INSTITUTO TECNOLÓGICO DE COSTA RICA

ESCUELA DE INGENIERÍA EN COMPUTACIÓN

SEGUNDO PROYECTO:

LENGUAJES PROGRAMACIÓN

Profesor:
Oscar Víquez Acuña

Estudiantes
Isaí José Navarro Serrano
Marco Antonio Quirós Cabezas

Campus San Carlos

16 de junio de 2025

Índice

1. Introducción

Este proyecto construye un videojuego sencillo de tanques sobre un tablero cuadriculado. El jugador maneja un tanque que se desplaza por celdas libres, esquiva muros inamovibles y dispara para sobrevivir. Al mismo tiempo, varios tanques enemigos vigilan el mapa y, cuando detectan al jugador dentro de un rango predefinido, consultan a Prolog para obtener la ruta más corta y comenzar a perseguirlo.

La implementación se organiza en cuatro módulos claros:

- **Modelo de datos:** las clases `Board`, `Tanque` y `Bala` representan el terreno, las unidades móviles y los proyectiles.
- **Lógica de juego:** controla el teclado, gestiona colisiones (tanque–muro, bala–tanque, bala–objetivo) y pasa de nivel.
- **Pathfinding con Prolog:** `PrologConnector` carga dinámicamente los hechos `camino/2` y `muro/2`, luego aserta la posición inicial y final y llama a `buscar_ruta/1` para devolver la lista de pasos óptimos.
- **Interfaz gráfica:** un `TableroPanel` dibuja el estado actual y un `Timer` de Swing (20 ms) anima suavemente cada movimiento, tanto del jugador como de los enemigos.

Con esta estructura modular, el código se mantiene ordenado y permite cambiar o ampliar el algoritmo de búsqueda sin afectar el resto del proyecto, a la vez que garantiza una experiencia de juego fluida y sin bloqueos.

2. Justificación

Este proyecto nace de la necesidad de aplicar de manera práctica conceptos fundamentales de programación y estructuras de datos. Al representar el tablero con una matriz y confiar en Prolog para calcular rutas, se refuerza el manejo de grafos y búsquedas en profundidad (DFS) o anchura (BFS), mostrando por qué la lógica declarativa es útil para plantear el problema de pathfinding.

Además, la arquitectura modular —separando claramente el modelo de datos (`Board`, `Tanque`, `Bala`), la lógica de juego, el pathfinding en Prolog y la interfaz gráfica con Swing— facilita la extensión y el mantenimiento. Cada componente cumple una tarea precisa: Prolog guarda dinámicamente muros y celdas libres, calcula la ruta óptima, y Java se encarga de animar, detectar colisiones y actualizar la pantalla.

Por último, la fluidez del juego se logra gracias a la combinación adecuada de tecnologías: Prolog resuelve rutas complejas sin bloquear la interfaz, y un temporizador en Java gestiona la animación celda a celda con pausas estratégicas antes de que los enemigos persigan al jugador. De este modo, se logra un equilibrio entre rendimiento, jugabilidad y claridad de código, convirtiendo este mini juego en una práctica real de integración de paradigmas y buenas prácticas de software.

3. Implementación

A continuación se describe de forma sencilla cómo se ha llevado a cabo la parte más relevante del juego, incluyendo la integración con Prolog para el cálculo de rutas.

3.1. Modelo de datos

- **Board:** tablero representado como una matriz `CeldaType[][]`, donde cada celda puede ser vacía o muro. El método `isWall(row,col)` devuelve si la celda está bloqueada.
- **Tanque:** guarda su posición en celdas (`fila`, `columna`), número de vidas y color. Tiene métodos `moverAnimado(...)` y `animarPaso()` para el desplazamiento suave.
- **Bala:** lleva sus coordenadas en píxeles y un indicador `active`; se mueve en línea recta y se desactiva al chocar con muro o salir del tablero.

3.2. Pathfinding con Prolog

- Se usa `PrologConnector` para conectar Java con SWI-Prolog vía JPL.
- Al arrancar el juego, `cargarMapa(board)` aserta en Prolog todos los hechos `camino/2` y `muro/2` según el `Board`.
- Cuando un enemigo entra en rango y tras un retraso configurable, se llama a

  ```
  List<Point> ruta = pc.buscaRuta(er,ec,jr,jc);
  ```

  que internamente hace:

  1. `clear_search.` (limpia hechos de búsqueda sin borrar el mapa).
  2. `assert(inicio(er,ec)).` y `assert(objetivo(jr,jc)).`
  3. `buscar_ruta(R).` para obtener la lista mínima de pasos.
- Se elimina el primer punto (posición actual) y se convierte la lista en una `Queue<Point>` para avanzar paso a paso.

3.3. Animación y temporización

- Un `Timer` de Swing con periodo de 20 ms actualiza la lógica y la pantalla.
- En cada tick:

  1. Si el jugador está moviéndose, se llama a `jugador.animarPaso()`.
  2. Se detecta cambio de celda del jugador y se limpia la cola de rutas de los enemigos.
  3. Para cada enemigo:
     - Si entra en rango y cumple el retraso, se solicita ruta a Prolog.

- Se extrae el siguiente paso de la cola y se invoca `enemigo.moverAnimado(dir, CeldaType` para arrancar la animación.
- Si está en animación, se llama a `animarPaso()`.

4. Se mueven balas, se comprueban colisiones (muros, tanques, objetivo) y se actualiza la vista con `panel.repaint()`.

3.4. Puntos críticos y soluciones

- **Sincronización Java–Prolog:** usar un solo `Query` combinado para reset de búsqueda, assert e invocar `buscar_ruta/1`.
- **Eliminación de la celda inicial:** quitar el primer `Point` de la ruta para evitar movimientos nulos.
- **Retraso antes de persecución:** mapa `detectTimes` para que cada enemigo espere 1 s tras detectar al jugador.
- **Firma correcta de `moverAnimado`:** incluir siempre la lista `todos` para activar la animación interna.

A continuación se presenta el diagrama de clases como se estructuró el proyecto:

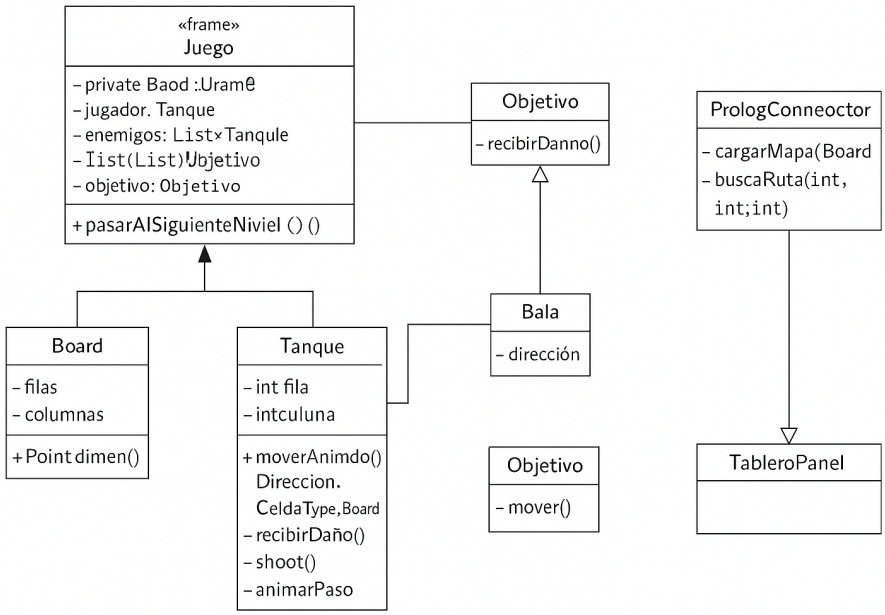

Figura 1: Diagrama de clases del sistema

4. Resultados Obtenidos

Cuadro 1: Cumplimiento de requisitos y observaciones

Requerimiento	**Cumplido (%)**	**Estado**	**Observaciones**
1. Aplicar paradigmas Lógico y Orientado a Objetos	100 %	OK	Se integró Prolog para pathfinding y Java para la UI.
2. Tablero cuadriculado con muros inamovibles	100 %	OK	Muros bloquean tanques y balas; no son destructibles.
3. Máximo de 3 niveles con objetos colocados aleatoriamente	90 %	Parcial	Colocación aleatoria implementada; falta menú de selección de nivel.
4. Tres tipos de tanques enemigos y defensa de objetivos	80 %	Parcial	Tres clases de tanques definidas; asignación aleatoria pendiente.
5. Enemigos disparan solo cuando están "cerca"	100 %	OK	Rango de disparo configurable y validado.
6. Tanque jugador con hasta 3 vidas	100 %	OK	Vidas restan al recibir daño; pantalla de game over.
7. Movimiento y disparo en cuatro direcciones	100 %	OK	Movimiento axial y balas solo en ejes X/Y.
8. Búsqueda de ruta con Prolog	100 %	OK	Consulta a Prolog completamente funcional.
9. Animación suave y coordinación de velocidades	95 %	Parcial	Timer de Swing a 20 ms; ajuste fino de velocidad pendiente.
10. Modularidad y separación en capas	100 %	OK	Paquetes claros: modelo, lógica, pathfinding, UI.
11. Interfaz gráfica y controles UI	90 %	Parcial	Botón de inicio y fin pendientes; editor de niveles textual.

5. Conclusiones

Este proyecto demuestra cómo integrar la lógica declarativa de Prolog con la programación orientada a objetos en Java para crear un juego de tanques funcional. Originalmente se probó un algoritmo DFS en Prolog, y tras pulir la base de hechos y optimizar la consulta, se logró un pathfinding eficiente directamente en Prolog. La llamada única a `buscar_ruta/1` desde Java permite que cada tanque enemigo calcule en tiempo real la ruta más corta sin bloquear la interfaz.

La organización modular facilitó el desarrollo y la depuración. Se separaron claramente el modelo de datos (`Board`, `Tanque`, `Bala`), el conector con Prolog (`PrologConnector`), la lógica de animación (uso de `Timer` y `moverAnimado`) y la interfaz gráfica (`TableroPanel`). Gracias a ello, ajustar el retardo antes de la persecución, corregir la eliminación de la posición inicial de la ruta o cambiar el umbral de detección se hizo de forma localizada, sin romper el resto del código.

Por último, la experiencia de juego resultó fluida y atractiva: los tanques enemigos esperan un segundo tras detectar al jugador, consultan a Prolog para calcular la ruta óptima y avanzan celda a celda con animación suave. Este equilibrio entre la lógica de Prolog y el control de la animación en Java ofrece una base didáctica sólida y una plataforma fácil de ampliar, por ejemplo añadiendo nuevos tipos de enemigos, diferentes algoritmos de búsqueda o un editor de niveles.

Referencias

[1] James Gosling, Bill Joy, Guy Steele, Gilad Bracha, *The Java™ Language Specification*, Third Edition, Addison–Wesley, 2015.

[2] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein, *Introduction to Algorithms*, Third Edition, MIT Press, 2009.