Project Proposal

Overview

Xiangqi, also known as Chinese Chess, is a very popular strategy board game in China. Like Chess, each player will control 16 pieces: 1 general, 2 advisors, 2 elephants, 2 horses, 2 chariots, 2 cannons and 5 soldiers. The player who controls red pieces moves first and the other one moves black pieces. The goal of this game is to checkmate the opponent's general.

Basic overview of the game rules in terms of things that are different from chess:

1. 90 positions on the board (more complex game trees)
2. Board is separated in the middle by river
3. Soldiers cannot be promoted
4. Elphants and advisors cannot move across the river
5. Other different rules for movement of pieces

In this project, we are going to implement a Xiangqi game in OCaml with features specified in Features to Implement section. Briefly, we plan to implement the game with a graphical interface and in a server-based fashion that allows multiple players to connect and play. We also aim to implement a Xiangqi AI that a single player can play against.

List of libraries

- Core
- Dream
- SQLite3

Note: we plan to process most of the logic in the back-end, therefore ReScript for front-end code is not needed in our project design.

Github Repo

Initial specifications: https://github.com/RaphelWei/FPSE-final-proj

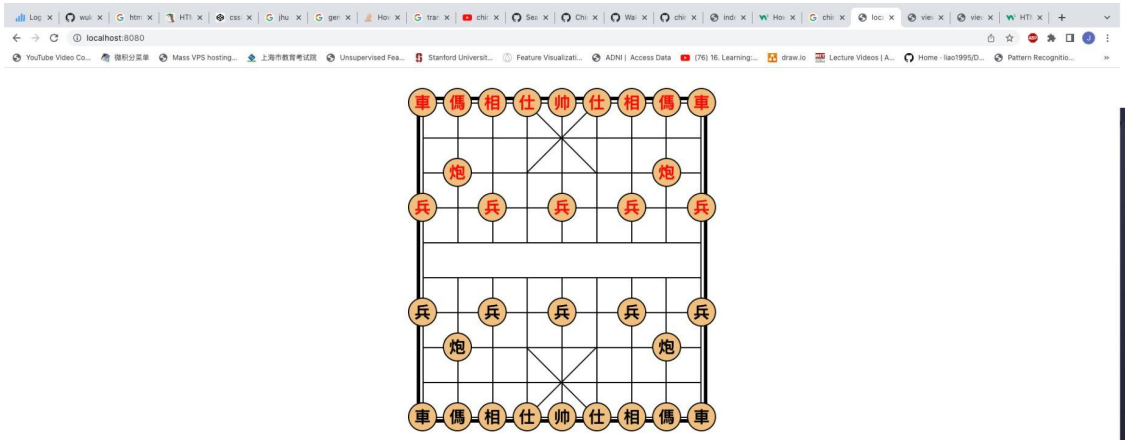
All current .mli files are included in the /src folder.

Mock use

Initial graphic interface in command line

A player input a pair of coordinates indicating the source piece and destination. If this is a valid move, the game renders the updated board and switch turn, otherwise it render a error message. The asteroid shows which player's turn it is (gray at endgame).

We will present a browser-based version in the future



Features and Implementation Plan

Implement according to the order:

1. Graphical user interface: as demonstrated in Mock Use
2. Game rules and hints on next valid move: show legal moves for the piece picked by the user
3. Two-player mode via web-server: use Dream WebSocket in server.ml to
    a. Keep the games and its update in memory, use Dream.sessions_memory to store sessions in server memory. Passes session IDs to clients in cookies
    b. Manage all clients' websocket with Hashtbl

    c. For each update, server send json string of updated chess board to all clients in the same session
4. Multiple game instances via web-server (*): multiple games can be opened on different machines simultaneously
    a. stores sessions and their corresponding game in an SQL database. Passes session IDs to clients in cookies.
5. Single-player mode against random AI: user play against an AI that generates random legal moves
6. Single-player mode against a minimax AI
    a. handle the problem of large game-tree with alpha-beta pruning (*)

*: implement if time allows

Tasks

Sihan Wei: Game logic, Server
Jiarui Wang: Game logic, AI
Yilin Chen: Game logic, Server