Chessbook

Chessence (v1.0): Software Specification

Chandler Ditolla
Tanner Emerson
Lambert Nguyen
Garineh Shamirian
Prithvi Undavalli

2-4-2017

Table of Contents

Glossary

Artificial Intelligence: The computers ability to differentiate between a good or bad move.

enum: C data type to associate words with numbers for readability and convenience.

evince: Linux command to access multiple document formats.

gtar: Linux command to compress multiple files into one easily portable package.

GUI: Graphical user interface used to display the board and its pieces.

struct: C data type to easily package other linked data types in one source for convenience.

1. Software Architecture Overview

1.1 Main data types and structures

```
enum COLOR{
   EMPTYCOLOR = 0,
   BLACK = -1,
   WHITE = 1
};

enum CLASS{
   EMPTY = 0,
   PAWN = 10,
   ROOK = 50,
   KNIGHT = 30,
   BISHOP = 31,
   QUEEN = 90,
   KING = 2000
};

struct MLIST{
   int Length;
   MENTRY *First;
};

struct MENTRY{
   MLIST *List;
   MENTRY *Next;
   MOVE *Move;
};

struct MOVE{
   int isCapture;
   PIECE *sPiece;
   PIECE *mPiece;
   int pointValue;
};


struct PIECE{
   CLASS Class;
   COLOR Color;
   int hasMoved;
   int row;
   int col;
   int enPassant;
};
```

```
struct PLAYER{
   COLOR Color;
   MLIST *moveList;
   int kRow;
   int kCol;
   int isHuman;
};

struct BOARD {
   PLAYER *Active;
   PLAYER *Inactive;
   PIECE *b[8][8];
   int Depth;
};
```

1.2 Major software components

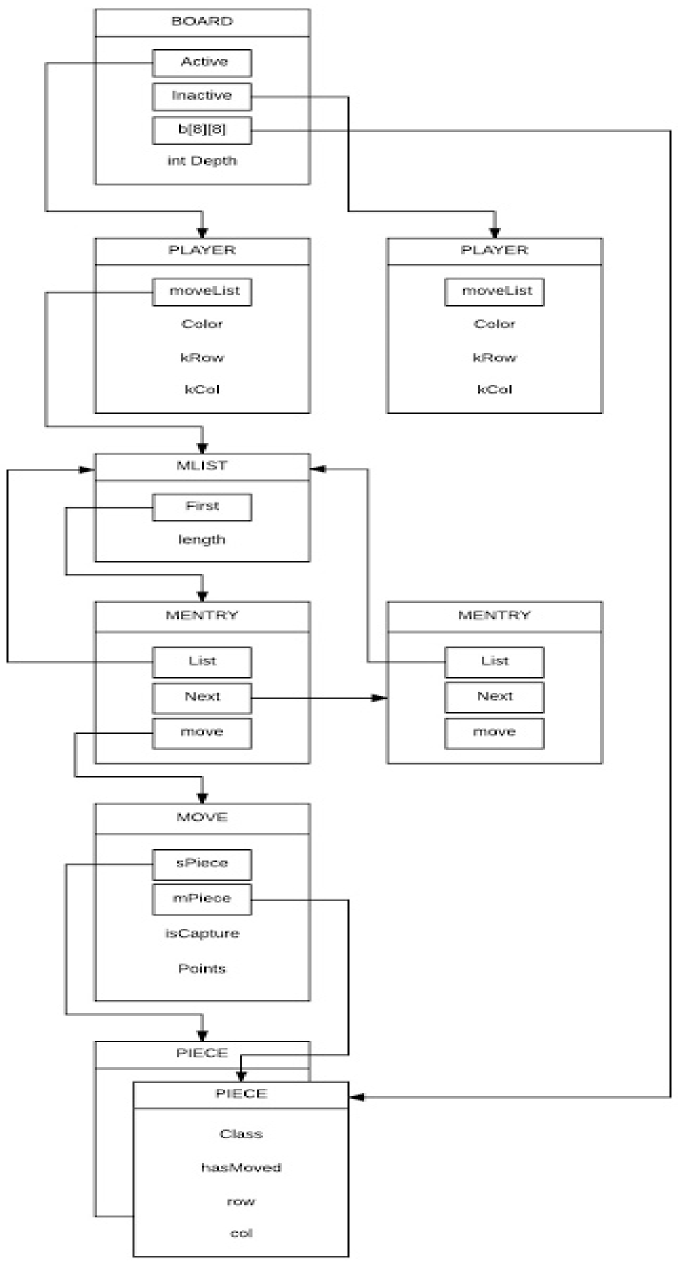
Chess (Main Module)
Rules (chess)   Strategy(AI)   Print Board   Move I/O
Std. C (libc)   math(libm)
Linux OS (RHEL-6-x86_64)

1.3 Module interfaces

```
void GetBestMove(BOARD *board);

BOARD *CopyBoard(BOARD *Oldboard, BOARD * Newboard);

int GetBestMax(BOARD *board, int depth, int alpha, int beta);

int GetBestMin(BOARD *board, int depth, int alpha, int beta);

int EvalBoard(BOARD *board);

int moveMenu(BOARD *gameBoard);

int isMoveValid(int initLocRowInt, int initLocColInt, int finalLocColInt, int

finalLocRowInt, BOARD *gameBoard);

CLASS PromotionPrompt();

void SaveCaptureLog(PIECE *pieceType, char name[]);

void SaveMessage(BOARD *game, char message[]);

char* GetPiece(PIECE *pieceType);
```

```
void SaveLog(PIECE *pieceType, BOARD *game, char move1[],char move2[], char
FILENAME[]);

int PrintMainMenu();

void LoadLog();

void EndOfGameLog();

int PrintEndMenu();

void Banner();

void LoadHelp();

void GenerateAllMoves(BOARD *gameBoard);

void GenearteMoves(PIECE *p, BOARD *gameBoard);

BOARD *Move(MOVE *move, BOARD *gameBoard);

MOVE *CreateMove(PIECE *p, PIECE *m, int isCapture);

void DeleteMove(MOVE *move);

void isCheck(PIECE *p, PIECE *m, BOARD *gameVoard, int isCapture);

void EmptyMoveList(MLIST *moveList);

void DeleteMoveList(MLIST *moveList);

void SortMoveList(MLIST *moveList);

MENTRY *CreateMoveEntry(MOVE *move);

MOVE *DeleteMoveEntry(MENTRY *e);

void AppendMove(MLIST *moveList, MOVE *move);

void PrintBoard(BOARD *board);
```

1.4 Overall Program Control Flow



2. Installation

2.1 System Requirements

Chessence is made for Linux. The computer requires a shell and the ability to run executables. It requires a console so that the Chessence can communicate with the user. The computer requires a monitor.

2.2 Setup and configuration

1. gtar xvzf chessence.tar.gz
2. evince chess/doc/chessence.pdf
3. chess/bin/chess

2.3 Building, Compilation, and Installation

1. Extract tar.gz.
2. Change into newly extracted directory.
3. Run build command: “make Chessence”.
4. Run command: “make test”

3. Documentation of Packages, Modules, interfaces

3.1 Detailed description of data structures

```
struct MLIST{
        int length;
        MENTRY * First;
};

struct MENTRY{
        MLIST *List;
        MENTRY * Next;
        MOVE *move;
};

struct MOVE{
        int isCapture;
        PIECE *sPiece;
        PIECE *mPiece;
        int pointValue;
};
```

MLIST, MENTRY, and MOVE structures create a single linked list in order to keep track of all of the possible moves that each piece can make at any given time in the game.

```
enum COLOR{
    EMPTYCOLOR = 0,
    BLACK = -1,
    WHITE = 1
};

enum CLASS{
    EMPTY = 0,
    PAWN = 10,
    ROOK = 50,
    KNIGHT = 30,
    BISHOP = 31,
    QUEEN = 90,
    KING = 2000
};
```

The enumerations are created in order to increase the readability and clarity of the program.

```
struct PIECE{
   CLASS Class;
   COLOR Color;
   int hasMoved;
   int row;
   int col;
   int enPassant;
};
```

The piece structure tracks all of the characteristics (color and class) of each piece, its location and whether not it has moved for special moves(en passant and castling).

```
struct PLAYER{
   COLOR Color;
   MLIST *moveList;
   int kRow;
   int kCol;
   int isHuman;
};
```

The player structure is used to track the move list and also to track the location of the king.

```
struct BOARD {
           PLAYER Active;
           PLAYER Inactive;
           PIECE *b[8][8];
           int Depth;
  };
```

The board structure will be used to track which which player is active/inactive at any given time. It will also have a board of PIECES that will keep track all of the active pieces on the board at any given time. The depth portion will be used in AI to determine how many levels (both black and white for one level) of play that the computer has simulated.

3.2 Detailed description of functions and parameters

void GetBestMove(BOARD *board);

Goes through the list of moves and compares each move point to calculate the best move after n-depth.

BOARD *CopyBoard(BOARD *Oldboard, BOARD * Newboard);

Creates a copy of the board with all of it's piece locations, active/inactive players and the depth so that it can be manipulated without losing data.

int GetBestMax(BOARD *board, int depth, int alpha, int beta);

Used in the AI to find the best move for white.

int GetBestMin(BOARD *board, int depth, int alpha, int beta);
Used in the AI to find the best move for black.
int EvalBoard(BOARD *board);
Used to add up the total score of a gameboard and is used for the bound of alpha beta pruning.
int moveMenu(BOARD *gameBoard);
Takes a user input for what piece and where they would like to move on the baord.
int isMoveValid(int initLocRowInt, int initLocColInt, int finalLocColInt, int finalLocRowInt, BOARD *gameBoard);
Checks the move menu input to make sure that the move is legal.
CLASS PromotionPrompt();
Prompts the user for the piece they would like to promote their pawn to.
void SaveCaptureLog(PIECE *pieceType, char name[]);
Records what pieces have been captured in the log.
void SaveMessage(BOARD *game, char message[]);
Appends save log with a message.
char* GetPiece(PIECE *pieceType);
Receives the class and color from the location of the board.
void SaveLog(PIECE *pieceType, BOARD *game, char move1[],char move2[], char FILENAME[]);
Appends the save log with the pieces moved in the last turn.
int PrintMainMenu();
Prints the main menu whenever called so that the user can make a choice on how they would like to proceed.
void LoadLog();
Opens the log.txt file so that the log of moves can be printed on the screen for the user to reade.
void EndOfGameLog();
Opens the log text file and prints all of the moves that were made during the previous game.
int PrintEndMenu();
At the end of the game, a menu is printed so that the user can make a choice on how they would like to continue.
void Banner();
When the game begins this function is called to display our banner with out application name.
void LoadHelp();
Opens the help.txt file so that the help can be printed to the screen for the user.
void GenerateAllMoves(BOARD *gameBoard);
Used to go through the board and generate all of the valid moves at any given time in the game.
void GenearteMoves(PIECE *p, BOARD *gameBoard);
Used to go through the board and generate all of the valid moves at any given time in the game.
BOARD *Move(MOVE *move, BOARD *gameBoard);
The function takes in a move and updates the gameboard to reflect the selected move.

MOVE *CreateMove(PIECE *p, PIECE *m, int isCapture);

Creates a move entry and frees all of its memory.

void DeleteMove(MOVE *move);

Deletes the move entries and frees all of the memory.

void isCheck(PIECE *p, PIECE *m, BOARD *gameVoard, int isCapture);

Checks all of your opponents moves to see if any of them are a threat to your king.

void EmptyMoveList(MLIST *moveList);

Ensures that all of the moves have been deleted so that when you delete the move list, you do not lose the locations.

void DeleteMoveList(MLIST *moveList);

Frees the memory associated with the move list after checking that all of the entries have been deleted.

void SortMoveList(MLIST *moveList);

Sorts the move list so that the AI can remove poor moves from the list. This allows the AI to execute its move more quickly.

MENTRY *CreateMoveEntry(MOVE *move);

Allocates memory for a move and assigns the move to that location.

MOVE *DeleteMoveEntry(MENTRY *e);

Deletes the move entries and releases all the of the memory associated with that move.

void AppendMove(MLIST *moveList, MOVE *move);

If a move is deemed to be legal and valid, it appends it to the list.

void PrintBoard(BOARD *board);

Takes the board as an input and runs through loops to display the board in the terminal.

MOVE GetBestMove(BOARD * board);

Goes through the list of moves and compares each move point to calculate the best move after n-depth.

BOARD SimMove(MOVE * move, BOARD * gameboard);

Takes a move operation and the gameboard. Calls copy board to make a copy for further operations, executes the move and returns the new simulated board.

void CopyBoard(BOARD * Oldboard, BOARD * Newboard);

Takes the old board and a uninitialized board and points the newboard to the old boards information.

void DestroyCopy(BOARD * Copied);

Deallocates the memory created from copyboard function.

void GenerateMoveList(PIECE *p, BOARD *gameboard);

For a specific chess piece it appends the respective players MLIST with all legal moves.

void AppendMove(MOVE *move, BOARD *gameboard);

Called by GenerateMoveList and simply appends the list of moves.

MOVE CreateMove(PIECE *p, PIECE *m, int isCapture);

Uses a initial location of a piece to generate a new legal move and return it.

void isCheck(BOARD *gameboard, PIECE *p, PIECE *m, int isCapture);

Checks if the given move will result in the king being in check.

void DeleteBoard(BOARD * board);

Deletes the board at the end of the game and frees all memory that was previously allocated for the board.

BOARD *InitializeBoard(void);

Creates the initial board when the game begins and allocates memory for all of the pieces. It will also call to have memory allocated to create player.

PLAYER *CreatePlayer(void);

Will be called from the initialize board function and will allocate memory for a player and then assign them to the board.

void DeletePlayer(PLAYER *player2Delete);

Will be called from the initialize board function and will delete the players and free the memory at the end of the game.

void PrintBoard(BOARD * board);

Takes board as an input and displays the board in the terminal.

void PrintMainMenui(void);

Prints the main menu when first running program.

void SaveLog(char start[],char end[], char FILENAME[]);

Opens and appends a log file after every move from player has been inputted.

void LoadLog(void);

Displays a log of moves from most recent game.

3.3 Detailed description of input and output formats

The user first selects an option of what kind of game to play. The menu selection can be made by entering a number between 1 and 5. The game displays a board in ascii format. It shows all of the pieces and has different characters representing the pieces. The black pieces have a 'X' inside of them and the white pieces have spaces inside of them. The moves are made by entering the position of the piece that the user desires to move and the location that the piece should go to. For example, "e5 e6". This would move the piece at e5 to e6. To castle, the user moves the king two spaces and the code will detect that the user wants to castle. The user can also quit the game.

4. Development and Timeline

4.1 Partitioning of tasks

Chandler Ditolla: Artificial Intelligence & Movement Logic
Tanner Emerson: GUI, Visual Output, Artificial Intelligence & Movement Logic
Lambert Nguyen: Menu & Options
Jesse Campbell: Artificial Intelligence & Movement Logic
Prithvi Undavalli: Menu, Movement, I/O

4.2 Team member responsibilities

Chandler Ditolla: Artificial Intelligence
Tanner Emerson: GUI and Assignment Turn-in
Lambert Nguyen: Makefile and Menu
Jesse Campbell: Game Logic and Move Validation
Prithvi Undavalli: I/O

5. Back Matter

Copyright:

References

Index