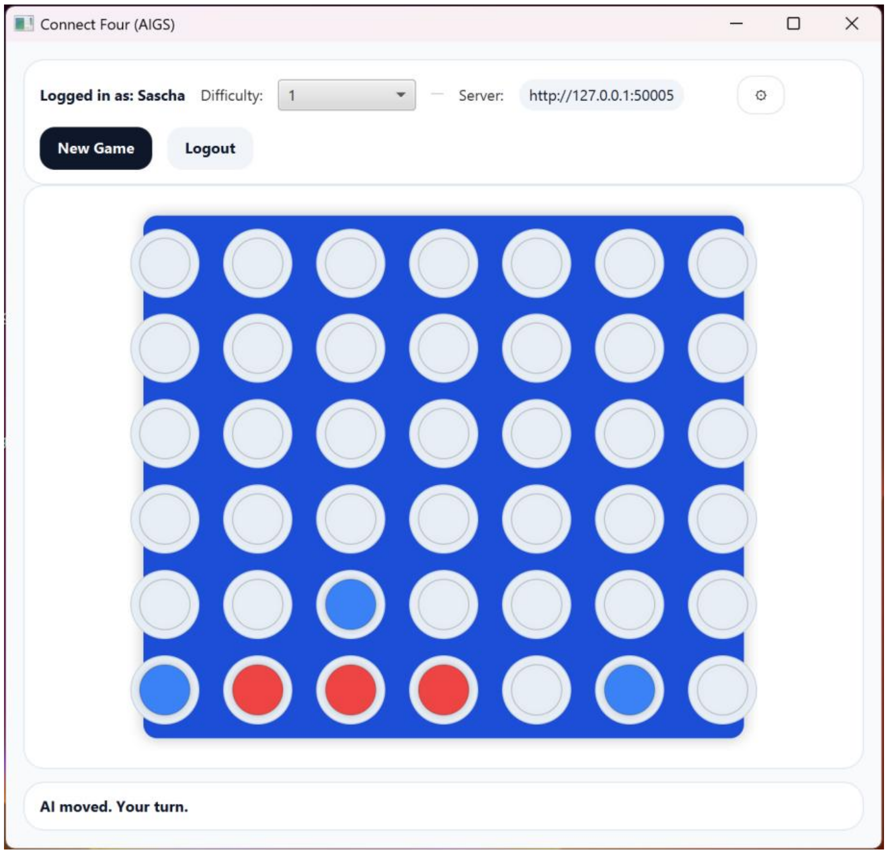
Project 3: AIGS Report

Project type: Connect Four, two player game

Author: Sascha Niederhauser & Alexander Burri
Course: 25HS Software Engineering (SEL)
Instructor: Prof. Dr. Bradley Richards



AI moved. Your turn.

1. Usage Instructions

The project consists of two separate applications that work together. A Spring Boot server that provides the REST game service, and a JavaFX client that provides the graphical user interface. Both projects were developed using IntelliJ, which handles Maven imports and runs configurations automatically.

Server Repository: https://github.com/AlexanderBurri1/aigs-spring-server-connectfour.git
Client Repository: https://github.com/AlexanderBurri1/aigs-connectfour-client.git

To run the project, both repositories have to be cloned from GitHub and opened as separate projects in IntelliJ. First, the server project is opened to import the maven configuration and download the dependencies. The server is then started by running the Spring Boot main class. Once running the sever is locally reachable.

After running the server, the client project can be opened as well in IntelliJ in the same way as before. IntelliJ again imports maven dependencies and prepares the JavaFX run configuration. The client can be started by running the MainApp class through the maven Plugin javafx:run. In the client the user can configure the server connection using the settings button. The configuration is stored locally so it remains available across restarts. From there, a user can register, log in, choose a difficulty, start a new game and play connect four by interacting with the game board.

To ensure these instructions are reliable, the project was tested on three different “fresh” computers by cloning both repositories, starting the server, starting the client and performing a complete run through including registration, login, game creation, playing and reaching the final state.

2. Design

The implementation follows a classic client-server architecture. All game logic, rules, move validation and AI decisions are handled on the server, while the client focuses on usability, visualization and communication with the API. The client does not decide the game outcome, it only displays what the server returns.

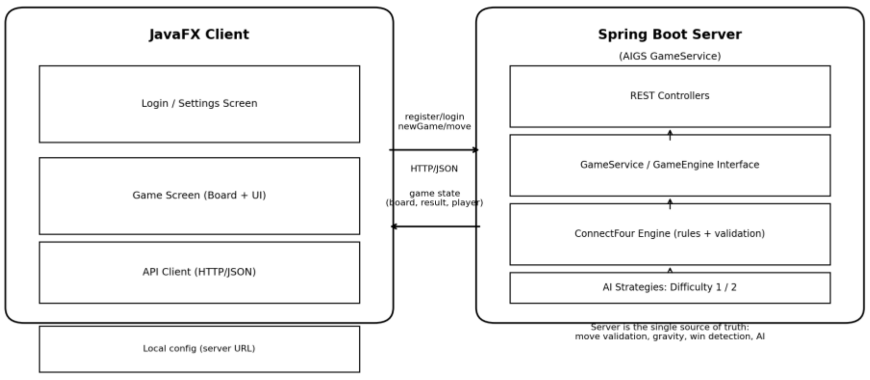

JavaFX Client
Login / Settings Screen
Game Screen (Board + UI)
API Client (HTTP/JSON)
Local config (server URL)
register/login
newGame/move
HTTP/JSON
game state
(board, result, player)
Spring Boot Server
(AIGS GameService)
REST Controllers
GameService / GameEngine Interface
ConnectFour Engine (rules + validation)
AI Strategies: Difficulty 1 / 2
Server is the single source of truth:
move validation, gravity, win detection, AI

Figure 1: Architecture overview

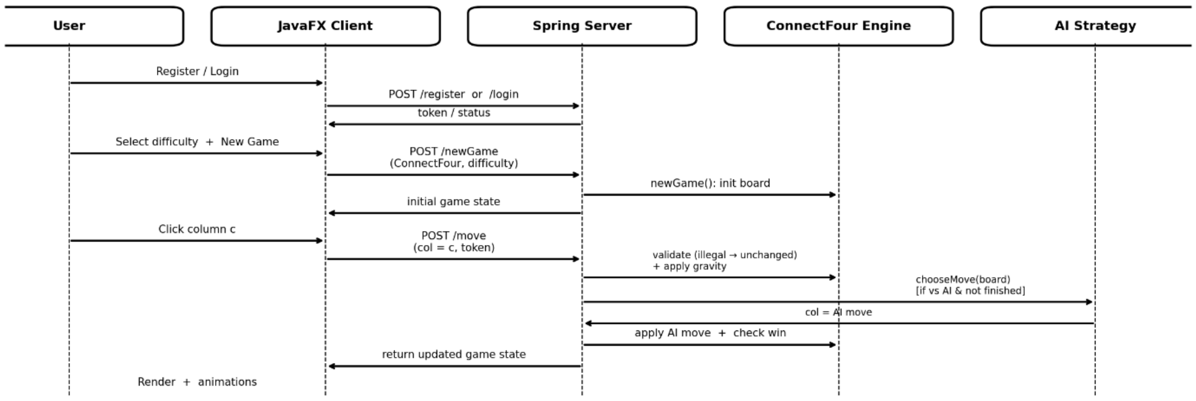

User
JavaFX Client
Spring Server
ConnectFour Engine
AI Strategy
Register / Login
POST /register or /login
token / status
Select difficulty + New Game
POST /newGame (ConnectFour, difficulty)
newGame(): init board
initial game state
Click column c
POST /move (col = c, token)
validate (illegal → unchanged) + apply gravity
chooseMove(board) [if vs AI & not finished]
col = AI move
apply AI move + check win
return updated game state
Render + animations

Figure 2: Request flow

On the server side, the project uses the existing AIGS game service structure where different games can be supported. A game type enumeration is used to distinguish games, and a common GameEngine interface defines the behaviour required for creating a new game and executing a move. The Connect Four implementation provides the concrete logic for a 6 x 7 board, including the gravity-based piece drop and the win condition detection for four connected pieces horizontally, vertically, or diagonally. The server also stores and manages the selected difficulty within the Game object when a game is created. This is important because the client only sends the difficulty once during game creation, afterwards the server continues the match and applies the selected difficulty internally for each AI response.

The difficulty setting is implemented by selecting different AI strategies. Difficulty 1 corresponds to a simpler behaviour (random or minimal strategy), whereas difficulty 2 corresponds to a stronger AI strategy. In our case, difficulty 2 is designed to be meaningfully more challenging by applying a strategy-based move selection. Because the AI runs on the server, the difficulty can be improved or adjusted without needing changes to the client.

On the client side, the JavaFX application is structured around two main screens. The first screen handles registration and login and provides access to the settings dialog. The second screen contains the actual game UI, including difficulty selection, “New Game”, logout, and the Connect Four board visualization. The board is displayed as a 6 x 7 grid of circular fields representing the slots. Player stones are rendered in red, Ai stones in blue, and empty slots remain unfilled. Interaction is intentionally straightforward so the user can simply click on a column to add a stone to the column. The client prevents invalid interaction by disabling full columns and by temporarily disabling input while a move is being processed to avoid double submissions.

To improve the user experience and make the gameplay clearer, the client includes visual feedback and animation. After a move request, the client receives the updated board from the server and compares it with the previous board to detect newly placed pieces. Those pieces are animated with a dropping effect into their target slot. Additionally, a short delay is introduced before animating the AI piece so that the user can clearly see the separation between their move and the AI’s response. When the match ends, a winner banner is shown on top of the board to clearly indicate the outcome. This is displayed as UI feedback on the final board state and end-of-game flag.

Finally, the UI was designed to remain usable under different window sizes. The layout avoids cutting off controls by using responsive containers, allowing the header to wrap and the board area to resize. Where necessary, scaling and scrolling ensure the board remains readable instead of hiding important information.