AlphaGun Chess Software Specification

V2.0

Designed by AlphaGun

Chaoran Nong
Yijian Wang
Yanjie Xu
Yifan Xu
Xiaoyan Yang
Yan Zhang

Table of contents

Glossary

AI: Artificial Intelligence

GUI: Graphical User Interface

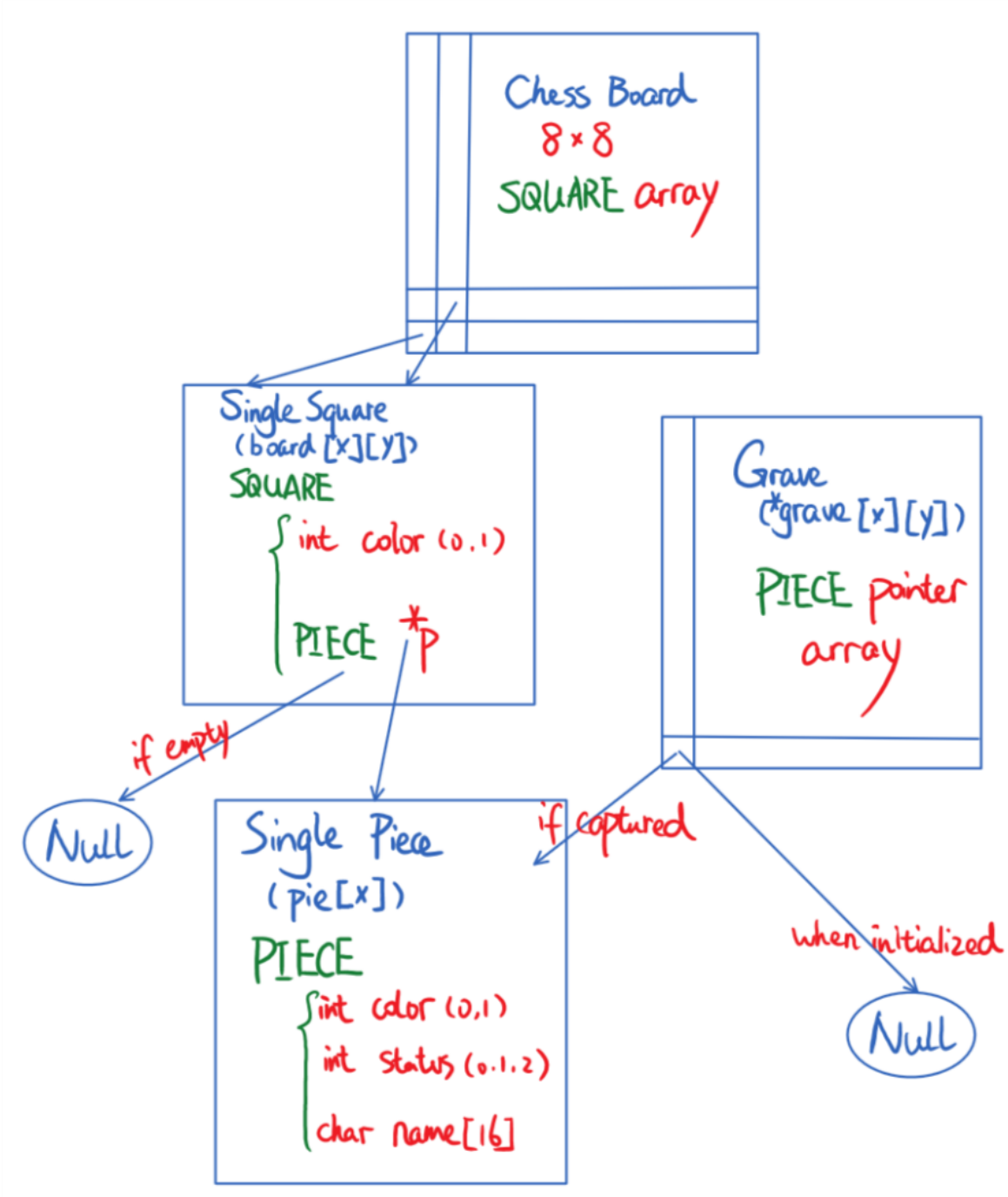
API: Application Program Interface

Module: One of several parts of a piece of computer software that does a particular job

Control Flow: The order in which the instructions or function calls of a programme are executed

1 Software Architecture Overview

1.1 Main data types and structures



1.2 Major software components

• Diagram of module hierarchy

<table>
<tr><td colspan="3">Chess (Main Module)</td></tr>
<tr><td>Strategy(AI.h)</td><td>User Interface(GUI.c)</td><td rowspan="3">X11 Graphics Library(libSDL)</td></tr>
<tr><td colspan="2">Rules&Move (rules.h,Piece.h)</td></tr>
<tr><td colspan="2">Environment(Board.h, Piece.h)</td></tr>
<tr><td>Math(libm)</td><td>Std.C(libc)</td><td>Stdio</td></tr>
<tr><td colspan="3">Linux OS (RHEL-6-x86_64)</td></tr>
<tr><td colspan="3">PC Hardware(x86_64 server)</td></tr>
</table>

1.3 Module interfaces

• API of major module functions

UserIO

```
int Saveboard(char outFileName[32], SQUARE board[][8], PIECE *grave[2][16],
LIST *lf);
int Loadboard(char inFileName[32], SQUARE board[][8], PIECE *grave[2][16]);
int NewLogFile(DIRECTION tar, DIRECTION des, int condition, LIST *lf);
int Undo(LIST *lf, SQUARE board[][8], PIECE *grave[][16]);
int DelLogFile(LIST *lf);
DIRECTION getPosition(char userinput[32]);
void Reset(SQUARE board[][8], PIECE *grave[][16], LIST lf);
```

Piece

```
DIRECTION Find_Piece(SQUARE board[][8], int color, int number);
PIECE *Create_Piece(char name, int color, int status, int value, int number);
void Delete_Piece(PIECE *p);
void SetPieces(SQUARE board[][8]);
int PieceLegalMove(SQUARE board[][8], int a, int b, int x, int y);
```

Rules

int Move(SQUARE board[][8], int tar_x, int tar_y,int des_x,int des_y,PIECE *grave[][16]);

int Check(SQUARE board[][8],int color, int x,int y);

int LegalMove(SQUARE board[][8], int tar_x,int tar_y, int des_x, int des_y);

1.4 Overall program control flow

- setup
- display the board
- repeat
  – white player makes a move
  – display the board
  – if black is in checkmate, white wins!
  – black player makes a move
  – display the board
  – if white is in checkmate, black wins! Let the games begin!

2 Installation

2.1 System requirements, compatibility

Operating System:

Linux OS

Processer:

Intel Pentium (@ 1.5 GHz)
AMD ATHLON (@ 1.6GHz)
Or later versions

Graphic Requirements:

Nvidia GT 9800
AMD Radeon HD 2000
Intel HD graphics 500
Or later products

Disk Space:

At least 100M HDD space

RAM:

At least 10M

2.2 Setup and configuration

Login the Linux platform
Extract the file from AlphaGun.tar
command: gtar xvzf AlphaGun.tar.gz

2.3 Building, compilation, installation

Go to the main folder and run makefile to build the program
command: make
After the compilation finished run the program
command: ./bin/AlphaGun
Start the game and follow the instruction:
choosing player vs.player
Or player vs. computer
Then enjoy the chess game

3 Documentation of packages, modules, interfaces

3.1 Description of data structures and critical snippets of source code

Type.h

```
typedef struct {
        int color;          /* piece color, 0=black, 1=white  */
        int status;         /* 0=initial position, 1=moved,  2=captured */
        char name;          /* piece name */
        int validsteps;     /* AI reference value */
        int value;          /* Value accquired when captured */
        int number;         /* Pieces Index */
} PIECE;

typedef struct{
```

```
        int color;          /* 0=black, 1=white */
        PIECE *p;           /* pointer to single piece */
} SQUARE;


typedef struct {
        int x;                      /* x coordinate */
        int y;                      /* y coordinate */
} DIRECTION;

/* For Logfile, Double Linked List */
typedef struct LogList LIST;
typedef struct LogEntry ENTRY;
struct LogList {
        int number;
        ENTRY *First;
        ENTRY *Last;
};
struct LogEntry{
        int index;
        ENTRY *Prev;
        ENTRY *Next;
        DIRECTION Tar;
        DIRECTION Des;
        int condition;
} ;

LIST *lf;                   /* Pointer to Logfile List */
SQUARE board[8][8];         /* 8x8 array to represent the board */
PIECE pie[32];              /* 32 pieces */
PIECE *grave[2][16];        /* A set of pointers to captured pieces */
```

3.2 Detailed description of functions and parameters

ChessGame.c

int main()

Print Menu & Second Menu, get User's input and call functions accordingly

- User chooses game mode:

1.Player vs. Player => call function Games2Player()
2.Player vs. Computer => call AI
3.Load Previous Game(beta) => call LoadGame()
4.Wrong input => call LOOP

Board.h

```
/* Reset all the piece and initialize the board */
int Reset_Board();
/* Print the board with new piece positions after any movement or operation */
int Print_Board();
```

Piece.h

```
/* Create a piece corresponding to the position, return PIECE pointer */
PIECE *Create_Piece(int x, int y);
/* Promote a pawn to an advance piece when he arrives the base line */
int Promotion(PIECE *p);
/* Print all of the available next movement for King */
int Legalmove_King(PIECE *p, SQUARE board[8][8]);
/* Print all of the available next movement for Queen */
int Legalmove_Queen(PIECE *p, SQUARE board[8][8]);
/* Print all of the available next movement for Bishop */
int Legalmove_Bishop(PIECE *p, SQUARE board[8][8]);
/* Print all of the available next movement for Knight */
int Legalmove_Knight(PIECE *p, SQUARE board[8][8]);
/* Print all of the available next movement for Rock */
int Legalmove_Rock(PIECE *p, SQUARE board[8][8]);
/* Print all of the available next movement for Pawn */
int Legalmove_Pawn(PIECE *p, SQUARE board[8][8]);
```

Rules.h

```
/* Return the pointer of the captured piece */
PIECE *Capture(PIECE *p, int x, int y);

/* Move piece under user's order but follow the rule */
int Move(int tar_x, int tar_y, int des_x, int des_y);
    /* Input target (x,y) and destination (x,y), call function legalmove; return int
    0,1 or 2; 0=cannot move; 1=move to a blank square; 2=move to a square
    with an opponent piece, call function capture */
```

```
/* Victory Conditions Check */
int Check(int tar_x, tar_y);
        /*  Give the position of the King, return 1 if check happen, return 2 if
        checkmate, return 0 if King is safe */
```

3.3 Detailed description of input and output formats

• Syntax/format of a move input by the user

(chess piece) [position 1]-[position 2]

Example:

Ng1-f3

Knight moves from g1 to f3

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

Move:

(chess piece) [position 1]-[position 2]

Capture:

(piece_name)x[destination_square]

End of game:

1-0 indicates White won, 0-1 indicates Black won, and ½–½ indicates a draw

Other notation:

!! (an excellent move)
! (a particularly good—and usually surprising—move)
!? (an interesting move that may not be best)
?! (a dubious move – one which may turn out to be bad)
? (a bad move; a mistake)
?? (a blunder)
+/− (clear white advantage)
+/= (white is slightly better)
= (equality in the position)
=/+ (black is slightly better)
−/+ (clear black advantage)
□ (the only possible move)
TN or N (a theoretical novelty)

Example :

Qg5xg3+

The Black Queen on g5 captured a piece on g3 and the opponent's King was checked

4 Development plan and timeline

4.1 Partitioning of tasks

+Part I

- Program Structure Initialization

Chess-Game.c	Main function
Board.h, Board.c	Welcome Menu, Create board, Set up data structure
UserIO.h, UserIO.c	Function Save & Load, Command Translation

- Building in Core Rules

Piece.h, Piece.c	Initialize pieces on board, Legal Movements Check, Pawn Promotion
Rules,h, Rules,c	Pieces Move, Check & Checkmate, Capture

+Part II

- GUI Design
- “Easy” Mode AI Programming

+Part III

- Test and Debug
- “Hard” and “Crazy” Mode AI Programming

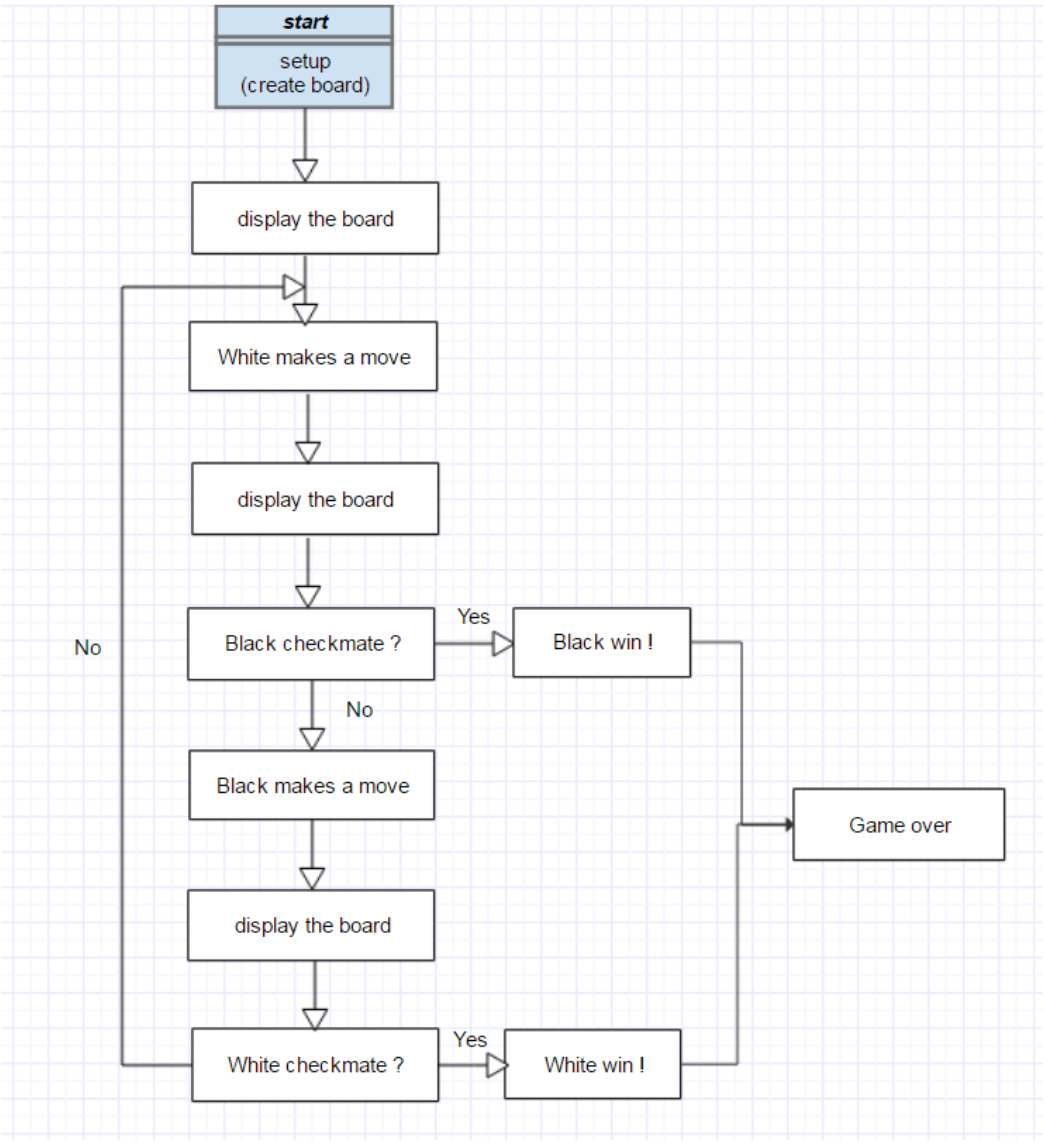
4.2 Team member responsibilities Back matter

Xiaoyan Yang:
AI.c, ChessGame.c, Board.c
Yanjie Xu:h
AI.c, Board.h, GUI.c
Yan Zhang:
Rules.c, ChessGame.c
Yijian Wang:
Piece.h, Piece.c
Yifan Xu:
UserIO.h, UserIO.c, ChessGame.c

Chaoran Nong:
GUI design and implementation

4.3 Overall chess program control flow

The basic overall control flow of a chess game is the following:



5 Appendix

5.1 Copyright

5.2 References

http://www.chesscorner.com/tutorial/basic/notation/notate.htm
https://en.wikipedia.org/wiki/Algebraic_notation_(chess)

5.3 Part of source code

Constant.h

```
#define Black 0
#define White 1

/* Piece value after being captured */
#define Kingvalue 100
#define Queenvalue 10
#define Bishopvalue 3
#define Knightvalue 3
#define Rookvalue 5
#define Pawnvalue 1

/* Piece ID number for SaveLog */
#define Pawn 6
#define Knight 5
#define Bishop 4
#define Rook 3
#define Queen 2
#define King 1
```

UserIO.h

```
int Saveboard(char outFileName[32], SQUARE board[][8], PIECE *grave[2][16],
LIST *lf); /* 0=Success 1=Error */
int Loadboard(char inFileName[32], SQUARE board[][8], PIECE *grave[2][16]);

int NewLogFile(DIRECTION tar, DIRECTION des, int condition, LIST *lf);
int Undo(LIST *lf, SQUARE board[][8], PIECE *grave[][16]);
int DelLogFile(LIST *lf);

DIRECTION getPosition(char userinput[32]);

void Reset(SQUARE board[][8], PIECE *grave[][16], LIST *lf);
```

Board.h

```
int Set_Board(SQUARE Board[][8]);
int Delete_Board(SQUARE Board[][8]);
int Delete_Grave(PIECE *Grave[][16]);
```

Piece.h

```
DIRECTION Find_Piece(SQUARE board[][8], int color, int number);
PIECE *Create_Piece(char name, int color, int status, int value, int number);
void Delete_Piece(PIECE *p);
void SetPieces(SQUARE board[][8]);
int PieceLegalMove(SQUARE board[][8], int a, int b, int x, int y);
```

Rules.h

```
int Move(SQUARE board[][8], int tar_x, int tar_y,int des_x,int des_y,PIECE
*grave[][16]);
int Check(SQUARE board[][8],int color, int x,int y);
int LegalMove(SQUARE board[][8], int tar_x,int tar_y, int des_x, int des_y);
```

Appendix

Table of Properties of a Every Piece

	Pawn	Knight	Bishop	Rook	Queen	King
ID Number[1]	±6	±5	±4	±3	±2	±1
Index Number[2]	8-15	1,6	2,5	0,7	3	4
Initial Value[3]	1	3	3	10	10	100

1:Positive numbers represent white pieces while negative numbers represent black pieces.
2:No sign for different colors.
3:Pawns will earn 1 more credit whenever it moves forward 1 block.