de diversão, alívio do estresse e momentos de descontração, muitas pessoas recorrem a essa forma de arte digital. Consequentemente, a indústria global tem crescido continuamente, atendendo a uma demanda crescente por produtos e experiências interativas.

Os *video games* foram criados com o propósito central de divertir. O primeiro deles, desenvolvido em 1958 pelo físico nuclear William Higginbotham, foi uma simulação interativa de uma partida de tênis, exibida em um osciloscópio para entreter os visitantes do Laboratório Nacional de Brookhaven. Conhecido como "Tennis For Two", esse pioneiro projeto marcou o início do papel dos *video games* como uma importante fonte de entretenimento humano, alimentando as necessidades lúdicas da sociedade desde então (Oliveira, 2020).

Figura 1 – Uma recriação do jogo original como comemoração ao aniversário de 50 anos

Fonte: Imagem retirada do site da Brookhaven (2024)

De acordo com Statista (2024) a indústria de jogos digitais arrecadou mais de 406 bilhões de dólares apenas no ano de 2023, com projeções de crescimento contínuo. Estima-se que, até 2029, esse valor alcance aproximadamente 666 bilhões de dólares, quase o triplo dos 265 bilhões arrecadados em 2019, demonstrando o impacto e a relevância crescente dos jogos eletrônicos no mercado global.

Apesar do impressionante crescimento da arrecadação na indústria de jogos digitais, os custos de desenvolvimento também aumentaram significativamente. Criar um jogo envolve uma equipe multidisciplinar, composta por programadores, artistas, *designers* de som, entre outros profissionais, todos os quais precisam ser remunerados (Hendrikx *et al.*, 2013). Além disso, existem outros fatores que contribuem para o investimento elevado para lançar um jogo no mercado (Digital, 2021).

Segundo Digital (2021), os jogos digitais podem ser classificados em três categorias

distintas: Jogos *indies*, geralmente desenvolvidos por uma ou poucas pessoas; Jogos AAA, criados por grandes estúdios e que envolvem centenas de profissionais; E jogos III, que ocupam um espaço intermediário entre os dois extremos. Os custos para desenvolver um jogo variam consideravelmente entre essas categorias, indo de zero até ultrapassar a faixa dos 256 milhões de dólares. Um jogo *indie* costuma custar entre 50 mil a 750 mil dólares, enquanto um jogo III pode variar de 1 milhão até 10 milhões de dólares. Já os jogos AAA ultrapassam a casa dos 10 milhões de dólares, com alguns títulos atingindo custos superiores a 100 milhões.

Com orçamentos multimilionários, jogos como *Red Dead Redemption* e *Call of Duty: Modern Warfare 2* que tiveram orçamentos estimados em 100 milhões e 250 milhões de dólares respectivamente (Valente, 2020), exemplificam os elevados custos envolvidos na produção de grandes títulos. Por outro lado, estúdios independentes, que lidam com restrições orçamentárias, precisam maximizar seus recursos de forma eficiente. Nesse contexto, a Geração Procedural de Conteúdo (*Procedural Content Generation* - PCG) emerge como uma alternativa para reduzir custos e otimizar recursos. Segundo Hendrikx *et al.* (2013), essa técnica consiste na criação automática de elementos de jogos por meio de algoritmos, ao invés de gerar manualmente cada recurso.

A PCG tem se mostrado uma ferramenta eficaz na indústria, sendo amplamente utilizada por diversos desenvolvedores para otimizar a criação dos jogos, reduzindo o tempo necessário para criação de insumos e aumentando a qualidade dos resultados. Isso torna o estudo dessa tecnologia relevante, especialmente para estúdios com orçamentos mais limitados, que buscam maximizar a eficiência na produção de jogos (Canedo, 2022).

Um exemplo notável citado por Gregory (2022) é o jogo *No Man's Sky*, desenvolvido pela *Hello Games*. Segundo o autor, este título conta com a impressionante quantidade de 18 quintilhões de planetas e luas, cada um com características únicas de fauna e flora, gerados de forma procedural. Esse vasto universo foi criado em apenas dois anos, um tempo relativamente curto considerando a imensidão e complexidade do jogo. Evidenciando que, com a utilização de PCG, os jogos conseguem gerar uma enorme quantidade de ativos dinâmicos a um custo consideravelmente baixo, especialmente em relação ao volume criado.

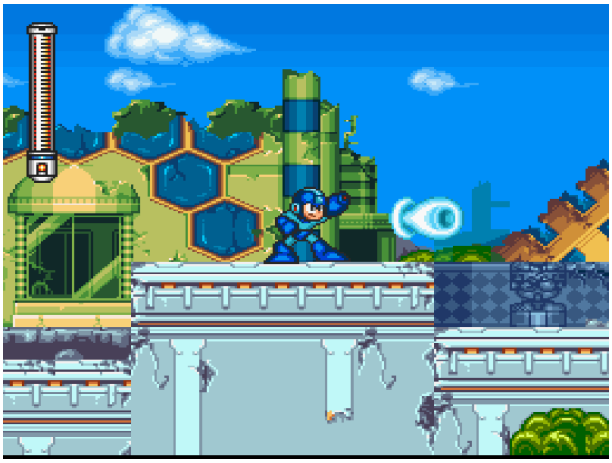
Existe uma ampla gama de variedade de itens que podem ser gerados proceduralmente, abrangendo desde texturas e imagens de jogos, até a criação de cidades, biomas, e estruturas inteiras, além de conteúdos derivados dos próprios jogos. Essas categorias são discutidas com mais detalhes na seção 2.1. Dentro desse conjunto de elementos gerados de forma procedural, existem as animações procedurais, que são animações criadas pelo computador em tempo real (Deglorie *et al.*, 2015).

De acordo com Deglorie *et al.* (2015), as animações procedurais oferecem uma solução eficiente para a criação de animações adaptáveis, que são altamente desejadas em ambientes dinâmicos, como jogos e simulações. Isso se deve ao fato de que a produção manual ou captura de movimento para animações envolve um custo elevado e não se ajusta automaticamente a diferentes situações. Por exemplo, a animação de um personagem caminhando em um terreno

irregular pode ser gerada de forma procedural para que a animação de caminhada do personagem se adapte ao terreno, economizando tempo e recursos, além de aumentar a flexibilidade do processo de animação e deixá-la mais realista.

O foco deste trabalho, é a aplicação de animações procedurais em jogos de plataforma em duas dimensões (2D), que são caracterizados por um estilo de jogabilidade em que o jogador controla um personagem que deve navegar por cenários compostos de plataformas — geralmente pulando, escalando ou balançando entre diferentes níveis — e que podem incluir elementos adicionais como combate ou tiro (Rogers, 2013). Exemplos clássicos do gênero é a série *Super Mario*, da Nintendo; a série *Sonic* da Sega; E *Mega Man* da Capcom.

Figura 2 – Mega Man 7 lançado no ano de 1995 pela Capcom



Fonte: Imagem retirada do site da Megaman (2024)

Apesar de seu grande potencial, a utilização de animações 2D geradas proceduralmente ainda é um recurso pouco explorado nos jogos de plataforma. Essas animações, desenvolvidas em tempo real por algoritmos, poderiam gerar interações mais dinâmicas e responsivas entre o personagem e o ambiente, ajustando-se de forma mais natural às ações dos jogadores. Essa adaptabilidade permitiria uma experiência de jogo mais fluida e imersiva. No entanto, sua aplicação nesse gênero específico de jogos permanece restrita, evidenciando uma área com grande margem para inovações futuras (Barbosa; Massukado; Nakamura, 2021).

Para desenvolvedores independentes, que frequentemente enfrentam restrições de orçamento e tempo, a redução de gargalos no desenvolvimento e a minimização da dependência de trabalho manual são fundamentais. Ao adotar abordagens mais eficientes, como o uso de técnicas procedurais, esses criadores podem otimizar seus recursos e acelerar a produção de conteúdo, resultando em um produto final mais refinado e rico. Isso não só aumenta as chances de sucesso do projeto, como também pode impulsionar o crescimento de estúdios, gerar empregos e contribuir positivamente para a economia local e nacional de onde os estúdios residem.

1.1 Objetivos

1.1.1 Objetivo Geral

O objetivo geral deste trabalho é apresentar e comparar técnicas de animação procedural aplicadas a jogos 2D, com foco no gênero plataforma. O estudo busca expor e discutir as limitações, vantagens e desvantagens das abordagens tradicionais e procedurais, além dos custos manuais envolvidos na criação dessas animações. O intuito é fornecer uma visão clara da área para desenvolvedores de jogos *indie*, auxiliando-os a tomar decisões que possam reduzir os custos de produção de seus projetos.

1.1.2 Objetivos Específicos

- Realizar uma revisão bibliográfica com o objetivo de examinar o estado da arte e identificar trabalhos que explorem técnicas de animação procedural em ambientes 2D.
- Desenvolver entidades e elementos de jogo do gênero de plataforma 2D, que aplique técnicas de animação procedural, como simulação de água, fogo, tecido. As entidades deverão ser capazes de interagir com o ambiente de forma dinâmica, adaptando seus movimentos e comportamentos aos obstáculos encontrados durante a locomoção pelo cenário.
- Criar os mesmos elementos de jogo utilizando técnicas tradicionais de animação, para permitir a comparação direta com as técnicas procedurais.
- Analisar e expor as limitações, vantagens, desvantagens, custos e o desempenho computacional das duas abordagens, ressaltando as diferenças em termos de escopo e eficiência. Também será considerado o nível de esforço humano necessário para a criação de cada tipo de animação, destacando o impacto desse fator no processo de produção.

1.2 Justificativa

Há diversos fatores que contribuem para que um jogo seja bem aceito pelo público e alcance sucesso nas vendas, garantindo a lucratividade do estúdio. Um desses fatores é a qualidade visual, especialmente as animações, que devem ser atrativas o suficiente para captar o interesse dos jogadores. No entanto, o desenvolvimento de jogos envolve custos elevados e demanda um significativo investimento, tanto financeiro quanto de esforço humano. O custo de criar animações envolventes é especialmente alto, o que representa um desafio para estúdios independentes, que geralmente possuem um orçamento limitado.

As animações tradicionais exigem que o artista crie manualmente cada quadro que compõe o movimento desejado. Dependendo da complexidade da ação e do personagem, isso pode demandar dezenas ou até centenas de quadros, exigindo muitas horas de trabalho para alcançar o resultado pretendido. Embora esse tipo de animação possa representar fielmente a

intenção do artista, ela tem a limitação de ser estática, sem se adaptar aos eventos ou ao ambiente do jogo.

Por outro lado, as animações procedurais não requerem a criação quadro a quadro. Em vez disso, é necessário apenas o *design* básico do personagem, com os quadros da animação sendo gerados automaticamente por algoritmos. Isso torna as animações mais dinâmicas e capazes de se adaptar a diferentes cenários e eventos, permitindo que o jogador possa experimentar variações no comportamento das entidades e do ambiente.

No entanto, dependendo da ação do personagem, programar essas animações pode ser complexo, exigindo que o programador tenha um bom domínio de técnicas avançadas de animação. Além disso, as animações procedurais podem ser mais custosas do ponto de vista computacional, já que exigem mais cálculos do que simplesmente exibir quadros estáticos em sequência.

Atualmente, a animação procedural é mais comum em ambientes tridimensionais (3D), enquanto o gênero de jogos 2D continua sendo menos explorado nesse aspecto. Poucos desenvolvedores a utilizam, optando por simplificar o comportamento das entidades não controladas pelo jogador, seja devido a escolhas de *design* ou por limitações financeiras e técnicas (Barbosa; Massukado; Nakamura, 2021).

Diante desse cenário, para ajudar estúdios independentes e desenvolvedores com orçamento limitado a otimizar tempo e reduzir custos, este trabalho visa apresentar técnicas de animação procedural que podem ser aplicadas em jogos 2D, especificamente no gênero de plataforma e com um ambiente dinâmico. O objetivo é fornecer um conjunto de técnicas que possam ser exploradas por desenvolvedores independentes, comparando as animações procedurais com as animações tradicionais, analisando suas vantagens e desvantagens, e avaliando o desempenho computacional de cada abordagem.

1.3 Trabalhos relacionados

Para a realização deste trabalho, foi tomado vários artigos, livros e trabalhos como base, mas dentre todos, os principais foram: O trabalho de Barbosa, Massukado e Nakamura (2021) com título de "Aplicação de animação procedural e Inteligências Artificiais independentes do jogador em jogos digitais", o trabalho de Hendrikx *et al.* (2013) com título "*Procedural Content Generation for Games: A Survey*", o trabalho de Yu (1999) com título "*Stylised procedural animation*".

O trabalho de Barbosa, Massukado e Nakamura (2021) teve como objetivo o desenvolvimento de um jogo simples de plataforma para explorar como a junção de inteligência artificial (IA) e animações procedurais podem contribuir para uma experiência diferenciada nesse gênero. O jogo foi concebido para integrar elementos tradicionais de plataforma com criaturas controladas por IA e animadas proceduralmente. Para isso, foi criado um *sandbox*[1] composto

[1] O estilo de jogo *sandbox*, caracteriza-se pelo jogador possuir ampla liberdade para explorar e alterar o mundo virtual, com poucas restrições, permitindo ao jogador decidir quais atividades realizar (TechTudo, 2022).

por sete salas interconectadas, cada uma projetada para testar as interações entre o jogador e as criaturas.

O projeto foi inspirado no jogo *Rain World*, onde simula um ecossistema de criaturas, executando várias salas ao mesmo tempo fora da vista do jogador, simulando interações entre as criaturas, fazendo com que tenha um mundo vivo e único a cada momento para cada jogador. Então, o jogo se ambienta em uma fábrica submarina, onde o jogador controla um robô que explora esse ambiente. As criaturas inspiradas em animais marinhos, foram planejadas para interagir tanto com o jogador quanto entre si. As animações das criaturas e do personagem principal foram geradas utilizando técnicas de cinemática inversa e direta.

Os resultados demonstraram que a aplicação de animações procedurais trouxe fluidez aos movimentos do personagem. No entanto, o trabalho apontou para áreas de melhoria, como a diversificação dos comportamentos das criaturas e maior expressividade nas suas animações. Mesmo com essas limitações, o jogo criado é funcional e oferece uma base expansível para futuros desenvolvimentos de jogos de plataforma que utilizam IA e animações procedurais.

O trabalho de Hendrikx *et al.* (2013) trás uma taxonomia de conteúdo de jogos PCG com seis camadas, que foram mostradas na seção 2.1. Essas camadas são organizadas hierarquicamente, de forma que as superiores dependem das inferiores para serem criadas. Para desenvolver essa taxonomia, os autores realizaram uma extensa revisão da literatura sobre o estado da arte em PCG, identificando no processo uma tendência interessante: à medida que a complexidade das camadas aumenta, a maturidade das técnicas diminui. Além disso, foi observado que, embora muitos jogos comerciais utilizem técnicas de PCG, esse uso geralmente é limitado a apenas um tipo específico de conteúdo.

Além de fornecer uma visão abrangente sobre a área, Hendrikx *et al.* (2013) também trás contribuições valiosas para pesquisas futuras, destacando a necessidade de desenvolver "*bits* de jogo" que ainda não foram bem desenvolvidos. Outras sugestões incluem a criação de geradores mais realistas e o estudo da relação entre realismo e desempenho, assim como o equilíbrio entre realismo e controle no desenvolvimento de PCG. Essas recomendações fornecem uma base importante para avanços futuros na área.

O trabalho de Yu (1999), aborda a animação de efeitos estilizados, investigando seus desafios e complexidade. Nele, foi proposto uma solução utilizando modelagem procedural para capturar a essência da animação desenhada à mão, ampliando um trabalho anterior do autor sobre síntese de desenho à mão com título de "*Computer generation of decorative tree images*".

As contribuições do trabalho incluem o desenvolvimento de uma estrutura de animação que permite a criação procedural de efeitos estilizados. A metodologia exige apenas que o usuário especifique condições iniciais, como posição, tamanho e cor, facilitando o processo criativo. Os resultados, demonstram a capacidade da abordagem em gerar animações em 2D e 3D, mantendo um estilo artístico.

O estudo avançou a área de animação assistida por computador e animação procedural, sendo o primeiro, segundo ele, a integrar modelos hierárquicos e controle paramétrico em

um único *framework*. As principais contribuições incluem a modelagem de objetos baseada no processo de desenho à mão, a inclusão de expressões artísticas no controle paramétrico, a expansão de animações 2D para 3D, e o desenvolvimento de técnicas de renderização estilizada.

1.4 Estrutura do trabalho

Este trabalho está organizado em cinco partes, cada qual sendo uma seção diferente. Na primeira seção, é apresentada uma introdução ao contexto e cenário atual do desenvolvimento de jogos, com foco na evolução do desenvolvimento deles. A partir desse panorama, o problema específico que motiva este trabalho é detalhado, juntamente com os objetivos gerais. Por fim, é descrito uma visão geral da estrutura do trabalho.

Na segunda seção, é realizada uma pesquisa no estado da arte das animações tradicionais e procedurais em jogos 2D. São exploradas as principais técnicas de cada abordagem, destacando suas aplicações e relevância. Esses conhecimentos serão fundamentais para a aplicação e análise experimental das animações desenvolvidas.

Na terceira seção, é apresentado toda a metodologia que rege o trabalho, também é detalhado o Documento de *Design* de Jogo (*Game Design Document* - GDD). Esse documento descreve o planejamento dos elementos desenvolvidos, incluindo suas funcionalidades, interações e aspectos técnicos. Também são abordados o *design* do personagem principal, da entidade aranha e dos demais elementos do jogo. Além disso, são descritos os cenários utilizados para os testes de cada componente, proporcionando uma visão abrangente do projeto.

A seção também explora o processo de desenvolvimento e implementação dos elementos tradicionais e procedurais. As técnicas definidas no GDD e explicadas na segunda seção são implementadas no motor de jogo Godot, e todas as decisões tomadas durante o processo de implementação são explicadas. A seção destaca as abordagens e os ajustes feitos para cada técnica de animação, tanto tradicional quanto procedural, enfatizando as particularidades de cada método.

Por fim, a seção trata dos testes de desempenho realizados em diferentes dispositivos. São descritos os métodos utilizados para testar o desempenho em dispositivos móveis e computadores de variadas configurações, além das especificações do *hardware* de cada um desses dispositivos. Com isso, pretende-se avaliar o comportamento de cada abordagem de animação sob diferentes condições de *hardware*.

Na quarta seção, os resultados dos testes de desempenho são apresentados e analisados. São discutidos os impactos de cada técnica, considerando aspectos qualitativos, como a qualidade visual e fluidez da animação, e quantitativos, como o consumo de recursos da Unidade Central de Processamento (*Central Processing Unit* - CPU) e a Unidade de Processamento Gráfico (*Graphics Processing Unit* - GPU). As vantagens e limitações de ambas as abordagens são comparadas, visando identificar as melhores práticas que possam auxiliar os desenvolvedores e reduzir o custo de desenvolvimento para diferentes contextos.

Na quinta e última seção apresenta as conclusões do trabalho, respondendo às questões

de pesquisas previamente definidas na seção 3, e destacando as contribuições e implicações dos resultados obtidos. As considerações finais discutem o potencial de aplicação das técnicas abordadas para redução de custos e democratização de desenvolvimento de jogos, beneficiando estúdios e desenvolvedores *indies* que, muitas vezes, não possuem os recursos necessários para investir.

2 REFERENCIAL TEÓRICO

Nesta seção, serão explorados os principais temas que fundamentam este estudo, todos embasados por uma pesquisa no estado da arte. Essa seção é essencial para estabelecer as bases conceituais da pesquisa, oferecendo uma análise das teorias e abordagens relevantes.

2.1 Geração Procedural de Conteúdo

A criação de jogos eletrônicos envolve múltiplos desafios. Para atrair jogadores, os jogos precisam ser divertidos, o que exige que apresentem diversas qualidades para garantir sua sobrevivência, especialmente em um mercado competitivo (Lopes, 2021), onde a inovação e a capacidade de gerar experiências únicas são diferenciais essenciais. Hoje, a produção de jogos de alta qualidade pode exigir o trabalho de vários indivíduos, incluindo artistas, *designers*, programadores e engenheiros de áudio para que os conteúdos do jogo possam ser feitos (Hendrikx *et al.*, 2013).

Nesse cenário, os jogos necessitam de vários recursos, que são criados pelos profissionais, como os desenhos e modelos digitais, animações, efeitos sonoros e músicas, a programação em si do jogo, personagens, história, níveis e muito mais, exigindo um alto valor de investimento por parte das empresas (Lopes, 2021), especialmente para jogos de grande porte. Como dito por (Hendrikx *et al.*, 2013), os custos de produção de um jogo que cresce rapidamente, já foram parcialmente responsáveis pela falência de jogos que foram, em algum momento, bem-sucedidos. Pois além dos recursos necessários para criar todos os conteúdos, é necessário investimento na distribuição, licenciamento, propaganda, testagem e em várias outras áreas, para que ele possa funcionar (Lopes, 2021).

Alguns métodos podem ser aplicados para que tenha a diminuição do trabalho necessário para a criação dos mesmos, e um dos métodos que têm se destacado é a PCG, que pode ser usado para reduzir tempo e dinheiro necessários na produção de jogos, também aumentando a diversão e o engajamento (Lopes, 2021).

A PCG para jogos pode ser definida como a aplicação de computadores para gerar conteúdo de jogo, a ideia principal é que os conteúdos dos jogos, não sejam criados manualmente, e sim por computadores (Hendrikx *et al.*, 2013). Ou seja, é o processo de algo ser desenvolvido por meio de um algoritmo ou procedimento (Green, 2016). Com uso de *scripts*, PCG pode ser aplicado a quase todos os aspectos de um jogo, como por exemplo, gerar uma cena de taverna aleatória a partir de itens pré-fabricados como mesas, barris, caixas entre outros. (Watkins, 2016).

Um exemplo de como a aplicação de PCG pode trazer benefícios, é quando se é necessário ter um grande número de personagens diferentes, ao invés de criar cada personagem individualmente, poderia criar um personagem único, e aleatorizar certas características, como cabelo, olhos, pele e roupas. Com isso, as técnicas processuais se tornam uma alternativa para criar mundos de jogos grandes e complexos em pouco tempo, sem aumentar a carga de trabalho dos artistas (Hendrikx *et al.*, 2013).

A redução no tempo ao criar conteúdo, como no exemplo anterior, permite investir em outras áreas que, anteriormente, ficavam sem recursos. Isso resulta em menos falhas, mais propagandas e o aumento na qualidade de trabalho dos funcionários. Além disso, um jogo que antes teria apenas um número pequeno e estático de recursos, tornando-se chato para os jogadores rapidamente, com a utilização de PCG, poderia oferecer um número maior de variedade de elementos diversificados, aumentando significativamente a rejogabilidade, diversão e engajamento dos jogadores. Isso ocorre porque o produto se transforma em um ambiente dinâmico, onde os jogadores podem sempre investigar, experimentar e descobrir coisas novas (Lopes, 2021).

Historicamente, a geração procedural começou com jogos que utilizavam caracteres ASCII para criar níveis automaticamente, como no clássico *Rogue*, desenvolvido em meados de 1980 (Watkins, 2016). Ainda segundo o autor, essa abordagem permitia que os jogos fossem menos dependentes da memória ao gravar novos níveis em tempo real, ao invés de pré-carregá-los, o que era essencial em sistemas com capacidade computacional limitada. A partir de 1996, as ferramentas comerciais já usavam PCG, tornando uma técnica estabelecida no mercado, mesmo que ainda fossem limitadas a jogos narrativos com interpretação de personagem como RPG's (Lopes, 2021).

De todos os conteúdos possíveis de serem gerados proceduralmente, Hendrikx *et al.* (2013) classificaram-se em seis classes principais, e além dessas classes de conteúdos, foram considerados conteúdos derivados. As classes apresentadas pelo autor são:

- *Bits* de jogo: São unidades fundamentais de conteúdo dentro de um jogo, que geralmente não interagem diretamente com o jogador quando analisados isoladamente.
  - Texturas: As texturas são imagens que acrescentam visuais à geometria e aos modelos, além de representarem elementos como menus. Muitos jogos agora utilizam artistas apenas para elementos mais complexos, enquanto aplicam técnicas procedurais para gerar a maioria das texturas.
  - Som: A criação de sons de forma procedural é possível, permitindo que sons possam ser gerados em tempo de execução, deixando mais orgânico e realista. Diferente da pré-gravação, que leva a limitações, como por exemplo, sons irrealistas ou incondizente com a cena que se passa.
  - Vegetação: A vegetação em jogos é utilizada para criar um ambiente mais realista e imersivo. Árvores podem, por exemplo, cobrir montanhas e juncos aparecer ao longo de rios. Além de seu papel estético, a vegetação pode ser funcional, servindo como abrigo, recurso para jogadores, e ajudando a definir o clima do ambiente, influenciando diretamente a jogabilidade.
  - Edifícios: Enquanto a vegetação é importante para uma ambientação realista em jogos que retratam em partes a natureza, edifícios são proporcionalmente importantes

em jogos que retratam o setor urbano. Para ter um aspecto realista, os edifícios necessitam ser diversos, e pertencentes a um estilo arquitetônico único.

  - Comportamento: O comportamento em jogos refere-se à forma como os objetos interagem entre si e com o ambiente. O uso de comportamento procedural é comum para criar a sensação de maior complexidade, permitindo que os jogadores explorem maneiras criativas de manipular ou aproveitar essas interações.
  - Fogo, Água, Pedra e Nuvens: Os elementos fogo, água, pedra e nuvens são utilizados em jogos para tornar o mundo mais realista, quando bem implementados.

- Espaço do jogo: O espaço do jogo é o ambiente onde ocorre a ação, é preenchido com *bits* do jogo. Existem mapas internos e externos, no qual a diferença consiste nos aspectos estáticos, culturais e técnicos, onde os mapas externos costumam ter temas consistentes, enquanto os internos contêm artefatos inter-relacionados e únicos.

  - Mapas internos: Mapas internos representam espaços divididos em salas conectadas por corredores, escadas ou organizados em masmorras. Também podem incluir cavernas com formas variadas ou edifícios com cômodos importantes em termos de tamanho, posição e conteúdo. Hendrikx *et al.* (2013) cita o exemplo do jogo *Prince of Pérsia*, no qual é ambientado em uma masmorra de múltiplos níveis.
  - Mapas externos: Mapas externos retratam a elevação e estrutura de terrenos externos. Por exemplo uma cidade, no qual é formada por vários elementos, como edifícios, casas, estruturas e estradas. No qual, esses elementos pertencentes aos mapas externos podem ser mapas internos, sendo a transição entre eles geralmente discreta.
  - Corpos d'água: Os elementos de corpos d'água, como rios e lagos, costumam funcionar como obstáculos ou áreas interativas no mapa dos jogos. Além disso, outros elementos, como áreas de teletransporte, montanhas, ravinas e grutas, também fazem parte do espaço de jogo, contribuindo para a diversidade e complexidade do ambiente.

- Sistemas de jogo: Os sistemas de jogo utilizam teoria e modelagem de sistemas complexos para gerar ou simular elementos dentro de um jogo, o que contribui para a sua credibilidade e atratividade.

  - Ecossistemas: Os ecossistemas determinam como a flora e fauna se posicionam, evoluem e interagem, utilizando algoritmos e regras específicas. Assim como os itens anteriores, ecossistemas gerados e evoluídos proceduralmente deixam o jogo mais realista, tornando o ambiente menos linear.
  - Redes de estradas: As redes de estradas constituem a base de um mapa ao ar livre, facilitando o transporte entre pontos de interesse e organizando o tráfego nas cidades. Os principais desafios na criação dessas redes incluem equilibrar a aleatoriedade e

estrutura, ao mesmo passo que transmite a visão do *designer* do jogo sobre a importância dos locais, como distância (por meio de estradas intermitentes), dificuldade (por meio de estradas interrompidas), relevância (estradas mais largas).

  - Ambientes urbanos: Constituídos de densos aglomerados de edifícios onde as pessoas vivem e interagem. Algoritmos procedurais criam essas cidades gerando primeiro as redes de estradas, subdividindo o terreno em lotes de construção e, em seguida, construindo edifícios.
  - Comportamento da entidade: O comportamento do jogador é fundamental para criar uma experiência realista em mundos virtuais, permitindo interações complexas entre jogadores e o ambiente. Além disso, personagens não jogáveis (*Non-Playable Character* - NPC) que reagem ao jogador desempenham um papel crucial nessa ilusão. Algoritmos procedurais também podem simular padrões de movimentos em grupo, contribuindo para uma maior sensação de realismo no ambiente do jogo, como por exemplo, inimigos formando estratégias em tempo real para capturar o jogador.

- Cenários de jogo: Os ambientes de jogo geralmente apresentam, de maneira sutil para jogadores, como os eventos do jogo ocorrem e se desenvolvem ao longo do tempo.

  - Quebra-cabeças: A busca pela solução de um quebra-cabeça dentro do jogo é, em si, uma parte essencial do jogo, proporcionando uma experiência gratificante. São exemplos de quebra-cabeças de jogos, enigmas, palavras cruzadas e finais de xadrez, com a dificuldade geralmente dependente do espaço de soluções e da experiência do jogador.
  - *Storyboards*: Os *storyboards* funcionam como ferramentas de *design* que auxiliam desenvolvedores e jogadores. São geralmente apresentados no formato de histórias em quadrinhos que mostram eventos através de painéis sequenciais, combinando elementos visuais e textuais. Além de orientarem os jogadores, esses *storyboards* podem proporcionar entretenimento, especialmente durante *cutscenes* [1] que se intercalam com a jogabilidade.
  - História: A história em um jogo é fundamental para manter o jogador motivado, dar lógica aos eventos e fornecer um objetivo. Ela pode ser revelada por *cutscenes*, missões ou estar integrada ao universo do jogo.
  - Níveis: Consiste em um espaço de jogo separado em que o jogador possa explorar, cumprir desafios e tarefas, desviando de obstáculos.

- *Design* de jogo: Refere-se a regras, objetivos e aspectos estéticos de um jogo, como temas gráficos e arcos dramáticos, que influenciam a experiência do jogador (Norman, 2002 apud Hendrikx *et al.*, 2013).

[1] *Cutscenes* são cenas nos jogos em que o jogador geralmente não tem controle, utilizadas para apresentar elementos importantes, como trechos da história, diálogos, lembranças, ou outras informações relevantes para o enredo (Press, 2025).

  - *Design* de sistema: São as regras (o que pode ser realizado no jogo?) e os objetivos (qual o objetivo que o jogador está tentando alcançar?) do jogo em si, componentes estéticos, como tema gráfico ou arco dramático, são considerados também como elementos do design (Norman, 2002 apud Hendrikx *et al.*, 2013). Um exemplo seria um jogo de RPG, que gera regras novas a cada partida.
  - *Design* de mundo: É o *design* de um cenário, história e tema (Brathwaite; Schreiber, 2008 apud Hendrikx *et al.*, 2013).

- Conteúdo derivado: É definido por Hendrikx *et al.* (2013), como um conteúdo criado como subproduto do mundo do jogo.
  - Notícias e transmissões: Jogos podem exibir notícias aos jogadores com base em suas ações e eventos no universo do jogo, que podem aparecer como transmissões televisivas ou artigos de jornal.
  - Placares: Tabelas de classificação de jogadores, podendo ser criadas proceduralmente para simular um universo vivo.

Essas classes podem ser hierarquizadas em forma de pirâmide, como é mostrado na figura 3, as classes do topo podem ser construídas com as classes da base (Hendrikx *et al.*, 2013). Seguindo essas categorias, este trabalho aborda elementos de *bits* do jogo, no qual é o fogo, água e tecido, junto com o comportamento, todos aplicados a jogos de plataforma 2D. Esses elementos essenciais servem como base para qualquer jogo que se enquadre nos gêneros mencionados, podendo ser adaptados conforme a necessidade de diferentes projetos.

Figura 3 – Hierarquia de conteúdos que podem ser gerados proceduralmente

Fonte: Imagem traduzida pelo autor, com base na pirâmide mostrada por Hendrikx *et al.* (2013)

2.2 Animação

Desde os primórdios, a humanidade tem buscado registrar o que vê ao seu redor. Como exemplo deste fato, tem-se as pinturas rupestres nas cavernas, criadas por nossos antepassados, que retratam a realidade em que viviam (Scabeni, 2019). Segundo Costa e Rosa (2015) "Antigamente a população primitiva deixava gravadas pinturas rupestres nas paredes das cavernas. Esses desenhos eram simples, mas podem ser considerados como um dos modos eficazes de comunicação utilizados pela humanidade antes da invenção da escrita".

Com o passar dos milênios, desenvolveram-se equipamentos que permitiram ao ser humano retratar a realidade com mais precisão e facilidade. Nos dias atuais, quando um artista cria uma animação tradicional, ele desenha cada movimento que deseja retratar. Por exemplo, ao animar uma bola caindo, o artista desenha cada etapa da queda em um desenho único, até que a animação esteja completa. Cada um desses desenhos é chamado de quadro, e a junção de vários quadros é o que cria a animação. Esse método foi sendo usado desde as primeiras invenções para criação de animações, assim como mostra Scabeni (2019):

> Objetos como o zootrópio (ou "roda da vida") tentaram imitar movimento a partir de trocas rápidas de imagens em um local de foco. Mais tarde, tivemos a chegada do folioscópio, ou flipper book, se se popularizou devido à sua simplicidade de uso: um simples bloco de folhas com vários desenhos que quando folheados rapidamente parecem ganhar vida.

As animações nos jogos, quando feitas de forma clássica, seguem o mesmo princípio da animação tradicional. Quando é visto um personagem andando, correndo ou pulando, o que está sendo exibido são quadros, previamente desenhados, e apresentados em sequência, criando, assim, a sensação de movimento (Barbosa; Massukado; Nakamura, 2021). No entanto, uma limitação dessas animações pré-definidas, diz respeito a elas ficarem restritas aos movimentos já programados, não permitindo a criação de movimentos mais complexos ou que se adaptem a diferentes situações.

Esta seção abordará a aplicação de animações tradicionais em jogos 2D, e seus princípios, assim como o uso de animações procedurais nos jogos e algumas de suas técnicas, que surgem como uma solução para as limitações impostas pelas técnicas tradicionais.

2.2.1 Aplicação de animações tradicionais em jogos 2D

A aplicação de animações tradicionais em jogos 2D é fundamentada em técnicas de animação quadro a quadro, uma abordagem que remonta às origens da animação. Essas técnicas continuam sendo relevantes para o desenvolvimento de jogos, oferecendo um arcabouço de métodos para que os artistas possam adicionar fluidez e personalidade aos personagens. Nesta seção, é explorado como animações são aplicadas em jogos 2D, as principais técnicas envolvidas e exemplos práticos de títulos que utilizam essas abordagens.

Na animação tradicional aplicada nos jogos 2D, é bastante utilizada a técnica quadro a quadro, que consiste na criação de desenhos individuais com pequenas variações entre eles,

exibidos em sequência para gerar a ilusão de movimento. Esse princípio é baseado na persistência da visão, redescoberto por Peter Mark em 1824, essa técnica de animação quadro a quadro é usada há décadas na indústria de jogos digitais, com títulos clássicos como *Street Fighter II* e *Mortal Kombat*, ambos os jogos lançado em 1992 (Mlanarczyki, 2023).

Para atrair o público, as animações precisam ser bem elaboradas e executadas, seja elas para jogos ou para qualquer outro tipo de mídia. Para auxiliar os animadores nessa tarefa, foi proposto por Thomas e Johnston (1995), 12 princípios da animação em seu livro *Disney Animation: The illusion of life*. Esses princípios podem ser definidos como "[...] técnicas para desenhar personagens em movimento que, após anos de uso, se tornaram fundamentais para a realização do processo de animação [...]" (Bahia; Izolani; Stein, 2020). Além desse trabalho, Os 12 princípios presentes no livro foram referenciados nos trabalhos de Scabeni (2019) e Bahia, Izolani e Stein (2020), eles são:

- *Squash and Stretch* (Comprimir e Esticar): Deformações feitas no personagem animado em momentos específicos para expressar mais claramente a ação que ele está tomando, e assim, dando a ilusão de volume e peso. O ótimo exemplo dado no livro, é uma bola quicando, no qual ela achata quando toca o chão, e estica quando está em alta velocidade no ar.

- *Anticipation* (Antecipação): Um ou mais movimentos antes da ação principal, para preparar o espectador.

- *Staging* (Encenação): É a representação visual, clara da cena, criando poses bem definidas nos personagens, no qual o espectador consegue entender a mesma com apenas um olhar rápido.

- *Straight Ahead and Pose-To-Pose* (Animação Direta e Posição Chave): Na animação direta, o animador desenha quadro a quadro sem um planejamento rígido, resultando em animações espontâneas e dinâmicas. Já na animação de posição chave, as poses principais são cuidadosamente planejadas e desenhadas para garantir clareza e controle, após isso é desenhado quadros intermediários para ligar as poses principais.

- *Follow Through and Overlapping Action* (Ação de Acompanhamento e Sobreposição): Essas técnicas garantem que diferentes partes do corpo de um personagem continuem se movendo após a ação principal ser interrompida. Por exemplo, quando um personagem corre e pára abruptamente, sua capa, roupas ou cabelo continuam em movimento por um curto período, seguindo o impulso original. Isso adiciona realismo e fluidez à animação, simulando o comportamento natural de objetos e apêndices com inércia.

- *Slow In and Slow Out* (Aceleração e Desaceleração): O conceito de *Slow In* e *Slow Out* envolve a técnica de cronometrar os desenhos principais de um personagem, onde as transições entre posições extremas são suavizadas com intermediários próximos a esses extremos, criando uma sensação mais fluida de movimento.

- *Arcs* (Arcos): Os movimentos naturais dos seres vivos geralmente seguem um arco, ao invés de trajetórias retas e mecânicas. Baseado nisso, nas animações tem-se a necessidade de fazer movimentos seguindo arcos, para que a ação pareça mais orgânica. Movimentos mais retos deixam a animação com sensação de robótica.

- *Secondary* Action (Ação Secundária): A ação secundária consiste em movimentos ou gestos que complementam a ação principal de uma cena, como por exemplo, uma figura triste enxugando uma lágrima enquanto se vira. Esses movimentos devem ser sutis e subordinados à ação primária para evitar conflitos ou distrações.

- *Timing* (Tempo): O tempo na animação é determinado pelo número de desenhos usados em cada movimento, afetando a duração e o ritmo da ação na tela. Ações mais devagar requerem mais quadros e ações mais rápidas requerem menos quadros.

- *Exaggeration* (Exagero): Exagero é a técnica de animação que deixa as cenas mais extremas, ampliando assim as emoções e ações, tornando-as mais claras, intensas e impactantes.

- *Solid Drawing* (Desenhos Sólidos): Esse princípio apresenta a ideia pela qual as formas dos desenhos dos personagens precisam ter peso, profundidade e equilíbrio. Ainda segundo o livro, para que os artistas consigam realizar bons desenhos sólidos, necessita-se dominar a habilidade de desenhar personagens de todos os ângulos.

- *Appeal* (Apelo): O apelo refere-se à capacidade de uma figura de atrair e manter a atenção do espectador, independentemente de sua natureza, seja heroica ou vilanesca. Um desenho fraco ou design ruim não possui apelo, enquanto um bem executado, com formas atraentes e movimentos fluidos, consegue capturar o interesse do espectador.

Compreender esses princípios proporciona aos animadores uma visão mais estruturada e lógica sobre como criar animações que transmitem de maneira eficaz as ideias por trás delas (Bahia; Izolani; Stein, 2020). Sua importância é tamanha, que esses princípios são utilizados até hoje em indústrias de animação, “Essa base de princípios é muito utilizada na animação de filmes com objetivo de contar a narrativa projetada para o espectador” (Mlanarczyki, 2023).

Diferente de uma série ou filme animado, um jogo tem o fator de interação do jogador com o personagem. O jogador deve conseguir movimentar o personagem com a agilidade que o estilo do jogo requer. Por exemplo, em jogos de luta, a agilidade deve ser rápida o suficiente para que o jogador tenha a sensação de estar em um combate. Por conta disso, um aspecto crucial da animação em jogos é a integração de diversas animações curtas dos personagens que são produzidas em sequência. Então, o animador tem que garantir que elas sejam fluidas e suaves, para não comprometer a experiência do jogador (Scabeni, 2019).

Por conta da interatividade dos jogos, para alcançar a agilidade necessária, a quantidade de quadros das animações dos personagens necessários para transmitir a ação que ele está fazendo, precisa ser menor. Com isto, o nível de dificuldades em criar animações para jogos é

diferente do nível de dificuldade em criar animação pro cinema, tendo assim, a necessidade de adequar os 12 princípios da animação para as animações em jogos (Bahia; Izolani; Stein, 2020). Um exemplo é o item da antecipação, essa técnica deve ser utilizada com cuidado, pois ela pode ter um resultado diferente do obtido em séries e filmes (Scabeni, 2019). Muita antecipação, resulta em um atraso na ação do personagem, o que pode acometer uma experiência ruim do jogador para com o personagem.

Ao mesmo tempo em que o princípio de antecipação deve ser aplicado com destreza no personagem principal, nos inimigos, a situação é oposta. Nesse caso, a antecipação é necessária para que o jogador consiga prever as futuras ações dos inimigos. Quando um inimigo ataca sem qualquer aviso prévio, o jogador pode se sentir frustrado, e injustiçado. Por outro lado, se o inimigo "telegrafa" a sua ação, o jogador tem a oportunidade de pensar em uma resposta ou estratégia, o que gera uma sensação de responsabilidade ao personagem quando é atingido, diminuindo o sentimento de injustiça (Scabeni, 2019).

Outra diferença significativa entre as animações do cinema e as animações de jogos, diz respeito ao estágio de retorno à posição natural do personagem. Tal estágio é irrelevante, ou até mesmo sem necessidade de implementação, pois, devido à natureza responsiva dos jogos, esses tipos de animação acabam deixando o jogo mais lento (Mlanarczyki, 2023).

Há diversos outros fatores que diferenciam a animação aplicada aos jogos daquelas aplicadas ao cinema, mas este trabalho não pretende abordá-los em sua totalidade. No entanto, estudos como o de Bahia, Izolani e Stein (2020) adaptam os 12 princípios da animação, proposto por Thomas e Johnston (1995), para o contexto dos jogos de luta em 2D, formando assim, os "12 princípios de animação para jogos de luta". De maneira geral, é fundamental que os animadores adotem uma abordagem diferente ao trabalhar com animação em jogos, considerando a interatividade constante do jogador (Scabeni, 2019).

Figura 4 – Imagem do jogo *Cuphead*

Fonte: Imagem retirada da página do jogo na plataforma *Steam* (Studio MDHR Entertainment Inc., 2017)

Atualmente no mercado de jogos digitais, existem inúmeros jogos de sucesso no qual implementaram técnicas de animação tradicional com muita maestria. O primeiro deles, que

é válido ressaltar, é o jogo *Cuphead* (Studio MDHR Entertainment Inc., 2017). No qual, em sua página na plataforma *Steam*, possui definição como um jogo 2D de tiro e ação, feito com uma temática clássica das animações da década de 1930, como percebe-se na figura 4, todas as animações foram feitas a mão, recriando as técnicas da década.

Figura 5 – Imagem do jogo *Skullgirls*

Fonte: Imagem retirada da página do jogo na plataforma *Steam* (Hidden Variable Studios, 2012)

Outro jogo notável na indústria dos jogos, e famoso por utilizar animações tradicionais, é o jogo *Skullgirls* (Hidden Variable Studios, 2012). Este é um jogo de luta 2D, com personagens femininas marcantes, com inúmeros poderes. Segundo sua descrição na plataforma *Steam*, o jogo conta com 14 personagens animados desenhados à mão.

Figura 6 – Imagem do jogo *Hollow Knight*

Fonte: Imagem retirada da página do jogo na plataforma *Steam* (Team Cherry, 2017)

Vale ressaltar também, o jogo *Hollow Kinight* (Team Cherry, 2017), que é um jogo 2D, do gênero ação em plataforma. Neste jogo, segundo sua página na *Steam*, seus desenhos foram feitos à mão, trazendo um visual interessante como é mostrado na figura 6. "*Hollow Knights* [...] Com um estilo de personagens simples, porém elegante, a utilização da animação quadro

a quadro torna as batalhas dinâmicas, desafiadoras e adiciona personalidade nos personagens, tornando-os mais únicos" (Mlanarczyki, 2023).

2.2.2 Animação Procedural

Com o avanço da tecnologia, tornou-se possível gerar quadros de animação em tempo real, através dos computadores. Esse processo é conhecido como animação procedural. Como explicado por Barbosa, Massukado e Nakamura (2021), "[...] cada quadro é gerado em tempo de execução por um processo algorítmico, ou seja, não é necessário elaborar as imagens de antemão". Com essa técnica, os animadores não precisam desenhar cada movimento individualmente, pois a animação é gerada instantaneamente, adaptando-se a qualquer situação. Dessa forma, a limitação das animações tradicionais é superada, permitindo maior flexibilidade e dinamismo.

O conceito de procedural na computação, consiste em uma definição muito ampla, não se limitando a um único tipo de aplicação. Qualquer criação que um computador consiga criar de forma "autônoma", sem ser completamente pré-definida, pode ser considerada procedural. Como afirma Yu (1999), qualquer elemento gerado por meio de um computador pode ser considerado procedural e, de forma simples, animação procedural envolve criar um objeto e aplicar um conjunto de instruções para modificar ou animar alguma de suas propriedades. Isso confere ao conceito, uma vasta gama de aplicações em diversas áreas da computação, dependendo apenas da habilidade do programador em desenvolver um algoritmo que execute a ação desejada (Barbosa; Massukado; Nakamura, 2021).

Técnicas de animação procedural são amplamente aplicadas em computação gráfica, especialmente para a simulação de elementos que exigem maior flexibilidade na animação. Exemplos incluem a simulação de partículas de fogo, e materiais como água, lava e gelatina. Essas técnicas também são usadas na simulação de movimentação de criaturas, como exemplificado no jogo *Rain World* (2017)[2], onde diversas criaturas de diferentes tipos e tamanhos se movem de forma realista pelo mapa, adaptando-se aos obstáculos como pedras, morros e árvores. Esses métodos baseados em procedimentos computacionais são empregados na computação gráfica desde a década de 1970 (Yu, 1999), e com o avanço do poder computacional, essas técnicas têm se tornado cada vez mais comuns.

Outro exemplo de aplicabilidade é apresentado por Abreu *et al.* (2017), que propôs uma ferramenta para criação procedural de coreografias de dança. Essa ferramenta utiliza poses e modelos pré-criados, tanto por usuários comuns, quanto por profissionais de dança, gerando coreografias completas para qualquer música selecionada. A ferramenta emprega um analisador de áudio para identificar os tempos das batidas e uma inteligência artificial baseada em redes neurais para selecionar as poses mais adequadas para cada momento da música.

Além das simulações completas de comportamentos de criaturas, técnicas de animação procedural podem ser empregadas de forma parcial em objetos. Um grande exemplo é o jogo

[2] *Rain World* (Videocult, 2017), é um jogo de plataforma 2D do gênero de sobrevivência, em que o personagem principal é um lesgato, que deve sobreviver e caçar em um ecossistema perigoso e desafiador.

Celeste (Inc.; Games, 2018), no qual é um jogo 2D do gênero de plataforma, onde você controle a garota Madeline que tem como objetivo subir ao topo da montanha Celeste, onde a personagem principal possui animações 2D tradicionais, mas seu cabelo é animado proceduralmente para se movimentar e se adequar às ações do jogador (Berry, 2018 apud Barbosa; Massukado; Nakamura, 2021).

Diante da ampla gama de aplicações das animações procedurais em jogos, existe também uma variedade de técnicas e conceitos matemáticos necessários para sua implementação, entre eles a cinemática inversa e direta, essenciais para a animação de corpos esqueléticos.

A animação de um ser humano em movimento é uma das mais complexas e trabalhosas de se criar (Roberts, 2007 apud Pangesti; Utami; Sunyoto, 2019). Para obter um resultado convincente, é necessário animar cada posição e seus quadros intermediários, sempre prestando atenção à anatomia e à perspectiva de cada parte do corpo (Pangesti; Utami; Sunyoto, 2019). Uma das soluções encontradas não só para os jogos, mas também para as animações voltadas ao cinema, é a técnica chamada de *Cut-Out*, que é separar os personagens em pedaços, desenhando cada uma dessas partes uma única vez, e utilizando-as para fazer as animações, funcionando como uma marionete em 2D (Lima, 2016).

Para exemplificar essa técnica, considere um artista que deseja animar um robô humanoide. Primeiramente, ele desenhará cada parte do corpo separadamente, como mostrado na figura 7. Após isso, é necessário juntar cada uma dessas partes em articulações, essa técnica é chamada de *rigging*. Assim, permitindo o deslocamento e simulação de movimentos (Lamin, 2020).

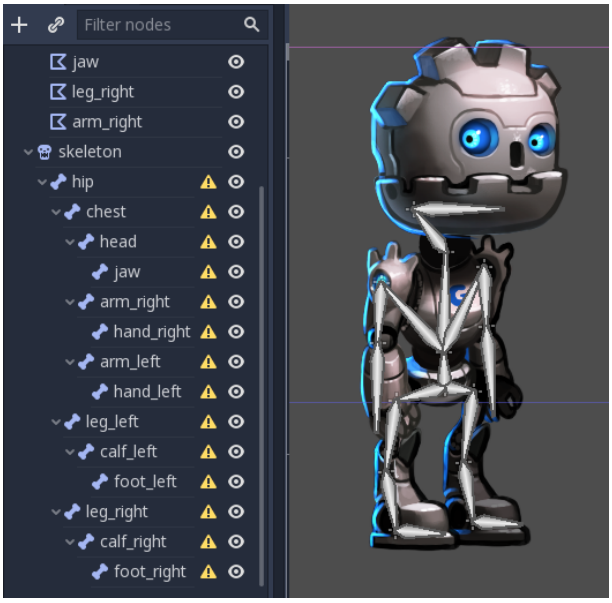
Figura 7 – Partes de um robô humanoide

Fonte: Godot Engine Documentation (2024a)

A técnica de "*rig*", segundo Simões (2023), são ossos virtuais que se ligam às partes do objeto que queira ser animado, representando a estrutura anatômica do mesmo. Isso permite que, quando uma parte do corpo se mova, todas as demais partes conectadas a ela se movam de forma sincronizada. No exemplo anterior, depois de ter feito o desenho das partes do personagem, ao aplicar a técnica de *rig*, associará os ossos a cada articulação do corpo, conforme mostra a figura

8. Dessa forma, ao mover, por exemplo, a mão do personagem, o restante do braço se articulará para acompanhar o movimento, eliminando a necessidade de redesenhar a posição exata do antebraço, braço e cotovelo em cada quadro.

Figura 8 – Finalização do processo de colocar ossos nas partes do robô humanoide



Fonte: Godot Engine Documentation (2024a)

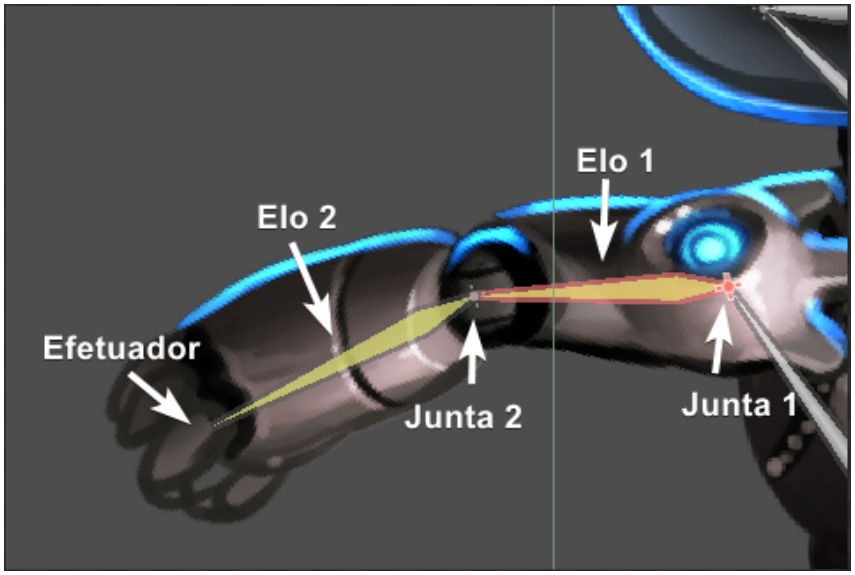
É válido ressaltar, que essa técnica confere uma estética particular aos personagens, deixando evidente o estilo 2D nos movimentos. Esse efeito cria uma ausência de profundidade tanto nos planos quanto na estrutura dos personagens, resultando em uma estética caraterística dessa forma de animação, com uma visualidade própria associada à técnica (Meyer, 2016).

Após a definição correta dos ossos, algoritmos de cinemática podem ser aplicados a esses elementos. O uso desses algoritmos torna a animação de posições e ações complexas mais eficiente e prática, como destacado por Pangesti, Utami e Sunyoto (2019). Sem a aplicação de cinemática, no exemplo anterior, no qual um artista deseja movimentar o braço do robô, ele precisaria ajustar manualmente os ângulos de todas as juntas que ligam os ossos do braço, até alcançar a posição desejada. No entanto, com o uso de algoritmos de cinemática, basta informar a posição final da mão, e os ângulos das juntas serão calculados e aplicados automaticamente.

A cinemática, amplamente utilizada na robótica, segundo Pangesti, Utami e Sunyoto (2019), refere-se ao estudo do movimento em sistemas articulados. Antes de abordar as equações da cinemática, é necessário entender as constantes e os termos envolvidos. No braço do robô humanoide tomado como exemplo, os segmentos do braço, antebraço e mão são chamados de elos, que são conectados por juntas. A mão, sendo o elo final e o ponto que executa tarefas, é conhecida como efetuador (Zacca *et al.*, 2020). Como é mostrado na figura 9.

Existem dois métodos de cinemática, com base no método de movimento, Cinemática Inversa (*Inverse Kinematics* - IK), e Cinemática Direta (*Forward Kinematics* - FK) (Pangesti; Utami; Sunyoto, 2019). A cinemática inversa lida com o mapeamento da posição e orientação do efetuador final, calculando os ângulos de cada junta, para que o efetuador alcance a posição de

Figura 9 – Elos e juntas do braço do robô humanoide



Fonte: Adaptação própria do exemplo de Godot Engine Documentation (2024a)

destino. Já a cinemática direta, mapeia todos os ângulos de junta para calcular a posição final do efetuador (Nugroho; Prihatmanto; Rohman, 2014). Em outras palavras, enquanto a IK calcula os ângulos necessários para que o efetuador atinja a posição desejada, a FK utiliza os ângulos das juntas já definidos para determinar a posição final do efetuador. Segundo Zacca *et al.* (2020), as equações da cinemática direta para um sistema de coordenadas 2D são:

$$X = L_1.cos\Theta_1 + L_2.cos(\Theta_1 + \Theta_2) + L_3.cos(\Theta_1 + \Theta_2 + \Theta_3) \tag{2.1}$$

$$Y = L_1.sen\Theta_1 + L_2.sen(\Theta_1 + \Theta_2) + L_3.sen(\Theta_1 + \Theta_2 + \Theta_3) \tag{2.2}$$

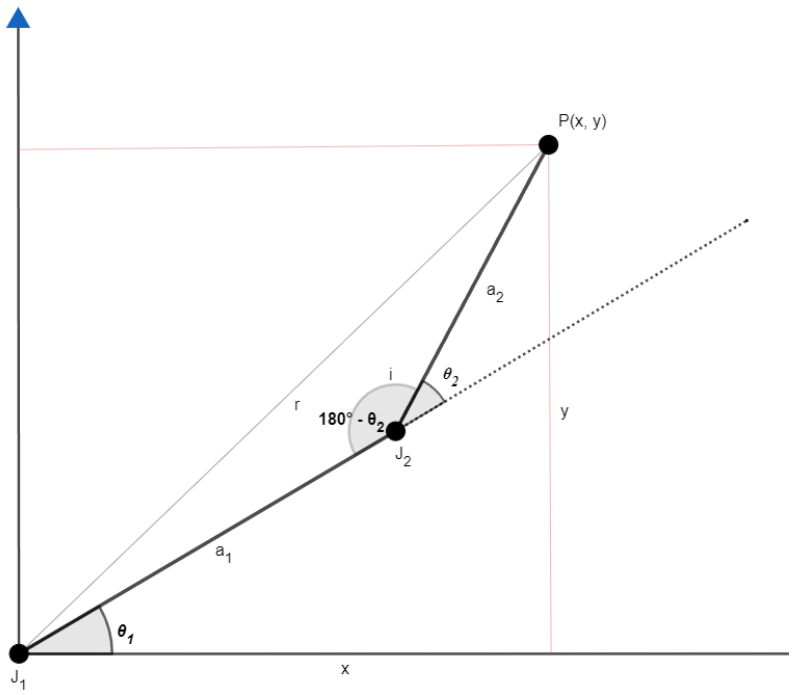
$$\phi = (\Theta_1 + \Theta_2 + \Theta_3) \tag{2.3}$$

As equações 2.1 e 2.2, retornam respectivamente, as coordenadas finais X e Y do efetuador. L_1, L_2 e L_3 são os comprimentos dos elos, Θ_1, Θ_2 e Θ_3 são os ângulos da junta 1, junta 2 e junta 3 respectivamente. Já, a equação 2.3, retorna o valor de ϕ, que representa a soma dos ângulos dos elos ao eixo x do plano cartesiano.

Para calcular a cinemática inversa, foi considerado um exemplo de um braço robótico composto por dois elos e duas juntas, como ilustrado na figura 10. A junta J_1 conecta o elo a_1 à base, que, neste caso, é a origem do plano cartesiano 2D. Já a junta J_2 conecta o elo a_2 ao elo a_1. O ponto $P(x,y)$ representa as coordenadas finais onde desejamos que o efetuador esteja após as transformações. Para atingir essa posição, é necessário calcular os ângulos θ_2 da junta J_2 e θ_1 da junta J_1.

Para derivar as equações utilizadas neste trabalho, foi tomado como base as equações apresentadas por Nunes (2016), que tratam da cinemática inversa de um braço robótico com três elos. No entanto, para adequação ao exemplo de um braço robótico com dois elos, foram feitas as devidas modificações com a ajuda do vídeo de Proyectos JC (2021), que mostra a demonstração matemática das equações da cinemática inversa.

Figura 10 – Exemplo de um braço robótico



Fonte: Adaptação própria inspirada em Proyectos JC (2021) e Nunes (2016).

O primeiro passo é determinar o ângulo θ_2 do braço robótico. Para isso, foi utilizado o ângulo interno i do triângulo obtuso[3] formado pelos vértices J_1, J_2 e o ponto $P(x,y)$. A relação entre esses ângulos é expressa na equação 2.4:

$$i = 180 - \theta_2 \tag{2.4}$$

Em seguida, foi aplicado o teorema de Pitágoras ao triângulo retângulo formado pelos segmentos r, x e y, onde r é a distância entre a junta J_1 do braço robótico e o ponto $P(x,y)$. O valor de r é dado pela equação 2.5. Além disso, o valor de r também pode ser obtido pela Lei dos Cossenos, aplicando-a ao triângulo obtuso formado pelos vértices J_1, J_2 e $P(x,y)$, como mostrado na equação 2.6.

$$r^2 = x^2 + y^2 \tag{2.5}$$

$$r^2 = a_1^2 + a_2^2 - 2.a_1.a_2.\cos(180 - \theta_2) \tag{2.6}$$

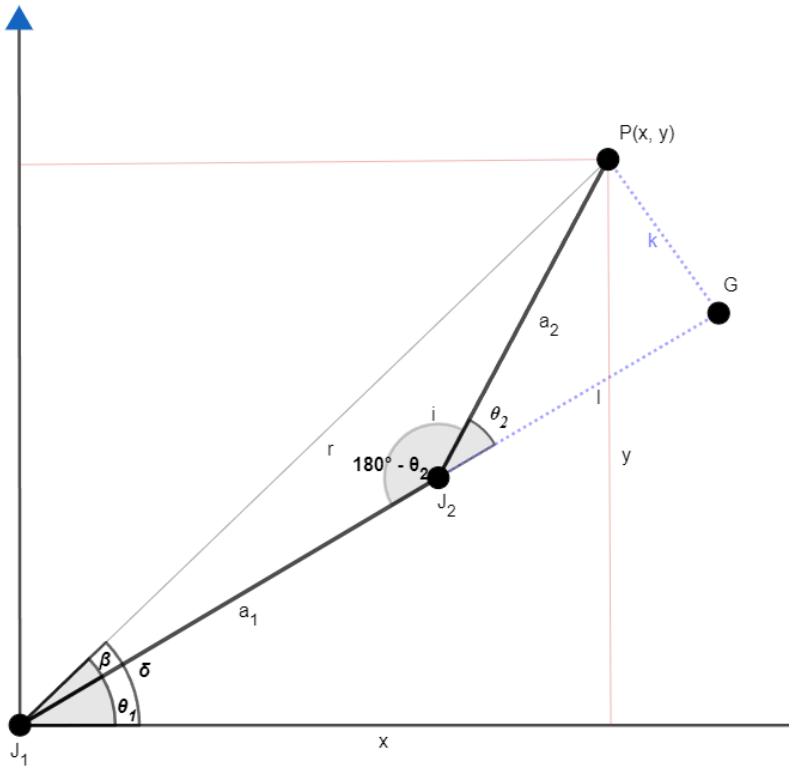
Ao igualar as equações 2.5 e 2.6, e utilizando a identidade trigonométrica $\cos(\pi - \theta) = -\cos(\theta)$, e após isso, isolando o ângulo θ_2. O resultado final é expresso pela equação 2.7:

$$\theta_2 = \pm \arccos\left(\frac{x^2 + y^2 - a_1^2 - a_2^2}{2.a_1.a_2}\right) \tag{2.7}$$

[3] Um triângulo é chamado de obtuso, quando este mesmo possui um ângulo interno maior que 90 graus e menor que 180 graus (Virtuous Tecnologia da Informação, 1998–2024).

Utilizando o valor de θ_2, o próximo passo é determinar o ângulo θ_1 por meio de relações trigonométricas simples, como ilustrado na figura 11.

Figura 11 – Associações de triângulos necessárias para o cálculo do ângulo θ_1



Fonte: Adaptação própria inspirada em Proyectos JC (2021) e Nunes (2016)

A relação entre θ_1, β e δ é estabelecida na equação 2.8. Para determinar θ_1, basta calcular os valores de β e δ, como mostra a equação:

$$\theta_1 = \delta - \beta \tag{2.8}$$

Para se encontrar o valor do ângulo δ, basta aplicar a função tangente no triângulo retângulo formado pelos segmentos x, y e r, conforme é mostrado na equação 2.9:

$$\delta = \arctan(\frac{y}{x}) \tag{2.9}$$

O cálculo de β é feito com base no triângulo retângulo formado pelos vértices J_1, G e $P(x,y)$. Sendo o cateto adjacente deste triângulo, o segmento formado por $a_1 + l$, e o cateto oposto, é formado pelo segmento k. Os valores de l e k são obtidos através do triângulo retângulo formado pelos vértices J_2, G e $P(x,y)$, como mostrados nas equações 2.10 e 2.11:

$$l = a_2 . \sin\theta_2 \tag{2.10}$$

$$k = a_2 . \cos\theta_2 \tag{2.11}$$

Com esses valores, β é calculado através da equação 2.12.

$$\beta = \arctan(\frac{a_2.\sin(\theta_2)}{a_1 + a_2.\cos(\theta_2)}) \tag{2.12}$$

Finalmente, ao substituir os valores de β e δ na equação 2.8, é chegado à forma final de θ_1 na equação 2.13:

$$\theta_1 = \arctan(\frac{y}{x}) - \arctan(\frac{a_2.\sin(\theta_2)}{a_1 + a_2.\cos(\theta_2)}) \tag{2.13}$$

2.3 Motor de jogo Godot

Para desenvolver qualquer tipo de jogo digital, é necessário contar com um conjunto de ferramentas que auxiliem na criação dos diversos sistemas que o compõem, como detecção de colisões, simulação física, renderização de gráficos, gerenciamento de recursos (imagens, fontes, sons, entre outros), entre outros aspectos fundamentais. Criar todas essas funcionalidades do zero demandaria tempo e esforço consideráveis. Para evitar essa reinvenção da roda, surgiram os motores de jogos, também conhecidos pelo termo em inglês *game engines*, que fornecem essas funcionalidades integradas em um ambiente de desenvolvimento apropriado.

Dentre os diversos motores disponíveis atualmente, destaca-se a Godot, uma *engine* gratuita, de código aberto, leve e altamente versátil. Licenciada sob a licença MIT, a Godot é mantida por uma comunidade ativa e independente ao redor do mundo (Barbosa; Massukado; Nakamura, 2021). Segundo a própria documentação oficial:

> Godot Engine é um motor de jogos multiplataforma repleto de recursos para criar jogos 2D e 3D a partir de uma interface unificada. Ele fornece um conjunto abrangente de ferramentas comuns para que os usuários possam se concentrar em criar jogos sem precisar reinventar a roda. Os jogos podem ser exportados num clique para várias plataformas, incluindo as principais plataformas de *desktop* (Linux, macOS, Windows), plataformas móveis (Android, iOS), as baseadas na Web e também consoles. (Godot Engine Documentation, 2025c)

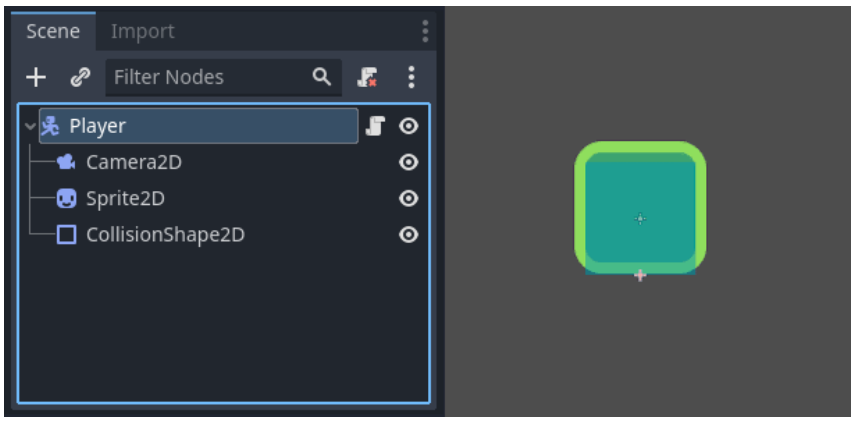
A Godot oferece suporte nativo a duas linguagens de programação principais: GDScript e C#. Para este projeto, optou-se por utilizar a linguagem GDScript, devido à sua simplicidade, integração nativa com o ambiente da Godot, e à familiaridade do autor com sua sintaxe e estrutura.

2.3.1 Cenas e Nodes e Árvore

Para a criação de um jogo na Godot, a estrutura fundamental é baseada nos conceitos de Cenas e *Nodes* (nós). Cada cena representa uma unidade funcional do jogo, podendo corresponder a um personagem, inimigo, objeto interativo, item decorativo ou até mesmo a interface gráfica do usuário. Essa flexibilidade permite ao desenvolvedor modelar qualquer elemento do jogo como uma cena, oferecendo ao mesmo uma estrutura modular e altamente reutilizável (Godot Engine Documentation, 2025h).

Dentro de cada cena, os elementos básicos são os *nodes*. Esses nós são unidades funcionais que desempenham papéis específicos, como exibir imagens, detectar colisões ou executar animações. Por exemplo, a estrutura de uma mesa pode ser composta por cinco *nodes* distintos: uma bancada e quatro pernas. Juntos, esses *nodes* formam a cena da mesa. Os *nodes* são organizados de forma hierárquica, criando uma estrutura em árvore (Godot Engine Documentation, 2025h). Na figura 12 é mostrado a composição da cena de um jogador dentro da Godot utilizando *nodes* básicos.

Figura 12 – Composição da cena de um jogador utilizando *nodes*



Fonte: Imagem tirada da documentação da Godot Engine Documentation (2025h)

O jogo completo é construído a partir da combinação e organização de múltiplas cenas, que são dispostas em uma estrutura hierárquica em forma de árvore, como ilustrado na figura 13. Cada ramo dessa árvore representa uma cena distinta, e cada cena, por sua vez, possui sua própria estrutura interna composta por *nodes*. Dessa forma, o jogo pode ser entendido como uma grande árvore formada pela interconexão de várias cenas, onde a Godot gerencia e executa cada uma delas de maneira integrada (Godot Engine Documentation, 2025h).

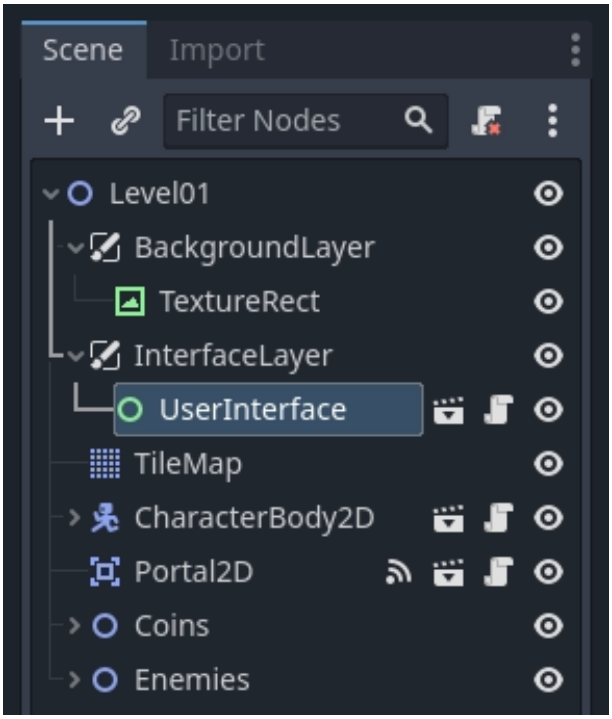
2.3.2 Nodes básicos disponibilizados pela Godot

A Godot disponibiliza uma grande variedade de *nodes* pré-definidos que servem como blocos de construção para diferentes tipos de jogos, sejam eles em 2D ou 3D. Esses *nodes* podem ser utilizados diretamente ou estendidos para criar comportamentos mais complexos (Godot Engine Documentation, 2025h). No desenvolvimento de jogos 2D, alguns dos *nodes* mais utilizados incluem: *Area2D*, *CollisionShape2D*, *CharacterBody2D*, *StaticBody2D*, *RigidBody2D*, *Sprite2D*, e *AnimationPlayer*, entre outros. A seguir tem uma descrição mais detalhadas dos principais *nodes* utilizado nesse projeto:

2.3.2.1 Area2D

Durante o desenvolvimento de jogos, a detecção de colisões entre objetos é um dos aspectos mais fundamentais. Para isso, a Godot oferece diversos *nodes* especializados nesse

Figura 13 – Composição da cena de um nível de jogo utilizando cenas e *nodes*



Fonte: Imagem tirada da documentação da Godot Engine Documentation (2025h)

tipo de interação, sendo um dos *nodes* que podem ser utilizados para a detectação de colisão o *Area2D* (Godot Engine Documentation, 2025e).

Segundo Godot Engine Documentation (2025e), o *node Area2D* tem como objetivo definir uma região no espaço 2D com o objetivo de detectar quando outros objetos entram e sai dessa área. Ele pode detectar tanto outras instâncias de *Area2D* quanto corpos físicos, como *RigidBody2D* e *CharacterBody2D*, sem necessariamente gerar uma resposta física à colisão.

Esse recurso é ideal para situações em que é necessário apenas identificar a sobreposição entre objetos, sem provocar reações físicas. Um exemplo comum é o de itens colecionáveis no cenário: quando o jogador passa sobre um item no chão, o jogo precisa detectar a sobreposição para permitir a coleta, mas sem interferir no movimento do jogador ou aplicar forças físicas. Nesse caso, o item pode ser representado por um *Area2D* combinado com uma textura visual.

2.3.2.2 CharacterBody2D

O *CharacterBody2D* é um *node* voltado para o controle de personagens e outras entidades que requerem movimentação personalizada por código. Ele realiza detecção de colisões com outros objetos, mas não responde automaticamente às leis da física da Godot, como forças, aceleração ou rebote. Em vez disso, cabe ao desenvolvedor definir manualmente esses comportamentos, oferecendo maior controle sobre a lógica de movimentação e interação (Godot Engine Documentation, 2025f).

Esse *node* é especialmente útil para implementar personagens jogáveis, inimigos ou qualquer entidade com movimentação ou comportamento específicos, que não dependem da

física automatizada da Godot. No contexto deste projeto, o *CharacterBody2D* foi utilizado tanto para o personagem principal quanto para as entidades do jogo, permitindo que seus comportamentos fossem programados de forma personalizada de acordo com as necessidades do *gameplay*.

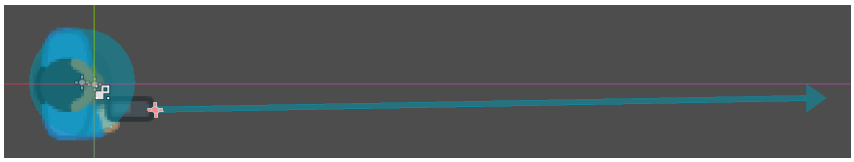
2.3.2.3 Raycast2D

O *node RayCast2D* é utilizado para detectar colisões ao longo de uma linha imaginária, traçada a partir de sua origem até um ponto final definido. Ele retorna informações sobre o primeiro objeto que intersecta esse raio, sem provocar qualquer efeito físico nos elementos detectados (Godot Engine Documentation, 2025d).

Assim como o *Area2D*, o *RayCast2D* serve apenas para detecção, sendo ideal para situações em que é necessário verificar a presença de objetos ao longo de um caminho sem aplicar interações físicas automáticas. Um exemplo de uso dessa técnica é na simulação de disparos instantâneos, especialmente para evitar o problema de "tunelamento"— uma falha comum em jogos quando projéteis muito rápidos atravessam objetos sem detectar colisões (KidsCanCode, 2025a).

Para contornar esse problema, o *RayCast2D* pode ser utilizado para representar o caminho de um projétil, detectando imediatamente qualquer obstáculo ou inimigo no trajeto. Uma vez identificada uma colisão, o desenvolvedor pode programar os efeitos desejados, como causar dano ou exibir partículas de impacto (KidsCanCode, 2025a), conforme ilustrado na Figura 14.

Figura 14 – Exemplo da cena de um jogador atirando com um RayCast2D



Fonte: Imagem tirada da documentação no KidsCanCode (2025a)

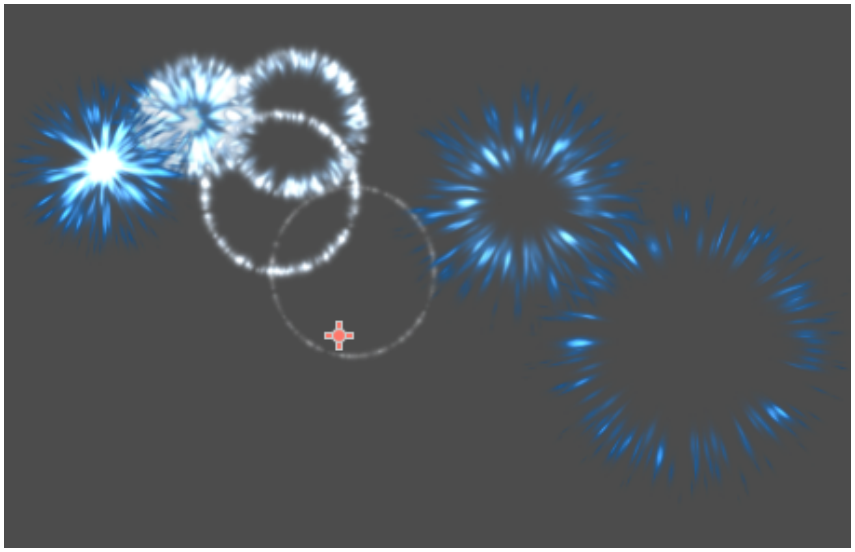
2.3.2.4 Sistema de partículas 2D

O sistema de partículas 2D da Godot permite a criação de partículas 2D altamente configuráveis, permitindo que o desenvolvedor crie diversos efeitos visuais por meio da emissão dessas pequenas unidades. Essas partículas são geradas em intervalos regulares e possuem um tempo de vida limitado. Durante esse período, elas seguem comportamentos previamente definidos, que podem ser variados por meio da introdução de elementos aleatórios, gerando um resultado mais natural e orgânico, dessa forma um sistema de partículas basicamente consiste em configurar propriedades físicas fundamentais e aplicar variações aleatórias para gerar comportamentos visuais dinâmicos (Godot Engine Documentation, 2025a).

Esse sistema é amplamente utilizado para simular fenômenos visuais como fogo, fumaça, faíscas, efeitos mágicos, entre outros (Godot Engine Documentation, 2025a). Por exemplo,

para criar um efeito de chama, pode-se aplicar uma textura base, ajustar a gravidade para cima, modificar a cor das partículas ao longo do tempo (passando de tons claros para escuros), entre outras configurações específicas que simulam o comportamento do fogo de forma convincente. Na figura 15 é mostrado um dos exemplos de uso do sistema, que cria efeito de fogos de artifícios.

Figura 15 – Exemplo de uso do sistema de partículas para criação de efeito de fogos de artifícios



Fonte: Imagem tirada da documentação da Godot Engine Documentation (2025h)

A Godot disponibiliza dois tipos de *nodes* para trabalhar com partículas em 2D: *GPUParticles2D* e *CPUParticles2D*. O *GPUParticles2D* é mais eficiente para manipular grandes volumes de partículas, pois utiliza a GPU do dispositivo. Já o *CPUParticles2D* realiza o processamento na CPU, sendo mais adequado para dispositivos com capacidades gráficas limitadas. Embora o desempenho do *CPUParticles2D* seja inferior em cenários com muitas partículas, ele pode ser mais estável em plataformas com menor poder de processamento (Godot Engine Documentation, 2025a). Por esse motivo, este projeto opta pela utilização do *CPUParticles2D*, garantindo melhor compatibilidade e desempenho em dispositivos mais modestos.

2.3.2.5 TileMap

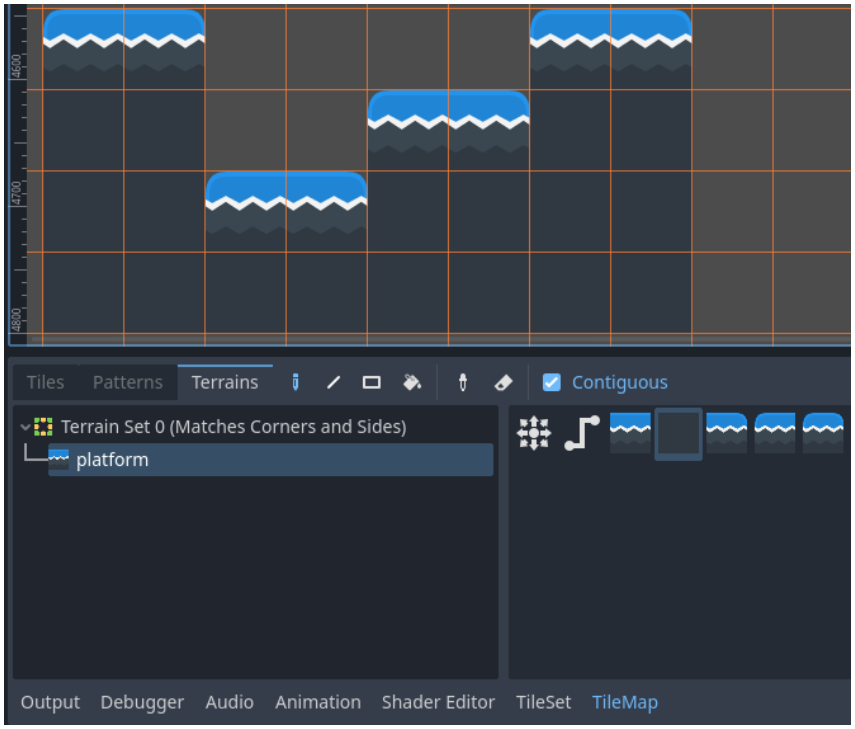
O *TileMap* é um *node* fornecido pela Godot que permite a criação de uma grade composta por texturas chamadas *tiles*. Com ele, o desenvolvedor pode "desenhar" diretamente sobre essa grade utilizando qualquer *tile* previamente configurado, facilitando a construção de grandes níveis de forma prática e organizada. Além disso, o *node TileMap* conta com otimizações próprias que garantem bom desempenho mesmo com uma grande quantidade de *tiles* desenhados, não impondo limitações significativas ao tamanho dos mapas (Godot Engine Documentation, 2025g).

Além da funcionalidade básica de desenho, o *TileMap* oferece diversos recursos adicionais que auxiliam o desenvolvimento de jogos, como a possibilidade de incluir *tiles* animados, adicionar colisão diretamente nos *tiles*, gerar malhas de navegação, e utilizar *tiles* automáticos (Godot Engine Documentation, 2025g). Com os *tiles* automáticos, o desenvolvedor precisa

apenas definir a área desejada, e o *TileMap* posiciona automaticamente os *tiles* apropriados, como cantos e bordas, de acordo com o contexto.

Neste projeto, o *TileMap* foi utilizado na criação da animação tradicional de água e também na construção dos mapas, tanto os tradicionais quanto os procedurais, desenhando superfícies como chão, paredes e tetos. Na figura 16 é mostrado um exemplo prático do uso do *TileMap*.

Figura 16 – Exemplo de uso do *node TileMap*



Fonte: Imagem tirada da documentação da Godot Engine Documentation (2025g)

2.3.3 Shaders na Godot

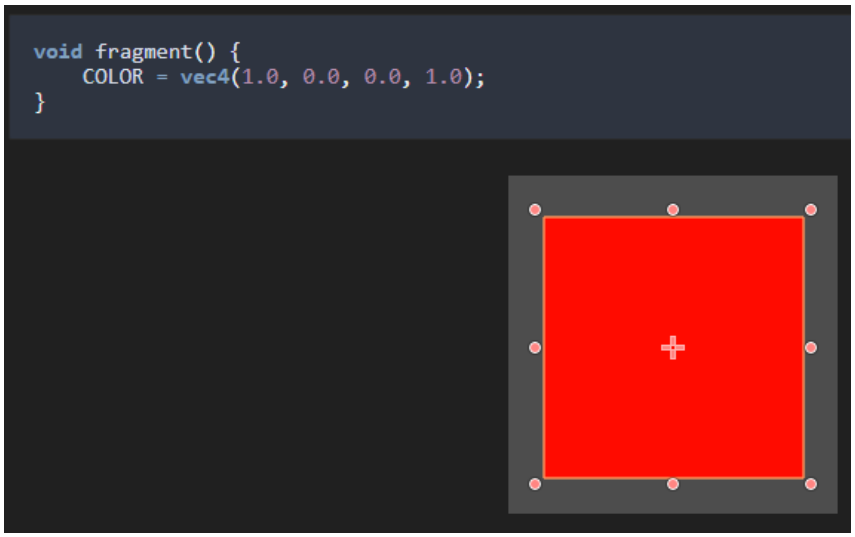
Shaders são programas especiais executados diretamente na GPU dos dispositivos. Com eles, é possível instruir a GPU sobre como renderizar uma imagem, possibilitando a criação de uma ampla variedade de efeitos visuais (Godot Engine Documentation, 2025b).

Inicialmente, *shaders* eram utilizados principalmente para o sombreamento em cenas 3D. Atualmente, no entanto, é possível aplicar *scripts* de *shader* a qualquer elemento que reproduza imagens, permitindo controle detalhado sobre como o motor de jogo desenha cada imagem, incluindo seus *pixels* e geometria (Godot Engine Documentation, 2025b).

Diferente de programas comuns executados na CPU, onde as instruções são processadas sequencialmente e o estado pode ser compartilhado entre diferentes partes do código, os *shaders* funcionam de forma paralela. Isso significa que o código é executado individualmente para cada *pixel* da imagem, de forma simultânea. Cada execução é isolada, sem armazenamento de estado ou troca de informações entre os *pixels*. Essa característica exige uma nova forma de pensar a programação e o desenvolvimento de efeitos visuais (Godot Engine Documentation, 2025b).

A GPU é altamente otimizada para esse tipo de operação, permitindo que os *shaders* sejam processados com grande eficiência. Em contrapartida, o acesso a informações em tempo de execução é limitado. Na prática, uma das saídas de um programa *shader* é a cor final de um *pixel* na imagem ou objeto sendo renderizado (KidsCanCode, 2025b). Por exemplo, se o desenvolvedor deseja que toda a imagem seja exibida em vermelho, basta retornar a cor vermelha na função *fragment* do programa *shader*, como ilustrado na figura 17.

Figura 17 – *Shader* de fragmento deixando uma imagem com todos os *pixels* vermelhos



Fonte: Imagem tirada da documentação da KidsCanCode (2025b)

3 METODOLOGIA

Esta seção é dedicada a apresentação dos procedimentos metodológicos empregados para a aplicação de animações tradicionais e procedurais em jogos de plataforma em 2D, e os procedimentos para a realização dos experimentos comparativos entre as técnicas de cada tipo de animação. São descritos os métodos, as técnicas e as ferramentas escolhidas para o desenvolvimento e análise qualitativa e quantitativa das animações, visando assegurar uma comparação embasada entre as abordagens. Além disso, são abordadas as estratégias de coleta de dados sobre desempenho e custo de implementação, considerando os cenários específicos em que cada técnica é aplicada, a fim de avaliar as vantagens e limitações de ambas no contexto de desenvolvimento de jogos com recursos limitados.

Neste trabalho, é proposto a metodologia que desenvolverá elementos de um jogo utilizando tanto técnicas de animação procedural quanto técnicas de animação tradicional, com foco em aplicações práticas e na comparação entre os métodos. Nessas condições, esse trabalho caracteriza-se como uma pesquisa experimental, pois, de acordo com Wazlawick (2020), a pesquisa experimental consiste em introduzir uma nova técnica ou método em um ambiente controlado, permitindo ao pesquisador observar e medir os efeitos produzidos por essa intervenção. No contexto deste projeto, isso significa utilizar abordagens distintas de animação e mensurar seus impactos em termos de desempenho, consumo de recursos, diminuição de custos, complexidade de implementação e aumento de realismo, oferecendo assim, uma compreensão mais sólida sobre as capacidades e limitações de cada método.

Além disso, esta pesquisa é caracterizada como primária, pois produz conhecimento original a partir de dados obtidos por meio de testes práticos, pois, segundo Wazlawick (2020), a pesquisa primária tem como base a exposição de novos conhecimentos a partir de observações. No presente estudo, isso se reflete na realização de testes de estresse que possibilitam uma comparação direta de desempenho entre os dois métodos de animação utilizados.

Em busca de conhecer o atual estado da arte sobre técnicas de animações procedurais e tradicionais que possam ser aplicadas em jogos de plataforma 2D, o primeiro passo consiste na pesquisa no estado da arte nas plataformas de publicação científica: Google Acadêmico, IEEE, SBC. Esse levantamento se estende a livros com temas de computação gráfica e programação de jogos. Seguindo assim, a definição dada por Wazlawick (2020) sobre pesquisa bibliográfica, no qual fala que a mesma consiste no levantamento e estudo de fontes publicadas como livros e artigos.

As *strings* de busca utilizadas abrangeram termos em português e em inglês, combinando variações dos temas centrais. Foram utilizadas expressões como:

- *"Procedural Animation"*, incluindo variações como *"Procedural Animation 2D Games"* e *"Procedural Animation 2D Survey"*, além das versões em português como "Animação Procedural" e "Animação Procedural em jogos 2D".

- Termos relacionados a jogos de plataforma 2D e técnicas de animação procedural e tradicional, como “Cinemática Inversa”, “Cinemática Direta”, *“Inverse Kinematic”*, *“Inverse Forward”*, “Jogo de plataforma 2D”, “Geração Procedural de Conteúdo”, “Animação em jogos 2D”.

Inicialmente, a pesquisa foi configurada para abranger intervalos de tempo de cinco anos. No entanto, diante da escassez de artigos diretamente relevantes nos resultados mais recentes, o intervalo temporal foi gradualmente ampliado, de forma a incluir publicações mais antigas que contribuíssem para os objetivos deste trabalho.

3.1 Elementos

Neste trabalho, foram desenvolvidos elementos que são *bits* de um jogo com o objetivo de aplicar e comparar técnicas de animação tradicional e procedural. Os elementos escolhidos foram: o jogador, controlado pelos usuários; a aranha, uma entidade inimiga que se movimenta de forma autônoma por meio de inteligência artificial; a água, um elemento de cenário comumente presente em jogos; o fogo, utilizado tanto em cenários quanto em mecânicas; e, por fim, o tecido, que contribui para a estética visual e oferece potencial para diversas interações e mecânicas.

Para cada um desses elementos (com exceção do jogador), foram elaboradas duas versões distintas: a primeira utiliza técnicas tradicionais de animação, enquanto a segunda emprega animações geradas proceduralmente. Essa abordagem permite uma comparação direta entre os dois métodos. A programação dos elementos busca explorar ao máximo as características e vantagens de cada técnica, resultando em implementações distintas, mas com os mesmos objetivos funcionais.

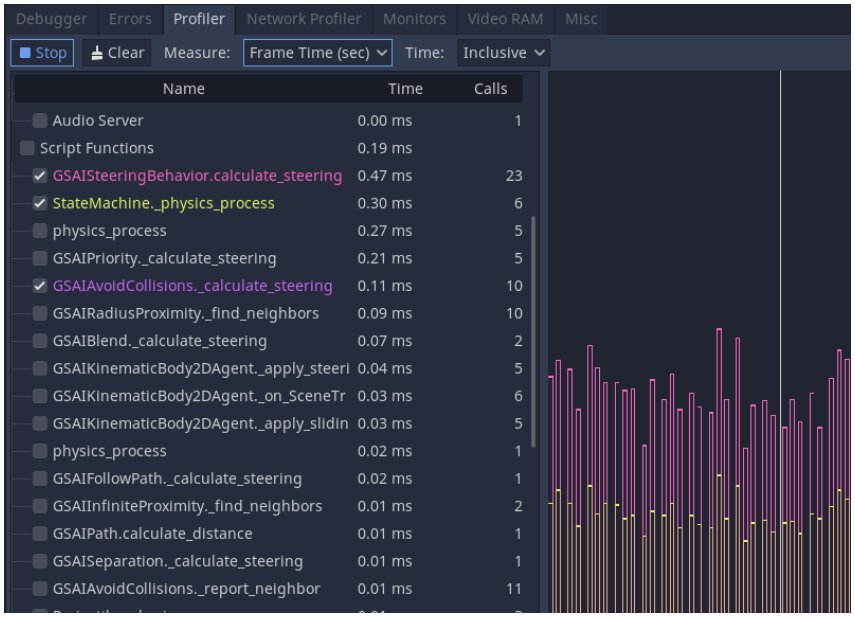
O jogador por sua vez será um elemento que estará presente tanto nas cenas procedurais quanto tradicionais, então incluirá ambas as técnicas. Também vale destacar que efeitos sonoros e trilha sonora não foram incluídos nas cenas finais, uma vez que este trabalho não contempla aspectos relacionados à sonorização de jogos em seu escopo.

Todo o protótipo foi feito utilizando o motor de jogos Godot, escolhido devido à familiaridade do autor com essa ferramenta. As animações tradicionais e os desenhos base para as animações procedurais deste projeto foram desenhados pelo próprio autor deste trabalho, utilizando a técnica de *pixel art*, que segundo Kopf e Lischinski (2011), é uma arte digital que se caracteriza por representar os detalhes das imagens no nível de *pixel*. Essa abordagem foi escolhida principalmente por dois motivos: em primeiro lugar, devido à familiaridade do autor com essa técnica de ilustração; e em segundo, porque o *pixel art* é amplamente adotado por desenvolvedores independentes, sendo uma técnica de desenho menos trabalhosa do que ilustrações 2D em alta resolução e, ao mesmo tempo, permitindo a criação de visuais marcantes e únicos. Assim, essa escolha está alinhada com a estética comumente adotada por projetos independentes.

3.2 Análise do desempenho de cada abordagem

O motor Godot oferece diversas ferramentas de depuração que permitem a análise detalhada do desempenho do jogo, incluindo o monitoramento do uso de memória RAM, CPU e GPU. Além disso, o Godot possui *profilers* para o acompanhamento do tempo de execução de cada função presente no projeto, permitindo detectar funções lentas e *bugs*[1], como é mostrado na figura 18, além de opções para visualização de caixas de colisão, além de polígonos de navegação no ambiente do jogo, proporcionando uma visão abrangente dos aspectos técnicos da aplicação (Godot Engine Documentation, 2024b).

Figura 18 – Visualizador de *profiler* do Godot



Fonte: Imagem retirada do site da Godot Engine Documentation (2024c)

A partir dessas ferramentas, foi conduzida uma análise de desempenho das diferentes abordagens de animação, por meio de testes em *hardware* de computadores *desktop* e *notebooks* de várias gerações, assim como em dispositivos móveis. Os testes incluíram cenários de estresse, aumentando gradualmente o número de entidades e efeitos na tela, com o objetivo de avaliar a estabilidade e a eficiência de cada técnica sob diferentes cargas de trabalho.

Os resultados coletados foram comparados entre as abordagens tradicionais e procedurais, considerando aspectos como fluidez das animações, impacto visual, e consumo de recursos. Os dados obtidos com o *profiler* do Godot permitiram uma avaliação quantitativa do impacto de cada técnica de animação no processamento, possibilitando identificar quais métodos são mais eficientes em termos de uso de CPU e GPU. Essa análise é relevante para otimizar o desempenho em dispositivos de menor capacidade, como *notebooks* mais antigos e dispositivos móveis, onde recursos limitados exigem técnicas de animação que equilibrem qualidade visual e uso de *hardware*.

[1] A palavra *bug* é um jargão da informática que se refere a erros em um programa de computador (Ribeiro, 2024).

3.3 Questões de pesquisa

Para guiar a análise comparativa entre as animações tradicionais e procedurais, foram definidas questões de pesquisa fundamentais que exploram os contextos, impactos e características específicas de cada método de animação. Ao final do trabalho, elas serão respondidas com base nos dados e observações feitas durante a etapa experimental. As questões de pesquisa são:

- **QP1**: Em quais contextos cada método de animação (tradicional e procedural) pode proporcionar uma redução nos custos de trabalho humano, e quais são suas principais vantagens e desvantagens em termos de qualidade visual, flexibilidade e complexidade de implementação?
- **QP2**: Qual é o impacto do desempenho computacional de cada método e suas técnicas específicas em diferentes tipos de *hardware*?
- **QP3** Em que medida cada método de animação impacta a jogabilidade e a experiência do usuário em jogos de plataforma 2D, e em quais situações as animações procedurais podem contribuir para aumentar o realismo e a responsividade das interações em tempo real, em comparação com as animações tradicionais?

3.4 Planejamento do projeto

Como o projeto tem como objetivo comparar duas abordagens de animação - uma baseada em técnicas procedurais e outra utilizando animação tradicional -, optou-se por iniciar o desenvolvimento pela cena procedural. Essa decisão se deve à maior complexidade envolvida na implementação desse tipo de animação. Após a finalização da cena procedural, a versão tradicional foi planejada e executada.

O planejamento foi documentado no Documento de *Design* de Jogo (*Game Design Document* - GDD), no qual é definido por Rogers (2013) como um documento detalhado que descreve todos os aspectos presentes no jogo. Com base nessa definição, foram registrados os elementos a serem criados e seus respectivos comportamentos. Para garantir que a animação procedural fosse aplicada ao maior número possível de elementos distintos, foram definidos cinco tipos principais: Jogador, Aranha, Água, Fogo e Tecido.

Abaixo, detalha-se o propósito de cada elemento dentro do projeto:

- O jogador: Como personagem principal e controlável pelo usuário, o jogador permite a interação ativa com os elementos do cenário. Além de contribuir para a navegação pelo ambiente, ele possibilita testar a resposta das animações procedurais em tempo real.
- A aranha: Escolhida como exemplo de criatura animada proceduralmente, a aranha servirá para demonstrar a aplicação dessas técnicas em inimigos movidos por inteligência artificial. A intenção é criar um comportamento dinâmico e realista, explorando a movimentação orgânica e adaptável.

- Água: A água é um elemento comum em jogos 2D e 3D, muitas vezes apresentando comportamento dinâmico e interativo. Neste projeto, a implementação procedural visa proporcionar fluidez e reatividade, permitindo que o jogador e os inimigos interajam com o líquido de maneira natural.
- Fogo: Além de ser um elemento visualmente impactante, o fogo pode desempenhar tanto um papel ambiental quanto mecânico, servindo como obstáculo ou arma. Sua animação procedural permitirá criar efeitos dinâmicos que variam conforme o ambiente e as interações dentro do jogo.
- Tecido: Elementos de fundo são essenciais para a ambientação e imersão do jogador. A escolha de tecidos como elementos animados proceduralmente visa demonstrar a flexibilidade dessas técnicas na simulação de objetos maleáveis, podendo ser utilizados tanto para efeitos visuais quanto para mecânicas de *gameplay* em jogos de plataforma.

Todos os elementos planejados para a cena procedural também foram replicados na cena com animações tradicionais, garantindo uma comparação direta entre as duas abordagens. Para isso, foram feitas as devidas adaptações e ajustes necessários, respeitando as limitações e características próprias desse estilo de animação, de modo a preservar a fidelidade visual e comportamental dos elementos em ambas as versões.

3.4.1 GDD Procedural

Nesta seção, serão detalhados o planejamento e a concepção dos objetos presentes na cena procedural. Serão descritas suas interações, limitações e propósitos, destacando como cada elemento foi projetado para se comportar dinamicamente dentro do jogo.

3.4.1.1 Jogador

Foi definido que o jogador seria um personagem de plataforma clássico, permitindo que o autor do projeto o controlasse para interagir com o ambiente ao seu redor. Como parte dessa interação, ele possui uma arma capaz de disparar projéteis que reagem ao cenário, adicionando uma mecânica eficaz para testar as interações.

Os movimentos disponíveis para o jogador incluem: andar para a direita e esquerda, pular, nadar, mirar a arma com o cursor do *mouse* e atirar, sendo possível disparar enquanto executa qualquer um desses movimentos.

Dado que o personagem utiliza uma arma, foi decidido que seu *design* seguiria a estética de um soldado. Suas animações adotam uma abordagem semi-procedural, onde as pernas, o torso e a cabeça são animados de maneira tradicional, enquanto os braços são controlados proceduralmente, ajustando-se dinamicamente para apontar sempre na direção do cursor do mouse. Essa técnica garante que o autor do projeto possa testar interações com o cenário de

vários ângulos e posições, independente da posição ou movimento do jogador, além de manter a estética do personagem, que parecerá estar constantemente mirando.

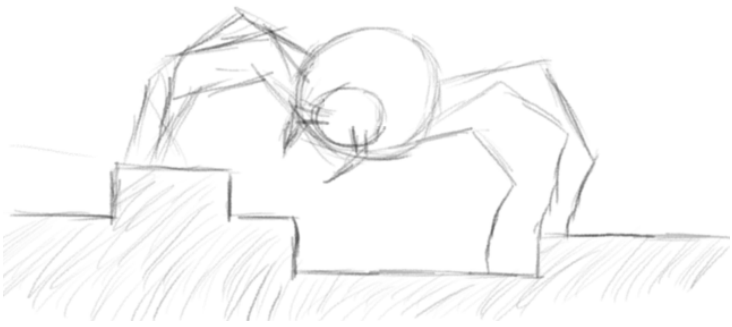
Como o jogador serve essencialmente como um elemento de controle para o autor do projeto, ele será mantido idêntico tanto na cena procedural quanto na cena animada tradicionalmente. Essa decisão se deve ao fato de que o personagem combina características das duas abordagens e ao diferencial de poder atirar em qualquer direção, tornando sua mecânica consistente e adaptável para ambos os cenários.

3.4.1.2 Aranha

No GDD, definiu-se que a estrutura da aranha seria composta por um corpo que é estruturado em duas partes: a cabeça junto do corpo, e seis pernas ligadas ao corpo, que por outrora são segmentadas em dois grupos: três pernas do lado direito e três do lado esquerdo. Cada perna é segmentada em três partes interconectadas, formando uma cadeia articulada em que uma das extremidades se liga ao corpo e a outra atua como o efetuador da aranha. Essa divisão permite uma animação mais dinâmica e realista, garantindo que cada conjunto de pernas se movam de forma sincronizada e coerente com a locomoção da criatura.

Quanto ao comportamento da aranha, ela poderá executar os seguintes movimentos: permanecer parada, andar para a direita e para a esquerda, além de escalar paredes e tetos, sempre ajustando sua orientação conforme a superfície em que se encontra. Isso significa que, ao escalar uma parede vertical, seu corpo ficará alinhado perpendicularmente ao solo, e ao caminhar pelo teto, ela adotará uma posição invertida, ficando de cabeça para baixo.

Figura 19 – Esboço de como a aranha foi planejada para se comportar em superfícies



Fonte: Imagem criada pelo próprio autor deste trabalho

Durante o deslocamento, a aranha manterá uma altura pré definida em relação à superfície abaixo dela. Quando parada, essa altura será ajustada para uma posição ligeiramente mais baixa, criando um efeito de levantamento ao caminhar e uma postura mais relaxada ao repousar. Esse detalhe contribui para um comportamento mais natural e crível da criatura.

Por fim, foi estabelecido que o movimento das pernas seguirá um ritmo realista, inspirado no deslocamento de aranhas reais. Além disso, as pernas deverão se adaptar ao terreno, permitindo

que a criatura caminhe suavemente sobre superfícies de diferentes formatos, sejam elas planas, inclinadas ou irregulares. Na figura 19 mostra o esboço feito da aranha de como ela foi planejada para se comportar nas superfícies.

3.4.1.3 Água

A água procedural foi planejada em seu GDD para preencher determinadas áreas do mapa, como grandes buracos, formando pequenos lagos e lagoas onde o personagem poderá entrar e nadar para atravessar.

A superfície da água reagirá dinamicamente a colisões causadas por fatores externos, como a entrada e saída do jogador, projéteis disparados por sua arma e a interação da aranha com a água. A água será deslocada no sentido das colisões ocorridas, propagando ondas em todas as direções tendo como epicentro o local da colisão. Por exemplo, se o jogador cair na água de cima para baixo, a água será deslocada para baixo; e se ele emergir, a água será deslocada para cima. Na figura 20 é mostrado o esboço de como a água foi planejada para se comportar quando algo impacta nela.

Figura 20 – Esboço de como a água foi planejada para se comportar quando algo impactar nela

Fonte: Imagem criada pelo próprio autor deste trabalho

A textura da água terá um nível de transparência que permitirá visualizar elementos submersos. Além disso, foi definido que objetos dentro da água apresentarão um efeito visual característico, reforçando a sensação de imersão e a interação do ambiente com os elementos do jogo.

3.4.1.4 Fogo

No GDD, o fogo foi planejado como um elemento puramente visual, sem efeitos diretos sobre outros elementos, como dano ou interação física. Apesar de esse tipo de funcionalidade ser comum em jogos, optou-se por essa abordagem devido a limitações de tempo e prioridades do projeto. Sua estética foi planejada para ser composta por partículas que iniciarão em tons vibrantes tendendo ao branco e, à medida que se dissipam, transitarão para tons mais escuros de laranja e vermelho, simulando a aparência de fogo real.

Além disso, o fogo contará com um efeito de fumaça que se elevará até uma determinada altura antes de se dispersar no ar. A fumaça seguirá uma direção específica, simulando a influência do vento, tornando o efeito mais dinâmico e natural.

O sistema de fogo também permitirá escalabilidade, podendo ser ajustado para diferentes tamanhos - pequenos, médios e grandes - sem comprometer a qualidade visual.

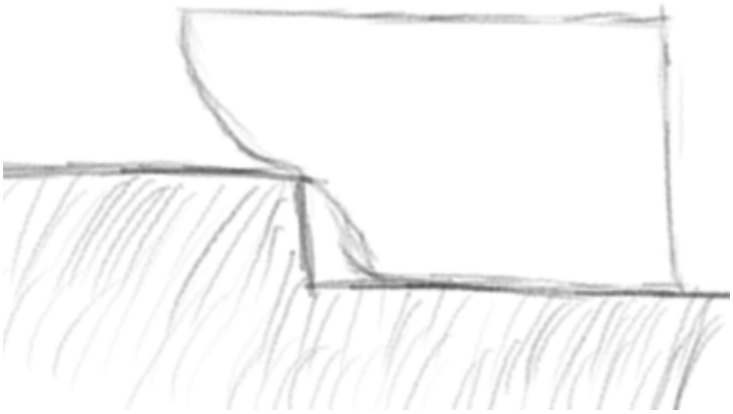
3.4.1.5 Tecido

O tecido foi planejado como um elemento decorativo que também interage com os objetos da cena, reagindo a colisões com o jogador, suas balas, a aranha e o ambiente ao redor.

Quando qualquer um desses elementos colidir com o tecido, ele se deforma de acordo com a direção da colisão. Por exemplo, se o jogador estiver andando para a direita e entrar em contato com o tecido, este se dobrará nessa direção, simulando um efeito de empurrão. O mesmo comportamento será aplicado para impactos causados por balas ou pela movimentação da aranha.

Além disso, o tecido se ajustará automaticamente às superfícies com as quais entra em contato, como o chão ou outras estruturas do cenário, moldando-se conforme a topografia do terreno. Por fim, foi definido que o tecido não terá interações com elementos como fogo ou água, mantendo seu comportamento restrito às colisões físicas com personagens e o ambiente sólido. Na figura 21 é mostrado o esboço do tecido em colisão com superfícies.

Figura 21 – Esboço de como o tecido foi planejado para se comportar com a colisão com superfícies



Fonte: Imagem criada pelo próprio autor deste trabalho

3.4.2 GDD Animações Tradicionais

O planejamento dos itens com animações tradicionais seguiu uma linha de adaptação dos elementos desenvolvidos para a versão procedural. Como a abordagem tradicional reduz

significantemente o fator de interatividade, a maioria dos objetos passou a cumprir um papel essencialmente visual, com menos reações dinâmicas.

A aranha animada tradicionalmente foi planejada para simular o comportamento da versão procedural por meio de animações específicas: andando para a direita e esquerda, pulando e caindo. Como não é possível animar as pernas de forma dinâmica nessa abordagem, optou-se por cenários mais planos. No entanto, para fins de comparação com a versão procedural, foram inseridos obstáculos que exigem que a aranha pule ou suba, acompanhados de suas respectivas animações para cada estado.

No caso da água, foi planejada uma animação de ondas em uma sequência repetida, dividida em duas partes principais: a borda da água, que terá movimento ondulante visível com uma coloração diferente para facilitar sua identificação, e a parte inferior, sem as bordas de cima, representando a profundidade da água. Utilizando essas animações para preencher o cenário, pretende-se criar uma ilusão de água semelhante à da versão procedural. Contudo, essa versão não oferece interatividade com o jogador, sendo apenas visual. Mas ainda assim, o jogador poderá entrar na área da água, reconhecendo-a como tal.

O fogo, assim como na versão procedural, foi projetado apenas como um elemento visual, sem interagir com outros objetos e elementos. Foi planejada uma animação em repetição, com partículas que iniciam em tons claros e vibrantes e terminam em tons mais escuros, além de uma fumaça animada para reforçar o efeito visual semelhante ao procedural.

O tecido, por fim, diferentemente da versão procedural, foi concebido na abordagem tradicional apenas como um elemento decorativo. Projetar um tipo de interação com tecidos utilizando animações tradicionais requer um tempo considerável e um número significativo de estados para alcançar um nível de interatividade semelhante ao da versão procedural. Por esse motivo, optou-se por mantê-lo puramente visual, sem interações. A animação de tecido planejada foi a de uma bandeira, por ser facilmente reutilizável em diversos cenários distintos de jogos.

3.4.3 Planejamento dos testes

Os testes de estresse foram planejados com o objetivo de avaliar separadamente o desempenho das animações tradicionais e procedurais, com cada tipo de objeto passando por uma série específica de experimentos.

Foi definido que cada objeto seria submetido a três níveis distintos de teste, variando a quantidade de instâncias instanciadas. No primeiro nível, será utilizada uma quantidade mínima de instâncias, definida de forma empírica. No segundo nível, uma quantidade intermediária, e no terceiro nível, o número máximo de instâncias que a máquina mais potente disponível ao autor suporta, que neste caso se trata de um computador *desktop* equipado com um processador Ryzen 5 4600G e 32 GB de memória RAM DDR4.

Além dos testes individuais, tanto para os itens com animação procedural quanto para os com animação tradicional, será realizado um teste combinado. Nesse teste especial, todos os objetos dos testes anteriores serão instanciados simultaneamente, permitindo observar o impacto

do uso simultâneo de diferentes elementos no desempenho geral da aplicação. Este teste também contará com 3 níveis distintos, em que cada um dos níveis terão o total instâncias aplicadas seguindo o mesmo estilo para os testes de itens individuais.

Cada teste terá a duração de 100 segundos. Durante esse período, a cada segundo, será coletado e armazenado na memória RAM um conjunto de dados estatísticos relacionados ao desempenho do jogo. Esses dados incluem:

- Número de quadros renderizados no último segundo (*Frame Per Second* - FPS).
- Tempo decorrido para a execução da função *_process()* da Godot.
- Tempo decorrido para a execução da função *_physics_process()* da Godot.
- Uso de memória RAM consumido pelo jogo.
- Uso de memória de vídeo consumido pelo jogo.
- Uso de memória para texturas consumido pelo jogo.

Esses dados permitirão uma análise detalhada do impacto de cada técnica no consumo de recursos do *hardware*, possibilitando comparações precisas entre os métodos tradicionais e procedurais.

Para obter uma visão mais ampla da performance em diferentes contextos, foi planejada a execução dos testes em diversas máquinas, desde modelos mais antigos até os mais recentes, incluindo também dispositivos móveis. A quantidade e os tipos de equipamentos utilizados nos testes foram limitados ao que estava disponível ao autor no momento. A seguir, será apresentada a lista de máquinas e dispositivos utilizados, juntamente com suas respectivas especificações técnicas.

- **Máquina 1** *Processador:* AMD Ryzen 5 4600G, 3.7GHz, 6 núcleos, 12 threads | *Memória RAM:* 32 GB RAM DDR4 3200MHz | *Placa de Vídeo:* Integrada Radeon Vega 7 com memória compartilhada com a memória RAM
- **Máquina 2** *Processador:* Intel Celeron N4020-Serie N, Dual Core (2 threads), 1.10 GHz | *Memória RAM:* 4 GB RAM DDR4 SDRAM 2400Mhz | *Placa de Vídeo:* Integrada Intel UHD Graphics 600 com memória compartilhada com a memória RAM
- **Máquina 3** *Processador:* Intel Celeron N2840, 2.16 GHz, 2 núcleos, 2 threads | *Memória RAM:* 2 GB RAM DDR3L-1333 MHz | *Placa de Vídeo:* Integrada Intel HD Graphics Bay Trail
- **Mobile 1** *Modelo:* Samsung M35 | *Processador:* SAMSUNG Exynos 1380 | *Memória RAM:* 8 GB | *Sistema Operacional:* Android 14

- **Mobile 2** *Modelo:* Samsung Galaxy A14 | *Processador:* SAMSUNG Exynos 1330 | *Memória RAM:* 4 GB | *Sistema Operacional:* Android 13
- **Mobile 3** *Modelo:* Samsung Galaxy A03 Core | *Processador:* Unisoc SC9863A | *Memória RAM:* 2 GB | *Sistema Operacional:* Android 13

Para evitar interferências de desempenho causadas por diferenças entre sistemas operacionais instalados nos *notebooks* e *desktops*, foi planejado executar os testes a partir de um sistema operacional rodando diretamente de um pendrive, utilizando Linux Ubuntu. Dessa forma, garante-se que todos os testes ocorram sob as mesmas condições de *software*, mantendo a consistência dos resultados. No entanto, essa abordagem não pôde ser aplicada aos dispositivos móveis, devido à complexidade envolvida na substituição do sistema operacional original desses aparelhos.

3.5 Execução do projeto

3.5.1 Execução dos elementos procedurais

3.5.1.1 Jogador

O componente "jogador" foi dividido em três partes distintas: o corpo (incluindo os pés, o torso e a cabeça), os dois braços e a arma. A ideia foi separar essas partes para uni-las posteriormente via código, permitindo que o corpo fosse animado de forma tradicional, enquanto os braços se movem livremente, apontando em direção ao cursor do mouse.

Inicialmente, foram desenhadas as animações tradicionais do corpo, representando os movimentos que o jogador realiza, como andar, pular e cair. Em seguida, os dois braços foram desenhados de forma estática para que suas animações pudessem ser controladas via código. Por fim, uma arma também foi desenhada de forma estática.

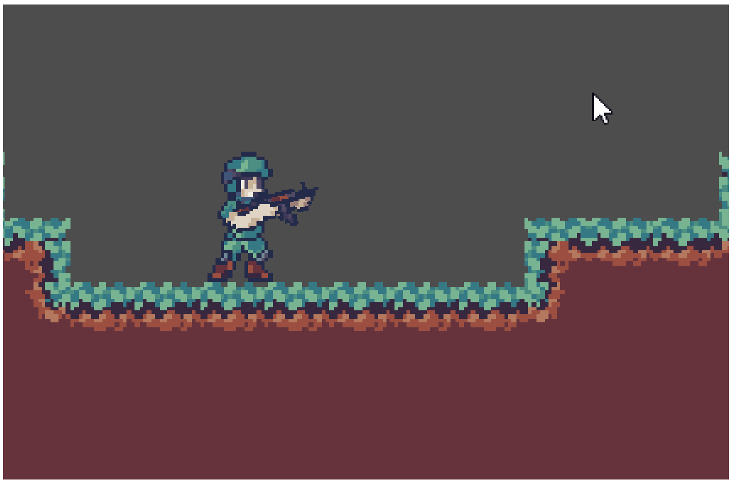
Para complementar as animações tradicionais do corpo, foi implementado um sistema de mira procedural responsável por controlar os braços e a arma. Esse sistema permite que o personagem aponte dinamicamente para a posição do cursor do mouse, oferecendo maior fluidez e responsividade no controle durante o jogo.

A lógica de mira está contida na função *aim(pos: Vector2)*, que recebe como parâmetro a posição do cursor. Primeiramente, determina-se se o cursor está à esquerda ou à direita do personagem, invertendo os *sprites* conforme necessário. Em seguida, o braço direito é suavemente rotacionado em direção ao cursor utilizando a função *lerp_angle*, que interpola entre o ângulo atual e o ângulo desejado com base em uma constante de velocidade pré-estabelecida, garantindo uma transição suave e natural.

Na extremidade do braço direito existe um marcador de posição chamado *Hand*. Esse marcador serve como referência para orientar automaticamente o braço esquerdo em sua direção (*hand.global_position*), simulando um apoio natural à arma. A própria arma também é orientada para essa mesma posição, garantindo coerência visual entre os membros e o equipamento.

Com esse sistema, o jogador, mostrado na figura 22, realiza seus movimentos animados de forma tradicional, enquanto os braços, de maneira procedural, mantém a arma sempre apontada em direção ao cursor do mouse.

Figura 22 – Versão final do componente “jogador”



Fonte: Imagem criada pelo próprio autor deste trabalho

3.5.1.2 Aranha

Para o desenvolvimento da aranha procedural, optou-se por uma abordagem modular, dividindo-a em duas partes independentes: o corpo e as pernas. O corpo é responsável pela locomoção da aranha pelo cenário, mantendo-se sempre a uma certa distância do chão. Já as patas cuidam da animação procedural, com movimentos que buscam imitar o comportamento realista de uma aranha, adaptando-se dinamicamente ao ambiente. Após a implementação separada, ambas as partes são integradas, estabelecendo comunicação entre si. A base da aranha foi inspirada no conteúdo de (Mizizizi, 2018).

3.5.1.2.1 Desenvolvimento do corpo

A concepção do corpo da aranha foi realizada utilizando o nó *CharacterBody2D*, fornecido pela Godot, em conjunto com o padrão de organização *state machine* para estruturar os comportamentos de forma clara e modular. Com base nisso, a primeira funcionalidade desenvolvida foi garantir que o corpo da aranha se mantenha sempre a uma distância pré-determinada do chão.

Para isso, foram posicionados dois *RayCast2D* voltados para baixo, sendo que um deles está levemente mais alto que o outro. Esses *raycasts* são usados para detectar a proximidade do solo, e um algoritmo simples regula a posição vertical do corpo com base nas seguintes regras:

- Se o *raycast* inferior não detectar colisão, significa que o corpo está alto demais, então o corpo tende a descer, simulando um agachamento.

- Se o *raycast* superior detectar colisão, significa que o corpo está baixo demais, então o corpo sobe até parar de detectar colisão.

A segunda funcionalidade implementada foi a movimentação horizontal da aranha, permitindo que ela se desloque para a direita ou para a esquerda. Isso foi feito inicialmente detectando se as teclas A ou D estão sendo pressionadas, e ajustando a coordenada X do corpo em resposta.

Para simular o efeito visual do corpo da aranha se elevar levemente ao caminhar, foram adicionados mais dois *RayCast2D*, posicionados um pouco mais baixos que os anteriores. Quando a aranha está em movimento, ela se alinha com esses *raycasts* inferiores, elevando ligeiramente o corpo. Quando parada, ela volta a se alinhar com os *raycasts* principais. Combinando essas técnicas com o sistema de ajuste vertical baseado em detecção de colisão, o corpo da aranha é capaz de andar em terrenos com variações de altura, subindo e descendo pequenos obstáculos de forma fluida e natural.

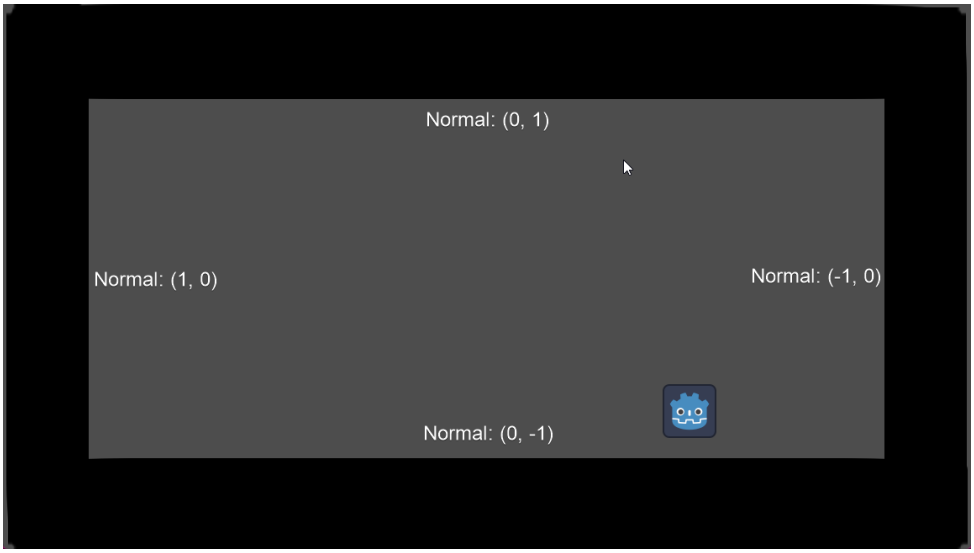
O passo seguinte no desenvolvimento foi implementar a mecânica que permite à aranha se locomover pelas paredes e pelo teto. Para isso, foram adicionados dois nós do tipo *RayCast2D* ao corpo da aranha, orientados lateralmente: um voltado para a direita e o outro para a esquerda, com o objetivo de detectar a presença de paredes. A lógica implementada consiste em verificar se a aranha está se movendo em uma direção (por exemplo, para a direita) e se o *raycast* correspondente está detectando uma parede. Quando essa condição é satisfeita, a normal da superfície detectada é capturada e utilizada para reorientar o corpo da aranha com base nessa normal. A partir desse momento, a parede passa a ser tratada como o novo chão relativo, permitindo que a aranha permaneça e se desloque sobre ela de forma natural.

Com a aranha posicionada em uma nova superfície, seja uma parede ou teto, foi necessário adaptar seu sistema de movimentação para que respondesse à nova orientação. Para isso, o movimento da aranha passou a ser calculado de forma relativa à normal da superfície atual em que ela se encontra. Essa normal é a direção oposta ao chão - por exemplo, no chão tradicional, onde é uma superfície, a normal é (0, -1), já uma parede à direita a normal será dada por (-1, 0). As normais das superfícies são mostradas com mais detalhes na figura 23.

Com base nisso, foi implementada uma lógica condicional que, ao verificar o valor da normal atual, ajusta o vetor de velocidade da aranha na direção apropriada.

- Se a normal da superfície for (0, -1) (chão), o movimento ocorre no eixo X, como normalmente.
- Se a normal for (-1, 0) (parede direita), o movimento passa a ser no eixo Y, a direita vira para cima, e a esquerda vira para baixo.
- Se a normal for (1, 0) (parede esquerda), o movimento também será no eixo Y, a direita vira para baixo, e a esquerda vira para cima.

Figura 23 – Normais das superfícies



Fonte: Imagem criada pelo próprio autor deste trabalho

- Se a normal for (0, 1) (teto), o movimento volta a ser no eixo X, porém invertido em relação ao chão.

Após essa implementação, a aranha passou a ser capaz de se locomover pelas paredes e pelo teto. Em testes realizados em um ambiente com formato quadrado, mostrado na figura 24, foi possível observar que ela consegue percorrer toda a volta do cenário de forma contínua. Vale ressaltar que foi necessário criar um estado específico na *state machine* dedicada à lógica da aranha para realizar a transição entre superfícies. Esse estado executa uma pequena animação de ajuste, garantindo uma transição suave entre o chão, as paredes e o teto.

Figura 24 – Protótipo da aranha subindo a parede

Fonte: Imagem criada pelo próprio autor deste trabalho

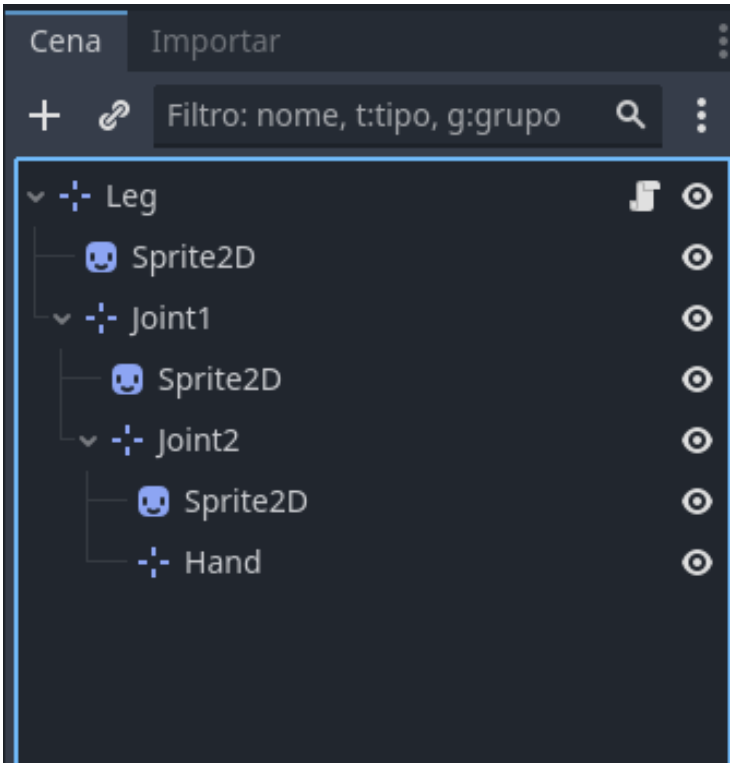
3.5.1.2.2 Desenvolvimento das pernas

Com o corpo da aranha finalizado e funcional, o próximo passo foi o desenvolvimento das pernas. A proposta central foi acoplar as pernas ao corpo de forma que, à medida que o corpo se movimenta, elas realizam uma animação característica de aranhas, ajustando-se dinamicamente ao cenário e às condições em que o corpo se encontra. Para isso, as pernas foram projetadas para serem controladas diretamente pelo corpo, executando movimentos específicos em momentos determinados, de modo a reforçar o realismo e a coerência da animação procedural.

Cada perna foi implementada como um objeto independente, responsável por articular-se com base em uma posição-alvo recebida. Ao receber essa posição, a perna executa uma animação em que sua extremidade - o efetuador - se desloca até o ponto especificado, com os segmentos intermediários se movendo de forma encadeada, enquanto a base da perna permanece fixa.

A estrutura de cada perna foi composta por três juntas articuladas, além da extremidade que representa o efetuador da aranha. A primeira junta é fixa ao corpo, enquanto as duas seguintes correspondem às articulações entre os elos. A última parte, o efetuador, é o ponto que efetivamente se desloca até a posição-alvo. Todas as juntas são organizadas hierarquicamente na árvore de nós da perna, aproveitando-se da estrutura de componentes da Godot, conforme ilustrado na figura 25. Para calcular os ângulos necessários ao movimento articulado da perna, foram utilizadas funções de IK, permitindo que a perna se mova de maneira natural e precisa.

Figura 25 – Estrutura da árvore da perna da aranha



Fonte: Imagem criada pelo próprio autor deste trabalho

Para simular o movimento da perna se elevando levemente durante um passo, foi introdu-

zida uma variação no caminho da animação. Entre a posição inicial e a posição-alvo, calcula-se um ponto intermediário - o ponto médio - ao qual é aplicada uma elevação com base na normal da superfície onde a aranha se encontra. Assim, a trajetória do efetuador da perna descreve uma curva suave, elevando-se no meio do percurso antes de atingir o alvo, conferindo maior naturalidade ao movimento.

Por fim, foi implementada uma funcionalidade de direção para as pernas. Como a aranha possui pernas tanto do lado direito quanto do lado esquerdo, é necessário que cada lado se movimente de maneira espelhada. Para isso, foi adicionada uma variável *booleana* configurável no editor da Godot, permitindo indicar se a perna pertence ao lado esquerdo. Quando ativada, essa variável ajusta automaticamente a orientação e o comportamento da perna, garantindo simetria e consistência nos movimentos.

3.5.1.2.3 Mesclando as pernas com o corpo

Com o desenvolvimento das pernas finalizado, o próximo passo consistiu em integrá-las ao corpo da aranha. Para isso, foram instanciadas seis pernas no corpo, sendo três posicionadas do lado esquerdo e três do lado direito, respeitando a organização previamente definida no planejamento.

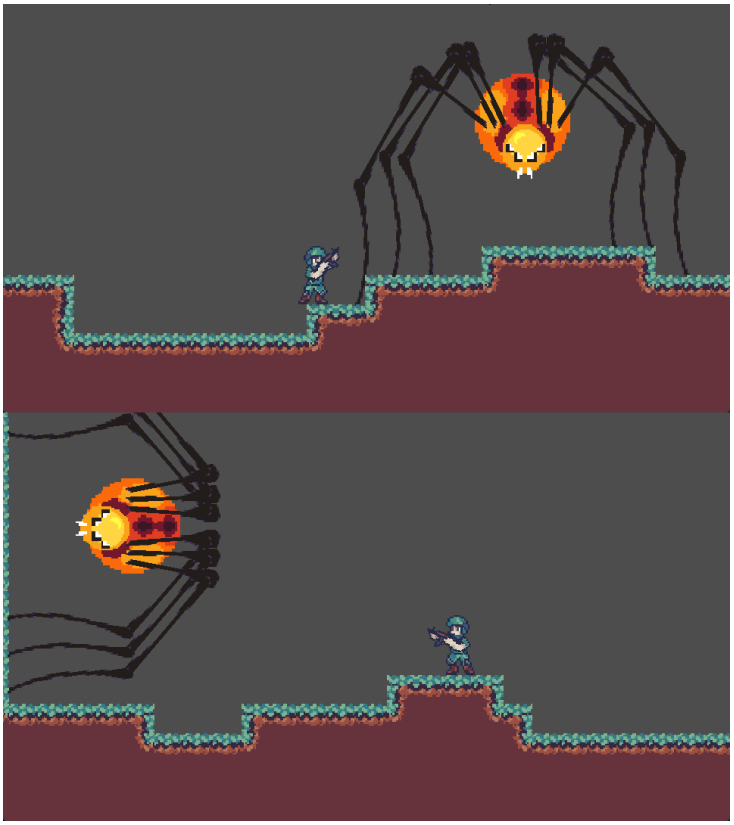
O primeiro desafio foi fazer com que as pernas se adaptassem ao terreno em que a aranha se encontrava. Para isso, foi utilizado um sistema de detecção por meio de *RayCast2D*. Cada perna possui um *RayCast2D* posicionado em sua base, apontando para baixo, totalizando seis sensores. Esses *raycasts* detectam a superfície abaixo da aranha e retornam a posição de colisão, que é então utilizada como o ponto-alvo da perna correspondente. Dessa forma, mesmo que o terreno seja irregular ou inclinado, as pernas se adaptam dinamicamente a ele.

Em seguida, foi necessário coordenar os movimentos das pernas para simular uma caminhada realista. Para isso, foi implementado um sistema de alternância em que, a cada ciclo, três pernas (duas de um dos lados e outra perna do lado oposto) se movem em direção ao próximo ponto-alvo, esse sistema foi definido de forma empírica pelo autor ao visualizar vídeos de aranhas reais se locomovendo. Assim que essas alcançam suas posições, outras duas pernas iniciam seu movimento, criando uma sequência fluida e contínua de passos. Um detalhe importante é que, uma vez que a extremidade da perna (o efetuador) atinge o ponto-alvo, ele permanece fixo naquela posição, mesmo que o corpo da aranha continue se movendo. Isso reforça o efeito visual de estabilidade e aderência ao terreno.

Por fim, com todos esses sistemas funcionando em conjunto, a aranha entra em um ciclo automatizado de locomoção. Em modo de caminhada, ela seleciona pares de pernas para avançar, respeitando um intervalo de tempo definido entre cada ciclo. Enquanto algumas pernas se movimentam, outras permanecem fixas, e os elos das pernas se articulam para manter a conexão com o ponto de ancoragem. Além disso, os *raycasts* se deslocam junto com o corpo, garantindo que os pontos-alvos estejam sempre atualizados com base nas novas superfícies detectadas, a versão final da aranha é mostrada na figura 26. Esse sistema demonstrou ser eficaz

inclusive durante transições para outras superfícies, como paredes ou tetos, pois os *raycasts* acompanham a rotação do corpo, adaptando automaticamente os alvos das pernas.

Figura 26 – Versão final da aranha procedural junto com a demonstração dela subindo a parede



Fonte: Imagem criada pelo próprio autor deste trabalho

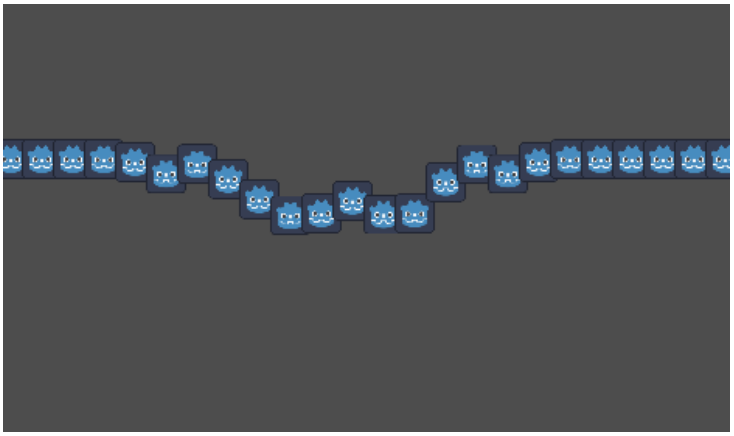
3.5.1.3 Água

A água procedural foi desenvolvida com o objetivo de criar uma superfície fluida e responsiva a colisões, permitindo que, ao entrar em contato com o jogador ou com entidades como a aranha, ela reaja de forma satisfatória, gerando ondas e oscilações proporcionais ao impacto da entidade. O sistema foi baseado em um projeto de código aberto HackTrout (2020), disponibilizado na plataforma do *Github*.

Para desenvolver esse efeito, a água foi fragmentada em pequenas instâncias chamadas *WaterSpring*, distribuídas de forma homogênea ao longo da superfície. Cada instância representa uma mola com movimento vertical, possuindo uma altura padrão — definida inicialmente —, e, ao receber uma força vertical, sendo ela para cima ou para baixo, tenta retornar à posição original utilizando um sistema de amortecimento.

Além de reagirem individualmente a colisões, as molas influenciam suas vizinhas, transferindo uma porcentagem da sua força atual para elas, o que gera o efeito de propagação de ondas pela superfície da água. Essa propagação pode ser recalculada múltiplas vezes dentro do mesmo quadro; quanto maior esse número de repetições, mais rápida e suave é a propagação das ondas. Esse efeito é demonstrado na figura 27.

Figura 27 – *WaterSpring* influenciando as vizinhas



Fonte: Imagem criada pelo próprio autor deste trabalho

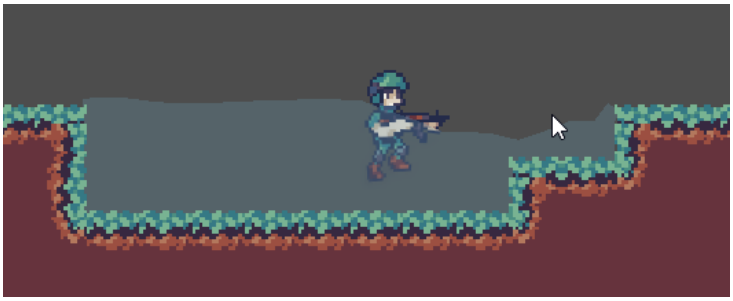
Para criar e gerenciar as instâncias de *WaterSpring*, foi desenvolvido um objeto chamado *WaterBody*. Esse objeto é responsável por instanciar as molas em tempo de execução, de acordo com as configurações definidas do objeto, e também por coordenar a influência entre molas vizinhas. A cada quadro, o *WaterBody* desenha um polígono, cuja borda superior é formada pelas coordenadas das *WaterSprings*, enquanto a parte inferior é desenhada como um retângulo convencional.

A cor do polígono foi definida como um tom de azul com certa transparência, simulando a aparência da água e permitindo que objetos ao fundo permaneçam visíveis como é mostrado na figura 28. Após isso, foi criado um *script* de *shader* para simular a distorção da luz através da água, provocando um efeito ondulatório nos objetos que se encontram atrás dela.

Por fim, foi adicionado um objeto *Area2D* à cada *WaterSpring*, denominado *WaterHurtCollision*, responsável por detectar colisões com entidades do mundo e repassar essas informações para a *WaterSpring* correspondente. Esse objeto recebe dois parâmetros no momento da colisão: a força do impacto e a direção (para cima ou para baixo). Cada entidade que deseja interagir com a água é responsável por fornecer esses parâmetros ao colidir com um *WaterHurtCollision*. Para facilitar essa comunicação, foi criado um objeto adicional, chamado *WaterHitCollision*, também do tipo *Area2D*, que gerencia a aplicação correta desses parâmetros, cabendo a cada entidade apenas configurar os valores desejados.

O resultado final é uma superfície de água que reage dinamicamente às colisões de entidades, conforme ilustrado na figura 28. A configuração para cada entidade é bastante simples, sendo necessário apenas instanciar um objeto *WaterHitCollision* e ajustar a força conforme a situação. Essa flexibilidade permite, por exemplo, que ao jogador cair de uma grande altura, o *WaterHitCollision* seja configurado para aplicar uma força maior, gerando um impacto mais intenso na superfície da água.

Figura 28 – Versão final da água procedural



Fonte: Imagem criada pelo próprio autor deste trabalho

3.5.1.4 Tecido

A tarefa de desenvolver o tecido procedural foi dividida em várias etapas pequenas, cada uma delas criando um subsistema, que ao serem combinados, resultam na simulação completa da animação do tecido. A base para a implementação foi baseada no conteúdo de HackTrout (2021) no qual trás uma abordagem genérica para simulações de tecidos em ambientes 2D.

O primeiro componente desenvolvido foi o objeto *RopePoint*, que representa a menor unidade do tecido. Trata-se de um nó individual baseado em *CharacterBody2D*, que permite a aplicação de forças externas por meio da função *apply_force*, a qual recebe um vetor como parâmetro e o converte em aceleração. Além disso, o *RopePoint* possui dois comportamentos adicionais: uma gravidade constante, que faz cair naturalmente, e uma funcionalidade de congelamento que, quando ativada, mantém o ponto fixo no espaço, impedindo qualquer movimento.

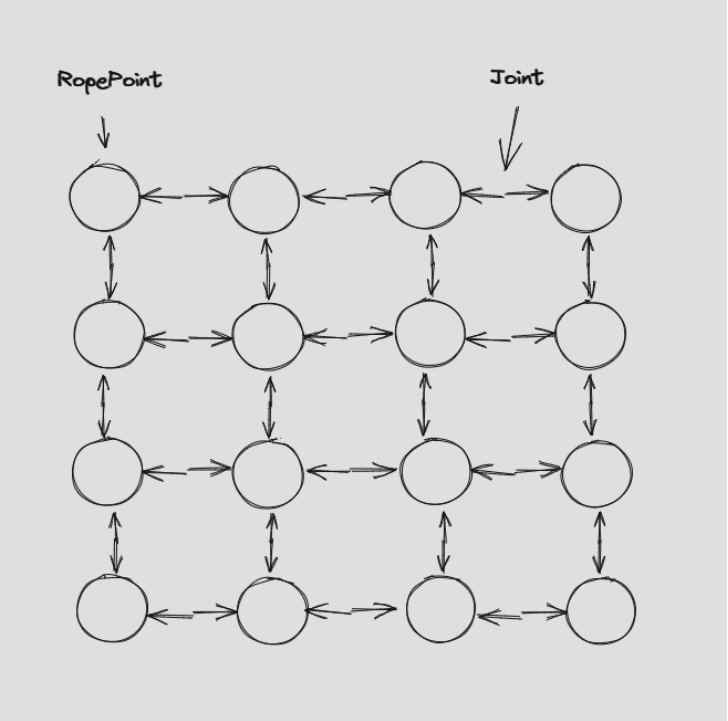
Em seguida, foi criado o objeto *Joint*, responsável por simular o comportamento de uma mola com propriedades ajustáveis. Esse objeto conecta dois corpos (*object_1* e *object_2*) mantendo uma distância-alvo (*target_length*) entre eles. A força elástica aplicada entre os corpos segue o princípio da Lei de Hooke, permitindo controlar a rigidez da mola através do parâmetro *stiffness* e a dissipação de energia com o parâmetro *damping*, o que evita oscilações abruptas e contribui para a estabilidade da simulação.

Durante cada ciclo de física, o sistema calcula a distância entre os dois objetos conectados. Caso essa distância seja diferente da desejada, é aplicada uma força proporcional ao erro, corrigindo a posição relativa entre os corpos. Essa força é suavizada pelo fator de amortecimento, garantindo estabilidade tanto numérica quanto visual.

Com esses dois componentes fundamentais — *RopePoint* e *Joint* — foi possível construir a estrutura do tecido procedural. Para isso, foi criado o objeto *Cloth*, que instancia uma malha retangular composta por diversos *RopePoints*, conforme os parâmetros *cloth_width* e *cloth_height*, que definem a quantidade de pontos na largura e altura da malha. Após a criação dos pontos, eles são conectados entre si utilizando objetos *Joint*, formando uma rede de ligações que simula o comportamento de um tecido maleável, conforme ilustrado na figura 29.

Vale destacar que, como os *RopePoints* são implementados com base no nó *CharacterBody2D*, eles interagem naturalmente com colisores do ambiente. Dessa forma, quando o tecido

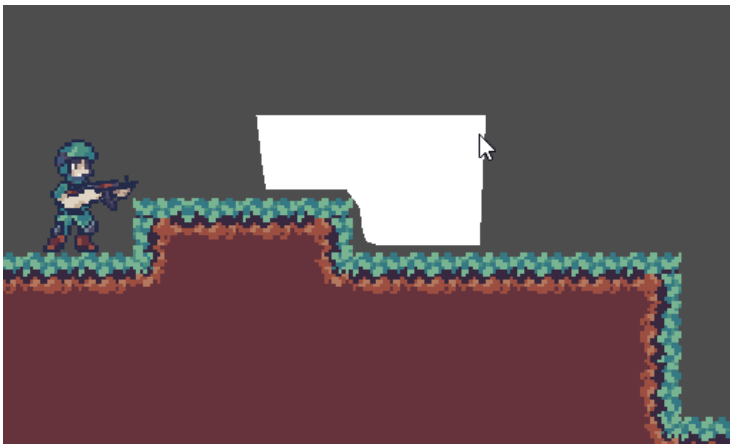
Figura 29 – Ilustração de como foi feito a conexão entre os *RopePoints* com os *Joints*



Fonte: Imagem criada pelo próprio autor deste trabalho

é posicionado sobre uma superfície sólida, sua estrutura se adapta ao contorno da colisão dessa superfície, moldando-se ao relevo presente, como pode ser observado no resultado final ilustrado na figura 30.

Figura 30 – Versão final do tecido procedural



Fonte: Imagem criada pelo próprio autor deste trabalho

Para possibilitar a interação de entidades com o tecido, foi desenvolvido um componente auxiliar chamado *RopePointHitCollision*, baseado em *Area2D*. Esse componente é responsável por detectar colisões com os *RopePoints* e aplicar neles uma força direcional. A entidade que colide com o tecido deve apenas configurar, continuamente, um vetor que representa a direção e intensidade da força a ser aplicada no momento da interação.

Por fim, a renderização visual do tecido procedural foi implementada na função *_draw()*, que desenha dinamicamente as áreas entre os pontos da malha. Cada quadrante do tecido pode ser composto por até quatro *RopePoints*, sendo subdividido em dois triângulos para facilitar o processo de renderização. A função percorre todos os quadrantes da malha, desenhando os triângulos correspondentes, o que resulta na formação visual completa do tecido na tela.

3.5.1.5 Fogo

Para a criação do fogo procedural, foi utilizado o sistema de partículas da Godot baseado no projeto de KidsCanCode (2019) no *Github*, junto com vídeo de swydev (2023), que possui comportamento altamente configurável e versátil, permitindo criar um efeito de fogo ajustável em tamanho, cor, forma e intensidade.

O processo começou com o desenho de uma pequena imagem em formato de gota d'água na cor branca, utilizada como textura base no sistema de partículas. Aproximadamente 5.000 dessas gotas são emitidas com um tempo de vida curto, de 0,4 segundos. Em seguida, foi configurada uma gravidade negativa, fazendo com que as partículas se movimentam para cima — simulando a elevação natural do calor —, com uma forma de emissão esférica. Assim, diversas partículas são lançadas para cima de maneira dispersa.

A velocidade inicial das partículas pode atingir até 290 *pixels* por segundo, proporcionando um movimento rápido. À medida que sobem, elas possuem rotação angular variável entre -45° e 45°, com controle refinado através de uma curva personalizada (*angular_velocity_curve*), o que confere ao fogo um comportamento instável e realista.

Na base da emissão, as partículas são geradas com escala padrão, mas à medida que sobem, sua escala diminui gradualmente. Esse comportamento é modulado por uma curva (*scale_amount_curve*), permitindo que cresçam e encolham suavemente ao longo do tempo, simulando as ondulações características das chamas.

A aparência visual do fogo é definida por uma rampa de cores (*color_ramp*), que determina a transição de tons ao longo da vida da partícula — geralmente passando de tons mais suaves (amarelo claro ou cinza) para cores mais intensas (vermelho, laranja), imitando a queima progressiva do fogo. A opacidade global das partículas foi reduzida para 45%, tornando-as parcialmente transparentes e evitando sobreposição visual excessiva.

Para complementar o efeito das chamas, foi criado um segundo sistema de partículas, desta vez para simular a fumaça. Esse sistema é posicionado atrás das partículas de fogo, reforçando a sensação de profundidade e realismo da combustão.

Assim como no caso do fogo, o desenvolvimento da fumaça começou com o desenho de uma imagem 2D, neste caso uma esfera branca. Essa imagem foi inserida no sistema, que emite cerca de 2.000 instâncias com tempo de vida prolongado (3 segundos), em uma área circular com raio de 23 *pixels*, garantindo uma dispersão lateral suave desde a origem.

A gravidade das partículas de fumaça também foi configurada com uma força ascendente, porém mais sutil, simulando o movimento característico da fumaça. A escala da velocidade foi

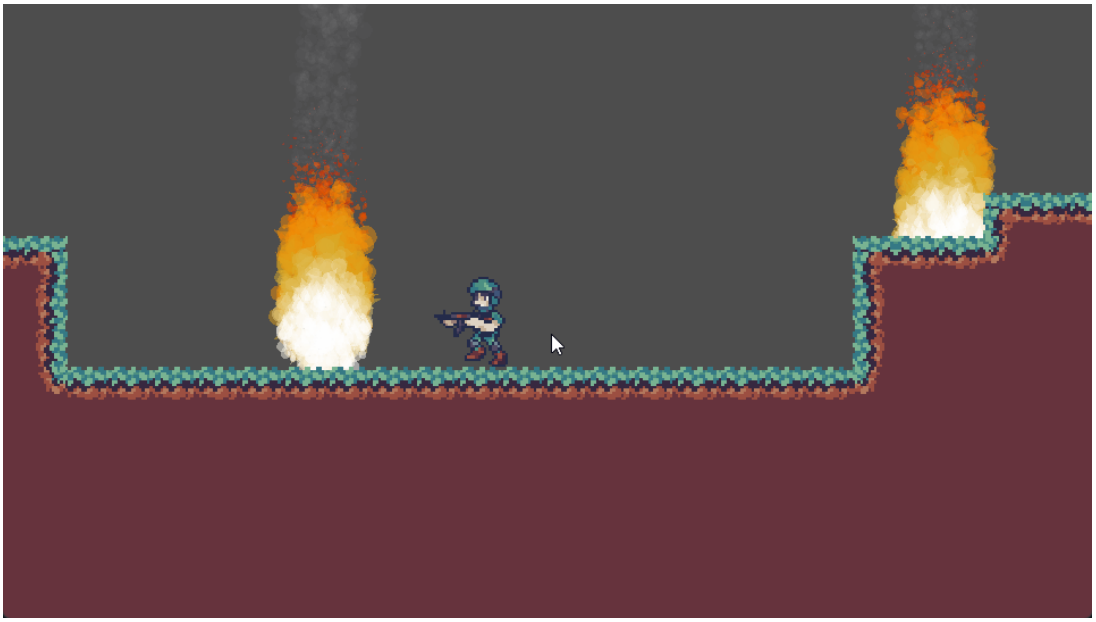
reduzida para 0,3, criando um movimento lento, flutuante e atmosférico.

Além disso, foi configurada uma rotação em torno do ponto emissor (*orbit_velocity_min*), proporcionando à fumaça um movimento circular e imprevisível, semelhante aos redemoinhos suaves formados por correntes de ar quente. Isso confere um comportamento mais orgânico e menos repetitivo ao efeito.

Diferentemente das partículas de fogo, as partículas de fumaça aumentam gradualmente de escala ao longo da vida útil, com valores variando entre 0,0 e 0,2, também controlados por uma curva personalizada (*scale_amount_curve*). Isso simula a expansão da fumaça à medida que ela sobe e esfria.

Por fim, a opacidade das partículas de fumaça foi significativamente reduzida, com valor final em torno de 0,011, tornando-as quase imperceptíveis individualmente, mas criando um efeito volumétrico quando somadas — ideal para representar fumaça ambiente de forma sutil e realista. A figura 31 mostra o resultado final do fogo animado proceduralmente.

Figura 31 – Versão final do fogo procedural



Fonte: Imagem criada pelo próprio autor deste trabalho

3.5.2 Execução dos elementos tradicionais

3.5.2.1 Aranha

A aranha tradicional foi desenvolvida de forma semelhante à aranha procedural, com algumas modificações relevantes, como a adição dos estados de pulo e queda, que não estão presentes na versão procedural por não serem necessários. Além disso, foram realizadas adaptações visuais específicas para acomodar as limitações da animação tradicional.

Como não é possível replicar o comportamento dinâmico das pernas da aranha em relação ao terreno, como ocorre na versão procedural, foi necessário implementar um estado de

pulo sempre que a aranha precisa subir uma elevação, e um estado de queda para representar sua descida. Essa abordagem, no entanto, introduz alguns problemas visuais, como pernas aparentemente fora do chão como é mostrado na figura 32. Para mitigar essas inconsistências, foi preciso manter os mapas mais planos. Em um jogo real, o ideal seria que a superfície na qual a aranha se move fosse predominantemente plana, a fim de preservar a coerência visual.

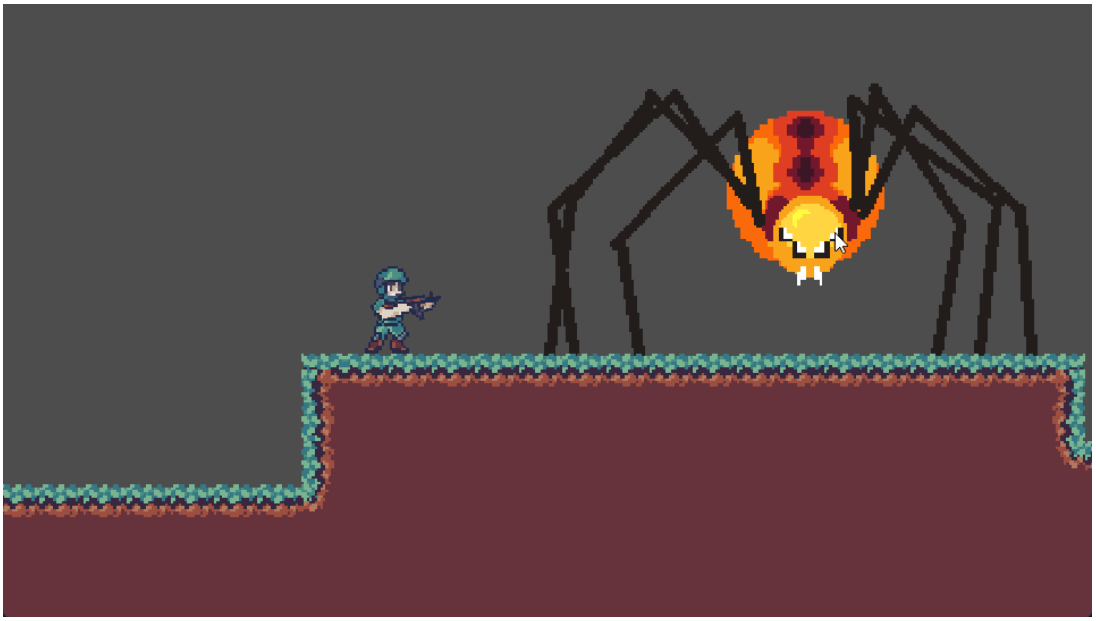
Figura 32 – Inconsistência visual quando a aranha tradicional se aproxima de uma borda

Fonte: Imagem criada pelo próprio autor deste trabalho

Outra modificação importante foi a simplificação dos detalhes das pernas. Devido ao tamanho considerável da aranha e à quantidade de pernas com múltiplos segmentos, animar todos esses elementos com riqueza de detalhes revelou-se um grande desafio. A solução adotada foi simplificar as pernas em linhas retas e formas mais básicas. Essa decisão foi tomada porque o autor encontrou grande dificuldade e gasto de tempo ao tentar animar os movimentos detalhados da aranha. Embora um animador experiente em *pixel art* presumivelmente conseguisse realizar essa tarefa com mais facilidade.

Apesar das limitações apresentadas, foi perceptível a maior liberdade artística proporcionada pela animação tradicional. Testes avulsos, como animações de ataque criadas pelo autor, puderam ser implementados com mais facilidade nesse modelo, uma vez que não exigem uma lógica matemática e programática complexa, como ocorre na abordagem procedural. Isso evidencia que, embora a animação tradicional tenha restrições em termos de responsividade e adaptabilidade ao ambiente, ela oferece maior flexibilidade criativa durante o processo de desenvolvimento visual. Na figura 33 é possível visualizar a versão final da aranha tradicional em um ambiente plano.

Figura 33 – Versão final da aranha animada de forma tradicional



Fonte: Imagem criada pelo próprio autor deste trabalho

3.5.2.2 Água

A animação tradicional da água seguiu princípios semelhantes à abordagem procedural, consistindo em uma área retangular preenchida com uma textura azul e uma zona de colisão que permite ao jogador detectar o contato com a água. No entanto, diferentemente da versão procedural, a superfície da água tradicional não reage a impactos externos. Em vez disso, apresenta uma animação contínua e sutil de ondas na parte superior.

Essa animação foi implementada utilizando o *node TileMap* da Godot, composto por duas partes principais: uma textura aplicada abaixo da borda superior e a própria textura da borda superior. Ambas as texturas apresentam animações suaves de ondulação, e, quando combinadas, produzem um movimento contínuo que simula o comportamento da superfície da água. Além disso, a borda superior possui uma animação adicional nas extremidades, com coloração distinta, facilitando a identificação visual da superfície da água.

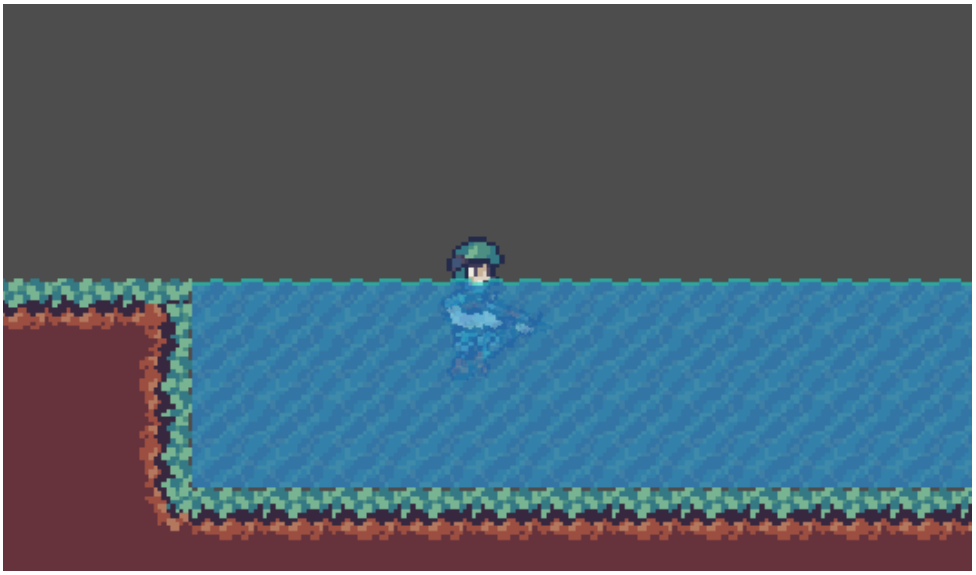
A coloração geral da água é azulada e possui opacidade reduzida, permitindo a visualização parcial de objetos submersos, como o próprio personagem, o que contribui para o realismo do efeito visual. Para cada instância de água adicionada ao mapa, é criado também um *node Area2D* que delimita a área de colisão. Esse recurso permite que o jogador detecte quando entra em contato com a água e, consequentemente, altere seu estado para o modo de natação.

A figura 34 apresenta a versão final da água animada de forma tradicional.

3.5.2.3 Tecido

A animação tradicional do tecido adotou uma abordagem distinta da versão procedural. Enquanto o tecido procedural apresenta alta responsividade a colisões externas, ajustando-se

Figura 34 – Versão final da água tradicional



Fonte: Imagem criada pelo próprio autor deste trabalho

dinamicamente às superfícies e modificando sua forma conforme os impactos, esse nível de interatividade não foi possível na animação tradicional. Para alcançar tal comportamento, seria necessário criar diversas animações complementares e um grande número de estados para cobrir todas as possíveis situações, o que tornaria o processo complexo e pouco viável.

Dessa forma, o tecido tradicional foi concebido como um elemento puramente visual. Optou-se por representar esse material por meio de uma bandeira, uma vez que, durante a busca por referências, observou-se que bandeiras são frequentemente utilizadas como representação estética desse tipo de material, sendo versáteis e adequadas a diversos estilos de jogos e narrativas.

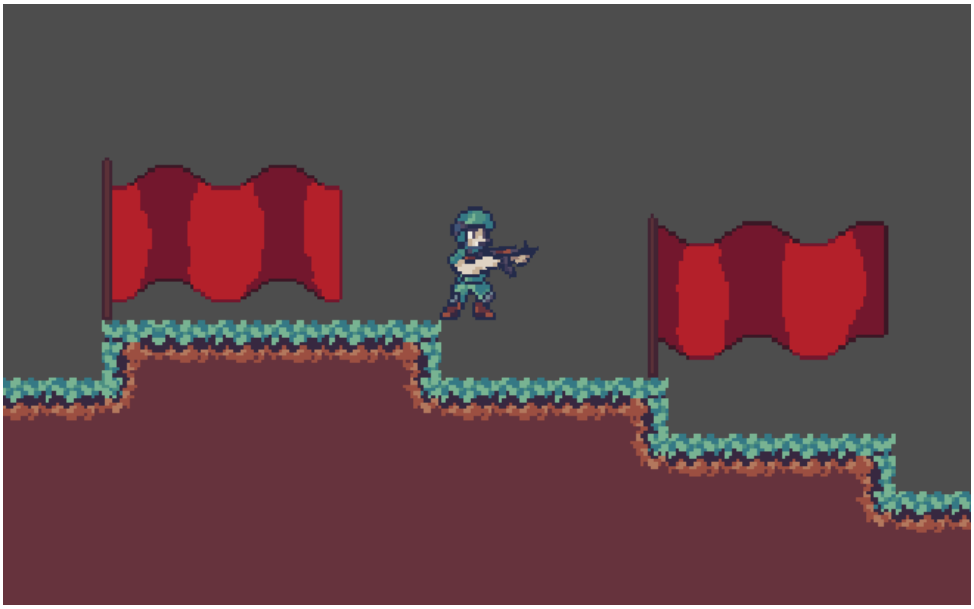
Para a animação da bandeira, foi inicialmente criada uma sequência simples que simula o movimento de um tecido sendo estendido e balançando ao vento. Em seguida, foram adicionadas sombras que representam as áreas em que o tecido se dobra ou recua, criando a sensação de volume e profundidade. Por fim, foi desenhado um pequeno bastão ao qual a bandeira está presa. Esse bastão pode ser facilmente substituído por outras variações, de acordo com a necessidade do cenário.

A figura 35 apresenta o resultado final da animação tradicional do tecido.

3.5.2.4 Fogo

A animação tradicional do fogo seguiu princípios semelhantes aos utilizados na abordagem procedural, utilizando como referência o vídeo de AdamCYounis (2020). A chama se inicia com tons mais claros e quentes na base e, à medida que sobe e esfria, assume colorações mais escuras, acompanhada de uma fumaça escura ao fundo que reforça a ambientação. Um elemento adicional presente apenas na versão tradicional foi a inclusão de uma textura de lenha na base das chamas. Esse detalhe foi concebido durante o processo de animação para integrar melhor o efeito visual ao ambiente do jogo.

Figura 35 – Versão final do tecido tradicional



Fonte: Imagem criada pelo próprio autor deste trabalho

A animação em *looping* foi realizada de forma direta, sem a utilização de quadros-chave. Inicialmente, definiu-se que a animação teria oito quadros. A partir disso, foram desenhadas as partículas de fogo em ascensão, cada uma com uma trajetória visual distinta, criando pequenas "narrativas" dentro da animação. O processo foi repetido até que a quantidade de partículas gerasse uma composição visual coesa, sem lacunas perceptíveis.

Em seguida, foi realizada a coloração das partículas. O centro da base foi preenchido com um tom de laranja intenso, que gradualmente escurece à medida que se afasta do centro e se aproxima do topo da chama. As partículas mais altas receberam tons mais escuros, simulando o resfriamento da chama conforme ela se dissipa.

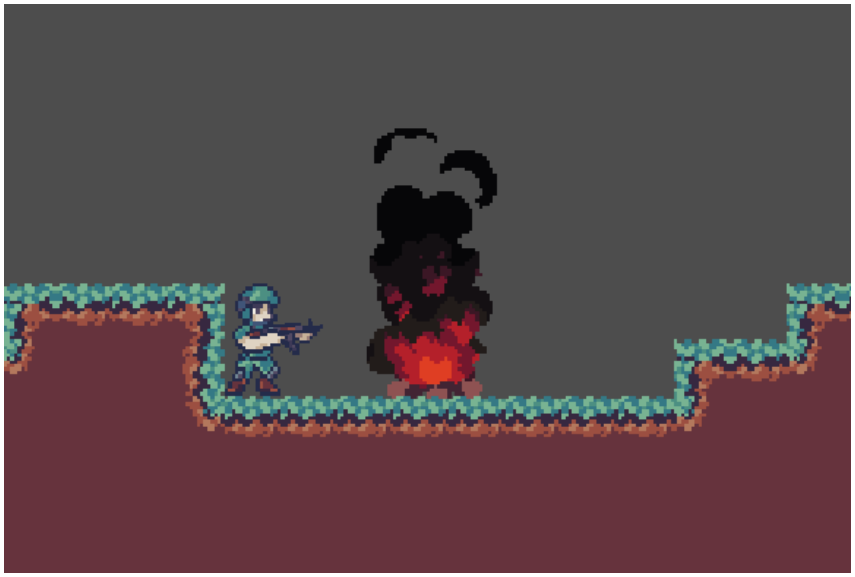
Por fim, a fumaça de fundo foi desenhada utilizando o mesmo princípio da animação das chamas, porém com partículas maiores e em uma única tonalidade. O topo da fumaça apresenta um efeito de dispersão, sugerindo o seu desaparecimento. A figura 36 ilustra a versão final do fogo animado de forma tradicional.

3.6 Execução dos testes

Foram realizados testes para cada um dos elementos desenvolvidos neste projeto, considerando tanto suas versões tradicionais quanto procedurais. Adicionalmente, foi conduzido um teste geral que combina todos os elementos de uma mesma categoria (procedural ou tradicional) simultaneamente, com o objetivo de avaliar o comportamento e o desempenho dos elementos quando utilizados em conjunto.

Cada teste foi dividido em três níveis, variando a quantidade de instâncias dos elementos para verificar a escalabilidade e o impacto no desempenho. O nível 1 representa uma quantidade reduzida de instâncias, enquanto o nível 3 apresenta uma densidade maior, com o intuito de

Figura 36 – Versão final do fogo tradicional

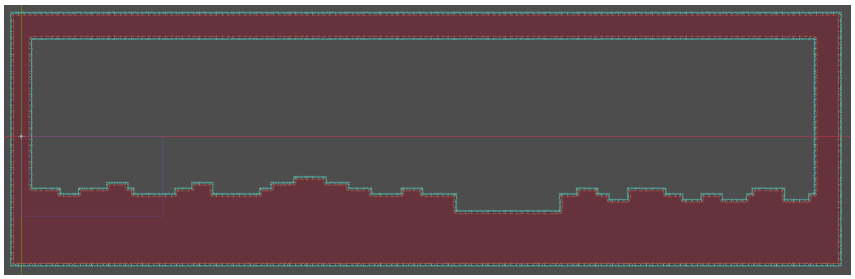


Fonte: Imagem criada pelo próprio autor deste trabalho

testar os limites da execução.

O primeiro passo para o desenvolvimento dos testes foi a criação dos ambientes de execução, ou seja, os mapas onde os testes seriam realizados. Considerando que as mecânicas dos elementos tradicionais diferem significativamente das mecânicas dos elementos procedurais, foram projetados dois mapas distintos. Para os elementos procedurais, o mapa foi construído com terrenos irregulares, contendo relevos, desníveis e buracos, permitindo que elementos como a aranha demonstrassem sua capacidade de locomoção em ambientes complexos, como pode ser visto na figura 37.

Figura 37 – Versão final do mapa de testes dos elementos procedurais



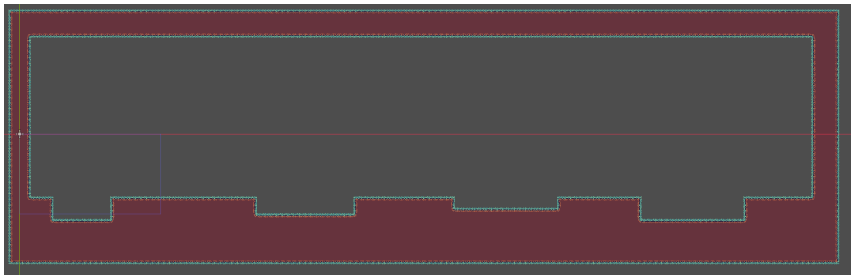
Fonte: Imagem criada pelo próprio autor deste trabalho

Já para os elementos tradicionais, optou-se por um terreno mais plano, embora ainda apresentasse depressões para conter a água e pequenos relevos. No geral, esse *design* é mais simples comparado ao mapa procedural, assim como é mostrado na figura 38.

Todos os testes com elementos procedurais foram realizados no mapa desenvolvido para esses elementos, assim como todos os testes com elementos tradicionais foram executados no mapa exclusivo para essa abordagem.

O passo seguinte foi definir a quantidade de instâncias que seriam utilizadas em cada

Figura 38 – Versão final do mapa de testes dos elementos tradicionais



Fonte: Imagem criada pelo próprio autor deste trabalho

Tabela 1 – Números de instâncias nos testes procedurais

Quantidade elementos	Aranha	Fogo	Tecido
Nível 1	30	8	3
Nível 2	60	16	6
Nível 3	90	24	10

Fonte: Elaborado pelo próprio autor deste trabalho.

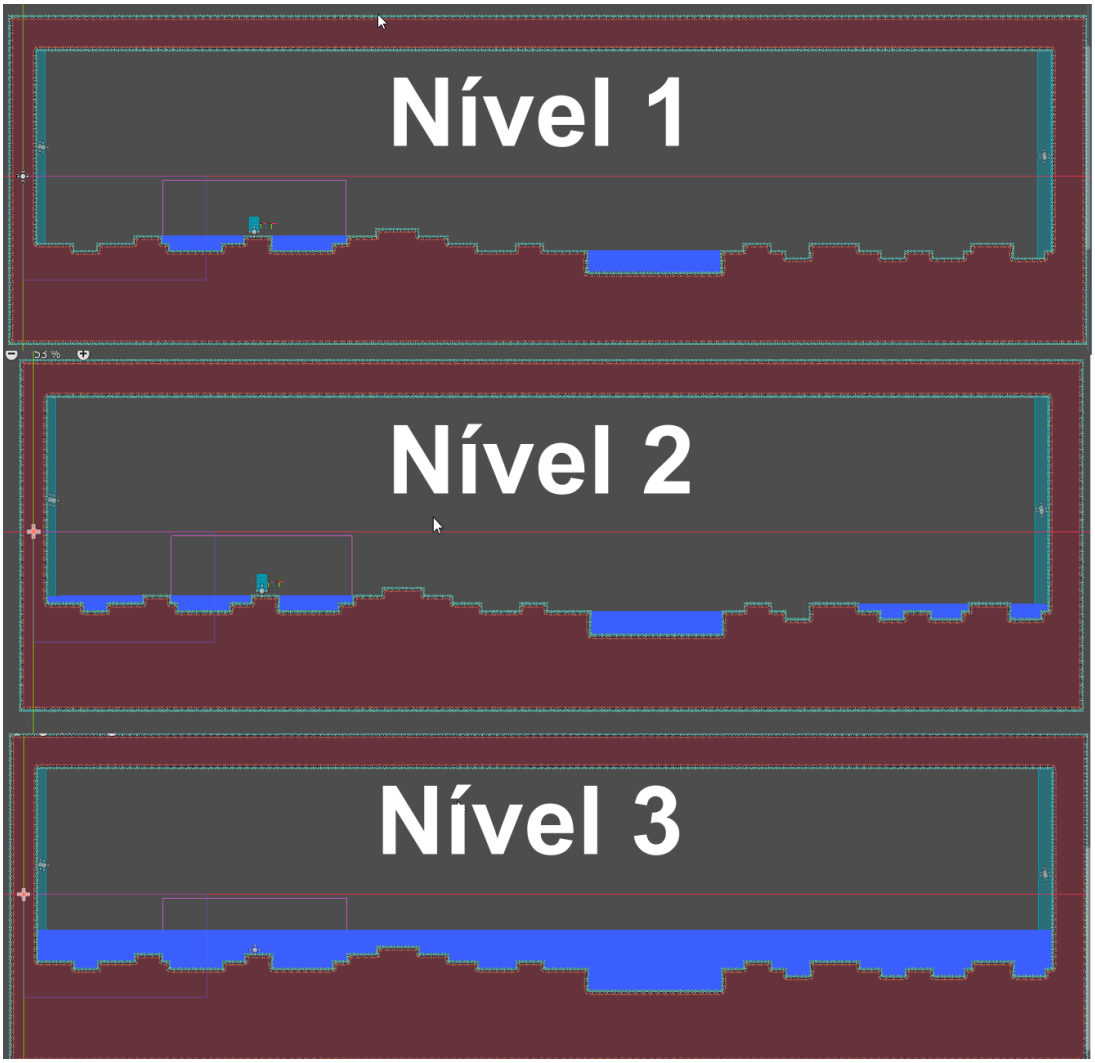
nível de teste. Para isso, foram realizados testes preliminares na máquina mais potente disponível ao autor do projeto (Máquina 1). O número máximo de instâncias foi determinado com base no ponto em que a taxa de quadros por segundo começou a se aproximar dos 30 FPS. Esse valor foi então estabelecido como a quantidade de instâncias para o nível 3. A partir disso, o nível 2 foi definido como correspondendo a dois terços da quantidade de instâncias do nível 3, enquanto o nível 1 foi definido como um terço da quantidade total do nível 3.

A tabela 1 apresenta a quantidade de instâncias utilizadas para os elementos procedurais em cada nível. Ressalta-se que o elemento “água” não possui um número fixo de instâncias, pois sua implementação é baseada em áreas contínuas. Dessa forma, o teste com água foi conduzido por meio do aumento progressivo do nível do fluido, culminando na inundação total da área em que o jogador se encontra. A variação desses níveis é ilustrada na figura 39.

No teste geral dos elementos procedurais, a quantidade de instâncias utilizada para o nível 3 de estresse correspondeu à mesma empregada no nível 1 de estresse dos respectivos elementos — com exceção do elemento água. Para os níveis de estresse 2 e 1, foram utilizados, respectivamente, dois terços e um terço da quantidade do nível 3, com alguns ajustes realizados de forma empírica. Assim, a distribuição do número de instâncias ocorreu da seguinte maneira:

- **Nível 1:** 7 instâncias de aranhas, 4 instâncias de fogo, a mesma área de água utilizada no nível 1 do teste específico da água e 1 instância de tecido.
- **Nível 2:** 20 instâncias de aranhas, 6 instâncias de fogo, mesma área de água do nível 2 e 2 instâncias de tecidos.
- **Nível 3:** 30 instâncias de aranhas, 8 instâncias de fogo, área de água correspondente ao nível 3 e 3 instâncias de tecidos.

Figura 39 – Níveis de água nos testes da água procedural



Fonte: Imagem criada pelo próprio autor deste trabalho

Foi observado no início dos testes que as aranhas animadas com a técnica tradicional apresentaram um desempenho inferior em comparação aos demais elementos tradicionais. Em contrapartida, todos os outros elementos desenvolvidos de forma tradicional mantiveram um ótimo desempenho, mesmo com um número elevado de instâncias, superando os correspondentes procedurais.

Devido a essa diferença, o número de instâncias dos elementos tradicionais — com exceção da aranha e da água — foi aumentado, a fim de possibilitar uma comparação mais justa. Isso porque, com quantidades menores, os elementos tradicionais mantinham consistentemente a taxa de 60 quadros por segundo, mesmo em dispositivos mais antigos. Outro fator considerado na definição dessas quantidades foi garantir que os elementos permanecessem dentro dos limites do mapa e do campo de visão da câmera, evitando distorções nos resultados dos testes em relação à otimizações da Godot.

Considerando o desempenho da aranha tradicional, o número de instâncias desse elemento foi igual ao da versão procedural, permitindo uma comparação mais justa em ambas

Tabela 2 – Números de instâncias nos testes tradicionais

Quantidade elementos	Aranha	Fogo	Tecido
Nível 1	30	320	120
Nível 2	60	640	240
Nível 3	90	960	400

Fonte: Elaborado pelo próprio autor deste trabalho.

as abordagens. Ressalta-se ainda que o elemento água utilizou a mesma área de atuação da versão procedural, garantindo paridade entre os testes. A tabela 2 apresenta a quantidade exata de instâncias utilizadas para os demais elementos.

No teste geral dos elementos tradicionais, para evitar sobrecarga em dispositivos com menor capacidade de processamento, optou-se por reduzir a quantidade de tecido e fogo em relação aos testes anteriores, dividindo seus respectivos valores por um fator de 3. Essa redução, no entanto, resultou em uma diminuição significativa da carga computacional do teste. Para equilibrar esse fator e manter a exigência de desempenho, decidiu-se aumentar a quantidade de aranhas tradicionais, já que esse elemento possui um consumo computacional mais elevado. A base para esse aumento foi a quantidade de aranhas utilizada no teste geral dos elementos procedurais, sendo multiplicada por 3. Com isso, a distribuição final dos elementos ficou da seguinte forma:

- **Nível 1:** 14 instâncias de aranhas, 106 instâncias de fogo, a mesma área de água utilizada no nível 1 do teste específico da água e 40 instância de tecido.
- **Nível 2:** 60 instâncias de aranhas, 213 instâncias de fogo, mesma área de água do nível 2 e 80 instâncias de tecidos.
- **Nível 3:** 90 instâncias de aranhas, 320 instâncias de fogo, área de água correspondente ao nível 3 e 133 instâncias de tecidos.

4 RESULTADOS E DISCUSSÕES

Nesta seção, são apresentados e analisados os resultados dos testes de estresse realizados com os elementos animados por meio de técnicas procedurais e tradicionais, utilizando as máquinas e dispositivos descritos na seção 3.

Como mencionado anteriormente na seção 3, os testes em computadores *desktops* e *notebooks* foram realizados a partir do sistema Ubuntu executado por um pendrive, para assim garantir a uniformidade dos resultados e evitar variações causadas por diferentes sistemas operacionais.

Após a execução dos testes e coleta dos dados, os valores de FPS foram processados e organizados em gráficos de barras. Essa visualização facilita a interpretação dos resultados e a comparação entre as técnicas de animação. Para cada elemento, foram comparadas as versões procedural e tradicional, reunindo os dados de todas as máquinas em um único gráfico para facilitar a análise.

Os gráficos mostram a média de FPS obtida em cada máquina e dispositivo móvel para os três níveis de estresse: nível 1, nível 2 e nível 3, representados pelas siglas L1, L2 e L3, respectivamente. Além disso, cada barra do gráfico é acompanhado por um intervalo de confiança de 95%, calculado com base no teste estatístico *t-student*. Este intervalo indica a margem de erro esperada em torno da média, permitindo uma análise mais robusta e confiável.

4.1 Teste de estresse do elemento aranha

A figura 40 apresenta os resultados de FPS obtidos nos testes de estresse com o elemento aranha, comparando as versões animadas por técnicas tradicionais e procedurais. De modo geral, observa-se que a versão tradicional tende a apresentar um melhor desempenho computacional na maioria dos dispositivos testados. A única exceção ocorre no dispositivo Mobile 3, onde a aranha animada proceduralmente apresentou um desempenho levemente superior à versão tradicional.

No teste realizado no dispositivo Mobile 3, com o elemento aranha sob nível de estresse 3, observou-se que a versão animada por técnica tradicional apresentou média de 1,05 FPS (desvio padrão de 0,22), enquanto a versão com animação procedural obteve média de 1,16 FPS (desvio padrão de 0,37). Esse comportamento se mostrou também nos níveis 1 e 2 de estresse do elemento aranha no dispositivo Mobile 3.

Como, nos demais dispositivos, a versão tradicional geralmente apresentou desempenho superior à versão procedural, esse resultado foi considerado atípico. Para investigar se essa diferença observada no Mobile 3 é estatisticamente significativa ou apenas fruto do acaso, foi realizada uma análise estatística mais aprofundada.

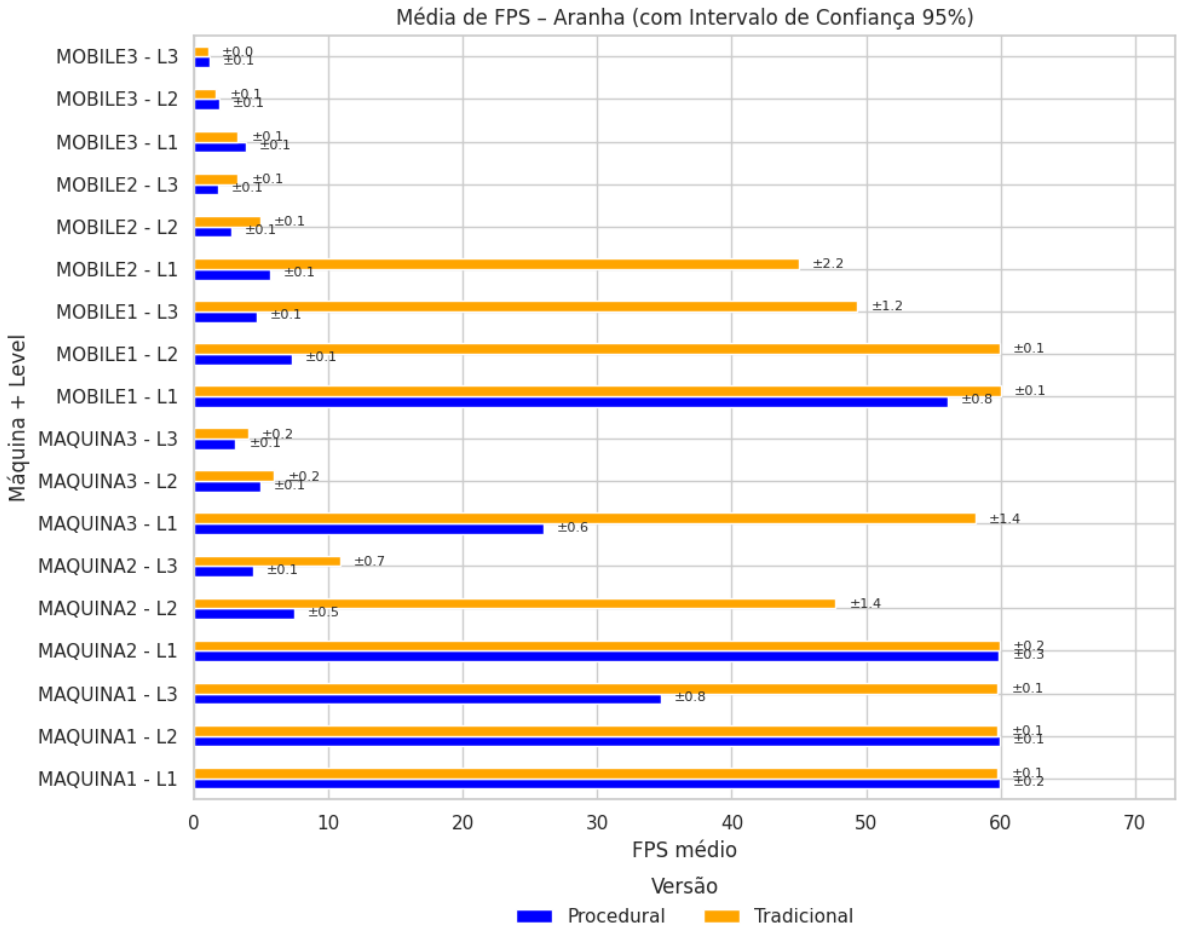
Inicialmente, foi verificado se os conjuntos de FPS das animações procedural e tradicional apresentavam distribuição normal, utilizando o teste de *Shapiro-Wilk*. O teste retornou um valor de p significativamente inferior a 0,05, indicando que os dados não seguem uma distribuição normal.

Dado que os dados não eram normalmente distribuídos, foi aplicado o teste de *Mann-Whitney U*, apropriado para comparações entre dois grupos independentes em distribuições não normais. O teste também resultou em um valor de p muito inferior a 0,05, demonstrando que a diferença entre os desempenhos médios é estatisticamente significativa.

Assim, conclui-se que, mesmo que a diferença entre os valores médios seja pequena, o desempenho da animação procedural foi consistentemente superior ao da animação tradicional no elemento aranha, especificamente no dispositivo Mobile 3.

Na Máquina 1 e no dispositivo Mobile 1, que são os dispositivos mais potentes entre os testados, a aranha procedural obteve resultados próximos a 60 FPS no Nível 1 de estresse, indicando que, com poucas instâncias, o desempenho é satisfatório em equipamentos mais recentes. No entanto, à medida que o número de instâncias aumenta — especialmente no Nível 3 — observa-se uma queda significativa na taxa de quadros. Mesmo na máquina mais potente, o FPS nesse nível variou entre 30 e 40.

Figura 40 – Resultados de FPS dos elementos procedural e tradicional da aranha



Fonte: Elaborado pelo próprio autor deste trabalho

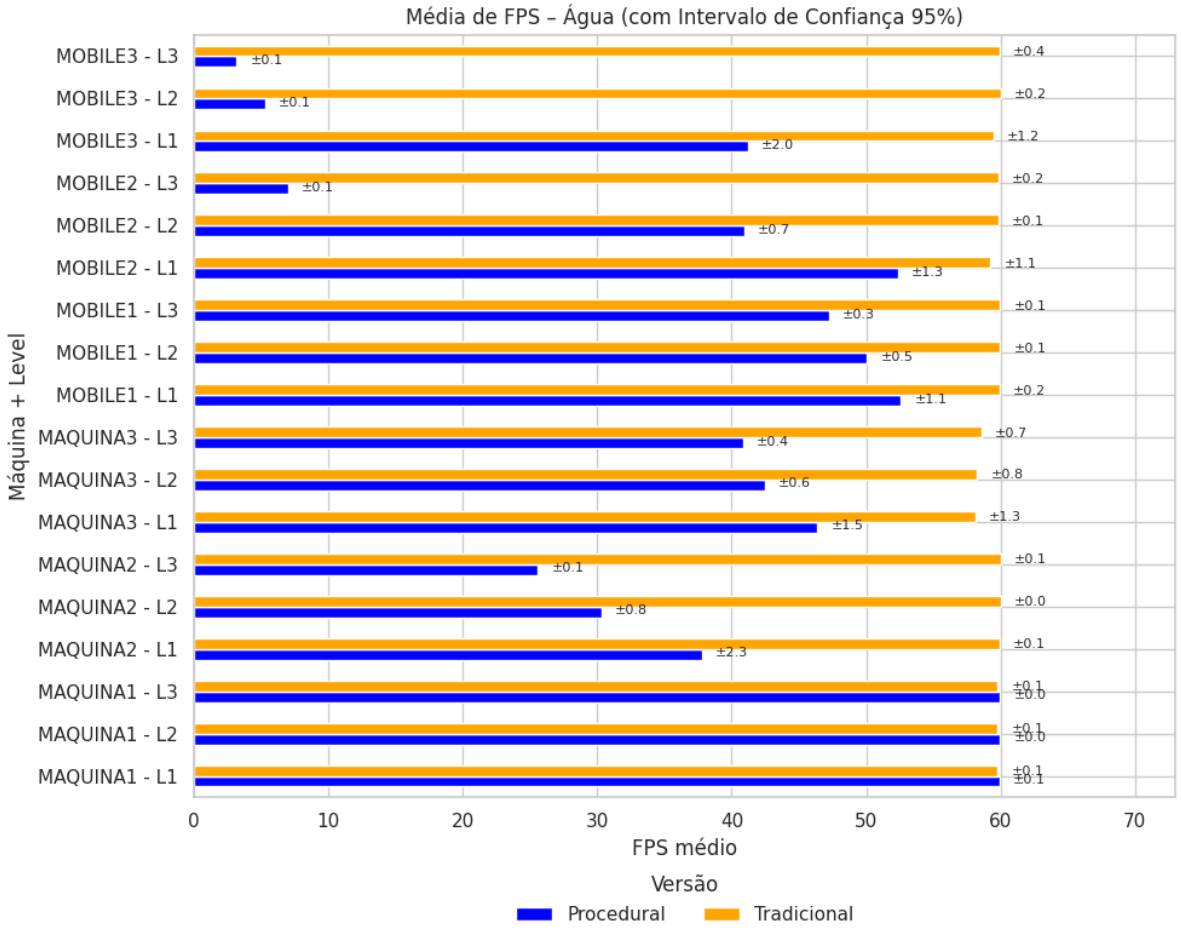
Nas demais máquinas, que são mais antigas e possuem menos recursos computacionais, até mesmo o nível 1 de estresse apresentou desempenho limitado, geralmente ficando abaixo de 30 FPS — com exceção da Máquina 2, que obteve resultados satisfatórios. Esses dados

evidenciam que elementos complexos como a aranha, que possuem movimentações autônomas e múltiplos membros animados proceduralmente, exigem um poder de processamento mais elevado, impactando diretamente no desempenho conforme o número de instâncias aumenta.

4.2 Teste de estresse do elemento água

No caso da água animada tradicionalmente, todas as máquinas e dispositivos móveis — desde os mais potentes até os mais limitados — apresentaram um ótimo desempenho. Em quase todos os testes, a taxa de quadros por segundo se manteve próxima a 60 FPS, com apenas algumas variações abaixo desse valor. Esse bom desempenho pode ser atribuído ao sistema de *TileMap* da Godot, que realiza otimizações internas dos elementos renderizados. É importante destacar que, tanto na água tradicional quanto na procedural, há uma caixa de colisão em toda a sua área, permitindo que o jogador detecte a presença do elemento corretamente, mas mesmo assim não foi o suficiente para diminuir o desempenho.

Figura 41 – Resultados de FPS dos elementos procedural e tradicional da água



Fonte: Elaborado pelo próprio autor deste trabalho

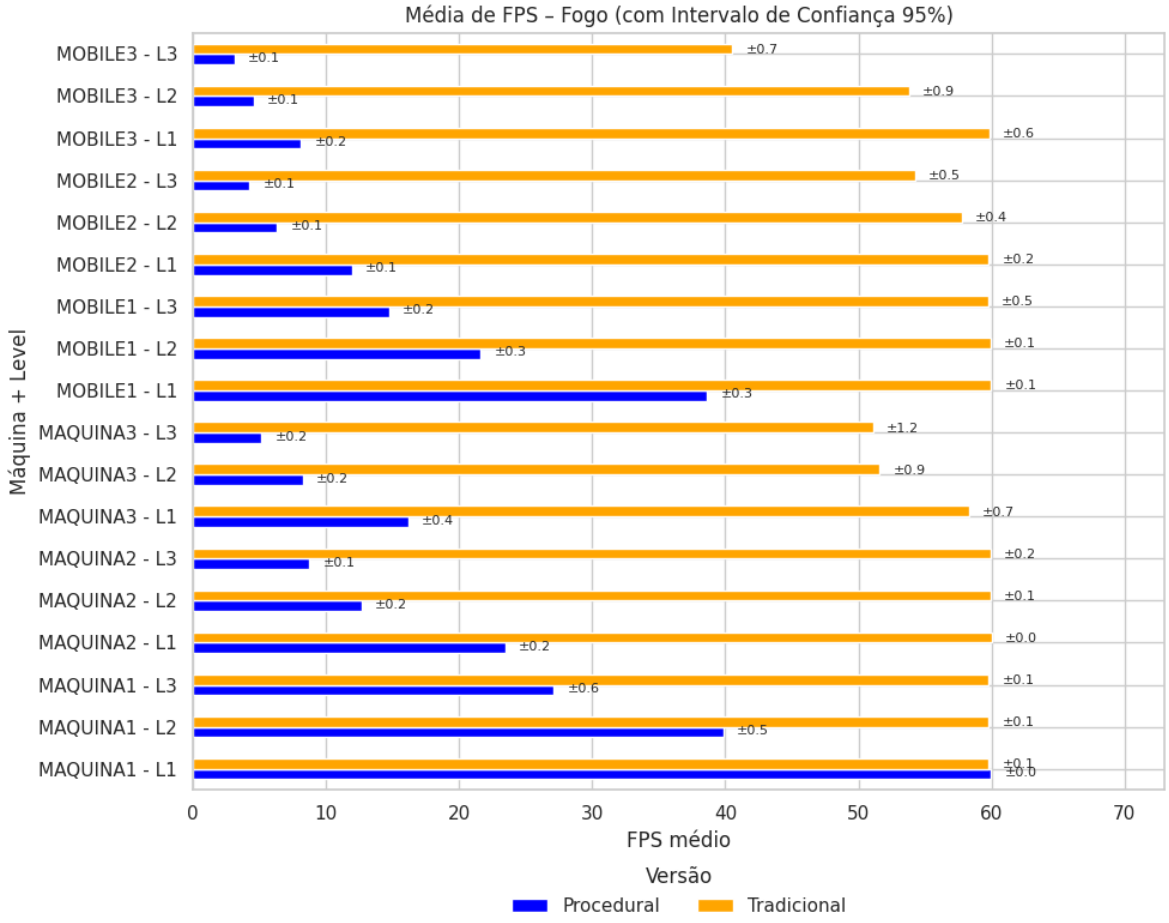
Já no caso da água animada proceduralmente, observou-se que a Máquina 1 manteve um desempenho estável de 60 FPS em todos os níveis de estresse, evidenciando sua alta capacidade

de processamento. No entanto, nas demais máquinas e dispositivos, seguiu-se um padrão: quanto maior a área coberta pela água, menor foi a taxa de FPS; inversamente, quanto menor a área, maior o desempenho.

Vale destacar casos específicos como o do dispositivo Mobile 2, que apresentou uma diminuição significativa de desempenho do Nível 2 para o Nível 3 de estresse. Já o dispositivo Mobile 3 demonstrou uma queda abrupta entre os Níveis 1 e 2, indicando que equipamentos mais antigos enfrentam maiores dificuldades ao lidar com áreas extensas de água e múltiplas caixas de colisões presentes nas unidades de água procedural.

4.3 Teste de estresse dos elementos fogo e tecido

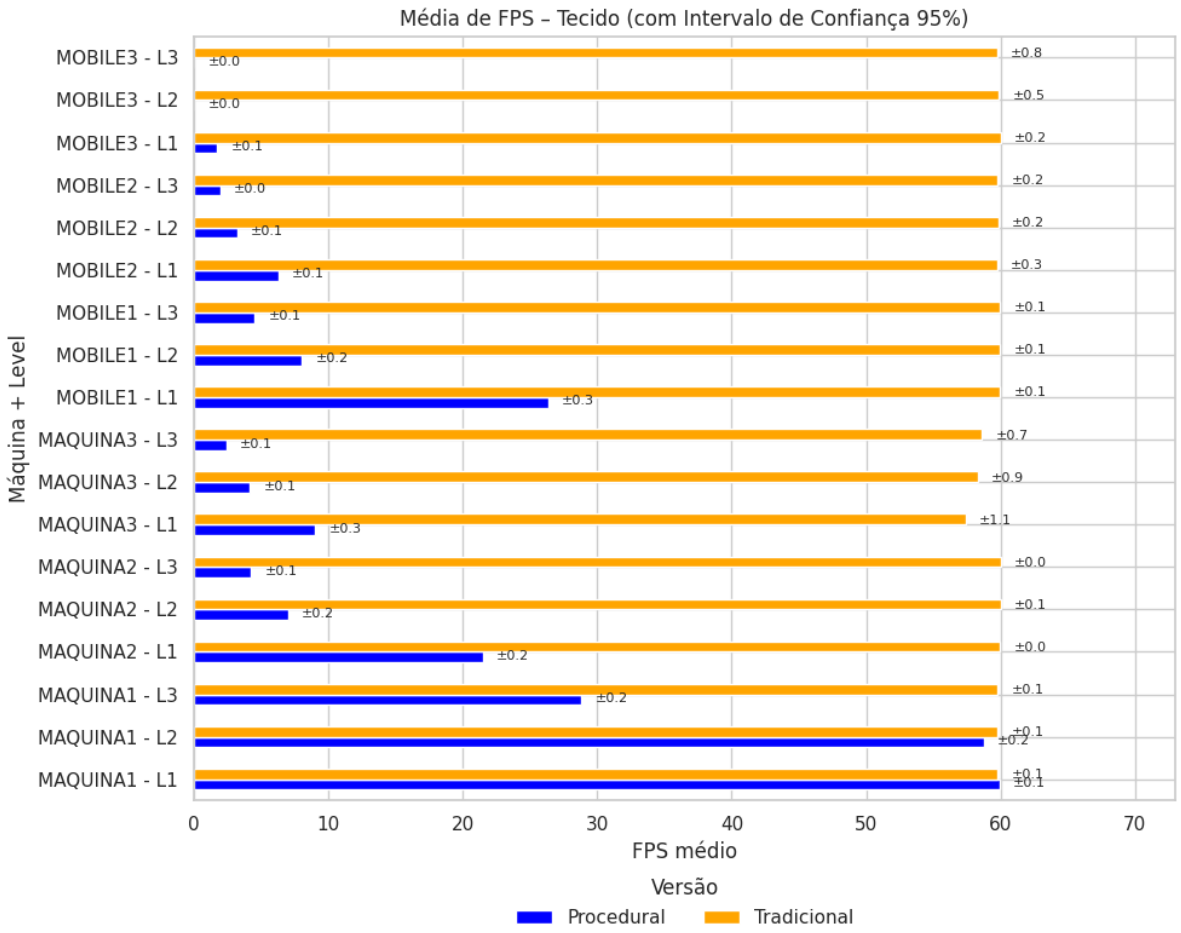
Figura 42 – Resultados de FPS dos elementos procedural e tradicional do fogo



Fonte: Elaborado pelo próprio autor deste trabalho

As figuras 42 e 43 apresentam os resultados dos testes de estresse dos elementos fogo e tecido, respectivamente. Nas versões animadas tradicionalmente, ambos os elementos demonstraram desempenho semelhante em todos os dispositivos testados, mantendo uma média de FPS entre 50 e 60. A única exceção foi o elemento fogo no dispositivo Mobile 3, que, no nível 3 de estresse, obteve uma média ligeiramente acima dos 40 FPS. Esses resultados mostram que,

Figura 43 – Resultados de FPS dos elementos procedural e tradicional do tecido



Fonte: Elaborado pelo próprio autor deste trabalho

mesmo em dispositivos antigos e com menor capacidade de processamento, elementos cuja animação é puramente estética tendem a apresentar ótimo desempenho, permitindo a inclusão de centenas de instâncias sem comprometer significativamente o desempenho.

Já em suas versões animadas proceduralmente, ambos os elementos seguiram o padrão esperado: quanto maior o número de instâncias, menor é o FPS; e quanto menor o número de instâncias, maior é o FPS. Também em ambos os elementos, os dispositivos com menos poder computacional mostraram um péssimo desempenho para a versão procedural.

O dispositivo Mobile 3 foi o que apresentou os piores resultados na versão procedural de ambos os elementos. No caso do elemento tecido, o dispositivo não conseguiu sequer executar os testes nos Níveis de estresse 2 e 3, travando completamente assim que os testes eram iniciados. Isso demonstra que, em dispositivos mais antigos, a simulação de um grande número de instâncias de tecido não se mostrou viável. Esse comportamento se deve à quantidade elevada de cálculos necessários em cada *RopePoint* e *Joint*, além dos cálculos de renderização para desenhar cada uma das instâncias em tempo real e as colisões presentes nesses pontos.

4.4 Teste de estresse geral dos elementos procedurais e tradicionais

A figura 44 apresenta os resultados de FPS obtidos no teste de estresse geral, tanto para os elementos animados proceduralmente quanto para os animados tradicionalmente. Nos testes, os elementos com animação tradicional demonstraram excelente desempenho nos dispositivos mais potentes, como a Máquina 1 e o dispositivo Mobile 1. Já as máquinas intermediárias mantiveram um desempenho aceitável nos níveis iniciais de estresse; no entanto, no nível 3, houve uma queda significativa na taxa de quadros, com médias abaixo de 10 FPS em todos os casos.

Os testes com animações procedurais apresentaram ótimo desempenho nos níveis de estresse 1 e 2 na Máquina 1, resultado atribuído ao alto poder de processamento desse equipamento. Em contraste, as demais máquinas tiveram desempenho insatisfatório, mesmo no nível 1 de estresse, com médias de FPS abaixo de 20. Nos dispositivos mobile, apenas o Mobile 1 obteve um desempenho razoável, com média próxima a 30 FPS no Nível 1 de estresse. Os demais testes nos outros dispositivos apresentaram resultados significativamente inferiores, com médias geralmente abaixo de 10 FPS.

O dispositivo Mobile 3, mais uma vez, foi o que apresentou o pior desempenho entre todos os equipamentos testados na versão procedural. O dispositivo não conseguiu sequer iniciar o teste no Nível 1 de estresse, repetindo o mesmo comportamento observado nos níveis 2 e 3 do teste com o elemento tecido procedural: o sistema travava assim que o teste era iniciado.

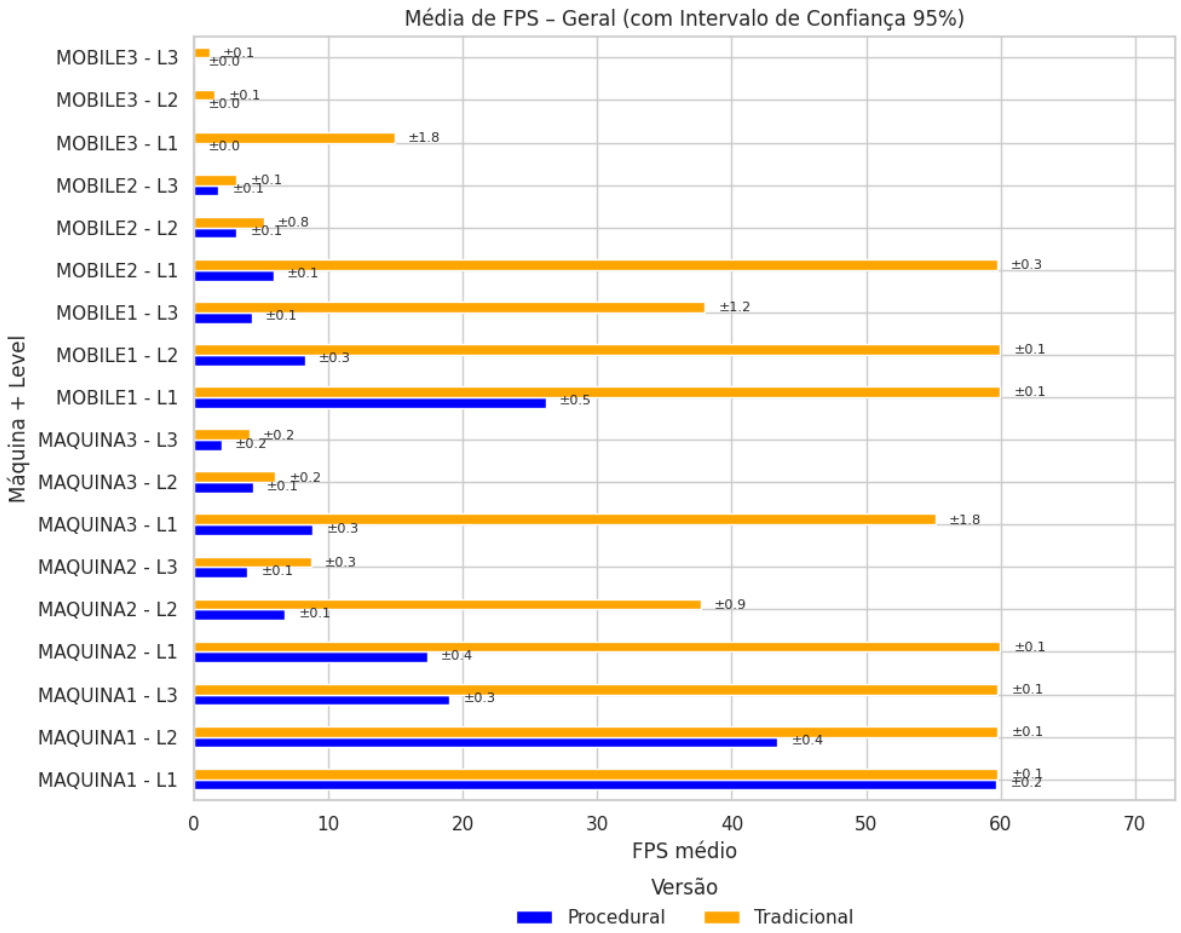
Esses resultados evidenciam que a execução simultânea de múltiplas entidades e elementos procedurais, que envolvem diversos cálculos em tempo real, não é bem suportada por dispositivos mais antigos ou com baixo poder computacional. Por outro lado, mesmo apresentando baixo desempenho, o dispositivo Mobile 3 foi capaz de executar os testes com elementos animados tradicionalmente.

Esse comportamento evidencia que elementos animados de forma tradicional tendem a demandar um poder computacional menor quando não há um grande número de instâncias ativas. Por outro lado, as animações procedurais, mesmo que ofereçam uma maior flexibilidade e dinamicidade visual, mostraram-se mais sensíveis ao aumento da complexidade e ao aumento de instâncias ativas. Essa limitação fica mais clara em dispositivos mais datados ou com menor capacidade de *hardware*, como o dispositivo Mobile 3, que não conseguiu executar os testes nos níveis mais altos de estresse.

Esses resultados mostram a importância de avaliar o equilíbrio entre qualidade visual, complexidade técnica e desempenho, quando se for fazer um jogo que busca ter compatibilidade em uma alta quantidade de dispositivos e máquinas diferentes.

Para uma análise mais detalhada, o restante dos testes de estresse encontram-se no apêndice A, uma vez que seu volume comprometeria a fluidez do texto principal. Neste apêndice, são apresentados dados como a média de colisões detectadas, a média dos tempos de execução das funções *physics_process* e *process* na Godot, além dos respectivos desvios padrão, permitindo uma análise mais precisa e aprofundada do comportamento do sistema sob diferentes níveis de estresse.

Figura 44 – Resultados de FPS de teste geral de estresse dos elementos procedural e tradicional



Fonte: Elaborado pelo próprio autor deste trabalho

Dados como o uso de memória RAM, memória de vídeo e memória destinada a texturas foram excluídos da análise principal devido ao seu impacto insignificante e ao grande volume de informações geradas. Observou-se que o consumo de RAM não ultrapassou 80 MB e geralmente ficando em torno de 50 MB, o uso de memória de vídeo manteve-se em torno de 40 MB, e o uso de memória para texturas ficou próximo de 10 MB. Dessa forma, optou-se por não incluí-los nos resultados detalhados, visto que não contribuíram significativamente para a interpretação do desempenho.

Por fim, destaca-se a realização de uma análise de correlação por meio do teste de *Kendall*, com o objetivo de verificar a existência de uma relação entre o número de pares de colisão detectados e a taxa de FPS, os resultados são mostrados na tabela 9 no apêndice B. Essa análise revelou alguns resultados interessantes. Por exemplo, no teste geral dos elementos procedurais na Máquina 1, sob o Nível 3 de estresse, foi obtido um valor de *Tau* = -0,568, indicando uma correlação negativa. Isso significa que, nesse cenário específico, à medida que aumentava a quantidade de colisões físicas detectadas, o desempenho diminuía. De fato, a média de FPS registrada nesse caso foi de apenas 19 quadros por segundo, evidenciando o impacto do aumento da complexidade física na performance.

Dois testes que apresentaram correlação próxima ao patamar considerado moderado (valores de *tau* entre 0,5 e 0,7, tanto para valores positivos quanto para negativos) foram os da versão procedural do elemento água, ambos realizados no Nível 1 de estresse, respectivamente na Máquina 2 e no dispositivo Mobile 1. Em ambos os casos, observou-se uma correlação negativa, com valores de aproximadamente -0,45 e -0,42, indicando que, quanto maior o número de colisões entre o jogador e a superfície da água, menor foi o desempenho em termos de FPS.

Os demais resultados do teste de *Kendall* apresentaram correlações abaixo do limite considerado moderado, mas estão documentados na tabela 9, localizada no apêndice B.

5 CONCLUSÃO E TRABALHOS FUTUROS

Com base nos elementos desenvolvidos, nos testes de estresse realizados e na experiência prática apresentada neste estudo, foi possível conduzir uma análise comparativa entre as animações procedurais e tradicionais aplicadas a jogos de plataforma 2D. A partir dessa análise, alguns aspectos dos dados obtidos também puderam ser extrapolados para outros tipos de jogos com características semelhantes. Além disso, as questões de pesquisas formuladas anteriormente foram respondidas com base nos resultados e observações coletadas ao longo do desenvolvimento e dos testes, sendo discutidas detalhadamente nesta seção.

5.1 Sumário da pesquisa

Sobre a questão de pesquisa **QP1**: Em quais contextos cada método de animação (tradicional e procedural) pode proporcionar uma redução nos custos de trabalho humano, e quais são suas principais vantagens e desvantagens em termos de qualidade visual, flexibilidade e complexidade de implementação?

A animação procedural, apesar de exigir maior conhecimento técnico e ser inicialmente mais desafiadora para desenvolvedores iniciantes, apresenta vantagens significativas na redução de custos de trabalho humano a médio e longo prazo. Uma vez compreendidos os conceitos fundamentais e os métodos necessários para alcançar resultados visuais satisfatórios, a animação procedural permite a criação eficiente de objetos com comportamentos dinâmicos, variações complexas e respostas aleatórias ao ambiente, sem a necessidade de múltiplos desenhos gráficos.

Além disso, o processo de ajustes e refatoração é consideravelmente mais ágil, já que não demanda a recriação de animações quadro a quadro - bastando, muitas vezes, a modificação de parâmetros no código. Um exemplo claro dessa vantagem é o elemento fogo animado proceduralmente: alterar sua intensidade ou tamanho requer apenas a manipulação de algumas variáveis. Em contraste, na versão tradicional, seria necessário redesenhar toda a sequência de animação para representar o novo comportamento, o que implica maior esforço artístico e mais tempo de produção.

Já a animação tradicional se mostra mais vantajosa em contextos onde o comportamento do objeto é simples, com poucas variações e um visual fixo. Nesses casos, desenhar manualmente as animações pode ser mais rápido e eficiente. Um exemplo clássico em jogos de plataforma é o de um baú se abrindo, que requer apenas alguns quadros representando a tampa se levantando - algo que pode ser feito com agilidade por meio de animação tradicional.

Ademais, a animação tradicional exige menos conhecimento técnico de algoritmos, sendo mais acessível para equipes iniciantes ou com pouca familiaridade com programação. O treinamento necessário para produzir esse tipo de animação é geralmente mais curto e simples do que aquele exigido para implementar sistemas procedurais, que envolvem lógica, estruturas matemáticas e domínio do motor de jogo.

Por fim, as animações tradicionais oferecem aos artistas maior controle criativo sobre o

resultado final. Ao aplicar os princípios da animação, é possível desenhar movimentos e estéticas com precisão e intenção artística, sem a necessidade de lidar com a complexidade matemática e computacional das animações procedurais. Isso permite que elementos animados de forma tradicional apresentem uma liberdade estética superior, sendo ideais para representar movimentos estilizados ou cenas visualmente específicas que seriam difíceis de reproduzir com exatidão por meio de algoritmos procedurais.

Sobre a questão de pesquisa **QP2**: qual é o impacto do desempenho computacional de cada método e suas técnicas específicas em diferentes tipos de *hardware*?

Os objetos desenvolvidos com propósito exclusivamente estético, utilizando animações tradicionais, apresentaram bom desempenho em todos os dispositivos testados, inclusive naqueles com menor poder computacional. Mesmo com um número elevado de instâncias ativas, o desempenho manteve-se estável, evidenciando que, quando o elemento animado não exige lógica complexa ou cálculos em tempo real, a animação tradicional é bastante eficiente. É importante destacar que, mesmo ao multiplicar o número de instâncias, a performance permaneceu satisfatória.

Por outro lado, elementos tradicionalmente animados que exigem recursos adicionais, como colisões grandes e múltiplas interações — a exemplo da aranha — apresentaram desempenho inferior aos elementos puramente visuais. Em máquinas menos potentes, com elevado número de instâncias, observou-se uma queda acentuada no FPS, embora ainda superior à das versões procedurais correspondentes.

Quanto às animações procedurais, observou-se um padrão claro: quanto maior o número de instâncias ativas, menor o FPS obtido. Além disso, o desempenho diminui proporcionalmente ao poder computacional do dispositivo. Mesmo em níveis baixos de estresse (nível 1), dispositivos menos potentes apresentaram resultados significativamente inferiores. Isso demonstra que animações procedurais têm um custo computacional elevado, principalmente por exigirem cálculos contínuos a cada quadro, como física, lógica e renderização dinâmica.

De forma geral, as animações procedurais demandam maior capacidade de processamento do que as animações tradicionais. Para garantir uma experiência satisfatória em máquinas e dispositivos mais antigos, é essencial que tais sistemas sejam cuidadosamente otimizados. Em contraste, animações tradicionais — especialmente aquelas de caráter estético e com baixa complexidade — demonstraram excelente desempenho mesmo com muitas instâncias ativas, sendo uma opção viável e eficiente para dispositivos com menor capacidade de processamento.

Sobre a questão de pesquisa **QP3**: Em que medida cada método de animação impacta a jogabilidade e a experiência do usuário em jogos de plataforma 2D, e em quais situações as animações procedurais podem contribuir para aumentar o realismo e a responsividade das interações em tempo real, em comparação com as animações tradicionais?

As animações procedurais impactam diretamente a jogabilidade e a experiência do usuário ao oferecer um nível elevado de responsividade e realismo, isso ocorre em todo tipo de jogo que os implementa, inclusive jogos de plataforma 2D. Como esses elementos são processados em

tempo real, suas animações podem reagir diretamente a variáveis e eventos do mundo do jogo. Um exemplo claro disso é o elemento aranha, cuja movimentação se adapta automaticamente a terrenos irregulares ou desníveis, proporcionando uma sensação natural e verossímil. Outro exemplo é o elemento água, que pode ser configurado para reagir a colisões externas de forma intensa ou sutil, criando efeitos visuais únicos dependendo da situação.

Esse tipo de comportamento dinâmico reforça o realismo e a imersão, sendo especialmente útil em jogos que exigem um nível maior de complexidade nas interações entre entidades e o ambiente. Em cenários com muitos obstáculos ou terrenos variados, a animação procedural se mostra mais eficaz do que a tradicional, pois elimina a necessidade de criar várias animações específicas para cada situação.

Além disso, a animação procedural é vantajosa para personagens ou inimigos de grande porte. Nesses casos, criar animações tradicionais que se adaptem de forma convincente a diferentes obstáculos pode ser extremamente trabalhoso e limitado, já que um pequeno obstáculos faz com que o inimigo faça uma elevação total, e caso não seja bem planejado e implementado, pode resultar em resultados esteticamente inusual. Já com técnicas procedurais, é possível garantir que grandes entidades reajam de maneira mais fluida e realista, mesmo em ambientes complexos a solução para o problema mencionado se torna relativamente simples.

Por outro lado, as animações tradicionais influenciam positivamente a experiência do jogador em elementos que exigem maior riqueza visual e controle artístico. Criar certos movimentos com animação procedural pode demandar lógica matemática complexa, enquanto a animação tradicional permite ao artista liberdade criativa para elaborar movimentos e reações com maior expressividade.

Dessa forma, jogos que adotam animações tradicionais, embora abram mão de parte da responsividade, podem apresentar visuais mais detalhados e expressivos, com reações específicas conforme a intenção do artista. Essa abordagem é ideal para objetos do cenário que não necessitam de resposta em tempo real, como baús, elementos decorativos e inimigos simples com poucos estados de animação.

Conclui-se que ambos os métodos possuem vantagens e limitações. Enquanto a animação procedural oferece maior realismo e interatividade em troca de maior uso computacional, a animação tradicional proporciona leveza e riqueza estética com menor custo de processamento. A escolha entre uma abordagem e outra depende dos objetivos do projeto, das capacidades da equipe de desenvolvimento e das limitações do *hardware* alvo.

5.2 Trabalhos futuros

Considerando as técnicas exploradas e os resultados obtidos ao longo deste estudo, este trabalho pode servir como base para investigações futuras em áreas relacionadas. A seguir, são apresentados alguns caminhos possíveis de expansão, que visam aprofundar a pesquisa sobre animações procedurais e tradicionais em jogos digitais. Esses tópicos podem contribuir para o avanço da área, oferecendo novos *insights* e aplicações práticas em contextos diversos.

- **Otimização de algoritmos procedurais em jogos de plataforma 2D para dispositivos de baixo desempenho**: Investigar, testar e implementar técnicas de otimização, como simplificação de cálculos físicos, paralelização por meio de *multithreading*, e uso eficiente de recursos computacionais, com o objetivo de melhorar o desempenho de elementos animados proceduralmente em dispositivos com poder computacional limitado.
- **Uso de animações procedurais em jogos de plataforma 2D para aprendizagem de comportamentos animais e conceitos de física**: Desenvolvimento de jogos sérios voltados para o público infantil, utilizando animações procedurais para representar o comportamento de animais ou a simulação de fenômenos físicos (como fluidos e gravidade), com o objetivo de promover a educação lúdica e interativa.
- **Estudo com usuários para medir o impacto na experiência e na imersão**: Realização de testes controlados com usuários reais para avaliar a percepção de realismo, responsividade, engajamento e diversão em jogos que utilizam animações procedurais em comparação com animações tradicionais, contribuindo para a compreensão do impacto dessas técnicas na experiência do jogador.

REFERÊNCIAS

ABREU, A. R. *et al.* Geração procedural de coreografias para jogos de dança. *In*: UNIVERSIDADE DO VALE DO RIO DOS SINOS, ESCOLA POLITéCNICA. **SBC – Proceedings of SBGames 2017, Workshop G2: Undergraduates**. Curitiba, Brasil: Sociedade Brasileira de Computação, 2017.

ADAMCYOUNIS. **Pixel Art Class - Animating Fire!** 2020. Disponível em: https://www.youtube.com/watch?v=b0iyDxpECJE. Acesso em: 04 mai. 2025.

BAHIA, S.; IZOLANI, L. G.; STEIN, M. **12 Princípios de Animação em Jogos de Luta**. Florianópolis, Brasil: SBC - Sociedade Brasileira de Computação, 2020. Programa de Graduação em Animação, UFSC. Disponível em: https://www.sbgames.org/proceedings2020/ArtesDesignFull/209727.pdf. Acesso em: 25 mai. 2025.

BARBOSA, A. V.; MASSUKADO, G. S.; NAKAMURA, R. **Aplicação de animação procedural e Inteligências Artificiais independentes do jogador em jogos digitais**. 2021.

BERRY, N. **Reddit Comment on Water Effects in Games**. 2018. Disponível em: https://www.reddit.com/r/gamedev/comments/9a0cfr/comment/e4rvrg2/?utm_source=share&utm_medium=web2x&context=3. Acesso em: 12 out. 2024.

BRATHWAITE, B.; SCHREIBER, I. **Challenges for Game Designers**. Rockland, MA: Charles River Media, Inc., 2008.

BROOKHAVEN, N. L. **The First Video Game?** 2024. Disponível em: https://www.bnl.gov/about/history/firstvideo.php. Acesso em: 19 out. 2024.

CANEDO, G. N. **Criação de mapas utilizando geração procedural**. Goiânia: [s.n.], 2022. Disponível em: https://repositorio.pucgoias.edu.br/jspui/handle/123456789/5077. Acesso em: 18 out. 2024.

COSTA, E. A. d. S.; ROSA, M. Historiando o desenvolvimento do desenho geomÉtrico: Possibilidades do retorno desse conhecimento no currÍculo escolar. **VIDYA**, v. 35, n. 1, p. 57–69, 2015. Disponível em: https://periodicos.ufn.edu.br/index.php/VIDYA/article/view/234. Acesso em: 16 set. 2024.

DEGLORIE, G. *et al.* **Procedural animation of human interaction using inverse kinematics and fuzzy logic**. 2015. 340–347 p.

DIGITAL, A. **How much does it cost to make a game?** 2021. Disponível em: https://www.aurochdigital.com/blog/2021/8/19/how-much-does-it-cost-to-make-a-game. Acesso em: 18 out. 2024.

Godot Engine Documentation. **2D Skeletons**. [S.l.], 2024. Disponível em: https://docs.godotengine.org/en/stable/tutorials/animation/2d_skeletons.html. Acesso em: 25 mai. 2025.

Godot Engine Documentation. **Overview of debugging tools**. 2024. Disponível em: https://docs.godotengine.org/en/stable/tutorials/scripting/debug/overview_of_debugging_tools.html. Acesso em: 25 mai. 2025.

Godot Engine Documentation. **The Profiler**. [S.l.], 2024. Disponível em: https://docs.godotengine.org/en/stable/tutorials/scripting/debug/the_profiler.html. Acesso em: 25 mai. 2025.

Godot Engine Documentation. **2D particle systems**. 2025. Disponível em: https://docs.godotengine.org/en/latest/tutorials/2d/particle_systems_2d.html. Acesso em: 17 mai. 2025.

Godot Engine Documentation. **Introduction to shaders**. 2025. Disponível em: https://docs.godotengine.org/en/stable/tutorials/shaders/introduction_to_shaders.html. Acesso em: 18 mai. 2025.

Godot Engine Documentation. **Introdução**. 2025. Disponível em: https://docs.godotengine.org/pt-br/4.x/about/introduction.html. Acesso em: 16 mai. 2025.

Godot Engine Documentation. **RayCast2D**. 2025. Disponível em: https://docs.godotengine.org/en/stable/classes/class_raycast2d.html. Acesso em: 16 mai. 2025.

Godot Engine Documentation. **Usando Area2D**. 2025. Disponível em: https://docs.godotengine.org/pt-br/4.x/tutorials/physics/using_area_2d.html. Acesso em: 16 mai. 2025.

Godot Engine Documentation. **Using CharacterBody2D/3D**. 2025. Disponível em: https://docs.godotengine.org/pt-br/4.x/tutorials/physics/using_character_body_2d.html. Acesso em: 16 mai. 2025.

Godot Engine Documentation. **Using TileMaps**. 2025. Disponível em: https://docs.godotengine.org/pt-br/4.x/tutorials/2d/using_tilemaps.html. Acesso em: 18 mai. 2025.

Godot Engine Documentation. **Visão geral dos principais conceitos do Godot**. 2025. Disponível em: https://docs.godotengine.org/pt-br/4.x/getting_started/introduction/key_concepts_overview.html. Acesso em: 16 mai. 2025.

GREEN, D. **Procedural Content Generation for C++ Game Development**. Birmingham, UK: Packt Publishing Ltd, 2016. ISBN 978-1-78588-671-3.

GREGORY, E. G. Undergraduate Thesis, **TCC**. 2022. Disponível em: https://dspace.mackenzie.br/items/02d4dd1f-df00-4a50-b2ef-23de75211404. Acesso em: 18 out. 2024.

HACKTROUT. **2D Dynamic Water**. 2020. Disponível em: https://github.com/HackTrout/2DDynamicWater. Acesso em: 26 abr. 2025.

HACKTROUT. **Simulating Cloth**. 2021. Disponível em: https://www.youtube.com/watch?v=D6biR0_lMfM. Acesso em: 15 mai. 2025.

HENDRIKX, M. *et al.* Procedural content generation for games: A survey. **ACM Trans. Multimedia Comput. Commun. Appl.**, Association for Computing Machinery, New York, NY, USA, v. 9, n. 1, fev. 2013. ISSN 1551-6857. Disponível em: https://doi.org/10.1145/2422956.2422957. Acesso em: 25 mai. 2025.

Hidden Variable Studios. **Skullgirls**. 2012. Disponível em: https://store.steampowered.com/app/245170/Skullgirls_2nd_Encore/. Acesso em: 25 mai. 2025.

INC., M. M. G.; GAMES, L. E. O. **Celeste**. 2018. Disponível em: https://www.celestegame.com/. Acesso em: 31 ago. 2024.

KIDSCANCODE. **Godot 3 Particles2D Demo**. 2019. Disponível em: https://github.com/kidscancode/godot3_particles2d_demo/tree/master. Acesso em: 15 mai. 2025.

KidsCanCode. **RayCast2D**. 2025. Disponível em: https://kidscancode.org/godot_recipes/4.x/kyn/raycast2d/index.html. Acesso em: 16 mai. 2025.

KidsCanCode. **Shaders: Intro**. 2025. Disponível em: https://kidscancode.org/godot_recipes/3.x/shaders/intro/index.html. Acesso em: 16 mai. 2025.

KOPF, J.; LISCHINSKI, D. **Depixelizing pixel art**. New York, NY, USA: Association for Computing Machinery, 2011. Disponível em: https://doi.org/10.1145/1964921.1964994. Acesso em: 15 mai. 2025.

LAMIN, B. K. **Animação como Ferramenta Publicitária: Produção de um Teaser Animado 2D para Lançamento de um Jogo Digital**. Florianópolis: [s.n.], 2020. Disponível em: https://repositorio.ufsc.br/bitstream/handle/123456789/218761/Relat%c3%b3rio%20PCC%20-%20Bruna%20Korb%20Lamin%202020%20Final.pdf?sequence=1&isAllowed=y. Acesso em: 14 out. 2014.

LIMA, L. B. R. de. **Execução de Animação em Formato Seriado: Potencializando o Tempo de Produção**. Florianópolis: [s.n.], 2016. Disponível em: https://repositorio.ufsc.br/bitstream/handle/123456789/164599/PCC2_final_v4.pdf?sequence=3. Acesso em: 14 out. 2024.

LOPES, L. J. Monografia, **Avaliação artística e técnica no desenvolvimento de um jogo com geração procedural de conteúdo**. Brasil: [s.n.], 2021. Disponível em: https://rd.uffs.edu.br/handle/prefix/5765. Acesso em: 25 mai. 2025.

MEGAMAN. **Jogos**. 2024. Disponível em: https://www.megaman.com.br/series. Acesso em: 21 out. 2024.

MEYER, L. L. Artigo Científico, **A Animação Volumétrica em O Sur(Real) Mundo de Any Malu: Do Quadro a Quadro à Técnica Digital de Recortes Aplicada à Personagem**. Pelotas/RS: [s.n.], 2016. Disponível em: https://acervosvirtuais.ufpel.edu.br/tccscinema/wp-content/uploads/sites/24/tainacan-items/16/6520/larissa_meyer_5020_2016_2.pdf. Acesso em: 14 out. 2014.

MIZIZIZIZ. **How to make a Limbo-style spider in Godot using procedural animation**. 2018. Disponível em: https://www.youtube.com/watch?v=qYwYgEGMdLA. Acesso em: 19 mai. 2025.

MLANARCZYKI, P. W. **Desenvolvimento de animações 2D quadro a quadro para um jogo "boss rush"**. 2023. Disponível em: https://repositorio.ufsc.br/handle/123456789/248612. Acesso em: 25 mai. 2025.

NORMAN, D. **The Design of Everyday Things**. Reprint edition. New York, NY: Doubleday/Currency, 2002. Originally published as "The Psychology of Everyday Things"in 1988 by Basic Books. ISBN 0-385-26774-6.

NUGROHO, S. A.; PRIHATMANTO, A. S.; ROHMAN, A. S. Design and implementation of kinematics model and trajectory planning for nao humanoid robot in a tic-tac-toe board game. *In*: **2014 IEEE 4th International Conference on System Engineering and Technology (ICSET)**. [S.l.: s.n.], 2014. v. 4, p. 1–7.

NUNES, R. F. **Mapeamento da Cinemática Inversa de um Manipulador Robótico Utilizando Redes Neurais Artificiais Configuradas em Paralelo**. Ilha Solteira, SP: [s.n.], 2016.

OLIVEIRA, E. L. F. de. **A Complexidade da Diversão nos Videogames: Um Estudo Ecossistêmico dos Modos de Sentir na Relação Jogo-Jogador em Celeste**. Manaus: [s.n.], 2020. Disponível em: https://tede.ufam.edu.br/handle/tede/8269. Acesso em: 18 out. 2024.

PANGESTI, A. R.; UTAMI, E.; SUNYOTO, A. Analysis of inverse kinematics method for human movement in 2d animation. *In*: **2019 1st International Conference on Cybernetics and Intelligent System (ICORIS)**. [S.l.: s.n.], 2019. v. 1, p. 189–194.

PRESS, C. U. **cutscene**. 2025. Disponível em: https://dictionary.cambridge.org/pt/dicionario/ingles/cutscene. Acesso em: 18 mai. 2025.

Proyectos JC. **Cinemática Inversa de Robot Planar 2gdl**. 2021. Disponível em: https://www.youtube.com/watch?v=7H-RdRkS6ro. Acesso em: 25 mai. 2025.

RIBEIRO, D. **Bug**. 2024. Disponível em: https://www.dicio.com.br/bug/. Acesso em: 05 nov. 2024.

ROBERTS, S. **Character Animation: 2D Skills for Better 3D**. 2nd. ed. Routledge, 2007. Disponível em: https://doi.org/10.4324/9780080488455. Acesso em: 25 mai. 2025.

ROGERS, S. **Level UP: Um Guia Para o Design de Grandes Jogos**. [S.l.]: Blucher, 2013.

SCABENI, P. H. **Aplicação de animação em jogos: desenvolvimento de animação 2D para jogo arcade**. 2019. Disponível em: https://repositorio.ufsc.br/handle/123456789/197308. Acesso em: 16 set. 2024.

SIMõES, G. H. P. **Emprego de técnicas e algoritmos para desenvolvimento de jogos realistas**. 2023. Disponível em: https://repositorio.pucgoias.edu.br/jspui/handle/123456789/7187. Acesso em: 25 mai. 2025.

STATISTA. **Video game market revenue worldwide from 2019 to 2029**. 2024. Disponível em: https://www.statista.com/forecasts/1344668/revenue-video-game-worldwide. Acesso em: 11 out. 2024.

Studio MDHR Entertainment Inc. **Cuphead**. 2017. Disponível em: https://store.steampowered.com/app/268910/Cuphead/. Acesso em: 25 mai. 2025.

SWYDEV. **How to make Fire in Godot 4 (Particles Tutorial)**. 2023. Disponível em: https://www.youtube.com/watch?v=qII0_DA9A1k. Acesso em: 04 mai. 2025.

Team Cherry. **Hollow Knight**. 2017. Disponível em: https://store.steampowered.com/app/367520/Hollow_Knight/. Acesso em: 25 mai. 2025.

TECHTUDO. **Saiba o que são games sandbox e os principais títulos do mercado**. 2022. Disponível em: https://www.techtudo.com.br/noticias/2014/12/saiba-o-que-sao-games-sandbox-e-os-principais-titulos-do-mercado.ghtml. Acesso em: 10 nov. 2024.

THOMAS, F.; JOHNSTON, O. **Disney Animation: The Illusion of Life**. Cidade: Disney Editions, 1995.

VALENTE, A. **Most Expensive Games Ever Made**. 2020. Disponível em: https://www.thegamer.com/most-expensive-games-ever-made/. Acesso em: 19 out. 2024.

VIDEOCULT. **Rain World**. 2017. Disponível em: https://rainworldgame.com/. Acesso em: 31 ago. 2024.

Virtuous Tecnologia da Informação. **O que são triângulos retângulos, acutângulos ou obtusângulos?** 1998–2024. https://www.somatematica.com.br/faq/triangulos.php. Acesso em: 25 mai. 2025.

WATKINS, R. **Procedural Content Generation for Unity Game Development: Harness the Power of Procedural Content Generation to Design Unique Games With Unity**. Birmingham, UK: Packt Publishing Ltd, 2016.

WAZLAWICK, R. S. **Metodologia de Pesquisa para Ciência da Computação**. [S.l.]: GEN LTC, 2020. ISBN 978-85-951-5109-3.

YU, J. **Stylised procedural animation**. Tese (PhD thesis) — University of Glasgow, College of Science and Engineering, School of Computing Science, Glasgow, UK, 1999. Copyright of this thesis is held by the author. Disponível em: https://theses.gla.ac.uk/id/eprint/6737. Acesso em: 25 mai. 2025.

ZACCA, F. *et al.* Cinemática inversa aplicada a manipulador robótico a partir de coordenadas capturadas pelo microsoft kinect. *In*: **Anais da VIII Escola Regional de Informática de Goiás**. Porto Alegre, RS, Brasil: SBC, 2020. p. 15–30. ISSN 0000-0000. Disponível em: https://sol.sbc.org.br/index.php/erigo/article/view/13859. Acesso em: 25 mai. 2025.

APÊNDICE A – DADOS DOS TESTES DE ESTRESSE

Tabela 3 – Métricas da Máquina 1

Elemento	Versão	Nível de Estresse	Colisões (média/dp)	Physics Process (ms)(média/dp)	Process (ms)(média/dp)
Água	Procedural	1	2.26/2.42	0.003/0.0147	0.0167/0.0022
Água	Procedural	2	3.43/2.28	0.0024/0.0012	0.0158/0.0003
Água	Procedural	3	4.67/1.65	0.0044/0.0051	0.0146/0.0013
Água	Tradicional	1	1.38/0.9	0.0008/0.0013	0.0172/0.0003
Água	Tradicional	2	1.39/0.69	0.0008/0.0012	0.0172/0.0002
Água	Tradicional	3	1.0/0.0	0.0008/0.0013	0.0172/0.0003
Aranha	Procedural	1	0.69/1.01	0.0042/0.0004	0.0144/0.0028
Aranha	Procedural	2	0.69/1.01	0.0085/0.0017	0.0113/0.0074
Aranha	Procedural	3	0.69/1.01	0.013/0.0023	0.0143/0.0115
Aranha	Tradicional	1	62.89/13.12	0.0026/0.0002	0.0156/0.0003
Aranha	Tradicional	2	116.83/20.08	0.0044/0.0011	0.014/0.0004
Aranha	Tradicional	3	160.49/28.07	0.0061/0.0011	0.0124/0.0005
Fogo	Procedural	1	0.69/1.01	0.0009/0.0018	0.0172/0.0003
Fogo	Procedural	2	0.69/1.01	0.0008/0.0016	0.0284/0.0016
Fogo	Procedural	3	0.69/1.01	0.0007/0.0019	0.0416/0.0022
Fogo	Tradicional	1	1.36/1.07	0.001/0.0015	0.0172/0.0003
Fogo	Tradicional	2	1.36/1.07	0.0012/0.0017	0.0172/0.0003
Fogo	Tradicional	3	1.36/1.07	0.0013/0.0021	0.0172/0.0003
Geral	Procedural	1	23.96/29.99	0.0038/0.0065	0.0153/0.0028
Geral	Procedural	2	137.5/105.2	0.0059/0.0015	0.018/0.0037
Geral	Procedural	3	262.54/145.48	0.0109/0.0016	0.0243/0.0069
Geral	Tradicional	1	18.86/6.48	0.0015/0.0018	0.0166/0.0003
Geral	Tradicional	2	109.06/20.41	0.0044/0.0014	0.014/0.0004
Geral	Tradicional	3	164.54/27.1	0.0062/0.0018	0.0124/0.0008
Tecido	Procedural	1	36.0/0.14	0.0031/0.002	0.0153/0.0027
Tecido	Procedural	2	3.65/7.3	0.0052/0.0013	0.0142/0.0043
Tecido	Procedural	3	5.5/10.98	0.0083/0.0012	0.022/0.0062
Tecido	Tradicional	1	1.36/1.07	0.0009/0.0018	0.0172/0.0003
Tecido	Tradicional	2	1.36/1.07	0.0008/0.0013	0.0172/0.0003
Tecido	Tradicional	3	1.36/1.07	0.0009/0.0016	0.0172/0.0003

Tabela 4 – Métricas da Máquina 2

Elemento	Versão	Nível de Estresse	Colisões (média/dp)	Physics Process (ms)(média/dp)	Process (ms)(média/dp)
Água	Procedural	1	2.26/2.42	0.0051/0.0006	0.0294/0.0073
Água	Procedural	2	3.43/2.28	0.0075/0.0021	0.031/0.0037
Água	Procedural	3	4.6/1.57	0.0104/0.0019	0.0281/0.006
Água	Tradicional	1	1.38/0.9	0.0035/0.0004	0.0164/0.0004
Água	Tradicional	2	1.39/0.69	0.0036/0.0024	0.0166/0.001
Água	Tradicional	3	1.0/0.0	0.0034/0.0019	0.0167/0.0012
Aranha	Procedural	1	0.69/1.01	0.0099/0.0013	0.0117/0.0078
Aranha	Procedural	2	0.69/1.01	0.0199/0.0024	0.021/0.0146
Aranha	Procedural	3	0.69/1.01	0.0301/0.0022	0.0298/0.0006
Aranha	Tradicional	1	53.94/14.43	0.0105/0.002	0.0111/0.0021
Aranha	Tradicional	2	108.83/21.72	0.018/0.0025	0.0056/0.0038
Aranha	Tradicional	3	165.81/23.96	0.0218/0.0023	0.0022/0.0027
Fogo	Procedural	1	0.69/1.01	0.0034/0.0042	0.0447/0.0025
Fogo	Procedural	2	0.69/1.01	0.0037/0.0051	0.0803/0.0034
Fogo	Procedural	3	0.69/1.01	0.0039/0.0059	0.113/0.0049
Fogo	Tradicional	1	1.36/1.07	0.0037/0.0027	0.0166/0.0015
Fogo	Tradicional	2	1.36/1.07	0.004/0.0038	0.0173/0.0025
Fogo	Tradicional	3	1.36/1.07	0.0047/0.0054	0.0178/0.0041
Geral	Procedural	1	20.2/27.06	0.0116/0.0155	0.0369/0.0109
Geral	Procedural	2	137.75/103.41	0.0166/0.0014	0.0466/0.002
Geral	Procedural	3	254.42/130.49	0.0316/0.0019	0.0634/0.0018
Geral	Tradicional	1	19.31/8.43	0.006/0.0036	0.0145/0.0019
Geral	Tradicional	2	112.62/23.27	0.0183/0.0031	0.0071/0.004
Geral	Tradicional	3	156.87/25.33	0.0229/0.004	0.0052/0.004
Tecido	Procedural	1	2.71/7.45	0.01/0.0035	0.0313/0.0085
Tecido	Procedural	2	5.66/9.63	0.0166/0.004	0.0524/0.0116
Tecido	Procedural	3	7.0/11.11	0.0245/0.0021	0.0796/0.0017
Tecido	Tradicional	1	1.36/1.07	0.0039/0.004	0.0165/0.002
Tecido	Tradicional	2	1.36/1.07	0.0039/0.0028	0.0164/0.0015
Tecido	Tradicional	3	1.36/1.07	0.0039/0.0037	0.0165/0.0021

Tabela 5 – Métricas da Máquina 3

Elemento	Versão	Nível de Estresse	Colisões (média/dp)	Physics Process (ms)(média/dp)	Process (ms)(média/dp)
Água	Procedural	1	2.26/2.42	0.0083/0.0118	0.0214/0.0116
Água	Procedural	2	3.43/2.28	0.0109/0.0063	0.0217/0.0091
Água	Procedural	3	4.65/1.58	0.0146/0.0071	0.0185/0.0172
Água	Tradicional	1	1.38/0.9	0.011/0.0231	0.0226/0.0248
Água	Tradicional	2	1.39/0.69	0.0094/0.0137	0.022/0.0128
Água	Tradicional	3	1.0/0.0	0.0117/0.0361	0.0208/0.0095
Aranha	Procedural	1	0.69/1.01	0.015/0.0065	0.0192/0.02
Aranha	Procedural	2	0.69/1.01	0.0261/0.0086	0.0396/0.0063
Aranha	Procedural	3	0.69/1.01	0.038/0.0041	0.0576/0.004
Aranha	Tradicional	1	55.25/14.69	0.0173/0.0144	0.0103/0.0188
Aranha	Tradicional	2	104.14/20.67	0.0306/0.0161	0.0036/0.0021
Aranha	Tradicional	3	159.01/24.95	0.0407/0.0196	0.0035/0.0026
Fogo	Procedural	1	0.69/1.01	0.0044/0.0061	0.0633/0.0022
Fogo	Procedural	2	0.69/1.01	0.0057/0.009	0.122/0.0212
Fogo	Procedural	3	0.69/1.01	0.0078/0.0132	0.203/0.0713
Fogo	Tradicional	1	1.36/1.07	0.0075/0.0076	0.0208/0.0069
Fogo	Tradicional	2	1.36/1.07	0.0122/0.0395	0.0255/0.0094
Fogo	Tradicional	3	1.36/1.07	0.0133/0.0347	0.031/0.0158
Geral	Procedural	1	23.06/25.81	0.0173/0.0234	0.0547/0.0141
Geral	Procedural	2	130.67/87.75	0.0247/0.0122	0.0887/0.0219
Geral	Procedural	3	228.29/136.35	0.6286/3.6367	0.2343/0.4801
Geral	Tradicional	1	17.92/7.49	0.0206/0.0592	0.0183/0.011
Geral	Tradicional	2	111.97/19.41	0.0292/0.0168	0.0074/0.0071
Geral	Tradicional	3	170.47/27.45	0.0383/0.0164	0.0091/0.0045
Tecido	Procedural	1	3.48/7.49	0.0141/0.0055	0.0499/0.0122
Tecido	Procedural	2	5.66/9.63	0.0229/0.0069	0.0942/0.0105
Tecido	Procedural	3	7.05/11.1	0.0366/0.009	0.1549/0.0154
Tecido	Tradicional	1	1.36/1.07	0.0122/0.0221	0.0246/0.021
Tecido	Tradicional	2	1.36/1.07	0.0136/0.0396	0.0245/0.0316
Tecido	Tradicional	3	1.36/1.07	0.0076/0.009	0.0235/0.0228

Tabela 6 – Métricas do dispositivo Mobile 1

Elemento	Versão	Nível de Estresse	Colisões (média/dp)	Physics Process (ms)(média/dp)	Process (ms)(média/dp)
Água	Procedural	1	2.26/2.42	0.008/0.001	0.018/0.0029
Água	Procedural	2	3.43/2.28	0.0104/0.0051	0.0167/0.0049
Água	Procedural	3	4.65/1.65	0.0133/0.0056	0.0163/0.0102
Água	Tradicional	1	1.38/0.9	0.0058/0.0054	0.0185/0.0017
Água	Tradicional	2	1.39/0.69	0.0063/0.0068	0.0185/0.0013
Água	Tradicional	3	1.0/0.0	0.0062/0.0073	0.0182/0.0012
Aranha	Procedural	1	0.69/1.01	0.01/0.0012	0.014/0.0124
Aranha	Procedural	2	0.69/1.01	0.0175/0.0016	0.0257/0.0012
Aranha	Procedural	3	0.69/1.01	0.0264/0.0019	0.0336/0.0008
Aranha	Tradicional	1	53.91/13.13	0.0112/0.001	0.0143/0.0017
Aranha	Tradicional	2	115.17/22.44	0.0157/0.0097	0.0092/0.0026
Aranha	Tradicional	3	165.92/26.13	0.0167/0.0047	0.005/0.0039
Fogo	Procedural	1	0.69/1.01	0.003/0.0082	0.0307/0.0024
Fogo	Procedural	2	0.69/1.01	0.0025/0.0065	0.0542/0.0024
Fogo	Procedural	3	0.69/1.01	0.0026/0.0072	0.0763/0.0041
Fogo	Tradicional	1	1.36/1.07	0.0055/0.007	0.0184/0.0015
Fogo	Tradicional	2	1.36/1.07	0.0066/0.0084	0.0186/0.0028
Fogo	Tradicional	3	1.36/1.07	0.0098/0.0396	0.0185/0.0038
Geral	Procedural	1	18.81/23.18	0.0094/0.0193	0.029/0.0143
Geral	Procedural	2	134.58/85.68	0.0138/0.0063	0.0464/0.0188
Geral	Procedural	3	248.03/137.78	0.025/0.0017	0.0587/0.002
Geral	Tradicional	1	19.34/8.8	0.0089/0.0062	0.0169/0.002
Geral	Tradicional	2	111.24/18.96	0.0146/0.0068	0.0114/0.004
Geral	Tradicional	3	167.11/24.02	0.0165/0.0065	0.0076/0.0061
Tecido	Procedural	1	2.6/7.38	0.0072/0.0063	0.0296/0.01
Tecido	Procedural	2	5.96/10.58	0.0129/0.0055	0.0555/0.0172
Tecido	Procedural	3	7.05/11.1	0.0211/0.005	0.0844/0.0282
Tecido	Tradicional	1	1.36/1.07	0.0059/0.0091	0.0187/0.0019
Tecido	Tradicional	2	1.36/1.07	0.0055/0.0063	0.0187/0.0015
Tecido	Tradicional	3	1.36/1.07	0.0053/0.0056	0.0188/0.0018

Tabela 7 – Métricas do dispositivo Mobile 2

Elemento	Versão	Nível de Estresse	Colisões (média/dp)	Physics Process (ms)(média/dp)	Process (ms)(média/dp)
Água	Procedural	1	2.26/2.42	0.0101/0.0009	0.0179/0.0059
Água	Procedural	2	3.43/2.28	0.0152/0.0065	0.017/0.0158
Água	Procedural	3	5.08/1.97	0.0193/0.0012	0.0063/0.0012
Água	Tradicional	1	1.38/0.9	0.0047/0.0071	0.0173/0.005
Água	Tradicional	2	1.39/0.69	0.0047/0.0079	0.0174/0.0041
Água	Tradicional	3	1.0/0.0	0.0044/0.0081	0.0174/0.0055
Aranha	Procedural	1	0.69/1.01	0.0217/0.0022	0.0369/0.0019
Aranha	Procedural	2	0.69/1.01	0.0425/0.0019	0.0794/0.0056
Aranha	Procedural	3	0.69/1.01	0.0664/0.0037	0.1134/0.0207
Aranha	Tradicional	1	53.53/15.01	0.0162/0.0013	0.0069/0.0052
Aranha	Tradicional	2	107.07/17.49	0.0277/0.0017	0.0052/0.0004
Aranha	Tradicional	3	165.16/22.97	0.04/0.0022	0.0058/0.0004
Fogo	Procedural	1	0.69/1.01	0.0036/0.0123	0.0847/0.0044
Fogo	Procedural	2	0.69/1.01	0.0034/0.0099	0.1591/0.0059
Fogo	Procedural	3	0.69/1.01	0.0036/0.0128	0.2367/0.0098
Fogo	Tradicional	1	1.36/1.07	0.0061/0.0102	0.0199/0.0055
Fogo	Tradicional	2	1.36/1.07	0.0063/0.0153	0.0292/0.0073
Fogo	Tradicional	3	1.36/1.07	0.0074/0.0175	0.0396/0.0086
Geral	Procedural	1	21.14/29.02	0.0157/0.0011	0.0635/0.0023
Geral	Procedural	2	129.73/86.72	0.0305/0.0015	0.1078/0.0035
Geral	Procedural	3	242.98/127.93	0.0601/0.0029	0.1546/0.0063
Geral	Tradicional	1	18.52/7.43	0.0108/0.0143	0.0159/0.0063
Geral	Tradicional	2	115.66/19.29	0.0282/0.0095	0.0133/0.0087
Geral	Tradicional	3	166.93/23.06	0.0404/0.0029	0.0214/0.0016
Tecido	Procedural	1	3.76/7.47	0.0184/0.009	0.0478/0.0196
Tecido	Procedural	2	5.66/9.63	0.034/0.0011	0.0872/0.0015
Tecido	Procedural	3	7.05/11.1	0.0568/0.0036	0.1429/0.0031
Tecido	Tradicional	1	1.36/1.07	0.005/0.0101	0.0184/0.0066
Tecido	Tradicional	2	1.36/1.07	0.0047/0.0083	0.0191/0.0055
Tecido	Tradicional	3	1.36/1.07	0.005/0.0137	0.0214/0.0066

Tabela 8 – Métricas do dispositivo Mobile 3

Elemento	Versão	Nível de Estresse	Colisões (média/dp)	Physics Process (ms)(média/dp)	Process (ms)(média/dp)
Água	Procedural	1	2.26/2.42	0.0144/0.0006	0.0088/0.0016
Água	Procedural	2	3.43/2.28	0.0236/0.0011	0.0219/0.0007
Água	Procedural	3	5.05/1.95	0.0399/0.0007	0.0101/0.0003
Água	Tradicional	1	1.38/0.9	0.0057/0.0004	0.0162/0.0014
Água	Tradicional	2	1.39/0.69	0.0057/0.0007	0.0162/0.0008
Água	Tradicional	3	1.0/0.0	0.0052/0.0003	0.0164/0.0011
Aranha	Procedural	1	0.69/1.01	0.0307/0.0012	0.0482/0.0011
Aranha	Procedural	2	0.69/1.01	0.0607/0.0024	0.096/0.0089
Aranha	Procedural	3	0.69/1.01	0.0953/0.0045	0.1399/0.0281
Aranha	Tradicional	1	57.41/16.02	0.0418/0.0041	0.0094/0.0006
Aranha	Tradicional	2	116.32/20.58	0.0788/0.0072	0.0102/0.0008
Aranha	Tradicional	3	164.92/23.17	0.1138/0.0066	0.011/0.0008
Fogo	Procedural	1	0.69/1.01	0.0053/0.0013	0.1213/0.009
Fogo	Procedural	2	0.69/1.01	0.0033/0.001	0.2165/0.0075
Fogo	Procedural	3	0.69/1.01	0.0047/0.0193	0.317/0.0127
Fogo	Tradicional	1	1.36/1.07	0.0071/0.0003	0.0181/0.0012
Fogo	Tradicional	2	1.36/1.07	0.0118/0.0336	0.0411/0.0331
Fogo	Tradicional	3	1.36/1.07	0.0103/0.0006	0.0509/0.0052
Geral	Tradicional	1	19.93/7.8	0.0213/0.0271	0.0188/0.0292
Geral	Tradicional	2	111.19/18.77	0.0767/0.0042	0.0242/0.0028
Geral	Tradicional	3	168.79/26.13	0.1196/0.0095	0.0307/0.005
Tecido	Procedural	1	3.76/7.47	0.0345/0.0025	0.3212/0.0154
Tecido	Tradicional	1	1.36/1.07	0.0059/0.0006	0.0165/0.0011
Tecido	Tradicional	2	1.36/1.07	0.0058/0.0004	0.0181/0.0006
Tecido	Tradicional	3	1.36/1.07	0.009/0.0291	0.0251/0.0325

APÊNDICE B – COMPARAÇÃO DE COLUNAS PELO TESTE DE *KENDALL*

Tabela 9 – Testes no qual existe correlação significativa (p < 0,05) no teste de *Kendall* entre FPS e pares de colisões

Maquina	Elemento	Versao	Nível de Estresse	Média FPS	Tau de Kendall	P-value
MAQUINA3	Geral	Procedural	1	8.82	-0.154899	0.044829
MAQUINA3	Aranha	Tradicional	1	58.14	-0.165783	0.038681
MOBILE2	Fogo	Tradicional	3	54.23	-0.179893	0.031742
MAQUINA1	Fogo	Tradicional	3	59.79	-0.200297	0.036296
MAQUINA1	Tecido	Tradicional	1	59.79	-0.200297	0.036296
MAQUINA2	Geral	Procedural	1	17.34	-0.208352	0.007111
MOBILE2	Geral	Procedural	1	5.89	-0.239872	0.003953
MAQUINA2	Geral	Procedural	2	6.76	-0.24743	0.002397
MAQUINA2	Água	Procedural	2	30.32	-0.27409	0.001059
MOBILE1	Geral	Procedural	2	8.24	-0.285598	0.000248
MAQUINA1	Água	Tradicional	2	59.79	-0.304083	0.001565
MAQUINA1	Tecido	Procedural	1	59.94	-0.327767	0.000994
MAQUINA1	Geral	Procedural	2	43.4	-0.344294	4,00E-06
MOBILE2	Água	Procedural	1	52.39	-0.367645	4,00E-06
MAQUINA3	Água	Procedural	1	46.38	-0.390593	2,00E-06
MOBILE3	Água	Procedural	1	41.22	-0.394134	0.0
MOBILE1	Água	Procedural	1	52.59	-0.428727	0.0
MAQUINA2	Água	Procedural	1	37.78	-0.458316	0.0
MAQUINA1	Geral	Procedural	3	19.0	-0.568862	0.0
MAQUINA2	Aranha	Tradicional	3	10.84	0.142302	0.048405
MOBILE2	Tecido	Tradicional	3	59.78	0.20218	0.033014
MOBILE1	Geral	Tradicional	3	38.01	0.216932	0.001892
MAQUINA3	Tecido	Procedural	1	9.02	0.226894	0.00896