Instituto Tecnológico de Costa Rica

Escuela de Ingeniería en Computadores

Paradigmas de Programación (CE1106)

Tarea #2:

BusCEMinas

Profesor:

Marco Rivera Meneses

Estudiantes:

Javier Mora Masis – 2022101555

Allan Zheng Tang – 2021141102

Steven Aguilar Alvarez – 2024202865

II Semestre 2025

27/09/25

Contenido

1. Funciones Implementadas

Generación de tablero

Esta funcionalidad crea dinámicamente tableros de tamaño configurable entre 8x8 y 15x15 celdas mediante un proceso de tres etapas. Primero, se genera una estructura de datos tipo lista que representa un tablero vacío con todas las celdas inicializadas como (0 0). Posteriormente, se distribuyen las minas de forma aleatoria utilizando un generador de números aleatorios que selecciona posiciones válidas y las marca como (1 0), evitando duplicados hasta alcanzar el porcentaje requerido según la dificultad.

El algoritmo finaliza calculando automáticamente los números de pistas para cada celda no-mina mediante un sistema que examina las ocho celdas adyacentes y cuenta las minas presentes. Este proceso utiliza una función de vecinos que valida coordenadas dentro de los límites del tablero y actualiza cada celda con el formato (0 N) donde N representa el número de minas adyacentes, generando así un tablero completamente funcional listo para jugar.

Descubrimiento de celdas

El mecanismo de descubrimiento implementa un sistema de decisión binaria que evalúa el contenido de cada celda al ser seleccionada por el usuario. Cuando se elige descubrir una celda, el sistema accede a la estructura de datos del tablero usando el índice calculado como (fila $\times$ columnas + columna) y examina el primer elemento de la tupla para determinar si es mina (1 N) o celda segura (0 N). Si es mina, se ejecuta inmediatamente la secuencia de fin de juego mostrando "MINA" en el botón y desplegando el mensaje de derrota.

Para celdas seguras, se revela el número de minas adyacentes almacenado en el segundo elemento de la tupla. Cuando este valor es cero, se activa automáticamente un algoritmo recursivo de propagación que utiliza la función vecinos para obtener coordenadas adyacentes válidas y continúa descubriendo celdas vacías de forma expansiva. Este proceso se detiene naturalmente al encontrar celdas con números positivos o alcanzar los bordes del tablero, creando el efecto característico de "apertura en cascada" del buscaminas clásico.

Sistema de banderas

El sistema de banderas utiliza un enfoque de modos que permite alternar entre "Modo: Descubrir" y "Modo: Bandera" mediante un botón dedicado. En modo bandera, los clicks directos colocan o quitan marcas "F" instantáneamente, eliminando diálogos interrumpidos y proporcionando una experiencia fluida similar al buscaminas clásico.

La integración con el contador de minas utiliza un algoritmo recursivo que cuenta las banderas activas en tiempo real, actualizando el display como "Minas: X" donde X representa minas restantes. El sistema protege celdas marcadas contra descubrimiento accidental y mantiene la integridad del estado del juego mediante validaciones automáticas entre modos.

Detección de estados de juego (Victoria/Derrota)

Esta funcionalidad detecta condiciones de victoria y derrota del jugador mediante la detección de minas y celdas descubiertas, en donde, el jugador recibirá un mensaje de derrota si se selecciona una

casilla con una mina dentro. Por el otro lado, habrá un contador de celdas no descubiertas, en el que, el momento que el jugador haya pasado por todas las celdas sin haber pisado una mina en todo el tablero, se dará la condición de victoria.

Interfaz de usuario

La interfaz utiliza el sistema de GUI de Racket organizando elementos en paneles jerárquicos con un diseño vertical que separa la información del juego (título, contador) de la zona de interacción (matriz de botones). El tablero se construye dinámicamente creando paneles verticales para cada columna y poblándolos con botones que representan las celdas individuales. Cada botón tiene asociado un callback que implementa la lógica de interacción, utilizando message-box/custom para presentar opciones de acción cuando se clickea una celda vacía.

El sistema de navegación entre pantallas emplea un mecanismo de contenedores intercambiables donde se ocultan todos los elementos hijos del panel principal antes de mostrar la nueva pantalla. La sincronización entre la lógica del tablero y la representación visual se mantiene mediante funciones que traducen coordenadas de matriz bidimensional a índices lineales y viceversa, permitiendo que las acciones del usuario en la interfaz se reflejen correctamente en la estructura de datos subyacente del juego.

Configuración de dificultad

El sistema de dificultad implementa un esquema porcentual que determina la densidad de minas basándose en el área total del tablero seleccionado. Los tres niveles predefinidos (fácil 10%, medio 15%, difícil 20%) se traducen matemáticamente usando la función round para calcular el número exacto de minas como (área_total $\times$ porcentaje), garantizando un mínimo de una mina independientemente del tamaño del tablero. Esta aproximación escalable permite que tableros más grandes mantengan proporciones de dificultad consistentes.

La selección de parámetros se realiza mediante componentes choice% que presentan listas predefinidas de opciones válidas para filas, columnas y dificultad. El sistema valida automáticamente las combinaciones seleccionadas y las convierte de cadenas de texto a valores numéricos antes de pasarlas al generador de tableros. Esta configuración modular permite ajustar fácilmente los parámetros del juego sin modificar la lógica central, manteniendo la flexibilidad del sistema mientras asegura valores válidos dentro de los rangos establecidos (8-15 para dimensiones).

2. Estructura de Datos Desarrolladas

Estructura principal: Lista de celdas (Tablero)

```
'((0 3) (1 0) (0 2) (0 1) (0 3) (0 1) (0 0) (0 1) (1 0))
```

Esta es la estructura de datos central del proyecto. Es una lista lineal que representa el tablero bidimensional de forma secuencial. Cada elemento de la lista es una tupla de dos elementos que representa una celda individual:

- **Primer elemento**: Indicador de mina (0 = no mina, 1 = mina)
- **Segundo element**o: Número de minas adyacentes

Ejemplo de interpretación:

- (0 3) = Celda sin mina con 3 minas adyacentes
- (1 0) = Celda con mina (el segundo valor es irrelevante)
- (0 0) = Celda sin mina y sin minas adyacentes

Lista de coordenadas (vecinos)

```
'((0 1) (1 0) (1 1) (2 1))
```

Lista que contiene tuplas de coordenadas (fila columna) que representan las posiciones de celdas vecinas válidas dentro de los límites del tablero.

Matriz de botones de la interfaz

```
'(((button1 button2 button3)    ; Columna 0
(button4 button5 button6)       ; Columna 1
(button7 button8 button9)))     ; Columna 2
```

Lista de listas que contiene los objetos botón de la interfaz gráfica, organizados por columnas para facilitar el acceso y manipulación visual del tablero.

Lista de opciones de configuración

```
'("8" "9" "10" "11" "12" "13" "14" "15")    ; tamaños
'("facil" "medio" "dificil")                ; dificultades
```

Listas simples de cadenas que definen las opciones disponibles en los menús de configuración.

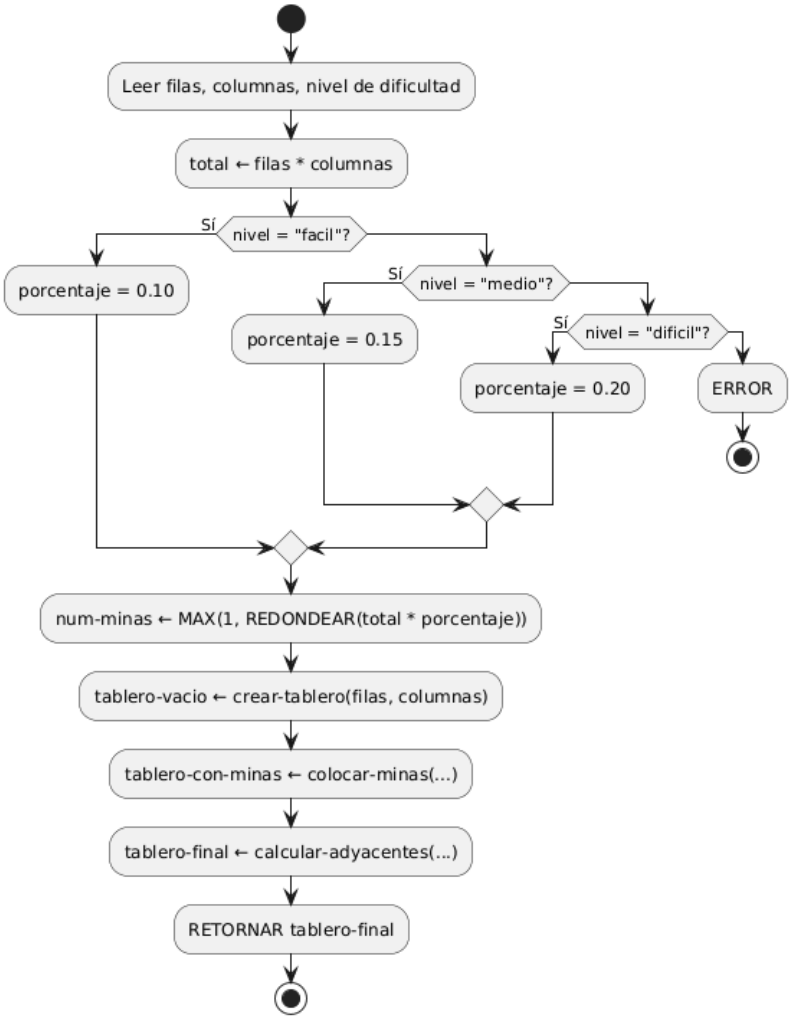
La estructura de datos principal (lista de celdas) es la más importante ya que encapsula toda la lógica del juego y se manipula exclusivamente mediante funciones recursivas, manteniendo los principios de programación funcional requeridos en el proyecto.

3. Algoritmos Desarrollados

Algoritmo de Generación de Tablero

Este algoritmo implementa un pipeline funcional de tres etapas secuenciales que transforma las especificaciones de entrada (dimensiones y nivel de dificultad) en una estructura de datos completa y funcional del tablero. El proceso inicia calculando el número total de minas basándose en el porcentaje correspondiente al nivel seleccionado, seguido de la creación de una lista lineal que representa todas las celdas del tablero inicializadas como vacías con el formato (0 0).

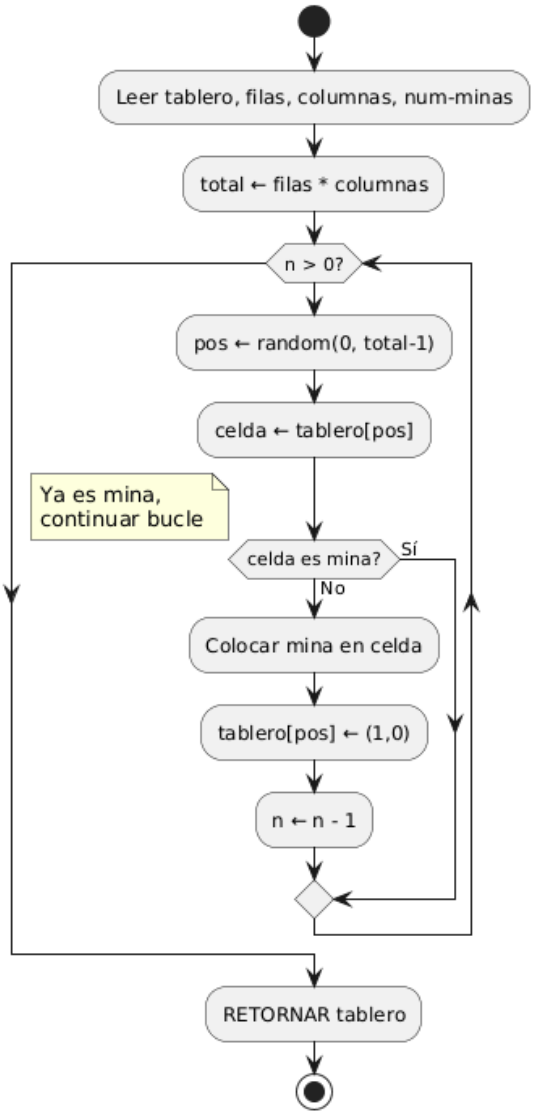
La segunda y tercera etapa del pipeline operan sobre versiones inmutables del tablero, donde primero se distribuyen las minas de forma aleatoria marcando las posiciones seleccionadas como (1 0), y posteriormente se calcula automáticamente el número de minas adyacentes para cada celda no-mina, actualizando su formato a (0 N) donde N representa el conteo de vecinos con minas. Este enfoque funcional garantiza que cada etapa produzca una nueva versión del tablero sin modificar las anteriores, cumpliendo con los principios de inmutabilidad requeridos.



Algoritmo de Colocación Aleatoria de Minas

Este algoritmo recursivo implementa una estrategia de distribución aleatoria de minas que garantiza la colocación exacta del número requerido sin duplicados. El proceso utiliza un enfoque de muestreo con rechazo, donde se genera una posición aleatoria dentro del rango válido del tablero y se verifica si la celda correspondiente ya contiene una mina. Si la posición está disponible, se coloca la mina y se decrementa el contador; si está ocupada, se genera una nueva posición aleatoria manteniendo el contador sin cambios.

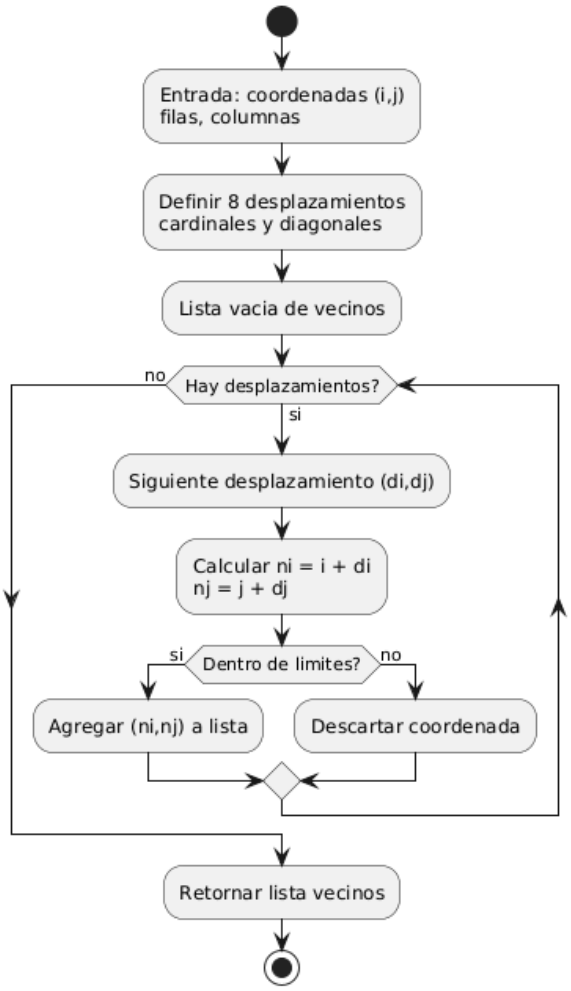
La recursión continúa hasta que el contador de minas por colocar llega a cero, momento en el cual se retorna el tablero con todas las minas distribuidas. Este método asegura una distribución verdaderamente aleatoria mientras mantiene la eficiencia en la mayoría de casos, aunque en tableros con alta densidad de minas puede requerir múltiples intentos para encontrar posiciones libres. La implementación utiliza la función reemplazar-en-lista para crear nuevas versiones del tablero sin mutar la estructura original.



Algoritmo de Cálculo de Vecinos

Determina las coordenadas válidas de todas las celdas adyacentes a una posición específica del tablero, implementando la lógica fundamental para identificar relaciones de vecindad en el juego. El sistema utiliza una matriz predefinida de ocho desplazamientos relativos que representan las direcciones cardinales y diagonales desde cualquier celda central: arriba-izquierda, arriba, arriba-derecha, izquierda, derecha, abajo-izquierda, abajo y abajo-derecha. Para cada desplazamiento, se calculan las nuevas coordenadas sumando los valores de desplazamiento a la posición original.

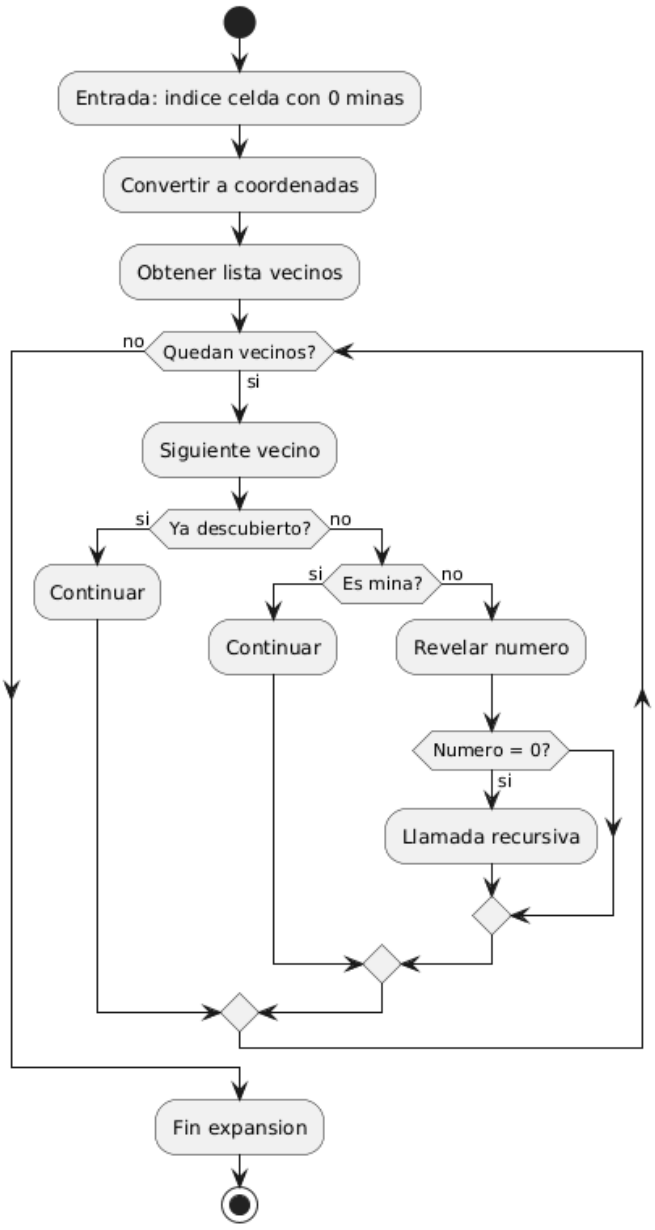
El algoritmo incorpora validación automática de límites para filtrar coordenadas que excedan las dimensiones del tablero, asegurando que solo se retornen posiciones válidas dentro del rango [0, filas-1] y [0, columnas-1]. Esta funcionalidad es esencial tanto para el cálculo inicial de minas adyacentes durante la generación del tablero como para el proceso de descubrimiento automático durante el juego. El resultado es una lista de tuplas de coordenadas que pueden ser utilizadas directamente por otros algoritmos para acceder a las celdas vecinas correspondientes.



Algoritmo de Descubrimiento Recursivo (Auto-expansión)

Este algoritmo implementa la característica más distintiva del buscaminas: la propagación automática en cascada que ocurre cuando se descubre una celda con cero minas adyacentes. El proceso inicia desde una celda que ha sido identificada como vacía (valor 0) y procede a examinar sistemáticamente todas sus celdas vecinas válidas utilizando el algoritmo de cálculo de vecinos. Para cada vecino, el sistema verifica primero si la celda ya ha sido descubierta anteriormente consultando el estado de su botón correspondiente en la interfaz, evitando así el procesamiento redundante y ciclos infinitos.

Cuando se encuentra un vecino no descubierto que además es seguro (no contiene mina), el algoritmo lo revela automáticamente mostrando su número de minas adyacentes en la interfaz. Si este vecino recién descubierto también tiene un valor de cero minas adyacentes, se activa una llamada recursiva que aplica el mismo proceso de descubrimiento sobre esa nueva celda. Esta propagación continúa expandiéndose hasta que se alcancen celdas con números positivos o se llegue a los bordes del tablero, creando el efecto visual característico de "apertura en área" que permite a los jugadores descubrir grandes regiones seguras con una sola acción.


Entrada: indice celda con 0 minas
Convertir a coordenadas
Obtener lista vecinos
no
Quedan vecinos?
si
Siguiente vecino
si
Ya descubierto?
no
Continuar
si
Es mina?
no
Continuar
Revelar numero
Numero = 0?
si
Llamada recursiva
Fin expansion

4. Problemas sin Solución

A pesar de estos desafíos técnicos, el proyecto logró implementar exitosamente todas las funcionalidades core del buscaminas sin mayores obstáculos en la lógica fundamental del juego. Los problemas encontrados se resolvieron mediante enfoques alternativos que, aunque diferentes a las implementaciones convencionales, mantienen la funcionalidad completa y la experiencia de juego esperada. La adopción de soluciones creativas como el sistema de diálogos para la interacción y el enfoque en la funcionalidad sobre la estética permitió completar todos los requerimientos establecidos en la especificación del proyecto, demostrando que las limitaciones técnicas pueden superarse mediante adaptaciones inteligentes del diseño.

5. Plan de Actividades

Actividad	Descripción	Tiempo Estimado	Fecha de Entrega	Responsable
SEMANA 1				
Diseño e Implementación de Estructuras de Datos	Crear las estructuras que representarán el estado completo del juego usando listas anidadas y programación funcional pura.	6 horas	10/09/25	Allan
Generación de Tablero Configurable	Implementar la función principal que genera un tablero vacío de dimensiones configurables, sin minas.	6 horas	12/09/25	Allan
Colocación Aleatoria de Minas	Desarrollar el algoritmo que coloca minas de forma aleatoria en el tablero según el nivel de dificultad especificado.	6 horas	14/09/25	Javier
SEMANA 2				
Cálculo de Minas Adyacentes	Implementar el algoritmo que calcula el número de minas adyacentes para cada celda del tablero.	6 horas	16/09/25	Javier
Descubrimiento Simple de Celdas	Desarrollar la funcionalidad básica para que el usuario pueda descubrir celdas individuales, manejando los casos de mina y no-mina.	6 horas	18/09/25	Javier
Sistema de Banderas	Implementar la funcionalidad que permite marcar y desmarcar celdas con banderas para indicar posibles ubicaciones de minas.	6 horas	20/09/25	Steven
SEMANA 3				
Descubrimiento Recursivo Automático	Desarrollar el algoritmo recursivo que automáticamente descubre celdas vecinas cuando se encuentra una celda con 0 minas adyacentes.	6 horas	22/09/25	Steven
Detección de Condición de Victoria	Implementar la lógica que detecta cuando se ha ganado el juego, verificando que todas las celdas sin mina han sido descubiertas.	6 horas	24/09/25	Allan
Interfaz Gráfica Completa	Crear una interfaz gráfica que permita visualizar el tablero e interactuar con el juego mediante clicks.	6 horas	25/09/25	Steven
Documentación, Pulido y Pruebas	Realizar la documentación completa del proyecto, pulir la experiencia de usuario y ejecutar pruebas para asegurar la calidad de la tarea.	6 horas	27/09/25	TODOS

6. Problemas Encontrados

Limitaciones Estéticas y de Personalización en Racket GUI

Durante el desarrollo de la interfaz gráfica, nos enfrentamos a significativas limitaciones en las capacidades de personalización visual del toolkit GUI de Racket. Inicialmente planeamos recrear la experiencia visual clásica del Buscaminas con colores específicos para cada número, iconografía personalizada para las minas, y un diseño más pulido con gradientes y efectos visuales modernos. Sin embargo, descubrimos que Racket GUI está orientado hacia funcionalidad sobre estética, ofreciendo opciones muy limitadas para personalizar colores de texto, fondos de botones, y elementos gráficos avanzados. Exploramos múltiples enfoques como modificar propiedades de estilo de botones, crear elementos gráficos personalizados mediante canvas, y utilizar librerías adicionales de imagen, pero estos intentos resultaron ser extremadamente restringidos, incompatibles entre plataformas, o requerían implementar toda la lógica de interacción desde cero, lo cual habría comprometido significativamente el tiempo de desarrollo y la estabilidad del proyecto.

Adoptamos un enfoque pragmático utilizando emojis Unicode para minas y banderas que proporcionan claridad visual inmediata sin requerir recursos gráficos externos, complementado con un diseño limpio basado en paneles jerárquicos y un header personalizado con fondo negro y texto blanco escalado. Esta solución mantiene funcionalidad completa mientras proporciona una experiencia visual clara y profesional dentro de las limitaciones del framework. La experiencia enseñó que la funcionalidad debe priorizarse sobre la estética cuando existen limitaciones de tiempo y herramientas, y que el uso creativo de recursos disponibles puede proporcionar soluciones elegantes y universales. Recomendamos evaluar frameworks alternativos como bibliotecas web o toolkits más flexibles para futuros proyectos que requieran mayor personalización visual, definir requisitos estéticos temprano para evaluar viabilidad técnica, y reconocer que las limitaciones pueden dirigir hacia soluciones más simples y mantenibles que a menudo resultan más efectivas que los diseños inicialmente planeados.

Implementación del sistema de modos

Durante el desarrollo del sistema de interacción, inicialmente planeamos usar el patrón clásico donde click izquierdo descubre celdas y click derecho coloca banderas, pero descubrimos que los botones de Racket/GUI no soportan detección de click derecho. Como solución temporal implementamos diálogos emergentes que mostraban opciones "Descubrir" y "Bandera" en cada click, pero esto interrumpía el flujo del juego. Finalmente desarrollamos un sistema de modos alternantes con un botón dedicado que cambia entre "Modo: Descubrir" y "Modo: Bandera", controlado por la variable modo-bandera?, que resultó ser una solución elegante y funcional que demuestra cómo las limitaciones técnicas pueden dirigir hacia diseños alternativos efectivos.

7. Recomendaciones

Para futuros proyectos de desarrollo de juegos utilizando programación funcional pura en Racket, recomendamos aprovechar al máximo las capacidades nativas del framework Racket/GUI, que, aunque inicialmente puede parecer limitado en personalización estética, ofrece una base sólida y estable para implementar funcionalidades complejas sin comprometer los principios del paradigma funcional. Es aconsejable definir expectativas visuales realistas desde el inicio del desarrollo, enfocándose en la claridad de la interfaz y la experiencia de usuario por encima de elementos puramente decorativos. Sugerimos explorar más profundamente las capacidades de canvas y drawing contexts para crear elementos visuales personalizados cuando sea necesario, y considerar el uso estratégico de símbolos Unicode y emojis como una solución elegante y multiplataforma para iconografía.

Para expandir las funcionalidades del juego manteniendo la coherencia con el paradigma funcional, recomendamos implementar un sistema de puntuación basado en tiempo y eficiencia, niveles de dificultad completamente personalizables que permitan al usuario definir exactamente el porcentaje de minas deseado, y funcionalidades adicionales como temporizador visual, estadísticas de partidas, y diferentes modos de juego. También sugerimos investigar las capacidades de animación y transiciones suaves disponibles en Racket/GUI para mejorar la retroalimentación visual durante el descubrimiento automático de celdas. Es importante mantener siempre la separación clara entre la lógica funcional del juego y la capa de presentación, documentando exhaustivamente las decisiones de diseño para facilitar futuras extensiones del proyecto sin comprometer la integridad arquitectural establecida.

8. Conclusiones

Este proyecto demostró exitosamente la aplicación práctica del paradigma de programación funcional en el desarrollo de una aplicación interactiva compleja como el Buscaminas. La implementación utilizando exclusivamente programación funcional pura, sin variables mutables, bucles imperativos o efectos secundarios, evidenció que es posible crear software robusto y funcional siguiendo estrictamente los principios de inmutabilidad y recursión. La separación modular entre la lógica del juego y la interfaz gráfica facilitó el desarrollo colaborativo del equipo y demostró la importancia de la abstracción de datos y la encapsulación de funcionalidades en módulos independientes que pueden desarrollarse y probarse de manera aislada.

La experiencia de desarrollo reforzó conceptos fundamentales como el diseño de estructuras de datos eficientes usando listas, la implementación de algoritmos recursivos para resolución de problemas complejos como el descubrimiento automático en cascada, y la importancia de la validación y manejo de casos límite en sistemas interactivos. A pesar de las limitaciones encontradas en la personalización visual de la interfaz, el proyecto cumplió satisfactoriamente todos los objetivos planteados, demostrando que las restricciones técnicas pueden superarse mediante soluciones creativas que mantienen la funcionalidad esencial. Esta experiencia consolidó nuestro entendimiento del paradigma funcional como una herramienta poderosa para el desarrollo de software confiable y mantenible, especialmente en contextos donde la corrección del comportamiento es más crítica que la sofisticación visual.

9. Bibliografía

Guzmán, J. E. (2006). Introducción a la programación con Scheme. Cartago: Editorial Tecnológica de Costa Rica.

Flatt, M., & Findler, R. (s. f.). *The Racket Guide*. https://docs.racket-lang.org/guide/

Racket. (2017, agosto 15). The Racket graphical interface toolkit. Racket. http://download.racket-lang.org/releases/6.9/doc/gui/

Wikipedia. (2025, septiembre 4). Buscaminas. En Wikipedia. https://es.wikipedia.org/wiki/Buscaminas