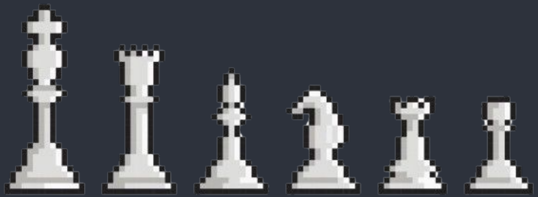


</

# Aphrodite Chess Engine

/>

*University of California, Irvine – EECS 22L*

Abraham Ou, David Liu, Jiwon Youn,
Michael Schinazi, Miles Jennings,
Sandeep Gullapalli

1 0 1 1   0 1 1   0 1   1 0 1 1 0 0 1   1 0   1 1 0 1 1   0 1 1   0 1   1 1 0 1 1 0   1 1 0 1 1 1   1 1 0 1

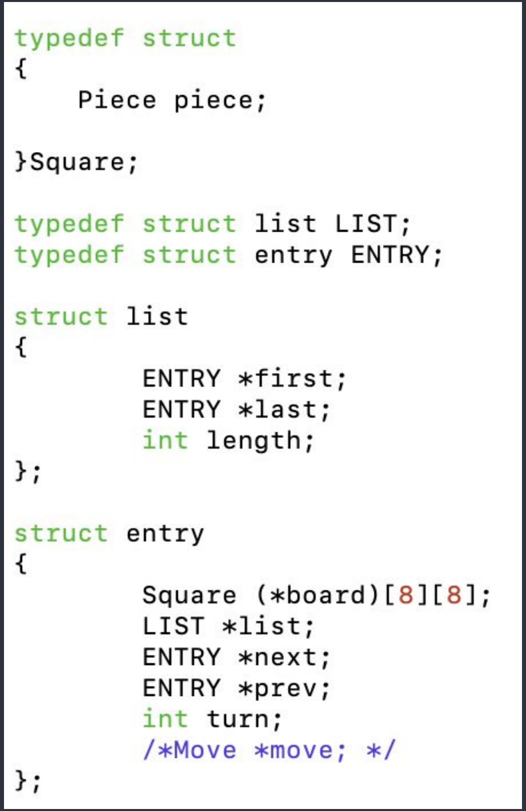
# </Overview

1 0 1 1 0 1 1 0 1 1 0 1 1 0 0 1 1 0 1 1 0 1 1 0 1 1 0 1 1 1 0 1 1 0 1 1 0 1 1 1 1 1 0 1

## </Status: Alpha Version

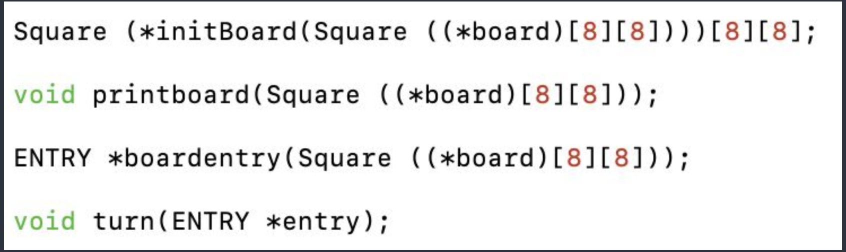
- Functioning User v.s User gamemode accessible through the Linux terminal
- White pieces represented by capital letters, black pieces are lowercase
- Each piece has their own legal moveset
  - Functioning castling moves for long and short side
  - Pawn Promotion
- Players can capture each others pieces with accordance to chess rules
- Move log records each player's moves
- Software Release Goals: User vs. Computer A.I.
  - Win Conditions, Random Move Generator, Minimax, and Options to Change Difficulty

# </Board Structure



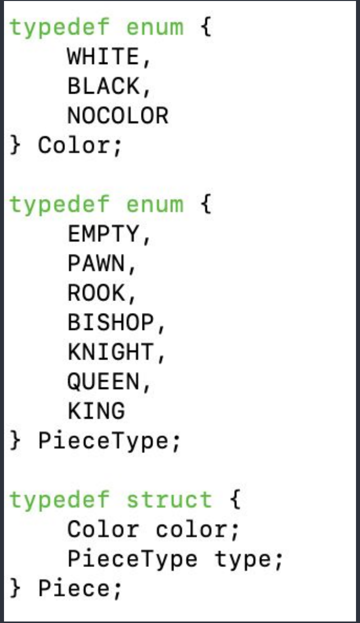
```c
typedef struct
{
    Piece piece;

}Square;

typedef struct list LIST;
typedef struct entry ENTRY;

struct list
{
        ENTRY *first;
        ENTRY *last;
        int length;
};

struct entry
{
        Square (*board)[8][8];
        LIST *list;
        ENTRY *next;
        ENTRY *prev;
        int turn;
        /*Move *move; */
};
```

Figure A



```c
Square (*initBoard(Square ((*board)[8][8])))[8][8];

void printboard(Square ((*board)[8][8]));

ENTRY *boardentry(Square ((*board)[8][8]));

void turn(ENTRY *entry);
```

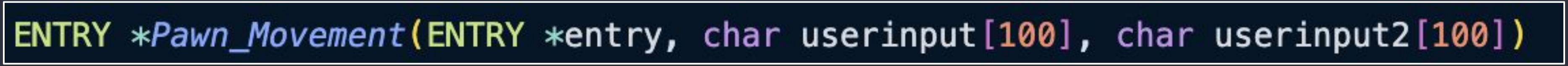
Figure B



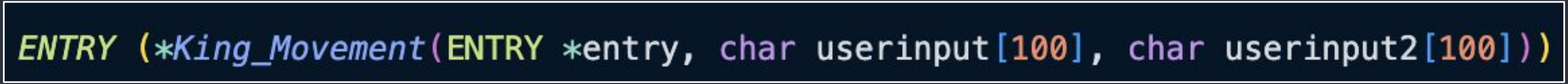
```c
typedef enum {
    WHITE,
    BLACK,
    NOCOLOR
} Color;

typedef enum {
    EMPTY,
    PAWN,
    ROOK,
    BISHOP,
    KNIGHT,
    QUEEN,
    KING
} PieceType;

typedef struct {
    Color color;
    PieceType type;
} Piece;
```

Figure C

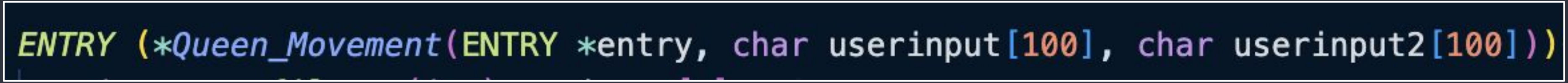
## </Movement Functions



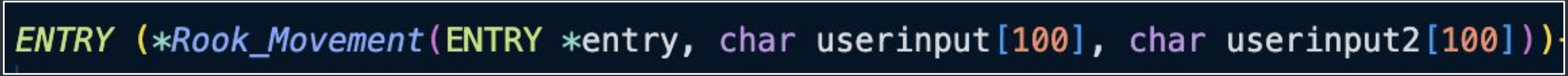
```
ENTRY *Pawn_Movement(ENTRY *entry, char userinput[100], char userinput2[100])
```



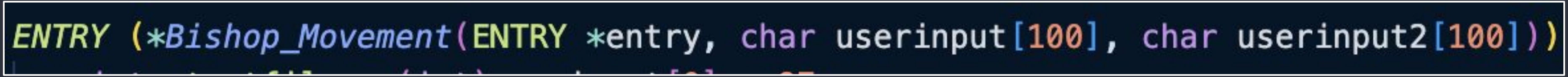
```
ENTRY (*King_Movement(ENTRY *entry, char userinput[100], char userinput2[100]))
```



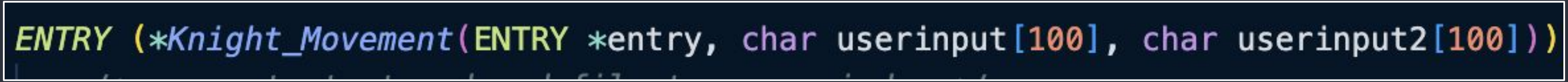
```
ENTRY (*Queen_Movement(ENTRY *entry, char userinput[100], char userinput2[100]))
```



```
ENTRY (*Rook_Movement(ENTRY *entry, char userinput[100], char userinput2[100]))
```

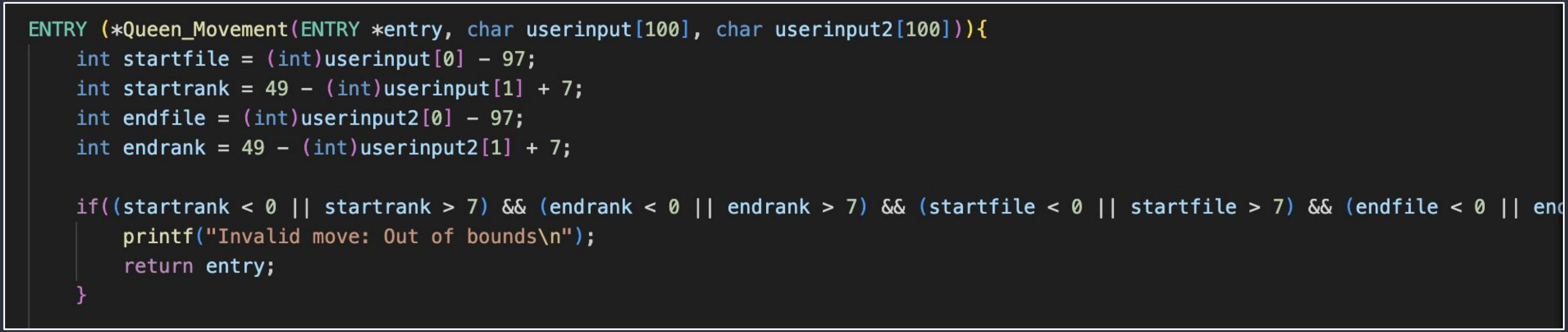


```
ENTRY (*Bishop_Movement(ENTRY *entry, char userinput[100], char userinput2[100]))
```

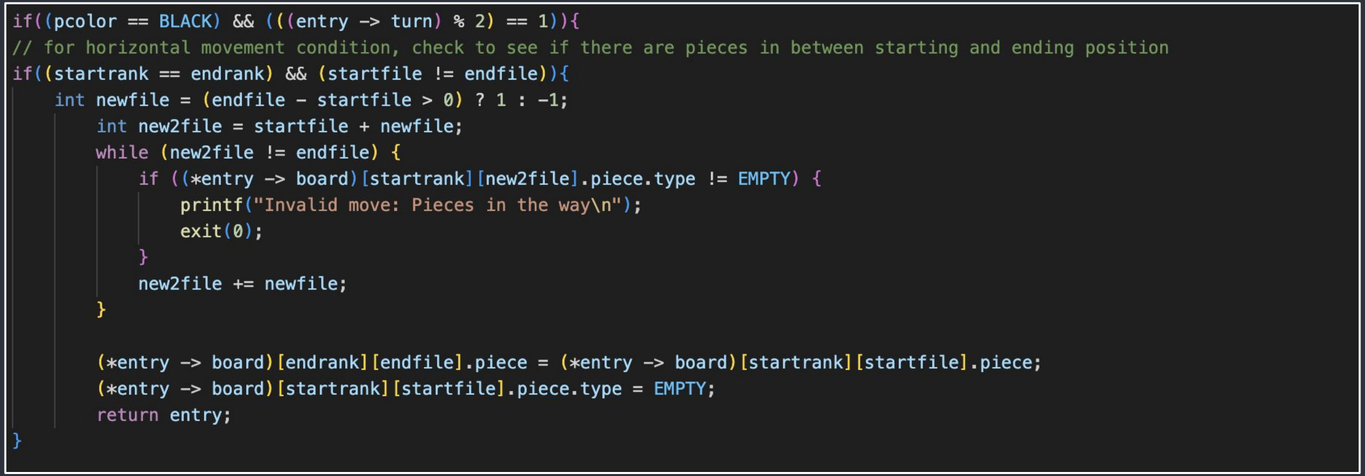


```
ENTRY (*Knight_Movement(ENTRY *entry, char userinput[100], char userinput2[100]))
```

# </Movement Function Example

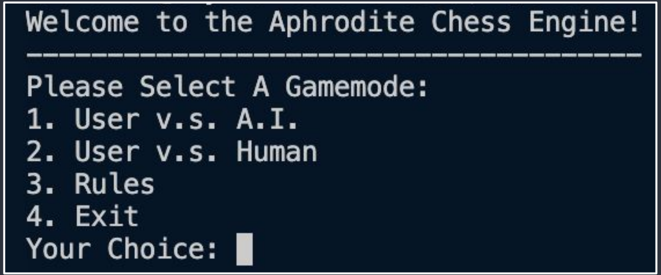


```
ENTRY (*Queen_Movement(ENTRY *entry, char userinput[100], char userinput2[100])){
    int startfile = (int)userinput[0] - 97;
    int startrank = 49 - (int)userinput[1] + 7;
    int endfile = (int)userinput2[0] - 97;
    int endrank = 49 - (int)userinput2[1] + 7;

    if((startrank < 0 || startrank > 7) && (endrank < 0 || endrank > 7) && (startfile < 0 || startfile > 7) && (endfile < 0 || en
        printf("Invalid move: Out of bounds\n");
        return entry;
    }
```



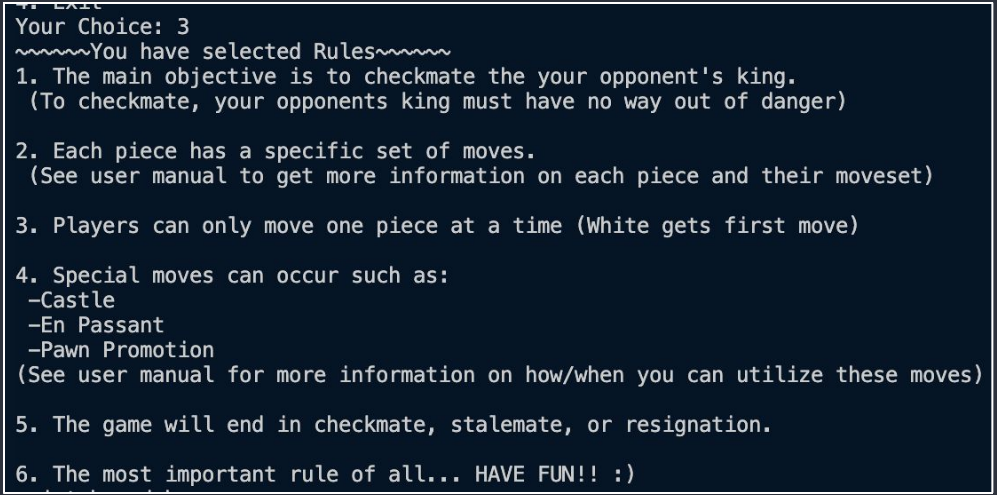
```
if((pcolor == BLACK) && (((entry -> turn) % 2) == 1)){
// for horizontal movement condition, check to see if there are pieces in between starting and ending position
if((startrank == endrank) && (startfile != endfile)){
    int newfile = (endfile - startfile > 0) ? 1 : -1;
        int new2file = startfile + newfile;
        while (new2file != endfile) {
            if ((*entry -> board)[startrank][new2file].piece.type != EMPTY) {
                printf("Invalid move: Pieces in the way\n");
                exit(0);
            }
            new2file += newfile;
        }

        (*entry -> board)[endrank][endfile].piece = (*entry -> board)[startrank][startfile].piece;
        (*entry -> board)[startrank][startfile].piece.type = EMPTY;
        return entry;
}
```

# </Demonstration



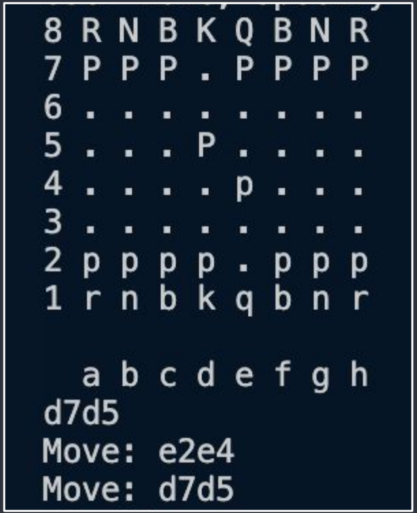
```
Welcome to the Aphrodite Chess Engine!
--------------------------------------
Please Select A Gamemode:
1. User v.s. A.I.
2. User v.s. Human
3. Rules
4. Exit
Your Choice: █
```

Figure A



```
Your Choice: 3
~~~~~~You have selected Rules~~~~~~
1. The main objective is to checkmate the your opponent's king.
 (To checkmate, your opponents king must have no way out of danger)

2. Each piece has a specific set of moves.
 (See user manual to get more information on each piece and their moveset)

3. Players can only move one piece at a time (White gets first move)

4. Special moves can occur such as:
 -Castle
 -En Passant
 -Pawn Promotion
(See user manual for more information on how/when you can utilize these moves)

5. The game will end in checkmate, stalemate, or resignation.

6. The most important rule of all... HAVE FUN!! :)
```

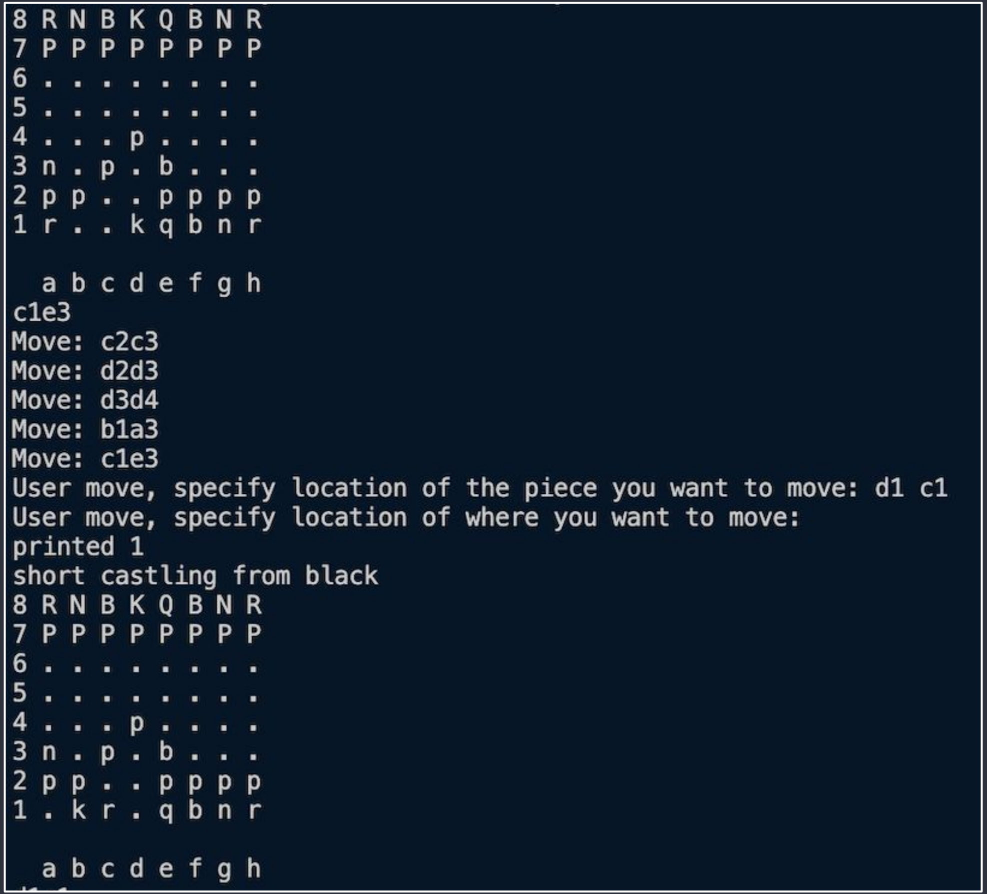
Figure B



```
8 R N B K Q B N R
7 P P P . P P P P
6 . . . . . . . .
5 . . . P . . . .
4 . . . . p . . .
3 . . . . . . . .
2 p p p p . p p p
1 r n b k q b n r

  a b c d e f g h
d7d5
Move: e2e4
Move: d7d5
```

Figure C

## </Demonstration continued



```
8 R N B K Q B N R
7 P P P P P P P P
6 . . . . . . . .
5 . . . . . . . .
4 . . . p . . . .
3 n . p . b . . .
2 p p . . p p p p
1 r . . k q b n r

  a b c d e f g h
c1e3
Move: c2c3
Move: d2d3
Move: d3d4
Move: b1a3
Move: c1e3
User move, specify location of the piece you want to move: d1 c1
User move, specify location of where you want to move:
printed 1
short castling from black
8 R N B K Q B N R
7 P P P P P P P P
6 . . . . . . . .
5 . . . . . . . .
4 . . . p . . . .
3 n . p . b . . .
2 p p . . p p p p
1 . k r . q b n r

  a b c d e f g h
```

## </Main Challenges

| Challenges | Why? |
|---|---|
| AI | Never dealt with AI before, hard to make computer respond to every strategy/move |
| Check and Checkmate | Many possible checkmate scenarios |
| Legal Movement | We had to create functions for each piece to make sure no illegal moves can be done and that our board successfully updates after each move |
| Creating the Board Structure | Creating the board was difficult because we had to draw out our own doubly linked list and create many structs |

## </What We Learned

{01}

- Incorporating structs, enum, pointers, doubly linked lists, dynamic memory allocation, and utilizing git.

{02}

- Communicating consistently is key.

{03}

- Being able to work with each others strengths and working together to fix problems.

{04}

- How to effectively apply what we learned about linked lists, structures, and other concepts from EECS 22 into improving our chessboard.

# </Q + A

Thank you for your time!

Any questions?