Python Chess

Part 1: the initial setup, defining the concept of a board, as well as fonts, frame rates, variables and images.

*import* pygame

pygame.init()

WIDTH *=* 1000

HEIGHT *=* 900

screen *=* pygame.display.set\_mode((WIDTH, HEIGHT), pygame.RESIZABLE)

pygame.display.set\_caption('Two-Player Pygame Chess!')

font *=* pygame.font.Font('freesansbold.ttf', 20)

medium\_font *=* pygame.font.Font('freesansbold.ttf', 40)

big\_font *=* pygame.font.Font('freesansbold.ttf', 50)

timer *=* pygame.time.Clock()

fps *=* 60

*# game variables and images*

white\_pieces *=* ['rook', 'knight', 'bishop', 'king', 'queen', 'bishop', 'knight', 'rook',

    'pawn', 'pawn', 'pawn', 'pawn', 'pawn', 'pawn', 'pawn', 'pawn']

white\_locations *=* [(0, 0), (1, 0), (2, 0), (3, 0), (4, 0), (5, 0), (6, 0), (7, 0),

       (0, 1), (1, 1), (2, 1), (3, 1), (4, 1), (5, 1), (6, 1), (7, 1)]

black\_pieces *=* ['rook', 'knight', 'bishop', 'king', 'queen', 'bishop', 'knight', 'rook',

    'pawn', 'pawn', 'pawn', 'pawn', 'pawn', 'pawn', 'pawn', 'pawn']

black\_locations *=* [(0, 7), (1, 7), (2, 7), (3, 7), (4, 7), (5, 7), (6, 7), (7, 7),

       (0, 6), (1, 6), (2, 6), (3, 6), (4, 6), (5, 6), (6, 6), (7, 6)]

captured\_pieces\_white *=* []

captured\_pieces\_black *=* []

*# 0 - whites turn no selection: 1-whites turn piece selected:*

*# 2- black turn no selection, 3 - black turn piece selected*

turn\_step *=* 0

selection *=* 100

valid\_moves *=* []

Part 2: loading in the images. The rest of their descriptions.

*# load in game piece images*

black\_queen *=* pygame.image.load(*r*'C:\Users\benjy\OneDrive\Desktop\py\_pro\black queen.png')

black\_queen *=* pygame.transform.scale(black\_queen, (80, 80))

black\_queen\_small *=* pygame.transform.scale(black\_queen, (45, 45))

black\_king *=* pygame.image.load(*r*'C:\Users\benjy\OneDrive\Desktop\py\_pro\black king.png')

black\_king *=* pygame.transform.scale(black\_king, (80, 80))

black\_king\_small *=* pygame.transform.scale(black\_king, (45, 45))

black\_bishop *=* pygame.image.load(*r*'C:\Users\benjy\OneDrive\Desktop\py\_pro\black bishop.png')

black\_bishop *=* pygame.transform.scale(black\_bishop, (80, 80))

black\_bishop\_small *=* pygame.transform.scale(black\_bishop, (45, 45))

black\_knight *=* pygame.image.load(*r*'C:\Users\benjy\OneDrive\Desktop\py\_pro\black knight.png')

black\_knight *=* pygame.transform.scale(black\_knight, (80, 80))

black\_knight\_small *=* pygame.transform.scale(black\_knight, (45, 45))

black\_rook *=* pygame.image.load(*r*'C:\Users\benjy\OneDrive\Desktop\py\_pro\black rook.png')

black\_rook *=* pygame.transform.scale(black\_rook, (80, 80))

black\_rook\_small *=* pygame.transform.scale(black\_rook, (45, 45))

black\_pawn *=* pygame.image.load(*r*'C:\Users\benjy\OneDrive\Desktop\py\_pro\black pawn.png')

black\_pawn *=* pygame.transform.scale(black\_pawn, (65, 65))

black\_pawn\_small *=* pygame.transform.scale(black\_pawn, (45, 45))

white\_queen *=* pygame.image.load(*r*'C:\Users\benjy\OneDrive\Desktop\py\_pro\white queen.png')

white\_queen *=* pygame.transform.scale(white\_queen, (80, 80))

white\_queen\_small *=* pygame.transform.scale(white\_queen, (45, 45))

white\_king *=* pygame.image.load(*r*'C:\Users\benjy\OneDrive\Desktop\py\_pro\white king.png')

white\_king *=* pygame.transform.scale(white\_king, (80, 80))

white\_king\_small *=* pygame.transform.scale(white\_king, (45, 45))

white\_bishop *=* pygame.image.load(*r*'C:\Users\benjy\OneDrive\Desktop\py\_pro\white bishop.png')

white\_bishop *=* pygame.transform.scale(white\_bishop, (80, 80))

white\_bishop\_small *=* pygame.transform.scale(white\_bishop, (45, 45))

white\_knight *=* pygame.image.load(*r*'C:\Users\benjy\OneDrive\Desktop\py\_pro\white knight.png')

white\_knight *=* pygame.transform.scale(white\_knight, (80, 80))

white\_knight\_small *=* pygame.transform.scale(white\_knight, (45, 45))

white\_rook *=* pygame.image.load(*r*'C:\Users\benjy\OneDrive\Desktop\py\_pro\white rook.png')

white\_rook *=* pygame.transform.scale(white\_rook, (80, 80))

white\_rook\_small *=* pygame.transform.scale(white\_rook, (45, 45))

white\_pawn *=* pygame.image.load(*r*'C:\Users\benjy\OneDrive\Desktop\py\_pro\white pawn.png')

white\_pawn *=* pygame.transform.scale(white\_pawn, (65, 65))

white\_pawn\_small *=* pygame.transform.scale(white\_pawn, (45, 45))

white\_images *=* [white\_pawn, white\_queen, white\_king, white\_knight,

white\_rook, white\_bishop]

small\_white\_images *=* [white\_pawn\_small, white\_queen\_small, white\_king\_small,

white\_knight\_small, white\_rook\_small, white\_bishop\_small]

black\_images *=* [black\_pawn, black\_queen, black\_king, black\_knight,

black\_rook, black\_bishop]

small\_black\_images *=* [black\_pawn\_small, black\_queen\_small, black\_king\_small,

black\_knight\_small, black\_rook\_small, black\_bishop\_small]

piece\_list *=* ['pawn', 'queen', 'king', 'knight', 'rook', 'bishop']

counter *=* 0

winner *=* ''

game\_over *=* False

Part 3: Drawing the board dimensions and details.

*def* *draw\_board*():

*for* i *in* range(32):

        column *=* i *%* 4

        row *=* i *//* 4

*if* row *%* 2 *==* 0:

            pygame.draw.rect(screen, 'light gray', [600 *-* (column *\** 200), row *\** 100, 100, 100])

*else*:

            pygame.draw.rect(screen, 'light gray', [700 *-* (column *\** 200), row *\** 100, 100, 100])

        pygame.draw.rect(screen, 'gray', [0, 800, WIDTH, 100])

        pygame.draw.rect(screen, 'gold', [0, 800, WIDTH, 100], 5)

        pygame.draw.rect(screen, '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!']

        screen.blit(big\_font.render(status\_text[turn\_step], True, 'black'), (20, 820))

*for* i *in* range(9):

            pygame.draw.line(screen, 'black', (0, 100 *\** i), (800, 100 *\** i), 2)

            pygame.draw.line(screen, 'black', (100 *\** i, 0), (100 *\** i, 800), 2)

        screen.blit(medium\_font.render('FORFEIT', True, 'black'), (810, 830))

Part 4: drawing the pieces on the board. The details have been already given.

*# draw pieces on the board*

*def* *draw\_pieces*():

*for* i *in* range(len(white\_pieces)):

        index *=* piece\_list.index(white\_pieces[i])

*if* white\_pieces[i] *==* 'pawn':

            screen.blit(white\_pawn, (white\_locations[i][0] *\** 100 *+* 22, white\_locations[i][1] *\** 100 *+* 30))

*else*:

            screen.blit(white\_images[index], (white\_locations[i][0] *\** 100 *+* 10, white\_locations[i][1] *\** 100 *+* 10))

*if* turn\_step *<* 2:

*if* selection *==* i:

                pygame.draw.rect(screen, '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':

            screen.blit(black\_pawn, (black\_locations[i][0] *\** 100 *+* 22, black\_locations[i][1] *\** 100 *+* 30))

*else*:

            screen.blit(black\_images[index], (black\_locations[i][0] *\** 100 *+* 10, black\_locations[i][1] *\** 100 *+* 10))

*if* turn\_step *>=* 2:

*if* selection *==* i:

                pygame.draw.rect(screen, 'blue', [black\_locations[i][0] *\** 100 *+* 1, black\_locations[i][1] *\** 100 *+* 1,

                                                  100, 100], 2)

Part 5: This is the rules each piece operates under. This is a long section/part.

*# checking all valid pieces on the board*

*def* *check\_options*(pieces, locations, turn):

    moves\_list *=* []

    all\_moves\_list *=* []

*for* i *in* range((len(pieces))):  *# Iterate through each piece*

        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 *==* 'bishop':

            moves\_list *=* check\_bishop(location, turn)

*elif* piece *==* 'knight':

            moves\_list *=* check\_knight(location, turn)

*elif* piece *==* 'king':

            moves\_list *=* check\_king(location, turn)

*elif* piece *==* 'queen':

            moves\_list *=* check\_queen(location, turn)

        all\_moves\_list.append(moves\_list)  *# Add moves for each piece to the list*

*return* all\_moves\_list

*## part lays out the valid moves for each piece. How they can move*

*# checking all valid moves for king*

*def* *check\_king*(position, color):

    moves\_list *=* []

*if* color *==* 'white':

        enemies\_list *=* black\_locations

        friends\_list *=* white\_locations

*else*:

        friends\_list *=* black\_locations

        enemies\_list *=* white\_locations

*# 8 squares to check for kings, they can go one square any direction*

    targets *=* [(1, 0), (1, 1), (1, *-*1), (*-*1, 0), (*-*1, 1), (*-*1, *-*1), (0, 1), (0, *-*1)]

*for* i *in* range(8):

        target *=* (position[0] *+* targets[i][0], position[1] *+* targets[i][1])

*if* target *not* *in* friends\_list *and* 0 *<=* target[0] *<=* 7 *and* 0 *<=* target[1] *<=* 7:

            moves\_list.append(target)

*return* moves\_list

*# check queen valid moves*

*def* *check\_queen*(position, color):

    moves\_list *=* check\_bishop(position, color)

    second\_list *=* check\_rook(position, color)

*for* i *in* range(len(second\_list)):

        moves\_list.append(second\_list[i])

*return* moves\_list

*# checking all valid moves for pawn*

*def* *check\_pawn*(position, color):

    moves\_list *=* []

*if* color *==* 'white':

*# Move forward one step*

*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))

*# Move forward two steps if on starting position*

*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))

*# Capture diagonally right*

*if* (position[0] *+* 1, position[1] *+* 1) *in* black\_locations:

            moves\_list.append((position[0] *+* 1, position[1] *+* 1))

*# Capture diagonally left*

*if* (position[0] *-* 1, position[1] *+* 1) *in* black\_locations:

            moves\_list.append((position[0] *-* 1, position[1] *+* 1))

*else*:  *# color == 'black'*

*# Move forward one step*

*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))

*# Move forward two steps if on starting position*

*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))

*# Capture diagonally right*

*if* (position[0] *+* 1, position[1] *-* 1) *in* white\_locations:

            moves\_list.append((position[0] *+* 1, position[1] *-* 1))

*# Capture diagonally left*

*if* (position[0] *-* 1, position[1] *-* 1) *in* white\_locations:

            moves\_list.append((position[0] *-* 1, position[1] *-* 1))

*return* moves\_list

*# valide moves for bishop*

*def* *check\_bishop*(position, color):

    moves\_list *=* []

*if* color *==* 'white':

        enemies\_list *=* black\_locations

        friends\_list *=* white\_locations

*else*:

        friends\_list *=* black\_locations

        enemies\_list *=* white\_locations

*for* i *in* range(4):  *# up-right, up-left, down-right, down-left*

        path *=* True

        chain *=* 1

*if* i *==* 0:

            x *=* 1

            y *=* *-*1

*elif* i *==* 1:

            x *=* *-*1

            y *=* *-*1

*elif* i *==* 2:

            x *=* 1

            y *=* 1

*else*:

            x *=* *-*1

            y *=* 1

*while* path:

*if* (position[0] *+* (chain *\** x), position[1] *+* (chain *\** y)) *not* *in* friends\_list *and* \

                    0 *<=* position[0] *+* (chain *\** x) *<=* 7 *and* 0 *<=* position[1] *+* (chain *\** y) *<=* 7:

                moves\_list.append((position[0] *+* (chain *\** x), position[1] *+* (chain *\** y)))

*if* (position[0] *+* (chain *\** x), position[1] *+* (chain *\** y)) *in* enemies\_list:

                    path *=* False

                chain *+=* 1

*else*:

                path *=* False

*return* moves\_list

*# checking all valid moves for rook*

*def* *check\_rook*(position, color):

        moves\_list *=* []

*if* color *==* 'white':

            enemies\_list *=* black\_locations

            friends\_list *=* white\_locations

*else*:

            friends\_list *=* black\_locations

            enemies\_list *=* white\_locations

*for* i *in* range(4):  *# down, up, right, left*

            path *=* True

            chain *=* 1

*if* i *==* 0:

                x *=* 0

                y *=* 1

*elif* i *==* 1:

                x *=* 0

                y *=* *-*1

*elif* i *==* 2:

                x *=* 1

                y *=* 0

*else*:

                x *=* *-*1

                y *=* 0

*while* path:

*if* (position[0] *+* (chain *\** x), position[1] *+* (chain *\** y)) *not* *in* friends\_list *and* \

                    0 *<=* position[0] *+* (chain *\** x) *<=* 7 *and* 0 *<=* position[1] *+* (chain *\** y) *<=* 7:

                    moves\_list.append((position[0] *+* (chain *\** x), position[1] *+* (chain *\** y)))

*if* (position[0] *+* (chain *\** x), position[1] *+* (chain *\** y)) *in* enemies\_list:

                        path *=* False

                    chain *+=* 1

*else*:

                    path *=* False

*return* moves\_list

*# check valid knight moves*

*def* *check\_knight*(position, color):

    moves\_list *=* []

*if* color *==* 'white':

        enemies\_list *=* black\_locations

        friends\_list *=* white\_locations

*else*:

        friends\_list *=* black\_locations

        enemies\_list *=* white\_locations

*# 8 squares to check for knights, they can go two squares in one direction and one in another*

    targets *=* [(1, 2), (1, *-*2), (2, 1), (2, *-*1), (*-*1, 2), (*-*1, *-*2), (*-*2, 1), (*-*2, *-*1)]

*for* i *in* range(8):

        target *=* (position[0] *+* targets[i][0], position[1] *+* targets[i][1])

*if* target *not* *in* friends\_list *and* 0 *<=* target[0] *<=* 7 *and* 0 *<=* target[1] *<=* 7:

            moves\_list.append(target)

*return* moves\_list

*# check for valid moves for just selected piece*

*def* *check\_valid\_moves*():

*if* turn\_step *<* 2:

        options\_list *=* white\_options

*else*:

        options\_list *=* black\_options

    valid\_options *=* options\_list[selection]

*return* valid\_options

*# draw valid moves on screen*

*def* *draw\_valid*(moves):

*if* turn\_step *<* 2:

        color *=* 'red'

*else*:

        color *=* 'blue'

*for* i *in* range(len(moves)):

        pygame.draw.circle(screen, color, (moves[i][0] *\** 100 *+* 50, moves[i][1] *\** 100 *+* 50), 5)

*# draw captured pieces on side of screen*

*def* *draw\_captured*():

*for* i *in* range(len(captured\_pieces\_white)):

        captured\_piece *=* captured\_pieces\_white[i]

        index *=* piece\_list.index(captured\_piece)

        screen.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)

        screen.blit(small\_white\_images[index], (925, 5 *+* 50 *\** i))

*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:

                        pygame.draw.rect(screen, '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:

                        pygame.draw.rect(screen, 'dark blue', [black\_locations[king\_index][0] *\** 100 *+* 1,

                                                               black\_locations[king\_index][1] *\** 100 *+* 1, 100, 100], 5)

*def* *draw\_game\_over*():

    pygame.draw.rect(screen, 'black', [200, 200, 400, 70])

    screen.blit(font.render(*f*'{winner} won the game!', True, 'white'), (210, 210))

    screen.blit(font.render(*f*'Press ENTER to Restart!', True, 'white'), (210, 240))

black\_options *=* check\_options(black\_pieces, black\_locations, 'black')

white\_options *=* check\_options(white\_pieces, white\_locations, 'white')

Part 6: The main Game Loop

run *=* True

*while* run:

    timer.tick(fps)

*if* counter *<* 30:

        counter *+=* 1

*else*:

        counter *=* 0

    screen.fill('dark gray')

    draw\_board()

    draw\_pieces()

    draw\_captured()

    draw\_check()

*if* selection *!=* 100:

        valid\_moves *=* check\_valid\_moves()

        draw\_valid(valid\_moves)

*# event handling*

*for* event *in* pygame.event.get():

*if* event.type *==* pygame.QUIT:

            run *=* False

*if* event.type *==* pygame.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)

*if* turn\_step *<=* 1:

*if* click\_coords *==* (8, 8) *or* click\_coords *==* (9, 8):

                    winner *=* 'black'

*if* click\_coords *in* white\_locations:

                    selection *=* white\_locations.index(click\_coords)

*if* turn\_step *==* 0:

                        turn\_step *=* 1

*if* click\_coords *in* valid\_moves *and* selection *!=* 100:

                    white\_locations[selection] *=* click\_coords

*if* click\_coords *in* black\_locations:

                        black\_piece *=* black\_locations.index(click\_coords)

                        captured\_pieces\_white.append(black\_pieces[black\_piece])

*if* black\_pieces[black\_piece] *==* 'king':

                            winner *=* 'white'

                        black\_pieces.pop(black\_piece)

                        black\_locations.pop(black\_piece)

                    black\_options *=* check\_options(black\_pieces, black\_locations, 'black')

                    white\_options *=* check\_options(white\_pieces, white\_locations, 'white')

                    turn\_step *=* 2

                    selection *=* 100

                    valid\_moves *=* []

*if* turn\_step *>* 1:

*if* click\_coords *==* (8, 8) *or* click\_coords *==* (9, 8):

                    winner *=* 'white'

*if* click\_coords *in* black\_locations:

                    selection *=* black\_locations.index(click\_coords)

*if* turn\_step *==* 2:

                        turn\_step *=* 3

*if* click\_coords *in* valid\_moves *and* selection *!=* 100:

                    black\_locations[selection] *=* click\_coords

*if* click\_coords *in* white\_locations:

                        white\_piece *=* white\_locations.index(click\_coords)

                        captured\_pieces\_black.append(white\_pieces[white\_piece])

*if* white\_pieces[white\_piece] *==* 'king':

                            winner *=* 'black'

                        white\_pieces.pop(white\_piece)

                        white\_locations.pop(white\_piece)

                    black\_options *=* check\_options(black\_pieces, black\_locations, 'black')

                    white\_options *=* check\_options(white\_pieces, white\_locations, 'white')

                    turn\_step *=* 0

                    selection *=* 100

                    valid\_moves *=* []

*if* event.type *==* pygame.KEYDOWN *and* game\_over:

*if* event.key *==* pygame.K\_RETURN:

                game\_over *=* False

                winner *=* ''

                white\_pieces *=* ['rook', 'knight', 'bishop', 'king', 'queen', 'bishop', 'knight', 'rook',

                                'pawn', 'pawn', 'pawn', 'pawn', 'pawn', 'pawn', 'pawn', 'pawn']

                white\_locations *=* [(0, 0), (1, 0), (2, 0), (3, 0), (4, 0), (5, 0), (6, 0), (7, 0),

                                   (0, 1), (1, 1), (2, 1), (3, 1), (4, 1), (5, 1), (6, 1), (7, 1)]

                black\_pieces *=* ['rook', 'knight', 'bishop', 'king', 'queen', 'bishop', 'knight', 'rook',

                                'pawn', 'pawn', 'pawn', 'pawn', 'pawn', 'pawn', 'pawn', 'pawn']

                black\_locations *=* [(0, 7), (1, 7), (2, 7), (3, 7), (4, 7), (5, 7), (6, 7), (7, 7),

                                   (0, 6), (1, 6), (2, 6), (3, 6), (4, 6), (5, 6), (6, 6), (7, 6)]

                captured\_pieces\_white *=* []

                captured\_pieces\_black *=* []

                turn\_step *=* 0

                selection *=* 100

                valid\_moves *=* []

                black\_options *=* check\_options(black\_pieces, black\_locations, 'black')

                white\_options *=* check\_options(white\_pieces, white\_locations, 'white')

*if* winner *!=* '':

        game\_over *=* True

        draw\_game\_over()

Part 7: the end of the script.

    pygame.display.flip()

pygame.quit()

**Changes that will need to happen. This will be done by section. Each section will have all important changes recorded within each part below in bullet point format.**

Part 1: the initial setup, defining the concept of a board, as well as fonts, frame rates, variables and images.

Part 2: loading in the images. The rest of their descriptions.

Part 3: Drawing the board dimensions and details.

Part 4: drawing the pieces on the board. The details have been already given.

Part 5: This is the rules each piece operates under. This is a long section/part.

Errors found:

Part 6: The main Game Loop

Part 7: the end of the script.