Chess XIV

Software Specification

Team 1

Hanchel Cheng

Quan Chau

Kevin Duong

Jamie Lee

Ryan Morrison

Andrew Trinh

University of California: Irvine

Table of Contents

Glossary 1

1: Software architecture overview 2

1.1 Main data types and structures 2

1.2 Major software components 4

1.3 Module interfaces 4

1.4 Overall program control flow 5

2: Installation 7

2.1 System Requirement 7

2.2 Setup and Configuration 7

2.3 Building, compilation, installation 7

3: Documentation of packages, modules, interfaces 8

3.1 Detailed description of data structures 8

3.2 Detailed description of functions and parameters 11

3.3 Detailed description of input and output formats 15

3.4 Detailed description of artificial intelligence for computer player 15

4: Development plan and timeline 17

4.1 Partitioning of tasks 17

4.2 Team member responsibilities 17

Back Matter 18

Copyright 18

Index 19

Glossary

AI: Artificial Intelligence

GUI: Graphical User Interface

Model: The set of modules responsible for the rules and logic of program

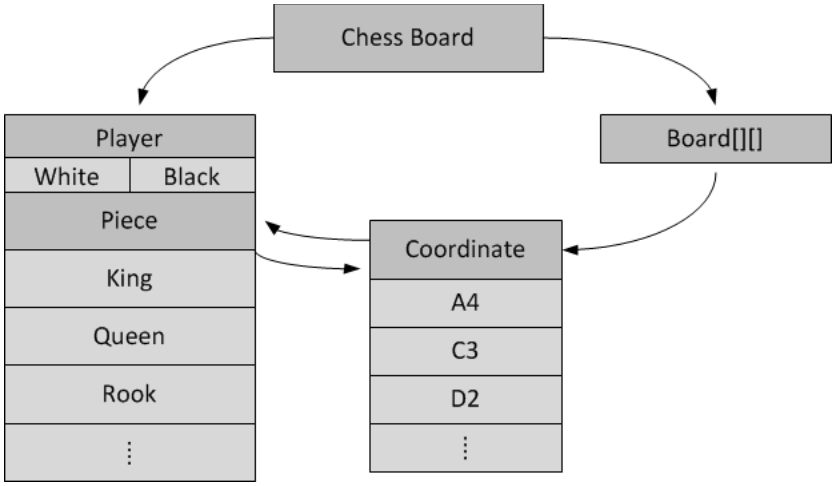
Control: The set of modules responsible for program flow

View: The set of modules responsible for program display

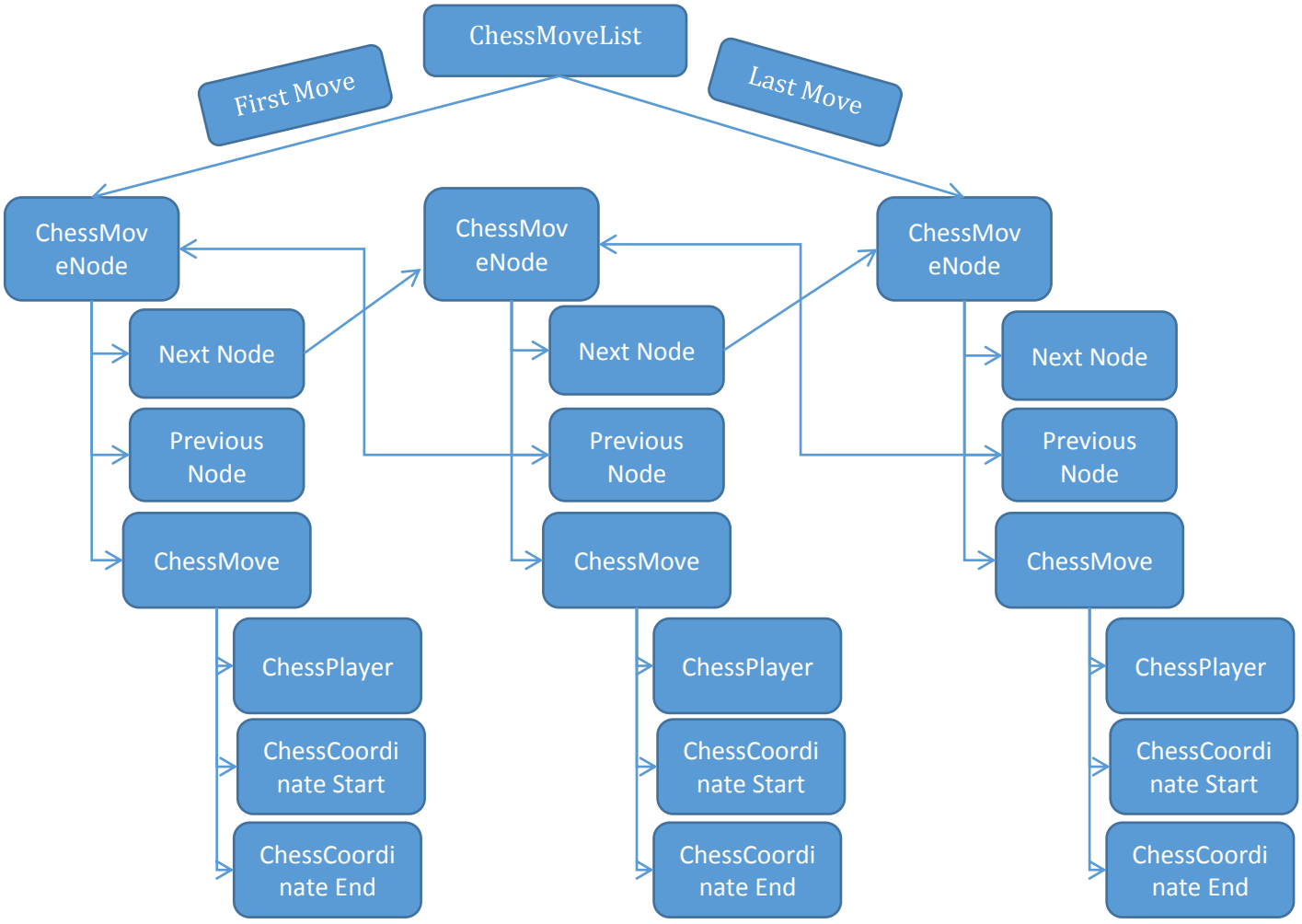
1: Software architecture overview

1.1 Main data types and structures

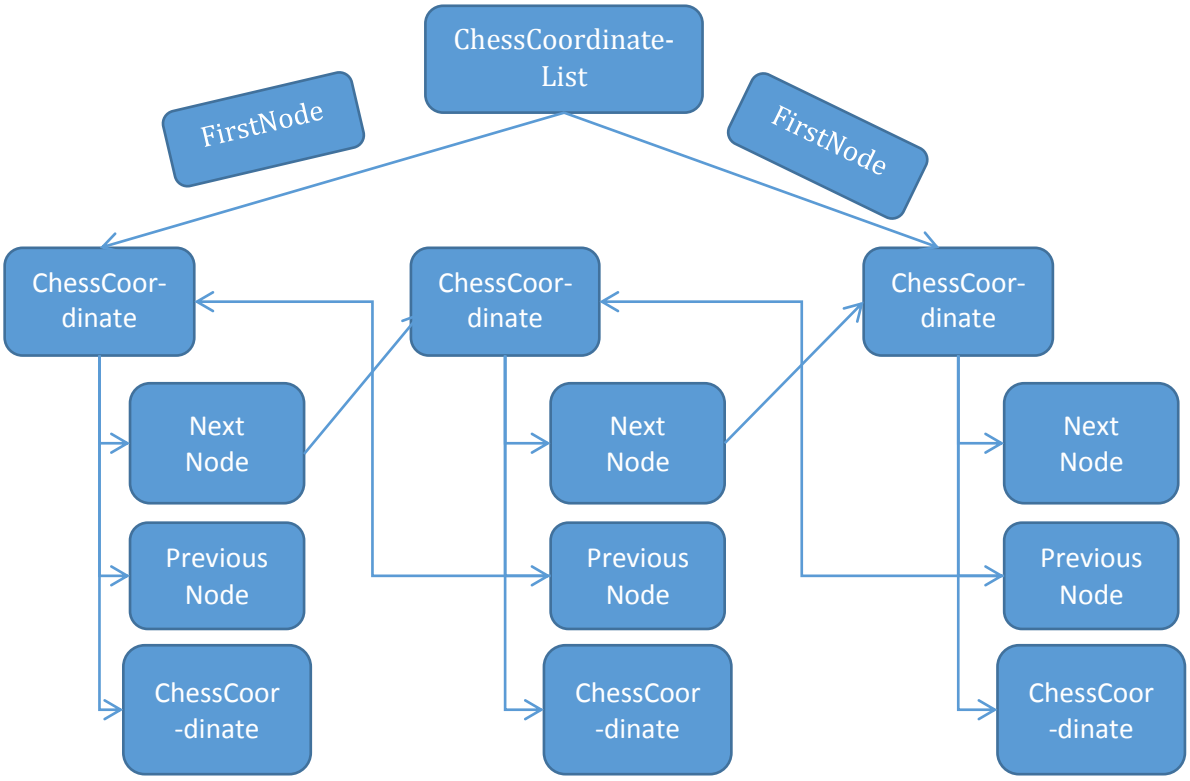
ChessBoard



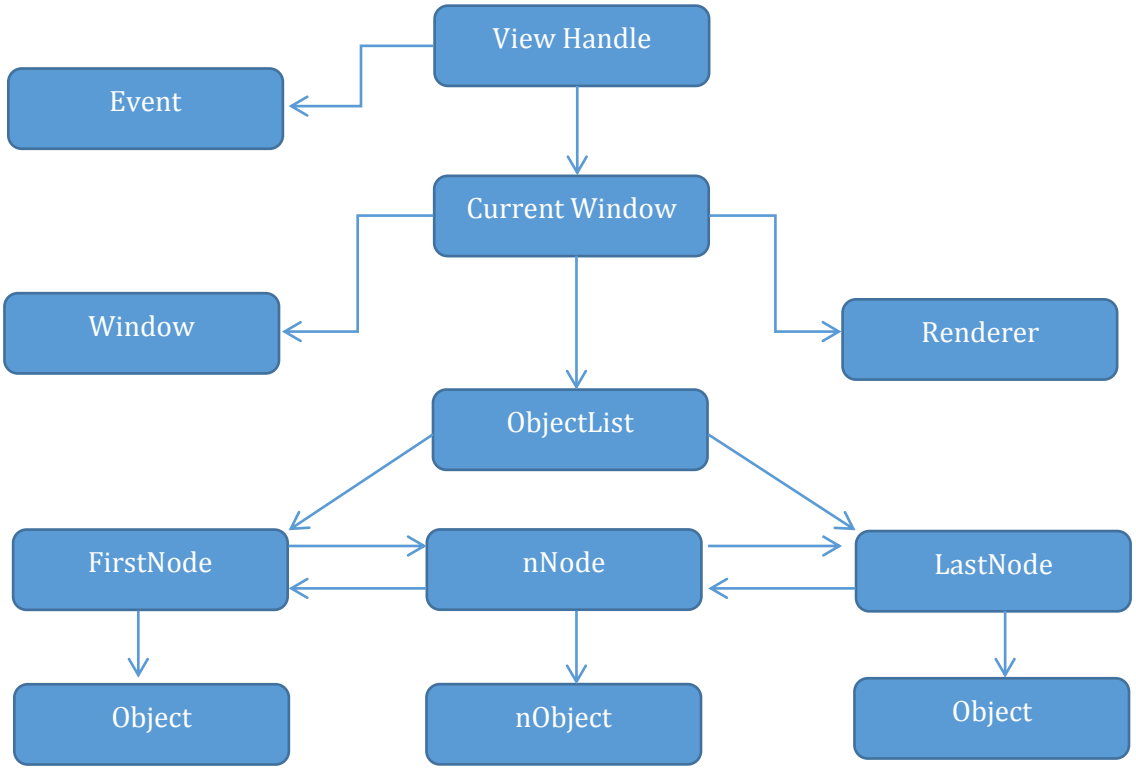
ChessMoveList



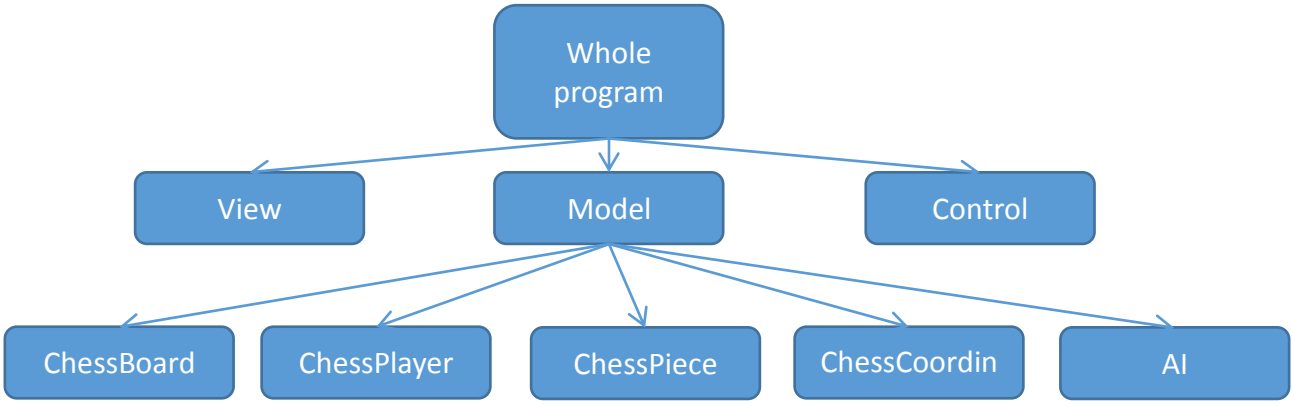
ChessCoordinateList



View



1.2 Major software components



1.3 Module interfaces

Model module:

ChessBoard* Model_Initialize(void);

ChessBoard* Model_PerformMove(ChessBoard*, ChessMoveList*, ChessMove*);

ChessBoard* Model_UndoLastMove(ChessBoard*, ChessMoveList*);

int Model_CheckLegalMove(ChessBoard*, ChessMove*);

ChessCoordinateList * Model_GetLegalCoordinates(ChessBoard*, ChessPiece*, ChessPlayer*, ChessMoveList*);

ChessCoordinateList * Model_GetAllLegalCoordinate(ChessBoard*, ChessPlayer *, ChessPlayer *, ChessMoveList*);

ChessBoard* Model_duplicateChessBoard(ChessBoard*, ChessBoard*);

ChessMove* Model_GetBestMove(ChessBoard*, ChessPlayer*);

int Model_CheckStalemate(ChessBoard*, ChessPlayer*, ChessMoveList*);

int Model_CheckCheckmate(ChessBoard*, ChessPlayer*, ChessMoveList*);

int Model_CheckCheckedPosition(ChessBoard*, ChessPlayer*, ChessMoveList*);

ChessBoard* Model_CleanUp(ChessBoard*, ChessPlayer*);

int writeToLogFile(char fname[100], ChessMoveList *);ChessMoveTypeEnum

Model_GetMoveType(ChessBoard * board, ChessMove *move);

View module:

PlayerControlEnum AskPlayerControl(ChessPlayer *);

AIDifficultyLevel AskAIDifficultyLevel(void);

void DisplayChessBoard(ChessBoard *);

void HighlightCoordinates(ChessBoard *, ChessCoordinateList *);

Event * View_GetEvent(void);

Boolean AskSaveLog(char *);

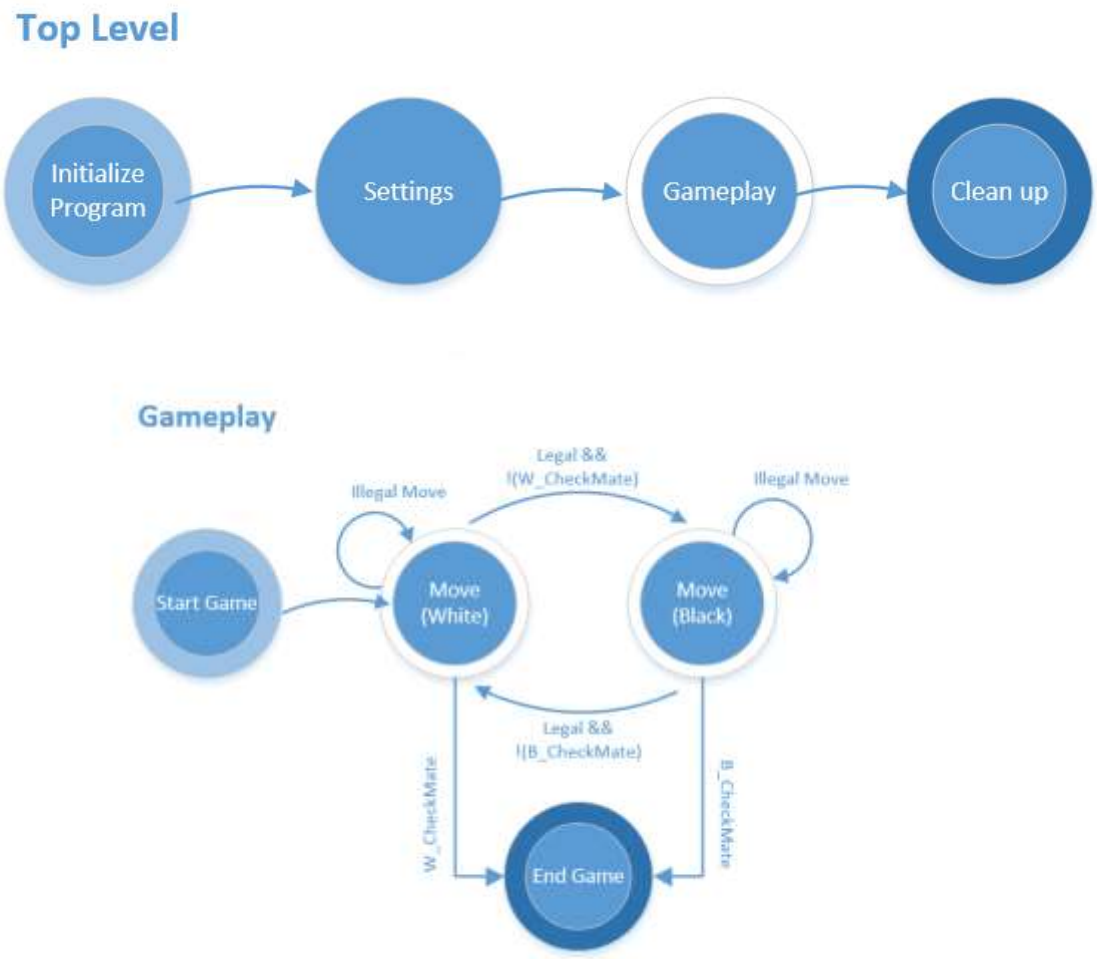
Control module:

void Control_Initialize(void);

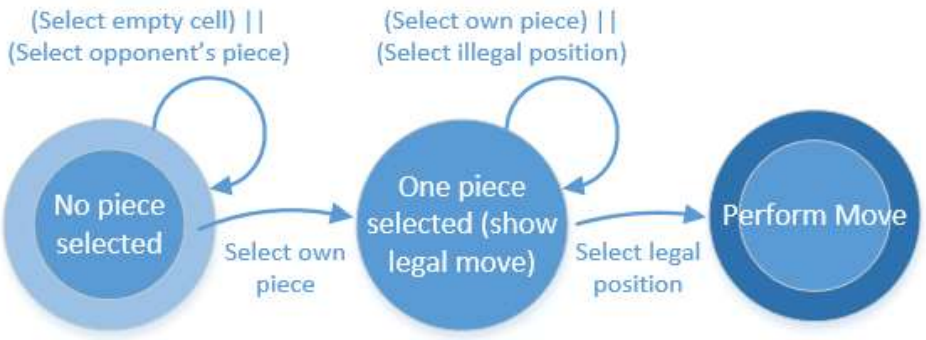
void Control_MainLoop(void);

void Control_CleanUp(void);

1.4 Overall program control flow



Move


(Select empty cell) ||
(Select opponent's piece)
(Select own piece) ||
(Select illegal position)
No piece selected
Select own piece
One piece selected (show legal move)
Select legal position
Perform Move

2: Installation

2.1 System Requirement

- Hardware: PC Hardware (x86_64 server)
- Operating system: Linux OS (RHEL-6-x86_64)
- Dependent third party software:
  i. gcc
  ii. GNU make
- Dependent libraries:
  i. SDL for graphical user interface: SDL2, SDL2_img, SDL2_gfx, SDL2_ttf, SDL2_mixer
  ii. Math library

2.2 Setup and Configuration

- SDL library installation: details are shown at http://libsdl.org

2.3 Building, compilation, installation

- The software comes in a tar.gz package. After downloading, extract the package by running:
  tar –zxvf ChessXIV.tar.gz
- Change into the directory by running:
  cd ChessXIV
- Compile the code by running:
  make
- Run the program by running:
  ./ChessXIV

3: Documentation of packages, modules, interfaces

3.1 Detailed description of data structures

Structures related to Chess game play:

```
typedef enum {Pawn, Rook, Knight, Bishop, Queen, King, None}  ChessPieceTypeEnum;

typedef enum {Human, AI}        PlayerControlEnum;
typedef enum {White, Black}        PlayerColorEnum;
typedef enum {Normal, EnPassant, Castling, Transformation}  ChessMoveTypeEnum;

typedef enum {False, True}       Boolean;

typedef struct {
  ChessCoordinate * Board[CHESS_BOARD_MAX_ROW][CHESS_BOARD_MAX_COL];
  ChessPlayer * WhitePlayer, * BlackPlayer;
} ChessBoard;
```

Description: ChessBoard hold all player, piece and coordinates

```
struct ChessCoordinateStruct {
  unsigned char Rank, File;
  ChessPiece * Piece;
  ChessBoard * MainBoard;
};
```

Description: ChessCoordinate hold the rank, file and the piece occupying that coordinate

```
struct ChessCoordinateListStruct {
  ChessCoordinateNode * FirstNode, * LastNode;
};
```

Description: Holds a linked list of ChessCoordinateNodes to generate possible move list

```
struct ChessCoordinateNodeStruct {
  ChessCoordinateNode * NextNode, * PrevNode;
  ChessCoordinate * Coordinate;
  ChessCoordinateList * List;
};
```

Description: Nodes for ChessCoordinateListStruct that point to coordinates to generate possible move list

```
struct ChessPlayerStruct{
    PlayerColorEnum        PlayerColor;
    AIDifficultyLevel      AIDifficulty;
    PlayerControlEnum      PlayerControl;
    time_t StartTime;
    double ElapsedTime;
    ChessPlayer * OtherPlayer;

    /*list all the pieces that could belong to a player*/
    ChessPiece * Pieces[16];
};
```

Description: ChessPlayer holds player color, player control (AI or human), start time of last move, total elapsed time so far, an array of all its pieces, and a pointer to the opponent

```
struct ChessPieceStruct{
    ChessPieceTypeEnum       Type;
    unsigned char            Index;
    ChessPlayer *            Player;
    ChessCoordinate *        Coordinate;
    Boolean                  AliveFlag;
    int                      MoveFirstFlag;
```

};

Description: ChessPiece holds the coordinate it stays, the index to distinguished with other pieces of same type, the player it belongs to, a counter to keep track of how many times it has moved (mostly used to check if opening move of piece) and alive flag to let people know it's alive

```
struct ChessMoveStruct{
    ChessPiece *          MovePiece;
    ChessCoordinate * StartPosition;
    ChessCoordinate * NextPosition;
    ChessPiece *          CapturePiece;
    Boolean CaptureFlag;
    Boolean check;
    ChessMoveTypeEnum       MoveType;
    ChessPieceTypeEnum Transform_IntoType;
};
```

Description: ChessMove holds the piece that moves, the start and end coordinates, if a piece is being captured, and what type of move it is (normal move versus special move such as en passant)

```
struct ChessMoveNodeStruct{
    ChessMoveList * PrevMove;
    ChessMoveList * NextMove;
    ChessMove * Move;
    ChessMoveList * List;
};
```

Desciption: Nodes for ChessMoveListStruct

```
Struct ChessMoveListStruct {
    ChessMoveNode * FirstNode, * LastNode;
};
```

Desciption: ChessMoveListStruct is the double linked list of Chess Move, used to display move history or undo last move

Data structures to communicate between View and Control:

typedef enum {SelectCoordinate, UndoMove} EventTypeEnum;

typedef struct {

    EventTypeEnum EventType;

    ChessCoordinate * Coordinate;

    ChessPlayer * Player;

} Event;

Description: This structures allows information passing between View and Control

3.2 Detailed description of functions and parameters

Model Module:

ChessBoard* Model_Initialize(void);

Description: Initializes the model by creating a ChessBoard.

ChessBoard* Model_PerformMove(ChessBoard*, ChessMoveList*, ChessMove*);

Description: Takes in the current board and a move and returns the board after the move is performed. This function increments the move counter by piece, appends to the ChessMoveList, and takes care of captures by updating the necessary fields of pieces involved.

ChessBoard* Model_UndoLastMove(ChessBoard*, ChessMoveList*);

Description: Gives the user the option to undo the previous move. Is able to restore all values to previous state (such as the alive flag of captured pieces and previous state if a transformation occurs).

int Model_CheckLegalMove(ChessBoard*, ChessMove*);

Description: Boolean function to check if the move entered is valid based on the current board and the piece at the position given.

ChessCoordinateList * Model_GetLegalCoordinates(ChessBoard*, ChessPiece*, ChessPlayer*, ChessMoveList*);

Description: Returns a list of possible coordinates for a particular piece specified by the function parameters. Also inputs the player in turn to properly return the possible spaces of the king (to avoid suicides for the player in turn).

ChessCoordinateList * Model_GetAllLegalCoordinate(ChessBoard*, ChessPlayer *, ChessPlayer *, ChessMoveList*);

Description: Calls Model_GetLegalCoordinates to form a list of all possible spaces in any particular turn.

ChessBoard* Model_duplicateChessBoard(ChessBoard*, ChessBoard*);

Description: Duplicates the chess board to simulate a move.

ChessMove* Model_GetBestMove(ChessBoard*, ChessPlayer*);

Description: Gives the 'best move' as determined by the program. Can be used to generate the next move for the computer.

int Model_CheckStalemate(ChessBoard*, ChessPlayer*, ChessMoveList*);

Description: Boolean function to check if the board is in stalemate based on the player in turn.

int Model_CheckCheckmate(ChessBoard*, ChessPlayer*, ChessMoveList*);

Description: Boolean function to check if the board is in checkmate for the player in turn.

int Model_CheckCheckedPosition(ChessBoard*, ChessPlayer*, ChessMoveList*);

Description: Boolean function to check if the current player in turn is in check.

ChessBoard* Model_CleanUp(ChessBoard*, ChessPlayer*);

Description: Cleans the board

int writeToLogFile(char fname[100], ChessMoveList *);

Description: Save the MoveList to a log file

ChessMoveTypeEnum Model_GetMoveType(ChessBoard * board, ChessMove *move);

Description: Returns the move type.

View Module:

View(View.h):

ChessPlayer * CurrentPlayer;

} ViewHandle;

ViewHandle * View_Initialize(void);

Event * SetOptions(ViewHandle *, ChessBoard *);

void DisplayChessBoard(ViewHandle * MainViewHandle, ChessBoard * MainBoard);

void HighlightCoordinates(ViewHandle * MainViewHandle, ChessBoard * MainBoard, ChessCoordinateList * CoordList);

Event * View_GetEvent(ViewHandle * MainViewHandle, ChessBoard * CurrBoard, Event *);

void View_DisplayEvent(ViewHandle * MainViewHandle, ChessBoard * CurrBoard, Event *);

void View_ConcludeGame(ViewHandle * MainViewHandle, ChessPlayer *);

Display (display.c, display.h)

void drawMainMenu(SDL_Renderer *renderer);

Description: renders the main menu graphics assets; this includes background image and all menu text.

void drawOnePlayerMenu(SDL_Renderer *renderer);

Description: renders the graphics assets for the one-player options menu; this includes background image and all text.

void drawTwoPlayerMenu(SDL_Renderer *renderer);

Description: renders the graphics assets for the two-player options menu; this includes background image and all text.

void drawAdvancedMenu(SDL_Renderer *renderer);

Description: renders the graphics assets for the advanced options menu; this includes background image and all text.

void drawGameplayScreen(SDL_Renderer *renderer, int mode, int time);

Description: renders the graphics assets for the gameplay screen; this includes sub-menus, counters, and all text.

void drawChessboard(SDL_Renderer *renderer);

Description: uses SDL primitive rendering to draw and color the chessboard.

void drawPieces(SDL_Renderer *renderer);

Description: renders the chess piece images to a texture, and then renders them to the chessboard at starting position.

void drawError_p1_Options(SDL_Renderer *renderer);

Description: renders a background color and an error message in a floating window. This is invoked when the user selects play in a one-player option menu without selecting all relevant options.

void drawError_p2_Options(SDL_Renderer *renderer);

Description: renders a background color and an error message in a floating window. This is invoked when the user selects play in a two-player option menu without selecting all relevant options.

void drawError_kbd_Input(SDL_Renderer *renderer);

Description: renders a background color and an error message in a floating window. This is invoked when the user attempts non-standard notation or incomplete keyboard inputs.

void drawError_mouse_Input(SDL_Renderer *renderer);

Description: renders a background color and an error message in a floating window. This is invoked when the user attempts to move piece off the board with the mouse.

void drawError_IllegalMove(SDL_Renderer *renderer);

Description: renders a background color and an error message in a floating window. This is invoked when the user commits an illegal move.

void drawWarning_BlackInCheck(SDL_Renderer *renderer);

Description: renders a background color and an error message in a floating window. This is invoked when the black king is in check.

void drawWarning_WhiteInCheck(SDL_Renderer *renderer);

Description: renders a background color and an error message in a floating window. This is invoked when the white king is in check.

void drawMessage_time_BlackWins(SDL_Renderer *renderer);

Description: renders a background color and an error message in a floating window. This is invoked when the white timer runs out and black wins.

void drawMessage_time_WhiteWins(SDL_Renderer *renderer);

Description: renders a background color and an error message in a floating window. This is invoked when the black timer runs out and white wins.

void drawMessage_mate_BlackWins(SDL_Renderer *renderer);

Description: renders a background color and an error message in a floating window. This is invoked when white is checkmated.

void drawMessage_mate_WhiteWins(SDL_Renderer *renderer);

Description: renders a background color and an error message in a floating window. This is invoked when black is checkmated.

Control module:

void Control_Initialize(void);

Description: This function initializes Model and View and get things started.

void Control_MainLoop(void);

Description: Run the main program

Void Control_CleanUp(void);

Description: Close windows and free all used memory then quit

3.3 Detailed description of input and output formats

Syntax/format of a move input by the user:

Move input by user is recorded by the computer as mouse click on the GUI. A click on the board will be translated into a coordinate and the program will perform computational logic on it.

Syntax/format of a move recorded in the log file:

The log file will keep track of the moves in accordance to the algebraic notation. Example of what it would look like is on the wikia page:
http://en.wikipedia.org/wiki/Algebraic_notation_%28chess%29

3.4 Detailed description of artificial intelligence for computer player

The artificial intelligence machine will consist of three different difficulties: beginner, intermediate, advanced.

The beginner difficulty's aim is to familiarize the player with the basic operations of chess and nothing more. The beginner artificial intelligence setting will consist of selecting a move from LegalChessMoves[] using a random number generator. The random number generator will be using the time of day as a seed, so that the moves do not become repetitive.

The intermediate difficulty's aim is to test the player's ability to handle pressure. It will make its selection from LegalChessMoves[] by prioritizing moves that perform a piece capture.

The advanced difficulty's aim is to test the player's ability to make smart trades. It will first attempt to perform a scholar's mate, and then it will proceed like the intermediate ai except it will prioritize capturing more valuable pieces.

4: Development plan and timeline

4.1 Partitioning of tasks

The whole program will be divided into four main areas:

- Gameplay: All functionality of a chess program such as move, undo, checkmate, check

- AI: The intelligence behind the computer-generated moves

- GUI: The display of program for user

- Control and Integration: Program flow and integration between modules

4.2 Team member responsibilities

As discussed above, four areas will be responsible by the following team members:

- Gameplay: Hanchel Cheng, Kevin Duong and Jamie Lee
- AI: Andrew Trinh
- GUI: Ryan Morrison
- Control and Integration: Quan Chau

4.3 Timeline

By January 27:

- Finish Command line output
- Finish basic moves, excluding castling, transforming and en passant
- King can not commit suicide
- Support Undo

By February 3:

- Finish GUI output
- Finish castling, transforming and en passant
- Finish AI
- Support Move history record

Back Matter

Copyright

Index

AI, 1, 15
artificial intelligence, 14
Control, 1
Control and Integration, 15
Control module, 4, 12
Dependent libraries, 6
Dependent third party software, 6
Gameplay, 15
GUI, 1, 15
Hardware, 6
Model, 1
Model module, 4
Model Module, 9
move input, 13
move recorded, 13
Operating system, 6
View, 1
View module, 4
View Module, 10