Chess Game Program
Software Version 1.00: May 10, 2020

Andres Cervantes, Bradley Chu,
James Guise, Taylor Togami,
& Trenton Nguyen
(Team 16)

University of California, Irvine ▪ Samueli School of Engineering
5200 Engineering Hall Irvine, CA 926297-2700▪+1-949-824-4333▪ugengr@uci.edu▪ +1-949-824-4334

Table of Contents

Glossary

Data Type - Classification of what *type* of data will be stored into a variable. They include but not limited to numbers, characters, arrays, structures.

Structure - Similar to object oriented programming, a structure allows for a programmer to reference a variable with different data types. All data types inside a structure are called members.

Function - A function is a block of code which aims to do a specific task. Functions may sometimes contain parameters or *inputs* in order to *output* or return a certain computed data type. It is important to note that not all functions will return a value or data type.

Parameter - Sometimes called Arguments. A parameter is the input to a function.

Return value - A return value is what a function outputs based on what is entered as a parameter or in some cases, based on variables in the scope of the function.

SQUARE - In this Chess program. SQUARE is a structure that contains three members: X and Y integer coordinates for referencing the square's position in the chess board, and a pointer to structure PIECE.

PIECE - A structure used in this Chess program that contains the two char or character members *team* and *type*. *Team* is used to refer to the color that the PIECE belongs to. *Type* is used to refer to what type of piece it is (Pawn, Bishop, Rook, Knight, King, Queen)

W - The character used to refer to the white colored pieces team. It is used in the game when assigning a team to the PIECE structure and when printing the SQUARE two dimensional array for the user to see.

B (Black) - The character used to refer to the white colored pieces team. It is used in the game when assigning a team to the PIECE structure and when printing the SQUARE two dimensional array for the user to see.

P (Pawn) - The character used in the PIECE structure as *type*.

B (Bishop) - Bishop. The character used in the PIECE structure as *type*.

R (Rook) - The character used in the PIECE structure as *type*.

K (King) - The character used in the PIECE structure as *type*.

Q (Queen) - The character used in the PIECE structure as *type*.

N (Knight) - The character used in the PIECE structure as *type*.

Chess.c / Chess.h - This file is the driver of the other libraries in our Chess program. Chess.c is responsible for calling the defined functions in the other program files, which result in Chess gameplay.

AI_MinMax.c / AI_MinMax.h - Files responsible for the evaluation of all possible moves for the AI chess mode, synonymous with the scoring of the moves.

AI_PosMoves.c / AI_PosMoves.h - Files responsible for the calculation of all possible moves.

Board.c / Board.h - These program files contain all functions relevant to the creation of a playable chess board, deleting a chess board, and the monitoring of proper rules.

Check.c / Check.h - Files responsible for the rules of Check and the calculation of check in Chess.

Checkmate.c / Checkmate.h - Files responsible for the rules of Checkmate and the calculation of check in Checkmate in Chess.

Movement.c / Movement.h - Files responsible for checking if the movement desired by a user is valid.

Scoring.c / Scoring.h - Files responsible for the scoring of all movements and/or captures of each piece.

Square.c / Square.h - Square.c contains all functions relevant to the creation of a SQUARE structure, deletion of a SQUARE structure, and management of the members of the SQUARE structure.

Piece.c / Piece.h - This program files contains the function definitions of creating a PIECE structure, assigning a team or type to a piece, and also the deletion of a PIECE structure.

Stdio.h - Standard Input Output. Responsible for input and output operations in the C programming language.

Stdlib.h - Standard Library. Responsible for memory allocation and freeing operation in the C programming language.

Header File - Header files (ending in .h) are used in our chess program to define data types used in or program files.

Game Log - A file containing a list of all the moves that were made during a chess match.

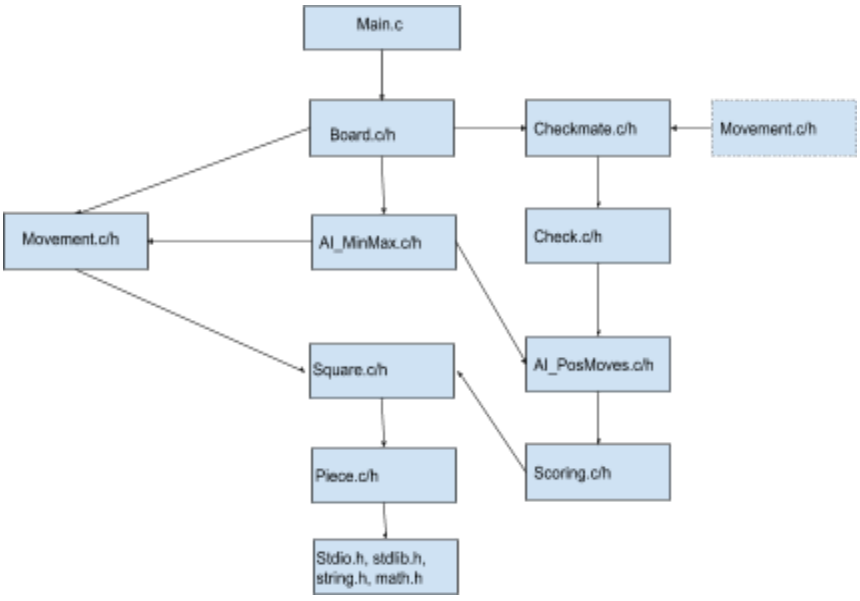
AI - Artificial Intelligence, used in the game for the user to play against a computer.

Software Architecture Overview

1.1 Main data types and structures

- The main data types and structures used for this project are:
  - The game board: 2D array
  - The pieces: structure
  - Movement: functions
  - Individual spots: structure containing pointers

1.2 Major software components



1.3 Main Module interfaces

- Setting up the board:
  - SQUARE * boardArray[8][8];
- Creating a piece:
  - PIECE * CreatePiece()
- Movement:
  - void changePieceTo(PIECE * p, int fromRow, int fromCol, int toRow, int toCol);
  - int movePieceTo(PIECE * p, int fromRow, int fromCol, int toRow, int toCol);
  - int isValidPieceMove(PIECE * p, int toRow, int toCol);
  - int isValidPawnMove(PIECE * p, int desRow, int desCol);
  - int isValidKingMove(PIECE * p, int desRow, int desCol);
  - int isValidBishopMove(PIECE * p, int toRow, int toCol);
  - int isValidRookMove(PIECE * p, int toRow, int toCol);
  - int isValidKnightMove(PIECE * p, int toRow, int toCol);
  - int isValidQueenMove(PIECE * p, int toRow, int toCol);

- The AI:
    - int AI_PosMoves(char aTeam);
    - int Scoring(PIECE * piece, int oRow, int oCol, int fRow, int fCol);
    - int AI_MinMax(char aTeam);
- Checkmate
    - int totalCheckmate(char teamBias);
    - int moveoutCheck(char teamBias);
    - int blockCheck(char teamBias);
    - int canBlock(char teamBias, int row, int col);
    - void getEnemies(char teamBias);

- Check
    - int Check(char teamBias);
    - int isAttacked(char teamBias, int row, int col);
    - Int isStillCheck(PIECE *p, int tRow, int tCol, char teamBias)

1.4 Overall program control flow

- Setup
    - Get user input for color, opponent, team, board(custom or classic), type of players
- Display/print the board
- Repeat:
    - 1st player makes a move
    - Add to game log
    - Check for en passant, castling, check, & checkmate
    - Display/print the board
    - If 2nd is in checkmate -> 1st player wins
        - Otherwise, black player makes a move
        - Add to game log
        - Check for en passant, castling, check, & checkmate
    - Display the board
    - If 1st player is in checkmate -> 2nd player wins

Installation

2.1. System requirements, compatibility

Red Hat Linux - Linux OS (RHEL-6-x86_64)
Available disk space: 100 MB

2.2. Setup and configuration

Download the latest version of the Chess Game file
Untar/unzip file (type tar -zxvf Chess_Beta_src.tar.gz)
Type “make” followed by the pressing of the Enter key. This will build and compile all program files that were installed in the previous step.

2.3. Building, compilation, installation

Type ./bin/Chess to run the chess game
Alternatively, can type make test to make the chess game and run it immediately after it’s compiled.

Documentation of Packages, Modules, and Interfaces

3.1 Data Structures

- PIECE structure (From Piece.h)

```
typedef struct {
    char team;
    char type;
    int row;
    int col;
    int moves;
    int prevRow;
    int prevCol;
} PIECE;
```

- Used to represent the chess piece that will be on the board
- General structure used to represent what team the piece is on (black or white) and the what the piece is (pawn, bishop, rook, king, queen, knight)
    - Team abbreviations: 'W' = white team and 'B' = black team
    - Piece abbreviations: 'P' = pawn, 'B' = bishop, 'R' = rook, 'K' = king, 'Q' = queen, and 'N' = knight
- Row and col are used to represent the position the piece is on the board
- Moves are used to indicate how many moves the piece has done, particularly used for the En Passant and Castling techniques
- prevRow and prevCol indicate the last position the piece was at

- SQUARE structure (From Square.h)

```
typedef struct {
    int x;
    int y;
    PIECE * piece_ptr;
} SQUARE;
```

- Used to represent the squares on the chess board
- X and y are the coordinates on the chess board which represent where the square is
- The piece_ptr points to a piece that's on the square, if there's no piece on the square then it is set to NULL.

- Board Array
```
SQUARE * boardArray[8][8];
```
(From Board.h)

    - Used to represent the physical chessboard, an 8x8 grid of squares
    - This array is a 2-D array to represent the 2-D 8x8 board of chess
- Board Data
```
typedef struct {
    PIECE * previouslyWhiteMovedPiecePtr;
    PIECE * previouslyBlackMovedPiecePtr;
    int netMoves;
}BOARD_DATA;
```
(From Board.h)

    - The previously moved pointers are used to indicate the last piece from each team that moved most recently, helps for En Passant
    - netMoves is used to count how many moves have been made

3.2 Detailed description of functions and parameters

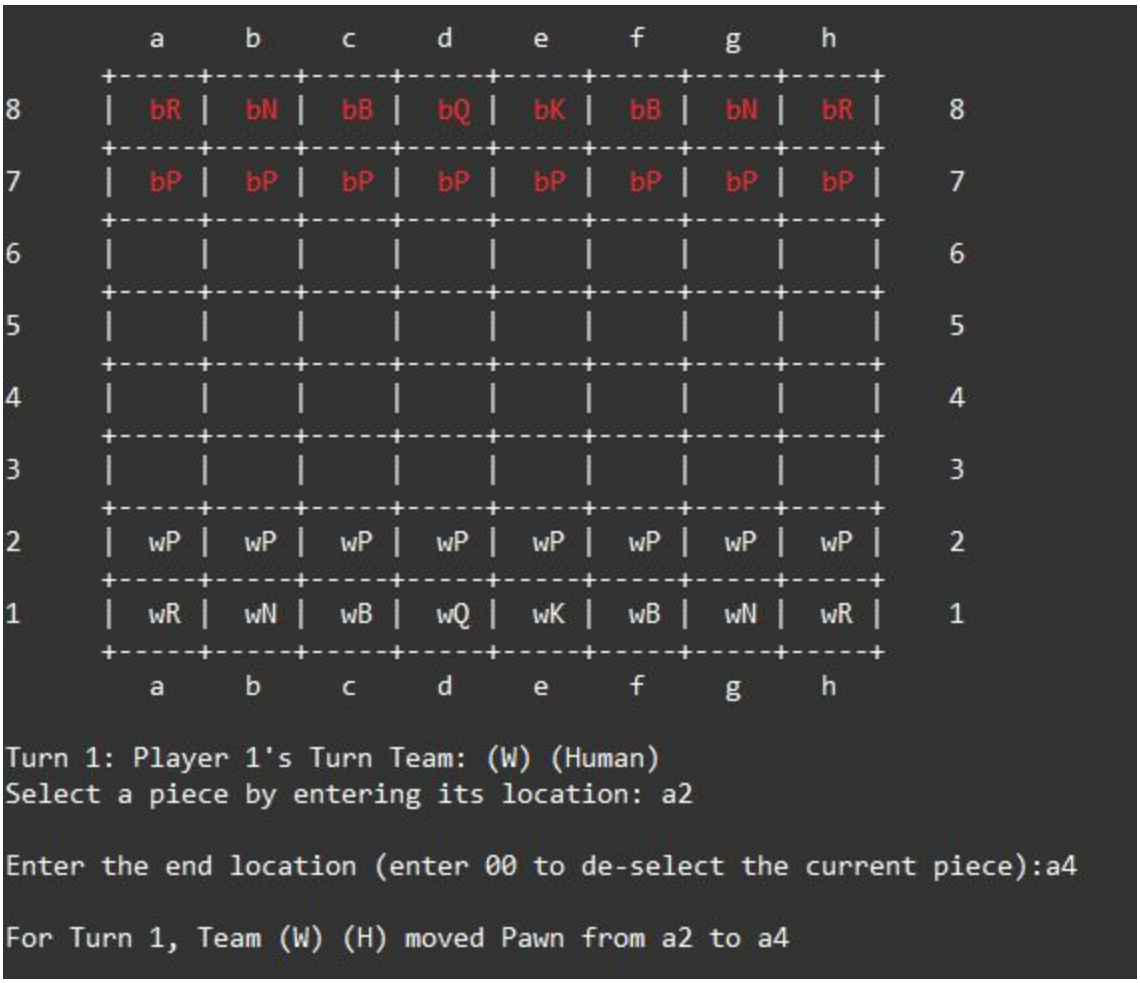
- Link “.c” files and “.h” files as well as linking them in a central Makefile, to make accessing functions and passing of any parameters easier
- CreateBoard()
    - Initializes the starting chess board based off of 2D array
- CreatePiece()
    - Creates a chess piece (uses malloc(sizeof()))
- DeletePiece(PIECE * piece)
    - Deletes a chess piece from the board (uses free())
- DuplicatePiece(PIECE * piece)
    - Makes a copy of a piece, used for movement
- DefaultPieceSetup()
    - Placing chess pieces in classic setup of chess
- CustomPieceSetup()
    - Allows user to place any piece anywhere on the board
- PrintBoard()
    - Prints out the chess board with current chess pieces
- CreateSquare()
    - Creates the square coordinates on the chess board for pieces to be able to move to (occupied new space using malloc(sizeof()))
- DeleteSquare(SQUARE * square)
    - Updates the square coordinate on the chess board where the chess piece moved from (vacant old space using free())

- PlayerTurn(char team)
    - Used to handle a player's turn by prompting a move and executing the move
- AI_PosMoves(char aTeam)
    - Finds all of the possible chess moves for that particular team/player and calls Scoring() to give each move a score
- Scoring(PIECE * piece, int oRow, int oCol, int fRow, int fCol)
    - Evaluates a chess move and gives it a score
- AI_MinMax(char aTeam)
    - Finds all current and future (limited) possible moves by calling AI_PosMoves() to find the best move at that specific turn for the AI
- movePieceTo(PIECE* p, int row, int col)
    - Moves chess piece from current spot to selected spot while also checking that a move is valid or is capturing
- main()
    - Layout of entire chess game (using while loop), taking user inputs, and calling other specific functions
- changePieceTo(PIECE * p, int fromRow, int fromCol, int toRow, int toCol);
    - Moves a chess piece without checking for any conditions.
- isValidPieceMove(PIECE * p, int toRow, int toCol)
    - Checks that any piece is making a valid move(calls each individual piece checker)
- isValidPawnMove(PIECE * p, int desRow, int desCol)
    - Checks if user made a legal pawn move or capture
- isValidKingMove(PIECE * p, int desRow, int desCol)
    - Checks if user made a legal King move or capture
- isValidBishopMove(PIECE * p, int toRow, int toCol)
    - Checks if user made a valid bishop move or capture
- isValidRookMove(PIECE * p, int toRow, int toCol)
    - Checks if user made a legal rook move or capture
- isValidKnightMove(PIECE * p, int toRow, int toCol)
    - Checks if user made a legal knight move or capture
- isValidQueenMove(PIECE * p, int toRow, int toCol)
    - Checks if the user made a legal queen move or capture
- totalCheckmate(char teamBias);
    - Checks if king is completely in check based on moveoutCheck and blockCheck
- moveoutCheck(char teamBias);
    - Checks if the valid spots around the king are in check or not
- blockCheck(char teamBias);
    - Determines if team can capture the attacking pieces that put the king in check, if not determine if team can block the pieces
- canBlock(char teamBias, int row, int col);
    - Checks if team can block the path between the king and the attacking pieces if they are not a knight or pawn

- getEnemies(char teamBias);
    - Returns an array of all of the enemy pieces
- Check(char teamBias);
    - Checks if the king's position is in the trajectory of any of the enemy pieces
- isAttacked(char teamBias, int row, int col);
    - Checks to see if any piece is in the trajectory of an enemy piece
- isStillCheck(PIECE *p, int tRow, int tCol, char teamBias)
    - Checks for all valid moves and if king will still be in check afterwards

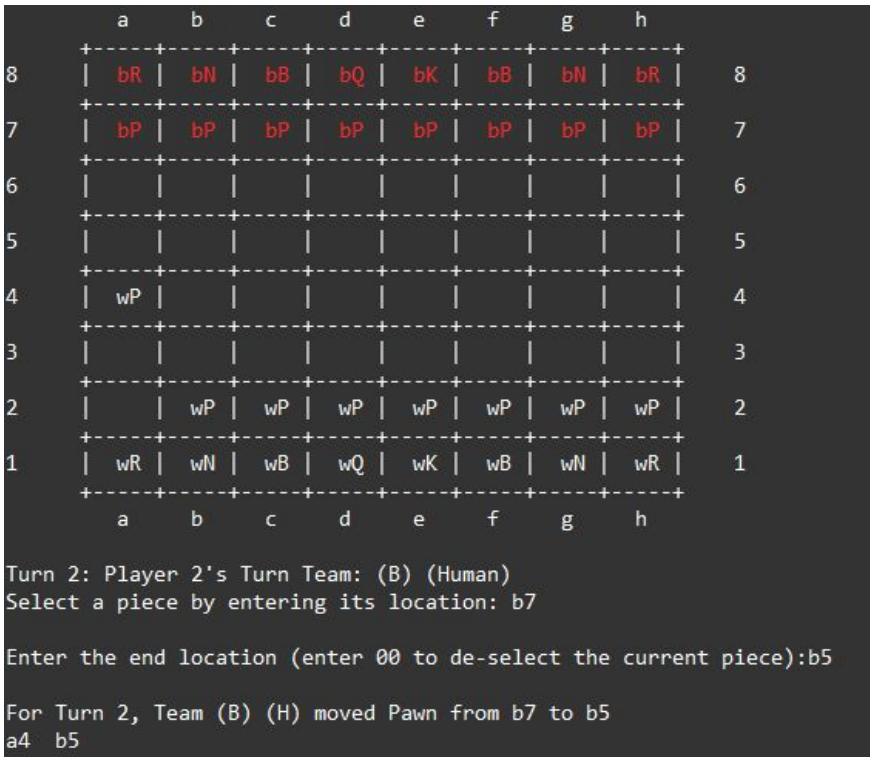
3.3 Detailed description of input and output formats

Input: User entering a piece they want to move and moving it



The program first prompted the player to enter a piece by entering its location on the board. The user then entered in f7 which is the location of the black pawn 3rd from the right. Then the program prompted the end location in which the user entered in f5.

The board will then be reprinted with the user's move already being executed. The game will now shift to the other player's turn and the process will continue until checkmate or the king is captured.



Output: Game log

The game log will record all actions made by the user and the opponent.

```
Gamelog

        a     b     c     d     e     f     g     h
      +-----+-----+-----+-----+-----+-----+-----+-----+
8     |     |     |     |     |     |     | wR  |     |    8
      +-----+-----+-----+-----+-----+-----+-----+-----+
7     |     |     |  bK |     |     |     |     |     |    7
      +-----+-----+-----+-----+-----+-----+-----+-----+
6     |     |     |     |     |     |  wR |     |     |    6
      +-----+-----+-----+-----+-----+-----+-----+-----+
5     |     |     |     |     |     |     |     |     |    5
      +-----+-----+-----+-----+-----+-----+-----+-----+
4     |     |     |     |     |     |     |     |     |    4
      +-----+-----+-----+-----+-----+-----+-----+-----+
3     |     |     |     |     |     |     |     |     |    3
      +-----+-----+-----+-----+-----+-----+-----+-----+
2     |     |     |     |     |     |     |     |     |    2
      +-----+-----+-----+-----+-----+-----+-----+-----+
1     |     |     |     |     |  wK |     |  wQ |     |    1
      +-----+-----+-----+-----+-----+-----+-----+-----+
        a     b     c     d     e     f     g     h

Turn 1: Player 1's Turn Team: (B) (Human)

For Turn 1, Team (B) (H) moved King from c7 to d7

        a     b     c     d     e     f     g     h
      +-----+-----+-----+-----+-----+-----+-----+-----+
8     |     |     |     |     |     |     | wR  |     |    8
      +-----+-----+-----+-----+-----+-----+-----+-----+
7     |     |     |     |  bK |     |     |     |     |    7
      +-----+-----+-----+-----+-----+-----+-----+-----+
6     |     |     |     |     |     |  wR |     |     |    6
      +-----+-----+-----+-----+-----+-----+-----+-----+
5     |     |     |     |     |     |     |     |     |    5
      +-----+-----+-----+-----+-----+-----+-----+-----+
4     |     |     |     |     |     |     |     |     |    4
      +-----+-----+-----+-----+-----+-----+-----+-----+
3     |     |     |     |     |     |     |     |     |    3
      +-----+-----+-----+-----+-----+-----+-----+-----+
2     |     |     |     |     |     |     |     |     |    2
      +-----+-----+-----+-----+-----+-----+-----+-----+
1     |     |     |     |     |  wK |     |  wQ |     |    1
      +-----+-----+-----+-----+-----+-----+-----+-----+
        a     b     c     d     e     f     g     h

Turn 2: Player 2's Turn Team: (W) (Human)

For Turn 2, Team (W) (H) moved Queen from g1 to g7

Kd7  Qg7#
```

In this example, the user's first turn then the second player's turn was saved into the game log. The board will always be printed after any move is made. The move will be written in text in the game log and every turn will be written to the file until checkmate or the king is captured.

Captures, check, and checkmate will also be mentioned by the game log.
Examples:

Capture:

```
For Turn 4, Team (B) (H) moved Pawn from b5 to a4
a4  xa4
```

(White team moved pawn from a2 to a4 previously, then black team had a pawn at b5 which captured the pawn at a4 which is indicated by the x in algebraic notation)

Check:

```
For Turn 11, Team (W) (H) moved Queen from e2 to e7

!----------Team B is in check! Please move out of check!----------!
```

```
For Turn 12, Team (B) (H) moved King from e8 to e7

Qxe7+  Kxe7
```

(The + indicates that the first team (W) put the other team (B) in check)

Checkmate:

```
For Turn 2, Team (W) (H) moved Queen from g1 to g7

Kd7  Qg7#
```

(In Algebraic Notation, the # indicates Checkmate)

```
!--------------------CHECKMATE: GAME OVER TEAM B LOSES--------------------!
```

The game will also print out this statement to the user to indicate checkmate

Development Plan and Timeline

4.1 Partitioning of Tasks

- Andres Cervantes
    a. User Manual: Computer Chess Section (1.1 - 1.3)
    b. Software Architecture: Glossary
    c. Code: Movement code Movement.c (Piece movement, integrating check/checkmate into movement), Board.c, Piece.c

- Bradley Chu
    a. User Manual: Functions and Features
    b. Software Architecture: Data structures
    c. Code: Makefile, Piece.c, Piece.h, Chess.c, Parts of: Square.c, Movement.c, Board.c, Check.c

- James Guise
    a. Software Architecture: Detailed description of functions and parameters (3.2)
    b. Installation
    c. Code: Makefile, AI_PosMoves.c, Scoring.c, AI_MinMax.c

- Taylor Togami
    a. User Manual: Title page, back matter
    b. Software Architecture: 1.1-1.4, reference, index
    c. Code: Square.h, Board.c, checkmate help, png files, Checkmate.c, Check.c

- Trenton Nguyen
    a. User manual: Table of Contents, Glossary
    b. Software Architecture: Partitioning of Tasks, Team Member Responsibilities (4.1-4.2)
    c. Code: Movement.c, Check.c, Checkmate.c

Timeline:
Week 1: Initial Setup of Git Hub and started setting up board, squares, and pieces. Set expectations for the team and meet and greet.
Week 2: Board, Squares, Pieces, and Movement code.
Week 3: Movement, setting up gameplay loop
Week 4: Capturing, AI.
Week 5: Debugged Movement and AI, developed check and checkmate
Week 6: Debugged check and checkmate, finished final version of program.

4.2 Team Member Responsibilities

- Manager: Andres Cervantes
    - Leads meetings and keeps meetings productive by moving from one subject to the next
    - Assigns deadlines and responsibilities with team approval
    - Turns in team assignments
- Presenter: Taylor Togami
    - Presents progress updates and effectively communicates with TAs when needed
    - Makes sure all members are clear on development plans and project infrastructure
- Recorder: James Guise
    - Documents and keeps track of important points during meetings
    - Brings up important information from previous meetings to go over or review
- Reflector: Bradley Chu and Trenton Nguyen
    - Ensures team are all agreeing on same points before moving onto next topic
    - Pass around the group’s ideas to better refine and come up with the final implementation.

Index