Chess Simulator

Software Specification

Authors:
Anna Ahn
Ishika Narain
Filiberto Alvarez
Bilal Malik
Christian Zeidan
Derek Tang
Producer: B-Chess Studios
Affiliation: BBB Software Inc.
Version: 1.0.0 / Release

Table of contents:

Glossary:

Software Spec Terms

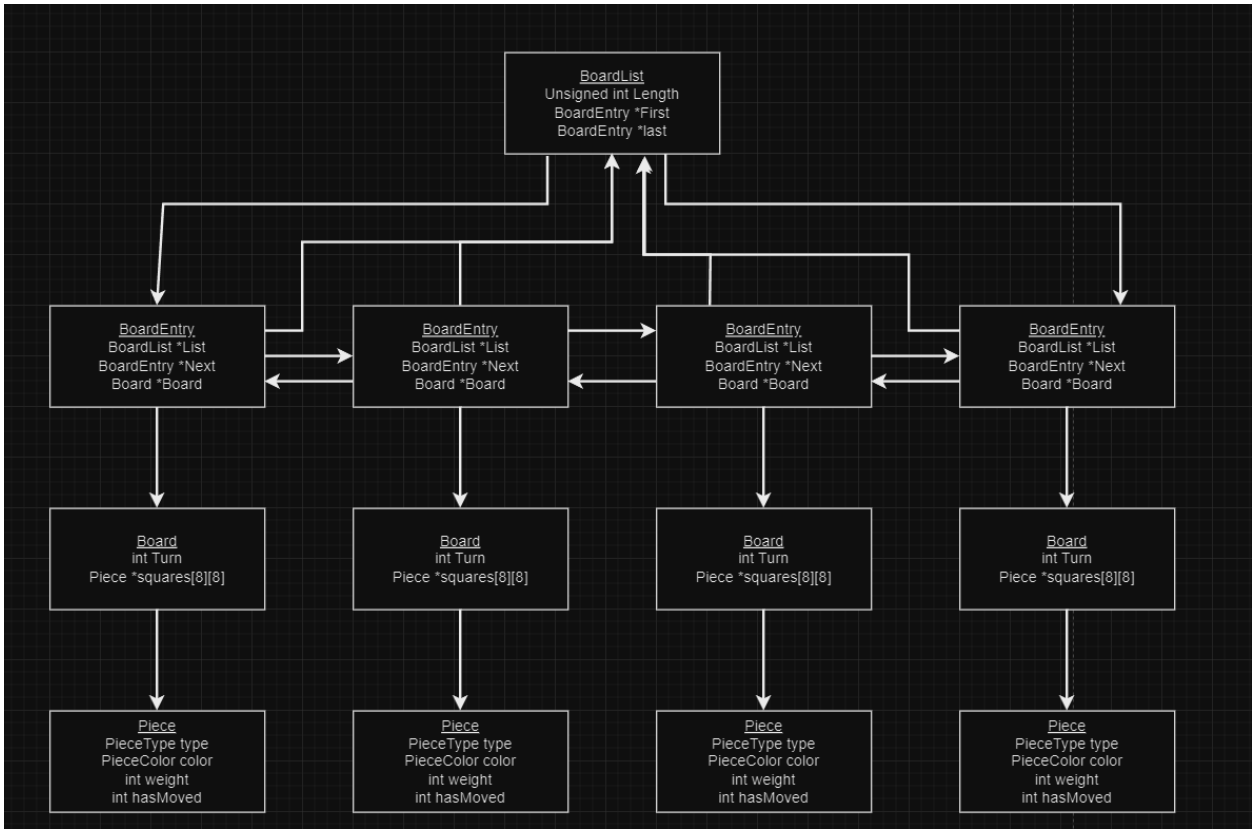
1. *Int* - a type that represents and integer
2. *Struct* - a type of custom data with a defined structure
3. *Enumerator* - a way to number a set of items with a list
4. *Pointer* - A type that links itself to a location and memory and calls on it whenever the pointer is called
5. *Doubly linked list* - a set of struct linked to each other in memory through pointers in their struct
6. *Void* - a key word that denotes a function return no value
7. *NULL* - a keyword that denotes that a type has no value
8. *Git* - A repository containing the code and files used
9. *Push* - Push code and files to the git repository
10. *Pull* - Pull code and files from the git repository
11. *Main* - The main function where the code is executed from
12. *Checkmate* - No move for the player possible which would get the king out of check
13. *Check* - When a move creates an attack on the king but there are moves to evade.
14. *En Passant* - Pawn captures another pawn that is on its same row and adjacent file, the enemy pawn had to have made its initial 2 square advance.
15. *Castling* -A special move where if either side of the king has no pieces between the rooks, the king and rook can move at the same time. During this move, the king and rook cross over each other, with the king moving 2 squares toward the rook, and moving the rook directly next to the king. The move can not be done when: the king is in check, the king has already moved, when there are pieces in between, the rook has moved, and when the king can be moved into a spot it would be checked.
16. *Knight* - Can move 8 spaces, where it can move 2 steps forwards or backward and 1 step to the left or right. The knight may also go 2 steps to the left or right and 1 step forward or backward. Has the ability to move over pieces.
17. *Bishop* - Can move diagonally across the board.
18. *Rook* - Can move horizontally and vertically across the board
19. *Queen* - can move horizontally, vertically across the board
20. *King* - Can move in any direction, but only one step at a time. Must never move into check and there is a special “castling” move for the king

21. *Pawn* - Can move only forward towards the end of the board, but captures 1 square diagonally. From its initial position, a pawn may make two steps, otherwise only a single step at a time. If the pawn reaches the end of the board, it is automatically promoted to another piece (usually a queen)
22. *Increment* - In timed games, a player may be granted added time after moves.

1 Software Architecture Overview

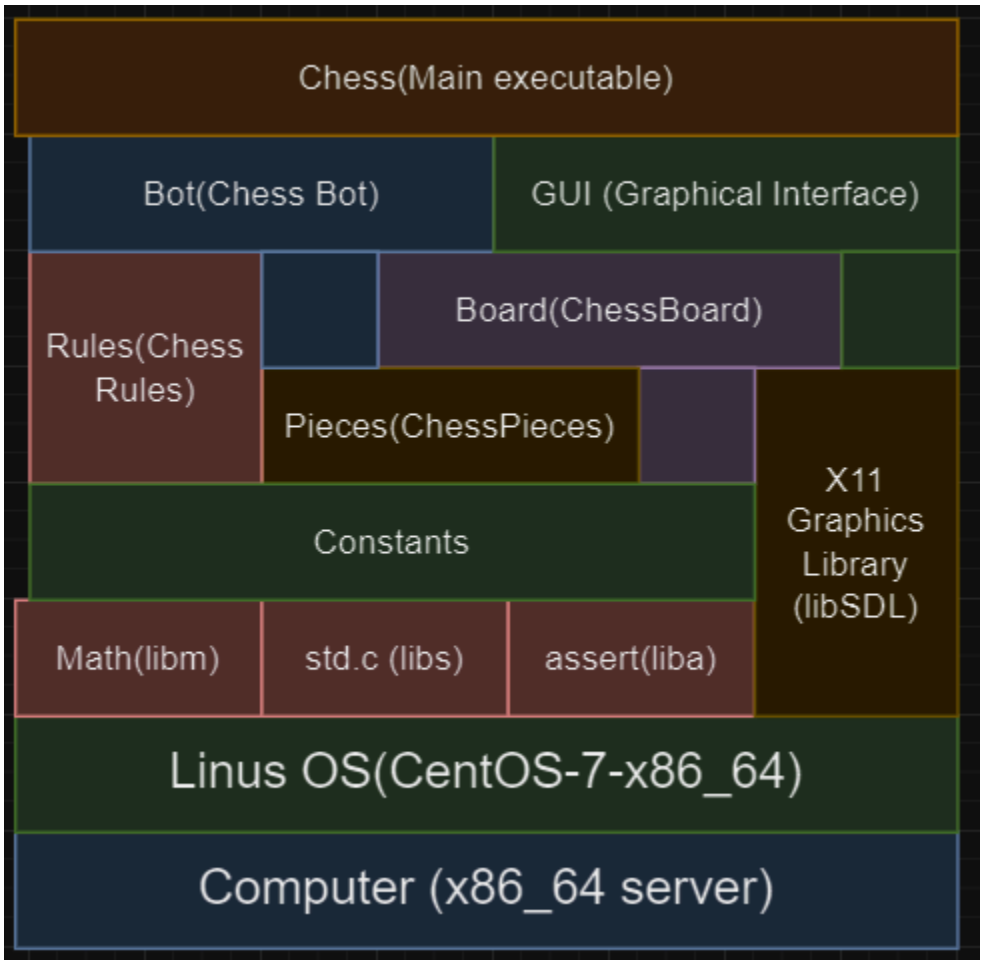
1.1 Main data types and structures

- Doubly linked list: Structure that saves/keeps the current/past moves made by the user/bot.
- Double arrays: Structure used to create/determine which chess pieces are on the chess board.
- Struct: defined data types
    - BoardList
    - BoardEntry
    - Board
    - Piece
- Int: used to represent pieces/turns/positions/etc.
- Pointer: used to point to other positions in the list or board
- Diagram of struct hierarchy



1.2 Major software components

• Diagram of Module Hierarchy



- Software Components
  - Chess
    - The main file which controls the moves and screens of the chess program
  - Bot
    - Custom file containing the code for the Chess bot
  - GUI
    - Custom file build on top of graphics library to display chess
  - Board
    - Custom File containing the code for the Linked list and Board
  - Pieces
    - Custom file contain the code for the Pieces struct

    - Rules
        - Custom file containing the code for Chess rules
    - X11 Graphics Library
        - C graphics Library
    - Constants
        - Custom file containing program constants
    - Math
        - Generic Math C Library
    - STD.c
        - Standard C Library
    - Assert
        - Library for asserting structs

1.3 Module interfaces

Constants.h

- Provides: Needed macros for other files
- Requires: Nothing
- Exported Functions: None

Rules.h

- Provides: Chess rules for other files
- Requires: Constants.h Board.h Pieces.h
- Exported Functions: See Sections 3.2 & 3.3 for detailed explanation of functions

Board.h

- Provides: Board structure for other files
- Requires: Constants.h Pieces.h
- Exported Functions: See Sections 3.2 & 3.3 for detailed explanation of functions

Pieces.h

- Provides: Pieces structure for other files
- Requires: Nothing
- Exported Functions: See Sections 3.2 & 3.3 for detailed explanation of functions

Bot.h

- Provides: Chess bot for other files
- Requires: Constants.h Board.h Pieces.h Rules.h Log.h
- Exported Functions: See Sections 3.2 & 3.3 for detailed explanation of functions

Log.h

- Provides: Functions to write to a .txt file for other files
- Requires: Board.h
- Exported Functions: See Sections 3.2 & 3.3 for detailed explanation of functions

1.4 Overall program control flow

While loop to exit program

- Switch statement for each state/menu
- Main:
    - PrintMenu(); -> need commands
    - take input; -> need commands
- Choose Color:
    - printf("choose color");
    - take input;
- Game state
    - Create list
    - Create board
    - print board; → need commands
    - While loop to check for end game
        - Chess implementation:
            - Make board - board.c
            - Append board - Board.c
            - Checks the flag for check
            - While loop to check validity
                - Check for piece - in board.c
                - Check for valid moves - in pieces.c
                - Get the user input move - main.c
                - Check user input is a valid space
            - Make the move - main.c
            - Log move
            - Check end (set end game flag to 1)
                - Check.c -> rules.c *flags
                    - Check if king is in check
                - Checkmate.c -> rules.c *flags
                    - Check if king is mated

        - Stalemate.c -> rules.c *flags
          - Check if game is in stalemate
      - Check flags <- main.c
  - Save the log to a text file
  - Delete the linked list
  - Winner/continue prompt

2 Installation

2.1 System requirements

- Windows/Mac with built in terminal
- Able to use/SSH into a linux server
- Download Xming(windows) to be able to display the chess GUI in separate window

2.2 Setup and configuration

- mkdir chess
- cd chess
- Download files chess directory in linux server
- tar -xvzf Chess_V1.0_src.tar.gz
- make all
- cd bin/
- ./Chess
- To clone repo use the command “git clone team12bondi.eecs.uci.edu:repos/chess.git”

2.3 Uninstalling

- Navigate to the parent directory of "chess/"
- Type "rm -rf chess/" to remove the entire "chess/" directory

2.2 Setup and configuration

- On linux server create git username with the given command
  - git config –global user.name “Team 12”
  - git config –global user.email team12@eecs.uci.edu
  - mkdir repos
  - chmod 770 repos
  - ls -la repos

    - cd repos
    - git init –bare –shared=0660 chess.git
    - ls -la
    - cd
    - mkdir project
    - cd project
- On linux server update git email with the given command
    - git add Chess_UserManual.pdf
    - git commit -m “First commit, add Chess_UserManual.pdf”
    - git push origin master
    - git status
    - ls -l
- Create a directory on your local linux server
    - git config –global user.name “First Last”
    - git config –global user.email user@eecs.uci.edu
    - mkdir project
    - cd project
    - git clone team12bondi.eecs.uci.edu:repos/chess.git
    - ls -l
    - cd chess
- Create a clone of the git repository using the command below
    - git clone ~/repos/chess.git
- Navigate to the chess folder in the directory
    - cd chess
- Type make into command line and hit enter
- Type ./chess to run the program

2.3 Building, compilation, installation

- make clean
- rm * (only do this in the program directory)
- cd .. -> rm -r -f [program directory]

3 Documentation of packages, modules, interfaces

3.1 Detailed description of data structures

Doubly linked list:

- BoardList
    - Unsigned int Length
        - Denotes length of the list
    - BoardEntry *First
        - Point to the first entry in the list
    - BoardEntry *Last
        - Points the last entry in the list
    - Source Code
        -
```
struct BoardList{
    unsigned int Length;    /* Length of List */
    BENTRY *First;          /* Pointer to First entry */
    BENTRY *Last;           /* Pointer to last entry */

};
```
- BoardEntry
    - BoardList *List
        - Points to the list it originated from
    - BoardEntry *Next
        - Points the the next entry in the list
    - BoardEntry *Prev
        - Points to the previous entry in the list
    - Board *board
        - Points to its matching board struct
    - Source Code
        -
```
struct BoardEntry{
    BLIST *List;    /* Pointer to BoarList */
    BENTRY *Next;   /* Pointer to next entry in List */
    BENTRY *Prev;   /* Pointer to Previous entry in list */
    BOARD *Board;   /* Pointer to current Board */

};
```
- Board
    - int Turn
        - Denotes the players turn for its matching board
    - Piece *squares[8][8]

        - A double array of Piece type pointers to represent the 8 by 8 board in memory
    - Source Code
        -
```
struct Board{
    int Turn;   /* int to represent current players turn */
    PIECE *squares[BOARDSIZE][BOARDSIZE]; /* An Array 8 x 8 of piece pointers */
};
```

- Piece
    - PieceType type
        - Represents the type of piece through a enumerator
    - PieceType color
        - Represent the color of the piece through a enumerator
    - int weight
        - Represents the weight/points of the piece
    - Int hasMoved
        - Flag for castling to check if rook has moved
    - Source Code
        -
```
// Enumeration for different types of chess pieces.
typedef enum {
    Pawn = 1, Knight, Bishop, King, Queen, Rook
} PieceType;

// Enumeration for piece colors.
typedef enum {
    White = 1, Black
} PieceColor;

// Struct to represent the properties of a chess piece.
typedef struct Piece {
    PieceType type;       // The type of the piece.
    PieceColor color;     // The color of the piece (which player controls it).
    int weight;           // The point value of the piece.
    int hasMoved;         // Flag to indicate if the piece has moved (for castling).
} PIECE;
```

3.2 Detailed description of functions and parameters

chess.c

- int main(void);
    - The main function contains the finite state machine which controls the current state of the program

rules.c

- void updateLastMove(LastMove *lastMove, int startX, int startY, int endX, int endY, int pieceMoved, int pieceCaptured, int wasPawnDoubleStep);
  Helper function that updates the last move made for the pawn piece
- void removePiece(BOARD *board, int x, int y);

Helper function to remove the piece from the board

- int isSquareUnderAttack(BOARD *board, int x, int y, int kingColor);
  Helper function to determine is a specific square is threatened
- int isMoveLegal(PIECE *piece, int startX, int startY, int endX, int endY, BOARD *board, LastMove *lastMove);
  Function to determine if the move by a piece is legal
- int isPawnMoveLegal(PIECE *piece, int startX, int startY, int endX, int endY, BOARD *board, LastMove *lastMove);
  Function to determine if specifically pawn move is legal
- int canPerformCastling(BOARD *board, int kingColor, int rookPositionX);
  Function to determine if a piece can perform castling
- int isCheckMate(BOARD *board, int kingColor);
  Determines if king is mated
- int isMoveSelfCheck(BOARD *board, int startX, int startY, int endX, int endY);
  Determines if a move puts itself in check
- int isKingInCheck(BOARD *board, int kingColor);
  Determines if king is in check
- int makeAndValidateMove(BOARD *board, int startX, int startY, int endX, int endY);
  Function to make the move and board and validate if the move is legal
- int isStalemate(BOARD *board, int playerColor);
  Function to see if stalemate has occured
- void pawnPromotion(BOARD *board, int x, int y, int pieceType);
  Function to promote pawn

board.c

- BoardList *CreateBoardList(void)
  - Creates boardlist
- BoardEntry *CreateBoardEntry(BoardList *BoardList)
  - Created a new Boardentry and appends it to the Board List
- void AppendBoardEntry(BoardList *BoardList, BoardEntry *BoardEntry)
  - Append Board Entry to the list
- void UnAppendBoardentry(BoardList *BoardList)
  - unappend Board Entry from the list
- Board *CreateBoard(BoardList *BoardList)
  - Creates a new board
  - Add it to a board entry
- Void DeleteBoardList(BoardList *BoaredList)

        - Delete the entire BoardList
    - Void DeleteBoardEntry(BoardEntry *BordEntry)
        - delete the Board Entry and its matching Board
    - Void DeleteBoard(Board *Board)
        - Delete the Board
    - Void WriteTurn(Board *Board, int turn)
        - Add the turn to the turn type in the board struct
    - BoardList *InitializeBoard(void)
        - Initialize the game by creating the list and the firs board with its pieces

pieces.c

- PIECE *CreatePiece(int type, int color, int weight)
    - Creates a piece pointer with desired attributes given
- Void DeletePiece(PIECE *piece)
    - Deletes piece
- int CheckPieceColor(PIECE *piece)
    - Checks to see what color the piece is
- int CheckPieceType(PIECE *piece)
    - Checks to see what type the piece is
- int CheckPieceWeight(PIECE *piece)
    - Checks to see what Weight the piece is
- Int SetPieceMoved(PIECE *piece,);
- void Promote(PIECE *piece,, int type)
    - Updates the piece struct to promote the piece

bot.c

- int botmove(*Board, validMove)
    - Bot calculates all moves it can make and selects random one
- int bestmove(*Board, validMove)
    - Bot calculates best move (point system)
- int firstmove(*Board, validMove)
    - Bot makes first move (rand())

gui.c

- void printMainMenu();
    - Prints the main menu to the terminal
- void printSetting();
    - Prints the settings to the terminal
- void printColorChoice();

    - Prints the color choice menu to the terminal
- void printChessBoard();
    - Prints the chess board to the terminal
- void printGameEnd();
    - Prints the type of ending to the terminal

log.c

- Null StoreMove(*pointer)
    - Stores the last move made to the log
- Null RemoveMove(void)
    - Removes the last move made in the log

3.3 Detailed description of input and output formats

chess.c

- int main(void);
    - Input: void
    - Output: error code (Expects a 1 for no error)

rules.c

- void updateLastMove(LastMove *lastMove, int startX, int startY, int endX, int endY, int pieceMoved, int pieceCaptured, int wasPawnDoubleStep);
    - Input: LastMove positions
    - Output: void
    - void removePiece(BOARD *board, int x, int y);
        - Input: Board, piece position
        - Output: void

- int isSquareUnderAttack(BOARD *board, int x, int y, int kingColor);
    - Input: Board, king color, and piece position
    - Output: void
- int isMoveLegal(PIECE *piece, int startX, int startY, int endX, int endY, BOARD *board, LastMove *lastMove);
    - Input: Piece struct, starting and ending positions, and board struct
    - Output: Integer

  - int isPawnMoveLegal(PIECE *piece, int startX, int startY, int endX, int endY, BOARD *board, LastMove *lastMove);
        - Input: Piece struct, starting and ending positions, and board struct

- Output: Integer

- int canPerformCastling(BOARD *board, int kingColor, int rookPositionX);
  - Input: Board struct, king color, and the position of the rook
  - Output: Integer
- int isCheckMate(BOARD *board, int kingColor);
  - Input: Board struct, king color
  - Output: Integer
- int isMoveSelfCheck(BOARD *board, int startX, int startY, int endX, int endY);
  - Input: Board struct, starting and ending positions
  - Output: Integer
- int isKingInCheck(BOARD *board, int kingColor);
  - Input: Board struct, king color
  - Output: Integer
- int makeAndValidateMove(BOARD *board, int startX, int startY, int endX, int endY);
  - Input: Board struct, starting and ending positions
  - Output: integer
- int isStalemate(BOARD *board, int playerColor);
  - Input: Board sturct, and player's color
  - Output: Integer

void pawnPromotion(BOARD *board, int x, int y, int pieceType);
- input: Board Struct, starting and ending positions, piece type
- output: void

- BoardList *CreateBoardList(void)
  - Input: void
  - Output: pointer to new BoardList
- BoardEntry *CreateBoardEntry(BoardList *BoardList)
  - Input: current BoardList
  - Output: Pointer to new board entry
- void appendBoardentry(BoardList *BoardList, BoardEntry *BoardEntry
  - Input: pointer to board entry
  - Output: void
- Void unappendBoardentry(BoardList *BoardList)
  - Input: current BoardList
  - Output: void
- *Board createBoard(BoardList * BoardList)

        - Input: The current Board List
        - Output: pointer to board
    - Void DeleteBoardList(BoardList *BoardList)
        - Input: current BoardList
        - Output: void
    - Void DeleteBoardEntry(BoardEntry *BoardEntry)
        - Input: current BoardList
        - Output: void
    - Void DeleteBoard(Board *Board)
        - Input: current BoardList
        - Output: void
    - Void WriteTurn(Board *Board, int turn)
        - Input: current Board, and desired turn int
        - Output: void
    - BoardList *InitializeBoard(void)
        - Input: void
        - Output: the current board list

pieces.c

- pieces *CreatePiece(int type, int color, int weight)
    - Input: int to represent type, color, and weight
    - Output: Pointer to piece created
- Void DeletePiece(Piece *piece)
    - Input: Pointer to desired piece
    - Output: Null
- int CheckPieceColor(Piece *piece)
    - Input: Pointer to piece desired
    - Output: int to represent what color it is
- PieceType CheckPieceType(Piece *piece)
    - Input: Pointer to piece desired
    - Output: PieceType enumerator
- Int CheckPieceWeight(Piece *piece)
    - Input: Pointer to piece desired
    - Output: int to represent points or weight of piece
- Int SetPieceMoved(Piece *piece)
    - Input: Pointer to piece desired
    - Output: flag to see if castelling is possible

- Void Promote(Piece *piece, PieceType type)
    - Input: desired piece and new type
    - Output: Null

bot.c

- int botmove(*Board, validMove)
    - Input: pointer to the board and valid moves
    - Output: list of all the valid moves the bot can make
- int bestmove(*Board, validMove)
    - Input: pointer to the board and valid moves
    - Output: list of all the best moves teh bot can make based on the point value of player's pieces
- int firstmove(*Board, validMove)
    - Input: pointer to the board and valid move
    - Output: list of all the valid moves the bot can make

gui.c

- void printMainMenu();
    - Input: void
    - Output: ASCII out to terminal
- void printSetting();
    - Input: void
    - Output: ASCII out to terminal
- void printColorChoice();
    - Input: void
    - Output: ASCII out to terminal
- void printChessBoard();
    - Input: void
    - Output: ASCII out to terminal
- void printGameEnd();
    - Input: void
    - Output: ASCII out to terminal

log.c

- Null StoreMove(*pointer)
    - Input: pointer to array of move made
    - Output: void
- Null RemoveMove(void)
    - Input: void

- Output: void

4 Development plan and timeline

4.1 Partitioning of tasks

- Main (chess.c)
- constants.h (constants and structure definitions)
- Chess Backend
    - rules.c / rules.h
    - board.c / board.h
    - pieces.c / pieces.h
- The bot (bot.c / bot.h)
- GUI (gui.c / gui.h)
- Log (log.c / log.h)

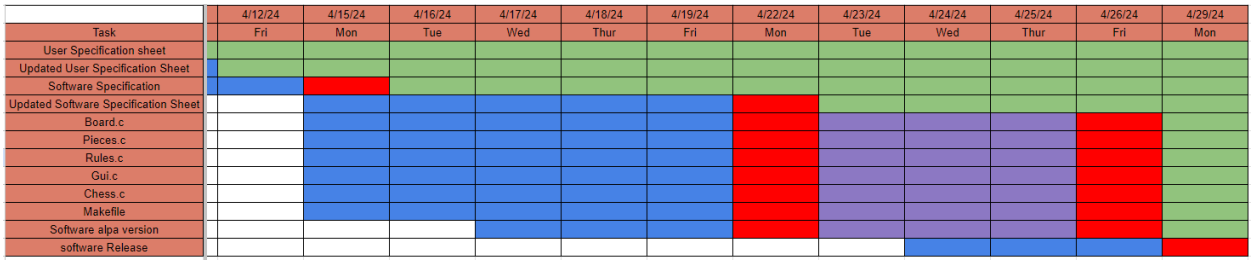
4.2 Team Member Responsibilities

- Ishika: GUI
- Anna: Chess Backend / The Bot
- Fili: Chess Backend (structs)
- Bilal: Chess Backend (rules) / GUI
- Christian: The Bot (“Zot Blue”) / Chess Backend
- Derek: main (chess.c) / ASCII

Timeline

Due Date	
Version 1 Inprogress	
Completed	
Version 2 inprogress	

	Date	4/1/24	4/2/24	4/3/24	4/4/24	4/5/24	4/8/24	4/9/24	4/10/24	4/11/24
Task	Day of Week	Mon	Tue	Wed	Thur	Fri	Mon	Tue	Wed	Thur
User Specification sheet										
Updated User Specification Sheet										
Software Specification										
Updated Software Specification Sheet										
Board.c										
Pieces.c										
Rules.c										
Gui.c										
Chess.c										
Makefile										
Software alpa version										
software Release										



References

Chess Rules on Canvas given by TA

[FIDE Laws of Chess]

Index

Copyright

© 1987 - 2024 B-Chess. All Rights Reserved