Nicholas Krell
BINF 6380
10/10/2022

# Chess Player Project Proposal

Introduction: For my java project, I would like to create a program that allows the user to play chess against either another user on the same computer or against the computer itself. I originally envisioned an old-school type of interface utilizing ascii art in the terminal, but I could also implement a GUI if that is preferred.

Organizational Scheme: The basic parts of the program will be: a “chess engine”, a “rendering engine”, an input interface, and if possible a chess AI.

Chess Engine: This would take the form of an object capable of remembering the locations of all the pieces, determining which moves are legal, adding and removing pieces from the board, and storing the board layout in a format that can be interpreted by the “rendering engine”. The board will most likely take the form of a multidimensional array that stores the locations of the piece in a simple character form. The board object will have the simple equations required to “move” the pieces in 2d space. The Pieces themselves will likely all be implemented from a piece interface that outlines the normal operations that a piece can undergo. This would be on its own thread.

Rendering Engine: This would be an object containing a series of routines that translate the layout of the board from the simple character array in the chess engine into either ascii art in the terminal or a true GUI. It would run on its own thread, where it would repeatedly “render” the board in real-time as moves are made in the chess engine. It would only be able to read data from the simple character array, while the chess engine would be able to both read and write data. This will prevent the two from interfering with each other and causing multithreading issues. The rendering engine would also be able to show the paths of potential moves before they are finalized by communicating with the input interface.

Input Interface: This could be either a GUI or a scanner looking for inputs in the terminal. It would communicate with the rendering engine to show which piece was selected, the moves that piece could take, and if the moves were legal or not. It would also need to communicate with the chess engine to determine which moves could be made. This three-way communication may pose a multithreading issue, but if the rendering engine updates quickly enough, the user would never see this. Upon finalizing a move, it would communicate the move to the chess engine so that the board layout could be updated. This could also have a timer, restart, and concede button. If the two-player hot-seat option is chosen, this input interface would also need to communicate whose turn it was, and relay that information to the chess and rendering engines.

Chess AI: This would be by far the hardest and most complicated part to implement. It could take the form of a Markov model training to react to either the user's moves or to the board layout. This training could be accomplished by feeding it the positional coordinates of pieces from professional chess games along with who one. Alternatively, it could also learn *from* the user by at first making a random move in response to the user, and eventually assigning scores to all potential moves until it determines which reaction is most likely to result in victory. This second training method could so be automated by pitting two instances of the chess AI against each other repeatedly. While chess AI is definitely not a promise, I am in Machine Learning and may be able to implement the program described above by the end of the semester.