FACULTAD DE
INGENIERÍA Y CIENCIAS

UNIVERSIDAD ADOLFO IBÁÑEZ

Proyecto:
Atrapar al Gato (Backend)

Lenguaje y Paradigmas de Programación

Alumnos: Ignacio León
Alonso Tamayo
Filomena Tejeda

Profesor: Justo Vargas
Fecha: 23 de junio del 2025

1 Introducción

Este proyecto tiene como propósito aplicar los conocimientos adquiridos en el curso de Lenguaje y Paradigmas, mediante el desarrollo de una aplicación web interactiva utilizando Java como lenguaje de programación y combinando tres paradigmas: procedural, orientado a objetos y funcional. El desafío consiste en construir un backend capaz de gestionar la lógica completa del juego “Atrapar al Gato”, permitiendo la interacción con un frontend web a través de una API REST.

El juego simula un tablero hexagonal en el cual un gato intenta escapar hacia el borde. El usuario, en cada turno, debe bloquear celdas del tablero para impedir su escape. El gato, por su parte, se mueve automáticamente usando una estrategia de IA basada en algoritmos de búsqueda A*, para encontrar el camino óptimo hacia el borde. La partida concluye cuando el gato es atrapado o logra escapar.

El backend fue diseñado de forma modular, siguiendo las buenas prácticas de desarrollo, y organizado en paquetes con responsabilidades claras: controladores REST, servicios de juego, repositorios de datos y estrategias de movimiento. Esta arquitectura permite una integración fluida con el frontend proporcionado, el cual se ejecuta en el navegador accediendo a los endpoints definidos.

El juego puede ejecutarse localmente compilando el proyecto con Maven e iniciando la aplicación mediante Spring Boot.

```
mvn spring-boot:run
```

Una vez ejecutado, se accede a la interfaz de juego desde un navegador web en `http://localhost:8080`, donde el usuario puede iniciar partidas y controlar la interacción con el tablero.

Además de cumplir con los requisitos técnicos del enunciado, este proyecto buscó fortalecer habilidades clave en el desarrollo de software: el uso de distintos paradigmas de programación en un mismo proyecto, el diseño limpio de clases y entidades, el uso de patrones de diseño como Strategy y Template Method, y la capacidad de estructurar un sistema complejo en capas funcionales bien definidas. Asimismo, representó una oportunidad para practicar la integración entre frontend y backend, la implementación de algoritmos de búsqueda, y la aplicación de principios de programación funcional en Java.

2. Objetivos

El proyecto tiene como finalidad principal desarrollar la lógica del backend, proyectando el movimiento del jugador, la lógica de juego, el tablero y la estrategia del gato, utilizando el **paradigma procedural, orientado a objetos y funcional** en el desarrollo de una solución de la lógica de jugadas y turnos, el diseño del tablero, juego y entidades y la estrategia del gato y análisis algorítmico, utilizando el lenguaje **Java**. A través del diseño y construcción de la aplicación *Atrapar al Gato*, se busca resolver un problema de lógica de un juego, el movimiento del jugador, el tablero y la estrategia del Gato.

Los objetivos específicos del proyecto son los siguientes:

1. **Comprender y aplicar los principios del paradigma procedural**, en el uso adecuado de estructuras de datos y control de flujo, como la lógica de turnos y validación de jugadas.
2. **Comprender y aplicar los principios del paradigma orientado a objetos**, con la representación de entidades, encapsulamiento y uso de colecciones, como List o Map, para el modelado del estado del juego, el tablero, el jugador, y el gato.
3. **Comprender y aplicar los principios del paradigma funcional**, tales como el uso de funciones puras, recursión, estructuras inmutables y ausencia de efectos secundarios.
4. **Diseñar e implementar un sistema modular en Java**, capaz de procesar el tablero, los turnos del jugador y el Gato, más la estrategia utilizada por este.
5. **Integrar un algoritmo de búsqueda informada (A*)**, adaptado a las restricciones del problema, priorizando recorridos energéticamente eficientes.
6. **Desarrollar una interfaz textual interactiva**, que permita al usuario visualizar el tablero, top 10 de partidas jugadas, reglas del juego y los turnos de cada participante (jugador y Gato).
7. **Fortalecer competencias en herramientas de apoyo al desarrollo**, tales como Git, control de versiones colaborativo y documentación del código.
8. **Fomentar el trabajo colaborativo y la organización modular del código fuente**, favoreciendo la mantenibilidad y la escalabilidad de la solución propuesta.

3. Descripción del problema

El presente proyecto tiene como propósito el desarrollo de una aplicación funcional en Java denominada **"Atrapar al Gato"**, cuyo objetivo central es desarrollar la lógica del backend, orquestando el movimiento del jugador, la lógica de juego, el tablero y la estrategia del gato, utilizando los tres paradigmas vistos en clase. El Gato empieza en el centro del tablero y el jugador tiene el primer turno para bloquear una celda tablero, luego el Gato tiene su turno de movimiento, en el que se mueve en la dirección más corta y alejada de una casilla bloqueada para llegar antes a la frontera del tablero de juego. La idea del juego es atrapar al Gato y no dejar que se escape.

Para abordar este problema, se requiere aplicar una estrategia de búsqueda informada que evalúe múltiples rutas posibles y determine aquella que minimice el camino para que el gato se escape, respetando las reglas impuestas por el entorno. La solución debe implementarse bajo un enfoque procedural, orientado a objetos y funcional.

4. Diseño y arquitectura del sistema

Para abordar el desarrollo del juego “Atrapar al Gato” de forma escalable y modular, se diseñó la aplicación con una arquitectura en capas, en la cual cada componente del sistema tiene una responsabilidad claramente definida. Esta decisión fue clave para aplicar los principios de separación de responsabilidades, mejorar la mantenibilidad del código y facilitar la integración con el frontend provisto.

Estructura general del sistema

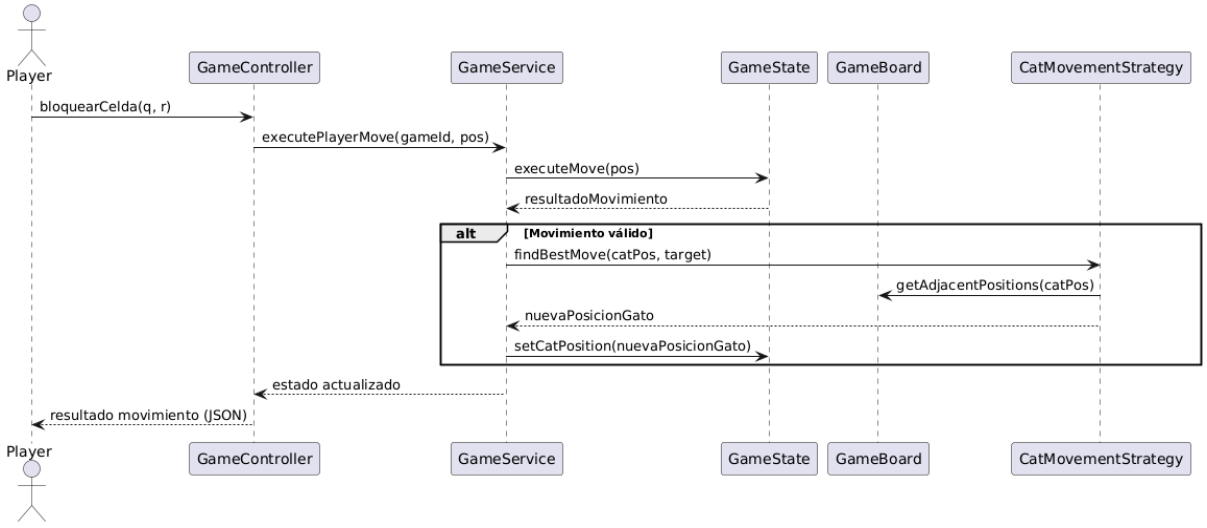
El sistema se divide en varios módulos principales, cada uno con responsabilidades específicas que contribuyen a su funcionalidad general:

```
atrapar-al-gato/
│
├── controller/
│   └── GameController.java       -- Controlador REST: expone endpoints para el frontend
│
├── impl/
│   ├── model/
│   │   ├── HexGameBoard.java     -- Modelo del tablero hexagonal
│   │   ├── HexGameState.java     -- Estado actual del juego (posiciones, estado del gato)
│   │   └── HexPosition.java      -- Representación de una celda en el tablero
│   │
│   ├── strategy/
│   │   ├── AStarCatMovement.java  -- Estrategia inteligente (A*) para movimiento del gato
│   │   └── BFSCatMovement.java    -- Estrategia BFS para movimiento alternativo del gato
│   │
│   ├── repository/
│   │   └── H2GameRepository.java  -- Repositorio persistente en H2
│   │
│   └── service/
│       └── HexGameService.java    -- Orquestador principal: maneja la lógica de juego
│
└── resources/static/
    ├── index.html              -- Frontend (vista del juego)
    ├── styles.css              -- Estilos
    └── game.js                 -- Lógica del cliente
```

- **GameController.java**: Punto de entrada del backend. Expone los endpoints REST que permiten al frontend iniciar partidas, bloquear celdas y consultar el estado del juego. Separa claramente la lógica de presentación de la lógica de negocio.
- **HexGameService.java**: Servicio central que orquesta el flujo del juego. Coordina el tablero, la estrategia del gato, la validación de jugadas y el repositorio de datos. Actúa como puente entre el controlador y las capas internas del sistema.
- **HexGameBoard.java**: Representa el tablero de juego. Implementa funciones para verificar celdas válidas, adyacencia entre posiciones, y registrar el bloqueo de celdas por parte del jugador.

- **HexGameState.java**: Modelo que mantiene el estado completo de la partida: posiciones bloqueadas, posición actual del gato, estado de la partida (victoria/derrota) y estadísticas del juego.
- **HexPosition.java**: Clase que modela una celda individual del tablero. Incluye operaciones geométricas para calcular distancias hexagonales y posiciones vecinas.
- **AStarCatMovement.java / BFSCatMovement.java**: Estrategias de inteligencia artificial para el movimiento del gato. Implementan algoritmos de búsqueda avanzados para encontrar el mejor camino hacia el borde del tablero.
  H2GameRepository.java: Repositorio que permite la persistencia de partidas en la base de datos en memoria H2. Utiliza programación funcional (Streams, Predicate, Function) para realizar consultas y transformaciones de datos.
- **Frontend (index.html, game.js)**: Interfaz gráfica que permite al usuario interactuar con el juego a través de la web. Se comunica con el backend mediante llamadas REST.

Diagrama de Flujo



Lógica algoritmo de búsqueda

El módulo **AStarCatMovement.java** implementa la lógica del algoritmo de búsqueda A*, utilizado para calcular el camino óptimo que el gato debe seguir para alcanzar el borde del tablero.

La heurística empleada combina dos factores:

- La **distancia al borde más cercano** (calculada en `getDistanceToBorder`).
- Un **peso asociado a las posiciones bloqueadas cercanas**, que favorece los caminos con menor congestión de obstáculos.

La prioridad de cada nodo en la búsqueda se determina mediante un **fScore = gScore + heurística**, lo que permite al algoritmo seleccionar en cada iteración el nodo que aparenta ser más prometedor en función de su costo acumulado y su proximidad al objetivo.

El algoritmo implementa una cola de prioridad (`PriorityQueue`) que contiene los nodos en el "open set" (nodos por explorar), así como un conjunto de posiciones ya visitadas ("closed set"). La expansión de nodos se realiza mediante funciones funcionales (`Predicate`, `Function`), asegurando una estructura inmutable y evitando efectos secundarios.

Entre los métodos más relevantes se destacan:

- `getPossibleMoves()`: Calcula las celdas adyacentes válidas (no bloqueadas).
- `getHeuristicFunction()`: Define la heurística de búsqueda combinando distancia al borde y obstáculos.
- `getGoalPredicate()`: Define el criterio de objetivo (alcanzar el borde del tablero).
- `getMoveCost()`: Costo uniforme por cada movimiento.
- `getFullPath()`: Ejecuta el algoritmo A*, generando el camino completo desde la posición actual del gato hasta una celda en el borde.

El camino resultante es reconstruido mediante `reconstructPath()`, partiendo del nodo objetivo alcanzado

6. Implementación por Paradigma

6.1 Paradigma Procedural

El paradigma procedural se aplicó principalmente en la **gestión de la lógica de turnos** y **flujo del juego**, organizando el comportamiento en métodos que ejecutan procesos secuenciales: Métodos como:

- `canExecuteMove()` y `performMove()` en `HexGameState` definen la validación y ejecución de jugadas de forma procedural.

- `updateGameStatus()` revisa secuencialmente las condiciones de victoria o derrota.

- El flujo de turnos (jugador - IA) sigue una estructura procedural definida en el servicio `HexGameService`.

El uso de este paradigma permitió que la lógica de cada turno sea **clara, controlada y fácil de seguir**, facilitando además la validación de condiciones antes y después de cada jugada.

6.2 Paradigma Orientado a Objetos (OOP)

El paradigma orientado a objetos fue el pilar central de la estructura del sistema, aplicándose en el modelado de todas las entidades del juego:

Entidades principales:

- `HexGameBoard`: Modela el tablero hexagonal, encapsula operaciones sobre posiciones.
- `HexGameState`: Encapsula el estado completo del juego, incluyendo la posición del gato, las celdas bloqueadas, el progreso y las estadísticas.
- `HexPosition`: Modelo inmutable de una celda, con operaciones geométricas propias.
- AStarCatMovement y BFSCatMovement: Implementan diferentes estrategias de movimiento, aplicando herencia y polimorfismo desde `CatMovementStrategy.`

Patrones aplicados:

- **Herencia**: `HexPosition` hereda de Position, especializando comportamiento.
- **Polimorfismo**: Distintas estrategias de movimiento implementan la misma interfaz.
- **Encapsulamiento**: Los atributos son privados y se accede a ellos mediante getters/setters donde corresponde.
- **Patrón Strategy**: Aplicado en la selección dinámica de la estrategia de movimiento del gato.

El uso de OOP favoreció un diseño **modular, reutilizable y extensible**, permitiendo agregar futuras variantes de IA o reglas de juego sin afectar la arquitectura base.

6.3 Paradigma Funcional

El paradigma funcional se empleó principalmente en las **estrategias de movimiento del gato** y en la manipulación de colecciones dentro de los repositorios y el tablero:

Uso de funciones de orden superior:

- Métodos como `getPositionsWhere(Predicate<HexPosition> condition)` en HexGameBoard permiten filtrar dinámicamente las posiciones disponibles.
- getHeuristicFunction() en AStarCatMovement devuelve funciones puras que encapsulan la heurística de evaluación de cada celda.

Uso de Streams y programación funcional:

- AStarCatMovement y BFSCatMovement utilizan **Streams** para recorrer y evaluar posiciones de forma funcional.
- Uso de **Predicate** para definir filtros.

- Uso de **Function** para encapsular operaciones matemáticas y heurísticas.
- Uso de estructuras inmutables y operaciones sin efectos secundarios.

7. Reflexiones

El desarrollo de este proyecto representó una oportunidad valiosa para aplicar de forma práctica los contenidos del curso, enfrentándonos a un problema que requería integrar distintos paradigmas de programación en un sistema completo.

Durante la implementación, aprendimos que el diseño de una arquitectura modular y clara facilita no solo la mantenibilidad del código, sino también su extensibilidad. Separar correctamente las responsabilidades en componentes como el tablero, el estado del juego y las estrategias de movimiento, permitió que cada parte pudiera evolucionar o corregirse sin afectar el resto del sistema.

Además, trabajar con algoritmos como **A*** y **BFS** reforzó nuestra comprensión de técnicas de búsqueda y optimización. Aplicar programación funcional en estos contextos, mediante el uso de **Streams**, **Predicate** y **Function**, nos permitió escribir código más expresivo y más fácil de razonar.

Uno de los desafíos más importantes fue lograr la correcta integración entre el backend y el frontend, garantizando que los endpoints REST ofrecieran la información esperada en todo momento. Esto nos enseñó la importancia de diseñar APIs coherentes y robustas.

Finalmente, la implementación de la base de datos en memoria (**H2**) nos permitió experimentar con operaciones CRUD en un contexto real, entendiendo mejor cómo diseñar repositorios y cómo aplicar programación funcional en consultas de datos.

En conclusión, el proyecto no solo nos permitió cumplir con los requisitos técnicos establecidos, sino que también fortaleció nuestras habilidades en diseño de software, uso de patrones, integración de múltiples paradigmas y desarrollo de aplicaciones web robustas y escalables.

8. Uso de IA: Transparencia y Validación

Durante el desarrollo del proyecto se utilizó **ChatGPT como herramienta de apoyo complementaria**, especialmente frente a los desafíos propios del lenguaje Java. Dado que se trata de un lenguaje altamente sensible al formato —donde una indentación incorrecta, un tab mal ubicado o un salto de línea inesperado pueden generar errores difíciles de rastrear—, la asistencia de IA fue clave para comprender cómo estructurar correctamente el código y solucionar errores sintácticos complejos.

La inteligencia artificial fue empleada para:

- Comprender de forma más profunda el funcionamiento del algoritmo A* en un entorno funcional.
- Obtener ejemplos prácticos de consultas SQL.
- Identificar errores por formato o mala indentación que impedían la compilación.
- Redactar partes del informe con mayor claridad y cohesión.

En todo momento, las respuestas proporcionadas por la IA fueron **analizadas críticamente y adaptadas** por los integrantes del equipo, asegurando que cada fragmento implementado fuese comprendido y estuviera alineado con los requisitos del enunciado. No se copiaron soluciones sin entendimiento previo, respetando así los principios éticos del aprendizaje autónomo y el propósito formativo del proyecto. Además estas respuestas se corroboraron con documentación y foros.