Proyecto Final: Atrapar al gato

Ingeniería civil Informática, TICS200

Universidad Adolfo Ibáñez

Profesor: Justo Vargas

2025-S1

Integrantes:
Josefa Juica
Cristobal Zeballos
Fernando Flores

1. Introducción

El siguiente proyecto corresponde al desarrollo de la aplicación backend para el juego "Atrapar al Gato". El objetivo central fue poner en práctica los distintos enfoques de programación aprendidos durante el semestre, integrando el estilo de programación estructurado, orientado a objetos y funcional en un sistema desarrollado en Java.

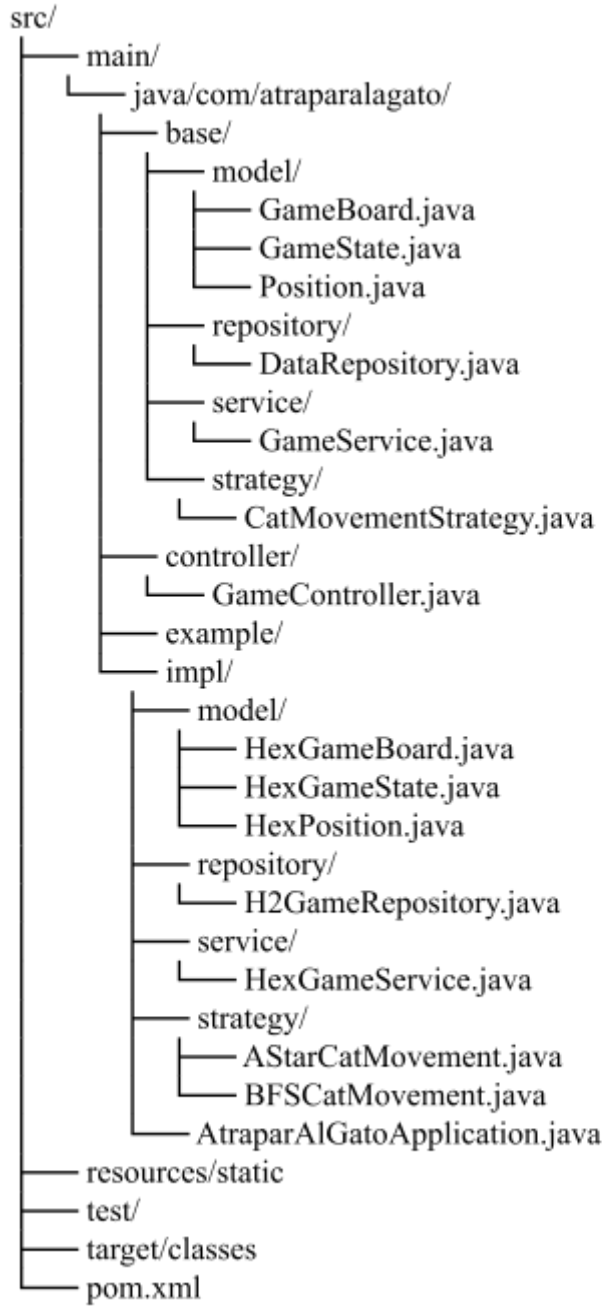
La lógica del juego se basa en un tablero hexagonal donde un gato intenta escapar hacia los bordes mientras el jugador bloquea celdas para evitar la fuga de este. El movimiento del gato está automatizado mediante una estrategia de búsqueda óptima, en nuestro caso implementada con un algoritmo BFS (Breadth-First Search), lo cual permite explorar rutas posibles de manera eficiente sin utilizar algoritmos de aproximación.

El proyecto fue desarrollado desde la base que se nos entregó, con una estructura modular que facilita su mantenimiento, extensión y testeo. Luego, nuestro trabajo se basó en optimizar la base que se nos entregó, lo cuál lo hicimos con los archivos que se encuentran en la carpeta *impl* y también modificando GameController..

Este proyecto no solo buscó satisfacer los requisitos técnicos, sino también fomentar la comprensión profunda de cómo los diferentes paradigmas aprendidos en clases pueden llegar a coexistir. Asimismo, se hizo énfasis en las buenas prácticas de diseño, incluyendo la separación de responsabilidades y el uso de patrones como Strategy para encapsular la lógica de movimiento del gato.

2. Diseño y arquitectura del sistema

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. Podemos ver la estructura en la siguiente imagen:



3. Desarrollo

Para este proyecto en particular ya se nos entregaba una parte importante del código, por lo que nuestro trabajo fue realizar el código de las clases del paquete *impl*, de esta forma optimizando el código original. Luego, las clases más importantes fueron:

- HexGameBoard.java: este representa un tablero hexagonal para el juego, y se encarga de manejar la lógica asociada con los movimientos permitidos, las posiciones bloqueadas, junto con la verificación de posiciones válidas y adyacentes.
- HexGamestate.java:esta representa el estado actual del juego (en progreso, ganado, perdido, etc.) en el contexto del tablero hexagonal.
- HexPosition.java: representa una posición en un tablero hexagonal, usando coordenadas axiales. Es esencial para mover al gato, bloquear casillas y calcular adyacencias en el juego.
- H2GameRepository.java: es una clase que implementa un repositorio de almacenamiento de partidas de juego (GameState).
- HexGameService.java: es el servicio principal se encarga de crear partidas, ejecutar movimientos del jugador y del gato, calcular estadísticas, sugerir jugadas y validar reglas.
- BFSCatMovement.java: por default nuestro código utiliza este algoritmo de búsqueda, este es de estilo Breadth-First Search, lo cual significa que busca la ruta más corta desde la posición actual del gato hasta una celda del borde del tablero. El objetivo es que el gato escape.

En nuestro código, utilizamos el algoritmo de movimiento por default BFS, ya que en HexGameService pusimos que si (if) el usuario elige “hard” como el modo de juego entonces se utiliza A*, sino (else) se utiliza BFS. Como nunca llegamos a integrar en el frontend que el usuario pudiera elegir el modo, nos quedamos con BFS.

En este algoritmo que trata de encontrar un camino desde la posición actual del gato hasta el borde del tablero, como mencionamos anteriormente, encontramos las siguientes funciones más importantes:

- getPossibleMoves(): devuelve los movimientos válidos desde una posición, es decir, aquellas celdas que no están bloqueadas.
- getNextMove(): invoca selectBestMove para determinar el próximo movimiento del gato.
- selectBestMove(): decide aleatoriamente (30% de las veces) entre caminos viables o, en su defecto, aplica BFS desde cada movimiento posible para encontrar el camino más corto al borde.

- bfsToGoal(): ejecuta BFS hasta encontrar una celda objetivo (en el borde) y retorna el camino si existe.
- reconstructPath(): reconstruye el camino desde la posición inicial hasta la meta usando el parent map construido durante la búsqueda.
- hasPathToGoal(): permite saber si existe alguna ruta disponible desde la posición actual.

Esta implementación asegura un comportamiento inteligente del gato, balanceando entre movimientos aleatorios y decisiones óptimas, lo cual incrementa la variabilidad y la dificultad del juego.

Con respecto a la programación orientada a objetos podemos decir que este fue la base de la estructura del sistema, permitiendo modelar de forma clara y extensible los distintos componentes del juego "Atrapar al Gato". Las entidades principales son las que mencionamos anteriormente, ahora solo destacaremos su función respecto a POO:

- HexGameBoard.java: hereda de GameBoard y protege sus datos internos, como las posiciones válidas y bloqueadas (validPositions, blockedPositions), usando estructuras como Set y métodos como blockPosition o getBlockedHexPositions.
- HexGamestate.java: hereda de GameState y la posición del gato, dificultad, historial de movimiento, tablero y el estado del juego están guardados dentro de la clase y solo se puede acceder o modificar a través de métodos públicos como getCatPosition(), setDifficulty(...), getMoveHistory(), etc.
- HexPosition.java: hereda de Position y define al igual que protege los datos internos (q, r) que representan coordenadas.
- H2GameRepository.java: hereda de DataRepository y guarda toda la lógica de conexión a la base de datos y serialización JSON dentro de la clase.
- HexGameService.java: hereda de GameService, toda la lógica para manejar el juego está encerrada dentro de esta clase.
- BFSCatMovement.java: hereda de CatMovementStrategy.

4. Uso de IA

Para este proyecto utilizamos la IA ChatGPT como herramienta complementaria para realizar partes del código y resolver errores de compilación. También la utilizamos para ayudar a

escribir partes del informe que involucran el código. Siempre tuvimos en cuenta que la IA se puede equivocar, por lo que siempre tuvimos en cuenta exactamente qué necesitábamos y lo que nos entregaba lo evaluamos críticamente.

5. Conclusión - Reflexiones

Este proyecto se nos hizo mucho más difícil como grupo a comparación de las aplicaciones, el enfoque de completar un código que estaba semi hecho fue muy complicado para nosotros. Esto se debe a que se nos dificultó entender el código entero y las cosas que teníamos que implementar. Más aún al hacer los códigos de los archivos de la carpeta *impl* no le dimos la importancia que le debimos haber dado a mantener la estructura dentro de estos archivos que se nos daba como base, lo cual resultó en muchos errores de compilación, errores de clase y tener que empezar desde cero varias veces.

Luego, hay cosas que pudimos lograr y otras que no. Logramos implementar uno de los algoritmos de búsqueda BFS y logramos implementar en el código (en el archivo HexGameService) que se elija entre A* o BFS (lo cual se logró conectar bien con HexGameState). Sin embargo, no logramos llevar esta elección al frontend. Tampoco logramos que se guardaran de manera correcta los movimientos totales ni la puntuación en el juego.

A pesar de esto y de haber tenido que entregar el trabajo con retraso y sin el funcionamiento total de lo que se esperaba, fue una muy buena instancia de aprender algo nuevo y perseverar para poder realizar el proyecto.