King Me!

Developer Manual
Version 1.0

Team 11
Marshall Payatt
Keenan Lau
Zhifang Zeng
Armando Rodriguez
Aria Javanmard

University of California Irvine

Table of Contents

Glossary

AI: Artificial Intelligence, in this case, a computer controlled player

GUI: Graphical User Interface

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

Control: The set of modules responsible for program flow

View: The set of modules responsible for program display

Quit: Quitting the game and returning to the main menu

Exit: Exiting and closing the program

Software Architecture Overview

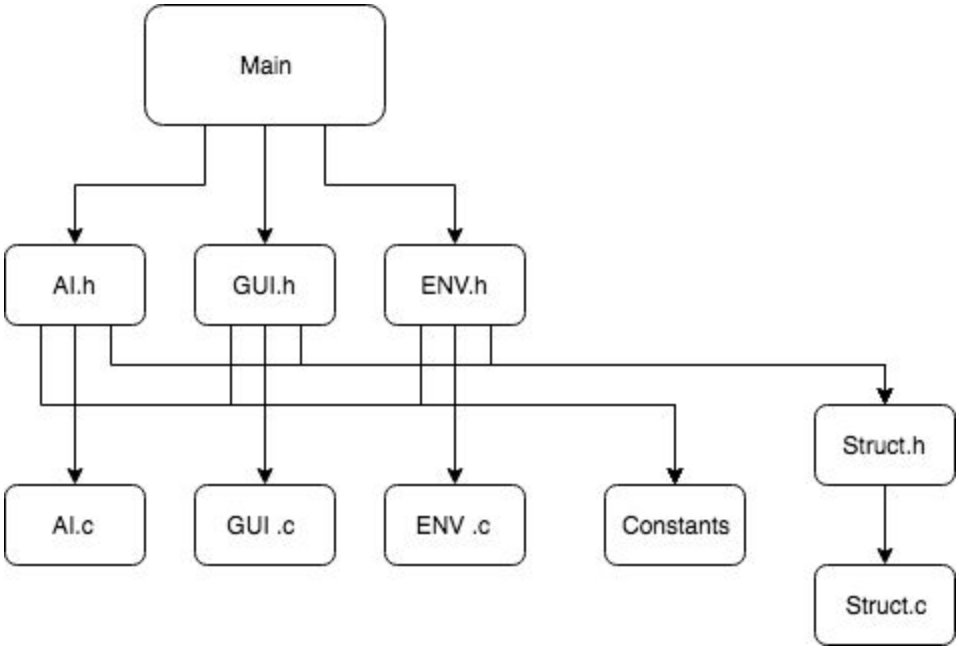
Main Data Types and Structures

For more information on the function and purposes of these structs, check the Detailed description of data structures section.

```
typedef struct _Player
{
   int id;
   int color;
   int identity;
   int difficulty;
} Player;

typedef struct _LegalMovesContainer
{
   int pos;
   vector legal_moves;
}LegalMovesContainer;
```

```
typedef struct _Node
{
    char* log;
    struct Node *next;
    struct Node *prev;
}Node;

typedef struct _Move{
    int piece;
    int start_pt;
    int end_pt;
    int captured;
    int captured_pos;
    int special_move;
} Move;

typedef struct _GameState
{
    int playerTurn;
    Castling castling_arr[2];
    int board[8*8];
    LegalMovesContainer container[16];
    int moves_vector_cnt;
    Node moves_stack;
} GameState;
```

Major Software Components



Module interfaces

GUI

- `int gui_init_window(int argc, char*argv[]);`
- `int gui_free_video();`
- `int gui_example();`
- `void gui_init(GameState *gameState,Player player_arr[2]);`
- `int gui_main_menu();`
- `Player gui_player_menu();`
- `int gui_play(GameState *gameState,Player *player);`
- `void gui_quit_window();`
- `void gui_refresh();`
- `void gui_player_HvC_menu(Player* player_arr);`
- `void gui_player_HvH_menu(Player* player_arr);`
- `void gui_player_CvC_menu(Player* player_arr);`
- `void gui_gameplay_window();`
- `int gui_play(GameState *gameState,Player *player);`
- `void gui_refresh(GameState *gameState,Player *player_arr);`

AI

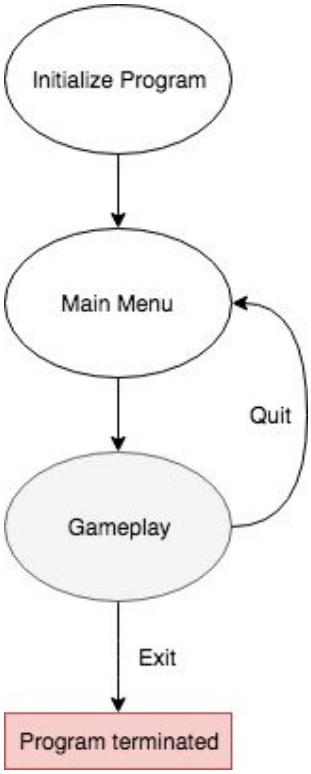
- `int ai_model1_play(GameState *gameState, Player *player);`
- `int ai_model1_simulate(GameState *gameState, Player *player, int depth)`
- `void ai_print_board(GameState *gameState)`

ENV

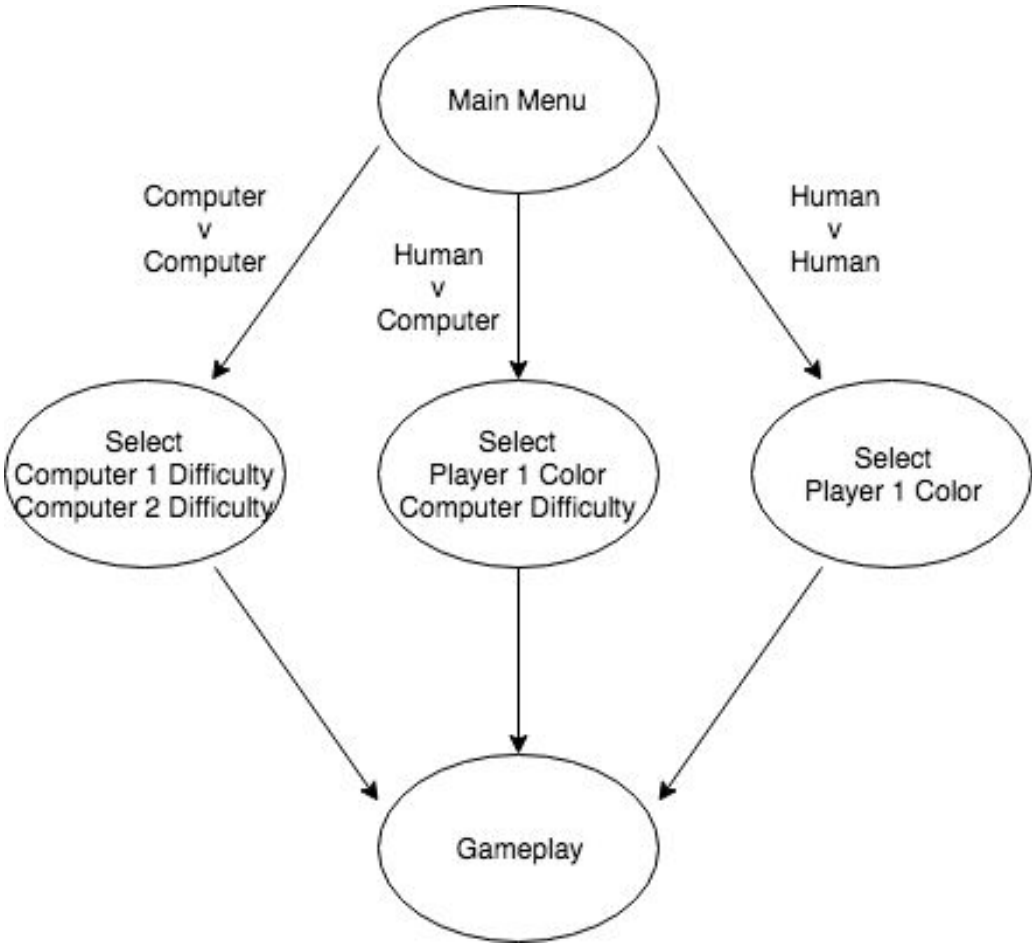
- `void env_play(GameState *gameState, Player *player, int start_pt, int end_pt);`
- `vector env_get_legal_moves(GameState *gameState, Player *player, int start_pt);`
- `vector env_get_legal_pawn(GameState *gameState, int start_pt);`
- `vector env_get_legal_knight(GameState *gameState, int start_pt);`
- `vector env_get_legal_castle(GameState *gameState, int start_pt);`
- `vector env_get_legal_bishop(GameState *gameState, int start_pt);`
- `vector env_get_legal_queen(GameState *gameState, int start_pt);`
- `vector env_get_legal_king(GameState *gameState, int start_pt);`
- `uchar env_check_end(GameState *gameState, Player *player);`
- `uchar env_is_threatened(GameState*,Player*);`
- `GameState env_copy_State(GameState *gameState);`
- `void env_free_container(GameState *gameState);`

Overall Program Control Flow

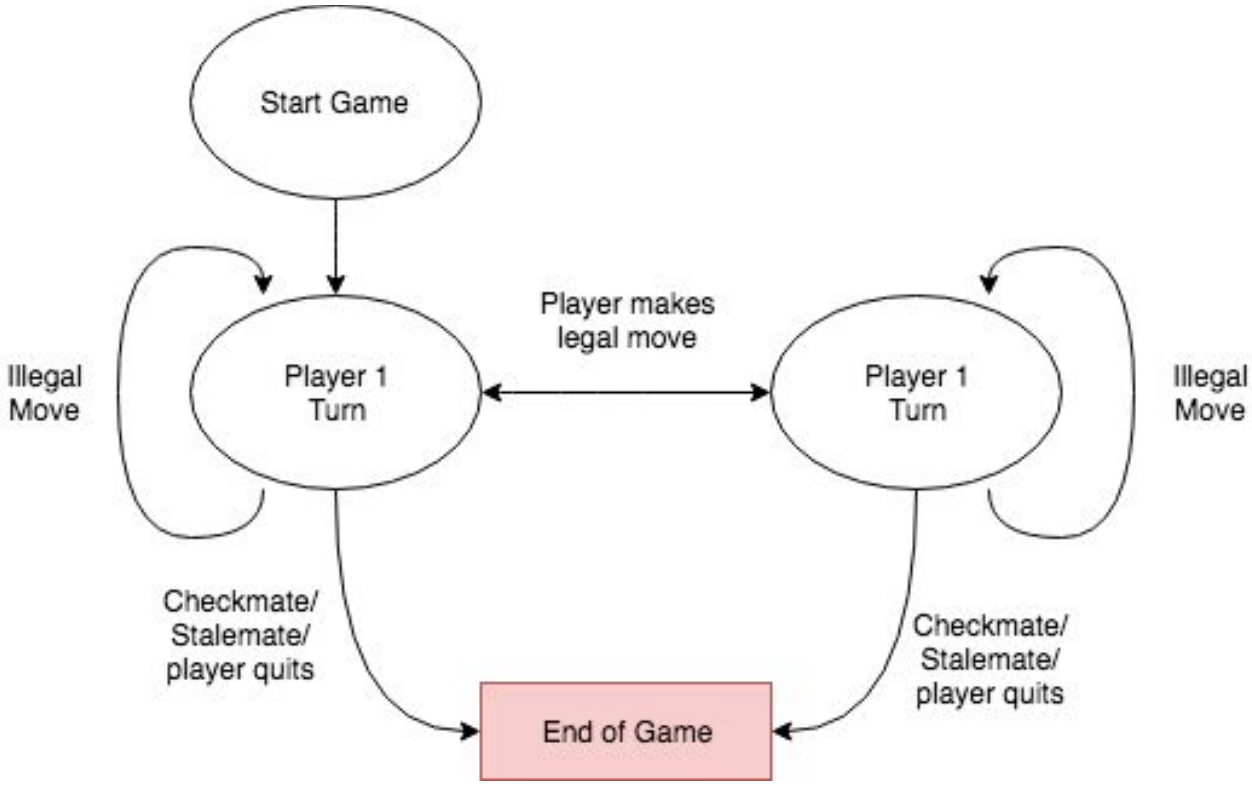
- Top Level Control Flow



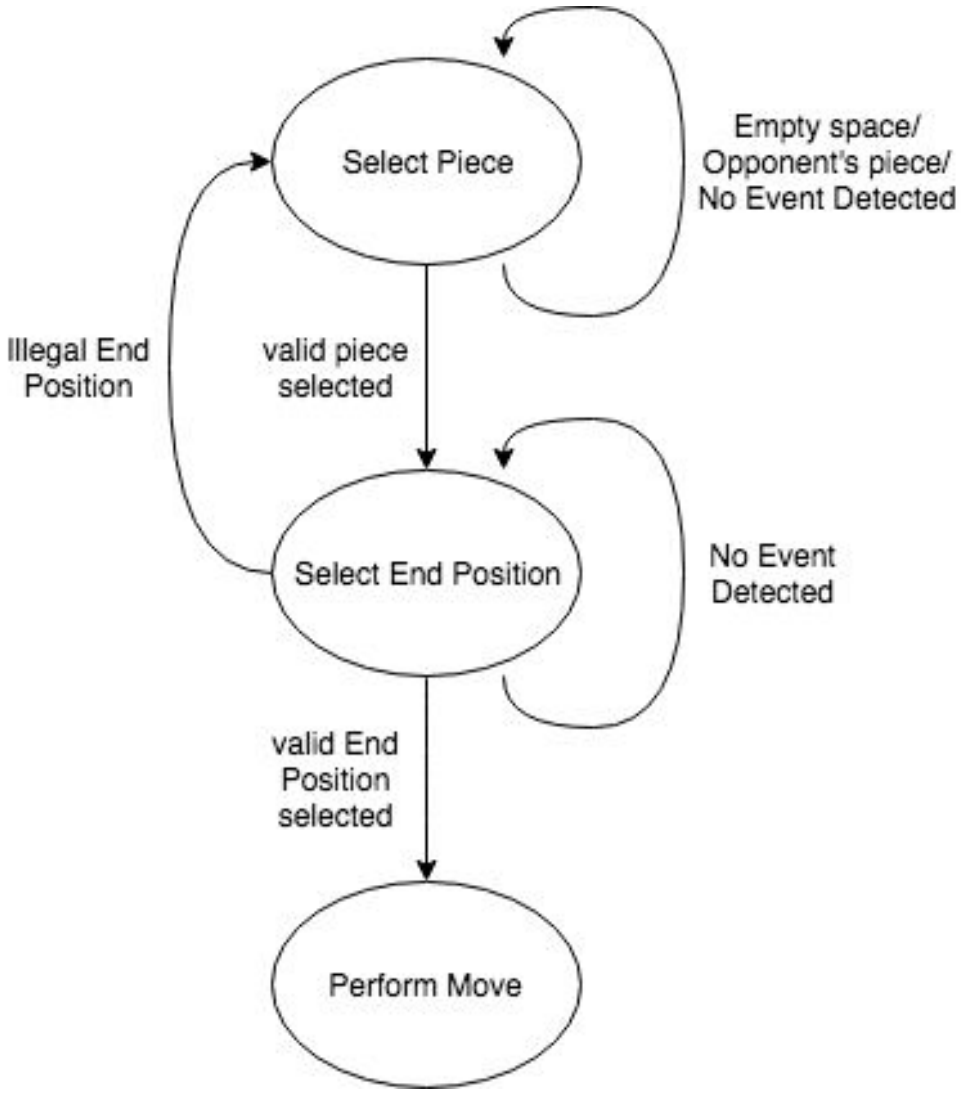
- Main Menu Control Flow



- Gameplay Control Flow



- Move Control Flow



Installation

System Requirements, Compatibility

- Hardware: PC Hardware (x86_64 server)
- Operating system: Linux OS (RHEL-6-x86_64)
- Dependent third-party software:
    - i. gcc
    - ii. GNU make
    - iii. cmake
- Dependent libraries:
    - i. GTK for graphical user interface

Setup and Configuration

- GTK + 2.0 library installation: Instructions can be found at https://www.gtk.org/download/index.php

Building, compilation, installation

Methods 1

- The software comes in a tar.gz package. After downloading, extract the package by running:
    - tar –zxvf chess.tar.gz
- Change into the directory by running:
    - cd chess
- Compile the code by running:
    - mkdir build
    - cd build
    - cmake ..
    - make
- Run the program by running
    - Return to root directory
    - ./bin/chess

Method 2

Script for ease of use:

- The software comes in a tar.gz package. After downloading, extract the package by running:
    - tar –zxvf chess.tar.gz

- Change into the directory by running:
    - cd chess
- To compile run this command:
    - bash SuperSimpleIntstallAndRun.sh

Documentation of packages, modules, interfaces

A Detailed Description of Data Structures

Player: The Player structure contains basic information on the player in question, including its id, color, if its an AI or player and difficulty if it is an AI. Two of the player structs are stored in the Game function. All are represented as Ints, with -1 meaning black and 1 meaning white.

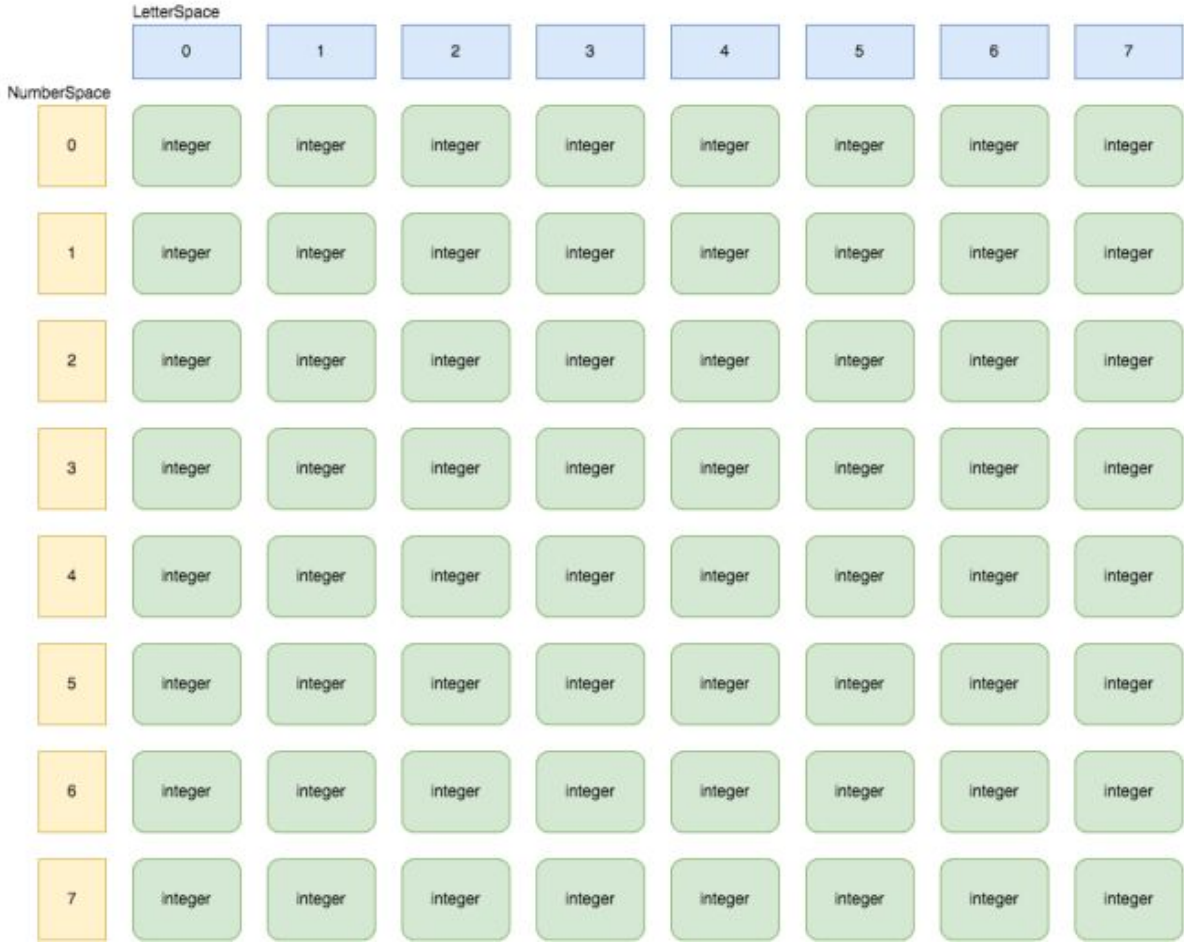
Gamestate: In short, it is a structure that contains the information on the current state of the game. This includes the playerturn(int), an array that keeps track of casting, the board, an array of legalMoveContainers, a move vector and a moves_stack. Its purpose is to not only exist to be passed from one player to another every turn but to allow the AI to simulate boards steps ahead by undoing.

LegalMovesContainer: This struct is intended to contain all of the relevant information for a single piece's possible legal moves. Starting with its pos, or rather its position, it is accompanied by a vector containing all possible endpoints.

Node: This Node contains a pointer to a log of type char and pointers to its next entry. This combines to make a structured list for the purpose of a move log for the game.

Move: This struct has all useable information about a move. The piece type moving, start and end position, type of captured piece if applicable, if a special move was used, etc.

In addition, the board itself is represented with an 8 by 8 array of integers, with certain numbers representing pieces. This is neither a struct or a data type, but this is important to the program's functioning.



Pieces are represented with the following Integer codes. Positive numbers 1-6 represent white pieces and (-)1-6 representing black pieces. Any 0 represents a blank space without a piece.

	White Pawn	White Knight	White Castle	White Bishop	White Queen	White King
# representation	1	2	3	4	5	6
Empty Spaces = 0						
	Black Pawn	Black Knight	Black Castle	Black Bishop	Black Queen	Black King
# representation	-1	-2	-3	-4	-5	-6

Detailed description of functions and parameters

GUI

- `int gui_init_window(int argc, char*argv[]);`
    - This function is used to initialize the window and creates a new GTK thread to show the window
    - int argc: number of arguments for function
    - char*argv[]: the arguments of the function
    - Return 0 if successful
- `void gui_init(GameState *gameState,Player player_arr[2]);`
    - This function displays the main menu and the players menu.
    - GameState *gameState: a given board state
    - PLayer player_arr[2]: an array of the two players in the game
- `int gui_main_menu();`
    - This function displays the main menu and returns the users choices
    - Returns the gamemode selected coded in an int

- `int gui_play(GameState *gameState,Player *player);`
- `void gui_player_HvC_menu(Player* player_arr);`
    - This function draws the Human vs Computer menu and gives information to player array
- `void gui_player_HvH_menu(Player* player_arr);`
    - Same as HvC
- `void gui_player_CvC_menu(Player* player_arr);`
    - Same as HvC
- `void gui_gameplay_window(GameState *gameState);`
    - This function draws the gameplay window after the mode is selected
- `int gui_play(GameState *gameState,Player *player);`
    - This function connects user input on the window and let the user makes their move and then disconnect
    - Return 0 for success
- `void gui_refresh(GameState *gameState,Player *player_arr);`
    - Refresh the gui board
- `void gui_quit_window();`
    - Quits the window

AI

- `int ai_model1_play(GameState *gameState, Player *player);`
  - This function provides the AI functionality
  - GameState *gameState: a given board state
  - Player *player: a given player
  - Returns 0 if continue
  - Reutnr 1 if game end (checkmate, quit, etc)
- `int ai_model1_simulate(GameState *gameState, Player *player, int depth)`
  - This function allows AI to simulate to “see” ahead
  - int depth: current depth of the simulation
- `void ai_print_board(GameState *gameState)`
  - This function prints the board, debugging

ENV

- `void env_play(GameState *gameState, Player *player, int start_pt, int end_pt);`
  - This function calls other functions in ENV.c and provides the gameplay functionality, such as moving pieces, controlling player turn and everything to do with gameplay
  - GameState *gameState: a given board state
  - Player *player: a given player
  - int start_pt: a selected piece to be moved
  - int end_pt: a selected end point when making a legal move
- `vector env_get_legal_moves(GameState *gameState, Player *player, int start_pt);`
  - This function generates a list of all legal moves for a given player on a given turn on a given coordinate.
  - Returns a vector of all legals spaces to move to
- `vector env_get_legal_pawn(GameState *gameState, int start_pt);`
  - This function returns all legals moves that the pawn piece can current perform
- `vector env_get_legal_knight(GameState *gameState, int start_pt);`
  - Same as pawn…
- `vector env_get_legal_castle(GameState *gameState, int start_pt);`
- `vector env_get_legal_bishop(GameState *gameState, int start_pt);`
- `vector env_get_legal_queen(GameState *gameState, int start_pt);`
- `vector env_get_legal_king(GameState *gameState, int start_pt);`
- `uchar env_check_end(GameState *gameState, Player *player);`
  - This function checks for ends state (check mate, stalemate etc)
  - Returns a number signaling 1 for end 0 for continue

- `uchar env_is_threatened(GameState*,Player*);`
    - This function checks if the king is in check
    - Returns a number signaling 1 for check 0 for no check
- `GameState env_copy_State(GameState *gameState);`
    - This function creates a copy of current game state
    - Returns the game state
- `void env_free_container(GameState *gameState);`
    - This function frees the memory containing the list of legal moves

Detailed description of input and output formats

Syntax/format of a move input by the user

On the user side, at the start of his/her turn, the user can click any of their pieces, then that piece will be highlighted yellow and all possible moves that piece the can make are also highlighted in green. Once the user selects one of those possible moves, then the selected piece is moved to that space and the turn ends.

On the developer side, the program starts detecting any mouse button press in the gui_play() function. Within that function, it will connect that mouse button signal to call back the gui_play_callback() function and passes the game state of the board as a parameter. Within the gui_play_callback() function, it interprets the exact pixel the user clicks and translates it into a grid coordinate on the board. With that information the program then checks if the user clicks a valid selection, in example the user clicks on a piece they own, then by checking what piece is selected, the program redraw the board with a different image of the piece to show that it is selected and a different image of an empty space or enemy piece to show the possible moves the selected piece can make, the program then waits for another user click. Finally, when the user clicks on a valid option to move the selected piece, the program will update the board to reflect the movement made, and redraws the board to show that the piece has been moved.

Any invalid selection from the user will restart the process and return to the state where the program is waiting for the user to click on a valid piece.

Menu Interfaces

In the file GUI.c the menu has been implemented. The function gui_init_window will create a layout and window with the title. For the main menu that has three option for the user (One player, Two player and Computer vs Computer) the function gui_main_menu() will put an image on the background and it will return the Gamemode also it has a signal which is connected to the main_menu_callback, this function can detect the mouse click and based on the X,Y coordinates of the of the clicks it well set the Gamemode depending on the position the

user choose on the window. The way the rest of the menu works is the same, we have a function for each option and based on that option we have callbacks for those options which will detect the position the user clicks.

Syntax/format of a move recorded in the stack

To save the moves, we use the JSMN libraries to transcribe the information of the move onto the stack. This way moves can be undone with the undo function.

Development plan and timeline

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, etc
- AI: How the computer decides to make a move
- GUI: The display of program for user
- Control and Integration: Program and control flow and integration between modules and functions

Team member responsibilities

As discussed above, four areas will be responsible by the following team members:
- Gameplay: Michael, Keenan, Marshall, Aria
- AI: Michael, Keenan
- GUI: Armando,Michael
- Control and Integration: Marshall, Keenan

Timeline

By January 27:
- Finish basic prototype of the game
- Finish basic moves
- Support past move list
- Support Undo

By February 3:
- Finish GUI output
- Finish castling, transforming and en passant
- Support AI

Back matter and Visual Presentation

Copyright

References

Index