

Froppy v.1.0  
A (Formal) Neuralnet study of approaching chess  
User Manual

Abel Jiminez, Aung Thu, Jason Duong, Jeremiah Blackburn, Khoi Trinh  
Affiliated with University of California, Irvine

February 5, 2018

**Abstract**



Figure 1: Team froppy was based off a frog

This is a User manual intended for individuals who are interested in our program.

# 1 Table of Contents

1. Chess Piece Definition
2. Glossary
3. Usage Scenario
4. Goals
5. Features
6. Installation
7. System Requirements
8. Setup & Configuration
9. Uninstalling
10. Chess Program Functions and Features
11. Error Messages
12. Index

## 2 Chess Piece Definition







Piece	Initials	Visual	Movement Restrictions	Number of Spaces	Capture Method	Special Attributes
Pawn	pw		Toward other end of board	1	Diagonally	Promoting
Rook	rk		Vertically or Horizontally	Any	Vertically or Horizontally	Castling
Bishop	bp		Diagonally	Any	Diagonally	---
Knight	kt		L Shape	3	L Shape	Can move over other pieces on the board
Queen	qn		Vertically, Horizontally, or Diagonally	Any	Vertically, Horizontally, or Diagonally	---
King	kg		Vertically, Horizontally, or Diagonally	1	Vertically, Horizontally, or Diagonally	Castling

Figure 2: Chess Definition

### 3 Glossary

- Attack : when a piece is moved to a position at which it is able to capture an opposing piece on the following move.
- Check : when the King is attacked, but still has at least one of the three following options possible for safety:
  - ⊙ There are Three Moves for Escaping Check:
  - ⊙ Eliminating the opposing attacker
  - ⊙ Putting a friendly piece in the way of the attacker
  - ⊙ Moving the King to a safe square
- Castling: a special move in which the King moves to the side of the Rook and then the Rook takes the place on the other side of the King.
  - ⊙ Both the King and Rook must have not been moved previously
  - ⊙ the spaces between the two pieces must be empty
  - ⊙ the king must not enter or cross over a space that would result in a check
- Checkmate: when the king is attacked and has no viable move for escaping.
- En Passant: a Pawn is captured when it has moved two spaces instead of one. The enemy captures the Pawn in a fashion as if it had only moved one space.
- File: column of a chess board (lettered a-h)8. Not to be confused with office files.
- Forfeit: to give up the match and take a loss.
- Stalemate (Draw): when the game ends with either player achieving checkmate.
- Ply: the official name for one player's turn.
- Promoting: after a Pawn reaches the opposite end of the board successfully, it can be traded for either a Knight, Bishop, Rook, or Queen.

## 4 Usage Scenario

Figure 1 below shows a typical representation of initial chess program gameplay.

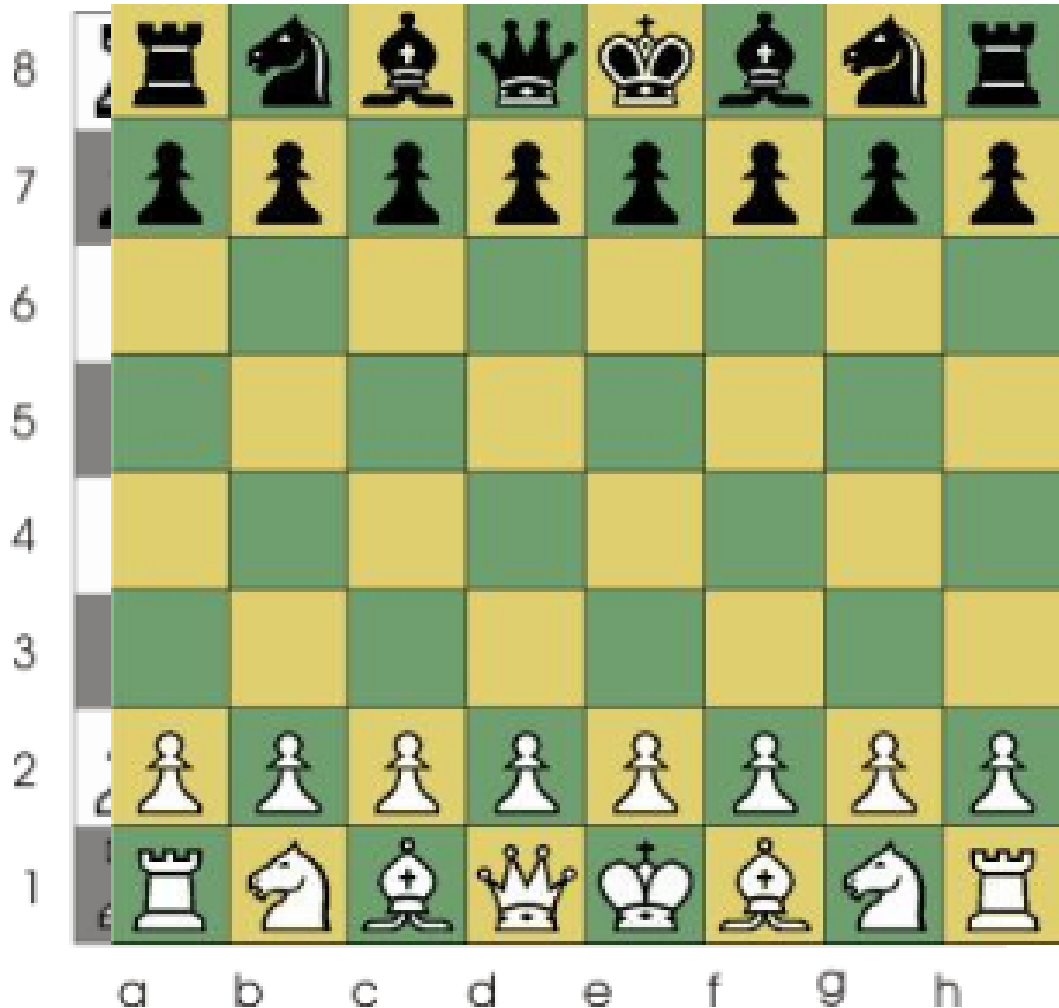


Figure 3: Game Board Sample

Figure 1: chess board with chess pieces in starting positions<sup>2,5</sup>

Each space on a chess board is specified by the rank and file of the board according to Figure 1 above. For example, the white queen and black king in Figure 1 above is in the d-file, first rank and e-file, eighth rank, or spaces d1 and e8, respectively<sup>8</sup>. The game begins with the user pressing Play on the menu shown in Figure 2 below, which will launch a chess board and a text interface for interacting with the chess board and the computer.

Each ply is specified by the initials of the piece, the current position of the piece, and the end position of the piece, separated by a single space. For example, according to Figure 1, either "kn b1 a3" or "kn b1 c3" as well as "kn g1 f3" or "kn g1 h3" would be a valid ply. Figures 2a and 2b illustrate an example of gameplay.

The game ends when either of the following occur (see Glossary for explanations): Checkmate  
Stalemate Insufficient Material

Three-fold repetition 50-move rule Forfeit

## 5 Goals

3 Goals Create an AI Chess program for humans to play against. All aspects of gameplay will be played according to official rules of chess. AI's strategy engine will be neural network-based<sup>1,4,6,7</sup>. Provide a graphical user interface (GUI) for easier user interaction.

## 6 Features

This chess program will:

- Provide a user-interactive menu to start the game
- Display the game board and allow the user to make moves
- Enable a human user to play against this chess program
- Allow the user to choose which side (Black or White) to play
- Log all the moves throughout a game, which can be accessible to the user
- Limit each AI move to a minute or less

## 7 Installation

### 7.1 Hardware Requirements

- Any processor affected by spectre or meltdown (Intel and Amd processors only)
- recommended processor: Intel(R) Xeon(R) CPU E5-2660 0 @ 2.20GHz
- 1gb Storage
- 30gb Ram

### 7.2 Software Requirements

- Linux operating system Linux operating system e.g. RHEL-6-x86\_64
- gcc, version 4.4.7 or later
- Make utility

### 7.3 How to Install

5.2 Setup and Configuration: Download the source code file Open a command prompt Run

```
make chess
```

Run

```
./chess
```

Start playing!

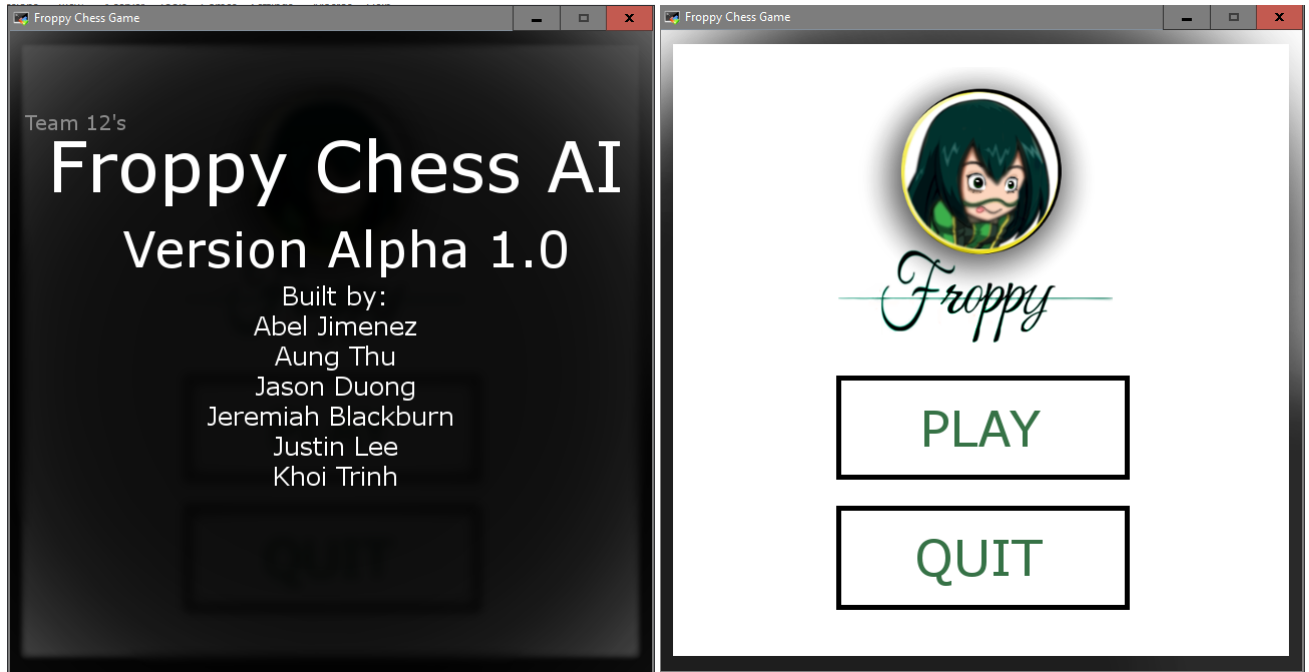
Keep the command prompt open to interact with the chess program

Note: to uninstall, run Run

```
make clean
```

## 8 Chess Program Functions and Features

- Menu User can choose to begin the game by pressing Play, or exit the game with “Quit”, illustrated in Figure 2 below



- Gameplay: User inputs a move by clicking twice: the piece and the new piece position \*Note: Illegal moves will result in error messages and prompt the user for a redo.



If the game ends in a checkmate, a congratulatory message pops up and the game ends. If it is a draw or if either player forfeits, there is a notification popup and the game ends.

## References

- [1] Baxter, Jonathan, et al. *KnightCap: A Chess Program That Learns by Combining TDλ with Game-Tree Search*. Australian National University, 10 Jan. 1999.

## 9 Error Messages

- You have entered an illegal move. Please enter another one Solution: Choose a valid move when trying again
- “Warning: You have been idle for too long.”  
Solution: make a valid move
- “You cannot move on the opponent’s turn. Wait for opponent to complete their turn before inputting your move.”  
Solution: Just wait for the opponent to finish making their chess move
- “Cannot complete Castle. Either Rook or King has previously been moved.” Solution: Because the right to castle has been lost, the player must make a valid move.
- “Moving King to that position would result in a Check!”  
solution: Because the player is not allowed to move their king in check, they must make choose another valid move.

## 10 Detailed Description of Functions & Parameters

This section provides the chess program’s function prototypes along with brief explanations.

ChessGUI.h

```
void DisplayWindow(char $$file, SDL_Surface $$screen, int delay);
```

Inputs: filename, screen, amount of delay

Outputs: N/A

Description: Takes a .bmp file from a directory and copies it to a screen for some length of time.

```
void Add_Box(SDL_Surface *image, SDL_Surface *screen, int x, int y);
```

Inputs: 2 existing screens, x-y coordinate

Output: N/A

Description: Applies one screen onto another using the inputted coordinate as the reference point for the top left corner of the applied screen.

```
void Add_BoxFile(char *file, SDL_Surface *screen, int x, int y);
```

Inputs: filename, screen, x-y coordinates

Output: N/A

Description: In the case that the desired image is not already saved as an SDL\_Surface, this function takes the .bmp file and applies it to the screen at the given coordinate. This is useful for when the the image is only used temporary or once and does not need to be saved as a surface.

```
void UpdateWindow(SDL_Surface *screen, int delay);
```

Inputs: screen, delay

Output: N/A



Description: Updates and displays the current screen and increases the amount of time it stays up. Used in while loops to continuously check for any changes in the screen.

```
void Exit(SDL_Surface *screen);
```

Inputs: screen

Output: N/A

Description: Frees the current working screen and follows SDL's exit protocol `SDL_Quit()`.

`basic_eval.h` (basic evaluation function for testing purpose)

```
int basicEvaluation(BSTATE* currentboard);
```

Inputs: the state of the current board.

Outputs: the evaluation score of the current board.

Description: This function evaluates the current state of the board and give a rating of whether this would lead to a win or lose. This function is used for testing non-neural network features and will be replaced with neural network in the final release.

`fenToBoardState.h` (Converts a string formatted in fen and parses the information to the board state, and is reversible)

```
void fenToBoardState(char* fen, BSTATE *b);
```

Inputs: the FEN string as well as the BSTATE structure pointer that recieves the FEN information  
Output: void.

Description: This function parses a FEN string and sends the information to a structure that holds the current state of the chess board.

```
void boardToFen(char *fen, BSTATE *b);
```

Inputs: The BSTATE pointer that represents the current board state, and the character string fen:

Output: void

Description: This function gets a BSTATE structure holding all the information on a given game, and parses it back into FEN, sending it into the character pointer fen;

`BoardState.h` (contains information on the board status)

The board is represented as a 8 by 8 array and each piece is encoded as an integer. The first digit represents the type of piece. It is represented as such: 1 is a pawn, 2 is a knight, 3 is a bishop, 4 is a rook, 5 is a queen, and 6 is the king. The next digit represents the type of piece 0 being white and 1 being black. i.e 11 is a black pawn and 1 is a white one.

```
void boardToVector(BSTATE *b, int *vector);
```

Inputs: BSTATE pointer as well as a int pointer named vector

Output: void

Description: This function will then get the BSTATE structure and parse it into a 1 dimensional long array to send it to the neural network. Converting the structure to the array makes it easier for the neural network to process information

`matrix.h` (provide matrix arithmetic functions)

```
void printMatrix(float *matrix, int width, int length);
```

Inputs: The matrix, and its width and length.

Outputs: The matrix shows up in the terminal.

Description: This function is used for debugging and will print out the input matrix. This function is global so that both CUDA code and CPU code can execute it.

```
void matrixMultiplication(float *h_a, float *h_b, float *h_result, int m, int n, int k);
```

Inputs: The two matrices that need to be multiplied, the matrix that the result will be saved as well as their dimensions.

Outputs: The results are saved into the supplied matrix.

Description: This function is used by CUDA GPU to multiply matrix together, there are usually multiple instances of this function running in parallel when the neural network is computing.

```
void transposeMatrix(float *inputmatrix, float *outputMatrix, const unsigned int row, const unsigned int col);
```

Inputs: The matrix to be transposed and the matrix that results will be saved to as well as the dimensions.

Outputs: The result will be saved to the supplied output matrix.

Description: This function is used for CUDA GPU to transpose matrix and multiple instances of the function will run in parallel with each other to allow quick computing.

```
void combine3Matrix(float *a, float*b,float*c, float*combine, int rowa, int rowb, int rowc);
```

Inputs: This will concatenate three one dimension matrix together and needs the pointer that the new matrix will be pointed at, it also requires the dimension of the input matrix of the concatenation.

Outputs: The result will be saved to the supplied output matrix.

Description: This is used by CUDA GPU to join many matrices together. Many threads can execute multiple instances of this function in parallel.

backprop.h (provide back propagation functions)

```
void dererror_output_outer(float *nn_output, float *desired_output, *float der_result, int length);
```

Inputs: The array that contains the actual output the neural network, an array that contains the desired output of the neural network, an array to save the derivative of the error with respect to the output of the outermost layer as well as the length of the arrays.

Outputs: The result will be saved to the supplied output array.

Description: This function used by CUDA GPU calculates the derivative of the error with respect to the output of the outermost layer, then calculates the derivative of the error with respect to the value of the node. Many threads can execute multiple instances of this function in parallel.

```
void dererror_output(float *weight_next_layer, float *dererror_val, *float der_result, int length);
```

Inputs: The array that contains the weight of the next layer, an array that contains the derivative of the error with respect to the value of the next node, an array to save the derivative of the error with respect to the output of the layer as well as the length of the arrays.

Outputs: The result will be saved to the supplied output array.

Description: This function is used by CUDA GPU to calculate the derivative of the error with respect to the output of inner layers as a stepping stone for calculating the derivative of the error with respect to the value of the node. Many threads can execute multiple instances of this function in parallel.

```
void dererror_val(float *dererror_output, float *derror_val_res, int length);
```

Inputs: The derivative of the error with respect to the output of the node as well as an array to save the the derivative of the error with respect to the value of the node, the length of the array is also

needed.

Outputs: The derivative of the error with respect to the value of the node is saved into the supplied array.

Description: This function is used by the CUDA GPU to compute the derivative of the error with respect to the value, which is required to calculate the gradient descent. Many threads can execute multiple instances of this function in parallel.

```
void grad_des_comp(float *output_prev_layer, float *dererror_val, float learning_rate, float *grad
```

Inputs: the output of the previous layer, the derivative of the error with respect to value in each node as well as the learning rate of the neural network, an array that the gradient descent will be saved to is also needed, the length of the supplied result array is required.

Outputs: the calculated gradient descent will be saved in the supplied array.

Description: this function is executed by CUDA GPU to compute the gradient descent of weights in order to minimize the loss function. Many threads can execute multiple instances of this function in parallel.

minmax.h (finds the best worst move for AI to make)

```
NODE *createNode(float value, MENTRY *move);
```

Inputs: a legal move and the value of evaluated board after the move

Outputs: pointer to the created node

Description: The value and move is used to initialize the node (using malloc). Other pointers in the node struct are set to NULL.

```
NODE *addChild(NODE* parent, float value, BSTATE *board);
```

Inputs: parent node and the value and board to initialize the child node.

Outputs: pointer to child node

Description: Creates a child node pointer by calling createNode() with the inputs and sets the child pointer of parent node to the created node.

```
NODE *addSibling(NODE *child, float value, BSTATE *board);
```

Inputs: child(sibling) node and board to initialize the sibling node.

Outputs: pointer to sibling node

Description: Create a node by calling createNode() with inputs value and board. Then sets the next pointer of child to created sibling node.

```
void removeNode(NODE *node);
```

Inputs: node

Outputs: void

Description: deletes the node with its child and siblings by recursively calling free();

```
NODE *alphabeta(NODE* node, WEIGHTS *weights, float alpha, float beta, PLAYER minmax);
```

Inputs: node that contains current board, weights for evaluation function, alpha and beta values for pruning, PLAYER which defines if the node is a maximizer or minimizer.

Outputs: pointer to node that contains best worst move.

Description: Use the minimax with alpha-beta pruning algorithm to find the node with best worst float value in the first layer of the tree. Then it returns the pointer to that node.

```
/* MENTRY *minmax (BSTATE *currentBoard, WEIGHTS *weights) */
```

Inputs: current board and weight for evaluation function.

Outputs: move entry that contains the best worst move.

Description: creates the tree by calling the function that finds all

legal moves in a loop. This loop is stopped by time. After the tree is created, `alphabeta()` is called. The pointer to `MENTRY` inside the node returned by `alphabeta()` is returned from this function.

## 11 Index

Attack 3 Artificial Intelligence (AI) 3, 6, 8  
Bishop 4  
Capture 3, 4 Check 3, 4, 5, 7, 8 Checkmate 3, 5, 7  
Download 6 Draw 3, 4, 7  
En Passant 3 File 3 Forfeit 3  
Graphical User Interface (GUI) 6, 7  
King 4 Knight 4  
Neural Network 6, 8  
Menu 5, 6, 7 Pawn 3, 4 Ply 4 Promoting 4  
Queen 4  
Rank 4, 5 Rook 3  
Stalemate 4 Strategy 6