Guía Técnica - Proyecto Atrapar al Gato

¿Qué se les entrega?

El punto de partida del proyecto es una aplicación web que ya incluye:

- Una interfaz de usuario (frontend) funcional con tablero visual.
- Un controlador REST base: `GameController`.
- Dos caminos de implementación: uno de ejemplo (funcional) y otro vacío para que ustedes lo completen.

⚙ Activación de la implementación

La clase `GameController` decide a qué lógica conectar según la propiedad:

```
game.use-example-implementation=true
```

Para activar la lógica de ustedes, deben cambiarla a:

```
game.use-example-implementation=false
```

Esto activará los métodos como `startGameWithStudentImplementation()` y el uso del paquete `impl`.

¿Qué deben construir?

Deben completar la lógica del juego en las clases del paquete `impl`, incluyendo:

- **HexGameService:** lógica principal del juego.
- **HexGameState:** estado de la partida (posición del gato, turnos, celdas bloqueadas).

- **HexGameBoard:** tablero hexagonal real.
- **CatMovementStrategy:** movimiento del gato (A*, BFS, etc.).
- **H2GameRepository:** persistencia usando base de datos.

Flujo de la partida (Ejemplo Actual)

1. El frontend llama a `/api/game/start`.
2. `GameController` decide si usa lógica de ejemplo o del estudiante.
3. Se crea el estado del juego y se guarda en memoria.
4. El jugador bloquea una celda → se actualiza el tablero y se mueve el gato.
5. El estado actualizado se devuelve al frontend para seguir la partida.

```
Frontend (JS/HTML)
      |
      v
GameController (REST)
      |
      v
ExampleGameService  <---->  InMemoryGameRepository
      |
      v
ExampleGameState <----> ExampleGameBoard
      |
      v
SimpleCatMovement
```

Arquitectura Base

Clase	Responsabilidad
GameController	Controlador REST. Decide qué lógica usar y expone los endpoints.
GameService	Orquestador. Maneja inicio, jugadas, y lógica general.

GameState	Contiene el estado del juego: posiciones, bloqueos, turnos.
GameBoard	Define el tablero y lógica de adyacencia, bloqueos, etc.
CatMovementStrategy	Define cómo se mueve el gato (aleatorio, BFS, A*).
DataRepository	Abstracción para guardar y recuperar partidas.

Consejos para empezar

- Empiecen implementando el movimiento del gato con lógica simple y luego mejoren con A* o BFS.
- El estado debe ser persistente entre llamadas, por eso deben implementar `H2GameRepository`.
- Apóyense en clases genéricas como `GameBoard<T>` y `GameState<T>`.

Lógica detallada y clases clave del backend

1. **Punto de partida: GameController**
   Clase: `GameController` (com.atraparalagato.controller.GameController)
   - Responsabilidad: Es el controlador REST principal.
   - Expone los endpoints HTTP para el frontend (ej: `/api/game/start`, `/api/game/block`, `/api/game/state/{gameId}`).
   - Decisión de implementación: Según la propiedad `game.use-example-implementation` en `application.properties`, decide si usa la implementación de ejemplo (`example`) o la de estudiantes (`impl`).
   - Por defecto: Usa la de ejemplo (`example`).

2. **Flujo de una partida (con la implementación de ejemplo)**
    1. **Iniciar un juego**
       Método: `startGameWithExample(int boardSize)`
       Llama a: `ExampleGameService.startNewGame(boardSize)`
       Crea: Un nuevo `ExampleGameState` y lo guarda en `InMemoryGameRepository`.
    2. **Bloquear una posición (jugada del jugador)**
       Método: `blockPositionWithExample(String gameId, HexPosition position)`
       Llama a: `ExampleGameService.executePlayerMove(gameId, position)`
       Lógica:
        - Busca el `ExampleGameState` por `gameId`.
        - Valida si el movimiento es legal.
        - Bloquea la celda en `ExampleGameBoard`.
        - Mueve al gato usando `SimpleCatMovement`.
        - Actualiza y guarda el estado del juego.
    3. **Obtener el estado del juego**
       Método: `getGameStateWithExample(String gameId)`
       Llama a: `ExampleGameService.getGameState(gameId)`
       Devuelve: Estado actual, posición del gato, celdas bloqueadas, etc.
3. **Clases principales y sus responsabilidades**

Clase	Responsabilidad
`Position`	Representa una posición en el tablero (abstracta).
`GameBoard<T extends Position>`	Lógica general del tablero (bloqueos, adyacencia, etc.).
`GameState<T extends Position>`	Estado del juego: posición del gato, estado del juego, etc.

`DataRepository<T, ID>`	Persistencia de datos (guardar, buscar, eliminar estados).
`CatMovementStrategy<T extends Position>`	Estrategia de movimiento del gato (simple, BFS, A*).
`GameService<T extends Position>`	Orquestador: iniciar partidas, ejecutar jugadas, validar.

Implementación de ejemplo (`example`):

- `ExampleGameBoard`: Tablero hexagonal simple.
- `ExampleGameState`: Gato parte en el centro, reglas básicas de victoria.
- `InMemoryGameRepository`: Guarda partidas en memoria.
- `SimpleCatMovement`: Movimiento aleatorio con preferencia por el borde.
- `ExampleGameService`: Orquesta usando las clases anteriores.

Implementación de estudiantes (`impl`):

- `HexGameBoard`, `HexGameState`, `H2GameRepository`, `AStarCatMovement`, `BFSCatMovement`, `HexGameService`.
- Son esqueletos que deben ser completados.
- Por defecto lanzan `UnsupportedOperationException` hasta que sean implementados.

4. **Resumen del flujo (ejemplo):**

```
Frontend llama a /api/game/start
        ↓
GameController decide usar ExampleGameService
        ↓
ExampleGameService crea ExampleGameState y lo guarda
        ↓
Frontend muestra el tablero y permite bloquear celdas
        ↓
Frontend llama a /api/game/block
```

```
        ↓
GameController llama a ExampleGameService.executePlayerMove
        ↓
Se ejecuta el movimiento, se mueve el gato, se guarda el estado
        ↓
Frontend actualiza el estado visual
```

5. **¿Dónde empezar si quieres extender la lógica?**
    - Implementa las clases en el paquete `impl` (HexGameService, HexGameState, etc.).
    - Cambia `game.use-example-implementation=false` en `application.properties`.
    - Completa los métodos `startGameWithStudentImplementation`, etc. en `GameController`.