1) a Readme file to explain a chees code

-the chess game is based on Two player mode

-it is created by using pygame

2) Functions used in this code

-draw_board Function

```
def draw_board():
    for i in range(32):
        column = i % 4
        row = i // 4
        if row % 2 == 0:
            pg.draw.rect(game_console , 'light gray', [600 - (column * 200), row * 100, 100, 100])
        else:
            pg.draw.rect(game_console , 'light gray', [700 - (column * 200), row * 100, 100, 100])
        pg.draw.rect(game_console , 'gray', [0, 800, WIDTH, 100])
        pg.draw.rect(game_console , 'gold', [0, 800, WIDTH, 100], 5)
        pg.draw.rect(game_console , 'gold', [800, 0, 200, HEIGHT], 5)
        status_text = ['White: Select a Piece to Move', 'White: Select a Destination',
                       'Black: Select a Piece to Move', 'Black: Select a Destination']
        game_console .blit(big_font.render(status_text[turn_step], True, 'black'), (20, 820))
        for i in range(9):
            pg.draw.line(game_console , 'gold', (0, 100 * i), (800, 100 * i), 2)
            pg.draw.line(game_console , 'gold', (100 * i, 0), (100 * i, 800), 2)
        game_console .blit(medium_font.render('Give up', True, 'red'), (810, 830))
```

this function is called to draw:

-the grid shapes of the chess game.

- an empty box to type instructions text during the game.

- and anoter empty box at the right to draw the deceased pieces.

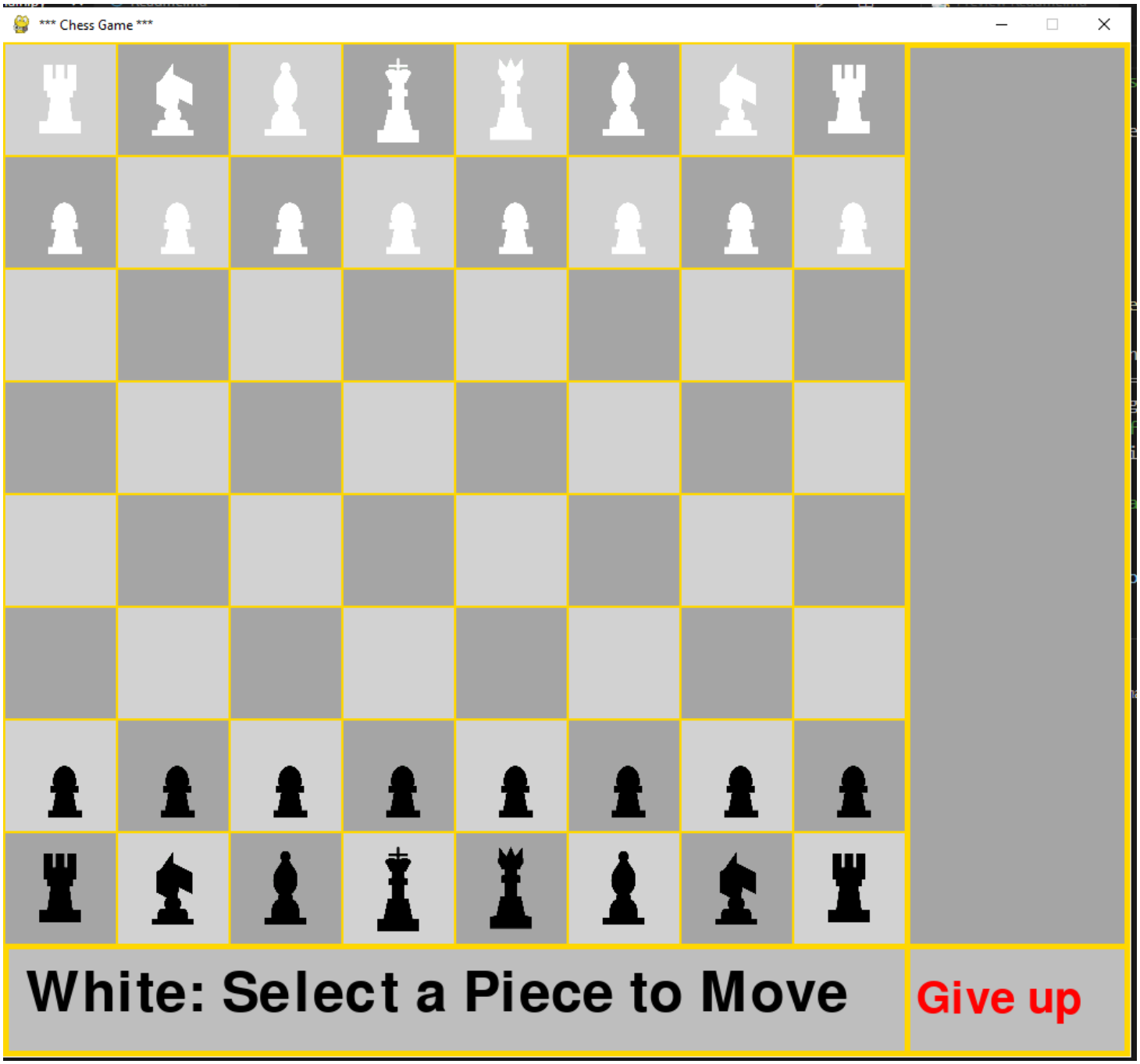
-draw pieces Function

```python
def draw_pieces():
    for i in range(len(white_pieces)):
        index = piece_list.index(white_pieces[i])
        if white_pieces[i] == 'pawn':
            game_console .blit(white_pawn, (white_locations[i][0] * 100 + 22, white_locations[i][1] * 100 + 30))
        else:
            game_console .blit(white_images[index], (white_locations[i][0] * 100 + 10, white_locations[i][1] * 100 + 10))
        if turn_step < 2:
            if selection == i:
                pg.draw.rect(game_console , 'red', [white_locations[i][0] * 100 + 1, white_locations[i][1] * 100 + 1, 100, 100], 2)

    for i in range(len(black_pieces)):
        index = piece_list.index(black_pieces[i])
        if black_pieces[i] == 'pawn':
            game_console .blit(black_pawn, (black_locations[i][0] * 100 + 22, black_locations[i][1] * 100 + 30))
        else:
            game_console .blit(black_images[index], (black_locations[i][0] * 100 + 10, black_locations[i][1] * 100 + 10))
        if turn_step >= 2:
            if selection == i:
                pg.draw.rect(game_console , 'blue', [black_locations[i][0] * 100 + 1, black_locations[i][1] * 100 + 1,100, 100], 2)
```

the main job of this function is :

- to draw and put the pieces at the right postion of the game grid like this :



-check options Function

this function it take the a list of the game pieces and all locations and the turn which refer to which player turn ant then it reurn a list of all moves that any piece could take

```python
def check_options(pieces, locations, turn):

    moves_list = []
    all_moves_list = []
    for i in range((len(pieces))):
        location = locations[i]
        piece = pieces[i]
        if piece == 'pawn':
            moves_list = check_pawn(location, turn)
        elif piece == 'rook':
            moves_list = check_rook(location, turn)
        elif piece == 'knight':
            moves_list = check_knight(location, turn)
        elif piece == 'bishop':
            moves_list = check_bishop(location, turn)
        elif piece == 'queen':
            moves_list = check_queen(location, turn)
        elif piece == 'king':
            moves_list = check_king(location, turn)
        all_moves_list.append(moves_list)
    return all_moves_list
```

- a pieces moves option Function

it is like check_pawn(position, color) , check_rook(position, color) ,check_king(position, color) and so on.

the check_pawn Function it detemine the intial move of the pawn like Two forward step and it also determine the diagonals moves during the game

```python
def check_pawn(position, color):
    moves_list = []
    if color == 'white':
        if (position[0], position[1] + 1) not in white_locations and \
                (position[0], position[1] + 1) not in black_locations and position[1] < 7:
            moves_list.append((position[0], position[1] + 1))
        if (position[0], position[1] + 2) not in white_locations and \
                (position[0], position[1] + 2) not in black_locations and position[1] == 1:
            moves_list.append((position[0], position[1] + 2))
        if (position[0] + 1, position[1] + 1) in black_locations:
            moves_list.append((position[0] + 1, position[1] + 1))
        if (position[0] - 1, position[1] + 1) in black_locations:
            moves_list.append((position[0] - 1, position[1] + 1))
    else:
        if (position[0], position[1] - 1) not in white_locations and \
                (position[0], position[1] - 1) not in black_locations and position[1] > 0:
            moves_list.append((position[0], position[1] - 1))
        if (position[0], position[1] - 2) not in white_locations and \
                (position[0], position[1] - 2) not in black_locations and position[1] == 6:
            moves_list.append((position[0], position[1] - 2))
        if (position[0] + 1, position[1] - 1) in white_locations:
            moves_list.append((position[0] + 1, position[1] - 1))
        if (position[0] - 1, position[1] - 1) in white_locations:
            moves_list.append((position[0] - 1, position[1] - 1))
    return moves_list
```

check valid moves Function

this function main objective is to estimate the vaild move of the selected piece

```python
297 def check_valid_moves():
298     if turn_step < 2:
299         options_list = white_options
300     else:
301         options_list = black_options
302     valid_options = options_list[selection]
303     return valid_options
304
```

draw valid Function

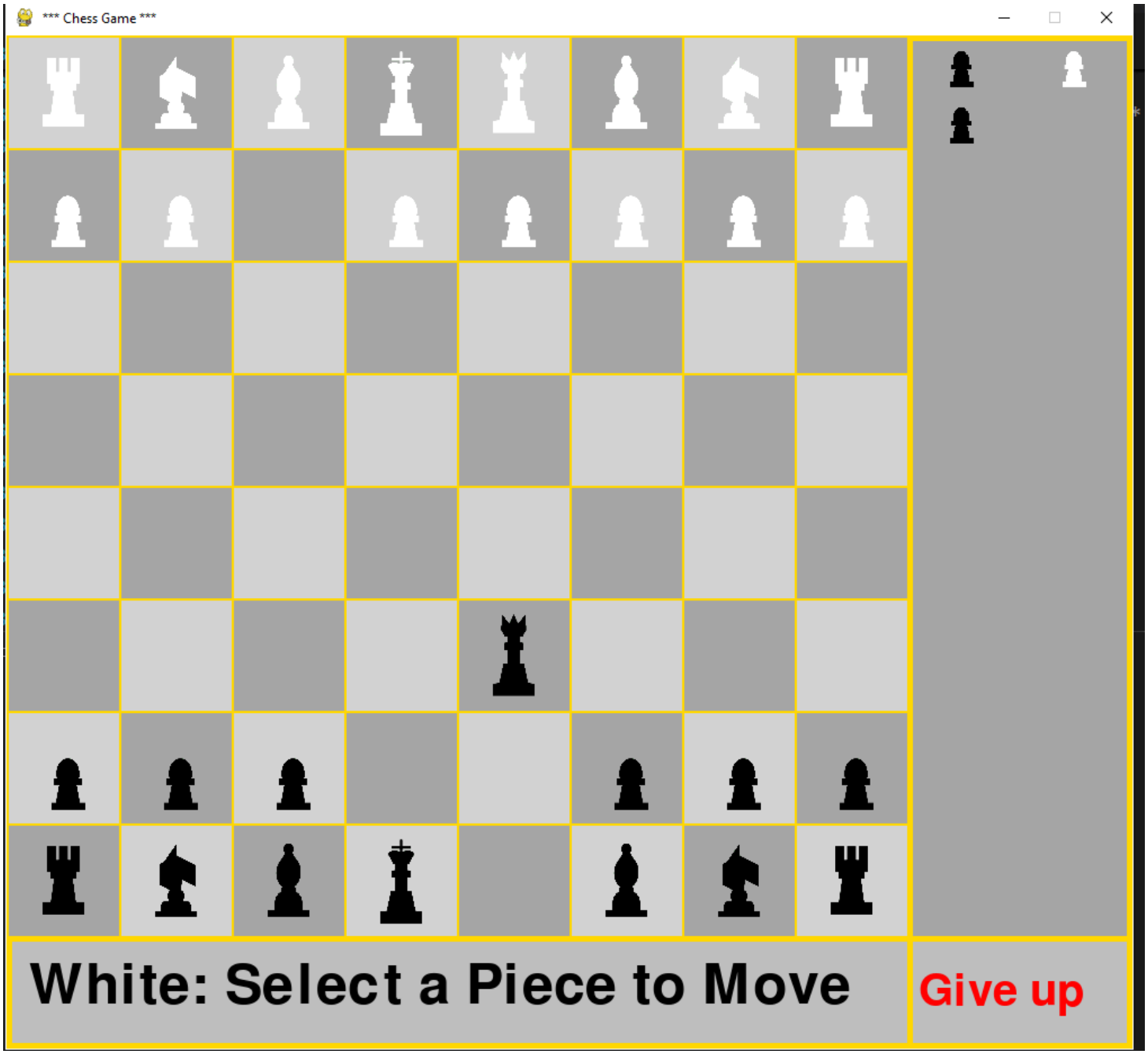
this function draw the valid moves of the selected piece

```python
def draw_valid(moves):
    if turn_step < 2:
        color = 'red'
    else:
        color = 'blue'
    for i in range(len(moves)):
        pg.draw.circle(game_console , color, (moves[i][0] * 100 + 50, moves[i][1] * 100 + 50), 5)
```

draw captured pieces Function

this function is draw the captured pieces in the empty box at the right

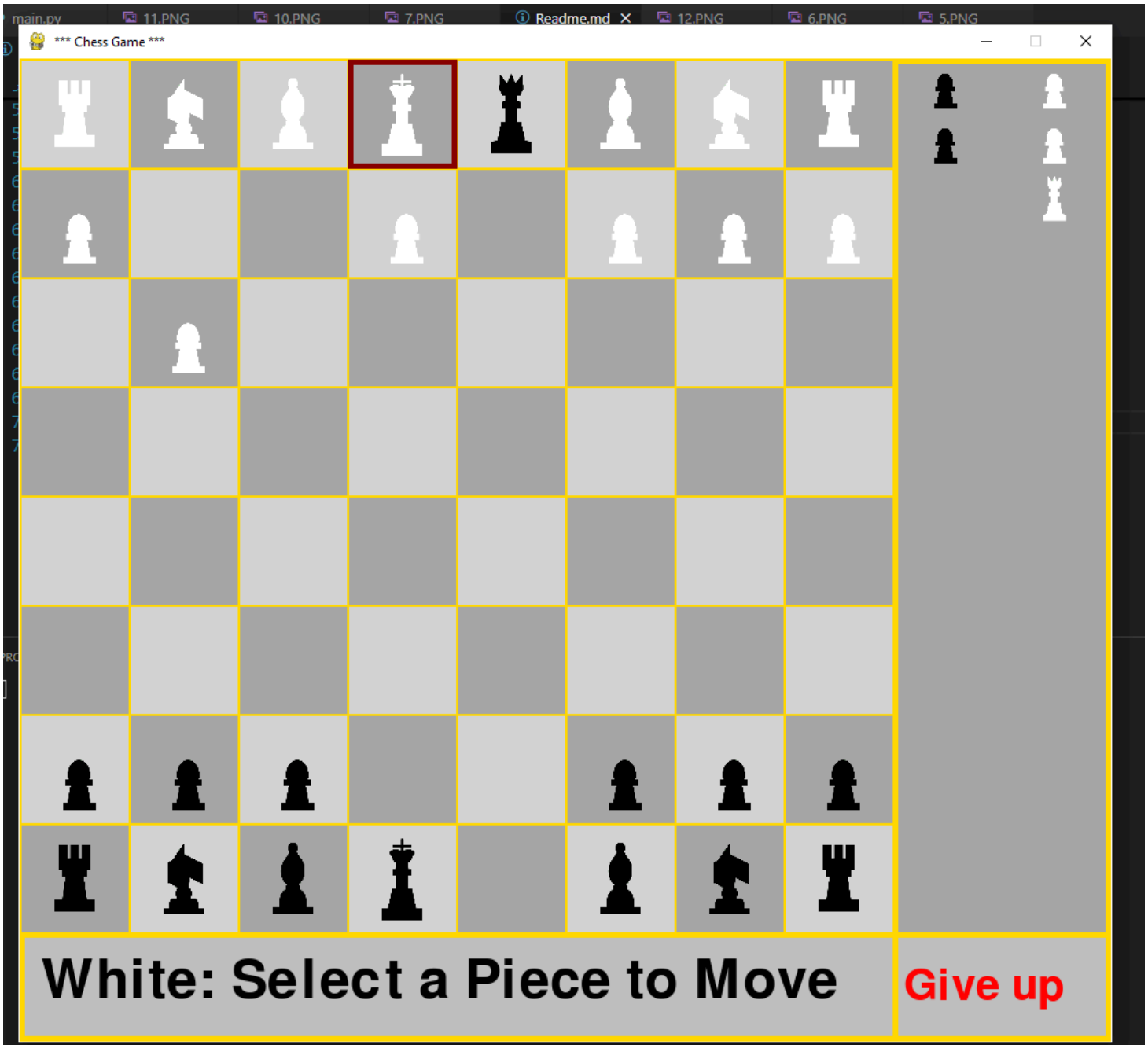
```
def draw_captured():
    for i in range(len(captured_pieces_white)):
        captured_piece = captured_pieces_white[i]
        index = piece_list.index(captured_piece)
        game_console .blit(small_black_images[index], (825, 5 + 50 * i))
    for i in range(len(captured_pieces_black)):
        captured_piece = captured_pieces_black[i]
        index = piece_list.index(captured_piece)
        game_console .blit(small_white_images[index], (925, 5 + 50 * i))
```



draw check Function

this function is draw a flashing square around the king if in the check

```python
def draw_check():
    if turn_step < 2:
        if 'king' in white_pieces:
            king_index = white_pieces.index('king')
            king_location = white_locations[king_index]
            for i in range(len(black_options)):
                if king_location in black_options[i]:
                    if counter < 15:
                        pg.draw.rect(game_console , 'dark red', [white_locations[king_index][0] * 100 + 1,
                                                                 white_locations[king_index][1] * 100 + 1, 100, 100], 5)
    else:
        if 'king' in black_pieces:
            king_index = black_pieces.index('king')
            king_location = black_locations[king_index]
            for i in range(len(white_options)):
                if king_location in white_options[i]:
                    if counter < 15:
                        pg.draw.rect(game_console , 'dark blue', [black_locations[king_index][0] * 100 + 1,
                                                                  black_locations[king_index][1] * 100 + 1, 100, 100], 5)
```

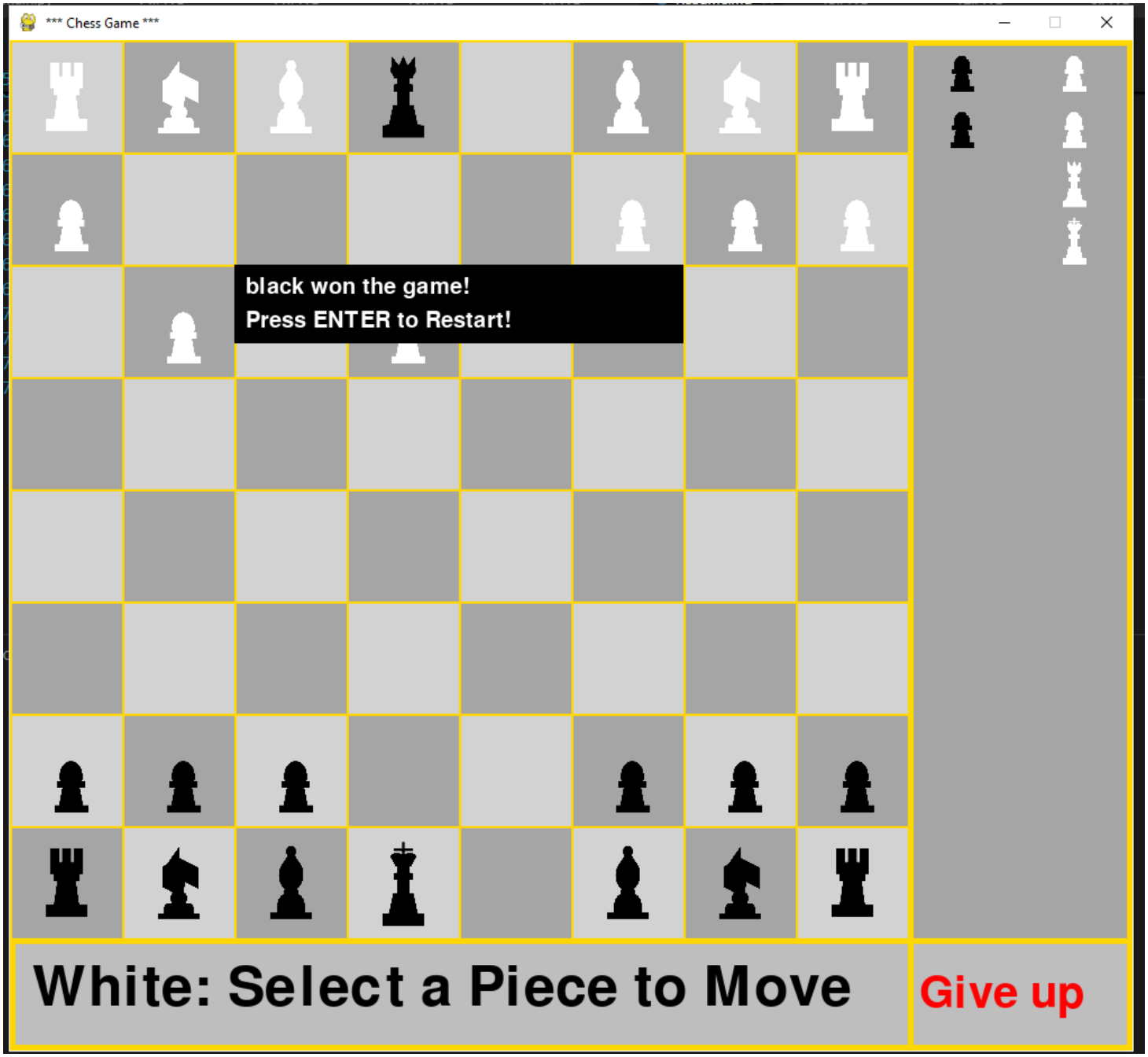


draw game over Function

this Function print who is the winner

```python
def draw_game_over():
    pg.draw.rect(game_console , 'black', [200, 200, 400, 70])
    game_console .blit(font.render(f'{winner} won the game!', True, 'white'), (210, 210))
    game_console .blit(font.render(f'Press ENTER to Restart!', True, 'white'), (210, 240))


# main game loop
```



3) MAIN LOOP

###at our main loop there is an event handlig

```
for event in pg.event.get():
    if event.type == pg.QUIT:
        run = False
    if event.type == pg.MOUSEBUTTONDOWN and event.button == 1 and not game_over:
        x_coord = event.pos[0] // 100
        y_coord = event.pos[1] // 100
        click_coords = (x_coord, y_coord)
```

-like handle if the user click the close button of the game console window

-and if the user clicked the mouse left button then it convert the where the user clicked to a point (x,y)

handling the Give up button

if the White pushed the Give up button

```
if turn_step <= 1:
    if click_coords == (8, 8) or click_coords == (9, 8):
        winner = 'black'
    if click_coords in white_locations:
```

if the Black pushed the Give up button

```
if turn_step > 1:
    if click_coords == (8, 8) or click_coords == (9, 8):
        winner = 'white'
    if click_coords in black_locations:
```