EECS 22L

Winter 2016

Team 11: nøl / C Gets Degrees

Chess v1.0 Software Specification

Table of Contents

Glossary

Files

- ai.c: Functions for the computer player to calculate and make a move.
- ai.h: Structure definition for “moveValue,” and function definitions
- board.c: Code for how the board will be set up.
- board.h: Definitions of the spaces (cells) for board.c.
- cell.c: Code for cells.
- cell.h: Definition for structure “cell.”
- chess.exe: The executable for the game (running this will run the game).
- defs.h: Definitions for global variables, including error code definitions.
- errors.c: Code for displaying error messages.
- errors.h: Function declarations for errors.c.
- gameDisplay.c: Function for updating the game board after moves.
- gameDisplay.h: Function declaration for updating the game board in the corresponding c file.
- license.txt: Contains terms and conditions, as well as permission for use of GNU, for development of this software.
- main.c: The code that links and relates other .c and .h files in order to create the executable.
- piece.c: Contains functions for printing locations, and checking available moves.
- piece.h: Declaration of structure for pieces, and the pieces.

Structures

- cell: Structure for spaces on the board to keep track of which piece to display at what location, and to help determine the validity moves.
- piece: Structure to keep track of each player’s pieces location, type, validity of special moves (such as castling), current location, and previous location.
- moveValue: Structure used in ai.c to keep track of a specified piece’s potential move and value of that move.

In-Game Functions

- [piece location] [target location]: Moves a piece to a different position, or displays an error if move is invalid.
- [piece location]: Shows available moves for a specified piece, if any.
- exit/quit: Closes the game.
- undo: Reverts to last move.

Software Architecture Overview

Data Types and Structures

In our implementation of Chess, we have three main data types that are dependent on each other.

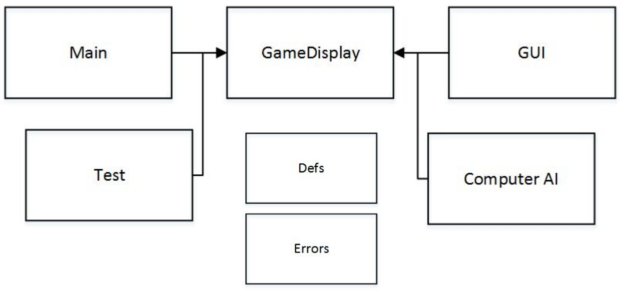
- Board
- Cell
- Piece
- moveValue

*Details regarding each module can be found in the "**API Documentation**" section*

Software Components, Module Hierarchy

Components will be divided into their relevant functions. Each component may have several modules.

- Main Component
- Test Components
- Game Components
- Computer A.I. Components
- GUI Components



Overview of Module Interfaces

Dependencies for the main modules

Global Dependencies: These are modules that are used throughout the program

- Errors: For error messages
- Defs: Constants and definitions

Main: Initial setup of the game

- gameDisplay: To facilitate the text-based displaying of the game board

Test: Used for testing of game logic

- gameDisplay: To facilitate the text-based displaying of the game board
- Computer AI: Calculating the best possible moves

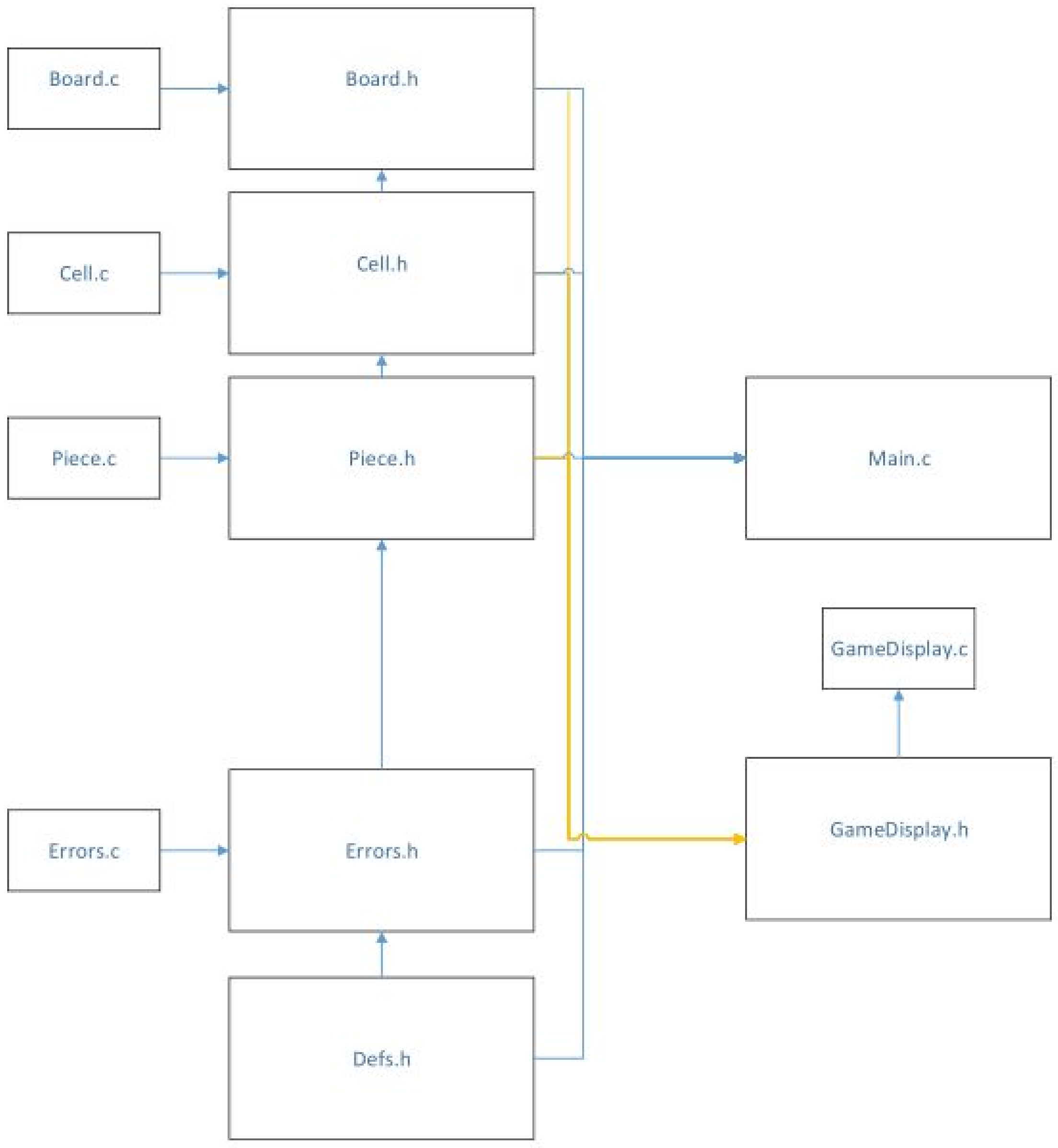
GUI: A top level component to facilitate the graphical displaying of the game board

- gameDisplay: To facilitate the text-based displaying of the game board

Computer AI: Calculating the best possible moves
- Board: Top level component of game logic

gameDisplay: To facilitate the text-based displaying of the game board
- Board: Top level component of game logic

Flowchart of Module Interfaces



Module Control Flow Diagram

Click here for the overall control flow diagram for Chess gameplay. [PDF]

Installation

System Requirements:

For Windows:

- Make sure the computer can run a terminal through a ssh client (such as PuTTy or MobaXterm).
- Computer should be able to access the EECS servers, specifically the Zuma (zuma.eecs.uci.edu) or CrystalCove (crystalcove.eecs.uci.edu) Linux servers and run a terminal.

For Macs:

- Open the program called Terminal on your Mac. It should already be an installed application on your Mac.
- Make sure to go to the New Remote Connection in the Shell option and access the EECS servers by typing zuma.eecs.uci.edu or crystalcove.eecs.uci.edu in the secure shell box.

Compiling and Configuration

- Connect to the EECS server
- Copy the game files from the server to your current directory by typing in the command line: `cp ~L16T11/Chess_Alpha_src.tar.gz .`
- Unpack the archive file
- Type `make install` to compile the executable program
- To play the game, type `chess` into your terminal and the program will initiate.
- Type `exit` or `quit` at any time to close the program.

Uninstalling

- To uninstall the game, navigate to the directory containing the game.
- Now type `make clean` to remove the files.
- You will no longer have the files for the game, but you can reinstall by typing `make install` again.

API Documentation

Game Logic

- board (inherits from cell)
    - cell *getCell (int cellID, struct board *board)
        - Returns a cell pointer given an ID and the board
    - board *createNewGame()
        - Creates a board and fills it with the starting pieces
    - board *createBoard()
        - Initializes a new empty board with 64 cells + 1 captured cell (cellID = -1)
    - void updateBoard()
        - Refreshes the board elements with the current locations of each piece.
    - void deleteBoard(struct board *board)
        - Frees only the board struct from memory.
    - struct board { int turn, cell *minus1, *cell0, *cell1, …, *cell63 }
        - Stores all cells and the current turn starting from 0. White turns are even, black turns are odd.
- cell (inherits from piece and board)
    - cell *createCell (int cellID, board *board)
        - Creates a new cell "object"
    - void replacePiece(struct cell *cell, piece *p)
        - Puts a piece inside a cell
    - void deleteCell(struct cell *cell)
        - Frees the cell and any pieces in it
    - void deleteAllCells(board *board)
        - Deletes all cells using deleteCell()
    - struct cell
        - char printPiece
            - A single character used for gameDisplay that represents the piece type and color/player located in the cell (if any)
        - int cellID
            - An integer corresponding to a cell's position on the board ranging from -1 to 63. -1 is used for captured pieces.
        - piece *piece
            - The piece that is contained within the cell
        - board *board

- piece (inherits from cell)
    - struct piece
        - enum hasMoved
            - Contains values `false`, `true` (in that order). A flag to indicate whether the piece has moved during the game.
        - enum player
            - Contains values `black`, `white` (in that order). A variable to indicate which player the piece belongs to.
        - enum type
            - Contains values `pawn,knight,king,queen,rook,bishop` (in that order). A variable to indicate what type the piece is.
        - int castleFlag
            - [DEPRECATED] A flag to indicate whether the piece can castle this turn.
        - int epFlag
            - A flag to indicate whether the piece can en passant this turn.
        - cell *loc
            - A pointer to the piece's current location.
        - cell *prev
            - A pointer to the piece's previous location.
    - int *checkAvailMoves(piece *p)
        - Checks where a specified piece can move. Used in movePiece
        - Returns a pointer with the list of available positions (cellIDs) that piece can move to. This list is terminated with a -2.
        - Uses subsidiary functions checkPawnMoves, checkRookMoves, ...
            - checkKingCheck is a subsidiary of checkKingMoves, and prevents the King from moving into check
    - int movePiece (piece*p, cell *target)
        - Moves a piece to the specified target.
        - Returns 0 if move is valid
        - Returns -1 if move is invalid
        - Returns -2 if no available move
        - Returns 1 for a pawn promotion
        - Returns 2 for kingside castle
        - Returns 3 for queenside castle
        - Uses subsidiary functions movePawn, moveKnight, ...
    - void printLoc(piece *p)

        - Prints the location of a specified piece with a new line after it.
    - void printCell(int cellID)
        - Prints the location of a specified cellID.
    - piece *createPiece(enum player player, enum type type, cell *cell)
        - Creates a piece and places it inside a cell.
    - void deletePiece(piece *p)
        - Deletes a piece. Is used inside deleteCell().
    - int pawnPromotion(piece *p)
        - Handles a pawn's promotion. Returns 1-4 based on the piece that it is promoting to (1: Queen, 2: Knight, 3: Rook, 4: Bishop)

<u>Game Display / User Input</u>

- errors
    - Error codes are defined in defs.h
    - void printe(int code)
        - Prints the description of an error corresponding to an error code. Terminates the program if an unexpected error occurs.
    - void printp(int code, piece *p)
        - Prints the description of an error corresponding to an error code, as well as the name of the relevant piece and its location. Terminates the program if an unexpected error occurs.
- gameDisplay
    - void updateGameDisplay(board *board)
        - Refreshes the display with the current locations of each piece, any messages pertaining to that move, and handles user input.
        - Uses printMessage and handleInput
    - void updateMessage(board *board)
        - Same as updateGameDisplay but without the board
    - void handleInput(board *board)
        - Receives input from the user on any desired action and performs functions accordingly.
        - Uses all the below functions
    - int toID(char *loc)
        - Compares a string from stdin with the possible cell locations (formatted like a4). Also handles commands like "exit", "help", and "load."
        - Returns the corresponding cellID
        - Returns -2 if there is no match (invalid command error)
        - Returns -3 if "exit" or "quit" is entered

        - Returns -4 if “undo” is entered
        - Returns -5 if “help” is entered
        - Returns -6 if “load” is entered
    - int moveSwitch(piece *piece, int destCell)
        - Attempts to move a piece to the destination cellID.
        - Returns 0 if the move does not result in a check
        - Returns 1 if the move results in check
        - Returns 2 if the move results in checkmate
        - Returns 3 if the move results in stalemate
    - void checkAvailMovesSwitch(piece *piece)
        - Used when a player selects a piece. Prints formatted available moves.
    - void createMoveLog()
        - Creates and writes a header in movelog.txt
    - void writeMoveLog(int turn, piece *piece, int capture, int promo, int castle, int check)
        - Appends move history to movelog.txt
    - const char * returnCell(int cellID)
        - Instead of printing a formatted cell location like printCell, this returns it.
    - void loadGame(char *fname, board *board)
        - Loads sequential moves from a text file.

Computer A.I.

- ai (inherits from cell, piece, board)
    - void aiMove(int diff, int team, board *board)
        - Determines the ai’s difficulty and sets parameters for aiChoice accordingly. No return.
    - aiChoice(int team, board *board, int lookAhd)
        - Decides ai’s move, and executes it.
    - int oppTeam(int team)
        - Returns the integer value of the opposing team (0 or 1, else fails).
    - void DeleteMoveValue(moveValue *moveValue)
        - Cleans and dealloctes memory of a specified moveValue. No return.
    - moveValue *ABPrune(int *piecePositions, int *enemyPositions, moveValue *bestMax, moveValue *bestMin, int lookahd, int team, int max, board *board)

        - Returns a moveValue containing the piece, the cell it will move to (int), and the value of that move (int). Works by alpha-beta pruning, which compares the value of potential moves of both players, tries to find a balance between the least harmful move and the most valuable move by checking if the next move is worse than the current best one, and if so it cuts off that branch and continues until it decides it's best move.
    - moveValue *calcMoveValue(int team, board *board, cell *moveLoc, int nextCellLoc)
        - Calculates and returns a moveValue containing the specified piece (from moveLoc), the id of the next cell, and the value of that move.
    - moveValue *CreateMoveValue(piece *p, int next, int value)
        - Allocates memory for and returns a moveValue containing the information specified by the parameters.
    - int pieceMoveCount(piece *p)
        - Returns the number of moves a specified piece has.
    - int *checkAIAvailMoves(piece *p)
        - Returns a pointer to the list of available moves of a piece. Similar to checkAvailMoves, but won't display errors for a piece with no moves.
    - int *checkPiecePos(int team, board *board)
        - Returns an integer pointer to a list of cell ids containing the pieces of a specified teams pieces.

Input/Output Formats

- For non-GUI version of the program, user makes moves through formatted strings
- Move log file is outputted as a formatted text file

Development Plan and Timeline

Timeline:

Note that dates indicate benchmark progress. Work is continuous, and can be seen in detail in the Github repository after public release.

1/6/16: Initial partitioning of tasks for writing user manual
1/13/16: Completed draft of user manual and partitioned tasks for software specification
1/20/16: Completed draft of software specification and partitioned tasks for programming modules
1/27/16: Alpha version of program complete. Consists of some piece tests, game setup menu, and the displaying of the board.
1/28/16: Started work on computer AI
1/29/16: Pawn movement complete
1/30/16: Move log module complete. Bishop, Rook, Queen movement complete.
1/31/16: Load moves module complete. Knight, King movement complete. All piece movement complete.
2/1/16: Check detection complete. Computer AI complete. All remaining bugs fixed.
2/3/16: Timer and undo move complete.
2/4/16: Public release.

Partitioning of tasks/Team contribution:

Aaron - Pawn movement, Data Types and Module Definitions, Move Log, Game Loading, User Input
Alden - Computer A.I.
Brad - Bishop initial testing, Computer A.I.
Bryan - Rook, Bishop, Queen movement
Carlos - King movement, Check detection
Jeff - Queen initial testing, Game timer
Syed - Knight movement
Tam - Knight movement

<u>Team member responsibilities:</u>

Pawn Movement: Checking available moves for the pawn, including en passant.

Bishop Movement: Checking available moves for the bishop

Rook Movement: Checking available moves for the rook

Knight Movement: Checking available moves for the knight

Queen Movement: Checking available moves for the queen

King Movement: Checking available moves for the King. This also includes castling, making sure that it cannot be in check, and winning conditions.

Data Types and Modules: Module dependencies, creating a framework/structure for data type inheritances.

User Input: Handling of possible user inputs like "quit", "undo", "a2 a3", "a2", etc. See API: gameDisplay for details.

Move Log: Output of formatted text file with moves of the last played game. See API: gameDisplay for details.

Game Loading: Reading and handling of text file with sequential moves. See API: gameDisplay for details.

Copyright and License

Project Contributors:

- Aaron “A.I.” Liao, 90811748
- Alden “The Pawn” Chico, 83197101
- Brad “Bishop” Leiserowitz, 21437905
- Bryan “The Board” Ou, 1174696
- Carlos “Castling” Cortes Gutierrez, 80207156
- Jeffrey “My Name’s Jeff” Han, 51999492
- Syed “Stalemate” Azeemuddin, 18393977
- Tam “The Timer” Nguyen, 12276989

Bibliography/References

"Let's Play Chess." USCF, n.d. Web. 13 Jan. 2016.
http://stackoverflow.com/questions/1139271/makefiles-with-source-files-in-different-directories
https://github.com/Arwid/chess
http://www.cplusplus.com/reference/cstdio/
http://www.tutorialspoint.com/c_standard_library/c_function_fgets.htm

Index