Documentación Mastergoal

Luis Morel – Sebastián Laguardia

Variables Globales y Definiciones:

1. Constantes

· tamX y tamY definen el tamaño de la cancha.

Estructuras:

1. Struct jugador:

Se utiliza esta estructura para representar a cada jugador dentro de juego. Cada jugador tiene:

- id: Un identificador (entero) único para cada jugador.
- pos_x: La posición en el eje X (fila) del tablero.
- pos_y: La posición en el eje Y (columna) del tablero.

Posteriormente se utilizará estas estructuras para hacer dos vectores de Struct jugador, uno para cada equipo.

2. Struct pelota:

Se utiliza esta estructura para representar la pelota dentro del tablero, con dos datos:

- pos_x: La posición de la pelota en el eje X (fila) del tablero.
- pos_y: La posición de la pelota en el eje Y (columna) del tablero.

Funciones:

1. imprimirCancha:

Esta función imprime el tablero de juego. Se imprime la cancha, los jugadores y la pelota. La pelota se marca con un '0', los jugadores del equipo rojo con 'R' y los jugadores blancos con 'B'.

· Parámetros:

- cancha[tamY][tamX]: La matriz que representa la cancha.
- pelota *p: Un puntero a la estructura que indica la posición de la pelota.

Itera sobre la matriz e imprime el contenido correspondiente de la cancha, jugadores y pelota.

2. pedirMovimiento:

Pide al usuario ingresar el ID del jugador que desea mover y las coordenadas a las que quiere moverlo.

- Parámetros:

- cancha[tamY][tamX]: La cancha actual.

- jugador team[]: Un array de jugadores del equipo que tiene el turno.

- tamaño_equipo: Número de jugadores en el equipo.

- turno: Turno actual (0 para el equipo rojo, 1 para el equipo blanco).

Retorno: Devuelve 1 si el movimiento fue válido, 0 en caso que el movimiento no fue válido.

3. moverJugador:

Intenta mover un jugador a una nueva posición.

- Parámetros:

- cancha[tamY][tamX]: La cancha actual.

- jugador *j: Puntero al jugador a mover.

- nueva_x, nueva_y: Nuevas coordenadas a las que el jugador desea moverse.

Verifica que el movimiento esté dentro de los límites y que sea un movimiento válido. Si todo es correcto, actualiza la posición del jugador.

Retorno: Devuelve 1 si el movimiento fue válido, 0 en caso contrario.

4. pedirMovimientoPelota:

Pide al usuario las coordenadas para mover la pelota.

- Parámetros:

- cancha[tamY][tamX]: La cancha actual.

- pelota *p: La pelota actual.

- turno: El turno actual (0 para equipo rojo, 1 para equipo blanco).

Pide al usuario las nuevas coordenadas y llama a la función moverPelota para intentar mover la pelota.

5. moverPelota:

Mueve la pelota a una nueva posición si el movimiento es válido.

· Parámetros:

- cancha[tamY][tamX]: La cancha actual.
- pelota *p: La pelota a mover.
- nueva_x, nueva_y: Nuevas coordenadas para la pelota.
- turno: El turno actual (0 para el equipo rojo, 1 para el equipo blanco).

Retorno: Devuelve 1 si el movimiento fue válido, 0 si no.

6. moverComp:

Selecciona al jugador más cercano a la pelota para moverlo hacia ella.

· Parámetros:

- cancha[tamY][tamX]: La cancha actual.
- jugador team[]: El equipo controlado por la IA.
- tamaño_equipo: Tamaño del equipo.
- pelota *p: La posición actual de la pelota.

7. moverPelotaComp:

Realiza movimientos de la pelota al azar por la computadora.

· Parámetros:

- cancha[tamY][tamX]: La cancha actual.
- pelota *p: La pelota a mover.
- turno: El turno actual (0 para el equipo rojo, 1 para el equipo blanco).

Retorno: 0 si no pudo mover la pelota y 1 si se pudo mover.

8. verificarPosesion:

Verifica quién tiene la posesión de la pelota con los cuadros adyacentes a la misma.

· Parámetros:

- cancha[tamY][tamX]: La cancha actual.
- pelota *p: La pelota a mover.
- turno: El turno actual (0 para el equipo rojo, 1 para el equipo blanco).

Retorno: 1 si el equipo tiene la posesión y 0 si no la tiene.

9. inicializarTablero:

Inicializa el tablero colocando a los jugadores en sus posiciones iniciales y la pelota en el centro.

· Parámetros:

- cancha[tamY][tamX]: La cancha a inicializar.
- jugador team_rojo[]: El equipo rojo.
- jugador team_blanco[]: El equipo blanco.
- tamaño_equipo: Tamaño del equipo.
- pelota *p: La pelota.

10. verificarGol:

Verifica si se ha marcado un gol. Un gol ocurre si la pelota llega a las posiciones de la portería.

· Parámetros:

- pelota *p: La posición de la pelota.

Retorno: Devuelve 1 si se marcó un gol, 0 si no.

Descripción y justificación de la estrategia utilizada

Documentación de Funciones GTK

1. `void on_tablero_button_clicked(GtkButton *button, gpointer user_data)`

Descripción:
Callback que gestiona el evento de clic sobre una celda del tablero. Controla la selección de piezas (jugador o pelota), verifica movimientos válidos, detecta goles y cambia el turno.

Parámetros:

- `GtkButton *button`: Botón del tablero que fue clicado.
- `gpointer user_data`: Información de la posición de la celda en el tablero, codificada como un entero.

Comportamiento:

1. Determina la fila y columna clicada.
2. Controla la selección inicial de jugador o pelota.
3. Valida y realiza movimientos.
4. Verifica si se anotó un gol.
5. Resetea la selección, actualiza el turno y redibuja el tablero gráfico.

---

2. `void cambiaTurno(void)`

Descripción:
Cambia el turno actual entre los jugadores y actualiza el mensaje correspondiente en la interfaz.

Parámetros:
Ninguno.

Comportamiento:

- Cambia el valor de la variable global `turno`.
- Actualiza la etiqueta en la interfaz para mostrar el turno actual.

---

3. `void graficarTableroEnGrid(GtkGrid *grid, char cancha[tamY][tamX])`

Descripción:
Dibuja el tablero de juego en el contenedor gráfico `GtkGrid`.

Parámetros:

- `GtkGrid *grid`: Contenedor donde se dibuja el tablero.
- `char cancha[tamY][tamX]`: Matriz que representa el estado actual del tablero.

Comportamiento:

- Limpia el contenido actual del grid.
- Crea botones para cada celda y los posiciona en el grid.
- Asigna estilos CSS según el contenido de la celda.

---

4. `void mostrarVentanaConfiguracion(GtkButton *button, gpointer user_data)`

Descripción:
Muestra la ventana de configuración para iniciar o ajustar el juego.

Parámetros:

- `GtkButton *button`: Botón que activa la ventana.
- `gpointer user_data`: Datos adicionales para el callback.

Comportamiento:

- Recupera la ventana de configuración desde el archivo Glade.
- Muestra la ventana.

---

5. `void aceptarConfiguracion(GtkButton *button, gpointer user_data)`

Descripción:
Guarda y valida las configuraciones ingresadas por el usuario, como nombres de jugadores, color de equipos y jugador inicial.

Parámetros:

- `GtkButton *button`: Botón que confirma las configuraciones.
- `gpointer user_data`: Builder GTK para acceder a los elementos de la interfaz.

Comportamiento:

1. Valida que los campos no estén vacíos.
2. Configura los nombres y preferencias de los jugadores.
3. Actualiza etiquetas en la interfaz principal.
4. Oculta la ventana de configuración.

---

6. `void iniciarJuego(GtkBuilder *builder)`

Descripción:
Inicializa el tablero, reinicia puntajes y configura el turno inicial.

Parámetros:

- `GtkBuilder *builder`: Builder GTK para acceder a los elementos de la interfaz.

Comportamiento:

- Inicializa las estructuras del tablero y jugadores.

- Actualiza los puntajes y el turno en la interfaz.
- Dibuja el tablero gráfico.

---

7. `void on_iniciar_partida_clicked(GtkButton *button, gpointer user_data)`

Descripción:
Callback para iniciar una nueva partida, mostrando la ventana de configuración.

Parámetros:

- `GtkButton *button`: Botón que inicia la partida.
- `gpointer user_data`: Builder GTK para acceder a la ventana de configuración.

Comportamiento:

- Muestra la ventana de configuración al usuario.

---

8. `void on_config_ventana_closed(GtkWidget *widget, gpointer user_data)`

Descripción:
Callback para manejar el cierre de la ventana de configuración. Actualiza las etiquetas de configuración en la ventana principal.

Parámetros:

- `GtkWidget *widget`: Ventana de configuración que se cierra.
- `gpointer user_data`: Builder GTK para acceder a elementos de la interfaz.

Comportamiento:

- Recupera las configuraciones seleccionadas.
- Actualiza las etiquetas de la ventana principal con los valores configurados.

---

9. `void on_boton_aceptar_clicked(GtkButton *button, gpointer user_data)`

Descripción:
Callback para el botón de aceptar en la ventana de configuración. Valida los campos y guarda las configuraciones.

Parámetros:

- `GtkButton *button`: Botón de aceptar.
- `gpointer user_data`: Builder GTK para acceder a los elementos de la interfaz.

Comportamiento:

- Valida nombres, color y jugador inicial.
- Cierra la ventana de configuración si todo es válido.
- Actualiza las etiquetas en la ventana principal.

---

10. `static void mostrar_acerca(GtkWidget *widget, gpointer data)`

Descripción:
Muestra el cuadro de diálogo "Acerca de".

Parámetros:

- `GtkWidget *widget`: Widget que activa el cuadro de diálogo.
- `gpointer data`: Datos adicionales.

Comportamiento:

- Muestra y oculta el cuadro de diálogo.

---

11. `static void mostrar_estadisticas(GtkWidget *widget, gpointer data)`

Descripción:
Muestra un cuadro de diálogo con las estadísticas de los jugadores.

Parámetros:

- `GtkWidget *widget`: Widget que activa la ventana.
- `gpointer data`: Datos adicionales.

Comportamiento:

- Lee las estadísticas de un archivo binario.
- Actualiza un `GtkTextView` con la información leída.
- Muestra el cuadro de diálogo con las estadísticas.

---

12. `void mostrarEstadisticas(GtkButton *button, gpointer user_data)`

Descripción:
Callback para mostrar estadísticas desde un archivo.

Parámetros:

- `GtkButton *button`: Botón que activa la visualización.
- `gpointer user_data`: Datos adicionales para el callback.

Comportamiento:

- Verifica si el archivo de estadísticas existe.
- Muestra un mensaje si no se encuentran estadísticas.
- Si existen, las despliega en la interfaz.

Estrategia:

La inteligencia artificial emplea una estrategia basada en la proximidad y la simplicidad del movimiento hacia la pelota. Esto asegura que el equipo siempre intente acercarse al balón de manera eficiente, sin movimientos innecesarios ni decisiones aleatorias que puedan perjudicar el progreso del juego.

Selección estratégica del jugador

Elige siempre al jugador que pueda realizar la acción más efectiva (el más cercano al balón).

A pesar de la estrategia, el movimiento es directo y no requiere cálculos complejos. Esto es adecuado para el contexto de un juego sencillo y garantiza que la jugabilidad sea predecible y justa.

Puntos débiles del programa

Falta conectar bien los callbacks