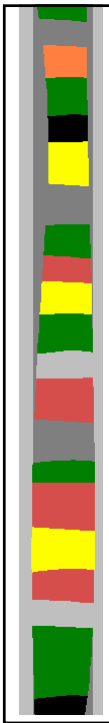
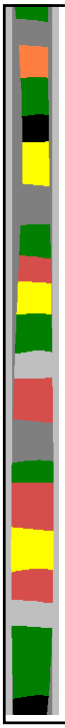


Online Chess

Project 3

Due date: April 17th



Introduction

- Third in series of three projects
- This project focuses on adding online support
  - 2 players on separate computers play each other
  - 1 player plays AI where AI running on separate computer
- You will work individually for this project!
  - No groups
  - Encouraged to talk about solutions with class mates
  - Can help each other debug code
  - Cutting and pasting or mailing code → too much

Details (1 of 2)

- Construct a client-server architecture
  - Server runs at "well-known" address and port
  - Player runs client on local machine and connects to server for online play
  - All communication is to and from client and server (ie- there is no client-client communication)
- Server is persistent
  - Waits for connections from exactly two players
  - Then starts players off in a game
  - When game over, returns to waiting for new connections to start next game
- Server keeps master game state
  - Controls where all the pieces are and whose turn it is
- Chess clients collect moves from the players (this is your proj1)
  - Sends player moves to server

Details (2 of 2)

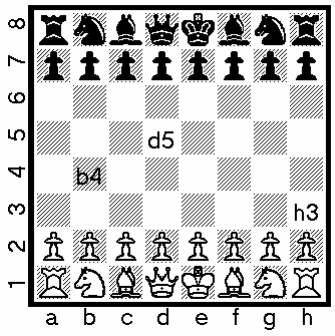
- Advanced:
  - "Think" time: does not have to spend same amount of time per move, but make sure doesn't think too long
  - Sophisticated Evaluation function: Basic count of material (ie- pieces) good, but a lot of additional enhancements that might be added
  - Opening vs. Mid-Game vs. End-Game: Evaluation of opening moves or end-game state may proceed differently than mid-game state.
  - Personality: Different versions of Blue (selected randomly, is fine) may be more aggressive or defensive or …
    - Can change with Evaluation
- You will use:
  - MinMax search algorithm
    - with AlphaBeta pruning
  - A board evaluation (lecture notes coming up have details)
- Implement in your choice of language (C, C++, Java, …)
  - Use a language you are familiar with!

Notes (1 of 2)

- Think about communication between server and client (*protocol*). Basic idea below:
- **Startup**
  - Connect (a TCP connection is established)
  - Client sends player name (a sequence of characters)
  - Server sends number representing player color (0 == white, 1 == black)
- **Play**
  - Server sends number representing turn (0 == white, 1 == black)
  - If Player's turn
    - Player sends MOVE (see format below)
    - Server sends number representing response (0 == ok, 1 == illegal move)
  - else
    - Server sends opponent MOVE
- **End**
  - Server sends number representing final winner (0 == white, 1 == black)
  - Close (TCP connection is closed)

Notes (2 of 2)

- MOVE message in format of FROM:TO, where FROM and TO represent squares on the chess board. The rows of the chess board are numbered 1-8 and the columns a-h, like so:

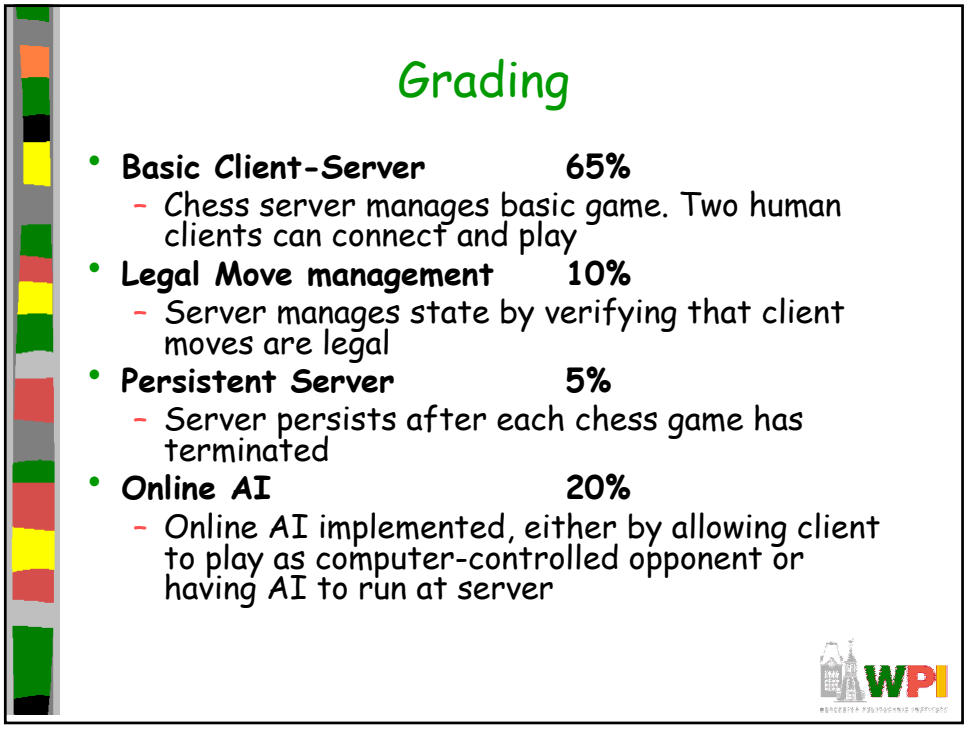


Links

- See project Web page
- Java implementation, basic TCP sockets
- C++ implementation, Linux and Windows sockets
  - TCP and UDP and other stuff, but probably only need TCP
- Chess links from project 1

Submission

- Will use command line (Unix) turnin
- Submit all source files necessary to compile and run programs
  - Any Makefiles, etc.
  - All art (as appropriate)
- README
  - Saying how to build, platform, etc.
  - Any command line parameters for either
  - Notes on how a game against the AI is launched


Grading
Basic Client-Server 65%
Chess server manages basic game. Two human clients can connect and play
Legal Move management 10%
Server manages state by verifying that client moves are legal
Persistent Server 5%
Server persists after each chess game has terminated
Online AI 20%
Online AI implemented, either by allowing client to play as computer-controlled opponent or having AI to run at server
WPI