EECS 22L

Winter 2016

Team 11: nøl / C Gets Degrees

Chess v1.0 Software Specification

Table of Contents

Glossary

Files

- 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.

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.

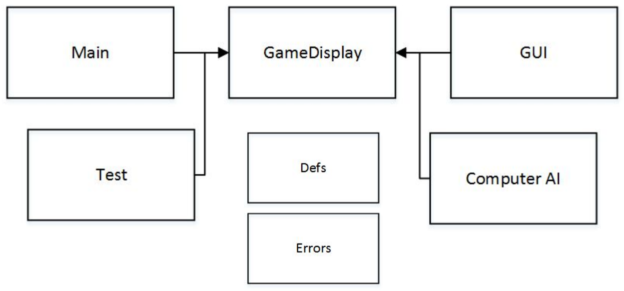
- Board
- Cell
- Piece

*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

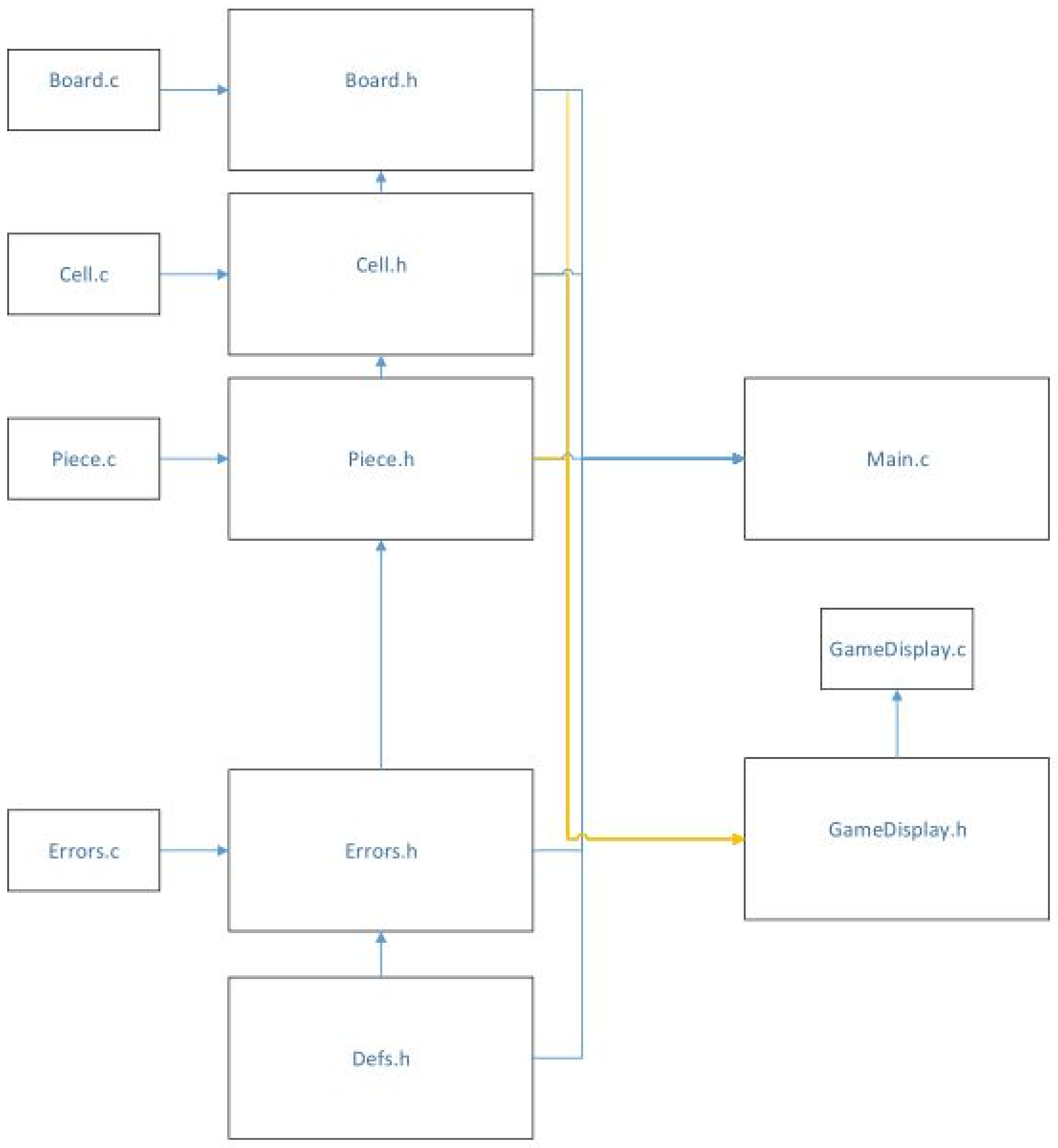
GUI: A top level component to facilitate the graphical displaying of the game board

- 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.c/h
    - 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 { cell *minus1, *cell0, *cell1, …, *cell63 }
- cell.c/h
    - 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.c/h
    - int *checkAvailMoves(piece *p)

        - Checks where a specified piece can move. Returns a pointer with the list of available positions (cellIDs) that piece can move to.
        - uses subsidiary functions checkPawnMoves, checkKingMoves, etc.
    - void printLoc(piece *p)
        - Prints the location of a specified piece.
    - piece *createPiece(enum player player, enum type type, cell *cell)
        - Creates a piece and places it inside a cell
    - void deletePiece(struct piece *p)
        - Deletes a piece. Is used inside deleteCell().
    - 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
            - 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.
- errors.c/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.c/h
    - void updateGameDisplay()

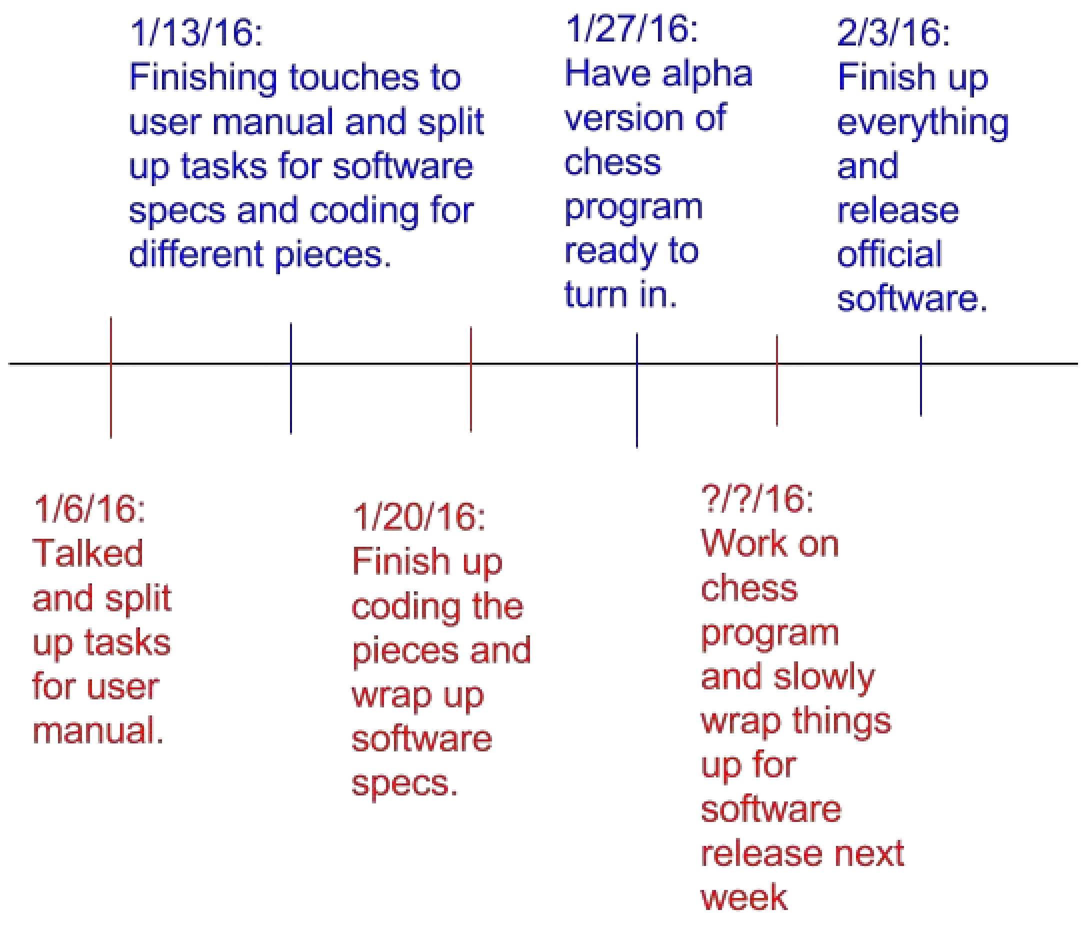
        - Refreshes the display with the current locations of each piece.
    - void updateBoard()
        - Refreshes the board with pieces moved.
    - void handleInput()
        - Receives input from the user on any desired action.

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:


1/13/16: Finishing touches to user manual and split up tasks for software specs and coding for different pieces.
1/27/16: Have alpha version of chess program ready to turn in.
2/3/16: Finish up everything and release official software.
1/6/16: Talked and split up tasks for user manual.
1/20/16: Finish up coding the pieces and wrap up software specs.
?/?/16: Work on chess program and slowly wrap things up for software release next week

Partitioning of tasks:

Aaron - Pawn movement, Data types and Structure definitions, A.I.
Alden - Pawn movement, A.I.
Brad - Bishop movement, A.I.
Bryan - Rook movement
Carlos - King movement
Jeff - Queen movement
Syed - Knight movement, User Input
Tam - Knight movement, User Input

Team member responsibilities:

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 Structures: Module dependencies, creating a framework/structure for data type inheritances.

User Input: Handling of possible user inputs like “quit”, “undo”, “a2 a3”, “a2”, etc.

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

"Let's Play Chess." USCF, n.d. Web. 13 Jan. 2016.

Index