TEC | Tecnológico de Costa Rica

Proyecto 4 – Match 3

Estudiantes

Geovanni González Aguilar

Gerny Diaz Hall

Quiriat Mata Araya

Escuela de Ingeniería en Computación

Curso

IC4700 - Lenguajes de Programación

Profesor

Allan Rodríguez Dávila

Grupo

60

II Semestre 2025

Manual de usuario y pruebas de funcionalidad:

Antes de comenzar, hay que asegurarse de tener instalado el siguiente software en el equipo:

- Node.js: Versión 16.0.0 o superior (Recomendada LTS v18 o v20)
  Se puede verificar si se tiene instalado ejecutando: ***node --version*** en la terminal.

- NPM (Node Package Manager): Se instala automáticamente con Node.js.
  Se puede verificar si se tiene instalado ejecutando: npm --version.

Despues se debe de tener un navegador Web Moderno: Google Chrome, Firefox, Edge o Safari (con soporte para WebSockets).

Instalación del Proyecto

El proyecto está dividido en dos módulos principales: server (Backend) y client (Frontend). Se debe instalar las dependencias para ambos por separado.

- **Paso 1: Clonar o Descargar el Proyecto**

  Descomprime el archivo del proyecto o clona el repositorio en tu carpeta de preferencia.

- **Paso 2: Instalar Dependencias del Servidor**

  Abre una terminal.

  Navega a la carpeta del servidor con el siguiente comando:

  ```
  cd Match-3/programa/server
  ```

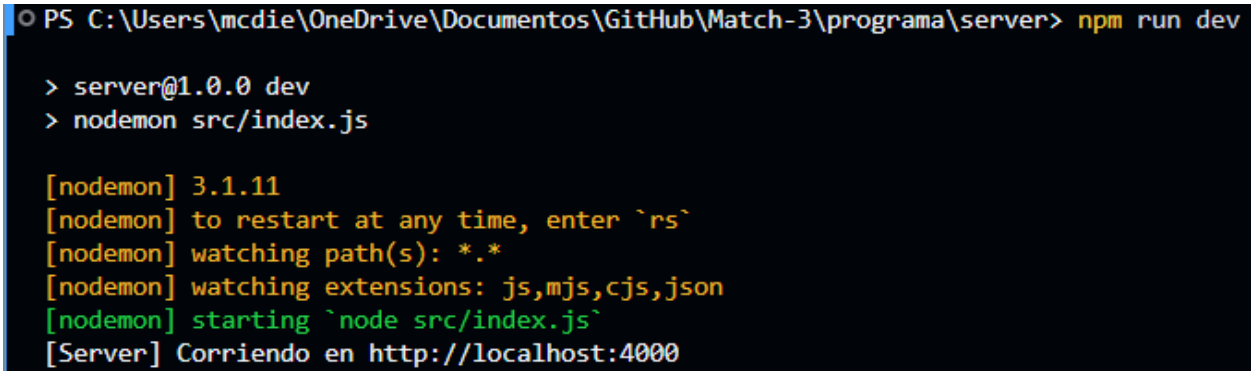
  Después se ejecuta el comando de instalación:

  ```
  npm install
  ```

  Esto descargará librerías como express, socket.io y cors.

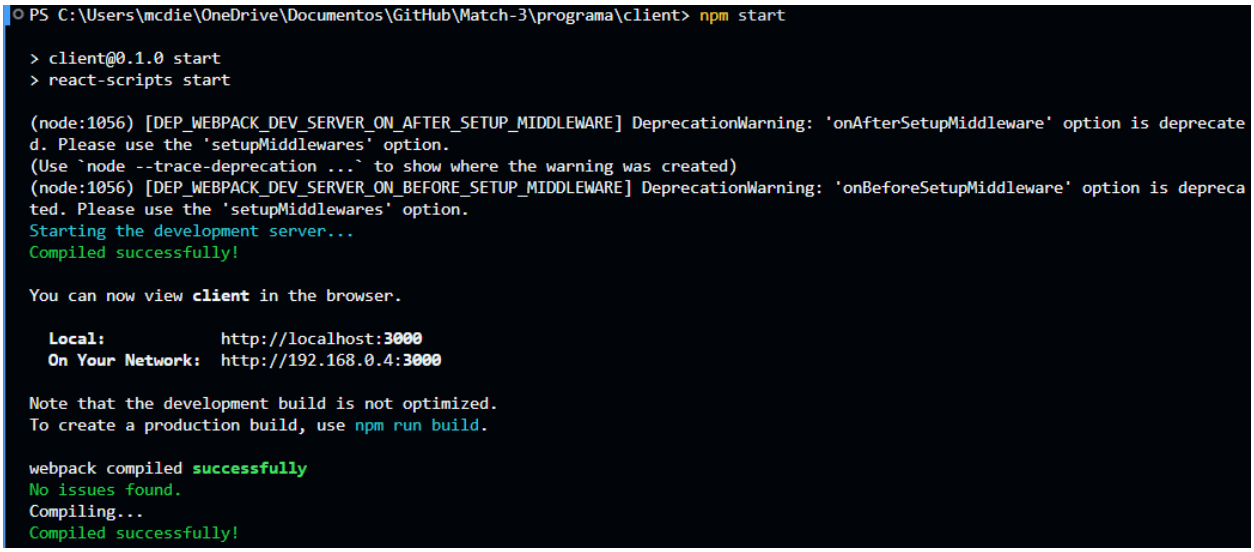
- **Paso 3: Instalar Dependencias del Cliente**
  En la misma terminal (o una nueva), hay que navegar a la carpeta del cliente:
  ```
  cd Match-3/programa/client
  ```
  Esto instalará react, react-dom y socket.io-client.

Una vez teniendo esto se puede ejecutar el proyecto, primero se abre una terminal en VS Code y se ejecuta el siguiente comando, que nos dara lo siguiente:



Por asi decirlo este es el backend, y al ver esto ya sabemos que todo esta bien y esta funcionando

Después tenemos el frontend que se ejecuta de la siguiente manera (imagen):



Apenas se compile, nos mostrara el juego como tal, que es lo que se muestra en la siguiente imagen:

Este es el menú principal del juego en el cual lo que se hace es pedir su nombre de usuario para poder jugar, una vez ingresado nos mostrara lo siguiente:

Aqui se pueden ver las opciones de crear partida, unirse a una partida y ver el ranking, en crear partida se puede ver como se puede crear una con dos modos de juego: por tiempo o por match (combinaciones)

En la tematica se puede cambiar la tematica en la cual se quiere jugar:

Una vez creado el juego, se manda al jugador creador a una sala de espera con los jugadores

Una vez se hayan unido todos, el host tiene que darle listo y presionar la tecla U, para iniciar el juego

Descripción del problema:

El objetivo general de este proyecto es desarrollar una mejor herramienta Web para el desarrollo de un juego en línea desde la perspectiva de la programación orientada a objetos, la investigación de estrategias de manejo de concurrencia y multiusuarios; de manera que las partes de la lógica y datos (Backend) que lo enmarcan sean diseccionadas minuciosamente para desarrollar una solución web (Frontend).

Con esto se busca:

1. Practicar las habilidades de modelado de aplicaciones de software.
2. Ejercitar la toma de decisiones sobre el dominio del problema y de la solución.
3. Aplicar los conceptos del Paradigma Orientado a Objetos en un proyecto programado.
4. Utilizar frameworks y herramientas de tendencia en el mercado.
5. Fomentar el trabajo en equipo.

Diseño del programa:

Diagrama UML

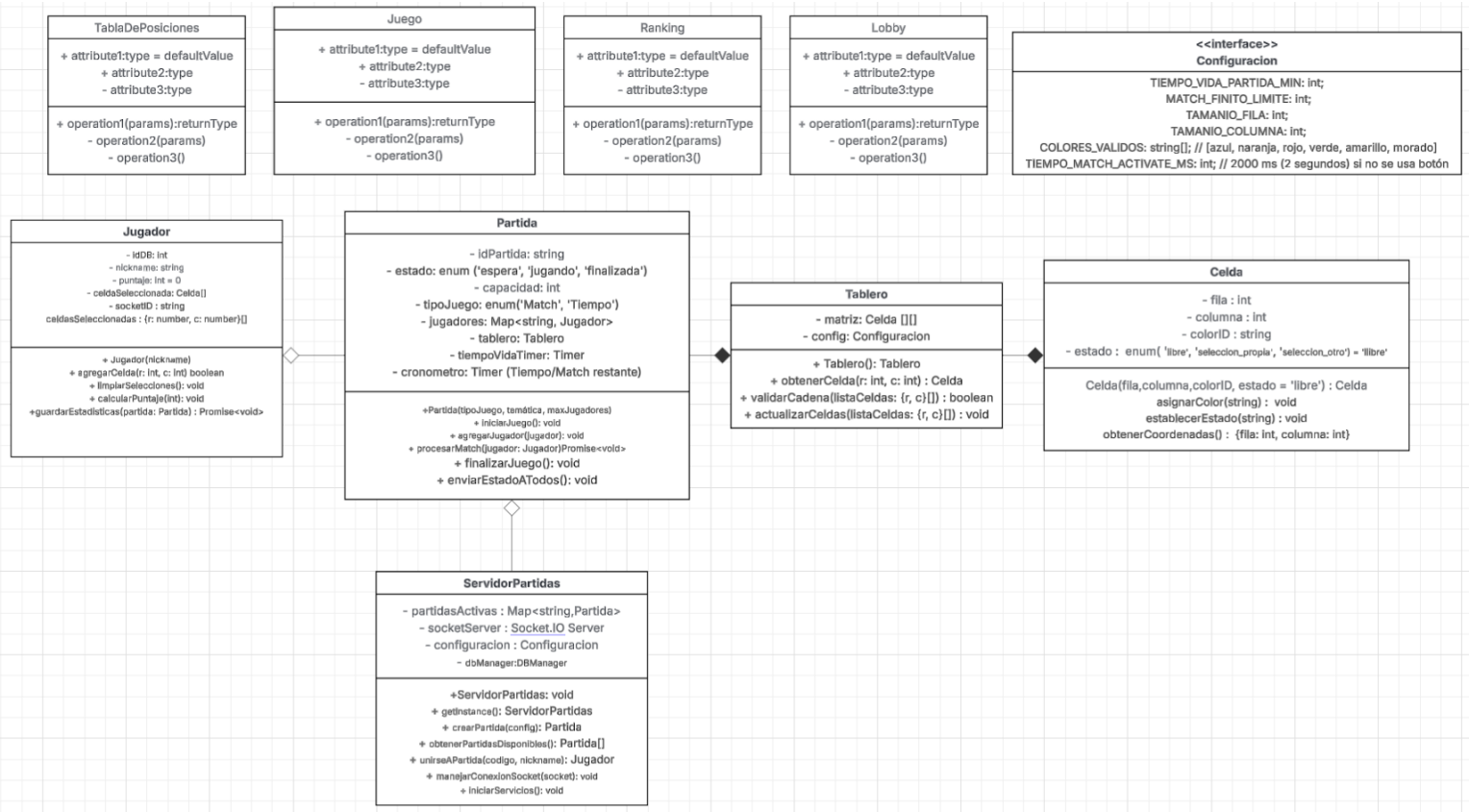


Diagrama de Paquetes:

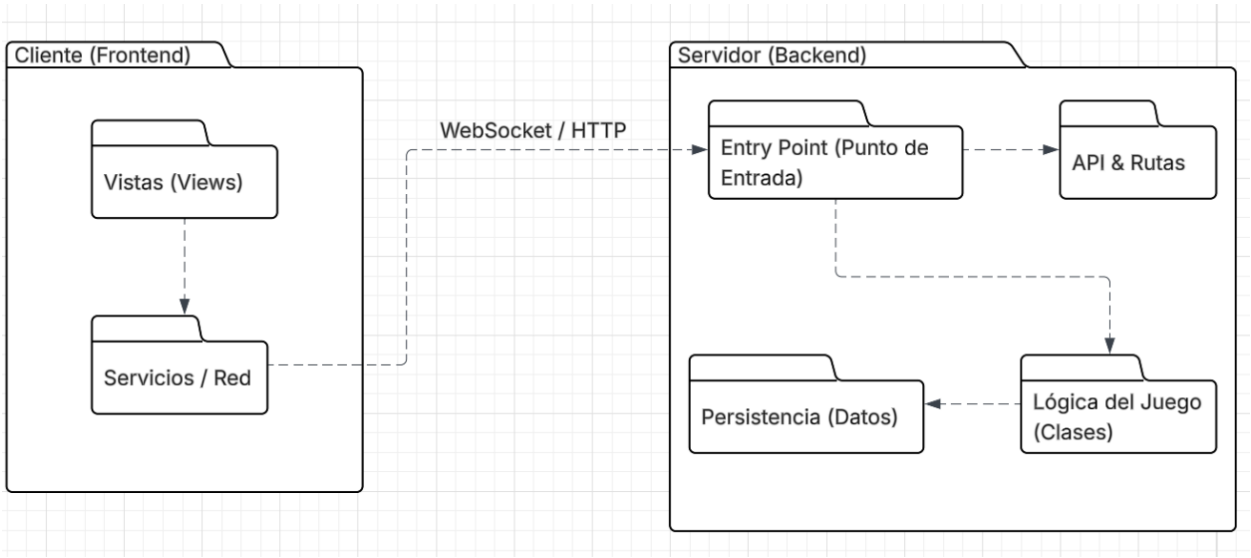


Diagrama de distribucion:

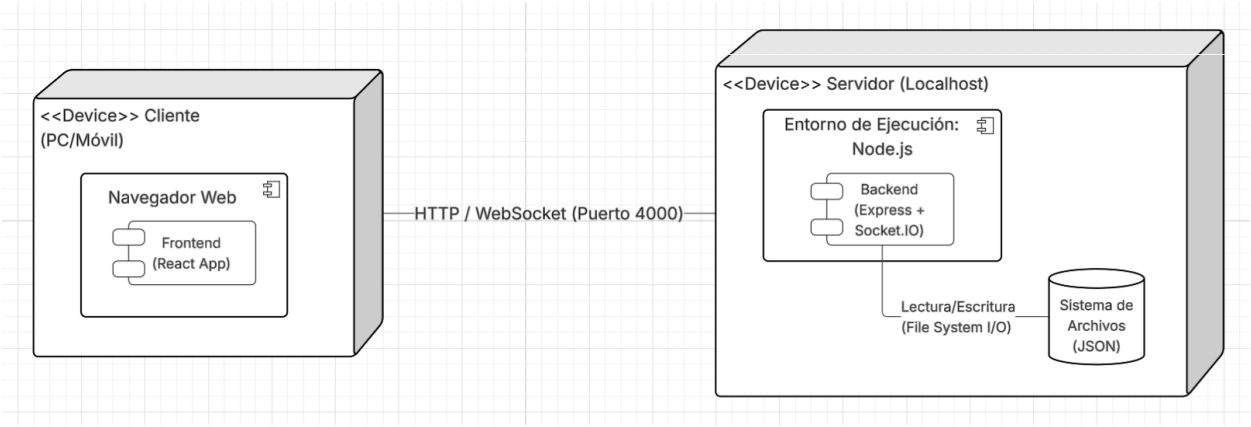


Diagrama de comunicacion:

Librerías usadas:

- **Backend (Servidor Node.js)**

Estas librerías se encuentran en el módulo del servidor y son responsables de la lógica del juego y la comunicación.

- express (^4.18.2): Framework web utilizado para levantar el servidor HTTP y manejar las rutas básicas de la API. Es la base sobre la que corre la aplicación del servidor.

- socket.io (^4.7.2): Librería fundamental del proyecto. Permite la comunicación bidireccional en tiempo real entre el servidor y los clientes. Se utiliza para la sincronización del tablero, turnos y chat.

- cors (^2.8.5): Middleware para habilitar Cross-Origin Resource Sharing. Es necesario porque tu frontend (React) y tu backend (Node) corren en puertos diferentes durante el desarrollo, y sin esto, el navegador bloquearía las conexiones.

- uuid (^9.0.1): Utilizada para generar identificadores únicos universales. En tu proyecto, esto se usa para asignar IDs únicos a las sesiones de los Jugadores y a las instancias de Partida.

- typescript (^5.9.3): Lenguaje de programación principal que añade tipado estático a JavaScript, mejorando la mantenibilidad del código.

- nodemon (^3.0.1): Utilidad que monitorea cambios en el código fuente y reinicia automáticamente el servidor, agilizando el proceso de desarrollo.

- **Frontend (Cliente React)**

Estas librerías componen la interfaz gráfica y la lógica del lado del cliente.

react (^19.2.0) & react-dom (^19.2.0): Librería base para construir la interfaz de usuario basada en componentes. Maneja el renderizado del DOM y el estado de las vistas (Lobby, Juego, etc.).

socket.io-client (^4.8.1): La contraparte del servidor. Permite al cliente React conectarse al servidor WebSocket para enviar eventos (como movimientos) y escuchar actualizaciones (como cambios en el tablero).

typescript (^4.9.5): Permite el uso de archivos .tsx y .ts en el cliente, asegurando que los props de los componentes React y las interfaces de los datos estén tipados correctamente.

web-vitals (^2.1.4): Librería para medir métricas de rendimiento y experiencia de usuario en la aplicación web.

react-scripts (5.0.1): Conjunto de scripts y configuraciones (de Create React App) que manejan el proceso de compilación, el servidor de desarrollo local y el empaquetado para producción.

@testing-library/react y relacionados: Conjunto de utilidades para realizar pruebas unitarias y de integración sobre los componentes de React.

Análisis de resultados:

Como tal, el proyecto tiene toda la lógica de manera funcional, el juego tiene buena y bonita interfaz gráfica, y se puede jugar online, el grupo considera que no les falto nada para completar el proyecto (excepto los puntos extra).

Descripción manual de los principales algoritmos:

1. Algoritmo de Detección de Grupos (Flood Fill / BFS) Ubicación: Tablero.ts -> detectarGrupoDesde Objetivo: Identificar todas las celdas contiguas del mismo color a partir de una celda seleccionada, expandiéndose en 8 direcciones (horizontal, vertical y diagonal). Lógica del Algoritmo: Este algoritmo implementa una búsqueda en anchura (Breadth-First Search - BFS).

Entrada:

Coordenada de la celda inicial (r, c).

Inicialización:

Se crea una Cola y se inserta la celda inicial.

Se crea un conjunto Visitados para evitar ciclos infinitos.

Se define el ColorObjetivo tomando el color de la celda inicial.

Ciclo de Búsqueda:

Mientras la Cola no esté vacía:

Extraer la celda actual.

Analizar sus 8 vecinos (arriba, abajo, izquierda, derecha y 4 diagonales).

Para cada vecino:

Si está dentro de los límites de la matriz Y tiene el ColorObjetivo Y no ha sido visitado:

Agregar a la Cola.

Marcar como Visitado.

Agregar a la lista de GrupoCompleto.

Validación Final:

Si el tamaño del GrupoCompleto es menor a 3, se descarta (retorna vacío).

Salida:

Lista de coordenadas que forman el bloque de color.

2. Algoritmo de Barrido de Matches Lineales Ubicación: Tablero.ts -> detectarMatches Objetivo: Escanear el tablero completo para encontrar líneas de 3 o más celdas del mismo color (solo horizontales y verticales) para procesar eliminaciones automáticas.

Lógica del Algoritmo:

Se realizan dos pasadas independientes sobre la matriz NxM (Filas x Columnas).

Barrido Horizontal:

Para cada fila r de 0 a N:

Recorrer columnas c. Comparar el color actual con el anterior.

Si es igual, incrementar contador.

Si es diferente: verificar si contador >= 3. Si es cierto, guardar las coordenadas previas en un Set de eliminación. Reiniciar contador.

Barrido Vertical:

Para cada columna c de 0 a M:

Recorrer filas r. Comparar el color actual con el superior.

Aplicar la misma lógica de conteo y acumulación en el Set.

Unificación:

El uso de un Set (Conjunto) asegura que si una celda forma parte de un match horizontal y uno vertical simultáneamente (formando una L o T), no se duplique en la lista de eliminación.

Salida:

Lista única de coordenadas a eliminar.

3. Algoritmo de Relleno con Prevención de Matches (Look-Back) Ubicación: Tablero.ts -> rellenarDespuesDeMatch y generarColorSinMatchEnPosicion Objetivo: Rellenar los espacios vacíos dejados por una jugada, asegurando que los nuevos colores generados aleatoriamente no formen inadvertidamente un nuevo match de 3 (para evitar reacciones en cadena no controladas al instante de la generación).

Lógica del Algoritmo:

Se aplica una heurística de "mirar atrás" (Look-Back) antes de asignar un color.

Iteración:

Para cada celda vacía en (fila, columna):

Detección de Patrones Prohibidos:

Se analizan los vecinos inmediatos para identificar riesgos de match:

Horizontal:

¿Las celdas en columna-1 y columna-2 tienen el mismo color X? -> Prohibir X.

Vertical:

¿Las celdas en fila-1 y fila-2 tienen el mismo color Y? -> Prohibir Y.

Intermedio (Sandwich):

¿La celda en columna-1 y columna+1 son iguales? (Previene patrón A _ A).

Filtrado:

Se crea una lista de ColoresDisponibles restando los colores prohibidos al conjunto total de colores válidos.

Asignación:

Si hay colores disponibles, elegir uno al azar de la lista filtrada.

Si todos los colores están prohibidos (caso raro de bloqueo), elegir uno totalmente al azar (fallback).

Salida: Matriz actualizada con nuevos colores estables.

4. Algoritmo de Validación de Cadena (Worker Thread) Ubicación: workerUtility.ts -> ejecutarValidacionEnWorker Objetivo: Validar si una secuencia de celdas seleccionada por el usuario constituye una jugada legal según las reglas del juego. Este cálculo se diseñó para correr en un hilo secundario (Worker) para no bloquear el Event Loop del servidor.

Lógica del Algoritmo:

Recibe una lista ordenada de coordenadas seleccionadas por el jugador.

Regla de Cantidad:

Verificar si longitud(lista) >= 3. Si no, Inválido.

Regla de Uniformidad:

Tomar el ColorObjetivo de la primera celda.

Iterar por toda la lista:
si alguna celda tiene ColorID != ColorObjetivo, Inválido.

Regla de Adyacencia Secuencial (Topología):

Iterar desde el índice i = 1 hasta el final.

Calcular distancia entre celda[i] y celda[i-1].

La distancia en filas (dr) y columnas (dc) debe cumplir: dr <= 1 y dc <= 1.

Esto valida que cada celda seleccionada es vecina inmediata de la anterior en la cadena (permitiendo diagonales), asegurando que el usuario no "saltó" celdas.

Salida: Booleano Valido/Invalido.

Bitácora:

https://github.com/Geovanni-Gonzalez/Match-3/tree/dev