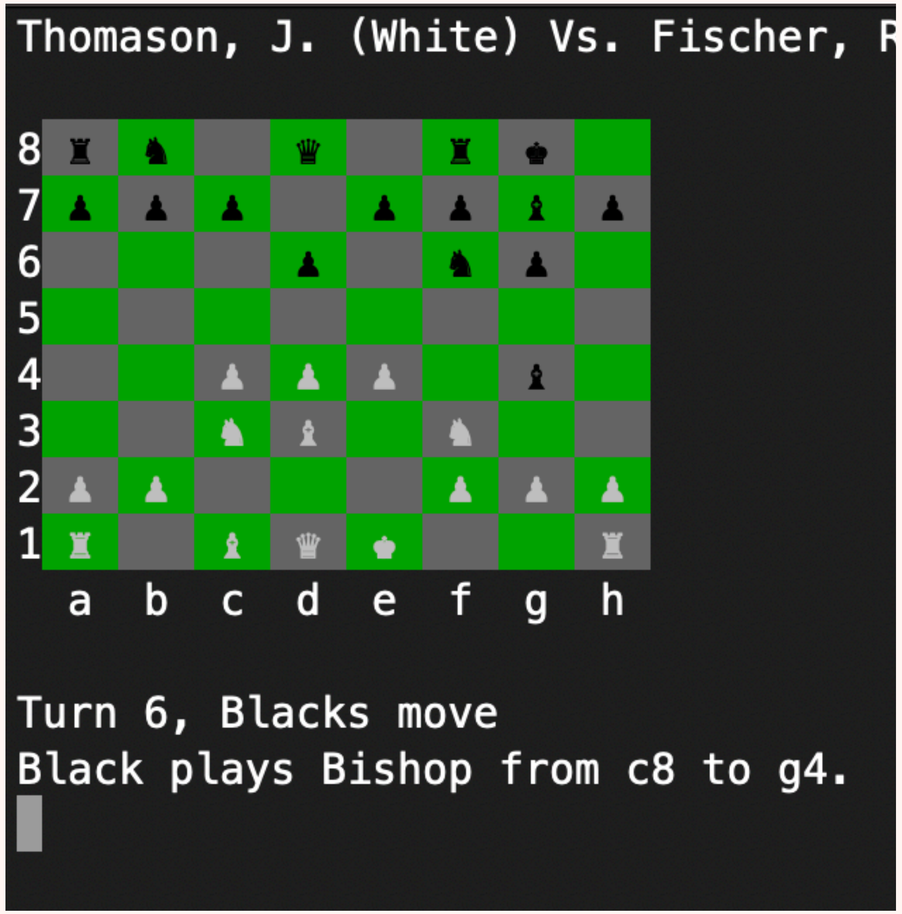
CHESS VIEWER

By Daniel Walters



BACKGROUND INFO

Portable Game Notation (PGN) is a standard plain text format for recording chess games. It records the moves played (in standard algebraic notation), and any related data such as player names, the outcome of the match, when and where it was played, and the rating of each player.

All online games and the majority of over-the-board games are recorded in .pgn format, and games dating back to as early as the 1400s have been moved to this format.

On move 1 white plays pawn to D4, then black plays Knight to F6. On move 2 white plays pawn to C4, then black plays pawn to G6. ...

An example PGN file:

[Event "USA-chJ"]
[Site "?"]
[Date "1955.??.??"]
[Round "?"]
[White "Thomason, J."]
[Black "Fischer, Robert James"]
[Result "0-1"]
[WhiteElo ""]
[BlackElo ""]
[ECO "E91"]

1.d4 Nf6 2.c4 g6 3.Nc3 Bg7 4.e4 d6 5.Nf3 O-O 6.Bd3 Bg4 7.O-O Nc6 8.Be3 Nd7
9.Be2 Bxf3 10.Bxf3 e5 11.d5 Ne7 12.Be2 f5 13.f4 h6 14.Bd3 Kh7 15.Qe2 fxe4
16.Nxe4 Nf5 17.Bd2 exf4 18.Bxf4 Ne5 19.Bc2 Nd4 20.Qd2 Nxc4 21.Qf2 Rxf4 22.Qxf4 Ne2+
23.Kh1 Nxf4 0-1

WHAT DOES CHESS VIEWER DO?

Chess Viewer can take any valid pgn file and replay the game on an ASCII chess board.
It comes with two modes, Automatic and Manual.

In Automatic mode, the game is played out in its entirety.

In Manual mode, the user can go to the next/previous move, or skip forwards/backwards several moves at their own convenience

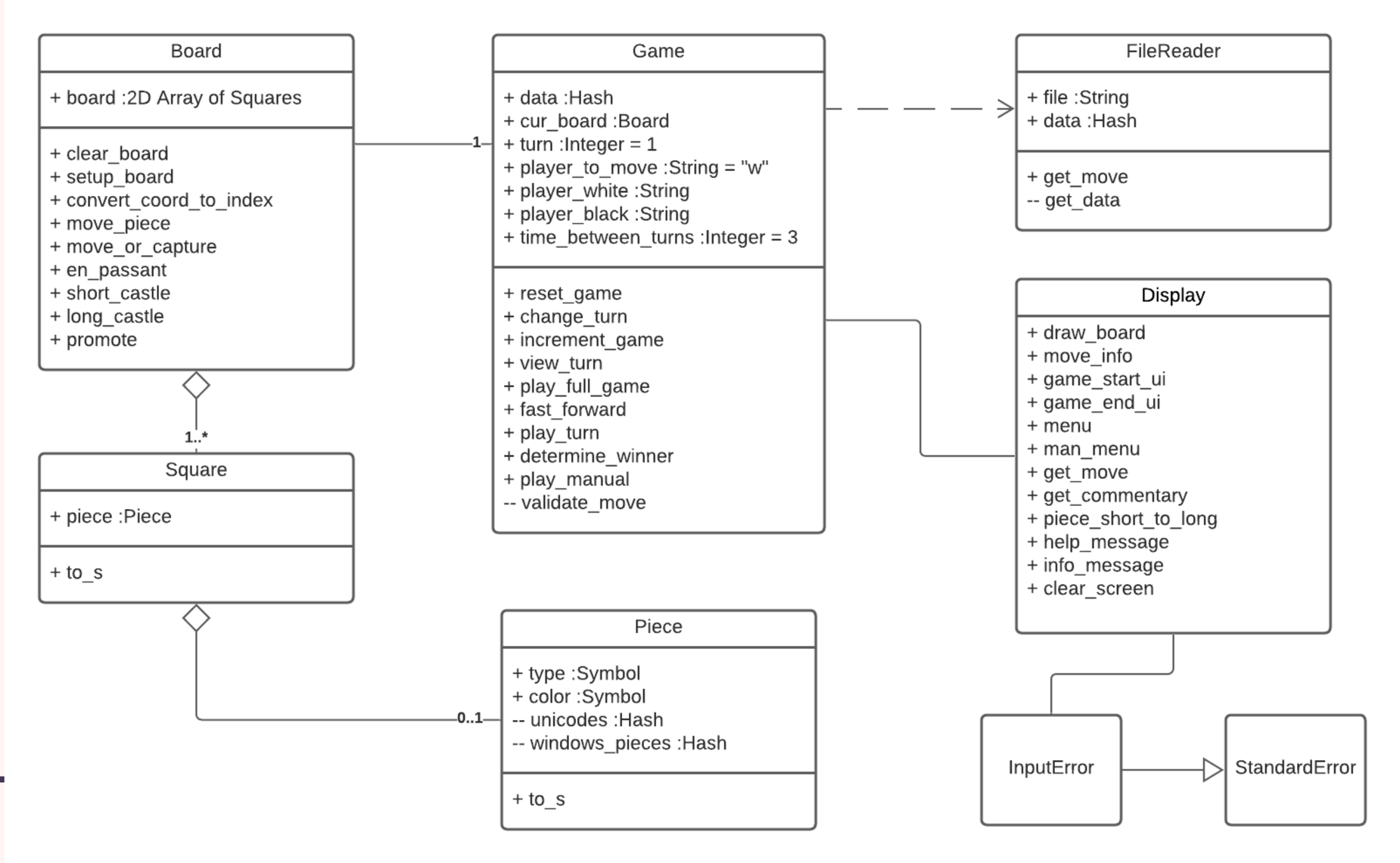
DEMONSTRATION

- Running from Bash Script
- Automatic Mode Demo
- Running index.rb directly
- Manual Mode Demo

CODE OVERVIEW

- Class Diagram
- FileReader
- Board
- Game
- Display
- Argument Handling

CLASS DIAGRAM



FILE_READER.RB

get_data

```ruby
def get_data
    json_file = Egd::Builder.new(File.read(@file)).to_json
    data = JSON.parse(json_file)
end
```

get_data converts pgn file to json format and then stores it in a hash


```
{"game_tags"=>
  {"Result"=>"0-1",
   "Event"=>"USA-chJ",
   "Site"=>"?",
   "Date"=>"1955.??.??",
   "Round"=>"?",
   "White"=>"Thomason, J.",
   "Black"=>"Fischer, Robert James",
   "WhiteElo"=>"",
   "BlackElo"=>"",
   "ECO"=>"E91"},
 "moves"=>
  {"1w"=>
    {"start_position"=>
      {"fen"=>"rnbqkbnr/pppppppp/8/8/8/8/PPPPPPPP/RNBQKBNR w KQkq - 0 1",
       "features"=>{}},
     "move"=>
      {"player"=>"w",
       "san"=>"d4",
       "lran"=>"d2-d4",
       "from_square"=>"d2",
       "to_square"=>"d4",
       "piece"=>"p",
       "move_type"=>"move"},
     "end_position"=>
      {"fen"=>"rnbqkbnr/pppppppp/8/8/3P4/8/PPP1PPPP/RNBQKBNR b KQkq d3 0 1",
       "features"=>{}}},
   "1b"=>
    {"start_position"=>
      {"fen"=>"rnbqkbnr/pppppppp/8/8/3P4/8/PPP1PPPP/RNBQKBNR b KQkq d3 0 1",
       "features"=>{}},
     "move"=>
      {"player"=>"b",
       "san"=>"Nf6",
       "lran"=>"Ng8-f6",
       "from_square"=>"g8",
       "to_square"=>"f6",
       "piece"=>"N",
       "move_type"=>"move"},
     "end_position"=>
      {"fen"=>"rnbqkb1r/pppppppp/5n2/8/3P4/8/PPP1PPPP/RNBQKBNR w KQkq - 1 2",
       "features"=>{}}},
```


get_move

```ruby
def self.get_move(move, data)
    begin
    from = data["moves"][move]["move"]["from_square"]
    to = data["moves"][move]["move"]["to_square"]
    type = data["moves"][move]["move"]["move_type"]
    piece = data["moves"][move]["move"]["piece"]
    data["moves"][move]["move"]["promotion"] ? promote_to = data["
    data["moves"][move]["move"]["captured_piece"] ? captured_piece
    data["moves"][move]["end_position"]["features"]["check"] ? chec
    data["moves"][move]["end_position"]["features"]["checkmate"] ?
    rescue
        return nil
    end
    {:type => type, :from => from, :to => to, :piece => piece, :pro
     :captured_piece => captured_piece, :check => check, :checkmate
end
```

Move such as “1w” passed into method, gathers the relevant information that the program needs and returns a different hash

BOARD.RB, SQUARE.RB, PIECE.RB

piece.rb

```ruby
require "os"
You, 2 days ago | 1 author (You)
class Piece

    attr_reader :type, :color

    @@unicodes = {  :Pawn => "\u265F",
                    :Knight => "\u265E",
                    :Bishop => "\u265D",
                    :Rook => "\u265C",
                    :Queen => "\u265B",
                    :King => "\u265A"}

    @@windows_pieces = {:Pawn => "P",
                        :Knight => "N",
                        :Bishop => "B",
                        :Rook => "R",
                        :Queen => "Q",
                        :King => "K"}

    def initialize(type, color)
        @type  = type
        @color = color
    end

    def to_s
        if OS.windows? then @@windows_pieces[type].colorize(color)
        else @@unicodes[type].colorize(color) end
    end
end
```

square.rb

```ruby
You, 6 days ago | 1 author (You)
require "colorize"
You, a week ago | 1 author (You)
class Square
    attr_accessor :piece

    def initialize(piece = nil)
        @piece = piece
    end

    def to_s
        if @piece
            " #{piece.to_s} "
        else
            "   "
        end
    end
end      You, a week ago • first commi
```

(2D Array of Squares)

Board

BOARD.RB

move_piece(move_info)

```
def move_piece(move_info)
    from_ind = convert_coord_to_index(move_info[:from])
    to_ind = convert_coord_to_index(move_info[:to])
    type = move_info[:type]

    case type
    when "short_castle"
        short_castle(from_ind[:row])
    when "long_castle"
        long_castle(from_ind[:row])
    when "promotion_capture", "promotion"
        promote(from_ind, to_ind, move_info[:promote_to])      You, 5
    when "ep_capture"
        en_passant(from_ind, to_ind)
    else
       move_or_capture(from_ind, to_ind)
    end
end

def move_or_capture(from_ind, to_ind)
    #grab piece to be moved
    piece_to_be_moved = @board[from_ind[:row]][from_ind[:col]].piece
    #place it where it is moving to
    @board[to_ind[:row]][to_ind[:col]].piece = piece_to_be_moved
    #empty the original square
    @board[from_ind[:row]][from_ind[:col]].piece = nil
end
```

move_info (FileReader.get_move) is passed in and coordinates turned into index position in the 2D array. Determines what kind of move happened and calls relevant function.

move_or_capture:
Grab the piece to be moved, place it on the square its moving to, remove it from its original square

convert_coord_to_index

```
def convert_coord_to_index(coord)
    col = ((coord[0].ord - 49).chr).to_i
    row = 7 - (coord[1].to_i - 1)

    {:row => row, :col => col}
end
```

Column: letter (which is always downcase) converted to ascii numeric value. -49 to change value to ascii numeric value of a number (a-c => 0-2...) This is converted back to a character and then into an integer (more info on next slide)

Row: flips the coordinate because (a,1) is bottom left of board but top left of array

ASCII TABLE

Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char
0	0	[NULL]	32	20	[SPACE]	64	40	@	96	60	`
1	1	[START OF HEADING]	33	21	!	65	41	A	97	61	a
2	2	[START OF TEXT]	34	22	"	66	42	B	98	62	b
3	3	[END OF TEXT]	35	23	#	67	43	C	99	63	c
4	4	[END OF TRANSMISSION]	36	24	$	68	44	D	100	64	d
5	5	[ENQUIRY]	37	25	%	69	45	E	101	65	e
6	6	[ACKNOWLEDGE]	38	26	&	70	46	F	102	66	f
7	7	[BELL]	39	27	'	71	47	G	103	67	g
8	8	[BACKSPACE]	40	28	(	72	48	H	104	68	h
9	9	[HORIZONTAL TAB]	41	29	)	73	49	I	105	69	i
10	A	[LINE FEED]	42	2A	*	74	4A	J	106	6A	j
11	B	[VERTICAL TAB]	43	2B	+	75	4B	K	107	6B	k
12	C	[FORM FEED]	44	2C	,	76	4C	L	108	6C	l
13	D	[CARRIAGE RETURN]	45	2D	-	77	4D	M	109	6D	m
14	E	[SHIFT OUT]	46	2E	.	78	4E	N	110	6E	n
15	F	[SHIFT IN]	47	2F	/	79	4F	O	111	6F	o
16	10	[DATA LINK ESCAPE]	48	30	0	80	50	P	112	70	p
17	11	[DEVICE CONTROL 1]	49	31	1	81	51	Q	113	71	q
18	12	[DEVICE CONTROL 2]	50	32	2	82	52	R	114	72	r
19	13	[DEVICE CONTROL 3]	51	33	3	83	53	S	115	73	s
20	14	[DEVICE CONTROL 4]	52	34	4	84	54	T	116	74	t
21	15	[NEGATIVE ACKNOWLEDGE]	53	35	5	85	55	U	117	75	u
22	16	[SYNCHRONOUS IDLE]	54	36	6	86	56	V	118	76	v
23	17	[ENG OF TRANS. BLOCK]	55	37	7	87	57	W	119	77	w
24	18	[CANCEL]	56	38	8	88	58	X	120	78	x
25	19	[END OF MEDIUM]	57	39	9	89	59	Y	121	79	y
26	1A	[SUBSTITUTE]	58	3A	:	90	5A	Z	122	7A	z
27	1B	[ESCAPE]	59	3B	;	91	5B	[	123	7B	{
28	1C	[FILE SEPARATOR]	60	3C	<	92	5C	\	124	7C	|
29	1D	[GROUP SEPARATOR]	61	3D	=	93	5D	]	125	7D	}
30	1E	[RECORD SEPARATOR]	62	3E	>	94	5E	^	126	7E	~
31	1F	[UNIT SEPARATOR]	63	3F	?	95	5F	_	127	7F	[DEL]

a = 97

0 = 48

97 - 49 = 48

GAME.RB

```ruby
def fast_forward(to, display = false)
    reset_game
    first_move = true

    if (to == "1w")
        play_turn
    else
        until "#{@turn}#{@player_to_move}" == to do
            !first_move ? increment_game : first_move = false
            play_turn
            if display
                view_turn
                sleep(@time_between_turn)
                Display.clear_screen
            end
        end
    end
end
```

-to is a move in the form "2w"
-Optional display argument to actually show the board

```ruby
def increment_game
    change_turn
    @turn += 1 if @player_to_move == "w"
end
```

change_turn just changes "w" to "b" and vice versa

```ruby
def play_turn
    move_info = FileReader.get_move("#{@turn}#{@player_to_move}", @data)
    @cur_board.move_piece(move_info)
end
```

'Automatic Mode' calls play_full_game

```ruby
def play_full_game
    Display.clear_screen
    Display.draw_board(@cur_board, @player_white, @player_black)
    Display.game_start_ui
    fast_forward(@data["moves"].keys[-1], true)
    Display.draw_board(@cur_board, @player_white, @player_black)
    Display.game_end_ui(determine_winner)
end
```

GAME.RB

Inside manual_mode, if user wants to go to a specific move

```
when "goto"
    move = Display.get_move
    valid  = validate_move(move)
    begin
        if valid
            first_move = false
            fast_forward(move)
        else raise InputError end
    rescue InputError => e
        puts e.message
        sleep(1)
    end
```

validate_move checks that it is located in the JSON

fast_forward to the move

Essentially restarting the game from scratch, playing up until the chosen move and then displaying the board

Going to the previous move works the same way, fast forwarding to move-1

DISPLAY.RB

```ruby
def self.draw_board(board_obj, white, black)
    board = board_obj.board

    puts "#{white} (White) Vs. #{black} (Black)"      You, 14 hours ago
    print "\n"
    board.each_with_index do |row, row_i|
        print "#{8-row_i}"
        row.each_with_index do |square, square_i|
            if (row_i + square_i).odd?
                print square.to_s.colorize(:background => :green)
            else
                print square.to_s.colorize(:background => :light_black)
            end
        end
        print "\n"
    end
    puts "  a  b  c  d  e  f  g  h "
    print "\n"
end
```

-Navigates 2D array of squares with nested ‘each’ loops
-Every 2nd square is coloured differently to get the look of a chess board.
-If a square has a piece on it, square.to_s calls piece.to_s

ARGUMENT HANDLING

```ruby
args_t =~ String []
path = ""
####ARGUMENT HANDLING#####
if ARGV.length > 0

    case
    when ARGV.include?("-h") || ARGV.include?("--help")
        Display.help_message
        exit
    when ARGV.include?("-i") || ARGV.include?("--info")
        Display.info_message
        exit
    end

    ARGV.each_with_index do |arg, i|
        if arg == "-t" || arg == "--time"
            begin
                if (ARGV[i+1]).to_i <= 0 then raise ArgumentError.new("Time must be greater than 0, using default(3)") end
                args_to_pass.push(ARGV[i+1].to_i)
            rescue ArgumentError => e
                puts e.message
                sleep(2)
                args_to_pass.push(3)
            end
        end
        if arg == "-p" || arg == "--path"
            begin
                if (!ARGV[i+1]) then raise ArgumentError.new("No Path detected, loading default") end
                path = ARGV[i+1]
            rescue ArgumentError => e
                puts e.message
                sleep(2)
                path = "pgn/FischerVsThomason.pgn"
            end
        end
    end

    if !ARGV.include?("-p") && !ARGV.include?("--path") then path = "pgn/FischerVsThomason.pgn" end
else
    path = "pgn/FischerVsThomason.pgn"
end
```

```
if there are aguments
    array=[]

    if they include -h or --help
        display help msg and exit
    if they include -i or --info
        display info msg and exit

    for each arg do
        if arg = -t or --time
            set time to arg at next index if its valid, else set default time
            push time to array
        if arg = -p or --path
            set path to arg at next index if its valid, else set default path
    end
end

    if -p or --path not in the arguments, set path to default
    if -t or --time not in the arguments, set time to default

    get data from path
    unshift data into array

    create a new game with arguments
```

```ruby
game = Game.new(*args_to_pass)
```

QUESTIONS?