

# **DOCUMENTACIÓN**

## **INDICE**

<b><u>1. Introducción</u></b>	<b><u>Pág. 02</u></b>
<b><u>2. Mapa de navegación</u></b>	<b><u>Pág. 04</u></b>
<b><u>3. Prototipado</u></b>	<b><u>Pág. 05</u></b>
<b><u>4. Desarrollo</u></b>	<b><u>Pág. 07</u></b>
<b><u>5. Bibliografía</u></b>	<b><u>Pág. 10</u></b>
<b><u>6. Consultas a ChatGPT</u></b>	<b><u>Pág. 12</u></b>
<b><u>6. Conclusiones</u></b>	<b><u>Pág. 23</u></b>

## 1. Introducción

Este proyecto consiste en el desarrollo de una página web que alberga dos juegos clásicos: un clon de **Wordle** llamado "**Eldrow**" y un clon de **Snake** llamado "**Ekans**". Ambos juegos han sido implementados utilizando tecnologías web básicas como **HTML**, **CSS** y **JavaScript**, sin recurrir a frameworks o librerías externas. El objetivo principal de este proyecto fue profundizar en el manejo de estas tecnologías, así como explorar la lógica detrás de la creación de juegos simples pero funcionales.

### Objetivos del Proyecto

1. **Aprendizaje práctico:** Aplicar conocimientos de HTML, CSS y JavaScript en un proyecto real.
2. **Desarrollo de juegos:** Implementar la lógica de dos juegos clásicos desde cero.
3. **Temas personalizados:** Crear un sistema de temas (claro/oscuro) que permita a los usuarios cambiar la apariencia de la página.
4. **Modo daltónico:** Incluir una opción para usuarios con daltonismo, mejorando la accesibilidad del proyecto.
5. **Documentación:** Generar una documentación completa que explique el proceso de desarrollo, las fuentes de inspiración y los recursos utilizados.

### Descripción de los Juegos

- **Eldrow:** Un clon del popular juego **Wordle**, en el que el jugador debe adivinar una palabra oculta en un número limitado de intentos. Cada intento proporciona retroalimentación sobre qué letras están correctas, presentes o ausentes en la palabra objetivo.
- **Ekans:** Un clon del clásico juego **Snake**, en el que el jugador controla una serpiente que crece al comer comida, evitando chocar con las paredes o consigo misma. El juego incluye obstáculos y aumenta la dificultad a medida que avanza.

### Tecnologías Utilizadas

- **HTML:** Para la estructura básica de las páginas.
- **CSS:** Para los estilos visuales, incluyendo temas claros/oscuros y diseño responsivo.
- **JavaScript:** Para la lógica de los juegos, manejo de eventos y funcionalidades dinámicas.

### Estructura del Proyecto

El proyecto está organizado en los siguientes archivos principales:

- **index.html:** Página principal que da la bienvenida al usuario y permite acceder a los juegos.
- **juegoEldrow.html:** Página del juego **Eldrow** (clon de Wordle).
- **juegoEkans.html:** Página del juego **Ekans** (clon de Snake).
- **style.css:** Estilos generales para la página principal.
- **styleEldrow.css:** Estilos específicos para el juego **Eldrow**.
- **styleEkans.css:** Estilos específicos para el juego **Ekans**.
- **script.js:** Lógica para el cambio de tema (claro/oscuro) en la página principal.
- **scriptEldrow.js:** Lógica del juego **Eldrow**.
- **scriptEkans.js:** Lógica del juego **Ekans**.

## Accesibilidad y Personalización

Se ha prestado especial atención a la accesibilidad y personalización del proyecto:

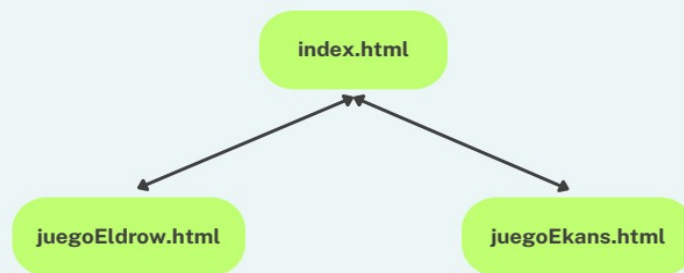
- **Temas claro/oscuro:** Los usuarios pueden cambiar entre un tema claro y oscuro según sus preferencias.
- **Modo daltónico:** Se ha implementado una opción para usuarios con daltonismo, cambiando los colores de los elementos para mejorar la visibilidad.
- **Diseño responsivo:** La página se adapta a diferentes tamaños de pantalla, permitiendo una experiencia de usuario óptima en dispositivos móviles y de escritorio.

## Conclusión de la Introducción

Este proyecto no solo ha servido como una excelente oportunidad para aplicar y reforzar conocimientos en desarrollo web, sino también como un desafío creativo para implementar juegos clásicos con un toque personal. A lo largo de la documentación, se detallarán los pasos seguidos, las decisiones de diseño y las fuentes de inspiración que han hecho posible este proyecto.

## 2. Mapa de navegación

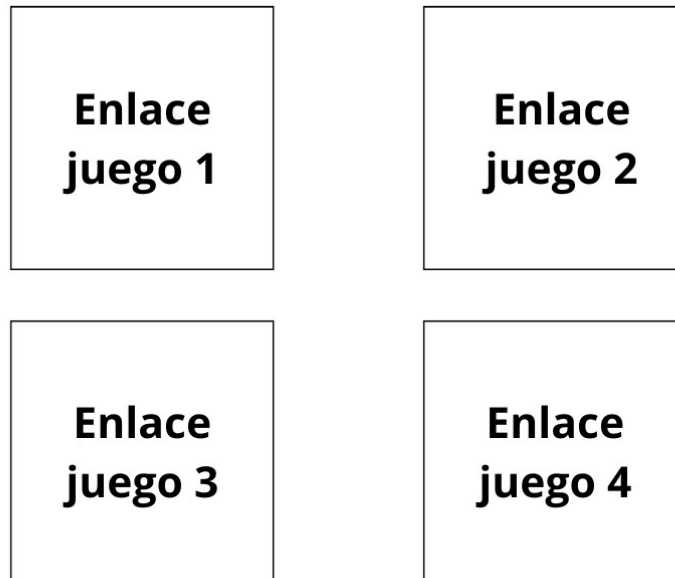
# Mapa de navegación



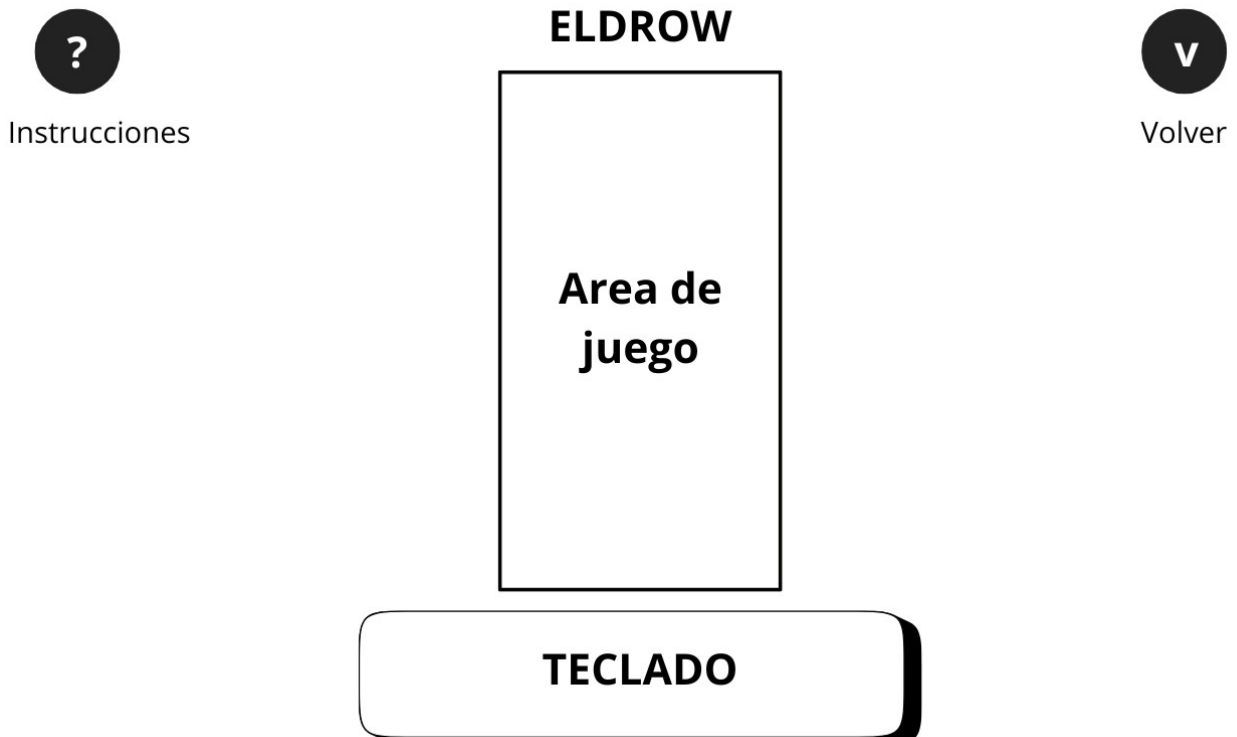
### 3. Prototipado

index.html

#### Mensaje bienvenida



juegoEldrow.html





Instrucciones

## EKANS



Volver

**Area de  
juego**

## 4. Desarrollo

En esta sección se detalla el proceso de desarrollo de la página web, incluyendo la estructura del código, las decisiones técnicas y las funcionalidades implementadas en los juegos **Eldrow** (clon de Wordle) y **Ekans** (clon de Snake). Además, se explica cómo se gestionaron los temas claro/oscuro y el modo daltónico.

### Estructura del Proyecto

El proyecto está organizado en los siguientes archivos principales:

- **HTML:**
  - **index.html:** Página principal que da la bienvenida al usuario y permite acceder a los juegos.
  - **juegoEldrow.html:** Página del juego **Eldrow** (clon de Wordle).
  - **juegoEkans.html:** Página del juego **Ekans** (clon de Snake).
- **CSS:**
  - **style.css:** Estilos generales para la página principal.
  - **styleEldrow.css:** Estilos específicos para el juego **Eldrow**.
  - **styleEkans.css:** Estilos específicos para el juego **Ekans**.
- **JavaScript:**
  - **script.js:** Lógica para el cambio de tema (claro/oscuro) en la página principal.
  - **scriptEldrow.js:** Lógica del juego **Eldrow**.
  - **scriptEkans.js:** Lógica del juego **Ekans**.

### Tecnologías Utilizadas

- **HTML:** Se utilizó para crear la estructura básica de las páginas, incluyendo los contenedores para los juegos, botones y mensajes de bienvenida.
- **CSS:** Se empleó para estilizar la página, incluyendo temas claros/oscuros, diseño responsivo y animaciones.
- **JavaScript:** Se implementó la lógica de los juegos, el manejo de eventos (como clics y pulsaciones de teclas), y la gestión de temas y modos de accesibilidad.

### Desarrollo del Juego Eldrow (Clon de Wordle)

El juego **Eldrow** es un clon de Wordle en el que el jugador debe adivinar una palabra oculta en un número limitado de intentos. A continuación, se describen los aspectos clave de su desarrollo:

#### *Lógica del Juego*

- **Generación de la palabra objetivo:** La palabra objetivo se selecciona aleatoriamente de una lista predefinida de palabras.
- **Validación de intentos:** Cada intento del jugador se compara con la palabra objetivo para determinar qué letras son correctas, están presentes o están ausentes.
- **Retroalimentación visual:** Las letras correctas se marcan en verde, las presentes en amarillo y las ausentes en gris.

#### *Interfaz de Usuario*

- **Cuadrícula de juego:** Se utilizó una cuadrícula de 5x6 celdas para mostrar los intentos del jugador.
- **Teclado virtual:** Se implementó un teclado en pantalla para que los jugadores puedan introducir sus intentos.

- **Mensajes de alerta:** Se muestran mensajes para indicar si el jugador ha ganado o perdido, y se ofrece la opción de reiniciar el juego.

### ***Estilos y Temas***

- **Temas claro/oscuro:** Los colores de la cuadrícula, el teclado y los mensajes cambian según el tema seleccionado.
- **Modo daltónico:** Se implementó una opción para cambiar los colores de las letras correctas y presentes, facilitando la distinción para usuarios con daltonismo.

### **Desarrollo del Juego Ekans (Clon de Snake)**

El juego **Ekans** es un clon del clásico juego Snake, en el que el jugador controla una serpiente que crece al comer comida, evitando chocar con las paredes o consigo misma. A continuación, se describen los aspectos clave de su desarrollo:

#### ***Lógica del Juego***

- **Movimiento de la serpiente:** La serpiente se mueve en una cuadrícula y cambia de dirección según las teclas presionadas por el jugador.
- **Generación de comida:** La comida aparece en posiciones aleatorias dentro de la cuadrícula, evitando superponerse con la serpiente o los obstáculos.
- **Obstáculos:** Se añadieron obstáculos que la serpiente debe evitar para aumentar la dificultad del juego.
- **Colisiones:** Se implementó la detección de colisiones con las paredes, la serpiente misma y los obstáculos.

#### ***Interfaz de Usuario***

- **Canvas:** Se utilizó el elemento `<canvas>` de HTML5 para renderizar el juego.
- **Controles:** El jugador puede controlar la serpiente usando las teclas de dirección o las teclas W, A, S, D.
- **Puntuación y nivel:** Se muestra la puntuación y el nivel actual en la parte superior de la pantalla.

### ***Estilos y Temas***

- **Temas claro/oscuro:** Los colores del fondo, la serpiente, la comida y los obstáculos cambian según el tema seleccionado.
- **Modo daltónico:** Se implementó una opción para cambiar los colores de la serpiente, la comida y los obstáculos, facilitando la distinción para usuarios con daltonismo.

### **Gestión de Temas y Modo Daltónico**

Se implementó un sistema de temas claro/oscuro y un modo daltónico para mejorar la accesibilidad y personalización de la página.

#### ***Temas Claro/Oscuro***

- **Cambio de tema:** El usuario puede alternar entre los temas claro y oscuro mediante un botón en la esquina superior derecha.
- **Persistencia:** La selección del tema se guarda en el `localStorage` para que se mantenga entre sesiones.

#### ***Modo Daltónico***

- **Activación:** El modo daltónico se activa mediante un botón en la esquina superior derecha.
- **Cambio de colores:** Los colores de los elementos clave (como las letras en Eldrow y la serpiente en Ekans) se ajustan para facilitar la distinción.



## Desafíos y Soluciones

- **Desafío 1:** Implementar la lógica de colisiones en el juego Ekans.
  - **Solución:** Se utilizó un sistema de detección de colisiones basado en la posición de la cabeza de la serpiente y los obstáculos.
- **Desafío 2:** Gestionar el cambio de temas en tiempo real.
  - **Solución:** Se utilizaron variables CSS personalizadas (`--color-fondo`, `--color-texto`, etc.) que se actualizan dinámicamente con JavaScript.
- **Desafío 3:** Implementar el modo daltónico sin afectar el rendimiento.
  - **Solución:** Se utilizaron clases CSS adicionales que se activan o desactivan según el modo seleccionado.

## Conclusión del Desarrollo

El desarrollo de esta página web ha sido un proceso desafiante pero enriquecedor, que ha permitido aplicar y reforzar conocimientos en HTML, CSS y JavaScript. La implementación de los juegos **Eldrow** y **Ekans**, junto con las funcionalidades de temas y modo daltónico, ha resultado en un proyecto completo y funcional que cumple con los objetivos iniciales.

## 5. Bibliografía

En esta sección se recopilan todas las fuentes de información, documentación, videos y consultas a ChatGPT que han sido utilizadas para el desarrollo de este proyecto. Estas fuentes han sido fundamentales para resolver dudas, implementar funcionalidades y mejorar la calidad del código.

### Documentación Oficial

- **HTML:**
  - [MDN Web Docs: HTML](#)
    - Documentación oficial de HTML, utilizada para entender la estructura básica de las páginas y los elementos semánticos.
- **CSS:**
  - [MDN Web Docs: CSS](#)
    - Documentación oficial de CSS, utilizada para estilizar la página y gestionar temas claro/oscuro.
  - [CSS Variables](#)
    - Guía sobre variables CSS, utilizadas para implementar los temas claro/oscuro.
- **JavaScript:**
  - [MDN Web Docs: JavaScript](#)
    - Documentación oficial de JavaScript, utilizada para implementar la lógica de los juegos y manejar eventos.
  - [LocalStorage](#)
    - Documentación sobre `localStorage`, utilizado para guardar las preferencias de tema y modo daltónico.
- **Canvas (HTML5):**
  - [MDN Web Docs: Canvas](#)
    - Documentación sobre el elemento `<canvas>`, utilizado para renderizar el juego Ekans.

### Páginas Web y Tutoriales

- **Wordle Clone:**
  - [How to Build a Wordle Clone](#)
    - Tutorial paso a paso para crear un clon de Wordle en JavaScript.
  - [Wordle Game Logic](#)
    - Explicación detallada de la lógica detrás del juego Wordle.
- **Snake Game:**
  - [Snake Game in JavaScript](#)
    - Tutorial para implementar un clon de Snake usando JavaScript y Canvas.
  - [Snake Game Logic](#)
    - Explicación de la lógica básica del juego Snake.
- **Temas Claro/Oscuro:**
  - [Dark Mode in CSS and JavaScript](#)
    - Guía completa para implementar un sistema de temas claro/oscuro.
- **Modo Daltónico:**
  - [Colorblind-Friendly Palettes](#)
    - Herramienta para seleccionar paletas de colores aptas para daltónicos.

### Videos

- **Wordle Clone:**

- [How to Build a Wordle Clone](#)
  - Video tutorial que explica cómo crear un clon de Wordle desde cero.
- **Snake Game:**
  - [Snake Game in JavaScript](#)
    - Video tutorial para implementar un clon de Snake usando JavaScript y Canvas.
- **Temas Claro/Oscuro:**
  - [Dark Mode with CSS and JavaScript](#)
    - Video que explica cómo implementar un sistema de temas claro/oscuro.

## Otras Fuentes

- **Stack Overflow:**
  - [Cambiar colores dinámicamente en CSS](#)
    - Discusión sobre cómo cambiar variables CSS dinámicamente con JavaScript.
  - [Detectar colisiones en Snake](#)
    - Explicación de cómo detectar colisiones en un juego de Snake.
- **GitHub Repositories:**
  - [Wordle Clone en GitHub](#)
    - Repositorio de un clon de Wordle en React, utilizado como referencia.
  - [Snake Game en GitHub](#)
    - Repositorio de un clon de Snake en JavaScript, utilizado como referencia.
- **ChatGPT:**
  - En el siguiente tema se listan algunas de las consultas realizadas a ChatGPT para resolver dudas específicas durante el desarrollo del proyecto, a modo de ejemplos. Se incluyen capturas de pantalla de las respuestas obtenidas. Las respuestas son a modo de orientación y en ningún momento se copia código que no se haya entendido. Además, y como se puede comprobar, las respuestas difieren en el modo en que finalmente se abordaron las dudas.

## 6. Consultas a ChatGPT

Lo hemos separado en un tema aparte, aunque estas consultas son parte de la bibliografía.

- **Consulta 1:** "Cómo implementar un sistema de temas claro/oscuro en una página web usando CSS y JavaScript."

- **Respuesta:** ChatGPT proporcionó un ejemplo de cómo usar variables CSS y JavaScript para cambiar entre temas claro y oscuro. Además del uso de localStorage para guardar la preferencia del usuario.
- **Captura de pantalla:**



## 2 Agregar el Botón en HTML

Dentro de tu archivo `index.html`, coloca un botón para cambiar entre los temas.

```
html Copiar Editar  
  
<button id="theme-toggle">Cambiar Tema</button>
```

## 3 Implementar el Cambio de Tema con JavaScript

Crea un archivo `script.js` para gestionar el cambio de tema.

```
js Copiar Editar  
  
document.addEventListener("DOMContentLoaded", () => {  
  const toggleBtn = document.getElementById("theme-toggle");  
  const body = document.body;  
  const currentTheme = localStorage.getItem("theme");  
  
  // Aplicar el tema guardado  
  if (currentTheme === "dark") {  
    body.classList.add("dark-mode");  
  }  
  
  // Evento para cambiar de tema  
  toggleBtn.addEventListener("click", () => {  
    body.classList.toggle("dark-mode");  
  
    // Guardar la preferencia en localStorage  
    if (body.classList.contains("dark-mode")) {  
      localStorage.setItem("theme", "dark");  
    } else {  
      localStorage.setItem("theme", "light");  
    }  
  });  
});
```

### 🔥 Explicación:

1. **CSS:** Define variables para los colores y cambia el tema aplicando `.dark-mode`.
2. **HTML:** Agrega un botón para cambiar de tema.
3. **JavaScript:**
  - Lee el tema guardado en `localStorage` y lo aplica al cargar la página.
  - Cambia el tema al hacer clic en el botón y guarda la preferencia.



### 🔥 Explicación:

1. **CSS:** Define variables para los colores y cambia el tema aplicando `.dark-mode`.
2. **HTML:** Agrega un botón para cambiar de tema.
3. **JavaScript:**
  - Lee el tema guardado en `localStorage` y lo aplica al cargar la página.
  - Cambia el tema al hacer clic en el botón y guarda la preferencia.

---

🔴 **Extra:** Si quieres que el tema se adapte automáticamente al del sistema, puedes modificar la lógica de `localStorage` para leer `window.matchMedia("(prefers-color-scheme: dark)").matches` al inicio.

¿Necesitas adaptarlo para Bootstrap o jQuery? 🚀

🔊 📄 👍 🗑️ ✎️ ↺

- **Consulta 2:** "Cómo manejar el movimiento de la serpiente en un juego de Snake usando JavaScript."
  - **Respuesta:** ChatGPT explicó cómo usar las teclas de dirección para cambiar la dirección de la serpiente y cómo detectar colisiones.
  - **Captura de pantalla:**



### 3. Manejo del movimiento de la serpiente

Define una función para actualizar la posición de la serpiente en función de la dirección:

```
js Copiar Editar

function updateSnake() {
  let head = { ...snake[0] }; // Copiar la cabeza actual

  // Cambiar la posición según la dirección
  if (direction === "UP") head.y -= box;
  else if (direction === "DOWN") head.y += box;
  else if (direction === "LEFT") head.x -= box;
  else if (direction === "RIGHT") head.x += box;

  // Agregar nueva cabeza
  snake.unshift(head);

  // Si la serpiente no come, eliminar la última parte (movimiento)
  if (head.x === food.x && head.y === food.y) {
    food = {
      x: Math.floor(Math.random() * (canvas.width / box)) * box,
      y: Math.floor(Math.random() * (canvas.height / box)) * box
    };
  } else {
    snake.pop();
  }
}
```

### 4. Dibujar la serpiente y la comida

Renderiza los elementos en el `canvas`:

```
js Copiar Editar

function draw() {
  ctx.clearRect(0, 0, canvas.width, canvas.height);

  // Dibujar la comida
  ctx.fillStyle = "red";
  ctx.fillRect(food.x, food.y, box, box);

  // Dibujar la serpiente
  ctx.fillStyle = "green";
  snake.forEach((segment) => {
    ctx.fillRect(segment.x, segment.y, box, box);
  });
}
```



## 5. Capturar teclas para cambiar la dirección

Usa `keydown` para modificar la dirección de movimiento:

```
js Copiar Editar

document.addEventListener("keydown", (event) => {
  const key = event.key;

  if (key === "ArrowUp" && direction !== "DOWN") direction = "UP";
  else if (key === "ArrowDown" && direction !== "UP") direction = "DOWN";
  else if (key === "ArrowLeft" && direction !== "RIGHT") direction = "LEFT";
  else if (key === "ArrowRight" && direction !== "LEFT") direction = "RIGHT";
});
```

## 6. Actualizar el juego en un intervalo

Ejecuta las funciones en un bucle usando `setInterval`:

```
js Copiar Editar

function gameLoop() {
  updateSnake();
  draw();
}

setInterval(gameLoop, 100);
```

• **Consulta 3:** "Cómo representar el teclado en la pantalla para un juego como Wordle, usando JavaScript"

- **Respuesta:** ChatGPT explicó cómo hacer una estructura de botones que representen las teclas del teclado.
- **Captura de pantalla:**

Como representar el teclado en la pantalla para un juego como Wordle, usando JavaScript

Puedes representar el teclado en la pantalla para un juego tipo *Wordle* usando HTML, CSS y JavaScript. La idea es crear una estructura de botones que representen las teclas del teclado y manejarlas con eventos de clic.

---

### 1. Estructura en HTML

Define el teclado en un `<div>` con botones representando cada tecla:

```
html
<div id="keyboard"></div>
```

---

### 2. CSS para el teclado

Aplica estilos para que parezca un teclado real:

```
css
#keyboard {
  display: grid;
  grid-template-columns: repeat(10, 1fr);
  gap: 5px;
  max-width: 400px;
  margin: 20px auto;
}

.key {
  padding: 10px;
  font-size: 16px;
  text-transform: uppercase;
  border: 1px solid #000;
  background: #ddd;
  cursor: pointer;
  text-align: center;
  border-radius: 4px;
}

.key:hover {
  background: #bbb;
}
```

### 3. JavaScript para generar el teclado

Generamos las teclas dinámicamente y las vinculamos con eventos:

```
js Copiar Editar

const keyboardLayout = [
  "qwertyuiop",
  "asdfghjkl",
  "zxcvbnm"
];

const keyboardDiv = document.getElementById("keyboard");

keyboardLayout.forEach(row => {
  const rowDiv = document.createElement("div");
  row.split("").forEach(letter => {
    const button = document.createElement("button");
    button.textContent = letter;
    button.classList.add("key");
    button.onclick = () => handleKeyPress(letter);
    rowDiv.appendChild(button);
  });
  keyboardDiv.appendChild(rowDiv);
});

function handleKeyPress(letter) {
  console.log(`Letra presionada: ${letter}`);
  // Aquí puedes agregar lógica para manejar el ingreso de letras en el juego.
}
```

#### Explicación:

1. Se define la disposición del teclado en un array de strings (`keyboardLayout`).
2. Se crea dinámicamente el teclado, generando botones y organizándolos en filas.
3. Cada botón tiene un evento `onclick` que llama a `handleKeyPress()`, donde puedes manejar la lógica del juego.

Este método hace que el teclado sea interactivo y se pueda adaptar fácilmente a diferentes diseños o tamaños de pantalla.

¿Quieres añadir funcionalidades como teclas especiales (borrar, enter) o efectos visuales? 🚀

🔊 📄 👍 🗣️ 🔗 🔄 ⌵

• **Consulta 4:** "Cómo implementar un modo daltónico en un juego cambiando los colores de los elementos."

- **Respuesta:** ChatGPT proporcionó un ejemplo de cómo usar clases CSS para cambiar los colores de los elementos en función del modo daltónico.
- **Captura de pantalla:**

Como implementar un modo daltonico en un juego cambiando los colores de los elementos

Implementar un **modo daltónico** en un juego implica cambiar los colores de los elementos para hacerlos más accesibles para personas con deficiencias en la visión del color. Existen diferentes tipos de daltonismo, como **protanopia (rojo-verde)**, **deuteranopia (verde-rojo)** y **tritanopia (azul-amarillo)**, por lo que es recomendable ofrecer varias opciones de configuración.

## 📌 Métodos para Implementarlo

### 1 Usando Paletas de Colores Alternativas

Cada elemento del juego (fondos, enemigos, UI) debería tener versiones con colores adaptados. Puedes crear un sistema que permita cambiar entre diferentes paletas dependiendo del tipo de daltonismo.

#### 🎨 Ejemplo de Paletas Alternativas

- **Normal:** 🟠 Rojo, Verde, Azul
- **Protanopia:** 🟡 Amarillo, Azul, Cian
- **Deuteranopia:** 🔵 Azul, Marrón, Amarillo
- **Tritanopia:** 🔴 Rojo, Verde, Rosa

Ejemplo en código:

```
js Copiar Editar

const colorPalettes = {
  normal: { enemy: "red", player: "green", background: "blue" },
  protanopia: { enemy: "orange", player: "yellow", background: "cyan" },
  deuteranopia: { enemy: "blue", player: "brown", background: "yellow" },
  tritanopia: { enemy: "red", player: "green", background: "pink" }
};

let currentMode = "normal";

function applyColorMode(mode) {
  currentMode = mode;
  document.documentElement.style.setProperty(
    "--enemy-color",
    colorPalettes[mode].enemy
  );
  document.documentElement.style.setProperty(
    "--player-color",
    colorPalettes[mode].player
  );
  document.documentElement.style.setProperty(
    "--background-color",
    colorPalettes[mode].background
  );
}
```

📌 Usa CSS variables para aplicar los colores dinámicamente.

## 2 Filtros de Color con CSS

Si tu juego es en 2D y basado en HTML, puedes aplicar un filtro CSS en todo el juego.

```
css
/* Filtros para cada tipo de daltonismo */
.protanopia { filter: url("#protanopia"); }
.deuteranopia { filter: url("#deuteranopia"); }
.tritanopia { filter: url("#tritanopia"); }
```

Y luego, en el código JavaScript, cambiar la clase del `body`:

```
js
document.body.className = "deuteranopia"; // Cambia según la selección del usuario
```

## 3 Shaders en Juegos 3D (OpenGL, Unity, Unreal)

Si tu juego es en 3D con motores como **Unity** o **Unreal Engine**, puedes usar **shaders** para modificar los colores en tiempo real.

Ejemplo de un **shader de corrección de color en Unity (ShaderLab)**:

```
c
Shader "Hidden/ColorBlindCorrection" {
    Properties {
        _MainTex ("Texture", 2D) = "white" {}
    }
    SubShader {
        Pass {
            CGPROGRAM
            #pragma vertex vert
            #pragma fragment frag
            sampler2D _MainTex;

            float4 frag(v2f_img i) : SV_Target {
                float4 color = tex2D(_MainTex, i.uv);
                // Simulación de protanopia
                float gray = dot(color.rgb, float3(0.567, 0.433, 0.0));
                color.rgb = float3(gray, gray, color.b);
                return color;
            }
            ENDCG
        }
    }
}
```

Esto hace que los colores rojos se representen en tonos más accesibles.

#### 4 Símbolos y Patrones Adicionales

Algunas personas no solo dependen del color, sino también de **formas y patrones**. Considera:

- **Barras y rayas** en vez de solo color.
- **Símbolos o íconos** en elementos interactivos.
- **Diferencias en brillo y contraste** para mejorar la legibilidad.

#### 🔥 Conclusión

Si el juego es en **2D con HTML/CSS**, lo mejor es usar **CSS variables y filtros**.

Si el juego es en **3D con Unity o Unreal**, usa **shaders personalizados**.

Siempre permite a los jugadores elegir el modo que les funcione mejor. 💡

¿En qué motor o lenguaje estás desarrollando el juego? Así te ayudo con código más específico. 🚀

🔊 📄 🍌 🗨️ 🔗 ↻

## 6. Conclusiones

El desarrollo de esta página web, que alberga dos juegos clásicos (**Eldrow**, un clon de Wordle, y **Ekans**, un clon de Snake), ha sido una experiencia enriquecedora y desafiante. A lo largo del proyecto, se han aplicado y reforzado conocimientos fundamentales en tecnologías web como **HTML**, **CSS** y **JavaScript**, sin recurrir a frameworks o librerías externas. Esto ha permitido un mayor entendimiento de cómo funcionan estas tecnologías en su forma más pura y cómo pueden combinarse para crear aplicaciones interactivas y funcionales.

### Logros Principales

1. **Aprendizaje Práctico:** Este proyecto ha servido como una excelente oportunidad para poner en práctica conceptos teóricos de desarrollo web. Desde la estructura básica de HTML hasta la lógica compleja de JavaScript, cada aspecto del desarrollo ha sido una oportunidad para aprender y mejorar.
2. **Implementación de Juegos Clásicos:** La creación de **Eldrow** y **Ekans** ha permitido explorar la lógica detrás de dos juegos muy diferentes. En el caso de **Eldrow**, se trabajó en la generación de palabras aleatorias, la validación de intentos y la retroalimentación visual. En **Ekans**, se implementó el movimiento de la serpiente, la generación de comida, la detección de colisiones y la gestión de obstáculos. Ambos juegos han sido diseñados para ser intuitivos y divertidos, manteniendo la esencia de los originales.
3. **Accesibilidad y Personalización:** Uno de los aspectos más destacados del proyecto ha sido la inclusión de funcionalidades que mejoran la accesibilidad y la experiencia del usuario. La implementación de un sistema de temas claro/oscuro y un modo daltónico no solo ha hecho que la página sea más inclusiva, sino que también ha permitido explorar cómo gestionar cambios dinámicos en la interfaz de usuario.

### Desafíos y Soluciones

Durante el desarrollo, surgieron varios desafíos técnicos, como la implementación de la lógica de colisiones en **Ekans**, la gestión dinámica de los temas claro/oscuro y la implementación del modo daltónico sin afectar el rendimiento. Cada uno de estos desafíos se abordó con soluciones creativas, como el uso de variables CSS personalizadas, la detección de colisiones basada en la posición de la serpiente y la implementación de clases CSS adicionales para el modo daltónico.

### Reflexiones Finales

Este proyecto ha demostrado que, incluso con tecnologías básicas como HTML, CSS y JavaScript, es posible crear aplicaciones web completas y funcionales. La elección de no utilizar frameworks o librerías externas ha permitido un mayor control sobre el código y una comprensión más profunda de cómo funcionan las tecnologías web subyacentes.

Además, la inclusión de funcionalidades como los temas claro/oscuro y el modo daltónico ha sido un recordatorio de la importancia de la accesibilidad en el desarrollo web. Crear experiencias que sean inclusivas y accesibles para todos los usuarios no solo mejora la calidad del producto, sino que también contribuye a un internet más equitativo.

En conclusión, este proyecto ha sido una excelente oportunidad para aplicar conocimientos, enfrentar desafíos técnicos y crear algo que no solo es funcional, sino también divertido y accesible. Esperamos que los usuarios disfruten de **Eldrow** y **Ekans** tanto como nosotros disfrutamos creándolos.

## Próximos Pasos

Aunque el proyecto está completo, siempre hay espacio para mejoras. Algunas ideas que se quedaron en el tintero futuras actualizaciones incluyen:

- **Añadir más niveles o modos de dificultad** en ambos juegos para aumentar la rejugabilidad.
- **Selección de nivel y tamaño del grid** en **Ekans** para que el usuario pueda elegir.
- **Implementar un sistema de puntuaciones** para que los usuarios puedan competir entre sí.
- **Explorar la integración de bases de datos** para guardar las preferencias de los usuarios o las puntuaciones más altas.
- **Ampliar la accesibilidad** con más opciones de personalización, como tamaños de fuente ajustables o compatibilidad con lectores de pantalla.

Este proyecto ha sido un punto de partida, y estamos emocionados por las posibilidades que el futuro puede traer.