# CHESS

OLIVIA PRICE
RACHEL AIRD
DEDE BARRIGAH
PETER DEA

# Distributing the Work

- Game Logic (Olivia and Peter)
- Interactivity (Everyone)
- Visualization (Rachel and Dede)
- Computer Players/A.I. (Olivia and Peter)
- README (Rachel and Dede)

# GAME LOGIC (CHESS BUT EPIC)

**OLIVIA AND PETER**

```
//set up the board
int board[8][8] = {
    {wr, wn, wb, wq, wk, wb, wn, wr},
    {wp, wp, wp, wp, wp, wp, wp, wp},
    {__, __, __, __, __, __, __, __},
    {__, __, __, __, __, __, __, __},
    {__, __, __, __, __, __, __, __},
    {__, __, __, __, __, __, __, __},
    {bp, bp, bp, bp, bp, bp, bp, bp},
    {br, bn, bb, bq, bk, bb, bn, br}};
```

The board is stored as a 2D array of integers, with each piece being represented by an integer.

We created functions to incorporate each piece's unique move set

```
if (piece % 10 == 1 || piece % 10 == 5) { // ROOK or QUEEN
  add_sweep_to_legal(&lm, board, 1, 0, i, j);
  add_sweep_to_legal(&lm, board, -1, 0, i, j);
  add_sweep_to_legal(&lm, board, 0, 1, i, j);
  add_sweep_to_legal(&lm, board, 0, -1, i, j);
}
```

Using these functions, we can create a list of every legal move

# GAME LOGIC (CHESS BUT EPIC)

**BIGGEST CHALLENGES**

## Handling Check

- When playing check, you cannot move any of the pieces unless trying to protect the king
- This function is sweeping the board to see if any pieces are checking the other player
- If check it returns one, letting the board know that the other player can only do certain moves

```
//check
int check(int analyze[8][8], int color) {
  int kingY;
  int kingX;
  int piece;
  for (int i = 0; i < 8; i++) {
    for (int j = 0; j < 8; j++) {
      piece = analyze[i][j]; // piece contains the piece we are checking

      if (color == 1) {

        // check for checks from white

        if (piece == wr || piece == wq) {
          if (sweep(analyze, -1, 0, i, j).endpiece == bk) {
            return 1;
```

## Staying Within Bounds

- If we try to access a square on the board that doesn't exist, we will get a runtime error

```
board[4][9];
```

array index 9 is past the end of the array (which contains 8 elements)
ccls

# INTERACTIVITY
## CHESS BUT EPIC & PRETTY COMBINED

- Using ./main in the shell terminal takes arguments to set the theme and number of human players
- The arrows on the keyboard are used to move the cursor
- Space bar is used to pick a piece and place it
- We gave this code to Rachel and Dede to implement the cursor and highlighted moves on the board

```
//HIGHLIGHT SQUARE YOU CAN GO TO
for (int index = 0; index < 25; index++){
  if (highlights[index][0] == i && highlights[index][1] == j) {
    attron(COLOR_PAIR(6));
  }
}

//HIGHLIGHT THE CURSOR
if(i == cursorY && j == cursorX){
  attron(COLOR_PAIR(5));
}
```

```
void moveCursor(){
  input = getch();
  if (input==KEY_UP && cursorY < 7){
    cursorY++;
  }
  if (input==KEY_DOWN && cursorY > 0){
    cursorY--;
  }
  if (input==KEY_LEFT && cursorX > 0){
    cursorX--;
  }
  if (input==KEY_RIGHT && cursorX < 7){
    cursorX++;
  }
}
```

# CHESS BUT PRETTY

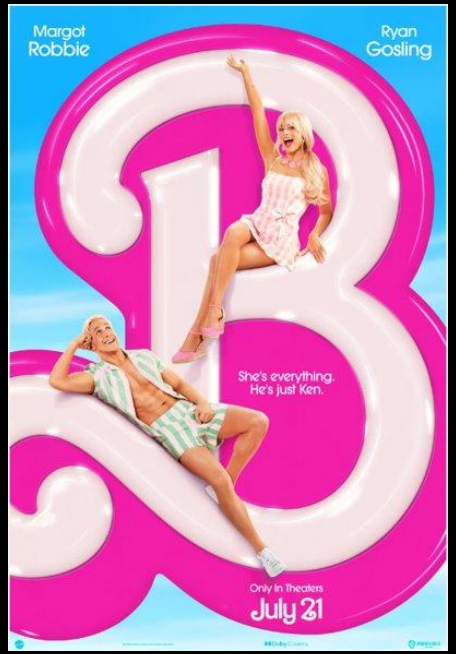
```
init_pair(13, COLOR_BLACK, COLOR_WHITE);
init_pair(14, COLOR_BLACK, COLOR_WHITE);
init_pair(15, COLOR_BLACK, COLOR_MAGENTA);
init_pair(16, COLOR_BLACK, COLOR_MAGENTA);
```

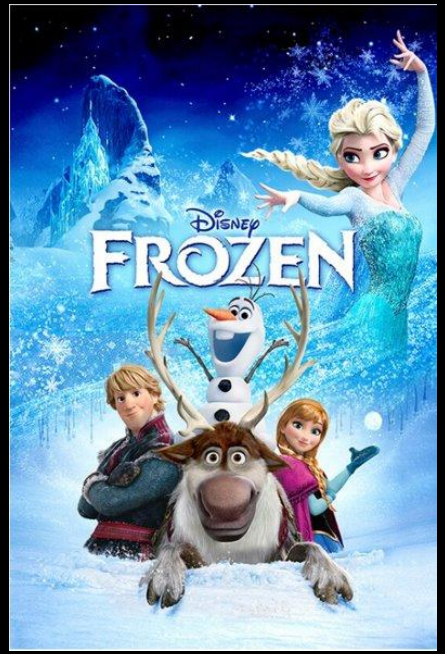
```
//blank tile
#define __  0
//assign each white piece an int
#define wr  11
#define wn  12
#define wb  13
#define wk  14
#define wq  15
#define wp  16
//assign each black piece an int
#define br  21
#define bn  22
#define bb  23
#define bk  24
#define bq  25
#define bp  26


int get_color(int piece){
  //input piece and return color (0 for white, 1 for black)
  if (piece > 20){return 1;}
  else if (piece < 20 && piece > 0){return 0;}
  else {return -1;}
}

//note the board is flipped vertically
int board[8][8] = {
{wr, wn, wb, wq, wk, wb, wn, wr},
{wp, wp, wp, wp, wp, wp, wp, wp},
{__, __, __, __, __, __, __, __},
{__, __, __, __, __, __, __, __},
{__, __, __, __, __, __, __, __},
{__, __, __, __, __, __, __, __},
{bp, bp, bp, bp, bp, bp, bp, bp},
{br, bn, bb, bq, bk, bb, bn, br}};
```

```
Solution 1:
      a    b    c    d    e    f    g    h
8     Q    -    -    -    -    -    -    -
7     -    -    -    -    Q    -    -    -
6     -    -    -    -    -    -    -    Q
5     -    -    -    -    -    Q    -    -
4     -    -    Q    -    -    -    -    -
3     -    -    -    -    -    -    Q    -
2     -    Q    -    -    -    -    -    -
1     -    -    -    Q    -    -    -    -

Solution 2:
      a    b    c    d    e    f    g    h
8     Q    -    -    -    -    -    -    -
7     -    -    -    -    -    Q    -    -
6     -    -    -    -    -    -    -    Q
5     -    -    Q    -    -    -    -    -
4     -    -    -    -    -    -    Q    -
3     -    -    -    Q    -    -    -    -
2     -    Q    -    -    -    -    -    -
1     -    -    -    -    Q    -    -    -

Solution 3:
      a    b    c    d    e
8     Q    -    -    -    -
7     -    -    -    -    -
6     -    -    -    Q    -
5     -    -    -    -    -
4     -    -    -    -    -
3     -    Q    -    -    -
2     -    -    -    -    Q
1     -    -    Q    -    -

Solution 4:
```

# CHESS BUT PRETTY

- 4 DIFFERENT THEMED BOARDS





# COMPUTER PLAYER

- A.I.
  - Function takes a list of legal moves as input
  - Outputs the move it wants to take

  - 1st Priority: Capture an enemy piece
  - 2nd Priority: Put enemy king in check
  - If all else fails it will do whatever move that pushes furthest into enemy territory

  There is another A.I. that plays purely random moves

```
amove myAi(int board[8][8], movesList moves){
  amove choice;
  int far = 8;
  // If all else fails, push a piece as far as it can
  // into enemy territory
  for (int i = 0; i < moves.num_moves; i++){
    if (moves.list[i].y2 < far) {
      far = moves.list[i].y2;
      choice = moves.list[i];
    }
  }
  // Second priority is to put the enemy king in check
  for (int i = 0; i < moves.num_moves; i++){
    if (moves.list[i].check) {
      choice = moves.list[i];
    }
  }
  // First priority is to capture a piece if able
  for (int i = 0; i < moves.num_moves; i++){
    if (moves.list[i].capture) {
      choice = moves.list[i];
    }
  }
  return choice;
}
```

# VIDEO PRESENTATION OF THE A.I.

Our AI (black) playing against a computer player that picks a random move (white).