GO CHESS!

Version: 0.1.0 preview

SOFTWARE SPECIFICATION

UCI EECS 22L

Developed by: Team (8) Scholar's Mate

Hussain Mahuvawala

Sean Han

Jianheng Zhou

Paribesh Thapaliya

Ryan Zhang

Table of Contents

Glossary

AI - Computer player

Board - Where the game pieces are played on. Consists of 8 by 8 squares of alternating color

Enum - A special kind of data type defined by the programmer

Library - Collection of header files used by programs

Makefile - Script used to build and compile program

Module - A component of the program

Pieces - Each piece of the game. There are 6 different types: King, Queen, Bishop, Knight, Rook, and Pawn

Structures - A data type that contains other data types

Variable - An abstract storage location with a name containing some quantity of data or object referred to as a value

Software Architecture Overview

1.1 Main Data Types and Structures

We have 5 main files:

1. main.c
2. AI.c
3. Board.c
4. Log.c
5. CheckMove.c
6. EndGame.c

Each of these files would have supplementary functions and data types. The files would be linked accordingly in the Make File.

main.c

We have the main function. This is the main file for the software. We also have structures defined for the following components.

1. Player (Stores whether the player is playing the white or black pieces)
2. Piece (Based on input made by player, the piece that is to be played is checked, the value of that piece is stored in Piece)
3. Move (String of characters the stores the move inputted by the user, example: e2 e4)

More about the structures defined can be seen in section 3.1.

AI.c

We have the functions and code that ensure that the AI will play legal moves and provide competition to the player. This File would contain functions such as BestMove() to find the best move to be played in a particular position.

Board.c

We have a two dimensional array (called ChessBoard) that would store the current board position. At the beginning of the game the board would be set to the default Classic game position.

We also have the following functions such as

1. PrintBoard() that would print the current board position whenever it is called.
2. UpdateBoard() that would update the position of a piece on the board when a legal move is played.

Log.c

We have a Doubly Linked list to store and log all the moves that are played in a game. More information can be found in the 3.1 section. For the Doubly Linked List we will include structures such as

1. ChessList - This would store the Length, pointers to the First and the Last element of the list
2. ChessEntry - This would store a pointer to the List it belongs to and also pointers to the Previous and Next Entries. It also stores the move played at that point in the game.

We will also include functions such as

1. CreateChessList() - allocates memory for the chess list
2. CreateChessEntry() - allocates memory for the chess entry
3. DeleteChessList() - deallocates memory used by the list
4. DeleteChessEntry() - deallocates memory used by the entry
5. AppendMove() - Adds a ChessEntry element to the tail of the list when called.
6. UndoMove() - Removes the last element of the ChessList when called.

CheckMove.c

We have functions that check if a move played by a piece is legal or not, i.e., in accordance with the basic rules of chess. We have functions such as

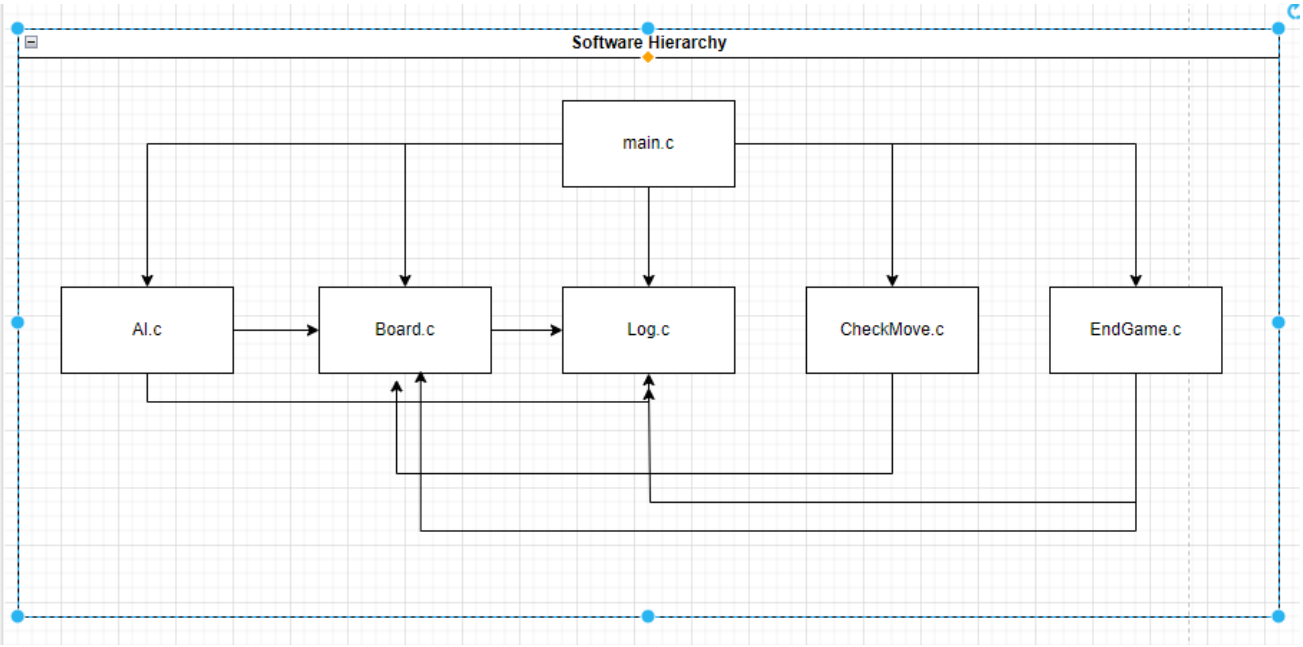
1. CheckMovePawn() - Checks if a move played by a pawn is legal.
2. CheckMoveKnight() - Checks if a move played by a knight is legal.
3. CheckMoveBishop() - Checks if a move played by a Bishop is legal.
4. CheckMoveRook() - Checks if a move played by a Rook is legal.
5. CheckMoveQueen() - Checks if a move played by the Queen is legal or not.
6. CheckMoveKing() - Checks if a move played by a King is legal or not.
7. SpecialMoves() - Ensures that moves such as “castling” and “en passant” are not considered illegal.

EndGame.c

We have a function that checks if the game has ended due to a checkmate, a stalemate or by threefold repetition. We also have a function EndGame() that will print a statement based on the result of the game and terminate the program.

1.2 Major Software Components

• Diagram of module hierarchy



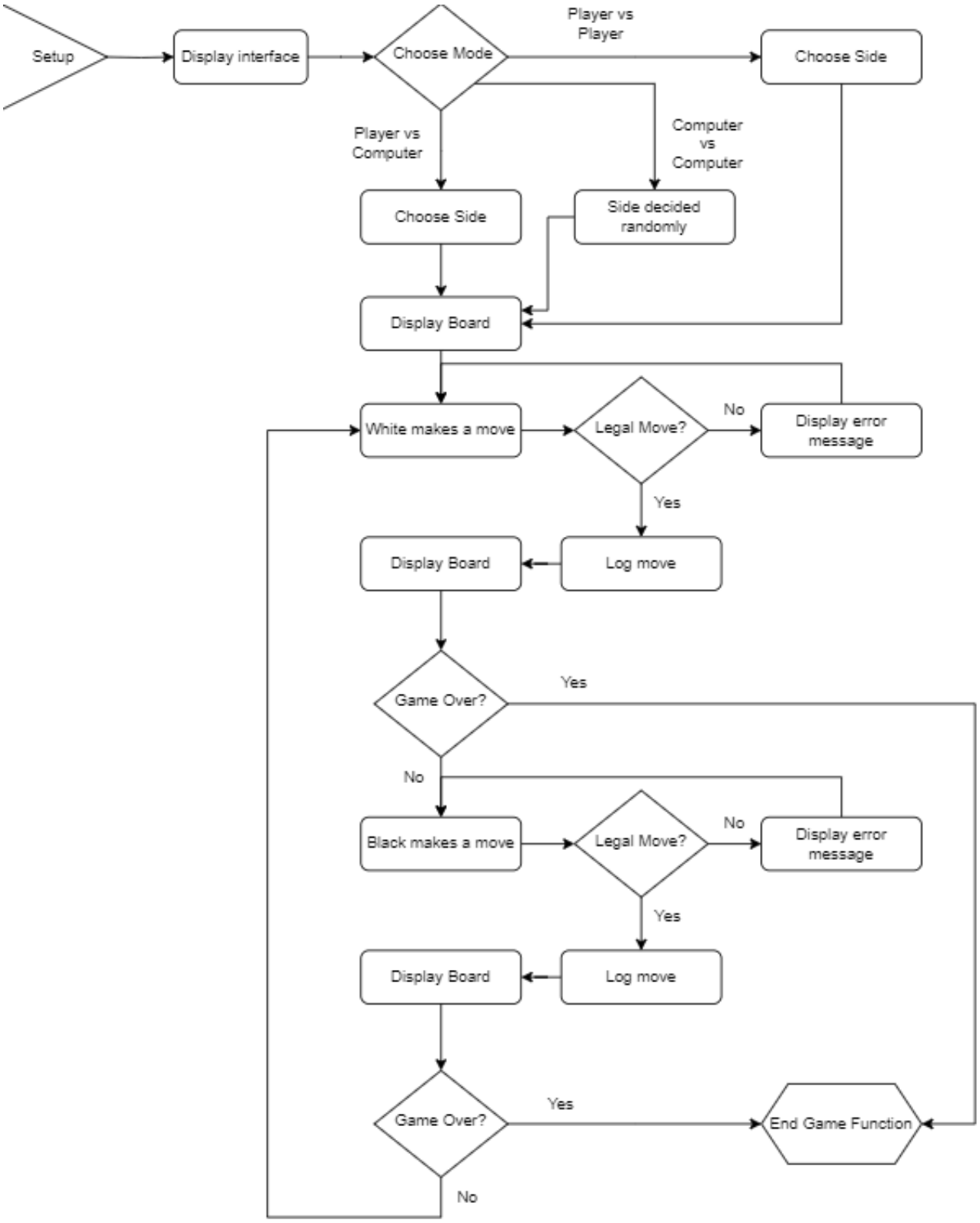
1.3 Module Interfaces

For this project we will be using the following libraries: -

1. stdio.h (Standard input output library)
2. stdlib.h (For memory allocation and deallocation)
3. assert.h (For asserting statements and ensuring smooth flow of the program)

We have not decided to use any other interfaces or APIs as of yet.

1.4 Overall Program Control Flow



Installation

2.1 System Requirements, Compatibility

- Operation System: 32-bit Linux, 64-bit Linux
- Hardware: PCs which are able to install and run Linux OS.
- 2G of ram (8+ recommended for optimal performance)
- 1G of disk space
- Monitor and keyboard

2.2 Setup and Configuration

- Download Go chess!
- Download the file Chess_V1.0.tar.gz
- Extract the contents of the file
- Run the installation command
- Software comes in a tar.gz package. After downloading Chess_V1.0.tar.gz, extract the package by running
    - tar -zxvf Chess_V1.0.tar.gz
- Change into directory using
    - cd Chess_V1.0

2.3 Building, Compilation, Installation

Command to build the program(Makefile shown below):

- make all

```
#Makefile

CC = gcc
CFLAGS = -Wall -std=c11

all: Chess

test:
    ./Chess

Clean:
    rm *.o Chess

main.o: main.c main.h
    $(CC) $(CFLAGS) main.c -o main.o
board.o: board.c board.h
    $(CC) $(CFLAGS) board.c -o board.o
AI.o: AI.c AI.h
    $(CC) $(CFLAGS) AI.c -o AI.o
CheckMove.o: CheckMove.c CheckMove.h
    $(CC) $(CFLAGS) CheckMove.c -o CheckMove.o
EndGame.o: EndGame.c EndGame.h
    $(CC) $(CFLAGS) EndGame.c -o EndGame.o
log.o: log.c log.h
    $(CC) $(CFLAGS) log.c -o log.o
Chess: main.c board.c AI.c CheckMove.c EndGame.c log.c
    $(CC) $(CFLAGS) main.o board.o AI.o CheckMove.o EndGame.o log.o -w -o Chess
```

Command to compile the program:

- ./Chess

Installation is described above in section 2.2

Documentation of packages, modules, interfaces

3.1 Detailed Description of Data Structures

```
struct Player {
   // Player color, white or black
   char color;
   // Whether the player is a computer
   bool isComputer;
};
```

The data structure of the player, showing which side of the player is and whether it is AI or human. The connection to AI tournament as well

```
enum PieceType {
   KING, QUEEN, ROOK, BISHOP, KNIGHT, PAWN
}; // Piece type enumeration definition
```

Using enum to represent the piece type, avoiding having an index for each piece which confused group mates sometimes.

```
struct Piece {
   // Piece type
   enum PieceType type;
   // Owning player
   struct Player *owner;
};
```

For each Pieces, we have a pointer point to the Piece Type and Player, so we can easily get access to the Pieces information

```
struct Position {
   // Horizontal coordinate
   int x;
   // Vertical coordinate
   int y;
};
```

Position information is reserved in Position Structure, when we use move functions, we have to have a start position and an end position, so the information is stored right here.

```
struct Move {
    // Starting position
    struct Position start;
    // Ending position
    struct Position end;
};
```
Which points to where the position information stored

```
struct Board {
    // Two-dimensional array representing pieces on each position of the board
    struct Piece *squares[8][8];
};
```
This is the 2D array which represents the chessboard. And each of the elements of the array points to the information of the single piece, so we can get any information about the board from this structure.

```
// Game structure definition
struct Game {
    // Chessboard
    struct Board board;
    // Current player
    struct Player *current_player;
    // Last move made
    struct Move *last_move;
};
```
This is a structure that records the current situation of the Chess game including player, the position of the last move and the board information. This struct is a pass-in variable for the AI code.

```
// BoardNode structure definition
struct BoardNode {
    // State of the board
    struct Board board;
    // Pointer to the next node
    struct BoardNode *next;
};
```

```
// BoardList structure definition
struct BoardList {
    // Pointer to the head node
    struct BoardNode *head;
    // Pointer to the tail node
    struct BoardNode *tail;
};
```

These two structs are the double linked list for recording the previous move of the two players, which is for write in logfile and for undo functions.

```
// Alpha Beta search result structure definition
struct AlphaBetaResult {
    // Best move
    struct Move best_move;
    // Best score
    int score;
};
```

This struct preserved the best move information for AI logic

3.2 Detailed Description of Functions and Parameters

main.c:

//initialize global variables
//Prints statements to welcome user
//Prints menu
//switch case/if-else statements for user choice of gamemode
//returns 1 if white returns 2 if black color choice by user
main(void);

Main function runs when the user starts the program. Calls other functions to run the program.

Board.c:

//Function to initialize the board
void init_board(struct Board *board);
The input parameter is the board of the chess game that is in play.
The function returns an initialized board for the game.

//Used to update the position of a piece on the board when legal move is made
void copy_board(struct Board *src, struct Board *dest);
The input parameters are: -
The Board that is printed for the user to see
The temporary board that gets updated when a move is played.
The function returns the main board updated with the most recent move.

AI.c:

// Alpha Beta search function definition
struct AlphaBetaResult alpha_beta_search(struct Game *game, int depth, int alpha, int beta, struct Player *player);
The input parameters are: -
The structure that holds the main information about the Game
An integer depth to analyze how many steps further we need to play for the best move

// Function to evaluate the position score
int evaluate_position(struct Game *game, struct Player *player);
The parameters are the structure Game that holds the key information.
And the structure Player that holds the information fo the player.
The function returns a score based on the evaluated position
// Function to generate all legal moves
struct Move *generate_legal_moves(struct Game *game, struct Player *player, int *num_moves);

Used to ensure the best move is made out of all possible choices and returns that move.

CheckMove.c:
The parameters for all the functions are the same and so are the return types.
The functions take the piece to move on the current board position, and the move entered by the user as the parameters.
The function returns true or false depending on whether the move is legal or not.

bool CheckMovePawn(structure Piece *board[8][8], int x, int y);
Used to check if the user input is a valid move for the pawn piece

bool CheckMoveKnight(structure Piece *board[8][8], int x, int y);
Used to check if the user input is a valid move for the knight piece

bool CheckMoveBishop(structure Piece *board[8][8], int x, int y);
Used to check if the user input is a valid move for the bishop piece

bool CheckMoveRook(structure Piece *board[8][8], int x, int y);
Used to check if the user input is a valid move for the rook piece

bool CheckMoveQueen(structure Piece *board[8][8], int x, int y);
Used to check if the user input is a valid move for the queen piece

bool CheckMoveKing(structure Piece *board[8][8], int x, int y);
Used to check if the user input is a valid move for the king piece

bool SpecialMoves(structure Piece *board[8][8], int x, int y);
Used to check if the user input is one of the valid special moves(en passant and castling)

EndGame.c:
The functions take the structure Game and the structure Player as inputs. These structures hold information regarding the player who just played a move and key information about the game.
The function returns a number that corresponds with code logic to see if the game has ended or not and checks how it has ended.

```
// Function to check if a player is in check
int is_in_check(struct Game *game, struct Player *player);

// Function to check if it is checkmate
int is_checkmate(struct Game *game, struct Player *player);
```

```
// Function to check if it is stalemate
int is_stalemate(struct Game *game, struct Player *player);
```

Log.c:

```
// Function to initialize the board
void init_board(struct Board *board);

// Function to copy the board
void copy_board(struct Board *src, struct Board *dest);
        Takes two structures of type Board. One that has the updated move.
        And one that is used as the main board in the program for printing.
        Updates the main board with the new position.

// Function to determine if a specified position is empty
int is_square_empty(struct Board *board, struct Position *pos);
        Takes the Board and a Position on the Board as parameters.
        Returns a value based on whether the specified position is empty or not.

// Function to place a piece at a specified position
void place_piece(struct Board *board, struct Piece *piece, struct Position *pos);
        Takes the Board, the Piece to be moved and the Position entered by the user.

// Function to get a piece at a specified position
struct Piece *get_piece_at(struct Board *board, struct Position *pos);
        Takes the Board and a position as the parameters.
        Returns the piece if there is a piece at the Position.

// Function to move a piece
int move_piece(struct Game *game, struct Move *move);
        Takes the game and the structure Move for the move inputted by the user as
parameters.

// Function to initialize the board list
void init_board_list(struct BoardList *list);
        Initializes the BoardList

// Function to insert a board node at the end of the list
void insert_board(struct BoardList *list, struct Board *board);
        Takes the BoardList and the Board as parameters and updates the BoardList
based on the position in the Board.

// Function to delete a board node from the end of the list
void delete_board(struct BoardList *list);
```

Takes the BoardList as a parameter and deletes a node from the end of the list

```
// Function to clear the board list
void clear_board_list(struct BoardList *list);
```

This function takes the boardList as its argument and clears it.

```
// Function definition for undoing a move
void undo_move(struct Game *game, struct BoardList *list);
```

This function takes the BoardList and the Game as arguments and undo's the previous move in the BoardList while also changing the Board Position.

3.3 Detailed Description of Input and Output Formats

In order to make a move, the user must input the start and end position of a piece. The program will display “Input the piece you would like to move:”. The user will first enter a character for the column(a-h) followed by a number for the row(1-8). The program will then prompt the user for another space on the board where the user would like to move the piece(shown below).

```
Input the piece you would like to move: e2
Input the space you would like to move to: e4
```

The board will then be updated and shown:

```
  +----+----+----+----+----+----+----+----+
8 | bR | bN | bB | bQ | bK | bB | bN | bR |
  +----+----+----+----+----+----+----+----+
7 | bP | bP | bP | bP | bP | bP | bP | bP |
  +----+----+----+----+----+----+----+----+
6 |    |    |    |    |    |    |    |    |
  +----+----+----+----+----+----+----+----+
5 |    |    |    |    |    |    |    |    |
  +----+----+----+----+----+----+----+----+
4 |    |    |    |    | wP |    |    |    |
  +----+----+----+----+----+----+----+----+
3 |    |    |    |    |    |    |    |    |
  +----+----+----+----+----+----+----+----+
2 | wP | wP | wP | wP |    | wP | wP | wP |
  +----+----+----+----+----+----+----+----+
1 | wR | wN | wB | wQ | wK | wB | wN | wR |
  +----+----+----+----+----+----+----+----+
    a    b    c    d    e    f    g    h
```

If the inputted space is not a valid move for the chosen piece or if the piece’s path is blocked, the user will receive an error message and be asked to choose another space(shown below).

```
Input the piece you would like to move: e2
Input the space you would like to move to: e6
Invalid move! Please choose another space: e4
```

After the game is completed, a log file will be given to the user that contains all the moves that were made during the game. The text document will display the move, type of piece, the player who made the move, and the initial/final position of the piece.

```
Log file:
Move    Type    White   Type    Black
1       P       e2-e4   P       e7-e6
2       K       e1-e2   P       f7-f5
3               ...             ...
```

Development Plan and Timeline

4.1 Partitioning of Tasks

1. Create structure
    - 1.1. Each piece has a type, color, and location
    - 1.2. Type of piece defines how it moves
    - 1.3. Special moves
2. Make a board
    - 2.1. 2D array of locations of pieces
    - 2.2. Location structure
        - 2.2.1. Occupancy (any chess blocking way)
        - 2.2.2. Color
3. Move pieces & check kings
    - 3.1. Check if the user input is a valid move
        - 3.1.1. Check if move will put king in check
        - 3.1.2. Check which moves will get the king out of check
            - 3.1.2.1. If there are no moves, game over(checkmate)
        - 3.1.3. Special variable for Kings and Rooks(castling)
        - 3.1.4. Pawns need to check opponent's last move(en passant)
    - 3.2. Check if any pieces can be captured
4. Output list of moves made during the game
    - 4.1. Download text document with the moves
5. AI best move logic
    - 5.1 Alpha-Beta Search tree
    - 5.2 Best score for the Best Move option

4.2 Team Member Responsibilities

Data structure (Board & List of Move) - Ryan
Main program flow - Hussain & Ryan
Check legal move & Check board state - Sean & Hussain
AI best move logic - Ryan & Jianheng
Makefile - Paribesh
Logfile & write-in File - Paribesh
Documentation - Group
Testing - Group

Back Matter

Copyright

References

EECS 22L: Project 1; Prepared by Prof. Rainer Doemer; 20 March, 2024.

EECS 22L: Chess Software Specification Grading Criteria; Prepared by Yutong Wang and Mao-Hsiang Huang; Prof. Rainer Doemer; 8 April, 2024

Image(cover page):
https://www.spreadshirt.com/shop/design/chess+icon+king+horse+logo+sticker-D6089b439660ca11715efed92?sellable=xrgJwr3479U5B7Dlrdag-1459-215

Index