Astronomia interativa: uma abordagem de software educativo para o ensino-aprendizagem

João Guilherme Cavalcante Alves Dias de Araújo, Vitor de Oliveira Silva

Universidade Cruzeiro do Sul (UNICSUL) – São Paulo – SP – Brasil

jgcadda@gmail.com, vitor.osilva02@gmail.com

Resumo: O universo é uma maravilha repleta de mistérios e fenômenos complexos. O estudo de astronomia enriquece nossa compreensão dessas maravilhas, desvendando os segredos que tornam o cosmos tão extraordinário. Infelizmente, essa matéria vem perdendo espaço nas escolas por diversas razões e o objetivo deste projeto é estudar esse processo, junto com a criação de uma solução tecnológica para o problema. Decidimos criar uma ferramenta gratuita e de fácil acesso em JavaScript para auxiliar o estudo da disciplina sobre a gravidade e energia escura, ambos conceituais importantes e fascinantes, com a esperança de ajudar professores a prender a atenção dos alunos e explicar esses conceitos de forma visual e interativa.

Palavras-chave: Software. Simulação. Educação. Física. Astronomia. Gravidade.

Interactive astronomy: an approach to teaching and learning using educational software

Abstract: The universe is a thing of wonder, filled with mysteries and complex phenomena. The study of astronomy enriches our understanding of these wonders, unveiling the secrets that make the cosmos so extraordinary. Sadly, this subject has been losing space in schools for multiple reasons, and the purpose of this paper is to study this process, along with the creation of a technological solution for the issue. We decided to create a free and easy-to-access tool in JavaScript to help the study of the subject on gravity and dark matter, both important and fascinating concepts, with the hope of helping teachers capture the attention of students and explain these concepts in a visual and interactive manner.

Keywords: Software. Simulation. Education. Physics. Astronomy. Gravity.

1. Introdução

Nossa intenção com o presente trabalho é entender o estado atual do ensino da ciência astronômica e, possivelmente, contribuir para sua longa trajetória com o uso de representações digitais e interativas da gravidade e da energia escura em ação, através de simulações. A astronomia é uma das ciências mais antigas de que se tem notícia – conforme demonstram indícios como o círculo de rochas de Nabta Playa, descoberto no Egito e feito há mais de sete mil anos (Betz, 2020) – e sua origem é a necessidade humana de entender a realidade em que vivemos e suas transformações. Um exemplo dessa demanda é a motivação para compreender as estações do ano e as suas mudanças.

Ao longo das eras, esse conhecimento passou por diversas transformações. As descobertas realizadas na ciência aumentaram nosso conhecimento a respeito da imensidão do universo, ao passo que o ensino a seu respeito tem, lamentavelmente, perdido espaço nas escolas. Essa perda de espaço tem como possíveis causas a falta de tempo para a abordagem adequada da disciplina, a escassez de materiais didáticos e a falta de formação adequada dos professores.

Há diversas ferramentas de representação que podem ser utilizadas no ensino; exemplos disso incluem imagens, maquetes e, mais recentemente, simulações baseadas em software. Tais simulações, sobretudo quando comportam recursos interativos, trazem diversas vantagens para o aprendizado científico, incluindo a possibilidade de visualizar fenômenos físicos aos quais não temos acesso a olho nu – como a relação entre a ocorrência de estações do ano e a angulação da terra em relação ao plano do sistema solar (Beserra, 2016) – e a comparação entre cenários partindo de um mesmo princípio físico – como a ação da gravidade no movimento de corpos com diferentes massas. Como nos ensina Neres (2017), a utilização de simulações pode ser de grande valia ao permitir a visualização de fenômenos astronômicos complexos, o desenvolvimento de hipóteses e a realização de experimentos por parte dos próprios alunos.

Softwares de simulação proporcionam uma incrível liberdade por criarem um mundo virtual no qual parâmetros, ângulos e escalas podem ser manuseados livremente. Esses programas podem auxiliar os discentes em assuntos que exigem visualizações tridimensionais ou progressões temporais, por vezes difíceis de representar por meios tradicionais e cruciais para o aprendizado da astronomia.

Considerando a presença cada vez maior da tecnologia na vida de todos, incluindo professores e alunos, essa ferramenta se tornou um recurso fundamental para a pedagogia, o que nos convida enquanto acadêmicos a entender sua utilização. Para Brianeza e Malacarne (2013), o uso da tecnologia se encaixa muito no terceiro momento pedagógico descrito por Angotti e Delizoicov (1994, apud Brianeza; Malacarne, 2013), que se refere à sistematização do conhecimento do aluno e é a etapa na qual a aprendizagem se consolida. Para analisar essa hipótese, as autoras realizaram atividades envolvendo astronomia com alunos do 8º ano do ensino fundamental, constatando que exercícios interativos e práticos com simulações de software, experimentos e jogos obtiveram um maior engajamento dos alunos em relação a outras atividades realizadas.

Vemos assim que as metodologias alternativas para o ensino de física vêm ganhando cada vez mais destaque. Porém, vale ressaltar que o interesse em formas mais eficientes de se ensinar ciências não vem de hoje. Como afirma Ribeiro (2020), desde os anos 1970, com o lançamento do Sputnik I – o primeiro satélite artificial do planeta –, o interesse na matéria de física aumentou globalmente e foi impulsionado pela corrida espacial. Houve assim um grande foco no desenvolvimento dessa disciplina, que na época já possuía uma aproximação com os computadores, mas também foram identificados desafios no seu ensino, muitos dos quais se perpetuam até hoje: pouco conhecimento matemático dos estudantes e concepções de mundo alternativas que não seguem a lógica científica, além de métodos tradicionais de ensino que se concentram excessivamente na utilização de cálculos em vez de uma abordagem mais conceitual. Em resumo, há indícios de que o ensino de ciências é prejudicado quando se afasta das atividades práticas e quando há exagero em aulas expositivas sem uma descrição contextual que se aproxime mais dos alunos.

Nesse contexto, portanto, uma possível solução para esses problemas é a conciliação de metodologias mais tradicionais com metodologias complementares, como o ensino de ciências a partir de simulações por software, sendo esse o objetivo do presente trabalho.

No entanto, antes abordar a criação do software, é necessário definir os fundamentos e delimitações do projeto. No caso, optamos por focar na simulação da gravidade e da energia escura, visando tanto a educação quanto a divulgação científica e buscando esclarecer a complexidade da dinâmica entre esses dois fenômenos que, como apontado por Menezes (2009), desempenham papéis cruciais na cosmologia moderna. Nela, entende-se que o universo está se expandindo aceleradamente – conforme mostra a lei de Hubble –, desafiando assim a compreensão tradicional da gravidade e sugerindo a presença da enigmática energia escura, que parece se opor à atração gravitacional entre massas e desafia o conceito de um universo estático, além de pôr em questão as previsões das equações de campo de Einstein, que previam o colapso do universo sobre si próprio por conta da gravidade.

2. Revisão da literatura

Citaremos agora alguns trabalhos acadêmicos que investigam o ensino de física e ciências no âmbito escolar mediado por simulações geradas por software.

Em “A astronomia no ensino fundamental e o uso do software JClic” (Brianeza; Malacarne, 2013), as autoras descrevem os resultados do uso de diversos recursos pedagógicos para transmitir conceitos da astronomia para alunos do 8º ano do ensino fundamental, dentre eles o software JClic – utilizado para que os alunos elaborassem um jogo de memória com os assuntos em pauta. Os diversos jogos, experimentos e simulações empregados, por fomentarem a participação dos alunos, favoreceram a absorção de conceitos científicos, tornando o processo de aprendizagem mais lúdico e próximo dos alunos.

O artigo “O uso de simuladores no ensino de astronomia”, de Becker e Strieder (2011), trata das possibilidades pedagógicas do uso do software Stellarium, capaz de

emular a observação de corpos celestes e fenômenos da astronomia. Nesse trabalho, há uma descrição de oficinas de educação continuada para professores do ensino fundamental e médio com o propósito de capacitar os frequentadores das oficinas no uso do Stellarium para fins didáticos.

O trabalho "Ensino de astronomia com os softwares Stellarium e Celestia", de Beserra e colaboradores (2016), oferece uma investigação dos fenômenos astronômicos capazes de serem simulados utilizando os softwares mencionados no título do artigo, considerando inclusive as concepções alternativas ou errôneas entre alunos – e até mesmo docentes – que podem ser evitadas com o estudo das emulações produzidas pelos programas. Este artigo é de interesse específico para a área da computação por incluir indicações a respeito da construção de scripts para o programa Celestia.

Em "Cosmologia: de Einstein à energia escura" (Alcaniz, 2007), discute-se as crescentes evidências da expansão acelerada do universo. Segundo o artigo, a compreensão clássica da gravidade vem sendo desafiada pela presença de uma energia que contraria a atração gravitacional. Essa energia, conhecida como energia escura, constitui aproximadamente 3/4 da densidade total de energia do universo e é a força motriz por trás de sua expansão acelerada. Este fenômeno enigmático, que se alinha com as observações de supernovas do tipo Ia, ressalta a importância de revisitar os princípios cosmológicos estabelecidos e considerar novas teorias que possam explicar a aceleração cósmica observada, potencialmente reformulando nosso entendimento sobre a estrutura e o destino do cosmos.

Encerrando esta breve seleção de artigos relevantes, mas sem esgotar o material acadêmico escrito a respeito do assunto, temos a dissertação de mestrado "O Stellarium como estratégia para o ensino de astronomia" (Neres, 2017). Partindo da teoria da aprendizagem significativa de David Ausubel, Neres (2017) descreve uma sequência de ensino e aprendizagem feita para professores de ensino médio que desejam utilizar o software Stellarium para o ensino da astronomia em aulas de física. Além de uma sugestão pedagógica embasada, o trabalho se destaca por incluir um manual para o uso do software e explicações de seus principais comandos para gerar simulações astronômicas.

3. Metodologia

Para o presente projeto, tem-se como metodologia a pesquisa bibliográfica e a construção de um software de simulação com finalidade educativa. A pesquisa bibliográfica será realizada nas plataformas Google Acadêmico e Portal CAPES a partir das palavras-chave "simulação", "física", "astronomia", "educação", "gravidade" e termos correlatos. Os artigos serão selecionados por critério de relevância, além de serem priorizadas publicações mais recentes.

O software, por sua vez, é desenvolvido em linguagem JavaScript, linguagem de programação para plataformas web, de modo que funcione em uma ampla gama de dispositivos (como computadores, notebooks e celulares tradicionais). Sua confecção obedece às seguintes etapas: a) desenvolvimento do software em HTML e CSS com as bases para a simulação, b) adição de uma interface de usuário com navegação fluida, c) criação de textos para elucidar os tópicos abordados, d) construção de um ambiente 2D

com funções que se aproximam dos cálculos de física e e) inserção de funcionalidades interativas que engajam o usuário na exploração dos conceitos científicos apresentados.

4. Resultados

4.1 Primeiros passos

Seguindo o ciclo de vida de desenvolvimento de software (SDLC, Software Development Life Cycle), já cumprimos a primeira fase, a concepção, em que, como discutido anteriormente neste artigo, exploramos as potenciais melhorias no ensino de conceitos físicos complexos. Esta fase inicial envolveu uma análise das necessidades educacionais e a identificação de áreas nas quais uma ferramenta interativa poderia oferecer ajuda.

Prosseguindo para a segunda fase, a definição de requisitos, temos a descrição das especificações técnicas e pedagógicas que o software deve atender. Os requisitos funcionais estabelecidos foram a capacidade de simular a gravidade e a energia escura em 2D e a capacidade de ajustar parâmetros como massa, distância e constantes cosmológicas. Já os requisitos não funcionais são usabilidade, compatibilidade com diversas plataformas e tempo de resposta otimizado na simulação.

A terceira fase é o projeto, no qual traduzimos os requisitos coletados em um plano para o desenvolvimento do software. Aqui temos o projeto da arquitetura do software centralizada em tecnologias web front-end, usando HTML5 para estruturação do conteúdo, CSS para estilização e JavaScript como linguagem de programação. A modelagem dos dados foi projetada para representar corpos celestes, enquanto a interface do usuário foi modelada para permitir aos usuários manipular variáveis e visualizar os resultados da simulação em tempo real. O núcleo do software, implementado em JavaScript, gerencia a lógica das fórmulas de física da simulação, incluindo cálculos detalhados de gravidade e energia escura. Essas abordagens têm como foco a escalabilidade, a manutenção, a facilidade de execução e a capacidade de funcionar em uma ampla quantidade de dispositivos, como computadores desktop, notebooks e celulares.

A quarta fase, a de codificação, contempla a efetiva implementação do projeto. Essa etapa é crucial, pois é onde as ideias e planos estabelecidos nas fases anteriores são transformados em um código funcional.

4.2. Desenvolvimento HTML

Começamos pelo cabeçalho HTML, que contém as configurações mais gerais e aspectos fundamentais da página, como metadados e a vinculação do CSS e do JavaScript ao HTML. Neste ponto, definimos o nome da página como “Cosmos em Equilíbrio: Gravidade e Energia Escura Desvendadas”. Também escolhemos uma imagem para ser o ícone, além de importar a fonte ‘Exo’ do Google Fonts para aprimorar a estética e o título da página.

```html
<!DOCTYPE html>
<html lang="pt-BR">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Cosmos em Equilíbrio: Gravidade e Energia Escura Desvendadas</title>
    <link href="https://fonts.googleapis.com/css2?family=Exo:wght@400;700&display=swap" rel="stylesheet">
    <link rel="icon" href="Arquivos/logo.webp" type="image/png">
    <link rel="stylesheet" href="styles.css">
    <script src="main.js" defer> </script>
</head>
```

Figura 1. HTML parte 01. Fonte: próprio autor.

Figura 2. Icone utilizado. Fonte: Canvas, Nuur Studio.

Após a seção de cabeçalho, elaboramos o corpo do documento, que contém elementos essenciais para a estrutura e o conteúdo da página. Entre eles, a tag ‘<img src...>’ insere uma imagem no documento (neste caso, com propósitos estéticos), as tags ‘<h1>’, ‘<h2>’ e ‘<h3>’ definem títulos de texto, estabelecendo uma hierarquia clara e organizada e a tag ‘<p>’ é usada para parágrafos de texto, fornecendo explicações e informações sobre a simulação. Na imagem a seguir, são exibidas as primeiras explicações como exemplos, enquanto as demais estão cortadas para simplificar a visualização. As tags ‘ ’ e ‘<section>’ organizam e estruturam o conteúdo da página web. Já os atributos ‘class’ e ‘id’ são cruciais para associar estas tags HTML ao CSS e ao JavaScript, permitindo assim sua manipulação e estilização eficazes. Além disso, utilizamos a tag ‘<a>’ para criar links para acessar este artigo e para acessar uma segunda página.

```html
<body>
    <section class="cabecalho">
        <img src="Arquivos/universo.png" alt="Imagem de fundo representando o espaço">
        <div class="Texto">

            <div>
                <h1>TGI</h1>
                <h2>Forças em Conflito: Uma Jornada Interativa pela Gravidade e Energia Escura</h2>
                <h3> João Guilherme Cavalcante 23384031 <br/>  &amp;amp; Vitor de Oliveira Silva 23384301</h3>
            </div>

        </div>
    </section>

    <section class="sobre-projeto">
        <div id="Texto2">

            <p>O universo é um vasto e intrigante espaço de mistérios. Neste projeto, exploramos duas forças fundamentais que o moldam: a gravidade e a energia escura. Através de simulações
            interativas, você terá uma visão única de como elas atuam e influenciam a dança das galáxias e estrelas,</p>

            <p>Para acessar a versão completa do trabalho, clique no link <a href="Arquivos/Astronomia Interativa - João Guilherme C A D de Araújo e Vitor O Silva 19 11 2023 (1).pdf"
            target="_blank"> "Astronomia Interativa.PDF"</a></p>
```

Figura 3. HTML parte 02. Fonte: próprio autor.

Figura 4. imagem de background. Fonte: próprio autor, através das ferramentas de IA do Photoshop.

Para finalizar a seção do HTML, criamos ‘<divs>’ com a classe ‘controle’ que abrigarão botões para interação com a simulação. Esses botões, manipulados via Javascript, ajustarão a força da gravidade e a força da energia escura, permitindo ao usuário utilizar funções de zoom-in e zoom-out e alternar entre diferentes tipos de simulação. Em seguida, temos a tag ‘<Canvas>’, que define uma área específica para gráficos gerados através do JavaScript, que será importante posteriormente.

```
47        <div class="controles-gerais">
48            <div class="controles-simulacao">
49                <button id="iniciar-simulacao">Simular Gravidade</button>
50                <button id="iniciar-simulacao2">Simular Energia Escura</button>
51                <button id="iniciar-simulacao3">Simular Os Dois</button>
52            </div>
53
54            <div class="controles-gravidade">
55                <button id="aumentar-gravidade">+ Gravidade</button>
56                <button id="diminuir-gravidade">- Gravidade</button>
57                <span id="valor-gravidade">G: 10</span>
58            </div>
59
60            <div class="controles-energia-escura">
61                <button id="aumentar-energia-escura">+ Energia Escura</button>
62                <button id="diminuir-energia-escura">- Energia Escura</button>
63                <span id="valor-energia-escura">E: 0.0010</span>
64            </div>
65            <div class="controles-zoom">
66                <button id="zoom-in">+ Zoom</button>
67                <button id="zoom-out">- Zoom</button>
68                <span id="valor-zoom">Z: 100%</span>
69            </div>
70        </div>
71
72        <div id="simulacao-container">
73            <canvas id="canvas"></canvas>
74        </div>
75    </section>
76 </body>
77 </html>
```

Figura 5. HTML parte 03. Fonte: próprio autor.

4.3. Desenvolvimento CSS

Na folha de estilo CSS, os elementos que estão no ‘body’ (‘font-family: ‘Exo’, sans-serif’, ‘color: white’ e ‘background-color: #000’) são utilizados para estabelecer, respectivamente, a fonte ‘Exo’ como padrão para todo o texto da página, a cor do texto como branca e a cor de fundo da página como preta. Essa combinação de estilos contribui para criar um design que remete ao tema espacial da página. Os outros elementos de estilo são utilizados para assegurar um posicionamento adequado e a clara visibilidade dos componentes da página, reforçando a coesão e a estética do layout. Por exemplo: com a classe ‘.cabecalho img’ as imagens são estilizadas para ocupar 100% da largura da tela mantendo sua proporção original e a classe ‘.Texto’ coloca o conteúdo textual absolutamente sobre outras camadas, centralizando-o sobre a imagem. Já as classes ‘.cabecalho’ e ‘.sobre-projeto’ são posicionadas de forma relativa e centralizada, fazendo com que a página se adapte como um todo a telas diferentes.

```css
body {
    font-family: 'Exo', sans-serif;
    color: white;
    margin: 0;
    padding: 0;
    background-color: #000;
}

.cabecalho {
    position: relative;
    text-align: center;
    padding: 1% 5%;
}

.sobre-projeto {
    position: relative;
    text-align: center;
    padding: 0% 15% 5%;

}

.cabecalho img {
    width: 100%;
    height: auto;
}

.Texto {
    position: absolute;
    top: 55%;
    left: 50%;

}
```

Figura 6. CSS parte 01. Fonte: próprio autor.

No trecho a seguir do código CSS, definimos o estilo para os controles da simulação, incluindo os de gravidade, energia escura e zoom. Cada conjunto de controles é estilizado para se apresentar em uma coluna flexível, com os itens alinhados ao centro e um espaçamento uniforme de 10px. Para os botões e spans, aplicamos um ‘padding’ de ‘10px’ por ‘20px’, um fundo na cor ‘#333’, texto em branco ‘#fff’, borda branca sólida e bordas arredondadas, com tamanho de 16px e alinhamento central. Os botões têm uma largura mínima de ‘100px’ e mudam para a cor ‘#6376E6’ quando passamos o mouse sobre eles, melhorando assim a interatividade e a experiência do usuário.

```css
.controles-gerais {
    display: flex;
    justify-content: center;
    align-items: center;
    gap: 50px;
    padding: 5% 10%;
}

.controles-simulacao,
.controles-gravidade,
.controles-energia-escura,
.controles-zoom{
    display: flex;
    flex-direction: column;
    align-items: center;
    gap: 10px;
}

button,
span {
    padding: 10px 20px;
    background-color: #333;
    color: #fff;
    border: 1px solid #fff;
    border-radius: 5px;
    font-family: 'Exo', sans-serif;
    font-size: 16px;
    text-align: center;
    min-width: 100px;
}

button {
    cursor: pointer;
}

button:hover {
    background-color: #6376E6;
}
```

Figura 7. CSS parte 02. Fonte: próprio autor.

Neste próximo trecho de código, configuramos o estilo para o contêiner da simulação, identificado pelo ID ‘#simulacao-container’. Definimos sua largura e altura como 100% para ocupar todo o espaço disponível, com posição relativa e overflow oculto. O contêiner é configurado para exibir seus elementos internos em ‘flex’, centralizando-os horizontal e verticalmente, e é delimitado por uma borda sólida de 1px na cor #ccc.

Para o elemento ‘canvas’, aplicamos um fundo preto (#000) e definimos seu ponto de transformação no centro. Este posicionamento garante que quaisquer transformações aplicadas ao ‘canvas’ ocorram em torno de seu ponto central.

Outra parte importante do CSS é a consulta de mídia, ‘@media only screen’, utilizada para aplicar estilos específicos a uma página web com base nas características do dispositivo em que está sendo visualizada.

```css
#simulacao-container {
    width: 100%;
    height: 100%;
    position: relative;
    overflow: hidden;
    display: flex;
    justify-content: center;
    align-items: center;
    border: 1px solid #ccc;
}

canvas {
    background: #000;
    transform-origin: center center;
    position: relative;
}

@media only screen and (max-width: 768px) {
    .Texto {
        top: 60%;
    }
    .Texto2 {
        font-size: 90%;
    }

    .Texto h1, .Texto h2, .Texto h3 {
        font-size: 90%;
    }
}
```

Figura 8. CSS parte 03. Fonte: próprio autor.

4.4. Desenvolvimento Javascript

Essa é a parte mais complexa do projeto, na qual implementamos diversas funções e cálculos cruciais para a simulação. Iniciamos com a definição de constantes-chave no código: ‘const G = 100’, ‘const taxaDeExpansao = 0.001’ e ‘const Limiar = 1900’. Essas constantes propiciam, respectivamente, uma aproximação da constante gravitacional, um valor padrão para a expansão do universo e um limiar para separar o domínio de ação da energia escura do domínio de ação da gravidade. Em seguida, definimos variáveis auxiliares que funcionam como pontos de controle nas funções, essenciais para a interatividade da simulação.

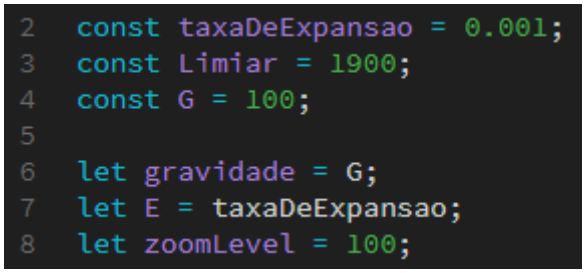

Figura 9. JS parte 01. Fonte: próprio autor.

Após a configuração inicial, o próximo passo é ir atrás dos objetivos específicos da simulação. Para isso, inicializamos um ‘array’ que servirá como repositório dos objetos da simulação. Além disso, desenvolvemos uma função de criação, responsável por configurar cada objeto com características únicas. Nessa função, atribuímos uma cor aleatória a cada objeto para facilitar a identificação visual. Também definimos seus atributos, como a posição e o raio para a circunferência do objeto. A massa de cada objeto é determinada com base em seu raio, estabelecendo uma relação direta entre o tamanho visual do objeto e sua massa física.

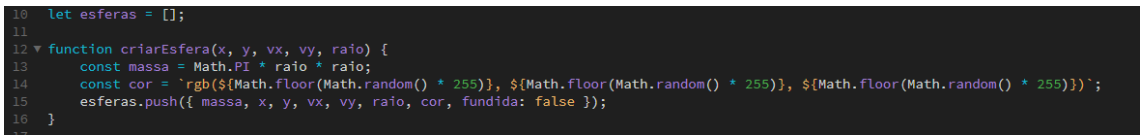

Figura 10. JS parte 02. Fonte: próprio autor.

Para garantir que a nossa simulação tenha um bom desempenho na questão gráfica, foram criadas algumas linhas de código ligadas ao ‘canvas’.

Primeiro, capturamos o elemento ‘simulacao-container’ e o elemento ‘canvas’ do ‘Document Object Model (DOM)’ usando seus respectivos IDs. Em seguida, adquirimos o contexto 2D do ‘canvas’, que nos permite realizar operações gráficas bidimensionais.

Definimos a largura e a altura do ‘canvas’ para preencher toda a área possível, assegurando que a simulação utilize o espaço máximo disponível. Então, calculamos e armazenamos as coordenadas centrais do ‘canvas’ nas variáveis ‘centroX’ e ‘centroY’, que serão usadas como referência para o desenho centrado dos gráficos na simulação.

Por fim, estabelecemos as seguintes funções: ‘ajustarTamanhoCanvas’, que garante que a área de desenho se adapte dinamicamente ao tamanho da janela do navegador; ‘desenharEsferas’, que renderiza cada esfera com um efeito visual detalhado, utilizando ‘ctx.createRadialGradient’ para criar um gradiente radial que confere às esferas um aspecto luminoso e tridimensional, com cores que desaparecem suavemente até a transparência nas bordas; e ‘desenharFundo’, que atualiza o ‘canvas’ aplicando uma camada de opacidade (definida por “ctx.fillStyle = ‘rgba(0, 0, 0, 4)’”) para efetivamente limpar os rastros deixados pelas esferas em movimento.

Essas técnicas asseguram que o ‘canvas’ seja continuamente renovado, mantendo a clareza da visualização. Nossa abordagem assegura que a simulação sempre utilize o espaço disponível de forma otimizada, proporcionando uma experiência visual consistente, independentemente das mudanças no ambiente de visualização.

```
const container = document.getElementById('simulacao-container');
const canvas = document.getElementById('canvas');
const ctx = canvas.getContext('2d');

canvas.width = window.innerWidth;
canvas.height = window.innerHeight;
const centroX = canvas.width / 2;
const centroY = canvas.height / 2;

function ajustarTamanhoCanvas() {
    canvas.width = window.innerWidth;
    canvas.height = window.innerHeight;
}

function desenharEsferas() {
    esferas.forEach(esfera => {
        let gradiente = ctx.createRadialGradient(
            esfera.x, esfera.y, 0,
            esfera.x, esfera.y, esfera.raio
        );
        gradiente.addColorStop(0, esfera.cor);
        gradiente.addColorStop(0.2, esfera.cor);
        gradiente.addColorStop(1, 'rgba(0, 0, 0, 0)');

        ctx.beginPath();
        ctx.arc(esfera.x, esfera.y, esfera.raio, 0, Math.PI * 2);
        ctx.fillStyle = gradiente;
        ctx.fill();
    });
}

function desenharFundo() {
    ctx.save();
    ctx.setTransform(1, 0, 0, 1, 0, 0);
    ctx.fillStyle = 'rgba(0, 0, 0, 4)';
    ctx.fillRect(0, 0, canvas.width, canvas.height);
    ctx.restore();
}

ajustarTamanhoCanvas();
window.addEventListener('resize', ajustarTamanhoCanvas);
```

Figura 11. JS parte 03. Fonte: próprio autor.

Nesse ponto, chegamos ao cerne da nossa simulação. A função ‘gravidade’ é responsável por aplicar as leis fundamentais da gravitação universal aos objetos simulados. Essa função percorre todas as esferas, calculando e aplicando as forças gravitacionais entre elas de acordo com a clássica equação da gravidade:

$$F = G * (m1 * m2) / r^2$$

Para cada par de esferas, a função calcula a distância entre elas e, utilizando a variável gravitacional 'gravidade', determina a força gravitacional mútua. Essa força é então decomposta nos componentes ‘ax’ e ‘ay’ (aceleração nos eixos x e y), que são usados para ajustar as velocidades das esferas, replicando assim o movimento realístico que esperaríamos sob a influência da gravidade. A verificação de distância assegura que as forças só sejam aplicadas quando as esferas não estiverem muito próximas, evitando cálculos desnecessários ou resultados físicos imprecisos.

```
function Gravidade() {
    esferas.forEach((esfera1, i) => {
        esferas.forEach((esfera2, j) => {
            if (i !== j && !esfera1.fundida && !esfera2.fundida) {
                const dx = esfera2.x - esfera1.x;
                const dy = esfera2.y - esfera1.y;
                const distancia = Math.sqrt(dx * dx + dy * dy);

                if (distancia < 1) return;

                    const forca = gravidade * (esfera1.massa * esfera2.massa) / (distancia * distancia);
                    const ax = forca * dx / distancia;
                    const ay = forca * dy / distancia;

                    esfera1.vx += ax / esfera1.massa;
                    esfera1.vy += ay / esfera1.massa;
                    esfera2.vx -= ax / esfera2.massa;
                    esfera2.vy -= ay / esfera2.massa;
            }
        });
    });
}
```

Figura 12. JS parte 04. Fonte: próprio autor.

O outro núcleo do nosso código é a função ‘expandirUniverso’, cuja tarefa é simular a expansão cósmica. Esta função passa sobre o ‘array’ de esferas, analisando cada par para determinar suas posições relativas (no caso, usando as coordenadas x e y para calcular a distância entre cada par de esferas). Quando essa distância supera o ‘Limiar’ estabelecido, o que indica que as esferas estão suficientemente distantes uma da outra, a função simula o efeito da expansão cósmica.

A expansão é modelada aplicando um fator ‘E’, que representa a taxa de expansão do universo, às diferenças das coordenadas x (dx) e y (dy) das esferas. O resultado é um ajuste nas posições das esferas que simula o movimento de afastamento mútuo, com cada esfera se movendo pela metade do cálculo da expansão em direções opostas. Este mecanismo de afastamento é fundamental para representar o fenômeno da expansão do universo em grande escala, onde as galáxias se afastam umas das outras à medida que o espaço entre elas se estende.

```
function expandirUniverso(Limiar) {
    for (let i = 0; i < esferas.length; i++) {
        for (let j = i + 1; j < esferas.length; j++) {
            const esfera1 = esferas[i];
            const esfera2 = esferas[j];

            const dx = esfera2.x - esfera1.x;
            const dy = esfera2.y - esfera1.y;
            const distancia = Math.sqrt(dx * dx + dy * dy);

            if (distancia > Limiar) {

                        const expansaoX = dx * E;
                        const expansaoY = dy * E;

                        esfera1.x -= expansaoX / 2;
                        esfera1.y -= expansaoY / 2;
                        esfera2.x += expansaoX / 2;
                        esfera2.y += expansaoY / 2;
            }
        }
    }
}
```

Figura 13. JS parte 05. Fonte: próprio autor.

Também temos a função de colisão, que é uma parte vital da simulação tratando da interação física entre as esferas. Ela percorre o ‘array’ de esferas, verificando cada par para determinar se ocorreu uma colisão. Quando a distância entre duas esferas é menor que a soma de seus raios, uma colisão é detectada. Nesse caso, as esferas colididas são substituídas por uma nova esfera, cujas propriedades – massa, raio, posição e velocidade – são calculadas com base na conservação da massa e do momento. O novo raio é determinado pela soma das áreas das esferas, enquanto a nova massa é a soma das massas das esferas colididas, de acordo com os princípios da física. As novas posições e velocidades são calculadas para refletir o movimento combinado das esferas após a colisão. Essa abordagem não apenas garante uma simulação realista das colisões como também preserva as leis de conservação, mantendo a integridade física do sistema simulado.

```
function Colisao() {
    let esferasParaRemover = new Set();
    for (let i = 0; i < esferas.length; i++) {
        for (let j = i + 1; j < esferas.length; j++) {
            const esfera1 = esferas[i];
            const esfera2 = esferas[j];
            if (!esferasParaRemover.has(esfera1) && !esferasParaRemover.has(esfera2)) {
                const dx = esfera2.x - esfera1.x;
                const dy = esfera2.y - esfera1.y;
                const distancia = Math.sqrt(dx * dx + dy * dy);

                if (distancia < esfera1.raio + esfera2.raio) {
                    const novaMassa = esfera1.massa + esfera2.massa;
                    const novoRaio = Math.sqrt(esfera1.raio * esfera1.raio + esfera2.raio * esfera2.raio);
                    const novoVx = (esfera1.vx * esfera1.massa + esfera2.vx * esfera2.massa) / novaMassa;
                    const novoVy = (esfera1.vy * esfera1.massa + esfera2.vy * esfera2.massa) / novaMassa;
                    const novoX = (esfera1.x * esfera1.massa + esfera2.x * esfera2.massa) / novaMassa;
                    const novoY = (esfera1.y * esfera1.massa + esfera2.y * esfera2.massa) / novaMassa;

                    criarEsfera(novoX, novoY, novoVx, novoVy, novoRaio);

                                esferasParaRemover.add(esfera1);
                                esferasParaRemover.add(esfera2);
                }
            }
        }
    }

    esferas = esferas.filter(esfera => !esferasParaRemover.has(esfera));
}
```

Figura 14. JS parte 06. Fonte: próprio autor.

Com essa estrutura, estabelecemos uma base sólida para a simulação. No entanto, para enriquecê-la com interatividade, implementamos no trecho seguinte funções acionadas por botões. Por exemplo, a função ‘aumentarGravidade’ dobra o valor da força gravitacional, contanto que esta não esteja excessivamente elevada, evitando assim possíveis bugs. Após essa alteração, a função ‘atualizarMostradorGravidade’ é chamada para atualizar a tag ‘’ com o valor atual da gravidade, permitindo que o usuário acompanhe as mudanças.

Seguindo essa mesma lógica, implementamos funções para diminuir a gravidade, ajustar a energia escura e modificar o zoom da simulação. Cada uma dessas ações é refletida na interface do usuário, proporcionando uma experiência dinâmica e interativa que permite explorar diferentes aspectos da simulação.

Porém, a função ‘aplicarZoom()’ se destaca das demais por ser responsável também por alterar a escala do canvas de forma proporcional ao nível de zoom. Ela define uma constante chamada 'escala', calculada a partir da variável ‘zoomLevel’ dividida por 100, e emprega o método ‘ctx.setTransform’ para ajustar adequadamente essa ‘escala’ ao contexto do desenho do canvas.

Figura 15. JS parte 07. Fonte: próprio autor.

Por último, mas não menos importante, temos os trechos finais de código da simulação: A função ‘animar’ é o coração dinâmico da nossa simulação, controlando o fluxo contínuo de atualizações e renderizações. Ela se inicia com a variável ‘ultimoTempo’, usada para calcular o intervalo de tempo (deltaTime) entre cada quadro da animação. Esta abordagem garante que a simulação progrida a um ritmo consistente, independentemente da velocidade do processador ou da carga da CPU.

Logo em seguida, há uma chamada das funções da simulação – ‘gravidade()’ ‘expandirUniverso()’ e ‘colisões()’ – e as esferas se movem com base em suas velocidades e no tempo transcorrido. Após a atualização das posições, as funções ‘desenharFundo()’ e ‘desenharEsferas()’ são executadas para limpar o quadro anterior e desenhar as esferas em suas novas posições.

A cada ciclo da animação, a função‘requestAnimationFrame(animar)’ é invocada. Essa função é uma ferramenta poderosa do navegador que solicita o próximo quadro da animação, criando um loop contínuo. Isso não apenas proporciona uma animação suave, como também é eficiente em termos de recursos, pois o navegador otimiza a performance da animação, pausando-a em abas inativas. Este método de animação garante uma experiência de usuário fluida e responsiva, essencial para simulações interativas como a nossa.

Existem variações da função ‘animar’, projetadas para simular a gravidade e a expansão do Universo de maneira independente. Essas variações são adaptadas para invocar, conforme o necessário, as funções específicas que representam essas forças, permitindo assim uma simulação mais focada em cada aspecto individual.

A função ‘animar()’ utilizada é determinada pela função ‘iniciar(x)’, ativada a partir de três botões distintos. Cada botão passa um valor único para a variável ‘x’, permitindo assim o controle sobre o aspecto da simulação que será executado.

A função ‘iniciar(x)’, além de iniciar a função ‘animar()’, também tem a capacidade de reiniciá-la. Para isso, verifica-se se o ID de frame de animação já existe; se sim, a função cancela esse frame usando o comando ‘cancelAnimationFrame’, garantindo que a simulação anterior seja completamente interrompida antes de iniciar uma nova. Em seguida, o array ‘esferas’ é reiniciado, assegurando que cada simulação comece com um conjunto limpo de dados. A função prossegue criando as esferas com diferentes tamanhos e posições, todas posicionadas em relação ao centro do canvas. Essas esferas são criadas utilizando a função ‘criarEsfera’, o que termina por desenvolver a simulação completamente.

Concluindo o código, configuramos eventos específicos para cada botão, assegurando que cada um deles execute sua função designada corretamente.

```
function animar(tempoAtual) {
    if (!ultimoTempo) ultimoTempo = tempoAtual;
        const deltaTime = (tempoAtual - ultimoTempo) / 1000;
        Gravidade();
        expandirUniverso()
        Colisao();

        esferas.forEach(esfera => {
            esfera.x += esfera.vx * deltaTime;
            esfera.y += esfera.vy * deltaTime;
        });

        desenharFundo()
        desenharEsferas();
        ultimoTempo = tempoAtual;
        animationFrameId = requestAnimationFrame(animar);
}

function iniciar(x){
    if (animationFrameId) {
        cancelAnimationFrame(animationFrameId);
    }
    esferas = [];
    criarEsfera(centroX, centroY, 0, 0, 10);
    criarEsfera(centroX - 50, centroY + 275, 0, 0, 15);
    criarEsfera(centroX + 150, centroY + 275, 0, 0, 15);
    criarEsfera(centroX + 653, centroY - 180, 0, 0, 5);
    criarEsfera(centroX - 215, centroY + 318, 0, 0, 10);
    criarEsfera(centroX + 304, centroY - 779, 0, 0, 10);
    criarEsfera(centroX - 345, centroY - 454, 0, 0, 10);
    criarEsfera(centroX + 860, centroY - 379, 0, 0, 10);
    criarEsfera(centroX + 200, centroY, 0, 0, 15);

    if (x == 1) animationFrameId = requestAnimationFrame(animar1);
    if (x == 2) animationFrameId = requestAnimationFrame(animar2);
    if (x == 3) animationFrameId = requestAnimationFrame(animar);

}

document.getElementById('iniciar-simulacao').addEventListener('click', function() {iniciar(1);});
document.getElementById('iniciar-simulacao2').addEventListener('click', function() {iniciar(2);});
document.getElementById('iniciar-simulacao3').addEventListener('click', function() {iniciar(3);});
document.getElementById('aumentar-gravidade').addEventListener('click', function() {aumentarGravidade();});
document.getElementById('diminuir-gravidade').addEventListener('click',function() {diminuirGravidade();});
document.getElementById('aumentar-energia-escura').addEventListener('click', function() {aumentarEnergiaEscura(); ...});
document.getElementById('diminuir-energia-escura').addEventListener('click', function() {diminuirEnergiaEscura(); ...});
document.getElementById('zoom-in').addEventListener('click', aumentarZoom);
document.getElementById('zoom-out').addEventListener('click', diminuirZoom);
```

Figura 16. JS parte 08. Fonte: próprio autor.

5. Testes

Com a etapa de codificação concluída, avançamos para a próxima fase do ciclo de vida de desenvolvimento de software: a fase de testes. A simulação está operando conforme o esperado, com a gravidade dominando em distâncias menores e gerando órbitas complexas, enquanto a energia escura provoca a expansão acelerada em distâncias

maiores. Tudo isso é apresentado em um design simples, mas funcional. A seguir, apresentamos imagens que ilustram o estado atual do nosso site:

Figura 17. Apresentação. Fonte: próprio autor.

Figura 18. Espaço de simulação. Fonte: próprio autor.

6. Implantação

Para que os usuários possam usufruir do projeto, ele está disponível para acesso integralmente através do link do GitHub: https://github.com/vtrOsv/TGI-Astronomia-Interativa

O site está hospedado temporariamente no domínio:

https://cosmosemequilíbriogravidadeeenergiaescuradesvendadas.com

7. Manutenção e Suporte

Esta é a etapa final do ciclo de vida de desenvolvimento do software, que é a manutenção e o suporte regular. Nosso plano é continuar a evolução do projeto, mesmo após a publicação deste artigo. Nosso foco será aprimorar a ferramenta existente, incorporar novas funcionalidades e corrigir quaisquer inconsistências identificadas, a fim de melhorar ainda mais a experiência e a eficácia do software. A seguir, apresentamos alguns dos nossos planos futuros para esta iniciativa.

Aprimorar a usabilidade dos botões da simulação. Observamos que, atualmente, eles podem não ser totalmente intuitivos. Para resolver isso, pretendemos oferecer explicações mais claras no texto do site e trabalhar no aperfeiçoamento funcional dos botões.

Planejamos ajustar as constantes da simulação para otimizar a visualização. Atualmente, a representação da energia escura requer uma grande distância entre as esferas para ser visualizada corretamente. Vamos recalibrar esses valores para assegurar uma experiência de visualização mais clara.

Temos a intenção de desenvolver uma câmera que possa se fixar a uma esfera específica. Essa melhoria visa resolver um desafio de visualização atual: com a perspectiva fixa, a expansão do universo pode resultar em uma visualização atípica, onde as esferas saem do campo de visão devido ao seu afastamento progressivo uma em relação à outra.

Planejamos implementar uma funcionalidade que permitirá aos usuários adicionar esferas na simulação com um clique do mouse. Atualmente, as esferas inseridas por meio do código tendem a colidir, ejetar-se ou entrar em órbita, o que pode tornar a simulação monótona com o passar do tempo. Ao dar aos usuários o poder de interagir diretamente e adicionar esferas, esperamos enriquecer a experiência da simulação.

Por fim, visamos expandir o projeto com a criação de novas abas que apresentem estilos e lógicas diferentes. Um exemplo disso é a introdução de uma simulação tridimensional utilizando a biblioteca Three.js.

8. Considerações Finais

Como mencionado anteriormente, a astronomia vem enfrentando desafios em manter sua presença nas escolas por variadas razões. O foco deste trabalho foi desenvolver uma solução para esse problema. Iniciamos uma análise da literatura para deduzir a viabilidade e possível importância do nosso projeto em portais acadêmicos. Diversos artigos encontrados e aqui citados corroboram com a noção de que métodos não-tradicionais de ensino, como o uso de softwares de simulação, jogos ou exercícios gamificados, geram mais engajamento por parte dos alunos. Com isso, concluímos que uma solução para esse desafio é a mudança parcial da estratégia de ensino, normalizando o uso de ferramentas tecnológicas como softwares para complementar o ensino de astronomia.

A partir desses achados e da mobilização do repertório adquirido no curso de Ciência da Computação, foi criado um software de uso didático em JavaScript com foco na simulação do princípio da gravidade e da energia escura em ação, a ser usado por professores de física e ciências e seus alunos em idade escolar para a apreensão de fenômenos astronômicos, buscando tornar acessível a visualização e a internalização de eventos e regras do mundo físico – que podem ser, por vezes, abstratos e de difícil acesso. Nosso projeto, portanto, tem o potencial de ajudar nesse cenário.

Referências

Alcaniz, J. S. **Cosmologia: de Einstein à energia escura**. Com Ciência, 10 ago. 2007. Disponível em: https://www.comciencia.br/comciencia/handler.php?section=8&edicao=27&id=306. Acesso em 18 nov. 2023.

Becker, W. R.; Strieder, D. M. O uso de simuladores no ensino de astronomia. **Encontro nacional de informática e educação**, v. 2, p. 398, 2011. Disponível em: https://www.yumpu.com/pt/document/read/45328555/o-uso-de-simuladores-no- ensino-de-astronomia-inf-unioeste. Acesso em 06 jun. 2023.

Beserra, D. W. et al. Ensino de astronomia com os softwares Stellarium e Celestia. In: **Congresso Internacional de Tecnologia na Educação**, 10, 2012. Recife: Senac/DR/PR, 2012. Disponível em: https://www.researchgate.net/publication/303920019_ENSINO_DE_ASTRONOMIA_COM_OS_SOFTWARES_STELLARIUM_E_CELESTIA. Acesso em 06 jun. 2023.

Betz, E. **Nabta Playa: The world's first astronomical site was built in Africa and is older than Stonehenge**. Astronomy, 20 jun. 2020. Disponível em: https://astronomy.com/news/2020/06/nabta-playa-the-worlds-first-astronomical-site-was-built-in-africa-and-is-older-than-stonehenge. Acesso em 06 jun. 2023.

Brianeze, S. R.; Malacarne, V. A Astronomia no Ensino Fundamental e o uso do software JCLIC. In: **O professor PDE e os desafios da escola pública paranaense 2012**, v. 01, p. 30-46. Disponível em: http://www.diaadiaeducacao.pr.gov.br/portals/cadernospde/pdebusca/producoes_pde/2012/2012_unioeste_cien_artigo_silvana_regina_brianeze.pdf. Acesso em 06 jun. 2023.

Neres, L. B. **O Stellarium como estratégia para o ensino de astronomia**. Ilhéus (BA): Dissertação de Mestrado – Universidade Estadual de Santa Cruz, 2017. Disponível em: http://nbcgib.uesc.br/mnpef/images/Arquivos/Dissertao-para-divulgao-em-biblioteca-digital--leomir.pdf. Acesso em 06 jun. 2023.

Ribeiro, J. P. M. Filmes e softwares educacionais no ensino de Física: Uma análise bivariada. In: **Research, Society and Development**, v. 9, p. 36984998, 2020. Disponível em: https://www.researchgate.net/publication/342598040_Filmes_e_softwares_educacionais_no_ensino_de_Fisica_Uma_analise_bivariada. Acesso em 06 jun. 2023.