Guia Arquitetônico e de Implementação para um Designer Interativo de Estruturas Hexagonais

Introdução

Sumário Executivo

Este documento apresenta um guia arquitetônico e um plano de implementação detalhado para a criação de uma aplicação web avançada, destinada ao design e visualização de estruturas baseadas em ladrilhos (tiles) hexagonais. O objetivo do projeto é desenvolver uma ferramenta interativa que permita aos usuários não apenas gerar e configurar grades hexagonais, mas também interagir com elas de maneira complexa. As funcionalidades centrais incluem a capacidade de clicar em um ladrilho para visualizar uma representação de um cubo isométrico e para ativar uma interface de seleção de direções, a partir da qual uma conexão visual pode ser estendida até a borda da grade. A aplicação visa fornecer um alto grau de controle visual e lógico, fundamentado em uma arquitetura de software robusta, performática e extensível.

Contexto e Abordagem

O desenvolvimento desta aplicação foi solicitado com base em um código de referência existente. No entanto, a análise inicial determinou que os repositórios de código fornecidos estavam inacessíveis.[1] Esta circunstância, longe de ser um impedimento, apresenta uma oportunidade única para projetar uma solução superior, construída a partir de primeiros princípios e fundamentada em uma análise rigorosa das melhores práticas da indústria e das tecnologias mais adequadas para cada

desafio específico. A abordagem adotada neste relatório, portanto, não busca replicar uma implementação desconhecida, mas sim construir um sistema otimizado, moderno e bem-documentado. A solução proposta é o resultado de uma pesquisa exaustiva sobre a matemática das grades hexagonais, bibliotecas de lógica e renderização, e padrões de arquitetura de software para aplicações gráficas interativas.

Estrutura do Documento

A estrutura deste relatório foi cuidadosamente organizada para guiar o leitor, presumivelmente um desenvolvedor técnico ou líder de projeto, através de uma jornada lógica e progressiva. O documento começa estabelecendo os fundamentos teóricos e matemáticos indispensáveis para qualquer trabalho com grades hexagonais. Em seguida, avança para a implementação prática, detalhando a renderização no canvas, a gestão da interatividade do usuário e o desenvolvimento das funcionalidades centrais, como o desenho do cubo isométrico e das linhas de conexão. Cada seção não apenas descreve "o que" fazer, mas também "por que" uma determinada abordagem é superior, contextualizando as decisões de engenharia com base em performance, manutenibilidade e escalabilidade. A seção final consolida todos os componentes em uma arquitetura de aplicação coesa e oferece recomendações para garantir a longevidade e o sucesso do projeto.

Seção 1: Fundamentos da Grade Hexagonal: Teoria e Estrutura de Dados

A construção de uma aplicação robusta e eficiente sobre uma grade hexagonal depende criticamente da solidez de seus fundamentos. As decisões tomadas nesta fase inicial sobre a representação matemática e a estrutura de dados da grade terão um impacto profundo na simplicidade, performance e poder de todos os algoritmos subsequentes. Esta seção estabelece essa base teórica, garantindo que o sistema seja construído sobre alicerces sólidos.

1.1 A Matemática Essencial por Trás dos Hexágonos

Antes de qualquer implementação, é imperativo compreender a geometria fundamental dos hexágonos regulares, que são polígonos de seis lados com todos os lados e ângulos internos iguais.

Orientação e Dimensões
Uma grade hexagonal pode ter duas orientações principais: pointy-top (com os topos pontiagudos) e flat-top (com os topos achatados).3 A escolha da orientação afeta os cálculos de posicionamento e as relações de vizinhança. O parâmetro fundamental de um hexágono é o seu
size, que representa a distância do centro a um de seus vértices. A partir do size, podemos derivar a largura (width) e a altura (height) do hexágono:

- **Orientação Pointy-Top:**
  - height=2×size
  - width=3×size
- **Orientação Flat-Top:**
  - height=3×size
  - width=2×size

O espaçamento entre os centros de hexágonos adjacentes também depende da orientação e é crucial para a renderização da grade.[3]

Cálculo dos Vértices
Para desenhar um hexágono no canvas, é necessário calcular as coordenadas de seus seis vértices. Dado um ponto central center e um size, os vértices podem ser encontrados usando trigonometria. Cada vértice está a um ângulo de 60 graus do próximo. A fórmula para encontrar o i-ésimo vértice (para i de 0 a 5) é 3:
vertexi.x=center.x+size×cos(anglerad)
vertexi.y=center.y+size×sin(anglerad)
Onde o ângulo em radianos, anglerad, depende da orientação. Para a orientação *pointy-top*, os ângulos começam em 30 graus (ou −6π radianos), enquanto para a *flat-top*, começam em 0 graus.[4]

1.2 Sistemas de Coordenadas: Uma Análise Comparativa Crítica

A escolha do sistema de coordenadas para representar a posição de cada hexágono

é, talvez, a decisão arquitetônica mais importante do projeto. Ela influencia diretamente a complexidade de algoritmos essenciais como cálculo de distância, busca de vizinhos e desenho de linhas. Existem três sistemas principais [3]:

- **Coordenadas de Deslocamento (Offset):** Este é o sistema mais intuitivo para quem vem de grades quadradas. Ele utiliza um par de coordenadas (col, row) e desloca a cada duas colunas ou linhas para criar o padrão hexagonal. Embora fácil de entender, ele torna a matemática de adjacência e distância desnecessariamente complexa, exigindo lógica condicional para lidar com as linhas ou colunas deslocadas.[3]
- **Coordenadas Cúbicas (Cube):** Este sistema é o mais poderoso para a implementação de algoritmos. Ele projeta a grade hexagonal 2D em um plano dentro de um espaço cartesiano 3D, utilizando três eixos (q, r, s) a 120 graus um do outro.[4] A propriedade fundamental deste sistema é que a soma das três coordenadas de qualquer hexágono é sempre zero:
  $q+r+s=0$

  Esta restrição é a chave para a sua elegância. Ela permite que operações como adição de vetores, subtração e multiplicação por escalar funcionem de forma idêntica à de um sistema cartesiano padrão. Isso simplifica drasticamente algoritmos complexos. Por exemplo, encontrar um vizinho em uma determinada direção é simplesmente uma questão de adicionar um vetor de direção constante.8 O cálculo da distância entre dois hexágonos também se torna uma fórmula direta.8
- **Coordenadas Axiais (Axial):** Este sistema é uma otimização de armazenamento para as coordenadas cúbicas. Uma vez que a restrição $q+r+s=0$ sempre se mantém, a terceira coordenada, s, é redundante e pode ser calculada a qualquer momento a partir das outras duas: $s=−q−r$. O sistema axial armazena apenas as coordenadas q e r, economizando espaço sem perder o poder computacional do sistema cúbico.[3]

Conclusão Arquitetural:
A estratégia ótima, que equilibra poder algorítmico e eficiência de armazenamento, é adotar uma abordagem híbrida. O sistema deve utilizar internamente as Coordenadas Cúbicas para todos os cálculos e algoritmos (distância, vizinhança, desenho de linha) para aproveitar sua simplicidade matemática. Para o armazenamento de dados e para a API pública do modelo de dados, devem ser utilizadas as Coordenadas Axiais. Esta decisão arquitetônica preventiva mitiga a complexidade futura, evitando o código condicional e os casos especiais que seriam necessários com coordenadas de deslocamento, especialmente para uma funcionalidade como "prolongar uma barra", que é fundamentalmente um problema de desenho de linha.

1.3 Seleção da Biblioteca de Lógica: honeycomb-grid

Com a base teórica estabelecida, o próximo passo é selecionar uma biblioteca para gerenciar a lógica da grade. A escolha ideal é uma biblioteca que implemente corretamente a matemática discutida e que promova uma boa arquitetura de software. Após uma análise comparativa, a biblioteca honeycomb-grid emerge como a escolha superior.[10]

A principal vantagem da honeycomb-grid é ser uma biblioteca "headless" (ou agnóstica ao renderizador). Isso significa que ela lida exclusivamente com a lógica da grade — coordenadas, traversals, distâncias — sem impor qualquer tecnologia ou método de renderização.[10] Esta característica é o pilar para uma arquitetura limpa, baseada na separação de responsabilidades (Separation of Concerns), onde o "modelo" (a lógica da grade) é completamente desacoplado da "visão" (a sua representação visual no canvas). A biblioteca é fortemente inspirada e compatível com os algoritmos de alta qualidade do Red Blob Games, garantindo uma base matemática robusta e confiável.[11]

Tabela 1: Comparativo de Bibliotecas de Grade Hexagonal

Biblioteca	Foco Principal	Dependência de Renderizador	Adequação para o Projeto
honeycomb-grid	Lógica de grade "headless" (coordenadas, traversals)	Nenhuma. Totalmente agnóstica.	**Excelente.** Promove uma arquitetura limpa e desacoplada, ideal para os requisitos de renderização customizados (grade 2D + cubo 3D).
JointJS	Diagramação completa (nós, links, ferramentas)	Sim, fortemente acoplada à renderização SVG.	**Razoável.** Poderosa, mas potencialmente excessiva. O forte acoplamento com SVG dificultaria a integração do cubo

			isométrico e a renderização customizada no canvas. [12]
oi.hexmap.js	Cartogramas baseados em HexJSON	Sim, renderiza em SVG.	**Baixa.** Focada em exibir layouts estáticos a partir de dados HexJSON, não adequada para um designer dinâmico e interativo. [13]

Esta análise comparativa demonstra que a escolha da honeycomb-grid não é arbitrária, mas sim uma decisão de engenharia informada. A separação entre lógica e renderização que ela impõe é a estratégia arquitetônica mais eficaz para atender aos requisitos multifacetados do projeto, que envolvem renderização 2D, projeção isométrica 3D e elementos de UI HTML.

1.4 Modelagem de Dados do "Tile" e Suas Conexões

Um "tile" (ladrilho) na nossa aplicação é mais do que apenas uma posição na grade. É um objeto de dados rico que armazena seu estado, suas propriedades e suas relações com outros tiles. Uma modelagem de dados bem-sucedida aqui é crucial para a extensibilidade futura da aplicação.

A estrutura de dados para um único tile deve ser definida da seguinte forma:

- **Propriedades de Identificação e Lógica:**
  - id: Um identificador único e consistente, que pode ser gerado a partir de suas coordenadas (ex: "q0r1").
  - coords: Um objeto contendo as coordenadas axiais { q: number, r: number } para armazenamento.
  - hexObject: Uma referência à instância do objeto Hex da biblioteca honeycomb-grid, usada para cálculos lógicos.
- **Propriedades de Estado da Interface:**
  - isSelected: Um booleano que indica se o tile é o principal selecionado pelo usuário (onde o cubo e o menu aparecem).
  - isHovered: Um booleano que indica se o cursor do mouse está sobre o tile,

para fins de realce.
    - isActive: Um booleano genérico para outros estados, como fazer parte de um caminho desenhado.
      (Inspirado pelos conceitos de estado em 14).
- Modelo de Conectividade:
  Para atender ao requisito de definir "a quantidade de conexões em cada tile", a estrutura de dados deve incluir um campo para armazenar essas conexões. Um modelo flexível e extensível é um array de objetos, onde cada objeto representa uma conexão:
    - connections: Um array, por exemplo: ``.
      Este modelo permite não apenas registrar a existência de uma conexão, mas também seu tipo e propriedades adicionais, abrindo caminho para futuras funcionalidades como caminhos com custos, diferentes tipos de terreno, etc. A base para este modelo relacional vem da análise das relações entre tiles, arestas e vértices.9

A combinação de um sistema de coordenadas poderoso, uma biblioteca de lógica desacoplada e um modelo de dados rico e extensível forma a base sobre a qual todas as funcionalidades subsequentes serão construídas com clareza e eficiência.

Seção 2: Renderização e Visualização no Canvas HTML5

Com o modelo de dados e a lógica da grade definidos, o próximo passo é traduzir essa estrutura abstrata em uma representação visual interativa. Esta seção detalha o processo de renderização da grade hexagonal no elemento <canvas> do HTML5, focando em performance, clareza visual e otimização.

2.1 Configuração do Ambiente de Renderização: Canvas 2D

A escolha da tecnologia de renderização é fundamental. Para este projeto, a **API Canvas 2D** é a opção mais adequada. Ela oferece o equilíbrio ideal entre a performance necessária para desenhar um grande número de formas geométricas e o controle de baixo nível sobre os pixels, que é essencial para implementar uma

detecção de cliques precisa e eficiente.[15]

Em comparação, o SVG (Scalable Vector Graphics), embora seja excelente para manipulação de objetos via DOM e ofereça escalabilidade perfeita, pode apresentar gargalos de performance quando o número de elementos na cena cresce para a casa dos milhares, o que é uma possibilidade em uma grade grande.[17] Além disso, a integração de renderizações mais complexas, como o cubo isométrico, e a manipulação de pixels para detecção de cliques são nativamente mais simples e performáticas no contexto do Canvas 2D. A API do Canvas nos dá um controle direto sobre o que é desenhado, quadro a quadro, o que se alinha perfeitamente com a arquitetura de renderização orientada a estado que será adotada.

2.2 Desenhando um Hexágono Individual

A base da renderização da grade é a capacidade de desenhar um único hexágono de forma eficiente. Para isso, é recomendável criar uma função reutilizável que encapsule essa lógica.

JavaScript

```
function drawHexagon(ctx, centerPoint, size, orientation) {
    //...
}
```

Esta função deve realizar os seguintes passos:

1. Calcular as coordenadas dos seis vértices do hexágono com base no centerPoint, size e orientation, utilizando as fórmulas trigonométricas discutidas na Seção 1.1.
2. Utilizar os comandos da API do Canvas para desenhar o polígono. O processo inicia com ctx.beginPath() para começar um novo caminho.
3. Um loop de 0 a 5 é usado para chamar ctx.lineTo(vertex.x, vertex.y) para cada um dos seis vértices, desenhando as arestas do hexágono.
4. ctx.closePath() é chamado para conectar o último vértice ao primeiro, fechando a forma.
5. Finalmente, ctx.stroke() desenha o contorno e ctx.fill() preenche o interior do

hexágono com as cores e estilos definidos previamente em ctx.strokeStyle e ctx.fillStyle.[5]

Para otimização, especialmente se todos os hexágonos tiverem o mesmo tamanho, a forma do hexágono pode ser definida uma única vez usando um objeto Path2D. Este objeto pode então ser reutilizado em cada chamada de desenho, evitando o recálculo dos caminhos e melhorando a performance.[19]

2.3 Renderização da Grade Completa e Otimização

A renderização da grade completa envolve iterar sobre a estrutura de dados da grade e desenhar cada hexágono individualmente. A estratégia de renderização deve seguir um fluxo de trabalho claro:

1. **Iteração sobre a Lógica da Grade:** O código de renderização irá iterar sobre o objeto Grid fornecido pela biblioteca honeycomb-grid.
2. **Conversão de Coordenadas:** Para cada hex na iteração, o método hex.toPoint() será invocado. Este método é a ponte crucial entre o mundo lógico da grade (coordenadas axiais/cúbicas) e o mundo visual do canvas (coordenadas de pixel). Ele abstrai toda a matemática de projeção, retornando um ponto (x, y) que representa o centro do hexágono no canvas.[17] Este desacoplamento é o que torna a arquitetura de separação de responsabilidades viável e elegante. O código de renderização não precisa ter conhecimento sobre a complexidade dos sistemas de coordenadas hexagonais; ele apenas solicita as coordenadas de pixel.
3. **Desenho Individual:** Com o ponto central em pixels, a função drawHexagon (descrita em 2.2) é chamada para renderizar o hexágono na tela.

O Loop de Animação (requestAnimationFrame)
As funcionalidades interativas do projeto, como o realce ao passar o mouse, exigem que a tela seja atualizada em tempo real. Uma abordagem ingênua seria redesenhar a grade a cada evento de mouse, o que é ineficiente e pode levar a problemas de performance e "tearing" visual.
A solução profissional é implementar um loop de renderização contínuo e otimizado usando window.requestAnimationFrame. Este método informa ao navegador que você deseja executar uma animação e solicita que o navegador chame uma função de atualização específica antes do próximo redesenho. Isso garante que a renderização seja sincronizada com o ciclo de atualização do navegador, resultando em animações

mais suaves e eficientes em termos de bateria.[14]

A arquitetura resultante é **orientada a estado**. As interações do usuário (como mover o mouse) não acionam diretamente o redesenho. Em vez disso, elas atualizam um objeto de estado central (por exemplo, state.hoveredHex). O loop de renderização, que é executado de forma independente, lê este objeto de estado em cada quadro e desenha a cena inteira de acordo com os valores atuais. Este padrão (Interação → Atualização de Estado → Renderização) é fundamental para aplicações gráficas modernas, pois separa as preocupações, melhora a performance e torna o comportamento da aplicação previsível e depurável.

2.4 Estilização Visual e Feedback ao Usuário

O feedback visual imediato é essencial para uma boa experiência do usuário. A API do Canvas oferece controle total sobre a aparência dos hexágonos.

- **Estilos Básicos:** As propriedades ctx.fillStyle, ctx.strokeStyle, e ctx.lineWidth podem ser usadas para definir a cor de preenchimento, a cor do contorno e a espessura da linha de cada hexágono, respectivamente.
- **Realce de Interação (Highlighting):** Dentro do loop de renderização, antes de desenhar cada hexágono, o código deve verificar o estado da aplicação. Por exemplo:

  ```JavaScript
  grid.forEach(hex => {
    const center = hex.toPoint();
    if (hex.equals(state.activeHex)) {
        ctx.fillStyle = 'blue'; // Cor para o hexágono ativo
    } else if (hex.equals(state.hoveredHex)) {
        ctx.fillStyle = 'lightblue'; // Cor para o hexágono sob o mouse
    } else {
        ctx.fillStyle = 'gray'; // Cor padrão
    }
    drawHexagon(ctx, center,...);
  });
  ```

  Este mecanismo garante que a representação visual seja sempre um reflexo fiel do estado atual da aplicação, fornecendo feedback instantâneo ao usuário sobre

qual hexágono está selecionado ou sob o cursor.[13]

Seção 3: Implementando a Interatividade do Usuário

A interatividade é o coração desta aplicação. É através das ações do usuário, como cliques e movimentos do mouse, que o designer ganha vida. Esta seção detalha a implementação da ponte entre as ações do usuário e o sistema lógico, focando em algoritmos de detecção precisos, gerenciamento de estado e a criação de uma interface de usuário intuitiva sobre o canvas.

3.1 Detecção de Cliques: O Algoritmo de Conversão de Pixel para Hexágono

A funcionalidade mais fundamental é identificar em qual hexágono o usuário clicou. A abordagem para resolver este problema de "pixel para hexágono" define a performance e a precisão da aplicação.

1. Captura e Normalização do Evento: O primeiro passo é adicionar um listener de eventos ao elemento canvas: canvas.addEventListener('click',...). Dentro do manipulador de eventos, é crucial obter as coordenadas do clique relativas ao próprio canvas, e não à janela do navegador. Isso é feito subtraindo a posição do canvas da posição do clique do cliente:
   const mouseX = event.clientX - canvas.getBoundingClientRect().left;
   const mouseY = event.clientY - canvas.getBoundingClientRect().top;
   A falha em normalizar as coordenadas resultará em detecções incorretas se o canvas não estiver posicionado no canto superior esquerdo da página.14
2. **O Algoritmo de Conversão Matemática (O(1)):** A abordagem mais robusta, performática e escalável para a conversão de pixel para hexágono é a matemática direta, que opera em tempo constante (O(1)), independentemente do número de hexágonos na grade. O processo, popularizado pelo Red Blob Games, envolve dois passos principais:
   - **Conversão para Coordenadas Fracionárias:** As coordenadas de pixel (x, y) são convertidas em coordenadas axiais ou cúbicas fracionárias (q_f, r_f, s_f) aplicando uma transformação de matriz inversa. A matriz exata depende da orientação e do tamanho dos hexágonos.[4]

    - **Arredondamento para Coordenadas Inteiras:** As coordenadas fracionárias devem ser arredondadas para os inteiros mais próximos para encontrar o hexágono correto. No entanto, um simples arredondamento de cada componente pode violar a restrição cúbica (q+r+s=0). O algoritmo hex_round resolve isso de forma elegante: ele arredonda q, r e s para os inteiros mais próximos (rq, rr, rs). Em seguida, calcula as diferenças q_diff = abs(rq - q_f), r_diff = abs(rr - r_f), s_diff = abs(rs - s_f). Se a soma dos arredondados não for zero, o componente com a maior diferença é ajustado para satisfazer a restrição.[3]

A biblioteca honeycomb-grid já implementa esta lógica de forma otimizada através do método Grid.pointToHex(x, y), abstraindo essa complexidade do desenvolvedor.[20]

Análise de Alternativas Inferiores:
É importante entender por que outras abordagens comuns são menos adequadas:

- **Distância ao Centro Mais Próximo:** Calcular a distância do ponto de clique ao centro de cada hexágono e escolher o mais próximo é uma abordagem O(N). Além disso, ela é imprecisa nas fronteiras equidistantes entre dois ou três hexágonos.[23]
- **Mapa de Cores (Color Picking):** Desenhar cada hexágono com uma cor única em um canvas oculto e, no clique, ler a cor do pixel correspondente para identificar o hexágono. Este método é lento, consome uma quantidade significativa de memória e é inflexível a mudanças de estilo.[24]
- **ctx.isPointInPath:** Esta função nativa do canvas pode verificar se um ponto está dentro do caminho atual. No entanto, para usá-la, seria necessário iterar sobre todos os hexágonos da grade, recriar o caminho de cada um e testar o ponto, resultando em uma complexidade O(N) que não escala bem.[19]

A combinação de um cálculo matemático direto (O(1)) com um padrão de gerenciamento de estado desacoplado é a marca de um sistema gráfico robusto e performático. O clique do usuário não deve acionar a lógica de renderização diretamente. Em vez disso, o resultado do cálculo (pixelToHex) deve ser usado para atualizar um objeto de estado central. O loop de renderização independente, por sua vez, reage a essa mudança de estado, garantindo um fluxo de dados unidirecional e previsível.

3.2 Gerenciamento Centralizado do Estado da Interface

Para evitar inconsistências e facilitar a depuração, a aplicação deve ter uma **única fonte da verdade** (Single Source of Truth) para seu estado. Um objeto de estado centralizado gerencia todas as informações dinâmicas da interface.

JavaScript

```
const state = {
    grid: { /* instância do grid da honeycomb */ },
    hoveredHex: null, // ID do hexágono sob o mouse
    activeHex: null,  // ID do hexágono clicado
    contextMenu: {
        visible: false,
        x: 0,
        y: 0,
        options: [/* 'northeast', 'east',... */]
    },
    activeLine: {
        startHex: null,
        path:
    }
};
```

Os manipuladores de eventos (event handlers) não devem manipular o DOM ou o canvas diretamente. Sua única responsabilidade é despachar "ações" que modificam este objeto de estado. Por exemplo, um clique do mouse atualiza state.activeHex. O loop de renderização (Seção 2.3) então lê este estado e atualiza a tela para refletir a mudança.

3.3 Implementando a "Área de Seleção de Direções" como um Menu de Contexto Customizado

A "área de seleção de direções" que aparece ao clicar em um ladrilho é melhor

implementada como um menu de contexto customizado, que se sobrepõe ao canvas. Isso requer a coordenação entre o mundo do Canvas e o mundo do DOM.

Padrão de Implementação:
O processo para criar um menu de contexto customizado é bem estabelecido 26:

1. **Estrutura HTML:** Crie um elemento no arquivo HTML para servir como o contêiner do menu. Estilize-o com CSS para que tenha position: absolute (ou fixed) e display: none por padrão.
2. **Captura do Evento:** Adicione um listener para o evento contextmenu no elemento canvas. Este evento é disparado com um clique do botão direito. Para simular o comportamento com um clique esquerdo, o evento click pode ser usado, e a lógica do menu de contexto ativada a partir dele.
3. **Prevenção do Comportamento Padrão:** Dentro do handler do contextmenu, a primeira chamada deve ser event.preventDefault(). Isso impede que o menu de contexto nativo do navegador seja exibido.
4. **Exibição e Posicionamento:** Quando o evento é disparado sobre um hexágono válido:
    - Popule dinamicamente o do menu com as opções de direção (que podem ser elementos <button> ou <a>).
    - Use as coordenadas do evento (event.clientX e event.clientY) para definir as propriedades top e left do estilo do menu, posicionando-o próximo ao cursor.
    - Altere o display do menu para block ou flex para torná-lo visível.
5. **Ocultação do Menu:** Adicione um listener de clique global ao objeto window. Se o menu estiver visível, este listener o ocultará (display: none). É importante chamar event.stopPropagation() nos cliques dentro do próprio menu para evitar que o evento se propague para o window e feche o menu imediatamente.[28]

A necessidade de sobrepor uma UI do DOM (o menu) sobre o canvas introduz um desafio de coordenação entre dois espaços de coordenadas distintos. O menu é posicionado usando coordenadas da viewport (DOM), enquanto a grade e o cubo são desenhados usando coordenadas internas do canvas. A tradução correta entre esses dois mundos, usando canvas.getBoundingClientRect(), é uma complexidade oculta, mas fundamental, para garantir que a UI apareça alinhada corretamente com os elementos no canvas.

Seção 4: Desenvolvimento das Funcionalidades Centrais

Com a infraestrutura de grade, renderização e interatividade estabelecida, esta seção foca na implementação das funcionalidades de alto valor que definem a aplicação: o desenho do cubo isométrico e a extensão das barras de conexão.

4.1 Desenhando o Cubo Isométrico: Integração com a Biblioteca Isomer.js

A tarefa de desenhar um cubo com perspectiva isométrica em um canvas 2D envolve matemática de projeção 3D que pode ser complexa de implementar do zero.[29] Para abstrair essa complexidade e acelerar o desenvolvimento, a decisão estratégica é utilizar uma biblioteca especializada. A

Isomer.js é uma escolha excelente para este fim, devido à sua API simples e foco específico em desenhar formas prismáticas como cubos em um ambiente isométrico.[32]

Desafio de Implementação e Solução Arquitetural: O Canvas Off-Screen
Um desafio significativo surge da coexistência da renderização 2D da grade e da renderização 3D isométrica do cubo. A Isomer.js, para criar a ilusão 3D, aplica transformações de matriz (rotação, escala, inclinação) diretamente no contexto de renderização do canvas.33 Aplicar essas transformações no contexto do canvas principal distorceria toda a grade hexagonal, o que é inaceitável.
A solução arquiteturalmente correta e mais robusta para este conflito de contextos de renderização é o uso de um **canvas off-screen** (um buffer de renderização fora da tela). Este padrão desacopla completamente os dois mundos de renderização.

Fluxo de Trabalho Detalhado para Desenhar o Cubo:

1. Criação do Canvas Off-Screen: Após o clique do usuário em um tile, um novo elemento <canvas> é criado em memória, sem ser anexado ao DOM. Este será o nosso canvas off-screen.
   const offscreenCanvas = document.createElement('canvas');
2. Instanciação do Isomer: Uma nova instância do Isomer é criada, tendo como alvo o canvas off-screen.
   const iso = new Isomer(offscreenCanvas);
3. Desenho do Cubo: A API simples do Isomer é usada para desenhar o cubo no canvas off-screen. As cores e dimensões podem ser configuradas conforme

necessário.
iso.add(Shape.Prism(Point.ORIGIN, width, length, height), color); 32

4. Renderização no Canvas Principal: No loop de renderização principal da aplicação (controlado por requestAnimationFrame), quando o estado indicar que um tile está ativo, o conteúdo do canvas off-screen é "carimbado" no canvas principal usando o método ctx.drawImage().
mainCtx.drawImage(offscreenCanvas, targetX, targetY);
As coordenadas targetX e targetY devem ser calculadas para posicionar o cubo corretamente sobre o tile ativo.

Este padrão de buffer off-screen é uma técnica profissional que resolve um problema complexo de forma elegante, permitindo que cada contexto de renderização (o 2D da grade e o isométrico do cubo) opere em seu próprio ambiente sem interferência.

Tabela 2: API Essencial da Isomer.js para Desenho de Cubos

Classe/Método	Descrição	Exemplo de Uso
new Isomer(canvas)	Construtor principal. Cria uma instância de cena isométrica ligada a um elemento canvas.	var iso = new Isomer(myCanvas);
Isomer.Point	Representa um ponto no espaço 3D (x, y, z). Point.ORIGIN é um atalho para (0,0,0).	var p = new Isomer.Point(1, 2, 0);
Isomer.Shape.Prism	Cria uma forma de prisma retangular (um cubo, se as dimensões forem iguais).	var cube = Isomer.Shape.Prism(p, w, d, h);
iso.add(shape, color)	Adiciona uma forma (e opcionalmente uma cor) à cena para ser renderizada.	iso.add(cube, redColor);
Isomer.Color	Cria um objeto de cor a partir de valores RGB (0-255).	var redColor = new Isomer.Color(200, 50, 50);

Esta tabela serve como um guia de início rápido, permitindo que a equipe de desenvolvimento se concentre nos componentes exatos da biblioteca necessários

para cumprir a tarefa.

4.2 Lógica do Seletor de Direções

O fluxo de interação para o seletor de direções é direto:

1. O menu de contexto customizado (Seção 3.3) é exibido com botões para cada uma das seis direções hexagonais (ex: Nordeste, Leste, Sudeste, etc.).
2. Quando o usuário clica em um desses botões, o manipulador de eventos associado é acionado.
3. Este handler deve ter acesso a duas informações cruciais: o tileId do hexágono de origem (que foi armazenado no estado da aplicação quando o menu foi aberto) e a direction selecionada (ex: 'northeast').
4. Com essas informações, ele invoca a função principal de desenho de linha, que executará o algoritmo descrito a seguir.

4.3 O Algoritmo de Desenho de Linha: Estendendo a Barra "Até a Borda"

A funcionalidade de "prolongar uma barra até a borda" é mais complexa do que parece. Ela não se trata de desenhar uma linha geométrica até a bordagem em pixels do canvas, mas sim de um problema de **raycasting lógico** dentro da grade hexagonal. A "borda" é um conceito definido pelos limites lógicos da grade, não pelos limites do canvas. A implementação correta deste algoritmo se desdobra em quatro fases distintas.

Fase 1: Obter o Vetor de Direção
Graças ao uso de coordenadas cúbicas, cada uma das seis direções cardeais em uma grade hexagonal corresponde a um vetor de deslocamento constante. Estes vetores podem ser pré-calculados e armazenados em uma estrutura de dados, como um array ou um mapa: const directionVectors = [Hex(1, 0, -1), Hex(1, -1, 0),...]; // Corresponde a E, SE,... 8
Fase 2: Encontrar o Hexágono da Borda (Raycasting Lógico)
Este passo determina o ponto final da linha.

1. Comece com o startHex, que é o tile clicado pelo usuário.
2. Obtenha o directionVector correspondente à direção escolhida pelo usuário.
3. Inicie um loop que avança passo a passo na direção escolhida: let currentHex =

startHex;

4. Em cada iteração, calcule o próximo hexágono na linha: let nextHex = hex_add(currentHex, directionVector);
5. Verifique se nextHex ainda está dentro dos limites lógicos da grade. Isso requer uma função grid.isWithinBoundaries(hex) que deve ser implementada com base na forma da grade (por exemplo, para uma grade retangular, verificaria se hex.q e hex.r estão dentro dos intervalos definidos).[34]
6. Se nextHex estiver fora dos limites, o loop termina. O currentHex é o último hexágono válido na linha e, portanto, o nosso endHex (o hexágono da "borda").
7. Se nextHex estiver dentro dos limites, atualize currentHex = nextHex e continue o loop.[35]

Fase 3: Gerar o Caminho Completo da Linha
Agora que temos um startHex e um endHex, precisamos encontrar todos os hexágonos que formam a linha reta entre eles. O algoritmo hex_linedraw, também do Red Blob Games, é perfeito para isso. Ele funciona através de interpolação linear (lerp) no espaço de coordenadas cúbicas. O algoritmo amostra vários pontos ao longo da linha reta que conecta os centros dos dois hexágonos e, para cada ponto amostrado, arredonda suas coordenadas cúbicas fracionárias para encontrar o hexágono inteiro que o contém.3 O resultado é um array de todos os hexágonos que compõem o caminho.
Fase 4: Renderizar a Linha
Com o array de hexágonos do caminho em mãos, a renderização é simples. No loop de renderização principal, itere sobre este array e desenhe cada hexágono com um estilo visual distinto (por exemplo, uma cor de preenchimento sólida e diferente) para formar a "barra" visual.
A decomposição desta tarefa aparentemente simples em fases distintas — raycast lógico, geração de caminho e renderização — é crucial para uma implementação correta e robusta, evitando a armadilha de confundir os limites lógicos da grade com os limites físicos do canvas.

Seção 5: Arquitetura da Aplicação e Recomendações Finais

A fase final do design arquitetônico é unir todos os componentes discutidos em uma estrutura de aplicação coesa, manutenível e escalável. Esta seção propõe uma arquitetura modular, descreve o fluxo de dados e eventos, e oferece recomendações para garantir a qualidade e a longevidade do software.

5.1 Proposta de Estrutura de Módulos (Separação de Responsabilidades)

A complexidade dos requisitos — lógica de grade, renderização 2D, projeção isométrica 3D, e manipulação de UI do DOM — exige uma separação rigorosa de responsabilidades. Tentar resolver todos esses problemas em um único script monolítico resultaria em um código acoplado e difícil de manter. A arquitetura modular proposta abaixo isola cada domínio de problema em seu próprio módulo, promovendo a testabilidade e a clareza.

- **GridLogic.js (O Cérebro):** Este módulo é o coração lógico da aplicação. Ele encapsula a instância da honeycomb-grid e contém todos os algoritmos puros, que não dependem de renderização ou DOM.
    - **Responsabilidades:** Gerenciar o objeto da grade, fornecer funções para conversão de coordenadas, cálculo de vizinhos, distância, o algoritmo hex_linedraw, e o raycasting lógico para encontrar a borda.
    - **Interface:** Expõe funções como initializeGrid(width, height), getHexPath(startHex, endHex), getHexAt(coords), findBorderHex(startHex, direction).
- **Renderer.js (O Motor Visual):** Este módulo é responsável por toda a interação com o elemento <canvas>.
    - **Responsabilidades:** Gerenciar o contexto de renderização principal e o canvas off-screen para o cubo. Implementar o loop de renderização com requestAnimationFrame. Desenhar a grade, os hexágonos, o cubo e as linhas com base no estado atual da aplicação.
    - **Interface:** Expõe métodos como init(canvasElement), renderScene(state), e a função de conversão pixelToHex(x, y) que internamente usa Grid.pointToHex.
- **StateManager.js (A Única Fonte da Verdade):** Este módulo contém o objeto de estado global da aplicação e gerencia as suas atualizações.
    - **Responsabilidades:** Manter o estado atual (activeHex, hoveredHex, etc.). Implementar um padrão simples de observador (ou pub/sub) para que outros módulos possam se inscrever e ser notificados sobre mudanças de estado.
    - **Interface:** Expõe getState(), setState(newState), subscribe(callback).
- **UIController.js (O Gerente de Interação):** Este módulo lida com toda a interação do usuário que ocorre fora do canvas (elementos DOM) e com a captura de eventos no canvas.
    - **Responsabilidades:** Adicionar e gerenciar todos os event listeners (clique, movimento do mouse, contextmenu). Criar, popular, exibir e ocultar o menu de

contexto. Lidar com quaisquer outros elementos de UI, como campos de entrada para definir o tamanho da grade.
    - **Interface:** Não expõe uma API pública, mas atua como o tradutor entre as interações do usuário e as atualizações de estado, chamando StateManager.setState().
- **main.js (O Ponto de Entrada):** Este é o script principal que inicializa a aplicação, instancia todos os módulos e estabelece as conexões entre eles.

Esta arquitetura modular garante que cada parte do sistema tenha uma responsabilidade única e bem definida, o que é a base para um software de alta qualidade.

5.2 Fluxo de Dados e Eventos: O Ciclo de Vida de uma Interação

A arquitetura proposta utiliza um fluxo de dados unidirecional, que é mais fácil de depurar e raciocinar do que sistemas com comunicação bidirecional complexa. O StateManager atua como o hub central, garantindo um fluxo previsível.

Exemplo de Fluxo (Clique para Desenhar Linha):

1. **Interação do Usuário:** O usuário clica com o botão direito em um hexágono.
2. **UIController:** Captura o evento contextmenu. Chama event.preventDefault(). Usa as coordenadas do evento para chamar Renderer.pixelToHex() e obter o hex clicado.
3. **Atualização de Estado:** O UIController chama StateManager.setState() para atualizar o activeHex e definir o menu de contexto como visível, com as coordenadas do clique.
4. **Notificação:** O StateManager notifica todos os seus "assinantes" (o Renderer e o UIController) de que o estado mudou.
5. **Re-renderização:**
    - O Renderer recebe a notificação, lê o novo estado e redesenha a cena completa: destaca o activeHex e desenha o cubo isométrico (usando o buffer off-screen).
    - O UIController recebe a notificação e atualiza o DOM para exibir o menu de contexto na posição correta.
6. **Seleção de Direção:** O usuário clica em uma opção de direção no menu de contexto.

7. **UIController:** Captura o clique no botão do menu. Chama StateManager.setState() para ocultar o menu e registrar a intenção de desenhar uma linha, armazenando o startHex (que é o activeHex) e a direction selecionada.
8. **Notificação:** O StateManager notifica novamente os assinantes.
9. **Cálculo e Renderização da Linha:**
    - O Renderer lê o novo estado. Ele vê que uma linha precisa ser desenhada.
    - Ele chama o GridLogic para calcular o caminho da linha: primeiro findBorderHex(startHex, direction) para obter o endHex, e depois getHexPath(startHex, endHex) para obter a lista de todos os hexágonos no caminho.
    - O Renderer armazena este caminho e redesenha a cena, colorindo os hexágonos do caminho para formar a barra prolongada.

Este ciclo claro e unidirecional (**Interação → Estado → Renderização**) é a chave para evitar bugs de estado e tornar a aplicação robusta.

5.3 Recomendações de Boas Práticas e Próximos Passos

Para garantir o sucesso a longo prazo do projeto, as seguintes práticas são recomendadas:

- **Persistência de Dados:** O design do modelo de dados para os tiles e suas conexões (Seção 1.4) deve ser feito com a serialização em mente. A estrutura deve ser facilmente conversível para e de JSON, o que permitirá implementar funcionalidades de salvar e carregar os designs dos usuários no futuro. A flexibilidade deste modelo de dados é o elemento mais crítico para a longevidade e o valor futuro da aplicação.
- **Performance em Larga Escala:** Para grades muito grandes (milhares de tiles), renderizar todos os hexágonos em cada quadro se tornará um gargalo de performance. A solução para isso é a **virtualização da renderização** (também conhecida como *culling*). Antes de cada renderização, o sistema deve calcular quais hexágonos estão atualmente dentro da área visível do canvas (a *viewport*) e desenhar apenas eles. Isso reduz drasticamente o número de operações de desenho e mantém a aplicação fluida.[36]
- **Testabilidade:** A arquitetura modular proposta facilita enormemente os testes automatizados. O módulo GridLogic.js, que contém a lógica mais complexa e crítica, é um "módulo puro" — ele não tem dependências de DOM ou de

navegador. Isso significa que ele pode ser submetido a testes de unidade rigorosos usando frameworks como Jest ou Mocha em um ambiente Node.js, garantindo a correção matemática dos algoritmos sem a sobrecarga de um navegador.

- **Extensibilidade Futura:** A aplicação foi projetada como uma plataforma. O modelo de dados flexível para os tiles e suas connections é o principal ponto de extensão. Ao projetá-los como objetos com propriedades customizáveis, a aplicação está preparada para a adição futura de novas funcionalidades com o mínimo de refatoração, como diferentes tipos de terreno (água, montanha), estruturas sobre os tiles, ou conexões com propriedades variadas (paredes, portas, caminhos com custos para algoritmos de pathfinding).

Referências citadas

1. acessado em dezembro 31, 1969, https://github.com/MrMenezesDev/Rascunhos/blob/master/R_241118/R_241118.js
2. acessado em dezembro 31, 1969, https://github.com/MrMenezesDev/Rascunhos/blob/master/R_241118/R_241118.html
3. Hexagonal Grids - Red Blob Games, acessado em agosto 13, 2025, https://www.redblobgames.com/grids/hexagons/
4. Hexagonal Grids - Red Blob Games, acessado em agosto 13, 2025, https://www.redblobgames.com/grids/hexagons-v1/
5. How to draw a hexagonal grid on HTML Canvas | Home, acessado em agosto 13, 2025, https://eperezcosano.github.io/hex-grid/
6. Interactive Hexagon Grid Tutorial Part 1 - Intro, acessado em agosto 13, 2025, https://inspirnathan.com/posts/170-interactive-hexagon-grid-tutorial-part-1/
7. Create a hexagonal map - greentec's blog, acessado em agosto 13, 2025, https://greentec.github.io/hexagonal-map-en/
8. Implementation of Hex Grids - Red Blob Games, acessado em agosto 13, 2025, https://www.redblobgames.com/grids/hexagons/implementation.html
9. Grid parts and relationships - Red Blob Games, acessado em agosto 13, 2025, https://www.redblobgames.com/grids/parts/
10. Honeycomb | A hexagon grid library made in TypeScript. - Abbe Keultjes, acessado em agosto 13, 2025, https://abbekeultjes.nl/honeycomb/
11. honeycomb-grid - NPM, acessado em agosto 13, 2025, https://www.npmjs.com/package/honeycomb-grid
12. Hexagonal Grid – Demo applications & examples - JointJS, acessado em agosto 13, 2025, https://www.jointjs.com/demos/hexagonal-grid
13. Hexmap Library - Open Innovations: Github, acessado em agosto 13, 2025, https://open-innovations.github.io/oi.hexmap.js/
14. Interactive Hexagon Grid Tutorial Part 2 - Color Fade Hover Effects - inspirnathan.com, acessado em agosto 13, 2025,

https://inspirnathan.com/posts/171-interactive-hexagon-grid-tutorial-part-2/
15. HTMLCanvasElement: getContext() method - Web APIs | MDN, acessado em agosto 13, 2025, https://developer.mozilla.org/en-US/docs/Web/API/HTMLCanvasElement/getContext
16. Pixel manipulation with canvas - Web APIs | MDN, acessado em agosto 13, 2025, https://developer.mozilla.org/en-US/docs/Web/API/Canvas_API/Tutorial/Pixel_manipulation_with_canvas
17. Honeycomb: hexagon grids in JavaScript | by Abbe Keultjes - Medium, acessado em agosto 13, 2025, https://medium.com/@Flauwekeul/honeycomb-hexagon-grids-in-javascript-555d2f9ac54f
18. Drawing shapes with canvas - Web APIs | MDN, acessado em agosto 13, 2025, https://developer.mozilla.org/en-US/docs/Web/API/Canvas_API/Tutorial/Drawing_shapes
19. detecting a click inside a hexagon drawn using canvas? - Stack Overflow, acessado em agosto 13, 2025, https://stackoverflow.com/questions/47656890/detecting-a-click-inside-a-hexagon-drawn-using-canvas
20. New demo release: Hexagonal Grid - JointJS, acessado em agosto 13, 2025, https://www.jointjs.com/blog/demo-wednesday-hexagonal-grid
21. More pixel to hex approaches - Red Blob Games, acessado em agosto 13, 2025, https://www.redblobgames.com/grids/hexagons/more-pixel-to-hex.html
22. Unity Tutorial: Convert Mouse Clicks to Hex Grid in Turn-Based Strategy Game Creation, acessado em agosto 13, 2025, https://www.youtube.com/watch?v=pla1K9ZtfxQ
23. Drawing Hexagon on Canvas, testing mouseclick event vs Hexagon - Stack Overflow, acessado em agosto 13, 2025, https://stackoverflow.com/questions/35036585/drawing-hexagon-on-canvas-testing-mouseclick-event-vs-hexagon
24. Hexagonal canvas/button - Getting Started - Xojo Programming Forum, acessado em agosto 13, 2025, https://forum.xojo.com/t/hexagonal-canvas-button/36181
25. Recognizing a hexagonal clickbox - Game Development Stack Exchange, acessado em agosto 13, 2025, https://gamedev.stackexchange.com/questions/15110/recognizing-a-hexagonal-clickbox
26. How to show a context menu for HTML5 canvas shape? | Konva ..., acessado em agosto 13, 2025, https://konvajs.org/docs/sandbox/Canvas_Context_Menu.html
27. How to add a custom right-click menu to a webpage? - Stack Overflow, acessado em agosto 13, 2025, https://stackoverflow.com/questions/4909167/how-to-add-a-custom-right-click-menu-to-a-webpage
28. Create complex context menu - Canvas Datagrid - JS.ORG, acessado em agosto 13, 2025, https://canvas-datagrid.js.org/examples/create-complex-context-menu/
29. Can you do an isometric perspective with HTML5 ? - Stack Overflow, acessado

em agosto 13, 2025, https://stackoverflow.com/questions/5186013/can-you-do-an-isometric-perspective-with-html5-canvas
30. Basic isometric projection in Javascript - Game Development Stack Exchange, acessado em agosto 13, 2025, https://gamedev.stackexchange.com/questions/48423/basic-isometric-projection-in-javascript
31. html5 canvas correctly draw a cube - javascript - Stack Overflow, acessado em agosto 13, 2025, https://stackoverflow.com/questions/44829028/html5-canvas-correctly-draw-a-cube
32. Isomer – an isometric graphics library for HTML5 canvas, acessado em agosto 13, 2025, https://jdan.github.io/isomer/
33. How to draw isometric cubes with Javascript and HTML canvas - Lieu Zheng Hong, acessado em agosto 13, 2025, https://www.lieuzhenghong.com/isometric_cubes/
34. hex-grid.js - GitHub, acessado em agosto 13, 2025, https://github.com/euoia/hex-grid.js?files=1
35. Clark Verbrugge's Hex Grids - Stanford, acessado em agosto 13, 2025, http://www-cs-students.stanford.edu/~amitp/Articles/HexLOS.html
36. Drawing isometric map in canvas / javascript - Game Development Stack Exchange, acessado em agosto 13, 2025, https://gamedev.stackexchange.com/questions/25340/drawing-isometric-map-in-canvas-javascript
37. Simple Isometric Tiling - Aschenblog, acessado em agosto 13, 2025, http://nick-aschenbach.github.io/blog/2015/02/25/isometric-tile-engine/