CosmoData

Colegio Santísima Trinidad de Salamanca

Sara González Cruz

Proyecto

Integrado (PI)

Tutor: Rafael Pérez Corro

Desarrollo de Aplicaciones Multiplataforma

Sara González Cruz

1. Estudio del problema y análisis del sistema

1.1. Introducción

El objetivo del proyecto “CosmoData” es crear una herramienta de consulta que acerque la inmensidad del espacio a cualquier persona de una forma rápida y accesible.

1.2. Finalidad

La finalidad del proyecto "CosmoData" es proporcionar a cualquier persona una herramienta de consulta que simplifique el acceso a información sobre el espacio de manera rápida y fácil. Se busca facilitar la comprensión y exploración del cosmos, permitiendo a individuos, independientemente de su nivel de conocimiento en astronomía, acceder a datos relevantes de manera accesible. La idea es acercar la vastedad del espacio de una manera que sea comprensible para todos, fomentando así el interés y la apreciación por el universo.

1.3. Objetivos

El proyecto "CosmoData" tiene como misión principal acercar la inmensidad del espacio a cada persona de manera rápida y accesible. Con este propósito en mente, se han delineado objetivos clave que sustentan este ambicioso proyecto. A continuación, se presenta una introducción a estos objetivos:

- **Usabilidad y Accesibilidad**

Interfaz Intuitiva: la interfaz de usuario intuitiva facilita la exploración del espacio, permitiendo que cualquier persona, incluso sin conocimientos previos, pueda disfrutar de la herramienta.

- **Experiencia Visual Inmersiva**

Visualización 3D: se han incorporado tecnologías de visualización tridimensional para ofrecer una experiencia inmersiva que permite a los usuarios explorar el espacio desde diferentes perspectivas.

- **Educación y personalización**

Contenido didáctico integrado: la página web ofrece secciones educativas que explican conceptos astronómicos de manera clara, haciendo que el aprendizaje sea parte integral de la experiencia.

2. Modelado de la solución

2.1. Recursos humanos

El proyecto "CosmoData" ha sido concebido y desarrollado en su totalidad por la autora, quien ha aplicado con destreza los conocimientos adquiridos tanto en sus estudios como en su tiempo

libre. En este contexto, los pilares fundamentales de su pericia se centran en los lenguajes de programación, JavaScript y HTML, así como en la hábil manipulación de datos mediante JSON.

Desde la fase inicial de conceptualización hasta la implementación final, la autora ha canalizado su experiencia y habilidades en estos lenguajes para dar forma a una herramienta de consulta que posibilita la exploración del espacio de manera veloz y accesible. La cuidadosa combinación de estos conocimientos ha resultado crucial en el desarrollo de una interfaz intuitiva y atrayente.

La autora, gracias a su autonomía y habilidades en JavaScript, HTML y JSON, ha desempeñado un papel central en la creación de una plataforma educativa. Esta no solo es personalizable según las preferencias individuales, sino que también está optimizada para proporcionar una experiencia completa a los usuarios interesados en sumergirse y comprender el cautivador mundo del espacio.

2.2. Recursos hardware

Durante el desarrollo del proyecto "CosmoData", se ha hecho un uso eficiente de los recursos hardware, centrándose principalmente en dos ordenadores distintos para garantizar una experiencia visual óptima en diversas pantallas. Los recursos hardware empleados se detallan a continuación:

- **Ordenador para Desarrollo**

Se utilizaron dos ordenadores dedicados al desarrollo y prueba del proyecto, cada uno con configuraciones que abarcan procesadores de última generación, suficiente memoria RAM y capacidades gráficas adecuadas para garantizar un rendimiento fluido durante la creación y la fase de prueba.

- **Conexión a internet**

La conexión a Internet desempeñó un papel fundamental en el desarrollo y la verificación de la herramienta. Se garantizó una conexión estable y de alta velocidad para facilitar la carga rápida de contenido, especialmente al integrar elementos visuales y actualizaciones en tiempo real.

Estos dos elementos clave, los ordenadores de desarrollo y la conexión a Internet, fueron esenciales para asegurar la compatibilidad y la experiencia de usuario óptima en diferentes dispositivos y pantallas. La elección cuidadosa de estos recursos ha contribuido a la eficiencia del desarrollo y la calidad del producto final.

2.3. Recursos software

El desarrollo del proyecto "CosmoData" se ha beneficiado significativamente de una cuidadosa selección de recursos software, destinados a optimizar la eficiencia, la colaboración y la integración de datos en la herramienta. A continuación, se detallan los recursos software clave empleados durante el proceso de desarrollo.

Desarrollo:

- GitHub (Integración de Control de Versiones):

Se utilizó como plataforma central para el control de versiones, permitiendo un seguimiento preciso de los cambios en el código fuente. La colaboración eficiente entre los desarrolladores se facilitó gracias a las funcionalidades de ramificación, fusión y la gestión centralizada del código.

- Visual Studio Code (Entorno de Desarrollo Integrado (IDE)):

Visual Studio Code fue el entorno principal de desarrollo, proporcionando una interfaz liviana y poderosa. Su extensa compatibilidad con lenguajes como Python, JavaScript y HTML facilitó la escritura de código, la depuración y la administración de extensiones para mejorar la productividad del desarrollo.

- API Ninjas (Integración de Datos):

La plataforma API Ninjas se utilizó para simplificar y agilizar la integración de diversas API relacionadas con la astronomía. Esta herramienta permite acceder y manipular datos astronómicos de manera eficiente, facilitando la actualización en tiempo real de la información disponible en el "Space Project".

- API Nasa (Integración de Datos):

La API de la NASA desempeñó un papel crucial en la obtención de datos auténticos y actualizados sobre el espacio. La integración de esta API permite acceder a una amplia variedad de información, desde imágenes de alta resolución hasta datos científicos verificados, enriqueciendo así la calidad y la autenticidad de la información presentada en la herramienta.

- Librería jQuery (http://ajax.googleapis.com/ajax/libs/jquery/1.8.1/jquery.min.js):

jQuery es una biblioteca de JavaScript que simplifica la manipulación del DOM, manejo de eventos, animaciones y peticiones AJAX en el desarrollo web. Utilizamos la versión 1.8.1 alojada en el CDN de Google para aprovechar su velocidad de carga, disponibilidad y confiabilidad.

- Librería PrefixFree (prefixfree.min.js):

PrefixFree es una biblioteca JavaScript que aborda la necesidad de agregar prefijos de proveedores a propiedades CSS. Su objetivo es simplificar el desarrollo al eliminar la necesidad de escribir manualmente prefijos para diferentes navegadores. Esta biblioteca puede mejorar la compatibilidad entre navegadores y facilitar el mantenimiento del código.

3. Planificación

3.1. Diseño del proyecto

Inicio del Proyecto: El proyecto "CosmoData" se inició con una fase crucial de planificación, centrada en la concepción de una aplicación que acercara la vastedad del espacio de manera rápida y accesible. La visión central del proyecto se basó en la comunicación e interacción eficiente con APIs (Interfaces de Programación de Aplicaciones) para obtener datos astronómicos auténticos y enriquecer la experiencia del usuario.

Enfoque en la Interacción con APIs: Desde el principio, se reconoció la importancia de las APIs en la obtención y gestión de datos relacionados con el espacio. La elección de APIs, como la proporcionada por la NASA, se fundamentó en la necesidad de acceder a información verídica y actualizada. Este enfoque permitió no solo ofrecer datos precisos, sino también asegurar la relevancia de la información presentada en tiempo real.

Simplicidad y Escalabilidad: Aunque la aplicación "Space Project" se concibió como una herramienta sencilla para acercar la astronomía a un público amplio, se tuvo en cuenta la escalabilidad y la adaptabilidad. Se reconoció que, al trabajar con casos de big data o operaciones que requieran recursos elevados en el futuro, la utilización de APIs proporciona una solución eficiente. Este enfoque permite que la aplicación crezca y evolucione fácilmente, respondiendo a las demandas cambiantes del usuario y manteniendo su utilidad en el tiempo.

Metodología de Desarrollo: Durante la planificación, se adoptó una metodología de desarrollo iterativa y ágil. Se establecieron hitos para la integración de nuevas funcionalidades y se programaron revisiones periódicas para evaluar el progreso y realizar ajustes según las necesidades emergentes.

Consideración de Recursos: La elección de trabajar con APIs también influyó en la consideración de recursos, ya que se buscó optimizar la eficiencia y minimizar la carga de datos. Esta decisión estratégica permitió una experiencia de usuario fluida y rápida, incluso en el manejo de información en tiempo real y actualizaciones frecuentes.

Estrategia: Integrar un juego de ‘Space Invaders’ en una página que busca fomentar el interés por el espacio ofrece múltiples beneficios. La interactividad y el entretenimiento atraen a un público más amplio, incluidos aquellos que pueden no tener un interés inicial en la astronomía. Además, la combinación de diversión y elementos educativos puede convertir el juego en una herramienta efectiva para el aprendizaje. Al proporcionar una experiencia inmersiva y atractiva, el juego puede generar curiosidad, motivar a los usuarios a explorar más sobre el espacio y difundir el mensaje principal de la página de manera memorable. En última instancia, la inclusión de un juego contribuye a hacer la información más accesible y atractiva para un público diverso, cumpliendo así el objetivo de acercar el espacio a todas las personas.

Arquitectura:

Mantener una buena arquitectura en un proyecto de frontend es crucial para su éxito a largo plazo. Una estructura bien diseñada facilita la escalabilidad, el mantenimiento, la colaboración entre equipos, la reusabilidad del código y la adaptabilidad a cambios en los requisitos. Además,

contribuye a un rendimiento óptimo, mejora la experiencia del usuario y permite una evolución continua del proyecto. Una arquitectura sólida proporciona una base estructurada que no solo agiliza el desarrollo inicial, sino que también asegura la sostenibilidad y eficiencia en el tiempo, crucial para el crecimiento y la evolución exitosa de la aplicación.

Análisis de la arquitectura:

Explicar cada archivo en un proyecto, ya sea HTML, CSS, JavaScript u otros, es esencial para garantizar la comprensión, colaboración y mantenimiento eficiente del código. Proporciona una documentación integrada que sirve como guía para desarrolladores actuales y futuros, facilita el aprendizaje, agiliza la resolución de problemas y mejora la revisión de código. La transparencia y comunicación dentro del equipo se fortalecen, permitiendo una alineación con las mejores prácticas de desarrollo y contribuyendo a un desarrollo de software más efectivo y sostenible a lo largo del tiempo.

GetApiNinjasData.html

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <script src="https://code.jquery.com/jquery-3.6.4.min.js"></script>      You, 3 weeks ago • Subiendo proyecto
    <link rel=stylesheet href="CSS/nav.css">
    <link rel=stylesheet href="CSS/infoApiNinjas.css">
    <title>Data</title>
</head>
<body>
    <header>
        <ul class="menu-bar">
            <li><a href="index.html">Inicio</a></li>
            <li><a href="planets.html">Planetario</a></li>
            <li><a href="choosePlanet.html">Datos</a></li>
            <li><a href="imgNASA.html">Imagen del día</a></li>
            <li><a href="games.html">Juegos y Entretenimiento</a></li>
            <li><a href="contact.html">Contacto</a></li>
        </ul>
    </header>
    <h1>Planet Information</h1>
    <!-- Botones para cada planeta -->
    <div class="planets-bar">
        <ul>
            <li onclick="getPlanetInfo('Mercury')">Mercury</li>
            <li onclick="getPlanetInfo('Venus')">Venus</li>
            <li onclick="getPlanetInfo('Earth')">Earth</li>
            <li onclick="getPlanetInfo('Mars')">Mars</li>
            <li onclick="getPlanetInfo('Jupiter')">Jupiter</li>
```

```html
    <h1>Planet Information</h1>
    <!-- Botones para cada planeta -->
    <div class="planets-bar">
        <ul>
            <li onclick="getPlanetInfo('Mercury')">Mercury</li>
            <li onclick="getPlanetInfo('Venus')">Venus</li>
            <li onclick="getPlanetInfo('Earth')">Earth</li>
            <li onclick="getPlanetInfo('Mars')">Mars</li>
            <li onclick="getPlanetInfo('Jupiter')">Jupiter</li>
            <li onclick="getPlanetInfo('Saturn')">Saturn</li>
            <li onclick="getPlanetInfo('Uranus')">Uranus</li>
            <li onclick="getPlanetInfo('Neptune')">Neptune</li>
        </ul>
    </div>


    <!-- Div para mostrar la información -->
    <div id="planetInfo" class="info-box"></div>

    <script src="JS/apiJS.js"></script>
        <!--HTML planeta girando-->
        <div id="planet"></div>
      <!--FOOTER-->
      <footer>
        <p>&copy; 2023 Sara González Cruz</p>
    </footer>
</body>
</html>
```

Lo fundamental de este código es que el div se utiliza para mostrar la información del planeta seleccionado. Se importa un script JavaScript para manejar la obtención de datos y se incluye un div adicional que podría representar visualmente un planeta giratorio. El pie de página contiene información de copyright y autoría. El HTML proporciona una interfaz interactiva para explorar datos planetarios, integrando funcionalidades informativas y visuales.

ApiJS.js

Clave API:

```
var apiKey = 'S2v+32RLvV/f4Ks5qm/1/w==eFgI1A6inHEInuds';
```

Se define una clave de API para autenticar las solicitudes a la API-ninjas.

a) **Función para tener información del planeta:**

```
// Función para realizar la solicitud a la API
function getPlanetInfo(planetName) {
    $.ajax({
        method: 'GET',
        url: 'https://api.api-ninjas.com/v1/planets?name=' + planetName,
        headers: { 'X-Api-Key': apiKey },
        contentType: 'application/json',
        success: function(result) {
            // Muestra la información en el div "planetInfo"
            displayPlanetInfo(result[0]);
        },
        error: function ajaxError(jqXHR) {
            console.error('Error: ', jqXHR.responseText);
        }
    });      You, 4 weeks ago • Subiendo proyecto
}
```

Esta función realiza una solicitud GET a la API-ninjas para obtener información sobre un planeta específico. La URL de la solicitud incluye el nombre del planeta como un parámetro de consulta, y se agrega la clave de API a través de los encabezados.

b) **Función para mostrar información del planeta:**

```
function displayPlanetInfo(planet) {
    var infoDiv = document.getElementById('planetInfo');
    infoDiv.innerHTML = ''; // Limpia el contenido anterior

    // Crea elementos de texto para mostrar la información
    var name = document.createElement('p');
    var mass = document.createElement('p');
    var radius = document.createElement('p');
    var period = document.createElement('p');
    var semiMajorAxis = document.createElement('p');
    var temperature = document.createElement('p');
    var distanceLightYear = document.createElement('p');

    // Asigna el texto correspondiente a cada elemento
    name.textContent = 'Name: ' + planet.name;
    mass.textContent = 'Mass: ' + planet.mass;
    radius.textContent = 'Radius: ' + planet.radius;
    period.textContent = 'Period: ' + planet.period;
    semiMajorAxis.textContent = 'Semi-Major Axis: ' + planet.semi_major_axis;
    temperature.textContent = 'Temperature: ' + planet.temperature;
    distanceLightYear.textContent = 'Distance (light years): ' + planet.distance_light_year;

    // Agrega los elementos al div "planetInfo"
    infoDiv.appendChild(name);
    infoDiv.appendChild(mass);
    infoDiv.appendChild(radius);
    infoDiv.appendChild(period);
    infoDiv.appendChild(semiMajorAxis);
    infoDiv.appendChild(temperature);
    infoDiv.appendChild(distanceLightYear);
```

```
You, 3 seconds ago | 1 author (You)
var apiKey = 'S2v+32RLvV/f4Ks5qm/1/w==eFgI1A6inHEInuds';        You, now • Uncommitted changes
// Función para realizar la solicitud a la API
function getPlanetInfo(planetName) {
    $.ajax({
        method: 'GET',
        url: 'https://api.api-ninjas.com/v1/planets?name=' + planetName,
        headers: { 'X-Api-Key': apiKey },
        contentType: 'application/json',
        success: function(result) {
            // Muestra la información en el div "planetInfo"
            displayPlanetInfo(result[0]);
        },
        error: function ajaxError(jqXHR) {
            console.error('Error: ', jqXHR.responseText);
        }
    });
}

// Función para mostrar la información del planeta en el documento
function displayPlanetInfo(planet) {
    var infoDiv = document.getElementById('planetInfo');
    infoDiv.innerHTML = ''; // Limpia el contenido anterior
```

```
    var infoDiv = document.getElementById('planetInfo');
    infoDiv.innerHTML = ''; // Limpia el contenido anterior

    // Crea elementos de texto para mostrar la información
    var name = document.createElement('p');
    var mass = document.createElement('p');
    var radius = document.createElement('p');
    var period = document.createElement('p');
    var semiMajorAxis = document.createElement('p');
    var temperature = document.createElement('p');
    var distanceLightYear = document.createElement('p');

    // Asigna el texto correspondiente a cada elemento
    name.textContent = 'Name: ' + planet.name;
    mass.textContent = 'Mass: ' + planet.mass;
    radius.textContent = 'Radius: ' + planet.radius;
    period.textContent = 'Period: ' + planet.period;
    semiMajorAxis.textContent = 'Semi-Major Axis: ' + planet.semi_major_axis;
    temperature.textContent = 'Temperature: ' + planet.temperature;
    distanceLightYear.textContent = 'Distance (light years): ' + planet.distance_light_year;

    // Agrega los elementos al div "planetInfo"
    infoDiv.appendChild(name);
    infoDiv.appendChild(mass);
    infoDiv.appendChild(radius);
    infoDiv.appendChild(period);
    infoDiv.appendChild(semiMajorAxis);
    infoDiv.appendChild(temperature);
    infoDiv.appendChild(distanceLightYear);

    // Crea un elemento de imagen
    var planetImage = document.createElement('img');
```

Esta función toma el objeto de información del planeta devuelto por la API y lo muestra en el documento. Crea elementos de texto para detalles como nombre, masa, radio, período, etc.

```
    // Crea un elemento de imagen
    var planetImage = document.createElement('img');

    // Asigna la ruta de la imagen según el nombre del planeta
    switch (planet.name.toLowerCase()) {
        case 'mercury':
            planetImage.src = '../IMG/mercury.png';
            planetImage.classList.add('planet', 'mercury');
            break;
        case 'venus':
            planetImage.src = '../IMG/venus.png';
            planetImage.classList.add('planet', 'venus');
            break;
        case 'earth':
            planetImage.src = '../IMG/earth.png';
            planetImage.classList.add('planet', 'earth');
            break;
        case 'mars':
            planetImage.src = '../IMG/mars.png';
            planetImage.classList.add('planet', 'mars');
            break;
        case 'jupiter':
            planetImage.src = '../IMG/jupiter.png';
            planetImage.classList.add('planet', 'jupiter');
            break;
        case 'saturn':
            planetImage.src = '../IMG/saturn.png';
            planetImage.classList.add('planet', 'saturn');
            break;
        case 'uranus':
            planetImage.src = '../IMG/uranus.png';
```

```
    // Asigna la ruta de la imagen según el nombre del planeta
    switch (planet.name.toLowerCase()) {···
    }

    // Agrega la imagen al div "planetInfo"
    infoDiv.appendChild(planetImage);
}
```

Además, agrega una imagen correspondiente al planeta en función de su nombre.

Así es como funciona. La a función getPlanetInfo(planetName) realiza solicitudes a la API mediante AJAX, incluyendo la clave de API para autenticación. Cuando la solicitud es exitosa, la función displayPlanetInfo(planet) presenta la información del planeta en el documento HTML. Se generan elementos de texto para detalles como nombre, masa, radio y período, mientras que una imagen correspondiente al planeta se agrega dinámicamente en función de su nombre. Este enfoque permite mostrar de manera interactiva y visual la información planetaria en el proyecto, proporcionando una experiencia informativa y atractiva para los usuarios interesados en la exploración del espacio.

API.js (NASA)

a) Uso de EventListener y función getData:

```
window.addEventListener("load", getData);

function getData() {
    const NASA_KEY = 'ge4KOkXIMyXnEuQGqSkRYqSTWZPSt66EZrhvl7IE';
    const PATH = `https://api.nasa.gov/planetary/apod?api_key=${NASA_KEY}`;

    fetch(PATH)
        .then(response => response.json())
        .then(result => {
            console.log(result); // Agrega este log para depuración
            showData(result);
        })
        .catch(error => console.error('Error fetching data:', error));
}
```

- Se añade un event listener para que cuando la ventana se cargue completamente, se ejecute la función getData.
- La función getData contiene la lógica para hacer una solicitud a la API de la NASA usando la clave proporcionada.
- La URL de la solicitud incluye la clave de la API y se utiliza la función fetch para obtener los datos. Se manejan las promesas resultantes.

b) Función showData

```
function showData({ date, explanation, media_type, title, url }) {
    console.log('showData called');

    const TITLE = document.querySelector('#title');
    TITLE.innerHTML = title;
    const DATE = document.querySelector('#date');
    DATE.innerHTML = date;
    const EXPLANATION = document.querySelector('#description');
    EXPLANATION.innerHTML = explanation;

    const MULTIMEDIA = document.querySelector('#c_multimedia');
    MULTIMEDIA.innerHTML = ''; // Limpiamos el contenido anterior

    // Validamos si es imagen o video
    if (media_type === 'video') {
        const iframe = document.createElement('iframe');
        iframe.className = 'embed-responsive-item';
        iframe.src = url;
        MULTIMEDIA.appendChild(iframe);
    } else {
        const img = document.createElement('img');
        img.src = url;
        img.className = 'img-fluid';
        img.alt = url;
        MULTIMEDIA.appendChild(img);
    }
}
```

- La función showData recibe un objeto destructurado con propiedades como date, explanation, media_type, title, y url.
- Se actualizan dinámicamente varios elementos HTML en la página con la información proporcionada.
- Se utiliza querySelector para seleccionar elementos por su ID y se actualizan sus contenidos.
- Se gestiona el contenido multimedia: si es un video, se crea un elemento iframe; si es una imagen, se crea un elemento img. Estos elementos se añaden al contenedor multimedia (#c_multimedia).

Estas funciones de JavaScript se conecta a la API de la NASA (Astronomy Picture of the Day - APOD) para recuperar información sobre la imagen o video astronómico destacado del día. Al cargar la ventana, la función getData realiza una solicitud fetch a la URL de la API de la NASA, que incluye una clave de API para la autenticación. Una vez que la respuesta es recibida, la función showData procesa la información JSON devuelta.
La función showData actualiza dinámicamente los elementos HTML en la página para mostrar detalles sobre la imagen o video, como el título, la fecha y una descripción. Además, verifica si el contenido multimedia es un video o una imagen y lo muestra en consecuencia. Si es un video, se crea un elemento iframe que se inserta en el contenedor multimedia (#c_multimedia). Si es una imagen, se crea un elemento img que se coloca en el mismo contenedor.

ImgNASA.html

a) Mostrar resultados:

```
    <div id="c_multimedia" class="embed-responsive b1by9 text-center c-media-container"></div>
</div>
<div class="multi">
    <script src="JS/app.js"></script>
</div>
```

Los resultados se muestran haciendo referencia al script en el HTML de nuestra página.

Planets.html (JS que maneja el HTML)

a) Inicialización de las variables

```
var e = $("body"),
    t = $("#universe"),
    n = $("#solar-system"),
```

Inicializamos variables para representar los elementos del cuerpo del documento (e), el universo (t), y el sistema solar (n). Estas variables se utilizan para referenciar elementos del DOM en el documento HTML.

b) Función r:

```
    r = function() {
      e.removeClass("").addClass("view-3D").delay(2000).queue(function(){
        $(this).removeClass("hide-UI").addClass("set-speed");
        $(this).dequeue();
      });
    },
    i = function(e) {
      t.removeClass().addClass(e);
    };
```

Esta función realiza una serie de acciones:

- Remueve las clases "view-2D" y "opening" del elemento con id "body" y agrega las clases "view-3D".
- Después de un retraso de 2000 milisegundos (2 segundos):
- Remueve la clase "hide-UI" y agrega la clase "set-speed".
- Usa dequeue() para indicar que la cola de funciones debe avanzar.

c) Función i(e):

```
};
i = function(e) {
  t.removeClass().addClass(e);
};
```

Esta función toma un parámetro e y modifica las clases del elemento con id "universe" para que reflejen la clase proporcionada.

d) Manejo de clics en el botón Toggle Data:

```
// Manejo de clics en el botón "toggle-data"
$("#toggle-data").click(function(t) {
  e.toggleClass("data-open data-close");
  t.preventDefault();
});
```

Cuando se hace clic en el botón con id "toggle-data", alterna las clases "data-open" y "data-close" en el elemento del cuerpo del documento. La llamada a preventDefault() evita que el comportamiento predeterminado del enlace (si es un enlace) se ejecute.

e) Manejo de clics en enlaces dentro de #data:

```
// Manejo de clics en enlaces dentro de #data
$("#data a").click(function(e) {
  var t = $(this).attr("class");
  n.removeClass().addClass(t);
  $(this).parent().find("a").removeClass("active");
  $(this).addClass("active");
  e.preventDefault();
});
```

Cuando se hace clic en un enlace dentro de un elemento con id "data", se realizan las siguientes acciones:

- Se obtiene la clase del enlace clicado (t).
- Se alteran las clases del elemento con id "solar-system" para reflejar la clase obtenida.
- Se manipula la clase "active" en los enlaces dentro del mismo contenedor para reflejar la selección.
- Se evita que el enlace ejecute su comportamiento predeterminado.

f) Manejo de Clics en Elementos con Clases "set-speed", "set-size", "set-distance":

```
// Manejo de clics en elementos con la clase "set-speed"
$(".set-speed").click(function() {
  i("scale-stretched set-speed");
});

// Manejo de clics en elementos con la clase "set-size"
$(".set-size").click(function() {
  i("scale-s set-size");
});

// Manejo de clics en elementos con la clase "set-distance"
$(".set-distance").click(function() {
  i("scale-d set-distance");
});
```

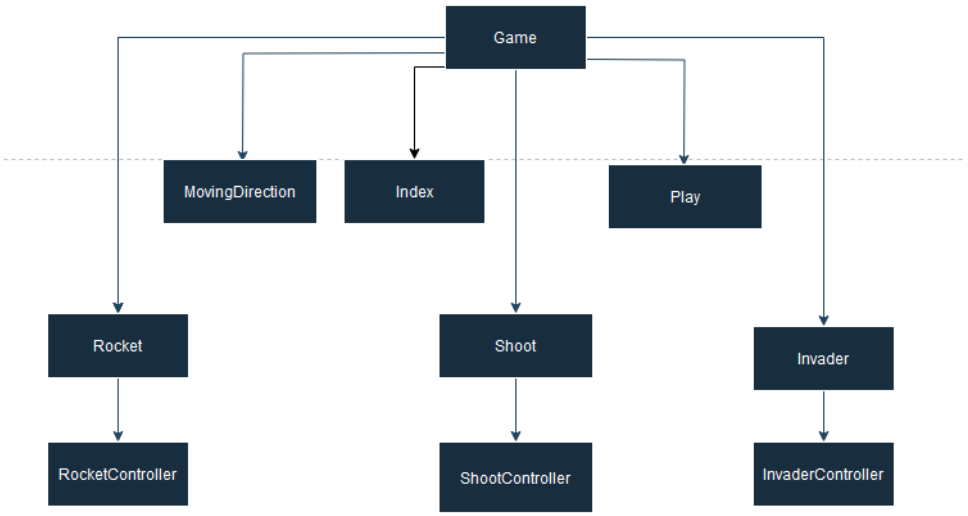
Cuando se hace clic en elementos con clases "set-speed", "set-size", o "set-distance", se llama a la función i con una cadena específica, lo que afecta las clases del elemento con id "universe".

g) Ejecución de r cuando cargamos la ventana:

```
// Ejecutar la función r() al cargar la ventana
r();
```

La función r() se ejecuta inmediatamente cuando la ventana se carga, lo que inicia el proceso de visualización y animación.

Game Arquitectura



Game HTML

```
<h1>Galaxy Raiders</h1>
<canvas id="game"></canvas>

<script type="module" src="./Game/play.js"></script>
```

a) Canvas:

El canvas es lo que proporciona el lienzo en el cual se realizarán las representaciones gráficas y animaciones asociadas al juego. Este enfoque se alinea con las prácticas modernas de desarrollo web, aprovechando las características avanzadas del estándar ECMAScript.

b) Llamada al JavaScript:

Hacemos la carga de un módulo JavaScript externo, play.js, que reside en el directorio ./Game/. La utilización de módulos en JavaScript permite la encapsulación de funcionalidades, facilitando la organización y comprensión del código. Esto contribuye a la reusabilidad y mantenimiento, ya que cada módulo puede concentrarse en una parte específica del juego.

c) Tipo module:

La elección de type="module" indica la adopción de la sintaxis y características de módulos de ES6, proporcionando beneficios tales como privacidad de variables, importación/exportación de funciones y clases, y la posibilidad de organizar el código en una estructura modular.

Game JavaScript

Clase Play.js

Es la clase que se va a encargar de llamar al juego desde el HTML.

a) Importación de modulos:

```
import InvaderController from "./InvaderController.js";
import Rocket from "./Rocket.js";
import ShootController from "./ShootController.js";
const canvas = document.getElementById("game");
const ctx = canvas.getContext("2d");
```

Aquí, se importan las clases InvaderController, Rocket, y ShootController desde módulos externos. Estas clases contienen la lógica para controlar los invasores, la nave del jugador y el control de disparos, respectivamente.

b) Configuración del canvas:

```
const canvas = document.getElementById("game");
const ctx = canvas.getContext("2d");

canvas.width = 600;
canvas.height = 600;
```

Se obtiene el elemento canvas del DOM y se establecen sus dimensiones. Luego, se obtiene el contexto 2D para realizar dibujos en el canvas.

c) Carga:

```
// FONDO
const background = new Image();       You, 6 days ago • add: changes
background.src = "Game/assets/game_background.jpg";
const gameSound = new Audio("Game/sounds/suits-you.mp3");
```

Cargamos el fondo del juego, tanto el sonido como la imagen de fondo.

d) Creamos los controladores y los objetos del juego:

```
const rocketShootController = new ShootController(canvas, 10, "red", true);
const invaderShootController = new ShootController(canvas, 4, "blue", false);
const invaderController = new InvaderController(
  canvas,
  invaderShootController,
  rocketShootController
);
const rocket = new Rocket(canvas, 3, rocketShootController, 5); // Pasar 5 como initialLives
```

Se crean instancias de los controladores y objetos del juego, como los controladores de disparos, el controlador de invasores y la nave del jugador.

e) Función game:

```
function game() {
  if (isGameOver) {
    resetLives();
    isGameOver = false;
  }
  checkGameOver();

  ctx.drawImage(background, 0, 0, canvas.width, canvas.height);
  displayGameOver();
  if (!isGameOver) {
    gameSound.play(3);
    invaderController.draw(ctx);
    rocket.draw(ctx);
    rocketShootController.draw(ctx);
    invaderShootController.draw(ctx);
  }
}
```

Es la lógica principal del juego, se ejecuta en un bucle (setInterval) y contiene la lógica principal del juego, como la actualización de los elementos del juego y la verificación de condiciones de fin de juego.

f) Función displayGameOver:

```
function displayGameOver() {
  if (isGameOver) {
    let image = didWin ? null : gameOverImage;

    if (didWin) {
      ctx.fillStyle = "white";
      ctx.font = "70px Arial";
      ctx.fillText("You Win", canvas.width / 3.5, canvas.height / 2);
    } else {
      const imageWidth = canvas.width / 2.2;
      const imageHeight = canvas.height / 2;
      const x = (canvas.width - imageWidth) / 2;
      const y = canvas.height / 4;

      ctx.drawImage(image, x, y, imageWidth, imageHeight);

      gameOverSound.currentTime = 0;
      gameOverSound.play();
    }
  }
}
```

Esta función maneja la lógica para mostrar la pantalla de Game Over, incluyendo mensajes de victoria o derrota y la reproducción de sonidos asociados.

g) Función resetLives:

```
function resetLives() {
  rocket.resetLives(); //
}
```

Restablece las vidas de la nave del jugador utilizando el método resetLives de la clase Rocket.

h) Función checkGameOver:

```
function checkGameOver() {
  if (rocket.isGameOver) {
    isGameOver = false;
    rocket.resetLives(); // Restablece las vidas del jugador
    invaderController.resetInvaders(); // Crea nuevos enemigos
    return;
  }

  if (invaderShootController.collideWith(rocket)) {

    rocket.decrementLives();
    if (rocket.lives === 0) {
      rocket.isGameOver = true;
    }
  }

  if (invaderController.collideWith(rocket)) {
    rocket.decrementLives();
    if (rocket.lives === 0) {
      rocket.isGameOver = true;
    }
  }

  if (invaderController.invaderRows.length === 0) {
    didWin = true;
    rocket.isGameOver = true;
  }
}
```

Esta función verifica las condiciones que determinan si el juego ha terminado, ya sea por derrota del jugador, colisiones con invasores o victoria al eliminar todos los invasores.

i) Configuración del bucle principal :

```
setInterval(game, 1000 / 60);
```

Establece un bucle principal que ejecuta la función game aproximadamente 60 veces por segundo, actualizando así el estado del juego y renderizando los cambios en el canvas.

Clase MovingDirection.js

En esta clase utilizamos un objeto literal para representar una enumeración de direcciones de movimiento. Cada dirección tiene un valor asociado que se asigna como propiedad del objeto MovingDirection.

a) Objeto MovingDirection:

```
const MovingDirection = {
    left: 0,
    right: 1,
    downLeft: 2,
    downRight: 3,
};
```

MovingDirection es un objeto literal que actúa como una enumeración. Cada propiedad del objeto representa una dirección de movimiento, y el valor asociado a cada propiedad es un número que sirve como identificador único para esa dirección.

b) Valores para las direcciones:

- left: 0 - Representa la dirección hacia la izquierda.
- right: 1 - Representa la dirección hacia la derecha.
- downLeft: 2 - Representa la dirección hacia abajo y a la izquierda.
- downRight: 3 - Representa la dirección hacia abajo y a la derecha.

c) Exportación del objeto:

```
export default MovingDirection;
```

De esta manera estamos exportando el objeto MovingDirection para que pueda ser importado en otros archivos de JavaScript. Al hacerlo, otros módulos pueden utilizar este objeto como una forma de acceder fácilmente a las direcciones de movimiento en el código.

Clase Invader,js

Esta clase es la encargada de representar al enemigo.

a) Constructor:

```
constructor(x, y, imageNumber) {
  this.x = x;
  this.y = y;
  this.width = 44;
  this.height = 32;

  this.image = new Image();
  this.image.src = `game/assets/enemy${imageNumber}.png`;
}
```

- El constructor toma tres parámetros: x y y representan las coordenadas del enemigo, y imageNumber se utiliza para cargar la imagen correspondiente del enemigo basándose en el patrón de nombres de archivos.
- this.x y this.y representan las coordenadas x e y del enemigo.
- this.width y this.height representan el ancho y alto del enemigo.
- Se crea una instancia de la clase Image y se asigna a this.image. La imagen se carga dinámicamente con el nombre basado en imageNumber.

b) Método draw:

```
draw(ctx) {
  ctx.drawImage(this.image, this.x, this.y, this.width, this.height);
}
```

Este método utiliza el contexto de dibujo ctx para dibujar la imagen del enemigo en las coordenadas x e y con el ancho y alto especificados.

c) Método move:

```
move(xVelocity, yVelocity) {
  this.x += xVelocity;
  this.y += yVelocity;
}
```

Este método se utiliza para cambiar las coordenadas x e y del enemigo según las velocidades proporcionadas en xVelocity e yVelocity.

d) Método collideWith:

```
collideWith(sprite) {
  if (
    this.x + this.width > sprite.x &&
    this.x < sprite.x + sprite.width &&
    this.y + this.height > sprite.y &&
    this.y < sprite.y + sprite.height
  ) {
    return true;
  } else {
    return false;
  }
}
```

Este método verifica si el enemigo colisiona con otro sprite, como un proyectil o un jugador. Devuelve true si hay una colisión, de lo contrario, devuelve false.

e) Método resetPosition:

```
resetPosition() {
  this.x = this.initialX;
  this.y = this.initialY;
}
```

Este método restablece la posición del enemigo a sus coordenadas iniciales (initialX e initialY). Sin embargo, en el código proporcionado, parece que faltan las propiedades initialX e initialY.

f) Método resetInvaders:

```
resetInvaders() {
  this.invaderRows.forEach((row) => {
    row.forEach((invader) => {
      invader.resetPosition();
    });
  });
}
```

Este método se utiliza para restablecer la posición de todos los invasores en un conjunto de filas (invaderRows). Cada invasor en cada fila tiene su posición restablecida llamando al método resetPosition().

Clase invaderController.js

Esta clase es la que se va a encargar de controlar la lógica de la clase Invader.

a) Propiedades:

```
invaderMap = [
  [1, 1, 1, 1, 1, 1, 1, 1, 1, 1],
  // ...
];
invaderRows = [];
currentDirection = MovingDirection.right;
xVelocity = 0;
yVelocity = 0;
defaultXVelocity = 1;
defaultYVelocity = 1;
moveDownTimerDefault = 30;
moveDownTimer = this.moveDownTimerDefault;
fireBulletTimerDefault = 100;
fireBulletTimer = this.fireBulletTimerDefault;
```

- invaderMap: Una matriz que representa la disposición inicial de los invasores en el juego.
- invaderRows: Un array que contiene las filas de invasores en el juego.
- currentDirection: La dirección actual de movimiento de los invasores.
- xVelocity y yVelocity: Velocidades actuales en los ejes x e y de los invasores.
- defaultXVelocity y defaultYVelocity: Velocidades predeterminadas de los invasores.
- moveDownTimerDefault y moveDownTimer: Temporizadores para controlar cuándo los invasores deben moverse hacia abajo.
- fireBulletTimerDefault y fireBulletTimer: Temporizadores para controlar la frecuencia de los disparos de los invasores.

➢ Más sobre invaderMap:

```
invaderMap = [
  [1, 1, 1, 1, 1, 1, 1, 1, 1, 1],
  [1, 1, 1, 1, 1, 1, 1, 1, 1, 1],
  [2, 2, 2, 3, 3, 3, 3, 2, 2, 2],
  [2, 2, 2, 3, 3, 3, 3, 2, 2, 2],
  [1, 1, 1, 1, 1, 1, 1, 1, 1, 1],
  [2, 2, 2, 2, 2, 2, 2, 2, 2, 2],
];
invaderRows = [];
```

Los números 1, 2 y 3 representan diferentes tipos de invasores.

Por otro lado, invaderRows es un array que contendrá las filas de invasores después de que se hayan creado y organizado según la configuración proporcionada en invaderMap. Este array se inicializa vacío al principio y se llena durante la creación de los invasores. Después de la

creación de los invasores, este array contendrá arrays de instancias de la clase Invader, organizados en filas según invaderMap.

b) Constructores:

```
constructor(canvas, invaderShootController, rocketShootController) {
  this.canvas = canvas;
  this.invaderShootController = invaderShootController;
  this.rocketShootController = rocketShootController;

  this.invaderDeathSound = new Audio("game/sounds/enemy-death.wav");
  this.invaderDeathSound.volume = 0.1;

  this.rocketDeathSound = new Audio("/Game/sounds/hurt.mp3");
  this.rocketDeathSound.volume = 0.1;

  this.createInvaders();
}
```

- El constructor toma tres parámetros: canvas (el lienzo del juego), invaderShootController (controlador de disparo de los invasores), y rocketShootController (controlador de disparo del jugador).
- Inicializa instancias de audio para sonidos de muerte de invasores y del jugador.
- Llama al método createInvaders para inicializar y colocar a los invasores.

c) Método draw:

```
draw(ctx) {
  this.decrementMoveDownTimer();
  this.updateVelocityAndDirection();
  this.collisionDetection();
  this.drawInvaders(ctx);
  this.resetMoveDownTimer();
  this.fireBullet();
}
```

Draw, se encarga de coordinar la lógica del juego, decrementa los temporizadores, actualiza la dirección y velocidad, realiza detección de colisiones, dibuja a los invasores y dispara balas.

d) Método collisionDetection:

```
collisionDetection() {
  this.invaderRows.forEach((invaderRow) => {
    invaderRow.forEach((invader, invaderIndex) => {
      if (this.rocketShootController.collideWith(invader)) {
        this.invaderDeathSound.currentTime = 0;
        this.invaderDeathSound.play();
        invaderRow.splice(invaderIndex, 1);
      }
    });
  });

  this.invaderRows = this.invaderRows.filter((invaderRow) => invaderRow.length > 0);
}
```

Este método se encarga de verificar las colisiones entre balas e invasores. Elimina invasores muertos y actualiza invaderRows.

e) Metodo fireBullet:

```
fireBullet() {
  this.fireBulletTimer--;
  if (this.fireBulletTimer <= 0) {
    this.fireBulletTimer = this.fireBulletTimerDefault;
    const allInvaders = this.invaderRows.flat();
    const invaderIndex = Math.floor(Math.random() * allInvaders.length);
    const invader = allInvaders[invaderIndex];
    this.invaderShootController.shoot(invader.x + invader.width / 2, invader.y, -3);
  }
}
```

Este método contiene la lógica para controlar la velocidad a la que los invasores disparan y la frecuencia de dichos disparos.

f) Métodos resetMoveDownTimer y decrementMoveDownTimer:

```
resetMoveDownTimer() {
  if (this.moveDownTimer <= 0) {
    this.moveDownTimer = this.moveDownTimerDefault;
  }
}

decrementMoveDownTimer() {
  if (
    this.currentDirection === MovingDirection.downLeft ||
    this.currentDirection === MovingDirection.downRight
  ) {
    this.moveDownTimer--;
  }
}
```

ResetMoveDownTimer se encarga de reiniciar el temporizador en el caso de que haya llegado a 0.

DecrementDownTimer se encarga de de decrementar el temporizador si los invasores se están moviendo hacia abajo.

g) Método updateVelocityAndDirection:

```
updateVelocityAndDirection() {
  for (const invaderRow of this.invaderRows) {
    if (this.currentDirection == MovingDirection.right) {
      this.xVelocity = this.defaultXVelocity;
      this.yVelocity = 0;
      const rightMostInvader = invaderRow[invaderRow.length - 1];
      if (rightMostInvader.x + rightMostInvader.width >= this.canvas.width) {
        this.currentDirection = MovingDirection.downLeft;
        break;
      }
    } else if (this.currentDirection === MovingDirection.downLeft) {
      if (this.moveDown(MovingDirection.left)) {
        break;
      }
    } else if (this.currentDirection === MovingDirection.left) {
      this.xVelocity = -this.defaultXVelocity;
      this.yVelocity = 0;
      const leftMostInvader = invaderRow[0];
      if (leftMostInvader.x <= 0) {
        this.currentDirection = MovingDirection.downRight;
        break;
      }
    } else if (this.currentDirection === MovingDirection.downRight) {
      if (this.moveDown(MovingDirection.right)) {
        break;
      }
    }
  }
}
```

Este método se encarga de actualizar la velocidad y dirección de los invasores según su posición actual y la posición de los invasores extremos.

h) Método moveDown:

```
moveDown(newDirection) {
  this.xVelocity = 0;
  this.yVelocity = this.defaultYVelocity;
  if (this.moveDownTimer <= 0) {
    this.currentDirection = newDirection;
    return true;
  }
  return false;
}
```

Lógica para mover a los invasores hacia abajo y cambiar la dirección de movimiento.

i) Método drawInvaders:

```
drawInvaders(ctx) {
  this.invaderRows.flat().forEach((invader) => {
    invader.move(this.xVelocity, this.yVelocity);
    invader.draw(ctx);
  });
}
```

Dibuja a todos los invasores en sus nuevas posiciones después de moverse.

i) Método drawInvaders:

```
createInvaders() {
  this.invaderMap.forEach((row, rowIndex) => {
    this.invaderRows[rowIndex] = [];
    row.forEach((invaderNumber, invaderIndex) => {
      if (invaderNumber > 0) {
        this.invaderRows[rowIndex].push(
          new Invader(invaderIndex * 50, rowIndex * 35, invaderNumber)
        );
      }
    });
  });
}
```

Lógica para crear y organizar instancias de la clase Invader basándose en invaderMap.

j) Método createInvaders:

```
createInvaders() {
  this.invaderMap.forEach((row, rowIndex) => {
    this.invaderRows[rowIndex] = [];
    row.forEach((invaderNumber, invaderIndex) => {
      if (invaderNumber > 0) {
        this.invaderRows[rowIndex].push(
          new Invader(invaderIndex * 50, rowIndex * 35, invaderNumber)
        );
      }
    });
  });
}
```

Lógica para crear y organizar instancias de la clase Invader basándose en invaderMap.

k) Método collideWith:

```
collideWith(sprite) {
  return this.invaderRows.flat().some((invader) => invader.collideWith(sprite));
}
```

Verifica si algún invasor colisiona con el sprite proporcionado (generalmente las balas del jugador).

l) Método resetInvaders y resetPosition:

```
resetInvaders() {
  this.invaderRows = [];
  this.createInvaders();
}

resetPosition() {
  this.x = this.initialX;
  this.y = this.initialY;
}
```

Se encarga de reiniciar las configuraciones.

Clase Rocket.js

a) Propiedades:

```
export default class Rocket {
  rightPressed = false;
  leftPressed = false;
  shootPressed = false;
```

Estas propiedades booleanas rastrean si las teclas de flecha derecha, izquierda y espacio están presionadas, respectivamente.

b) Constructor:

```
constructor(canvas, velocity, shootController, initialLives) {
  this.canvas = canvas;
  this.velocity = velocity;
  this.shootController= shootController;
  this.initialLives = initialLives;
  this.lives = initialLives;

  this.x = this.canvas.width / 2;
  this.y = this.canvas.height - 75;
  this.width = 50;
  this.height = 48;
  this.image = new Image();
  this.image.src = "game/assets/space_ship.png";
  document.addEventListener("keydown", this.keydown);
  document.addEventListener("keyup", this.keyup);
}
```

El constructor inicializa las propiedades de la nave, establece la posición inicial, carga la imagen y registra los eventos del teclado.

c) Métodos de dibujo y actualización:

```
draw(ctx) {
  if (this.shootPressed) {
    this.shootController.shoot(this.x + this.width / 2, this.y, 4, 10);
  }
  this.move();
  this.collideWithWalls();
  this.drawLives(ctx);
  ctx.drawImage(this.image, this.x, this.y, this.width, this.height);
}
```

Este método se encarga de dibujar la nave en el lienzo. Si shootPressed es true, dispara un proyectil. Luego, se llama a move() para actualizar la posición, collideWithWalls() para evitar que la nave salga de los límites y drawLives(ctx) para mostrar el número de vidas.

d) Métodos de actualización:

```
collideWithWalls() {
  //izquierda
  if (this.x < 0) {
    this.x = 0;
  }

  //derecha
  if (this.x > this.canvas.width - this.width) {
    this.x = this.canvas.width - this.width;
  }
}
```

Es para que la nave del jugador no salga de los límites del lienzo. Si la posición x de la nave es menor que 0, se ajusta a 0, evitando que se salga por el lado izquierdo. Si x es mayor que el ancho del lienzo menos el ancho de la nave, se ajusta para evitar que se salga por el lado derecho.

```
move() {
  if (this.rightPressed) {
    this.x += this.velocity;
  } else if (this.leftPressed) {
    this.x += -this.velocity;
  }
}
```

Este método actualiza la posición horizontal (x) de la nave según las teclas de flecha presionadas. Si rightPressed es true, aumenta x según la velocidad (velocity), moviendo la nave hacia la derecha. Si leftPressed es true, disminuye x, moviendo la nave hacia la izquierda.

```
decrementLives() {
  this.lives--;
}
```

Se encarga de decrementar la cantidad de vidas (lives) de la nave en 1. Generalmente se llama cuando la nave colisiona con algún enemigo.

e) Eventos de teclado:

```
keydown = (event) => {
  if (event.code == "ArrowRight") {
    this.rightPressed = true;
  }
  if (event.code == "ArrowLeft") {
    this.leftPressed = true;
  }
  if (event.code == "Space") {
    this.shootPressed = true;
  }
};

keyup = (event) => {
  if (event.code == "ArrowRight") {
    this.rightPressed = false;
  }
  if (event.code == "ArrowLeft") {
    this.leftPressed = false;
  }
  if (event.code == "Space") {
    this.shootPressed = false;
  }
};
```

Los eventos de teclado (keydown y keyup) manejan las acciones cuando se presionan y se liberan las teclas de flecha derecha, izquierda y espacio.

Clase Shoot.js

a) Constructor:

```
constructor(canvas, x, y, velocity, shootColor) {
  this.canvas = canvas;
  this.x = x;
  this.y = y;
  this.velocity = velocity;
  this.shootColor = shootColor;

  this.width = 5;
  this.height = 20;
}
```

- canvas: Almacena la referencia al lienzo en el que se encuentra el proyectil.
- x, y: Representan las coordenadas iniciales del proyectil en el lienzo.
- velocity: Es la velocidad del proyectil, indicando qué tan rápido se desplaza hacia arriba.
- shootColor: Es el color del proyectil.
- width, height: Representan las dimensiones del proyectil.

b) Método draw:

```
draw(ctx) {
  this.y -= this.velocity;
  ctx.fillStyle = this.shootColor;
  ctx.fillRect(this.x, this.y, this.width, this.height);
}
```

- this.y -= this.velocity: Mueve el proyectil hacia arriba restando la velocidad a la coordenada y. Esto simula el movimiento ascendente del proyectil en el lienzo.
- ctx.fillStyle = this.shootColor: Establece el color de relleno del contexto de dibujo (ctx) al color del proyectil.
- ctx.fillRect(this.x, this.y, this.width, this.height): Dibuja un rectángulo (el proyectil) en las coordenadas (this.x, this.y) con las dimensiones (this.width, this.height).

c) Método collideWith:

```
collideWith(sprite) {
  if (
    this.x + this.width > sprite.x &&
    this.x < sprite.x + sprite.width &&
    this.y + this.height > sprite.y &&
    this.y < sprite.y + sprite.height
  ) {
    return true;
  } else {
    return false;
  }
}
```

- sprite: Representa a otro objeto en el lienzo (puede ser la nave enemiga o el jugador).
- Este método verifica si el proyectil colisiona con el objeto sprite mediante la comparación de coordenadas y dimensiones. Retorna true si hay colisión, de lo contrario, retorna false.

Clase ShootController.js

a) Propiedades:

```
shoots = [];
timeTillNextShootAllowed = 0;
```

- shoots: Almacena la lista de proyectiles disparados.
- timeTillNextShootAllowed: Controla el tiempo que debe pasar antes de permitir el próximo disparo.

b) Constructor:

```
constructor(canvas, maxShootsAtATime, shootColor, soundEnabled) {
  this.canvas = canvas;
  this.maxShootsAtATime = maxShootsAtATime;
  this.shootColor = shootColor;
  this.soundEnabled = soundEnabled;

  this.shootSound = new Audio("game/sounds/laser_sound.wav");
  this.shootSound.volume = 0.1;
}
```

- canvas: Referencia al lienzo.
- maxShootsAtATime: Número máximo de proyectiles permitidos en el aire al mismo tiempo.
- shootColor: Color de los proyectiles.

- soundEnabled: Indica si los sonidos están habilitados.
- shootSound: Objeto de sonido para reproducir el efecto de sonido del disparo.

c) Método draw:

```
draw(ctx) {
  this.shoots = this.shoots.filter(
    (shoot) => shoot.y + shoot.width > 0 && shoot.y <= this.canvas.height
  );

  this.shoots.forEach((shoot) => shoot.draw(ctx));
  if (this.timeTillNextShootAllowed > 0) {
    this.timeTillNextShootAllowed--;
  }
}
```

- Filtra los proyectiles para eliminar aquellos que han salido de la pantalla.
- Dibuja cada proyectil restante.
- Reduce el tiempo hasta el próximo disparo permitido.

d) Método collideWith:

```
collideWith(sprite) {
  const shootThatHitSpriteIndex = this.shoots.findIndex((shoot) =>
  shoot.collideWith(sprite)
  );

  if (shootThatHitSpriteIndex >= 0) {
    this.shoots.splice(shootThatHitSpriteIndex, 1);
    return true;
  }

  return false;
}
```

- Verifica si alguno de los proyectiles colisiona con un objeto (sprite).
- Si hay colisión, elimina ese proyectil de la lista y devuelve true; de lo contrario, devuelve false.

e) Método shoot:

```
shoot(x, y, velocity, timeTillNextShootAllowed = 0) {
  if (
    this.timeTillNextShootAllowed <= 0 &&
    this.shoots.length < this.maxShootsAtATime
  ) {
    const shoot = new Shoot(this.canvas, x, y, velocity, this.shootColor);
    this.shoots.push(shoot);
    if (this.soundEnabled) {
      this.shootSound.currentTime = 0;
      this.shootSound.play();
    }
    this.timeTillNextShootAllowed = timeTillNextShootAllowed;
  }
 }
}
```

- Permite realizar un nuevo disparo si se cumplen las condiciones: tiempo hasta el próximo disparo permitido y número máximo de proyectiles en el aire.
- Crea un nuevo objeto Shoot y lo añade a la lista de proyectiles.
- Reproduce el sonido de disparo si los sonidos están habilitados.
- Establece el tiempo hasta el próximo disparo permitido.

styles.css (Planetario)

He considerado importante resaltar los siguientes aspectos del css:

Animaciones del planetario:

a) **Animación de órbita:**
Inicia la definición de una animación llamada orbit.

A lo largo de la animación, va desde 0% hasta 100%.

En el 0%, el elemento tiene una rotación en el eje Z de 0deg.

En el 100%, el elemento tiene una rotación en el eje Z de -360deg.

Esto crea una animación completa de rotación alrededor del eje Z.

→ El elemento gira alrededor de sí mismo en un círculo completo en el plano horizontal.

```
@keyframes orbit {
  0% {
    transform: rotateZ(0deg); }

  100% {
    transform: rotateZ(-360deg); } }
```

b) Animación de subórbita:

Inicia la definición de una animación llamada suborbit.

A lo largo de la animación, va desde 0% hasta 100%.

En el 0%, el elemento tiene una rotación en el eje X de 90deg y una rotación en el eje Z de 0deg.

En el 100%, el elemento tiene una rotación en el eje X de 90deg y una rotación en el eje Z de -360deg.

Esto crea una animación que simula una rotación tridimensional alrededor de un eje.

→ El elemento rota como si estuviera orbitando alrededor de algo más grande, pero manteniendo siempre la misma cara mirando hacia arriba.

```
@keyframes suborbit {
  0% {
    transform: rotateX(90deg) rotateZ(0deg); }

  100% {
    transform: rotateX(90deg) rotateZ(-360deg); } }
```

c) Animación de inversión:

Inicia la definición de una animación llamada invert.

A lo largo de la animación, va desde 0% hasta 100%.

En el 0%, el elemento tiene una combinación de rotaciones en los ejes X, Y y Z.

En el 100%, el elemento tiene una rotación en el eje X de -90deg, una rotación en el eje Y de 0deg, y una rotación en el eje Z de 0deg.

Esto crea una animación que invierte la rotación tridimensional, comenzando con una posición específica y terminando con otra.

→ El elemento rota y da vueltas de una manera un poco más compleja, como si estuviera cambiando de dirección.

```
@keyframes invert {
  0% {
    transform: rotateX(-90deg) rotateY(360deg) rotateZ(0deg); }

  100% {
    transform: rotateX(-90deg) rotateY(0deg) rotateZ(0deg); } }
```

Duración animaciones:

a) **Mercurio:**

La posición (#mercury .pos), el planeta (#mercury .planet), y la órbita (#mercury.orbit) tienen una duración de animación de 2.89016 segundos.

```
#mercury .pos,
#mercury .planet,
#mercury.orbit {
  animation-duration: 2.89016s;
}
```

b) **Venus:**

La posición (#venus .pos), el planeta (#venus .planet), y la órbita (#venus.orbit) tienen una duración de animación de 7.38237 segundos.

```
#venus .pos,
#venus .planet,
#venus.orbit {
  animation-duration: 7.38237s;
}
```

c) **Tierra:**

La posición (#earth .pos), el planeta (#earth .planet), y la órbita (#earth.orbit) tienen una duración de animación de 12.00021 segundos.

```
#earth .pos,
#earth .planet,
#earth.orbit {
  animation-duration: 12.00021s;
}
```

Además, para los elementos dentro de la órbita de la Tierra (#earth .orbit .pos y #earth .orbit), la duración de animación es de 0.89764 segundos.

```
#earth .orbit .pos,
#earth .orbit {
  animation-duration: 0.89764s;
}
```

d) Marte:

La posición (#mars .pos), el planeta (#mars .planet), y la órbita (#mars.orbit) tienen una duración de animación de 22.57017 segundos.

```
#mars .pos,
#mars .planet,
#mars.orbit {
  animation-duration: 22.57017s;
}
```

e) Júpiter:

La posición (#jupiter .pos), el planeta (#jupiter .planet), y la órbita (#jupiter.orbit) tienen una duración de animación de 142.35138 segundos.

```
#jupiter .pos,
#jupiter .planet,
#jupiter.orbit {
  animation-duration: 142.35138s;
}
```

f) Saturno:

La posición (#saturn .pos), el planeta (#saturn .planet), la órbita (#saturn.orbit), y el anillo (#saturn .ring) tienen una duración de animación de 353.36998 segundos.

```
#saturn .pos,
#saturn .planet,
#saturn.orbit,
#saturn .ring {
  animation-duration: 353.36998s;
}
```

g) Urano:

La posición (#uranus .pos), el planeta (#uranus .planet), y la órbita (#uranus.orbit) tienen una duración de animación de 1008.20215 segundos.

```
#uranus .pos,
#uranus .planet,
#uranus.orbit {
  animation-duration: 1008.20215s;
}
```

h) Neptuno:

La posición (#neptune .pos), el planeta (#neptune .planet), y la órbita (#neptune.orbit tienen una duración de animación de 1977.49584 segundos.

```
#neptune .pos,
#neptune .planet,
#neptune.orbit {
  animation-duration: 1977.49584s;
}
```

→ Estas duraciones determinan cuánto tiempo tomará para que cada elemento complete su animación, creando así la ilusión de movimiento en la representación visual de los planetas y sus órbitas

Velocidades de órbita:
Todo esto saldrá cuando cliqueamos en el planeta que queremos ver.

```
.set-speed dl.infos dd span:after {
  content: 'Orbit Velocity';
}
```

a) Velocidad del Sol:
El sol no tiene movimiento, está parado. Por lo tanto está a 0 km/h.

```
.set-speed #sun dl.infos dd:after {
  content: '0 km/h';
}
```

b) Velocidad de Mercurio:
Significa que el planeta Mercurio tiene una velocidad de órbita y rotación que toma 2.89016 segundos para completar un ciclo. Y así con todos los planetas dependiendo de lo que tarden en completar sus ciclos.

```
.set-speed #mercury dl.infos dd:after {
  content: '170,503 km/h';
}
```

Distancias de cada planeta en el Sistema Solar:

a) **Referentes:**

Distancia respecto al Sol.

```
.set-distance dl.infos dd span:after {
  content: 'From Sun';
}
```

Distancia respecto a la Tierra.

```
.set-distance #sun dl.infos dd span:after {
  content: 'From Earth';
}
```

b) **Distancia de los planetas:**

Para cada planeta, se proporciona la distancia desde el sol en kilómetros.

```
.set-distance #mercury dl.infos dd:after {
  content: '57,909,227 km';
}
```

Conclusiones:

El inicio del proyecto "CosmoData" se caracterizó por una planificación cuidadosa centrada en la interacción con APIs, reconociendo su papel fundamental tanto en la obtención de datos astronómicos como en la futura escalabilidad de la aplicación. Esta aproximación permitió sentar las bases para una herramienta educativa, accesible y preparada para evolucionar en consonancia con los avances tecnológicos y las necesidades de los usuarios.

4. Fase de pruebas

4.1. Pruebas realizadas

Durante la fase de pruebas, se llevó a cabo una exhaustiva evaluación del sistema con el objetivo de garantizar su correcto funcionamiento e identificar áreas de posible mejora. Se implementó una batería de pruebas integral, diseñada para descubrir tanto errores lógicos como para identificar oportunidades de optimización en la ejecución del sistema.

- Creación de una batería de pruebas:

Se desarrolló una batería de pruebas completa, abarcando diversos escenarios y funcionalidades clave de la aplicación "CosmoData". Estas pruebas se diseñaron meticulosamente para evaluar la robustez del sistema y asegurar que cumpliera con los estándares de calidad establecidos. La batería de pruebas abordó aspectos críticos, desde la navegación básica hasta la interacción con datos en tiempo real provenientes de las APIs utilizadas.

- Descubrimiento de errores lógicos:

El enfoque de las pruebas se centró en descubrir posibles errores lógicos que pudieran afectar el rendimiento o la precisión de la aplicación. Se simularon situaciones diversas para evaluar cómo el sistema respondía a diferentes escenarios, asegurándose de que las funcionalidades críticas se comportan de manera coherente y fiable.

- Identificación de Oportunidades de Mejora:

Además de detectar posibles errores, las pruebas se utilizaron para identificar oportunidades de mejora en la ejecución del sistema. Se evaluaron aspectos como la velocidad de carga, la eficiencia en la manipulación de datos y la respuesta a las interacciones del usuario. Esta fase no solo buscó validar el buen funcionamiento, sino también impulsar la optimización continua del sistema.

- Corroboración Razonable del Funcionamiento:

La ejecución exitosa de la batería de pruebas proporcionó una corroboración razonable del buen funcionamiento del sistema. Se verificó que la aplicación "CosmoData" respondiera de manera consistente a las expectativas y que los usuarios pudieran disfrutar de una experiencia fluida y sin contratiempos al explorar el espacio.

La fase de pruebas se llevó a cabo meticulosamente, empleando una batería de pruebas integral para evaluar la funcionalidad del sistema, descubrir posibles errores lógicos y señalar áreas de mejora en su ejecución. Este enfoque garantiza la calidad y eficacia del producto final.

5. Conclusiones finales

5.1. Grado de cumplimiento de los objetivos fijados

En el marco de los objetivos establecidos para el proyecto CosmoData, se ha logrado el cumplimiento exitoso de la totalidad de los mismos. Sin embargo, es importante señalar que, quizás, la única excepción radica en la incorporación de herramientas específicas que permitan a los usuarios crear filtros personalizados y proporcionar mecanismos adicionales para comprender de manera más profunda la información presentada y sus diversas aplicaciones.

En detalle, los logros obtenidos mediante la implementación y creación de diversas funcionalidades son:

1. **Creación de un Mapa Interactivo:**

Se ha desarrollado e integrado un mapa interactivo que permite a los usuarios explorar visualmente datos cósmicos relevantes.

2. **Creación de Tarjetas de Datos de Astronomía:**

Se han implementado tarjetas informativas que presentan datos esenciales sobre cuerpos celestes, galaxias y otros elementos astronómicos.

3. **Integración de un Juego Temático:**

Se ha desarrollado e implementado un juego relacionado con la temática astronómica, brindando a los usuarios una experiencia lúdica y educativa en el contexto de CosmoData.

Es importante destacar que, aunque no se haya implementado una base de datos para el formulario de contacto HTML, el resto de los objetivos han sido alcanzados con éxito. En futuras iteraciones, se podría considerar la incorporación de herramientas adicionales para que los usuarios puedan personalizar sus búsquedas y obtener una comprensión más profunda de los datos astronómicos presentados en CosmoData.

5.2. Propuesta de modificaciones o ampliaciones futuras del sistema implementado

El sistema actualmente implementado, denominado CosmoData, ha alcanzado con éxito los objetivos establecidos en su fase inicial, proporcionando a los usuarios una experiencia interactiva y educativa sobre la astronomía. Sin embargo, para fortalecer y mejorar continuamente la plataforma, se proponen diversas modificaciones y ampliaciones en futuras etapas de desarrollo.

4. **Base de Datos para el Formulario de Contacto:**

Se sugiere la implementación de una base de datos para gestionar la información recopilada a través del formulario de contacto en el sitio. Esto permitirá un seguimiento más efectivo de las consultas de los usuarios y facilitará la comunicación bidireccional.

5. **Integración de Contenido Multimedia Adicional:**

Ampliar la variedad de contenido multimedia, como videos educativos, entrevistas con expertos astronómicos y animaciones interactivas. Esto enriquecerá la experiencia del usuario y proporcionará múltiples formas de aprender sobre el cosmos.

6. **Funcionalidades de Personalización del Usuario:**

Desarrollar funciones que permitan a los usuarios personalizar su experiencia en CosmoData. Esto podría incluir la creación de perfiles, la selección de preferencias astronómicas y la capacidad de guardar y compartir sus descubrimientos o configuraciones favoritas.

7. **Implementación de Juegos Educativos Adicionales:**

Introducir nuevos juegos educativos temáticos que aborden aspectos específicos de la astronomía, ofreciendo a los usuarios diversas formas de interactuar con la información astronómica de manera divertida y didáctica.

8. **Actualización Continua de Datos Cósmicos:**

Establecer un sistema automatizado para la actualización regular de datos astronómicos, garantizando que la información presentada en CosmoData esté siempre actualizada y refleje los últimos descubrimientos científicos.

9. **Mejoras en la Accesibilidad y Responsividad:**

Realizar mejoras en la accesibilidad del sitio, asegurando que sea fácilmente navegable para personas con discapacidades visuales u otras limitaciones. Además, optimizar la responsividad del diseño para adaptarse a una variedad de dispositivos y tamaños de pantalla.

10. **Incorporación de Colaboraciones y Comunidades:**

Facilitar la participación activa de la comunidad astronómica, permitiendo a los usuarios contribuir con contenido, compartir eventos astronómicos locales y establecer una plataforma para la colaboración entre entusiastas y profesionales.

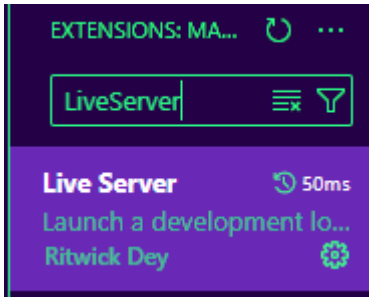
Estas propuestas representan una visión a futuro para CosmoData, buscando no solo consolidar su posición como una herramienta educativa de primer nivel sobre astronomía, sino también expandir y mejorar constantemente la experiencia de los usuarios.

6. Documentación del sistema desarrollado

CosmoData no requiere ningún tipo de instalación. Es recomendable abrir el index.html con la extensión de Visual Studio Code llamada LiveServer para que todo ocurra en tiempo real.

6.1. Manual de instalación

Entramos a Visual Studio Code y vamos a la sección de aplicaciones. Una vez allí buscamos la extensión Live Server y la instalamos.



6.2. Manual de uso

Usar la aplicación Live Server en CosmoData es algo muy sencillo y práctico. Únicamente tenemos que desplazarnos hasta el index.html de nuestro proyecto y clicar con el botón derecho del ratón en la opción de Open With Live Server.

De esta manera tendremos el proyecto en nuestro navegador en tiempo real.

7. Bibliografía

NasaGov (no date) *NASA Open Apis*, *NASA*. Available at: https://api.nasa.gov/?ref=hackernoon.com (Accessed: 16 December 2023).

ApiNinjas (no date) *Planets api*, *Planets API - API Ninjas*. Available at: https://api-ninjas.com/api/planets (Accessed: 16 December 2023).

Free 8-bit Sounds (No date) *F*. Available at: https://pixabay.com/es/sound-effects/search/8-bit/ (Accessed: 16 December 2023).

588ku, P.P. (no date) *Pixel png transparent images free download: Vector files*, *Pngtree*. Available at: https://pngtree.com/so/pixel (Accessed: 16 December 2023).

Planet PNG images - free download on Freepik (2019) *Freepik*. Available at: https://www.freepi/free-photos-vectors/planet-png (Accessed: 16 December 2023).

(2022). YouTube. 28 March. Available at: https://www.youtube.com/watch?v=qCBiKJbLcFI (Accessed: 16 December 2023).