CLI CHESS APP

By: Nam Nguyen 2021

DESIGN

- Why Chess?
- Game Play
- Implementation Logic
- Features

Why Chess?

Logically challenging

Played globally, no language barrer

Love the game

Test my knowledge of Ruby

Develop a better understanding

Design Process

Mouse interaction in terminal?

Keyboard to move terminal cursor to select moves

Verbal input?

Long algebraic notation
i.e. Bishop to b5: f1b5

Difficult to use for non-experienced chess player

How to print board and chess pieces in terminal

Game Play

How the user will interact with app:

1) Select piece

2) Select move

Important feature: Show a piece‘s possible moves

- Not everyone is a professional chess player
- Broadens target audience
- Better User Experience

Reasoning

LEVEL UP!

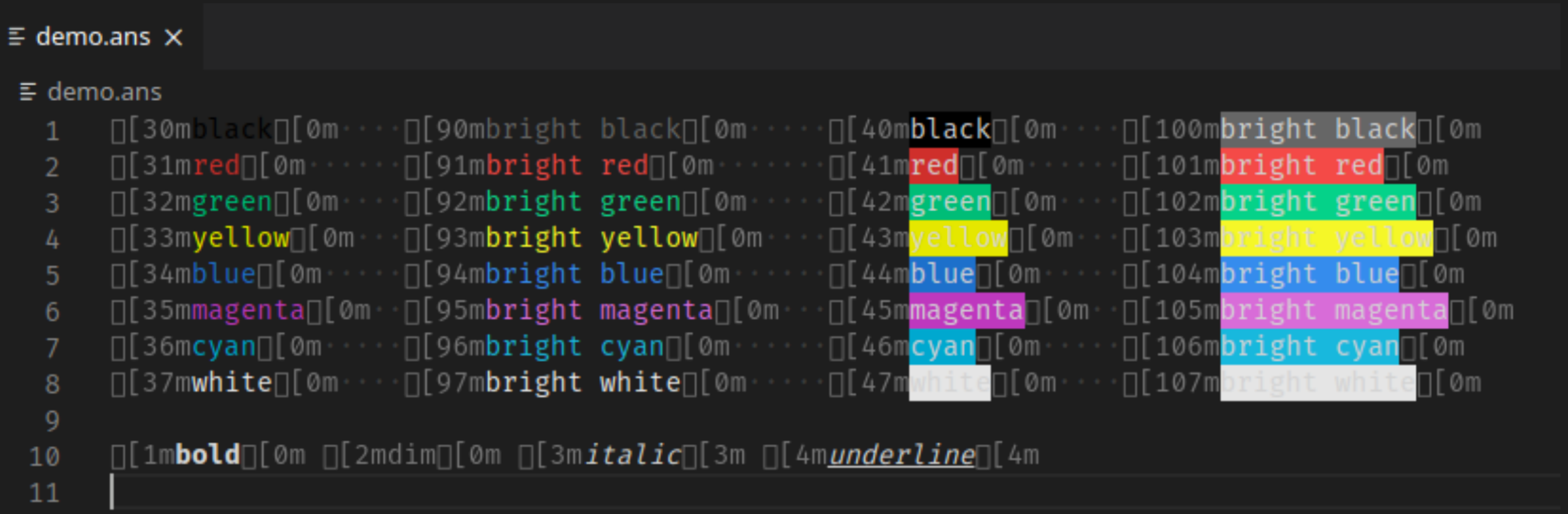
Needed to test move mechanics for all pieces:

-> Why not incorporate move validation with possible moves feature

Implementation:

- Learn how to customise font colour and background in terminal
- ANSI escape codes `\e[91m TEXT \e[0m`

ANSI Escape Codes



Features

Single Player (vs computer)

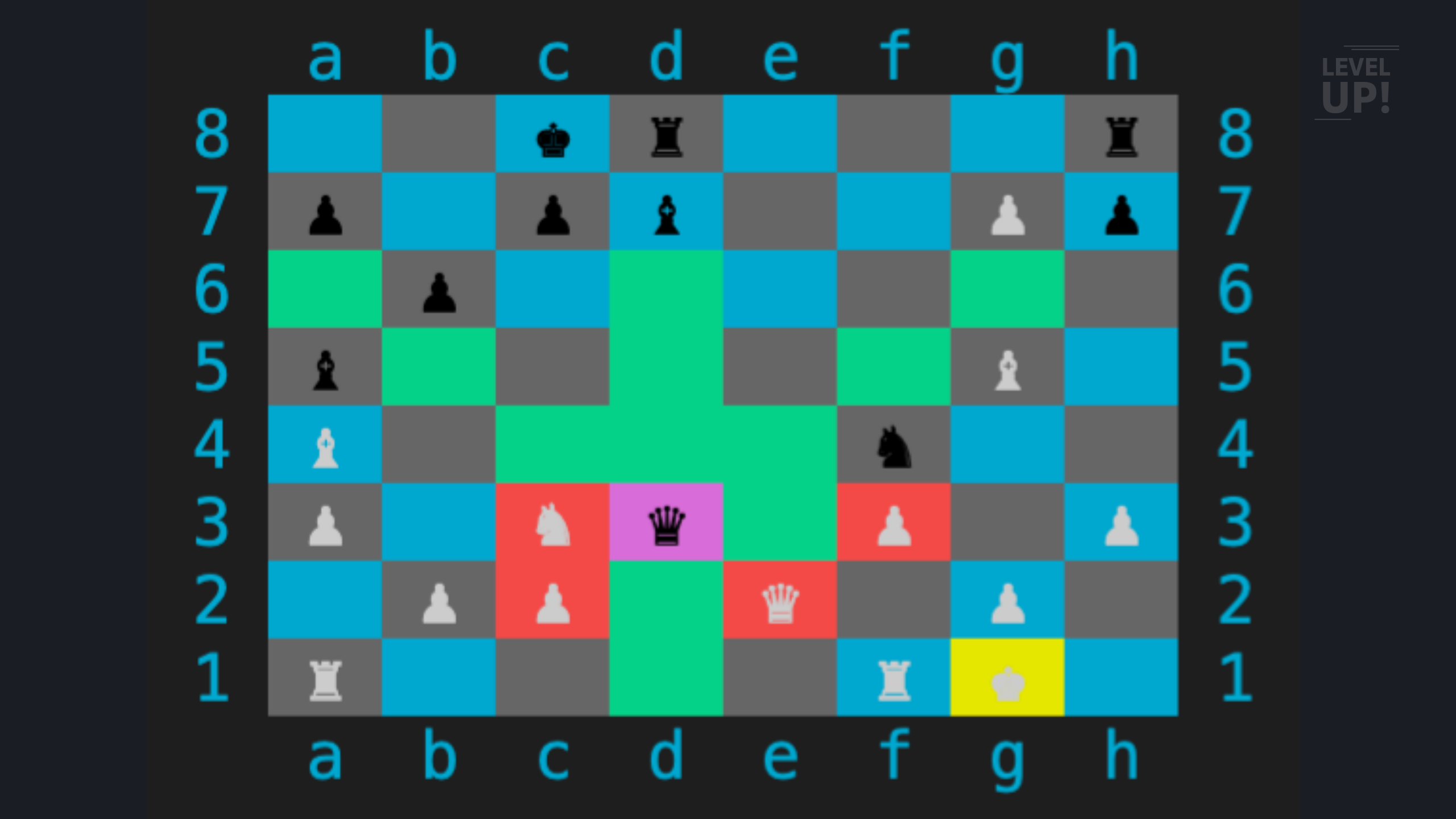
Two Player

Save Game

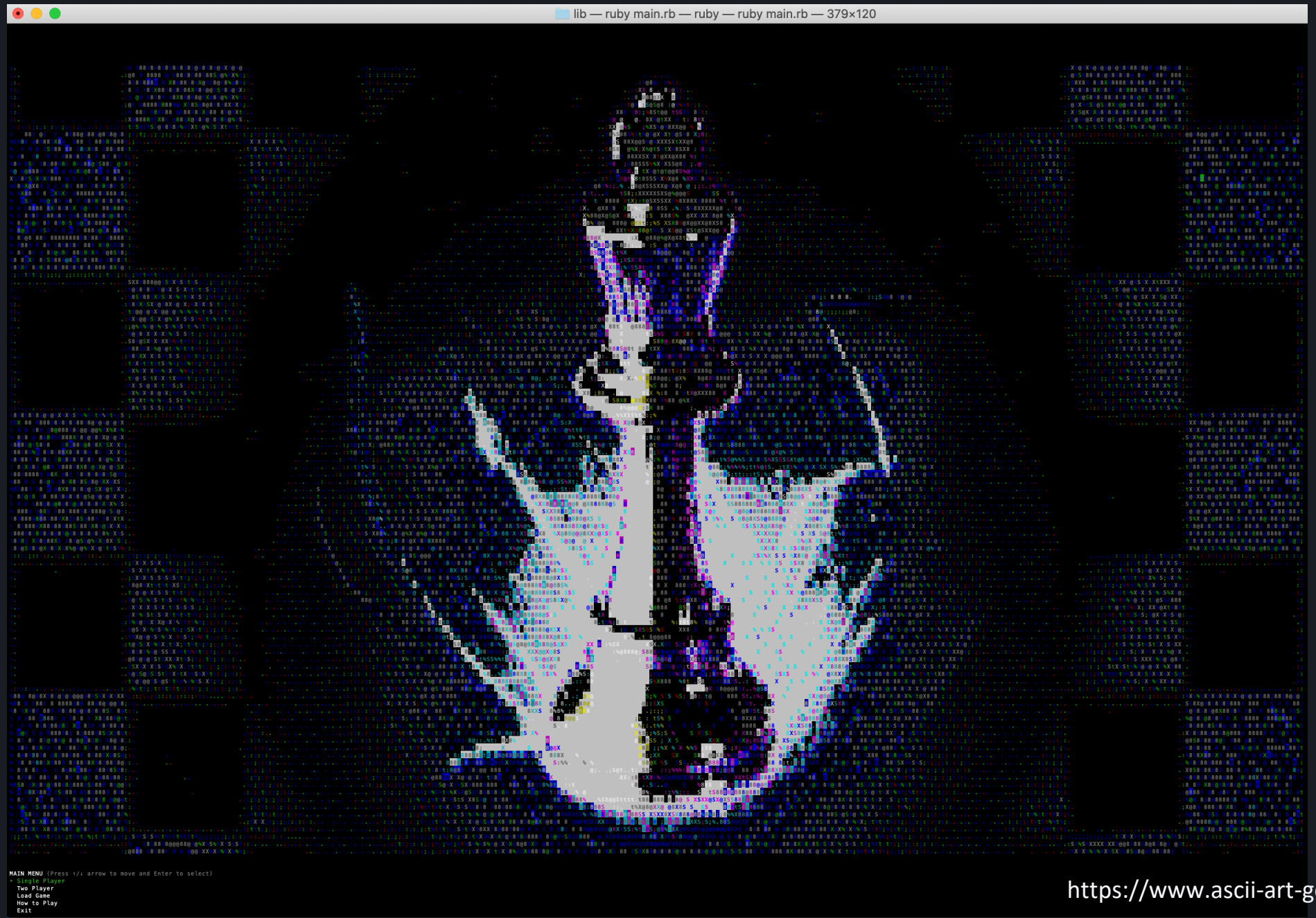
Load Game

Shows:

- Last Move
- Selected Piece
- Possible moves
- Possible captures


a b c d e f g h
8 7 6 5 4 3 2 1
LEVEL UP!

Landing Banner

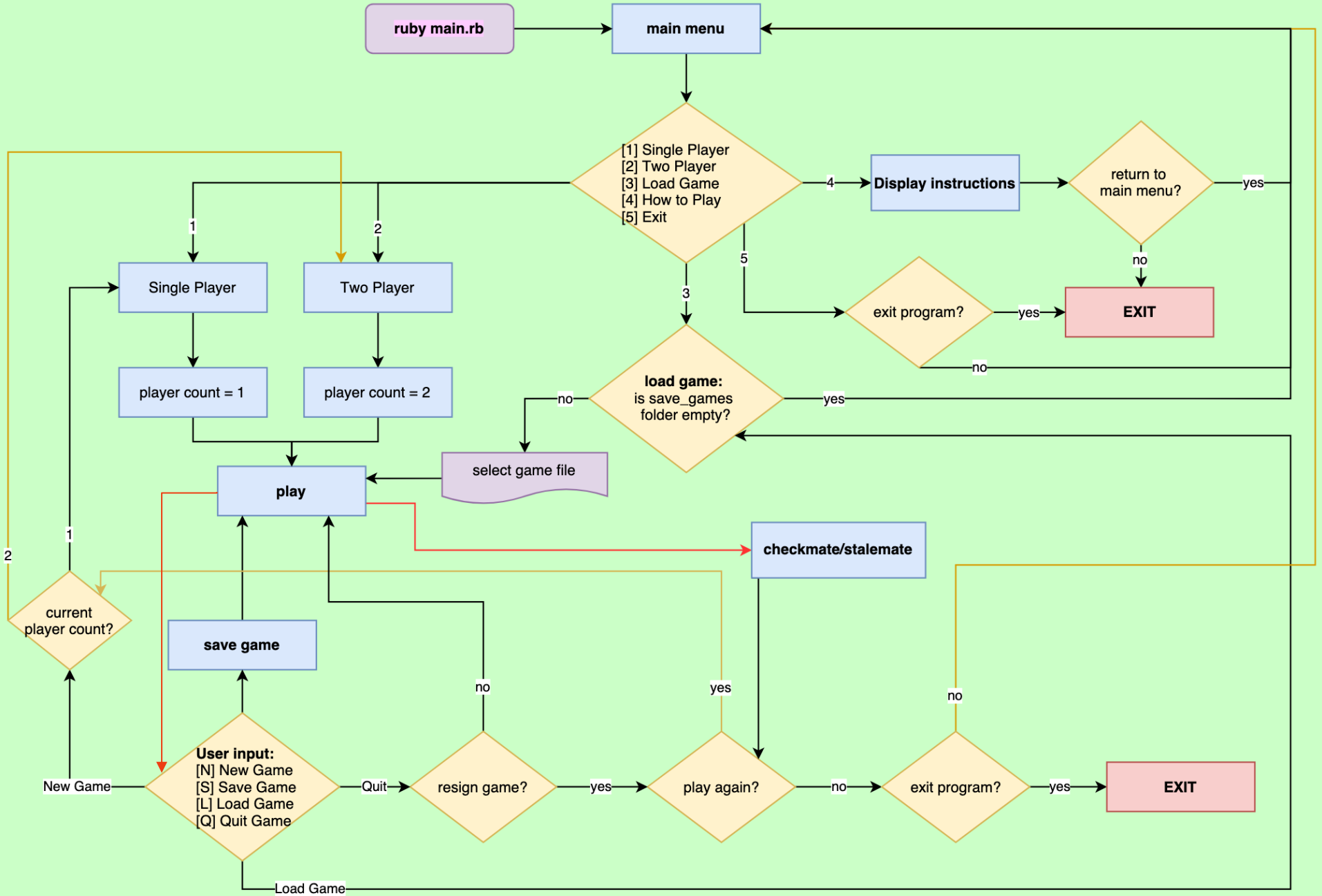


https://www.ascii-art-generator.org/

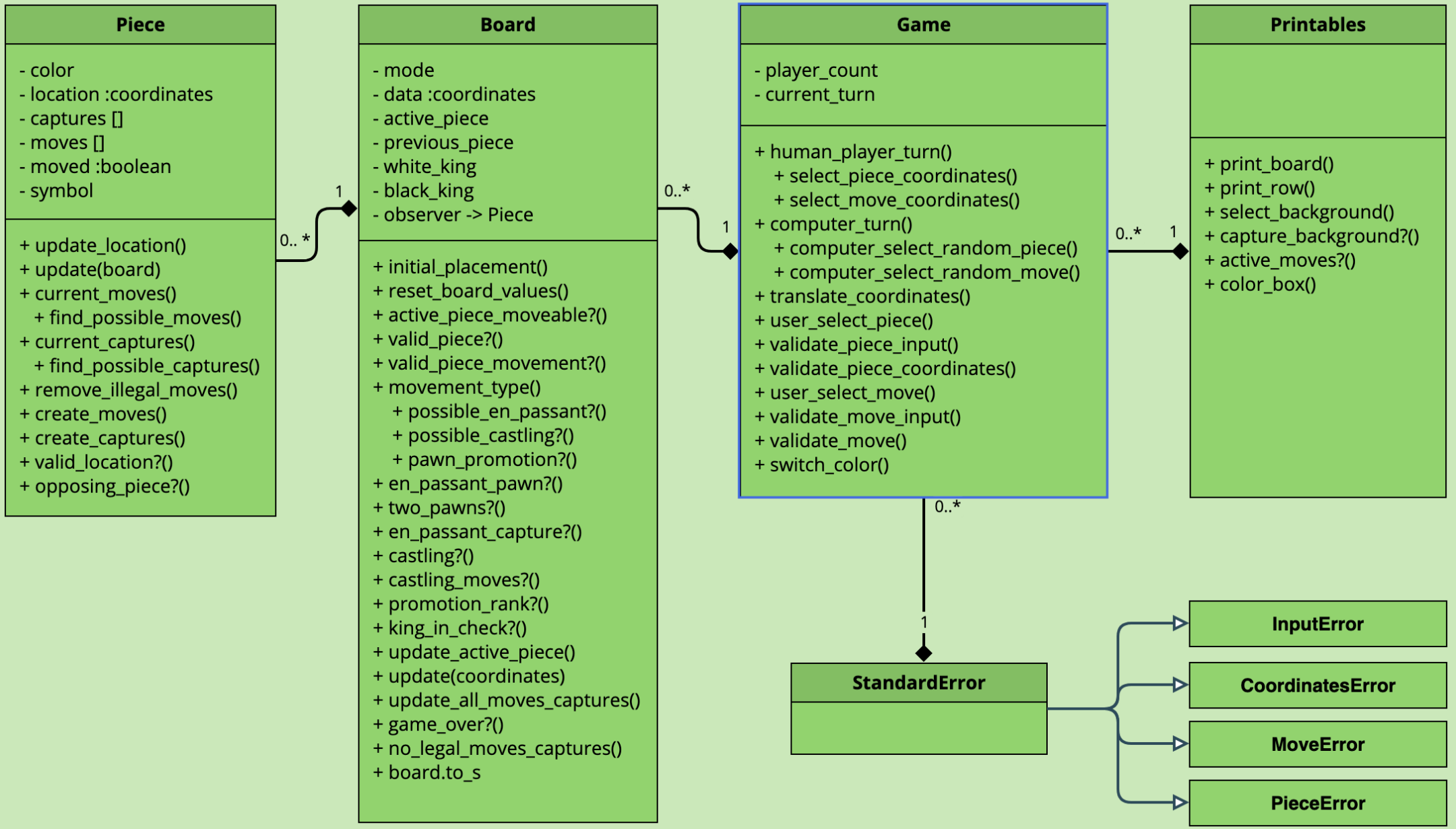
LEVEL UP!

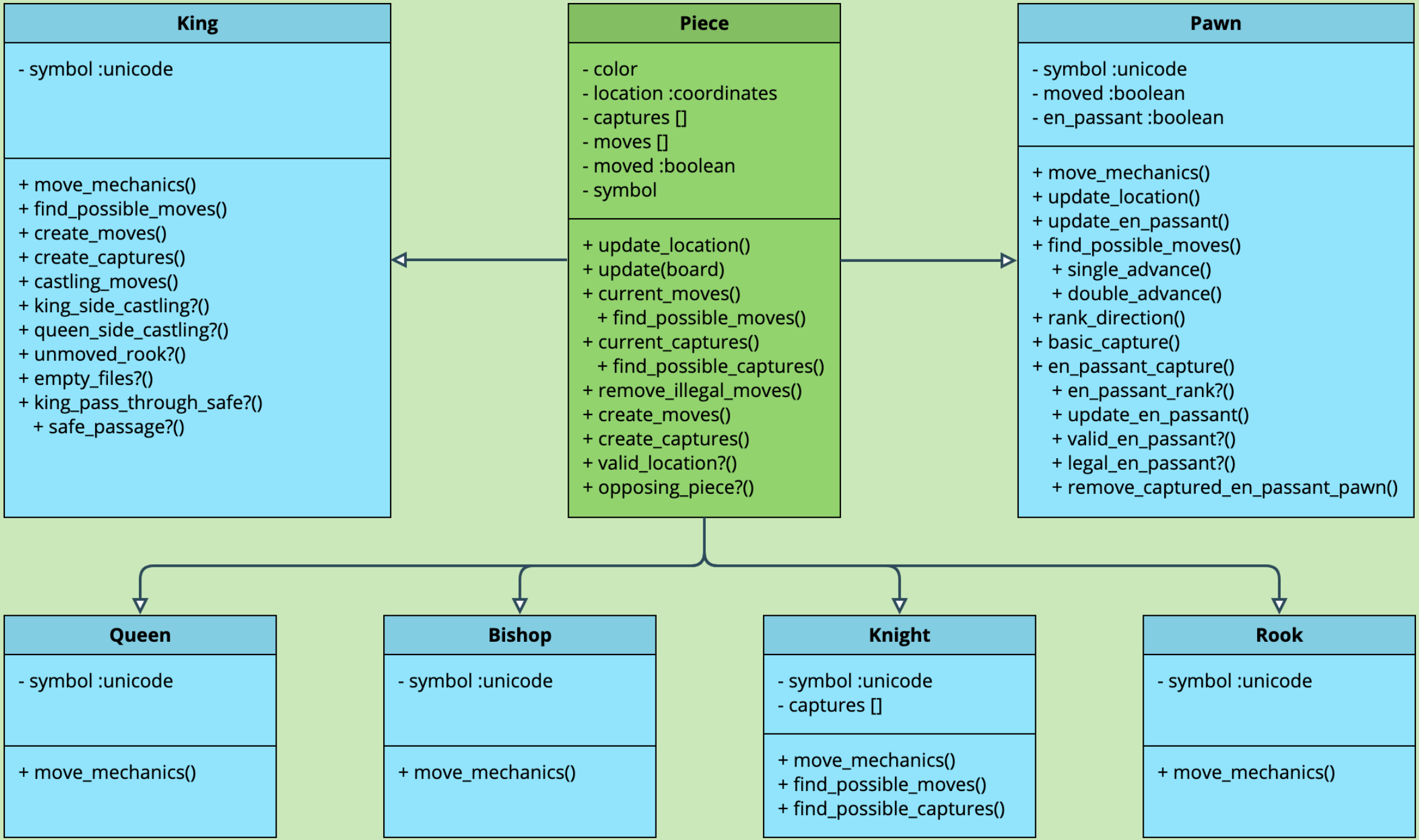
STRUCTURE OF APP

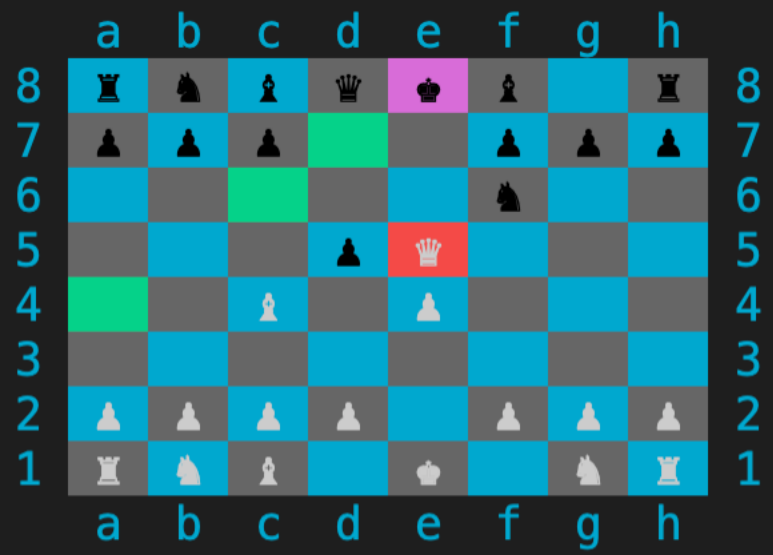
Terminal Chess Control Flow



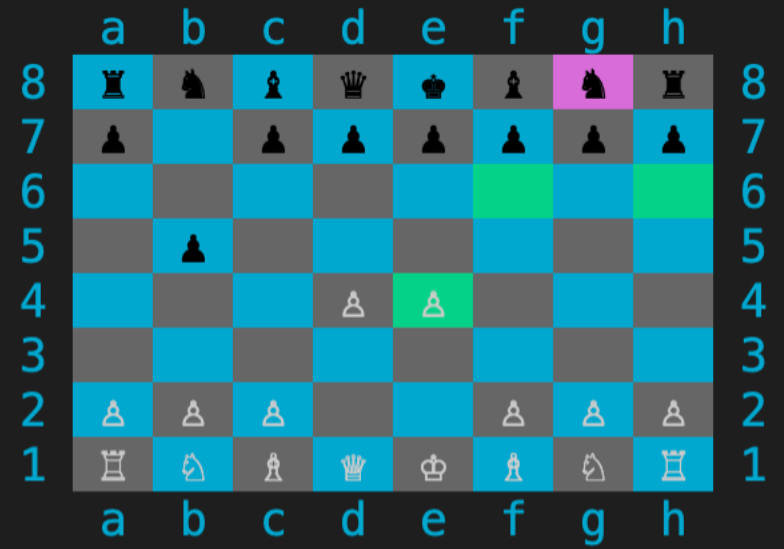
UML Class Diagram




King
- symbol :unicode
+ move_mechanics()
+ find_possible_moves()
+ create_moves()
+ create_captures()
+ castling_moves()
+ king_side_castling?()
+ queen_side_castling?()
+ unmoved_rook?()
+ empty_files?()
+ king_pass_through_safe?()
+ safe_passage?()
Piece
- color
- location :coordinates
- captures []
- moves []
- moved :boolean
- symbol
+ update_location()
+ update(board)
+ current_moves()
+ find_possible_moves()
+ current_captures()
+ find_possible_captures()
+ remove_illegal_moves()
+ create_moves()
+ create_captures()
+ valid_location?()
+ opposing_piece?()
Pawn
- symbol :unicode
- moved :boolean
- en_passant :boolean
+ move_mechanics()
+ update_location()
+ update_en_passant()
+ find_possible_moves()
+ single_advance()
+ double_advance()
+ rank_direction()
+ basic_capture()
+ en_passant_capture()
+ en_passant_rank?()
+ update_en_passant()
+ valid_en_passant?()
+ legal_en_passant?()
+ remove_captured_en_passant_pawn()
Queen
- symbol :unicode
+ move_mechanics()
Bishop
- symbol :unicode
+ move_mechanics()
Knight
- symbol :unicode
- captures []
+ move_mechanics()
+ find_possible_moves()
+ find_possible_captures()
Rook
- symbol :unicode
+ move_mechanics()

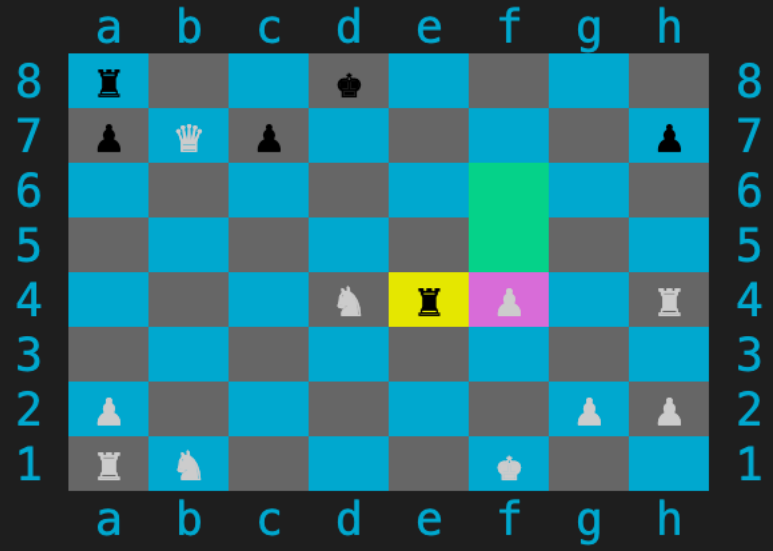


King moves like Queen

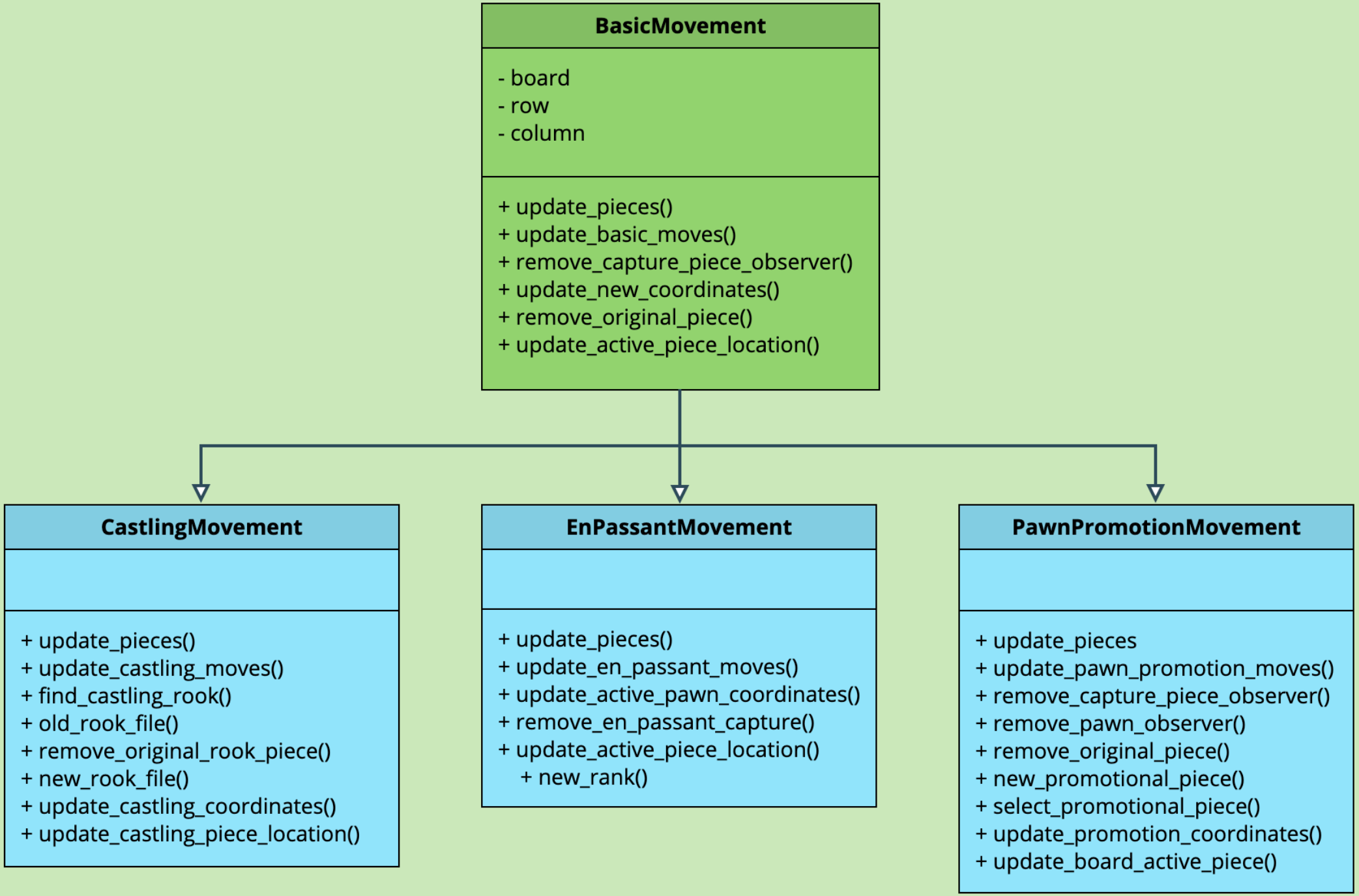


Knight move mechanics also have to be explicitly defined

Please enter coordinates for a legal move highlighted ■ or capture ■.



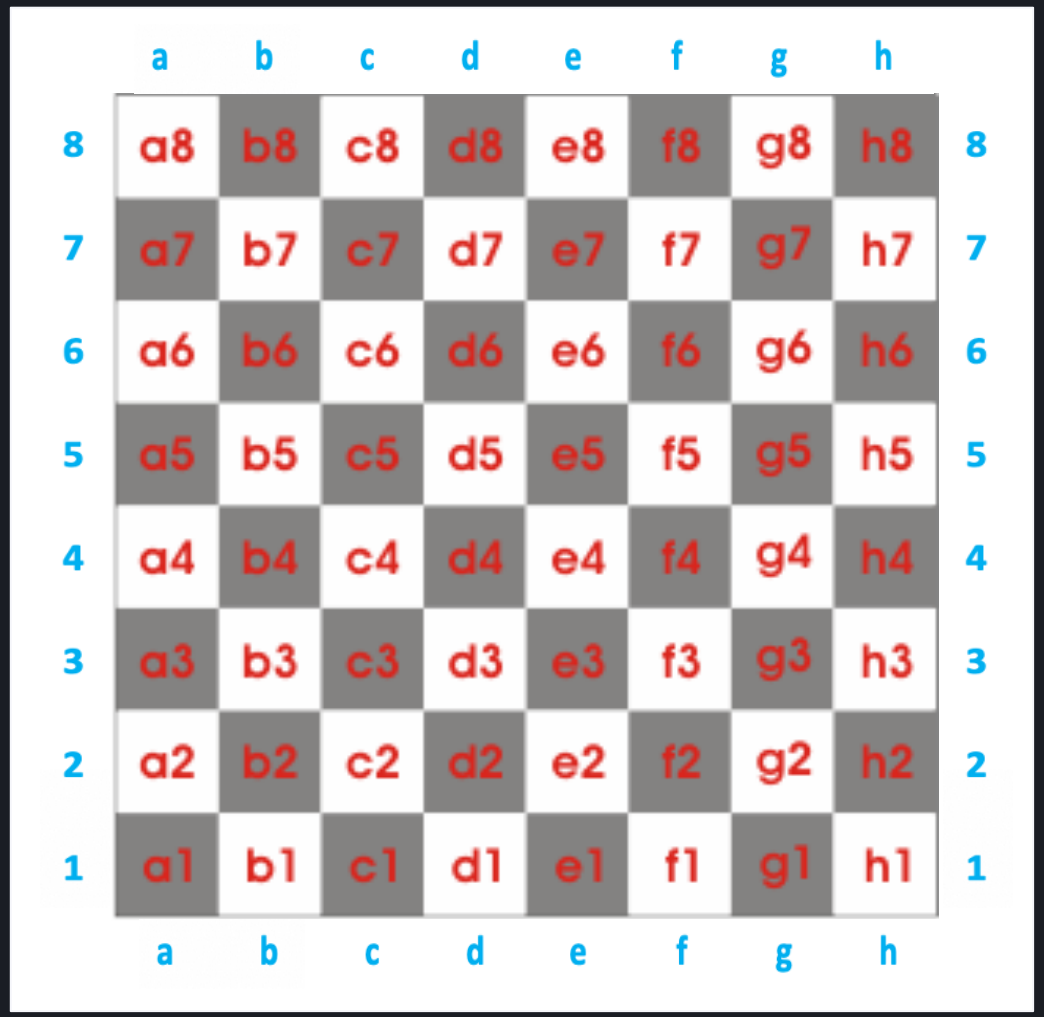
Pawn can still double advance


BasicMovement
- board
- row
- column
+ update_pieces()
+ update_basic_moves()
+ remove_capture_piece_observer()
+ update_new_coordinates()
+ remove_original_piece()
+ update_active_piece_location()
CastlingMovement
+ update_pieces()
+ update_castling_moves()
+ find_castling_rook()
+ old_rook_file()
+ remove_original_rook_piece()
+ new_rook_file()
+ update_castling_coordinates()
+ update_castling_piece_location()
EnPassantMovement
+ update_pieces()
+ update_en_passant_moves()
+ update_active_pawn_coordinates()
+ remove_en_passant_capture()
+ update_active_piece_location()
+ new_rank()
PawnPromotionMovement
+ update_pieces
+ update_pawn_promotion_moves()
+ remove_capture_piece_observer()
+ remove_pawn_observer()
+ remove_original_piece()
+ new_promotional_piece()
+ select_promotional_piece()
+ update_promotion_coordinates()
+ update_board_active_piece()

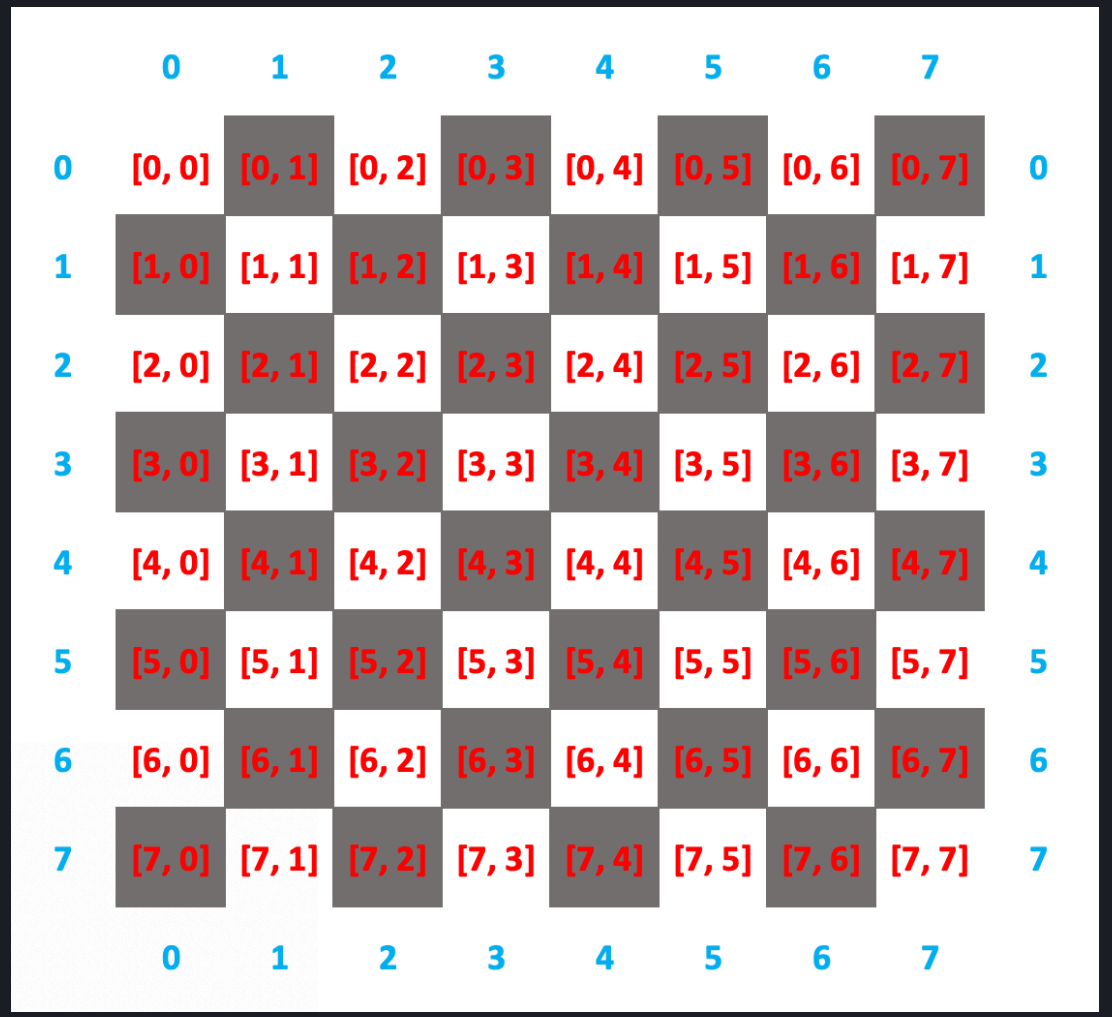
The Build Process

- Chess Notation Conversion
- Move Mechanics
- Extrapolation
- Validation
- Save/Load
- Special Movement

Translating Chess Notation



Chess Notation



Coordinate Notation

Translating Chess Notation

Convert Chess Notation into corresponding coordinates

- Split into individual characters eg. “c2“
  coordinates = letter_number.split(//) => [“b“, “2“]

- Convert file “e“ to it‘s counterpart column value => 2

- Translate the rank into the correct row coordinate => 6

- e2 => [2, 6] [:column, :row]

row = coordinates[1]
column = coordinates[0]

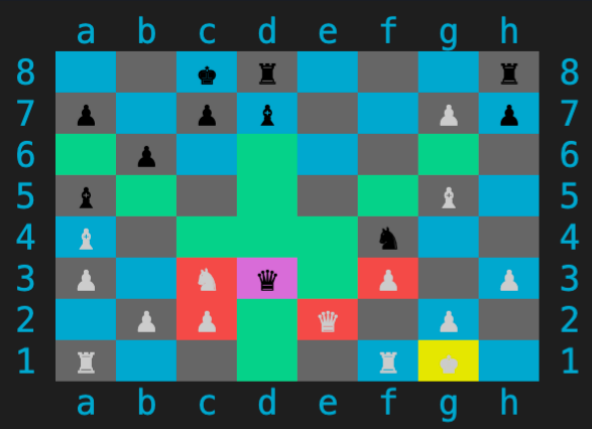
After we convert from Chess Notation to coordinate form, generally, we’re used to seeing it [row, column] format.

Translating Chess Notation

LEVEL UP!

```ruby
  # returns the coordinates as a hash from the user's chess notation input:
  # ie. d2  letter(column) and number(row)
  def translate_notation(letter_number)
    coordinates = letter_number.split(//) # splits into each character
    translate_row(coordinates[1])
    translate_column(coordinates[0])
    { row: @row, column: @column }
  end

  private

  # converts file (a-h) to a number 0-7 using ASCII numbers for characters
  def translate_column(letter)
    @column = letter.downcase.ord - 97
  end

  # converts the rank (8-1) to a number 0-7
  def translate_row(number)
    @row = 8 - number.to_i
  end
end
```

```
~/Coding/terminal-chess main >16 !15 ?1 irb
@irb(main):001:0> "a".ord - 97
=> 0
@irb(main):002:0> "b".ord - 97
=> 1
@irb(main):003:0> "c".ord - 97
=> 2
@irb(main):004:0> "d".ord - 97
=> 3
@irb(main):005:0>
```



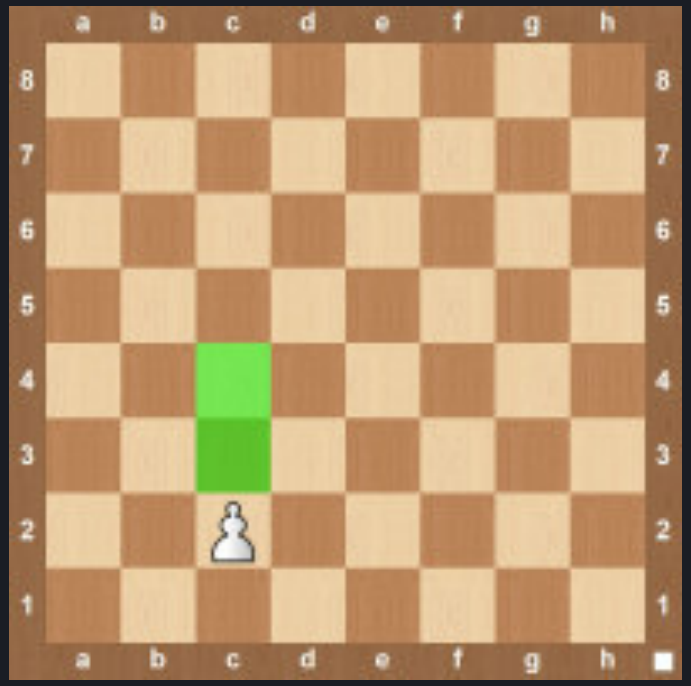
Square Background Color Selection

LEVEL UP!

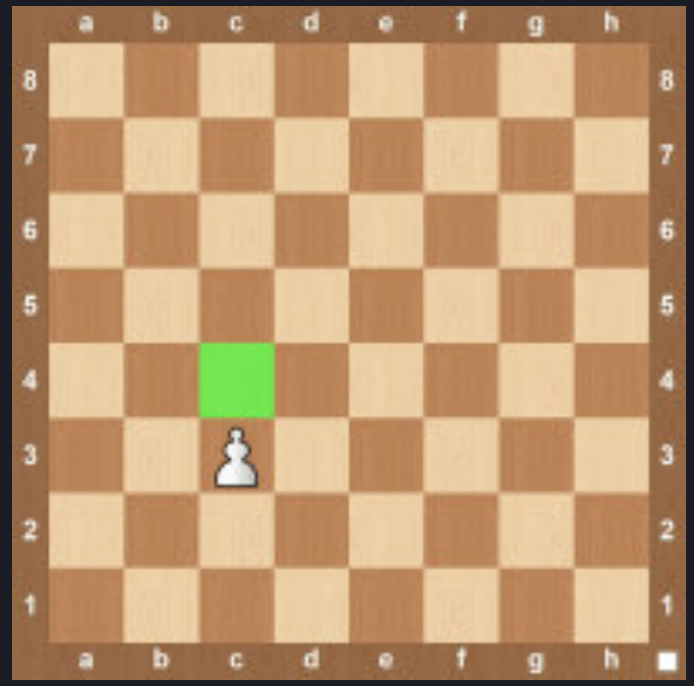
```ruby
# returns background color based on specific conditions:
# 105 = magenta background (active piece to move)
# 101 = red background     (possible captures)
# 102 = green background   (possible moves)
#  43 = yellow background  (previous piece that moved)
#  46 = cyan background    (even)
# 100 = gray background    (odd)
def select_background(row_index, column_index)
  if @active_piece&.location == [row_index, column_index]
    105
  elsif capture_background?(row_index, column_index)
    101
  elsif @previous_piece&.location == [row_index, column_index]
    43
  elsif active_moves?(row_index, column_index)
    102
  elsif (row_index + column_index).even?
    46
  else
    100
  end
end
```

```ruby
# determines if the selected piece has any possible captures, returns the coordinates
def capture_background?(row, column)
  @active_piece&.captures&.any?([row, column]) && @data[row][column]
end

# determines possible legal moves for selected piece and returns the coordinates
def active_moves?(row, column)
  @active_piece&.moves&.any?([row, column])
end

# sets the font colours for each square based on specific conditions
# 107 = white (chess pieces)
#  30 = black (chess pieces)
def print_box(_row_index, _column_index, box, background)
  if box
    text_color = box.color == :white ? 107 : 30
    color_box(text_color, background, box.symbol)
  else
    color_box(30, background, '  ')
  end
end

# prints the final box/square with the specified text color, background/highlight color,
# and string (in this case, ascii chess symbol)
def color_box(text_color, background, string)
  print "\e[#{text_color};#{background}m#{string}\e[0m"
end
end
```

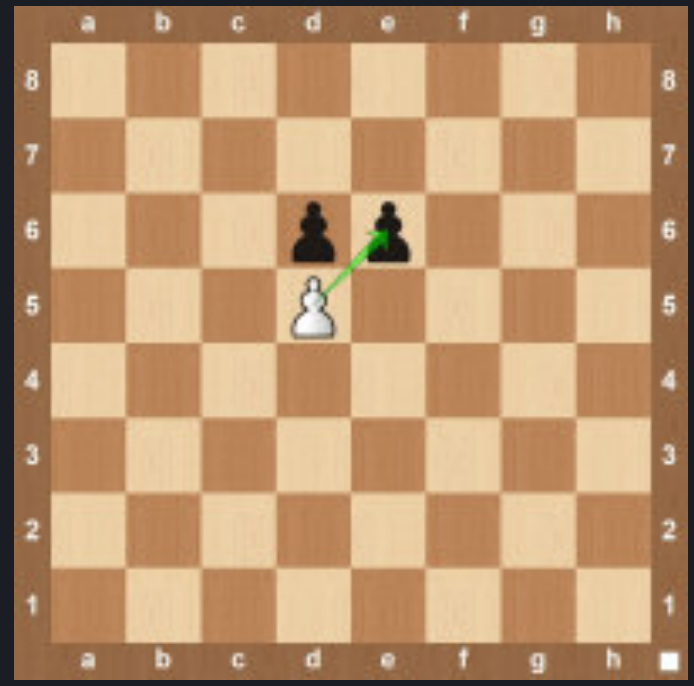
Pawn Move Mechanics



Double Advance
(first turn only)



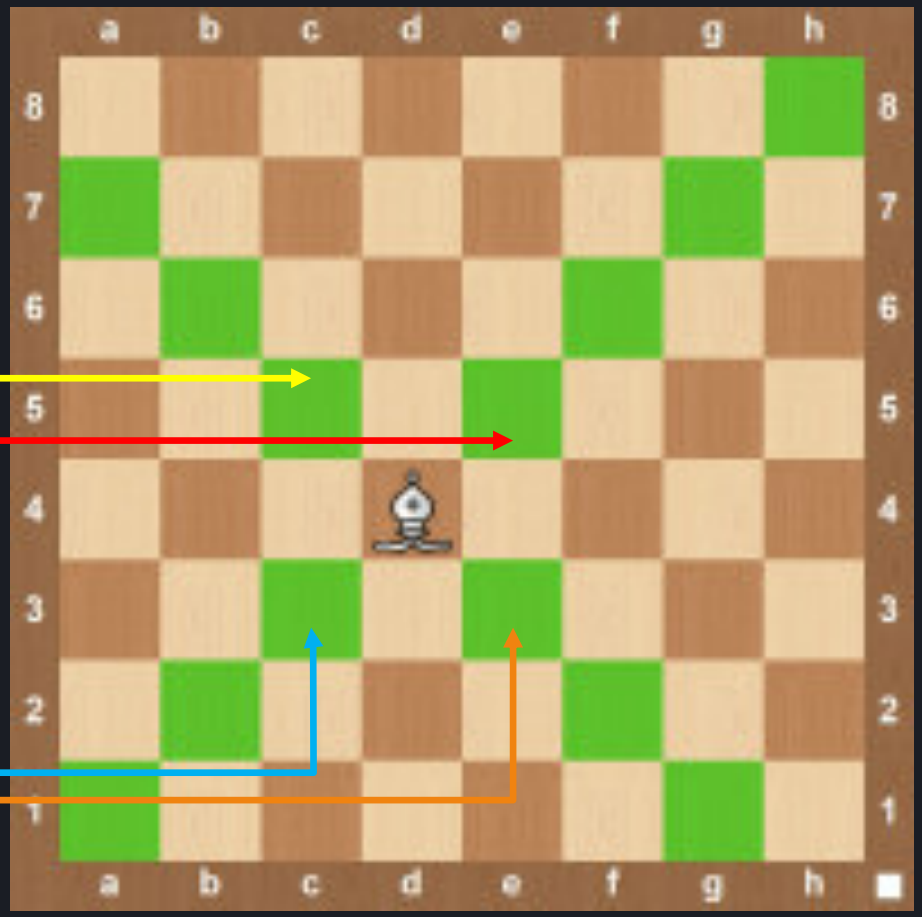
Single Advance
(Standard)



Captures
(on diagonal
forward only)

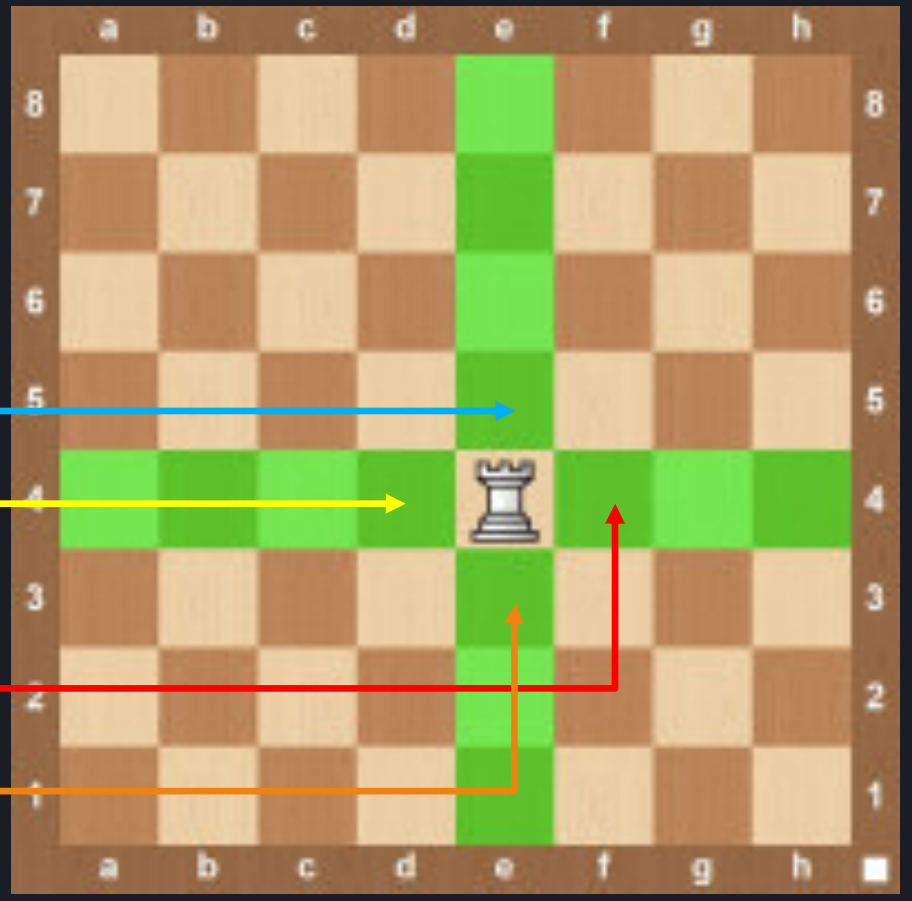
Bishop Move Mechanics

```ruby
# Move mechanics for bishop
class Bishop < Piece
  def initialize(board, attributes)
    super(board, attributes)
    @symbol = " \u265D "
  end

  private

  def move_mechanics
    [[1, 1], [1, -1], [-1, 1], [-1, -1]]
  end
end
```

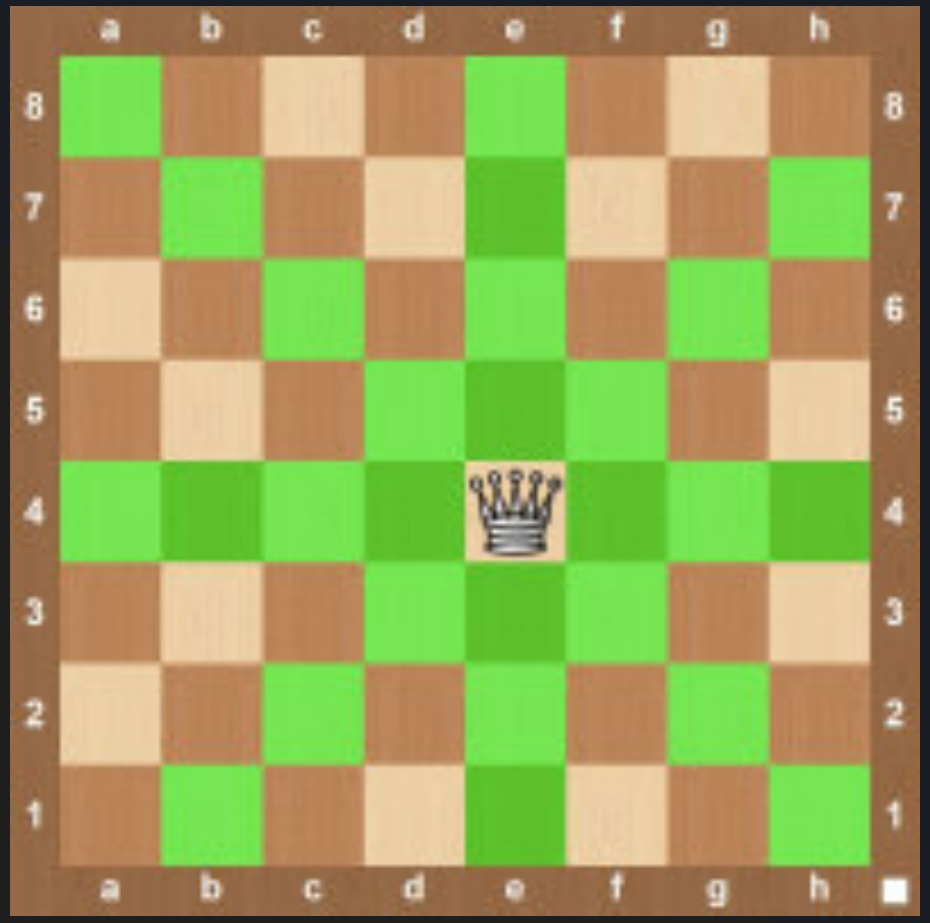


Rook Move Mechanics

```ruby
# move mechanics for rook
class Rook < Piece
  def initialize(board, attributes)
    super(board, attributes)
    @symbol = " \u265C "
  end

  private

  def move_mechanics
    [[1, 0], [-1, 0], [0, 1], [0, -1]]
  end
end
```

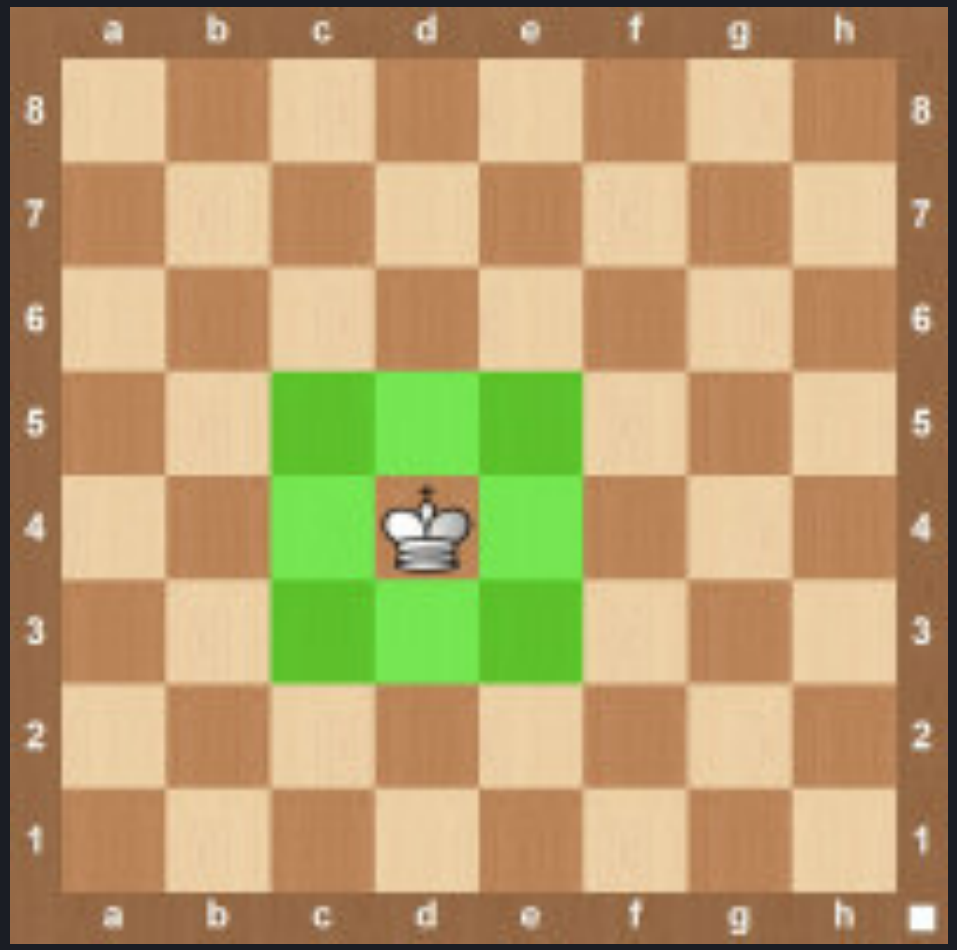


Queen Move Mechanics




```ruby
# move mechanics for queen
class Queen < Piece
  def initialize(board, attributes)
    super(board, attributes)
    @symbol = " \u265B "
  end

  private

  def move_mechanics
    [[1, 1], [1, 0], [1, -1], [0, 1], [0, -1], [-1, 1], [-1, 0], [-1, -1]]
  end
end
```

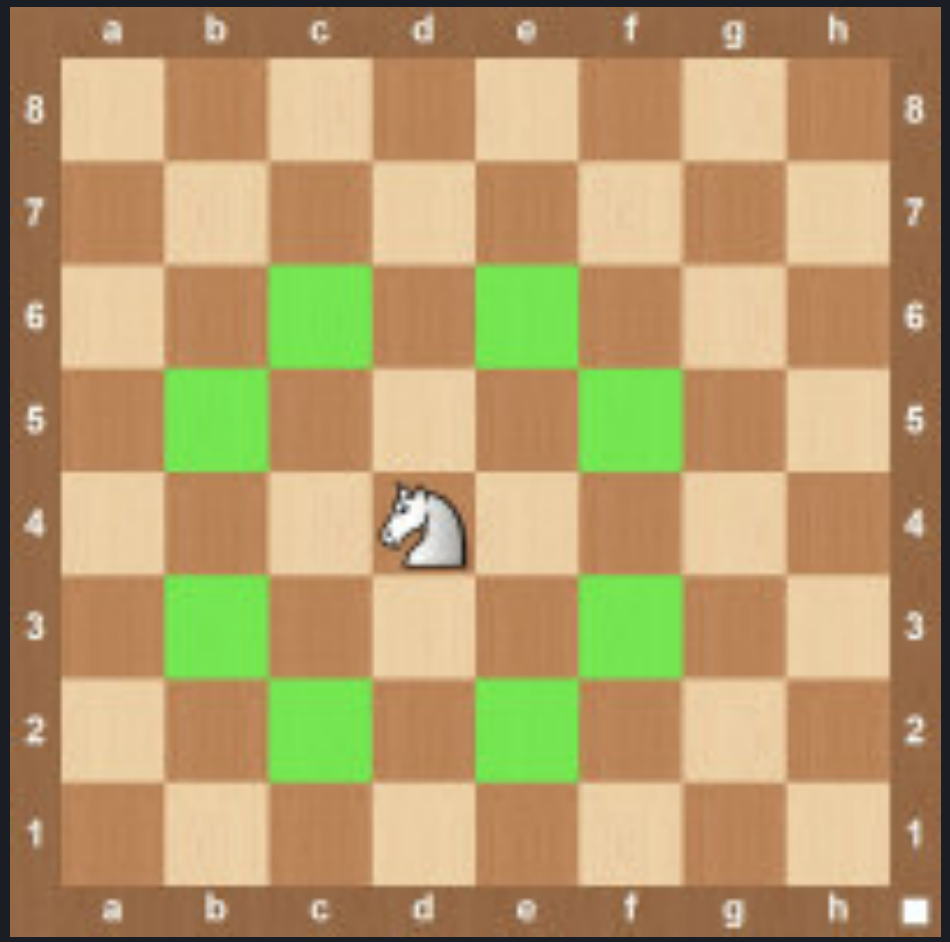
King Move Mechanics




```ruby
def move_mechanics
  [[1, 1], [1, 0], [1, -1], [0, 1], [0, -1], [-1, 1], [-1, 0], [-1, -1]]
end
```

Knight Move Mechanics

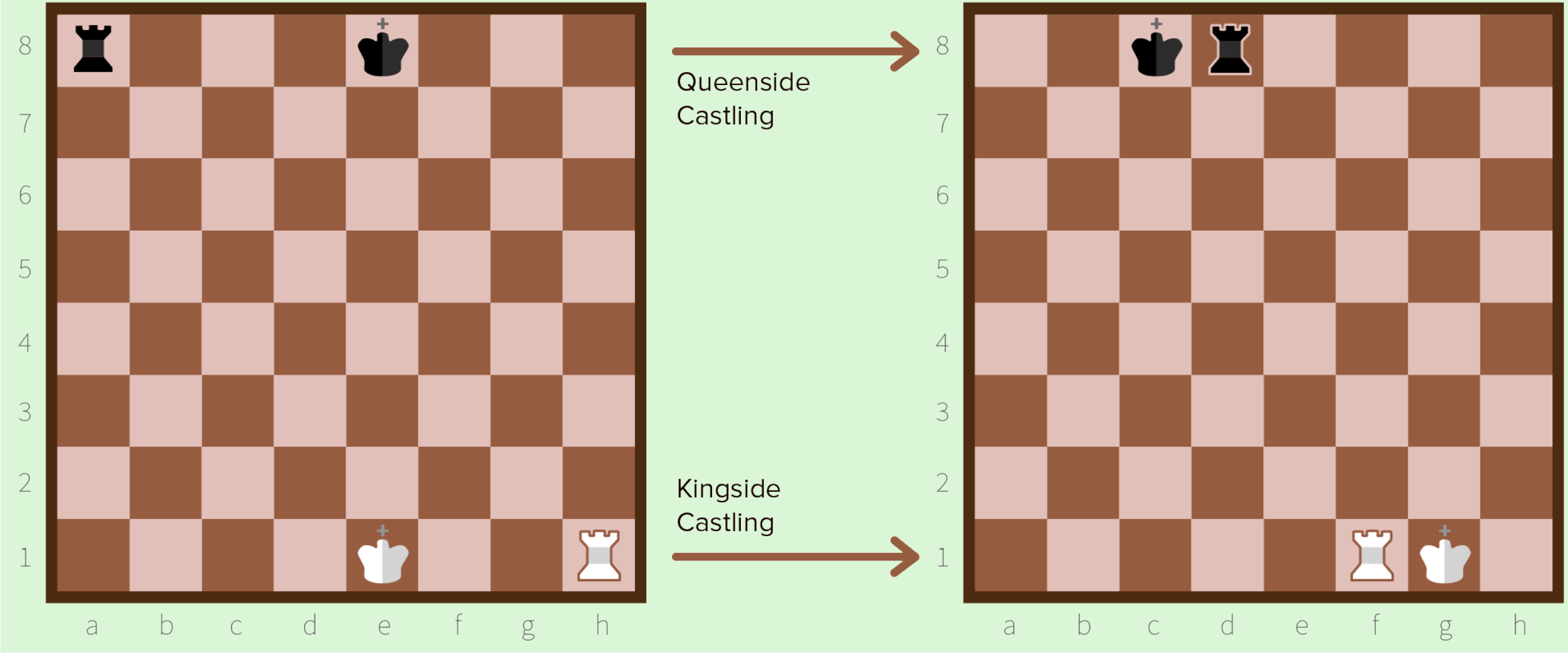
```ruby
# move mechanics for knight
class Knight < Piece
  def initialize(board, attributes)
    super(board, attributes)
    @symbol = " \u265E "
    @captures = []
  end

  # iterates over move mechanics and adds valid location to possible_moves
  def find_possible_moves(board)
    possible_moves = []
    move_mechanics.each do |move|
      rank = @location[0] + move[0]
      file = @location[1] + move[1]
      next unless valid_location?(rank, file)

      possible_moves << [rank, file] unless board.data[rank][file]
    end
    possible_moves
  end

  # iterates over move mechanics and adds location as a capture if it's a
  # valid move and also opponent's piece
  def find_possible_captures(board)
    possible_captures = []
    move_mechanics.each do |move|
      rank = @location[0] + move[0]
      file = @location[1] + move[1]
      next unless valid_location?(rank, file)

      possible_captures << [rank, file] if opposing_piece?(rank, file, board.data)
    end
    @captures = possible_captures
  end

  private

  def move_mechanics
    [[2, 1], [2, -1], [-2, 1], [-2, -1], [1, 2], [1, -2], [-1, 2], [-1, -2]]
  end
end
```



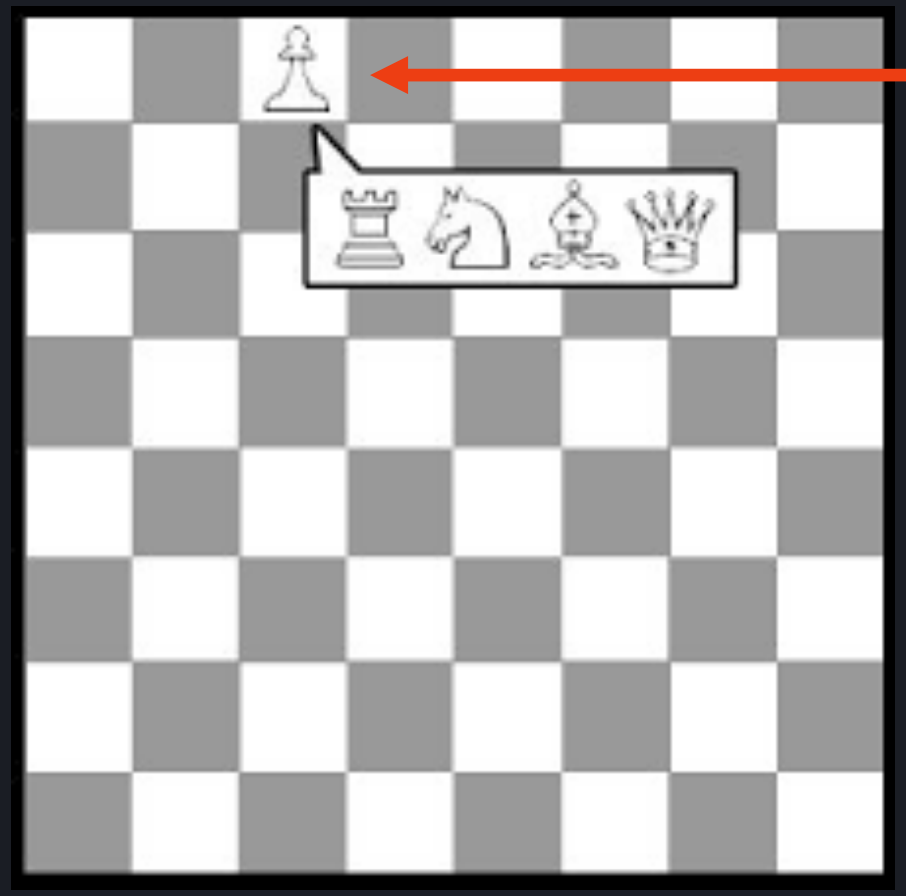
L Shape Moveset

Special Move Mechanics

Castling

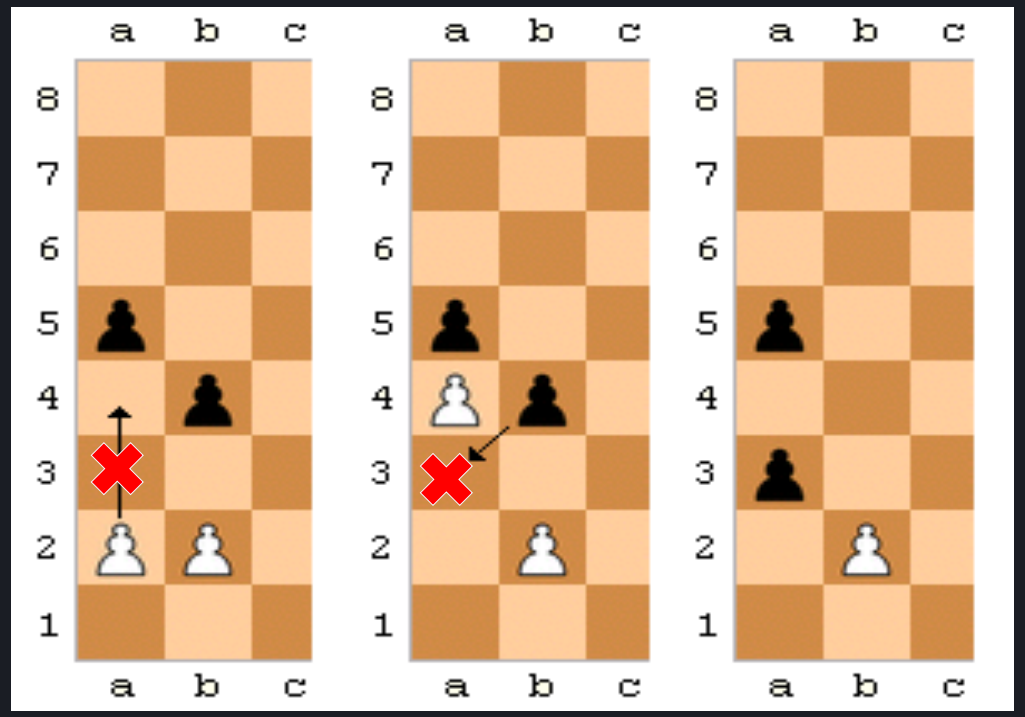


Pawn Promotion



Pawns can be promoted to Rook, Knight, Bishop or Queen when it reaches the 8th Rank

En Passant



En Passant is the French word for in passing. En Passant is a legal move to take an 'in passing' pawn. If your pawn advances 2 squares, crossing the attacking square of the opponent's pawn, your opponent has the legal right to capture your pawn. If the opponent moves another piece instead of taking your pawn, the opponent will no longer be allowed to capture your passing pawn in their next move.

Extrapolation

Will keep extrapolating piece moveset while it's within board, and push coordinates to result []. Will break out of loop when a piece is reached

Creates a temporary board and moves the piece. Checks if King is safe (more in Move Validation)

Returns array of possible moves within board

Extrapolates moveset to the end of the board or until it hits a piece

Only returns coordinates if opponent's piece

Used inject to refactor implementation of move mechanics into logic for capture

```ruby
  def current_moves(board)
    possible_moves = find_possible_moves(board)
    @moves = remove_illegal_moves(board, possible_moves)
  end

  def current_captures(board)
    possible_captures = find_possible_captures(board)
    @captures = remove_illegal_moves(board, possible_captures)
  end

  def find_possible_moves(board)
    moves = move_mechanics.inject([]) do |memo, move|
      memo << create_moves(board.data, move[0], move[1])
    end
    moves.compact.flatten(1)
  end

  def find_possible_captures(board)
    captures = move_mechanics.inject([]) do |memo, move|
      memo << create_captures(board.data, move[0], move[1])
    end
    captures.compact
  end
```

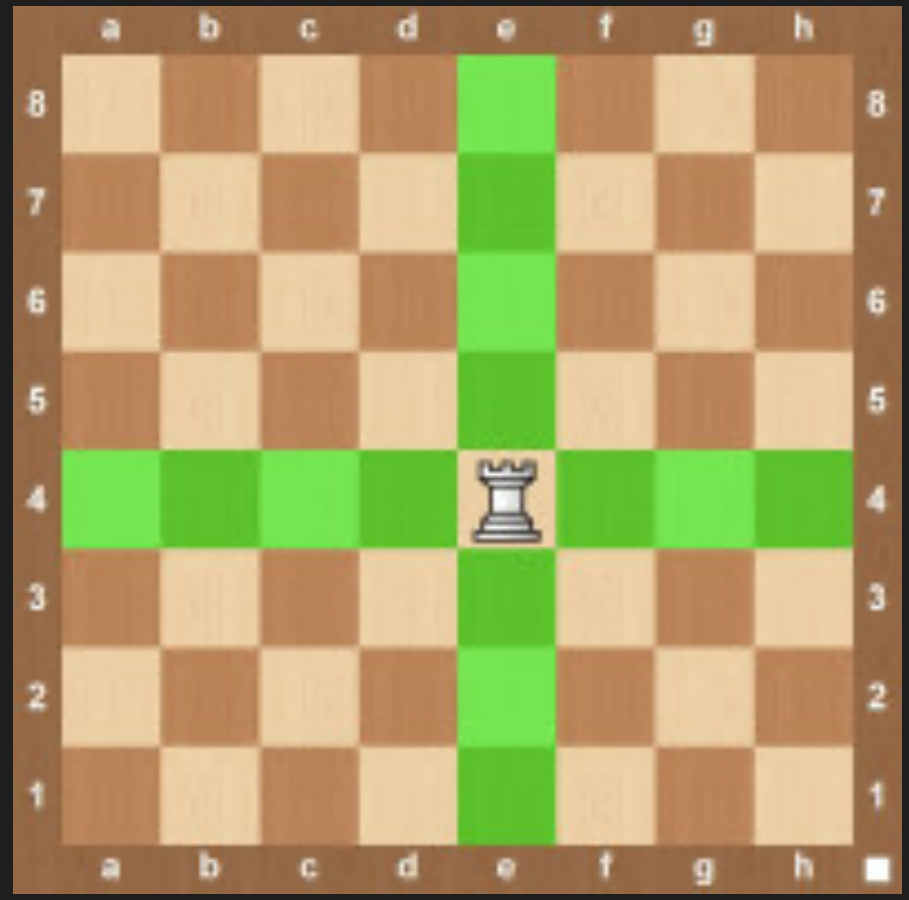
```ruby
  # adds moves until it reaches a piece or still within the board,
  # based on each piece's move mechanics
  def create_moves(data, rank_change, file_change)
    rank = @location[0] + rank_change
    file = @location[1] + file_change
    result = []
    while valid_location?(rank, file)
      break if data[rank][file]

      result << [rank, file]
      rank += rank_change
      file += file_change
    end
    result
  end

  # adds capture if piece's move set reaches opponent's piece,
  # dependent on the piece's move mechanics.
  def create_captures(data, rank_change, file_change)
    rank = @location[0] + rank_change
    file = @location[1] + file_change
    while valid_location?(rank, file)
      break if data[rank][file]

      rank += rank_change
      file += file_change
    end
    [rank, file] if opposing_piece?(rank, file, data)
  end

  # ensures pieces don't go outside the board, only locations
  # inside board are valid.
  def valid_location?(rank, file)
    rank.between?(0, 7) && file.between?(0, 7)
  end

  # checks if piece is opposing/not the same color
  def opposing_piece?(rank, file, data)
    return unless valid_location?(rank, file)

    piece = data[rank][file]
    piece && piece.color != color
  end
end
```

Extrapolation

```
14    def move_mechanics
15      [[1, 0], [-1, 0], [0, 1], [0, -1]]
16    end
17  end
```

```
40    def find_possible_moves(board)
41      moves = move_mechanics.inject([]) do |memo, move|
42        memo << create_moves(board.data, move[0], move[1])
43      end
44      moves.compact.flatten(1)
45    end
```

```
64    # adds moves until it reaches a piece or still within the board,
65    # based on each piece's move mechanics
66    def create_moves(data, rank_change, file_change)
67      rank = @location[0] + rank_change
68      file = @location[1] + file_change
69      result = []
70      while valid_location?(rank, file)
71        break if data[rank][file]
72
73        result << [rank, file]
74        rank += rank_change
75        file += file_change
76      end
77      result
78    end
```



Move Validation

```ruby
def remove_illegal_moves(board, moves)
  return moves unless moves.size.positive?

  temp_board = Marshal.load(Marshal.dump(board))
  validator = MoveValidator.new(location, temp_board, moves)
  validator.verify_possible_moves
end
```

Creates a duplicate temporary board and runs through all the piece's possible moves and captures and see if it's a legal move

```ruby
def verify_possible_moves
    @king_location = find_king_location
    @board.data[@current_locoation[0]][@current_locoation[1]] = nil
    @move_list.select do |move|
        legal_move?(move)
    end
end

private

# finds location of King to determine if selected piece can move or not based on
# King's location and if King will be safe based on the user's desired move,
# if it's not, will raise MoveError message alert
def find_king_location
    return if @current_piece.symbol == " \u265A "

    if @current_piece.color == :black
        @board.black_king.location
    else
        @board.white_king.location
    end
end
```

Goes through all the moves and determines if it is a legal move based on the King's safety. Does so by removing the piece from it's current position

Move Validation

```ruby
# moves board/pieces around to the possible move and checks if the King is safe
def legal_move?(move)
    captured_piece = @board.data[move[0]][move[1]]
    move_current_piece(move)
    king = @king_location || move
    result = safe_king?(king)
    @board.data[move[0]][move[1]] = captured_piece
    result
end

def move_current_piece(move)
    @board.data[move[0]][move[1]] = @current_piece
    @current_piece.update_location(move[0], move[1])
end

# determines if the King is safe (not under check, returns false).
# If King's location matches any possible captures for any of opponent's pieces.
def safe_king?(kings_location)
    pieces = @board.data.flatten(1).compact
    pieces.none? do |piece|
        next unless piece.color != @current_piece.color

        captures = piece.find_possible_captures(@board)
        captures.include?(kings_location)
    end
end
```

Checks to see if it is a legal move, captures the piece if it's a possible capture in the piece's moveset. The King's new location is either the King's current location or if the King is making the move, the King's move.

Moves the location of the piece to captured piece.

This checks to see if the king is safe:
Only tests if piece is opponent's piece for whoever's turn it is. Finds all possible captures, and if any of those matches the location of the King for the current player's turn, it is under check => returns true

SAVE LOAD

Originally was going to use YAML, which is essentially like JSON, but much simpler. It's essentially the css sass equivalent.

However, I came across an even simpler method, using Ruby's in-built Marshal Module.

Serializes all the data using the Marshal.dump method => transforms data structure into a binary string, which is then written to a file.

Deserializes and reconstitutes the data using Marshal.load method.

SAVE LOAD

```ruby
module Serializer
  def save_game
    Dir.mkdir 'saved_games' unless Dir.exist? 'saved_games'
    filename = create_filename
    File.open("saved_games/#{filename}", 'w+') do |file|
      Marshal.dump(self, file)
    end
    puts "Game was saved as \e[96m#{filename}\e[0m\n"
    sleep(2)
    play # user can continue playing game after saving
  rescue SystemCallError => e
    puts "\e[91mError while writing to file #{filename}.\e[0m"
    puts e
  end

  def create_filename
    date = Time.now.strftime('%Y-%m-%d').to_s
    time = Time.now.strftime('%H-%M-%S').to_s
    "Chess #{date} at #{time}"
  end
```

To save a game, we first make a directory, saved_games, unless it already exists.

Then we create a filename with the following format: Chess YYYY-MM-DD and HH:MM:SS.

Open up the newly created file and writes/serializes all the object data to the file using Marshal.dump.

SAVE LOAD

```ruby
def load_game
  file_name = find_saved_file
  File.open("saved_games/#{file_name}") do |file|
    Marshal.load(file)
  end
rescue IOError => e
  puts "\e[91mError while loading file #{file_name}.\e[0m"
  puts e
end

def find_saved_file
  saved_games = create_game_list
  if saved_games.empty?
    puts 'There are no saved games to play yet!'
    main_menu
  else
    print_saved_games(saved_games)
    file_number = select_saved_game(saved_games.size)
    saved_games[file_number.to_i - 1]
  end
end

def create_game_list
  game_list = []
  return game_list unless Dir.exist? 'saved_games'

  Dir.entries('saved_games').each do |name|
    game_list << name if name.match(/(Chess)/)
  end
  game_list
end
```

To load a saved game, open it and deserialize the data using Ruby's in-built Marshal.load method.

To select which game to load, we first have to create an array to display the game files to load"

game_list[ ]

If there are no saved games, then it will output, "There are no saved games to play yet!", otherwise, it will print to the screen a list of the saved games.

In saved_games directory, if any filenames contain Chess in their name, it pushes that file to the game_list array.

SAVE LOAD

```ruby
57    def print_saved_games(game_list)
58      puts "\e[96m[#]\e[0m File Name(s)"
59      game_list.each_with_index do |name, index|
60        puts "\e[96m[#{index + 1}]\e[0m #{name}"
61      end
62    end
63
64    def select_saved_game(number)
65      file_number = gets.strip
66      return file_number if file_number.to_i.between?(1, number)
67
68      puts "\e[91mInput Error!\e[0m Please enter a valid file number."
69      select_saved_game(number)
70    end
71  end
```

Prints out each saved game with it's index value (+1) and it's filename.

User is prompted to select a game to load based on its file number (index), input is converted into an integer, and only returns the input for the filenumber if it's between 1 and save_games.size.

If it is outside of this range, then prompts user to input a valid file number and calls the method again, so loops until it returns a number that is in the correct range of number of files that exist.

The Testing Process

- TDD
- Early bugs
- RSpec

TDD Process

Majority of TDD process was Manual Testing as hard to write test cases to test specific board conditions with so many pieces etc.

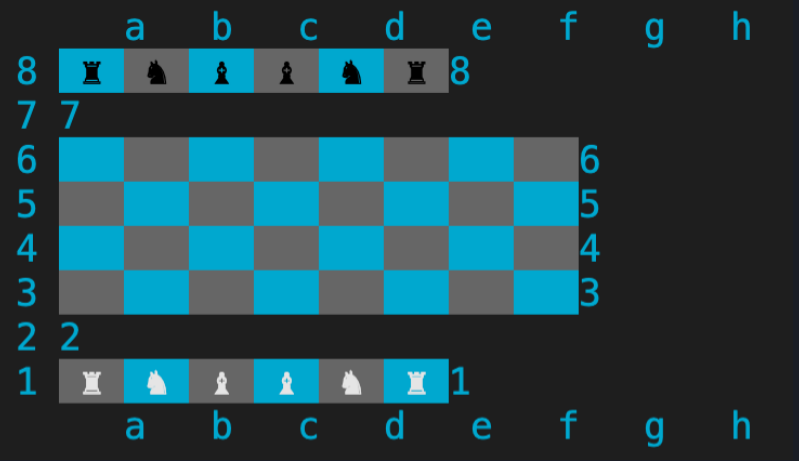
Did RSpec tests to check if:

- Chess Notation was correctly converted into the right coordinates.

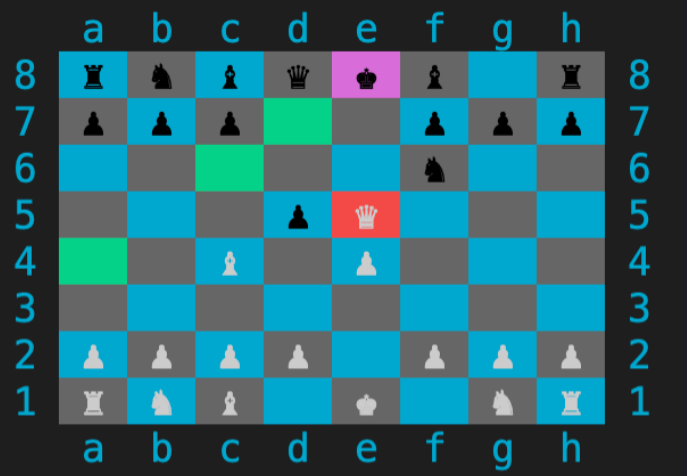
- Board initialized correct with pieces all in the correct positions.

- Save and Load functionality works as expected.

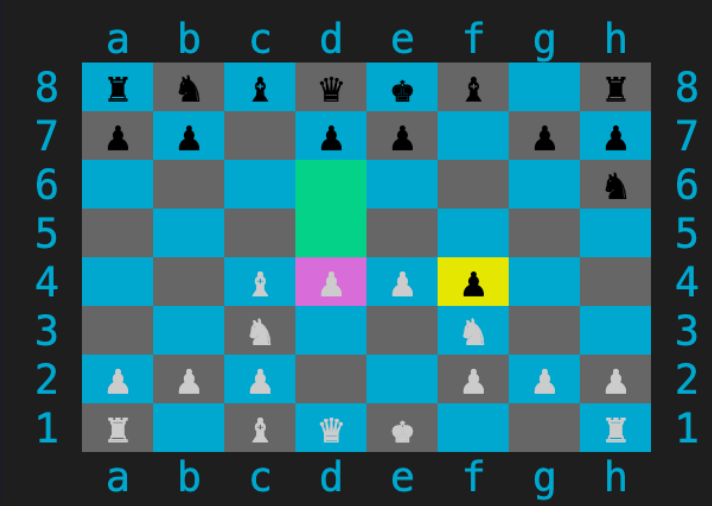
Kings, Queens, and Pawns not initializing.



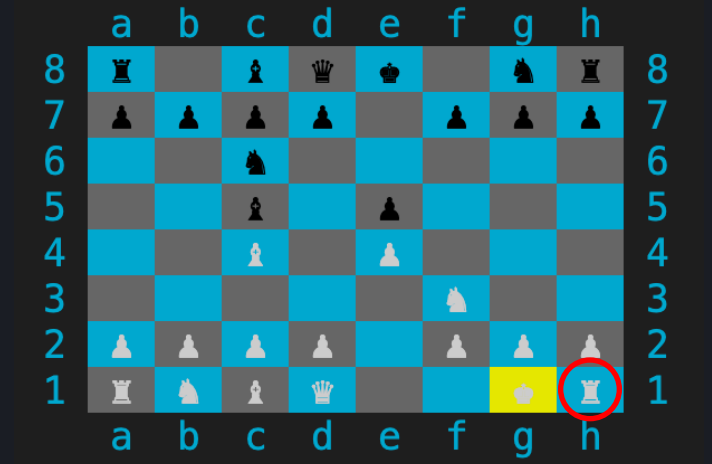
King behaving like a Queen. Needed to explicitly define move mechanics so it doesn't get extrapolated like for Bishops, Rooks and Queen.
(Inheritance from parent Piece Class)



Pawns still had double advance available after first move



Castling, King could move 2 spaces, but Rook failed to castle.



RSpec Testing

LEVEL UP!

Converting Chess Notation into Grid values:
CORE FUNCTIONALITY

```
~/Coding/terminal-chess/spec main >1 !1 ?1 rspec notation_converter_spec.rb -f d

NotationConverter
  #translate_notation
    when user input is a8
      returns row: 0 column: 0
    when user input is b7
      returns row: 1 column: 1
    when user input is c6
      returns row: 2 column: 2
    when user input is d5
      returns row: 3 column: 3
    when user input is e4
      returns row: 4 column: 4
    when user input is f3
      returns row: 5 column: 5
    when user input is g2
      returns row: 6 column: 6
    when user input is h1
      returns row: 7 column: 7

Finished in 0.00548 seconds (files took 0.12219 seconds to load)
8 examples, 0 failures
```

```ruby
RSpec.describe NotationConverter do
  subject(:translator) { described_class.new }

  describe '#translate_notation' do
    context 'when user input is a8' do
      it 'returns row: 0 column: 0' do
        user_input = 'a8'
        result = translator.translate_notation(user_input)
        expect(result).to eq({ row: 0, column: 0 })
      end
    end

    context 'when user input is b7' do
      it 'returns row: 1 column: 1' do
        user_input = 'b7'
        result = translator.translate_notation(user_input)
        expect(result).to eq({ row: 1, column: 1 })
      end
    end

    context 'when user input is c6' do
      it 'returns row: 2 column: 2' do
        user_input = 'c6'
        result = translator.translate_notation(user_input)
        expect(result).to eq({ row: 2, column: 2 })
      end
    end

    context 'when user input is d5' do
      it 'returns row: 3 column: 3' do
        user_input = 'd5'
        result = translator.translate_notation(user_input)
        expect(result).to eq({ row: 3, column: 3 })
      end
    end

    context 'when user input is e4' do
      it 'returns row: 4 column: 4' do
        user_input = 'e4'
        result = translator.translate_notation(user_input)
        expect(result).to eq({ row: 4, column: 4 })
      end
    end

    context 'when user input is f3' do
      it 'returns row: 5 column: 5' do
        user_input = 'f3'
        result = translator.translate_notation(user_input)
        expect(result).to eq({ row: 5, column: 5 })
      end
    end

    context 'when user input is g2' do
      it 'returns row: 6 column: 6' do
        user_input = 'g2'
        result = translator.translate_notation(user_input)
        expect(result).to eq({ row: 6, column: 6 })
      end
    end

    context 'when user input is h1' do
      it 'returns row: 7 column: 7' do
        user_input = 'h1'
        result = translator.translate_notation(user_input)
        expect(result).to eq({ row: 7, column: 7 })
      end
    end
  end
end
```

RSpec Testing

Correct Starting Board Piece Positioning

```ruby
RSpec.describe Board do
  subject(:board) { described_class.new }

  describe '#initial_placement' do
    before do
      board.initial_placement
    end

    it 'has bottom row (1st Rank) of white pieces' do
      expect(board.data[7].all? { |piece| piece.color == :white }).to be true
    end

    it 'has sixth row (2nd Rank) of white pieces' do
      expect(board.data[6].all? { |piece| piece.color == :white }).to be true
    end

    it 'has second row (7th Rank) of black pieces' do
      expect(board.data[1].all? { |piece| piece.color == :black }).to be true
    end

    it 'has top row (8th Rank) of black pieces' do
      expect(board.data[0].all? { |piece| piece.color == :black }).to be true
    end

    it 'has 1st Rank queen side Rook in correct position' do
      expect(board.data[7][0].instance_of?(Rook)).to be true
    end

    it 'has 1st Rank queen side Knight in correct position' do
      expect(board.data[7][1].instance_of?(Knight)).to be true
    end

    it 'has 1st Rank queen side Bishop in correct position' do
      expect(board.data[7][2].instance_of?(Bishop)).to be true
    end

    it 'has 1st Rank Queen in correct position' do
      expect(board.data[7][3].instance_of?(Queen)).to be true
    end

    it 'has 1st Rank King in correct position' do
      expect(board.data[7][4].instance_of?(King)).to be true
    end

    it 'has 1st Rank king side Bishop in correct position' do
      expect(board.data[7][5].instance_of?(Bishop)).to be true
    end

    it 'has 1st Rank king side Knight in correct position' do
      expect(board.data[7][6].instance_of?(Knight)).to be true
    end

    it 'has 1st Rank king side Rook in correct position' do
      expect(board.data[7][7].instance_of?(Rook)).to be true
    end

    it 'has 8th Rank queen side Rook in correct position' do
      expect(board.data[0][0].instance_of?(Rook)).to be true
    end

    it 'has 8th Rank queen side Knight in correct position' do
      expect(board.data[0][1].instance_of?(Knight)).to be true
    end

    it 'has 8th Rank queen side Bishop in correct position' do
      expect(board.data[0][2].instance_of?(Bishop)).to be true
    end

    it 'has 8th Rank Queen in correct position' do
      expect(board.data[0][3].instance_of?(Queen)).to be true
    end

    it 'has 8th Rank King in correct position' do
      expect(board.data[0][4].instance_of?(King)).to be true
    end

    it 'has 8th Rank king side Bishop in correct position' do
      expect(board.data[0][5].instance_of?(Bishop)).to be true
    end

    it 'has 8th Rank king side Knight in correct position' do
      expect(board.data[0][6].instance_of?(Knight)).to be true
    end

    it 'has 8th Rank king side Rook in correct position' do
      expect(board.data[0][7].instance_of?(Rook)).to be true
    end

    it 'has 2nd Rank of pawns' do
      expect(board.data[6].all? { |piece| piece.instance_of?(Pawn) }).to be true
    end

    it 'has 7th Rank of pawns' do
      expect(board.data[1].all? { |piece| piece.instance_of?(Pawn) }).to be true
    end
  end
end
```

```
~/Coding/terminal-chess/spec main >1 !1 ?1 rspec board_spec.rb -f d

Board
  #initial_placement
    has bottom row (1st Rank) of white pieces
    has sixth row (2nd Rank) of white pieces
    has second row (7th Rank) of black pieces
    has top row (8th Rank) of black pieces
    has 1st Rank queen side Rook in correct position
    has 1st Rank queen side Knight in correct position
    has 1st Rank queen side Bishop in correct position
    has 1st Rank Queen in correct position
    has 1st Rank King in correct position
    has 1st Rank king side Bishop in correct position
    has 1st Rank king side Knight in correct position
    has 1st Rank king side Rook in correct position
    has 8th Rank queen side Rook in correct position
    has 8th Rank queen side Knight in correct position
    has 8th Rank queen side Bishop in correct position
    has 8th Rank Queen in correct position
    has 8th Rank King in correct position
    has 8th Rank king side Bishop in correct position
    has 8th Rank king side Knight in correct position
    has 8th Rank king side Rook in correct position
    has 2nd Rank of pawns
    has 7th Rank of pawns

Finished in 0.52254 seconds (files took 0.13608 seconds to load)
22 examples, 0 failures
```

RSpec Testing

Saving and Loading Games

```ruby
RSpec.describe Serializer do
  let(:dummy_class) { Class.new { extend Serializer } }

  describe '#select_saved_game' do
    context 'when user input is valid' do
      it 'returns valid user input' do
        input = '1'
        allow(dummy_class).to receive(:gets).and_return(input)
        result = dummy_class.select_saved_game(3)
        expect(result).to eq('1')
      end
    end

    context 'when user input is invalid' do
      it 'outputs an input error warning' do
        warning = "\e[91mInput Error!\e[0m Please enter a valid file number."
        expect(dummy_class).to receive(:puts).with(warning).once
        valid_input = '1'
        invalid_input = 'a'
        allow(dummy_class).to receive(:gets).and_return(invalid_input, valid_input)
        dummy_class.select_saved_game(3)
      end
    end

    describe '#save_game' do
      before do
        allow(dummy_class).to receive(:puts)
        allow(Marshal).to receive(:dump)
      end

      it 'opens a file' do
        expect(File).to receive(:open)
        dummy_class.save_game
      end

      it 'dumps the file' do
        expect(Marshal).to receive(:dump)
        dummy_class.save_game
      end

      it 'does not raise an error' do
        expect { dummy_class.save_game }.not_to raise_error
      end
    end

    describe '#load_game' do
      before do
        allow(dummy_class).to receive(:find_saved_file)
        allow(Marshal).to receive(:load)
      end

      it 'opens a file' do
        expect(File).to receive(:open)
        dummy_class.load_game
      end

      it 'loads the file' do
        expect(Marshal).to receive(:load)
        dummy_class.load_game
      end
    end
  end
```

RSpec Testing

Saving and Loading Games

```
~/Coding/terminal-chess/spec main !8 ?15 rspec serializer_spec.rb -f d

Serializer
  #select_saved_game
    when user input is valid
      returns valid user input
    when user input is invalid
      outputs an input error warning
    #save_game
      opens a file (FAILED - 1)
      dumps the file (FAILED - 2)
      does not raise an error (FAILED - 3)
    #load_game
      opens a file
      loads the file

Failures:

  1) Serializer#select_saved_game #save_game opens a file
     Failure/Error: dummy_class.save_game

     NameError:
       undefined local variable or method `play' for #<Class:0x00007fb312135480>
     # /Users/Wammies/Coding/terminal-chess/lib/serializer.rb:13:in `save_game'
     # ./serializer_spec.rb:37:in `block (4 levels) in <top (required)>'

  2) Serializer#select_saved_game #save_game dumps the file
     Failure/Error: dummy_class.save_game

     NameError:
       undefined local variable or method `play' for #<Class:0x00007fb311a06628>
     # /Users/Wammies/Coding/terminal-chess/lib/serializer.rb:13:in `save_game'
     # ./serializer_spec.rb:42:in `block (4 levels) in <top (required)>'

  3) Serializer#select_saved_game #save_game does not raise an error
     Failure/Error: expect { dummy_class.save_game }.not_to raise_error

       expected no Exception, got #<NameError: undefined local variable or method `play' for #<Class:0x00007fb3119f5e68>> with backtrace:
         # /Users/Wammies/Coding/terminal-chess/lib/serializer.rb:13:in `save_game'
         # ./serializer_spec.rb:46:in `block (5 levels) in <top (required)>'
         # ./serializer_spec.rb:46:in `block (4 levels) in <top (required)>'
     # ./serializer_spec.rb:46:in `block (4 levels) in <top (required)>'

Finished in 6.06 seconds (files took 0.11454 seconds to load)
7 examples, 3 failures

Failed examples:

rspec ./serializer_spec.rb:35 # Serializer#select_saved_game #save_game opens a file
rspec ./serializer_spec.rb:40 # Serializer#select_saved_game #save_game dumps the file
rspec ./serializer_spec.rb:45 # Serializer#select_saved_game #save_game does not raise an error
```

```
Traceback (most recent call last):
        7: from /Users/Wammies/Coding/terminal-chess/lib/main.rb:53:in `<main>'
        6: from /Users/Wammies/Coding/terminal-chess/lib/main.rb:53:in `loop'
        5: from /Users/Wammies/Coding/terminal-chess/lib/main.rb:55:in `block in <main>'
        4: from /Users/Wammies/Coding/terminal-chess/lib/main.rb:35:in `play_game'
        3: from /Users/Wammies/Coding/terminal-chess/lib/serializer.rb:26:in `load_game'
        2: from /Users/Wammies/Coding/terminal-chess/lib/serializer.rb:26:in `open'
        1: from /Users/Wammies/Coding/terminal-chess/lib/serializer.rb:27:in `block in load_game'
/Users/Wammies/Coding/terminal-chess/lib/serializer.rb:27:in `load': end of file reached (EOFError)
```

RSpec Testing

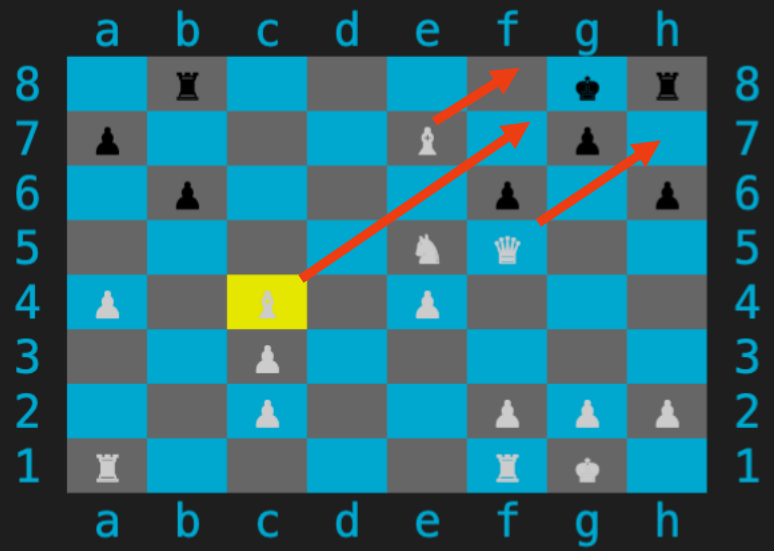
Saving and Loading Games

```
~/Coding/terminal-chess/spec main !8 ?12 rspec serializer_spec.rb -f d

Serializer
  #select_saved_game
    when user input is valid
      returns valid user input
    when user input is invalid
      outputs an input error warning
    #save_game
      opens a file
      dumps the file
      does not raise an error
    #load_game
      opens a file
      loads the file

Finished in 6.03 seconds (files took 0.11839 seconds to load)
7 examples, 0 failures
```

Manual Testing

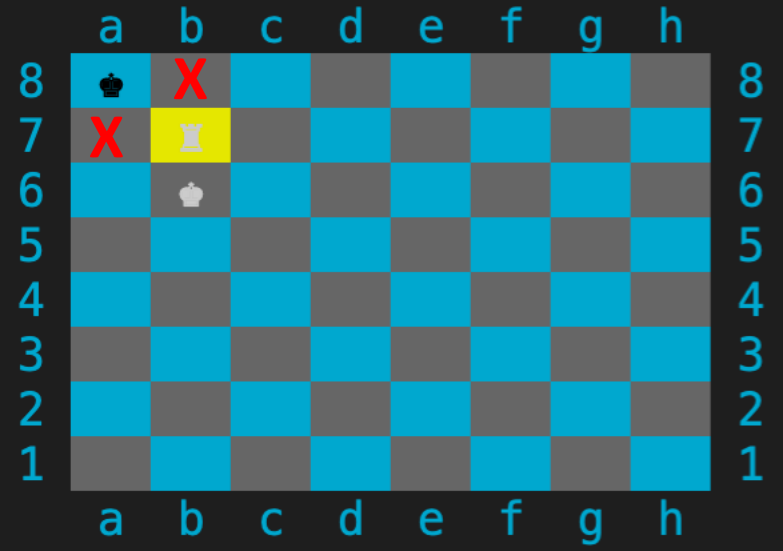
LEVEL UP!

Checkmate




```
CHECKMATE! WHITE WINS!!

MENU (Press ↑/↓ arrow to move and
▸ 1 - Single Player
  2 - Two Player
  3 - Load Game
  4 - How to Play
  5 - Quit
```

Stalemate

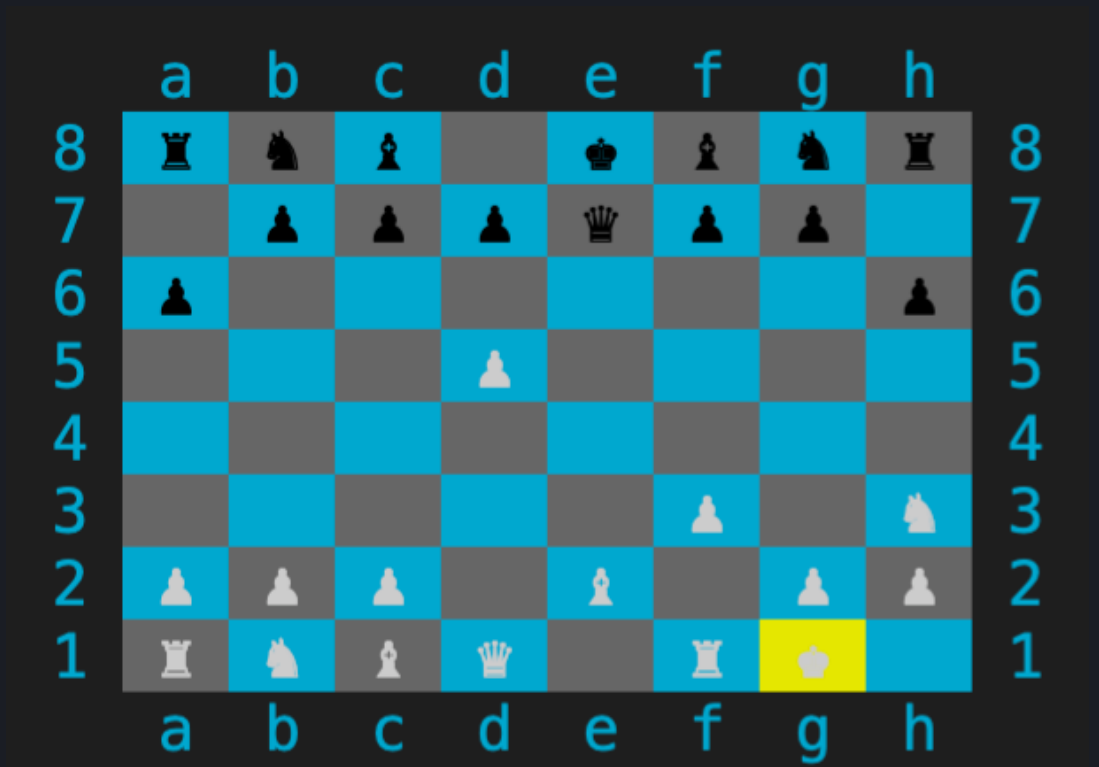



```
STALEMATE! Game is a draw.

MENU (Press ↑/↓ arrow to move ar
▸ 1 - Single Player
  2 - Two Player
  3 - Load Game
  4 - How to Play
  5 - Quit
```

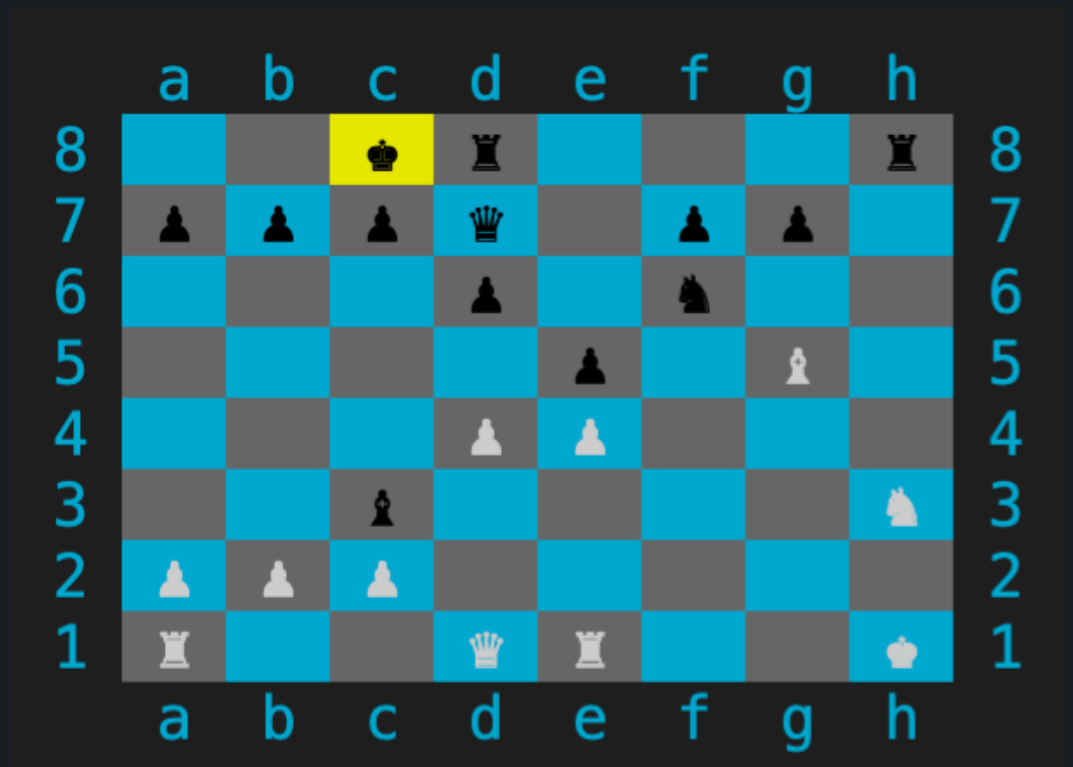
Manual Testing

LEVEL UP!

Castle King Side

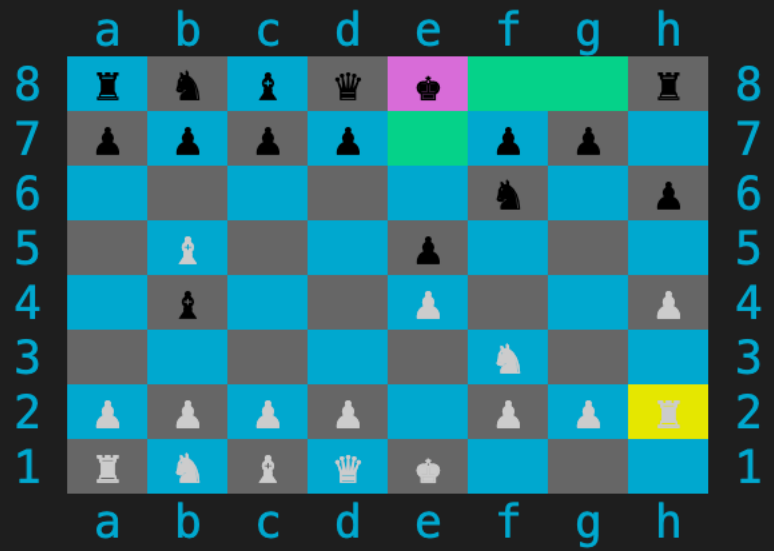


Castle Queen Side

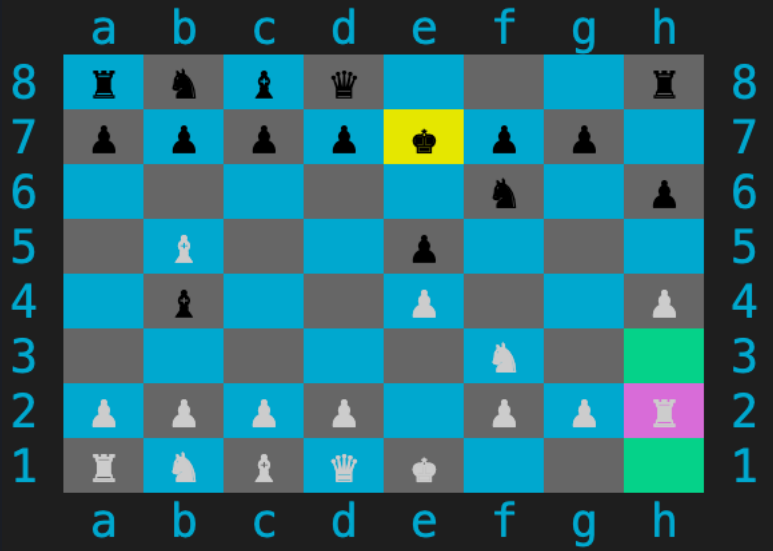


Manual Testing

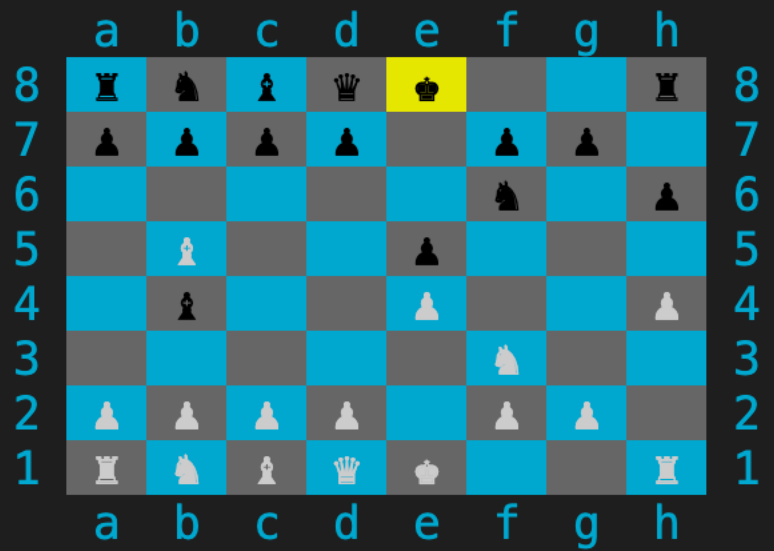
1] Moves White King side Rook



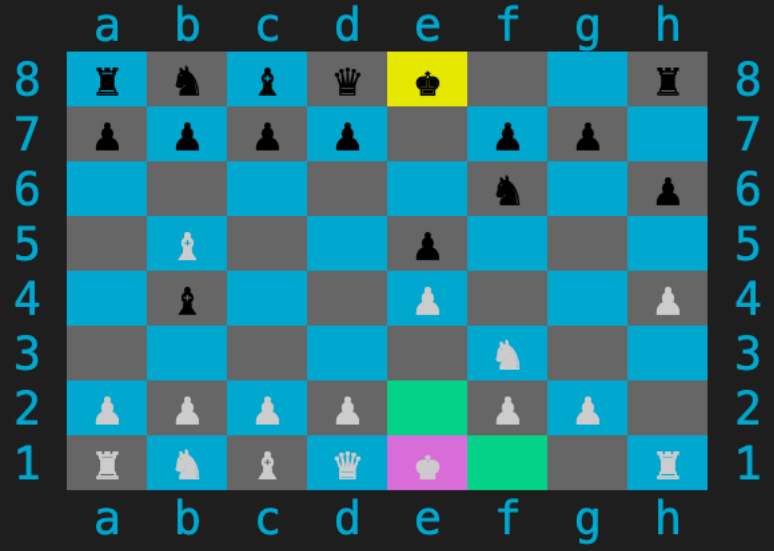
2] Moves Black King



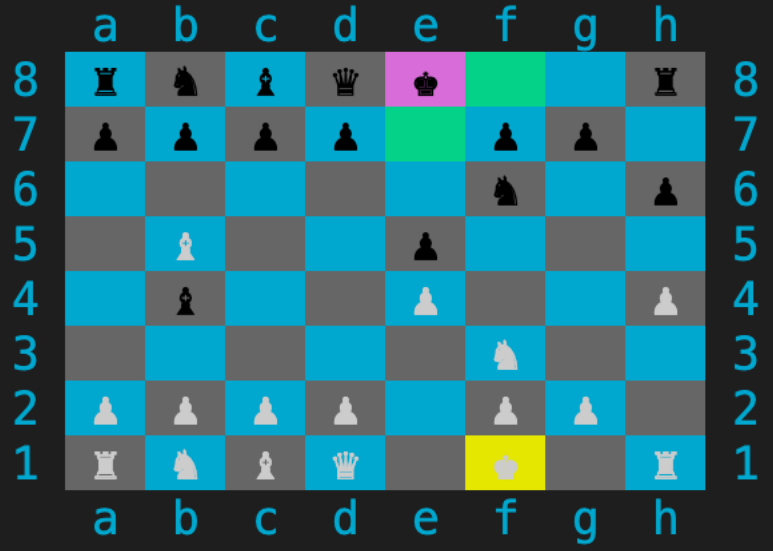
3] Moves White Rook and Black King back



4] White King can no longer castle

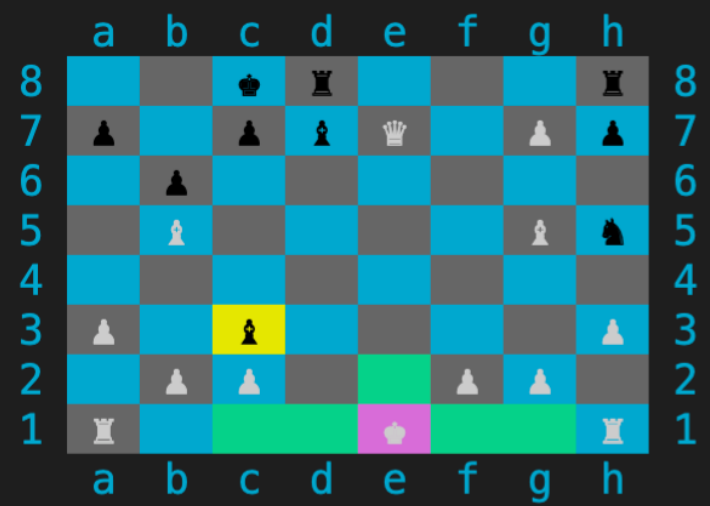


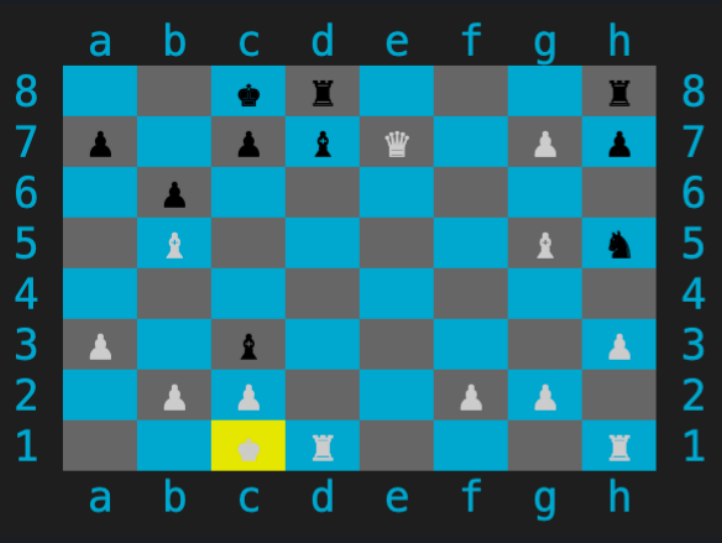
5] Black King can no longer castle



Manual Testing

King can’t castle OUT of check

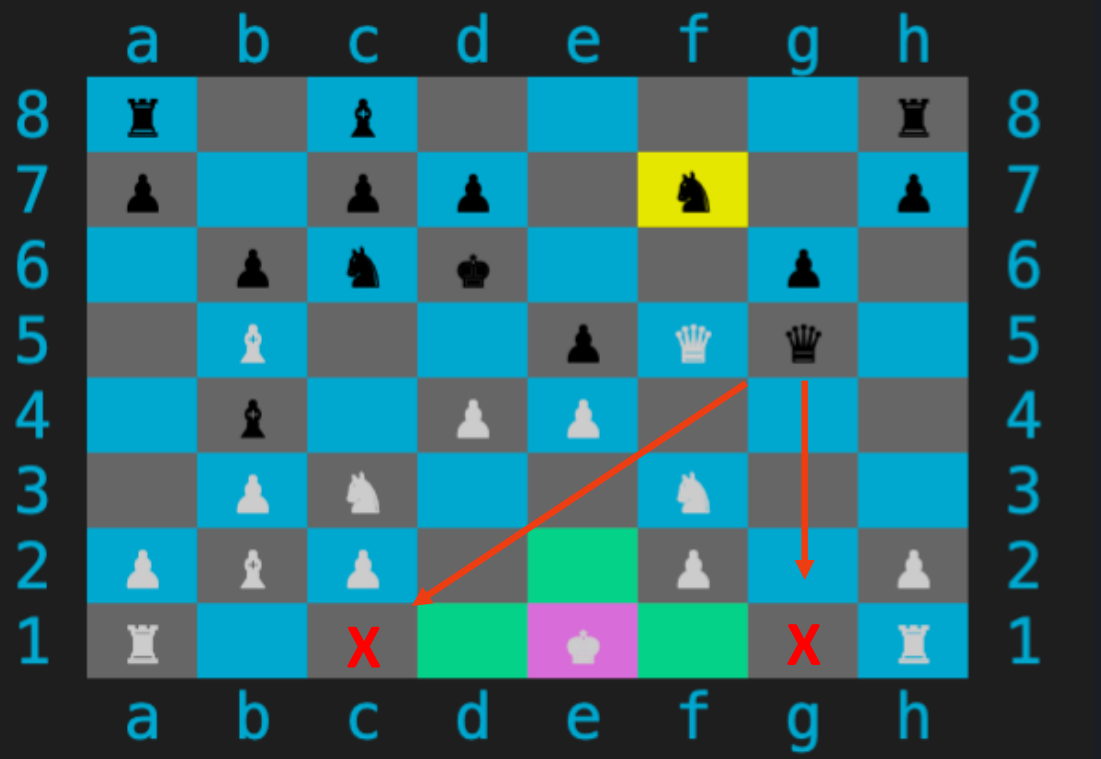




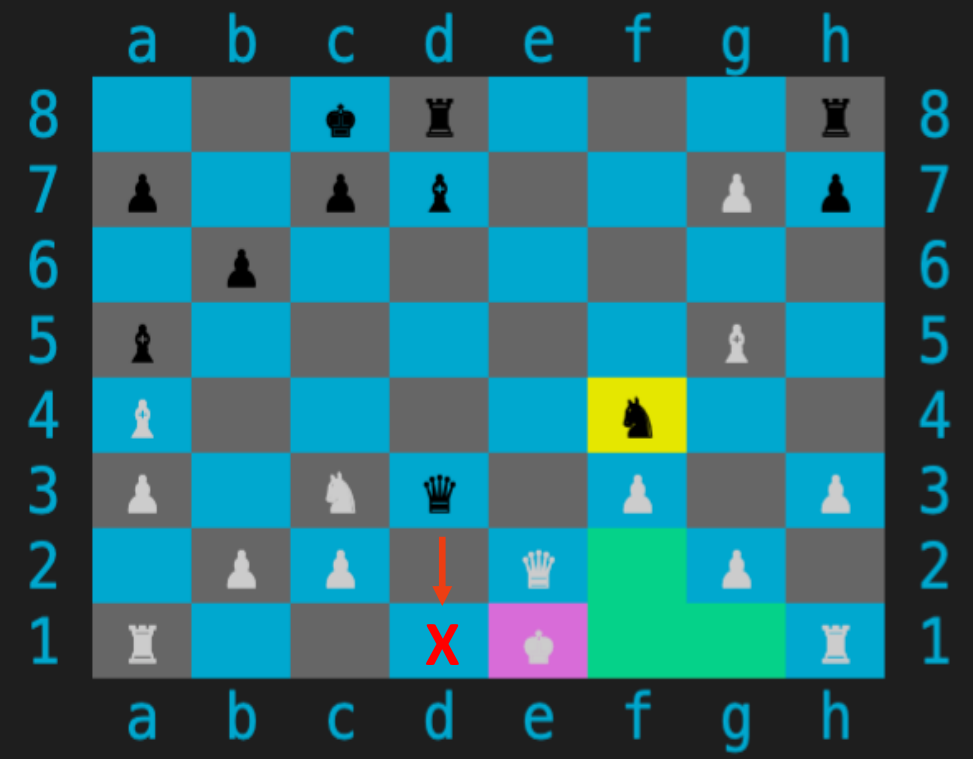
```
You have the option to castle:

Your King ♔ will move 2 spaces
and will castle with the rook.
Your Rook ♖ will be moved to the
square that the king passes through.


Please enter coordinates for a legal move:
square(s) highlighted ■ or capture ♟.

c1
```

Manual Testing

King cannot castle INTO check



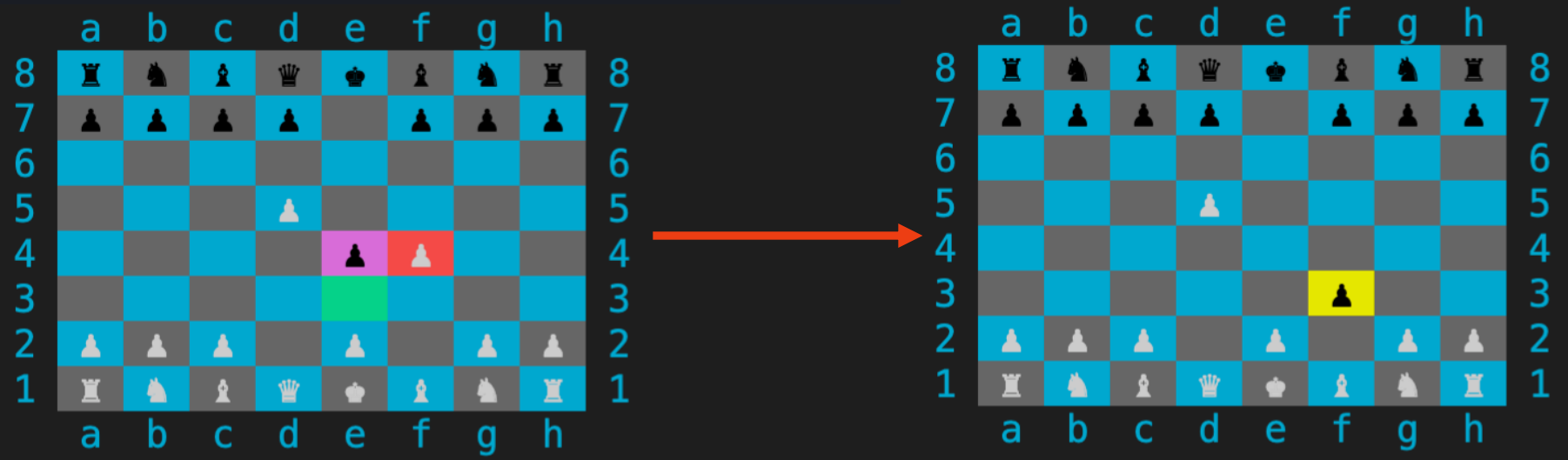
King cannot castle THROUGH check



Manual Testing

LEVEL UP!

En Passant

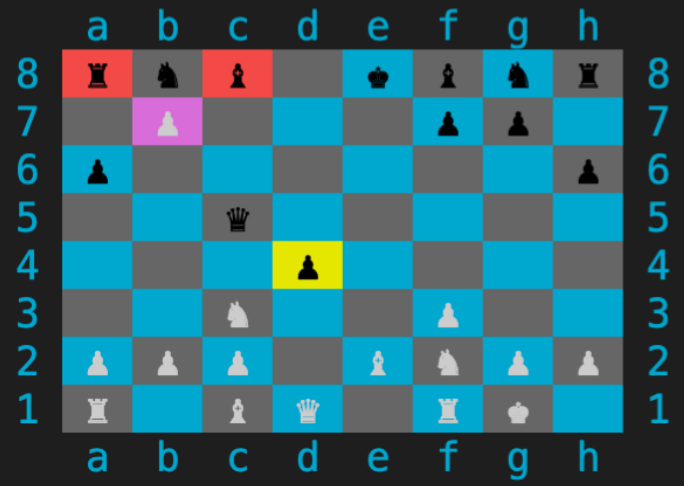


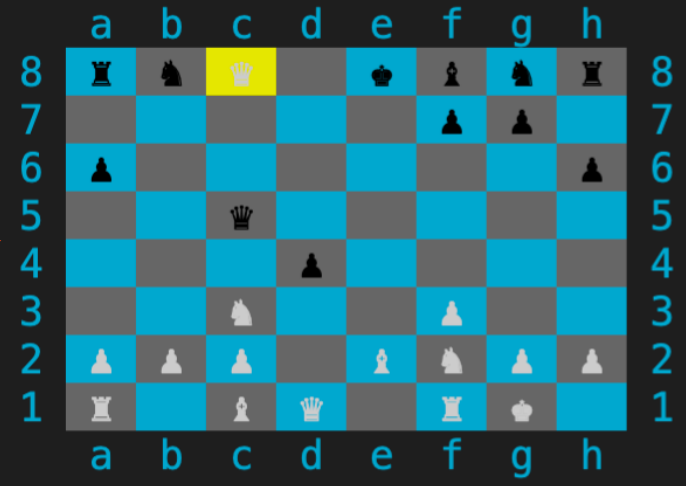
Option to capture the opposing pawn that just moved.
To capture this pawn en passant (in passing),
please enter the highlighted coordinates.
As part of en passant, your pawn will be moved
to the square in front of the captured pawn.

Please enter coordinates for a legal move:
square(s) highlighted ▬ or capture ♟.

Manual Testing

Pawn Promotion to Queen



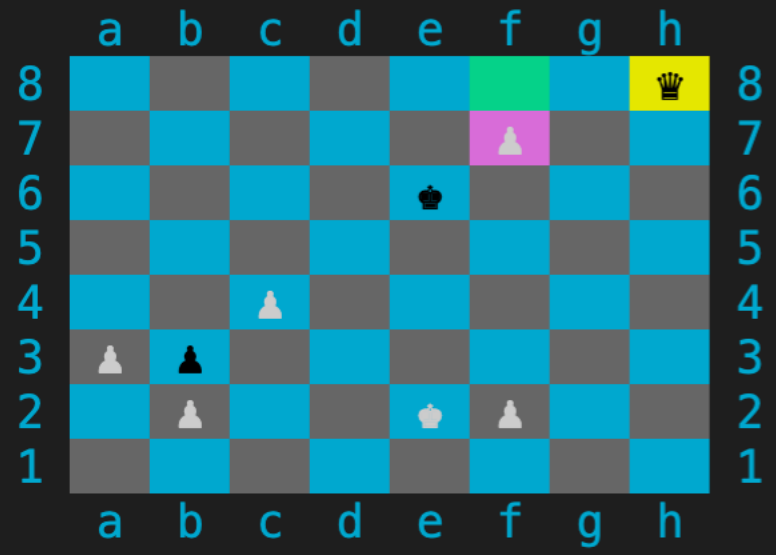


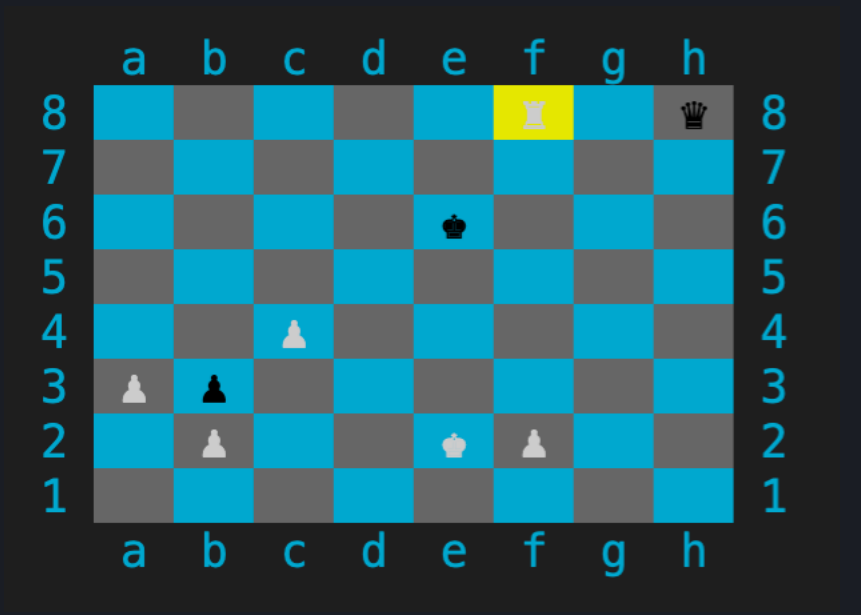
```
Please enter coordinates for a legal move:
square(s) highlighted ■ or capture ♟.

c8
To select the piece you wish to promote your pawn to,
enter it's corresponding number:

[1] for a ♕ Queen
[2] for a ♗ Bishop
[3] for a ♘ Knight
[4] for a ♖ Rook

5
Input error! Please enter 1, 2, 3, or 4.
```

```
BLACK TO MOVE:

WARNING! Your ♔ King is currently in check!
```

Manual Testing

Pawn Promotion to Rook

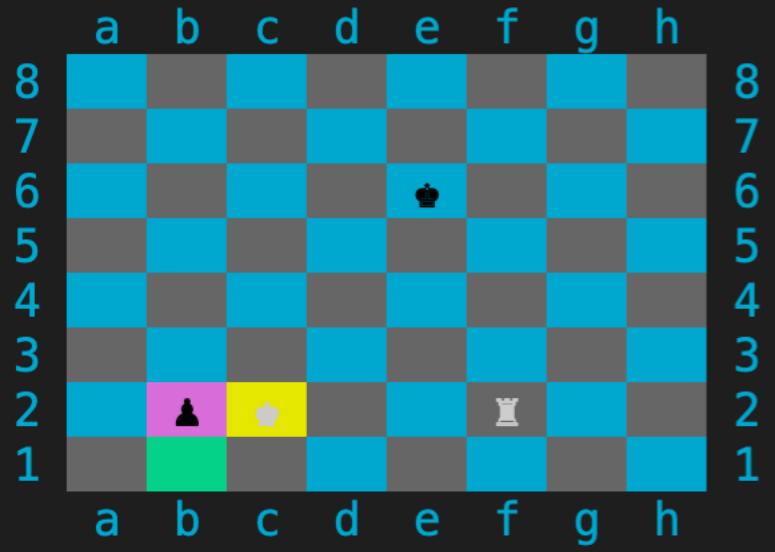


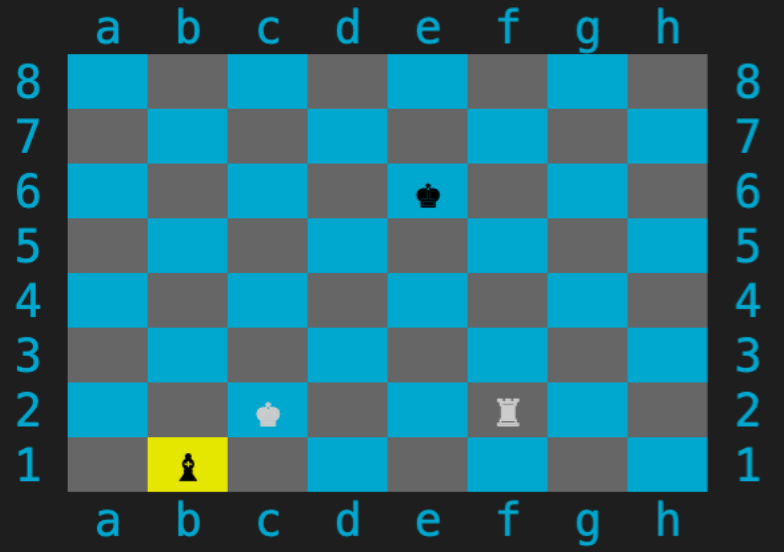



```
Please enter coordinates for a

f8
To select the piece you wish to
[1] for a ♕ Queen
[2] for a ♗ Bishop
[3] for a ♘ Knight
[4] for a ♖ Rook

4
```

Manual Testing

Pawn Promotion to Bishop



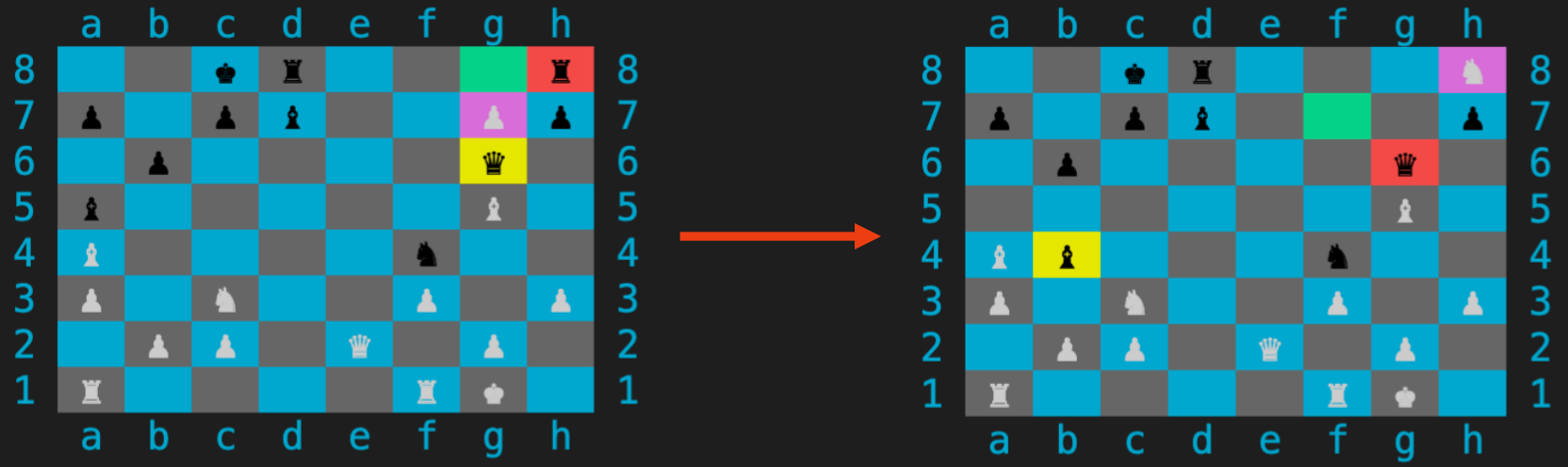



```
Please enter coordinates for a

b1
To select the piece you wish to
[1] for a ♕ Queen
[2] for a ♗ Bishop
[3] for a ♘ Knight
[4] for a ♖ Rook

2
```

```
WHITE TO MOVE:
WARNING! Your ♔ King is currently in check!
```

Manual Testing

Pawn Promotion to Knight

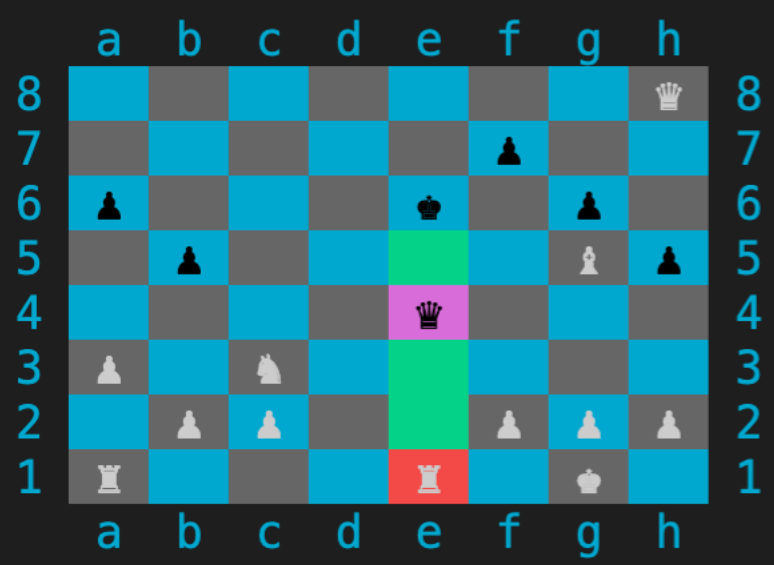


```
Please enter coordinates for a legal move:
square(s) highlighted ■ or capture ♟.

h8
To select the piece you wish to promote your pawn to,
enter it's corresponding number:

[1] for a ♕ Queen
[2] for a ♗ Bishop
[3] for a ♘ Knight
[4] for a ♖ Rook

3
```

Manual Testing

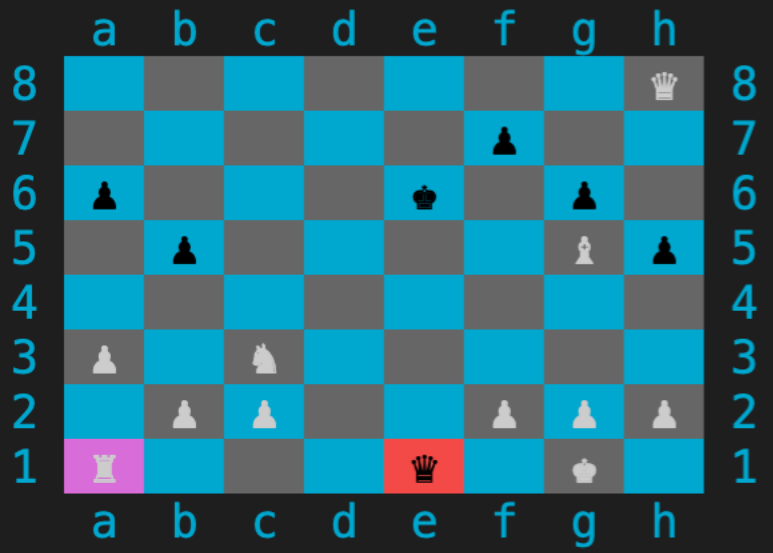
LEVEL UP!

Move Validation:
Queen can only move on file because pinned to King by the white Rook. Not able to move in other directions as not a legal move



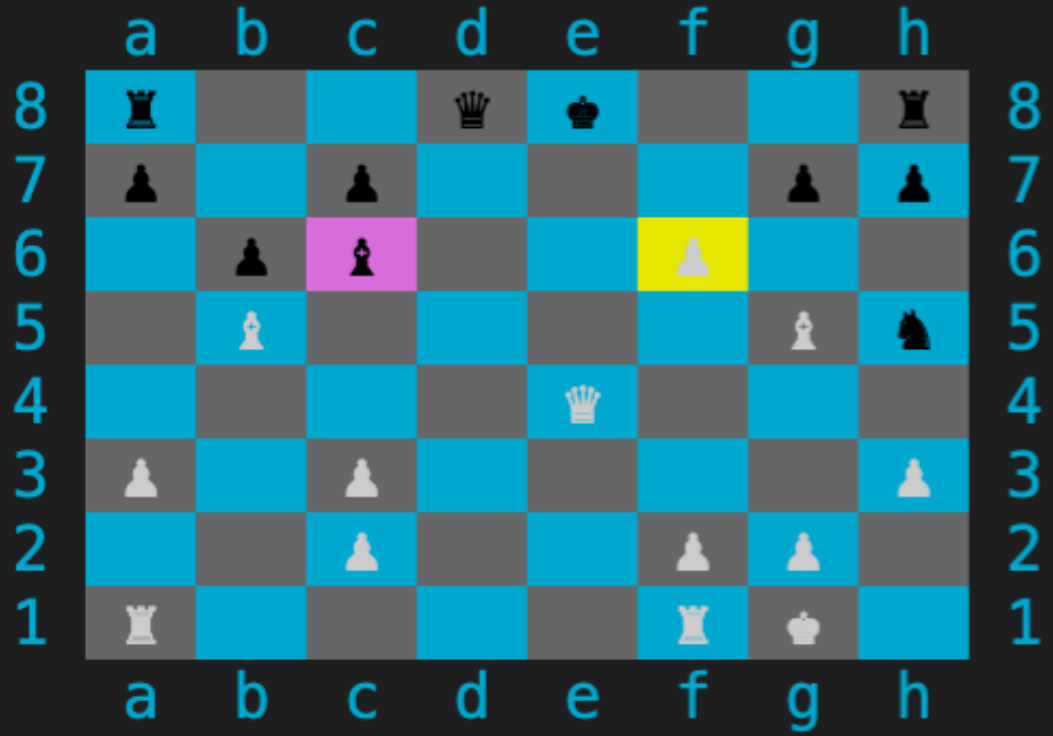

```
Please enter coordinates for a

e1
```

Move Validation:
Rook can only capture Queen as that is the only legal move as King is in check.




```
Please enter coordinates for a

```

Manual Testing




```
BLACK TO MOVE:

WARNING! Your ♔ King is currently in check!


Please enter the coordinates of the piece you wish to move.

c6
There are no legal moves for this piece and/or
King is under check.
Please select another piece to move.
```

Move Validation

Very complex case: White's e pawn has just captured a piece, resulting in a discovered check on the black King from the white Queen.

The Black Bishop cannot capture the checking piece (White Queen), as it is currently pinned to the King by the White Bishop on b5.

Game alerts with PieceError, no legal moves for the piece and/or King is under check.

Only legal moves are: Queen e7 and King f7 or f8

Error Handling

```ruby
# declares an error message when user enters an invalid input
class InputError < StandardError
  def message
    <<~HEREDOC
      \e[91mInvalid input!\e[0m
      Please enter a \e[94mletter\e[0m [a-h]and\e[94m number\e[0m [1-8].
      \e[94m eg: d2\e[0m
    HEREDOC
  end
end

# declares an error message when user enters coordinates that aren't player's pieces,
# i.e. opponent's pieces or empty square
class CoordinatesError < StandardError
  def message
    <<~HEREDOC
      \e[91mInvalid coordinates!\e[0m
      Please enter file and rank of a piece that is \e[94myour color\e[0m.
    HEREDOC
  end
end

# declares an error message when user enters invalid coordinates for move
class MoveError < StandardError
  def message
    <<~HEREDOC
      \e[91mInvalid move!\e[0m
      Please enter file and rank of a \e[92mvalid move\e[0m.
    HEREDOC
  end
end

# declares an error message when user selects a piece with no legal moves
class PieceError < StandardError
  def message
    <<~HEREDOC
      There are\e[91m no legal moves\e[0m for this piece and/or
      \e[91mKing is under check\e[0m.
      Please \e[94mselect another piece to move.
    HEREDOC
  end
end
```

```ruby
def validate_piece_input(input)
  raise InputError unless input.match?(/^[a-h][1-8]$|^[nslq]$/i)
end

def validate_move_input(input)
  raise InputError unless input.match?(/^[a-h][1-8]$|^[nslq]$/i)
end

def validate_piece_coordinates(coordinates)
  raise CoordinatesError unless @board.valid_piece?(coordinates, @current_turn)
end

def validate_move(coordinates)
  raise MoveError unless @board.valid_piece_movement?(coordinates)
end

def validate_active_piece
  raise PieceError unless @board.active_piece_moveable?
end
```

Error Handling

```ruby
loop do
    main_menu
rescue StandardError, NoMemoryError, ScriptError, SecurityError, SystemStackError => e
    puts e.message
end
```

```ruby
def select_piece_coordinates
  input = user_select_piece
  return unless @player_count.positive?

  coordinates = translate_coordinates(input)
  validate_piece_coordinates(coordinates)
  @board.update_active_piece(coordinates)
  validate_active_piece
rescue StandardError => e
  puts e.message
  retry
end

def select_move_coordinates
  input = user_select_move
  coordinates = translate_coordinates(input)
  validate_move(coordinates)
  coordinates
rescue StandardError => e
  puts e.message
  retry
end
```

```ruby
module Serializer
  def save_game
    Dir.mkdir 'saved_games' unless Dir.exist? 'saved_games'
    filename = create_filename
    File.open("saved_games/#{filename}", 'w+') do |file|
      Marshal.dump(self, file)
    end
    puts "Game was saved as \e[96m#{filename}\e[0m\n"
    sleep(2)
    play # user can continue playing game after saving
  rescue SystemCallError => e
    puts "\e[91mError while writing to file #{filename}.\e[0m"
    puts e
  end
  def load_game
    file_name = find_saved_file
    File.open("saved_games/#{file_name}") do |file|
      Marshal.load(file)
    end
  rescue IOError => e
    puts "\e[91mError while loading file #{file_name}.\e[0m"
    puts e
  end
```

Review

- Challenges
- Favourite Parts

Challenges

I've never encountered so many errors in my life before.

It was challenging to identify and debug errors when I had all the error handling in place:

disable the error handling to locate what went wrong, each time! (should've built a method to do this)

There was 2 times where I literally could not figure out where the errors were coming from and it broke the entire game when it crashes, especially towards nearly the end of development when I was refactoring and adding a bunch of prompts.

Had to do 2 git rollbacks (create a new branch going back to a previous commit in history).

Had my first experience resolving conflicts in GitHub.

Figuring out how to castle and enpassant was extremely hard, had to break it down into each and every step so that the computer would understand the instructions and process it correctly.

Favourite Parts

The euphoric sense of accomplishment and satisfaction coding something so challenging and difficult.

Feels like if I just climbed Mt. Everest.

I spent a fair bit of time using the “RUBBER DUCK” method. Mum thought I was going crazy talking to myself about non-sense. I was quite skeptical at first, but I can now confidently say, IT WORKS!

Discovered a GEM! (pun intended), Rubocop is such an amazing linter!! I was already implementing best practices using AirBnB’s styling guidelines, but this detects anything that may cause errors like not having a space before/after operators and commas etc, it even autocorrects incorrect indentation. Even tells you how you could code something using a particular method instead.

Getting up at 4am for 2 hours of uninterrupted coding amongst busy schedule.


thank
you