Compsci Final Project Report
Project name: ChessHD
Team Members: Maxwell DeRienze

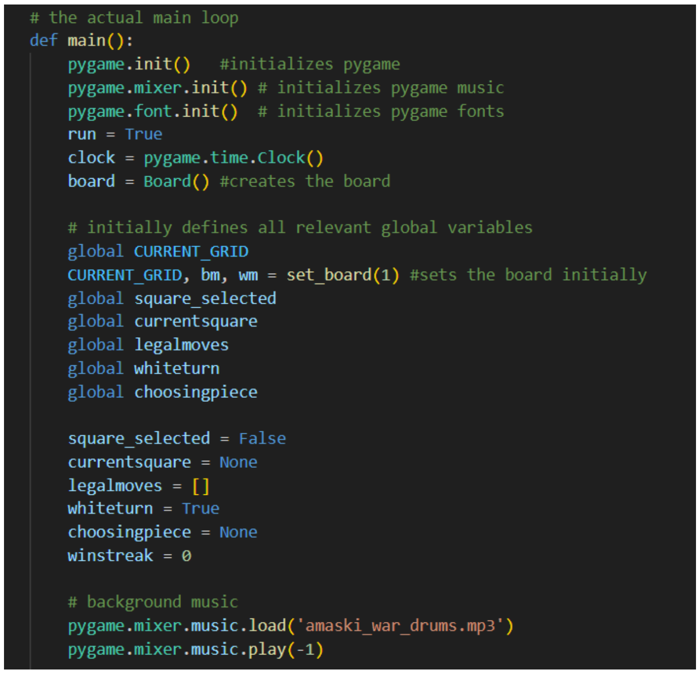
Goal + Introduction

My proposal for the project outlined that I was making a modified version of chess. My goal was to have a rouge-like style version of chess in which the pieces on each side were randomly generated, and after winning or losing that game you'd begin another one with a different difficulty. While s, the final project achieves all these goals.

Project Structure

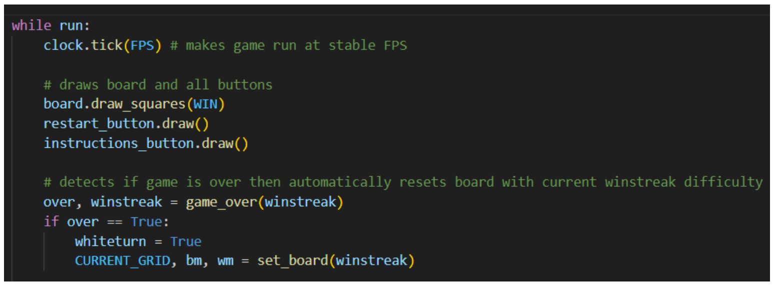
The entire project was made in python along with some extra modules, most notably pygame, along with copy and random. The first thing I did was follow a checkers tutorial in order to get the board displayed, along with establishing some constants.[1]

The first thing the main loop does is initialize pygame along with specific parts of it that I had used (fonts and music), as well as initialize all the global variables. It will then run the set_board() function which will generate random pieces. Then it plays the background music.



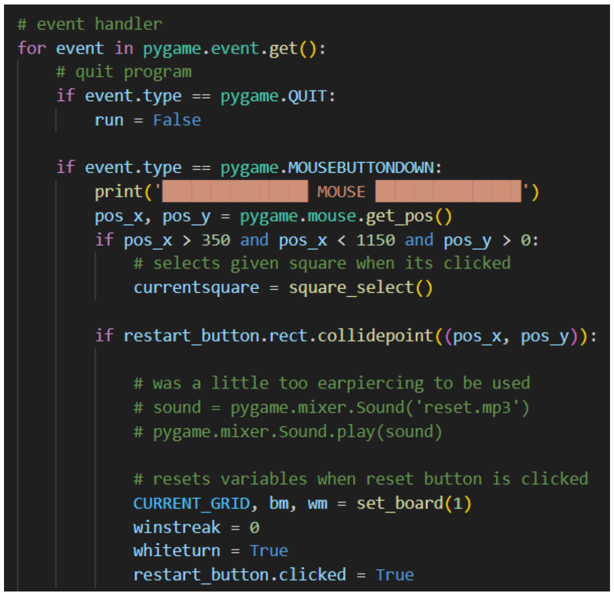
```
# the actual main loop
def main():
    pygame.init()   #initializes pygame
    pygame.mixer.init() # initializes pygame music
    pygame.font.init()  # initializes pygame fonts
    run = True
    clock = pygame.time.Clock()
    board = Board() #creates the board

    # initially defines all relevant global variables
    global CURRENT_GRID
    CURRENT_GRID, bm, wm = set_board(1) #sets the board initially
    global square_selected
    global currentsquare
    global legalmoves
    global whiteturn
    global choosingpiece

    square_selected = False
    currentsquare = None
    legalmoves = []
    whiteturn = True
    choosingpiece = None
    winstreak = 0

    # background music
    pygame.mixer.music.load('amaski_war_drums.mp3')
    pygame.mixer.music.play(-1)
```

After this is the While run loop. This is the actual core of the game and what we see, since pygame.display.update() is run every iteration. The first thing done in this loop is having a clock set to tick at 30 FPS, keeping the game running at the same speed on different devices. After this the board.draw_squares() method is called to actually display the squares. Then 2 objects part of the Button class are drawn onto the screen. After this there's a function called game_over() that detects when the game has ended (by constantly checking if either king has been killed). When this happens a message is displayed, a the winstreak is changed according to whether or not the player has won the game, and the set_board() function is called using winstreak to determine how many pieces to generate next time.



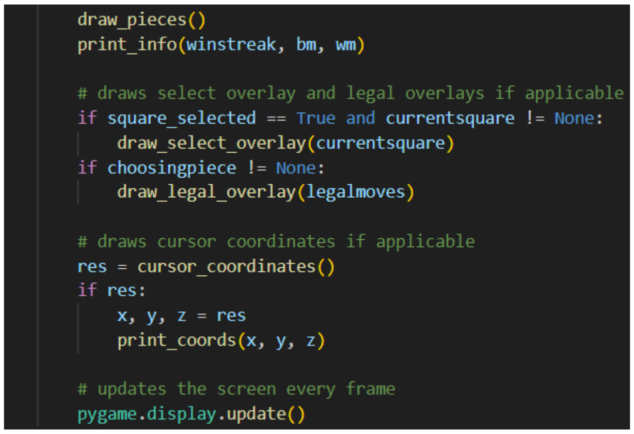
```python
while run:
    clock.tick(FPS) # makes game run at stable FPS

    # draws board and all buttons
    board.draw_squares(WIN)
    restart_button.draw()
    instructions_button.draw()

    # detects if game is over then automatically resets board with current winstreak difficulty
    over, winstreak = game_over(winstreak)
    if over == True:
        whiteturn = True
        CURRENT_GRID, bm, wm = set_board(winstreak)
```

After this is the event handler, which handles all input from the player. First and most important is quitting the program, which is done by clicking the X of the window which sets run to False. After this it will detect if the player has clicked on the grid and using the square_select() function will return the square clicked on. This function is how the player moves or interacts with each piece. If the reset button is pressed the board along with other variables are reset.



```python
# event handler
for event in pygame.event.get():
    # quit program
    if event.type == pygame.QUIT:
        run = False

    if event.type == pygame.MOUSEBUTTONDOWN:
        print('███████████████ MOUSE ███████████████')
        pos_x, pos_y = pygame.mouse.get_pos()
        if pos_x > 350 and pos_x < 1150 and pos_y > 0:
            # selects given square when its clicked
            currentsquare = square_select()

        if restart_button.rect.collidepoint((pos_x, pos_y)):

            # was a little too earpiercing to be used
            # sound = pygame.mixer.Sound('reset.mp3')
            # pygame.mixer.Sound.play(sound)

            # resets variables when reset button is clicked
            CURRENT_GRID, bm, wm = set_board(1)
            winstreak = 0
            whiteturn = True
            restart_button.clicked = True
```

After this the pieces are drawn, and the information that's seen on the right is printed, as well as the cursor's square, the select and legal move overlays if they exist.



```python
    draw_pieces()
    print_info(winstreak, bm, wm)

    # draws select overlay and legal overlays if applicable
    if square_selected == True and currentsquare != None:
        draw_select_overlay(currentsquare)
    if choosingpiece != None:
        draw_legal_overlay(legalmoves)

    # draws cursor coordinates if applicable
    res = cursor_coordinates()
    if res:
        x, y, z = res
        print_coords(x, y, z)

    # updates the screen every frame
    pygame.display.update()
```

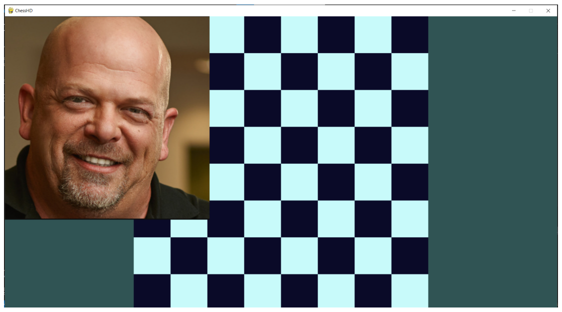
All assets are stored and referred to locally within the ChessHD folder. The folder consists of 4 used python files, 6 used audio files, and 18 used image files.

Known bugs & Limitations:

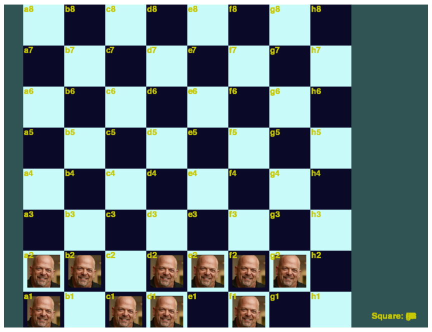
The first one is the most apparent, but the player must control both the white and black squares. Ideally it would've been the computer playing with its own chess AI to detect better moves. However any sort of implementation of an AI would've taken at least another week of development. While the same experience can be gained currently, it's relying on the honor system. The next biggest limitation is a lack of player customization and interface. The game currently is very bare bones, the only thing with functionality aside from the pieces is the restart button. Ideally there would be a full UI system in which the player could customize certain aspects like volume, pausing, etc.

BUGS

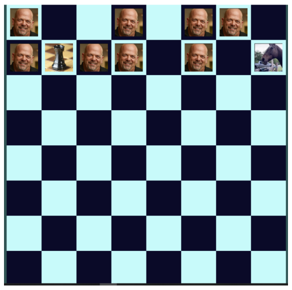
Earliest known screenshot of ChessHD:

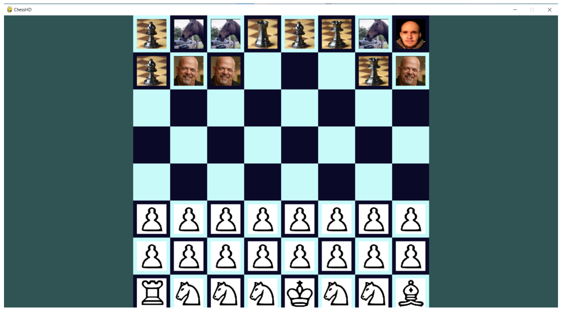


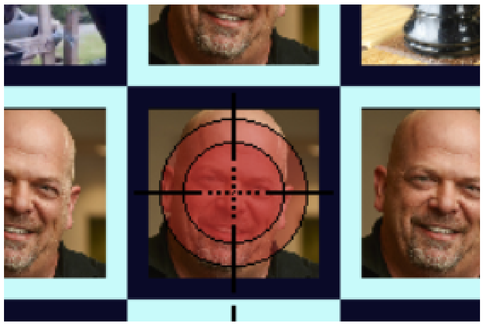
I wasn't resizing images

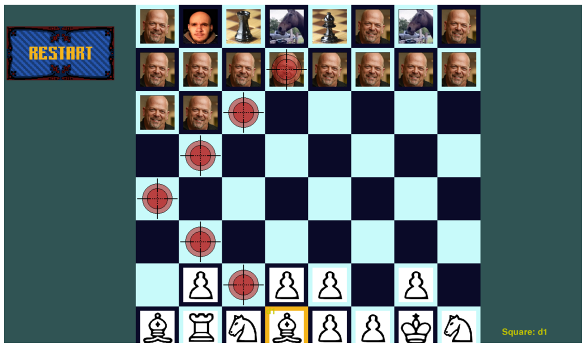


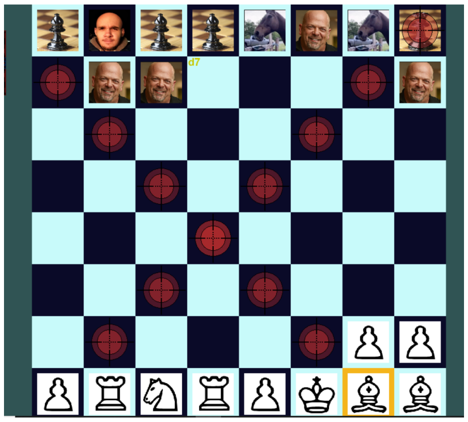
this is what happens when the board isn't redrawn each frame


before the implementation of pieces filling a row completely before spawning on the next one

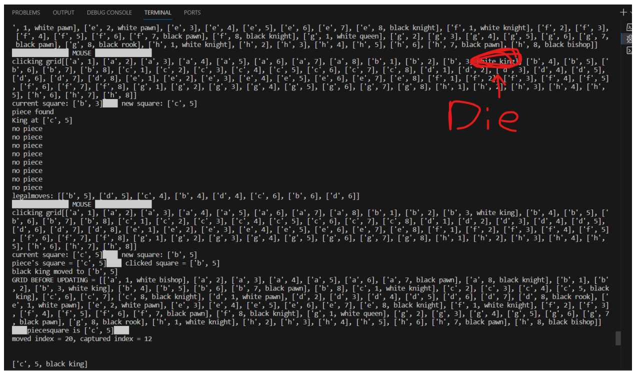

first implementation of the white side


first bullseye over Rick Harrison as i begin to implement movement





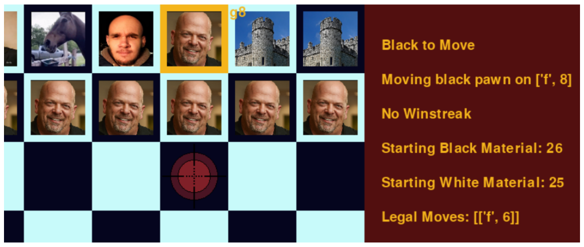
Hard to tell but the Bishop's movement kept giving me problems, in the first image you can see the bishop bounces off walls. And in the second image it for some reason picks a square that it can go in all 4 directions from.



```
', 1, white pawn], ['e', 2, white pawn], ['e', 3], ['e', 4], ['e', 5], ['e', 6], ['e', 7], ['e', 8, black knight], ['f', 1, white knight], ['f', 2], ['f', 3],
 ['f', 4], ['f', 5], ['f', 6], ['f', 7, black pawn], ['f', 8, black knight], ['g', 1, white queen], ['g', 2], ['g', 3], ['g', 4], ['g', 5], ['g', 6], ['g', 7,
 black pawn], ['g', 8, black rook], ['h', 1, white knight], ['h', 2], ['h', 3], ['h', 4], ['h', 5], ['h', 6], ['h', 7, black pawn], ['h', 8, black bishop]]
                MOUSE
clicking grid[['a', 1], ['a', 2], ['a', 3], ['a', 4], ['a', 5], ['a', 6], ['a', 7], ['a', 8], ['b', 1], ['b', 2], ['b', 3, white king], ['b', 4], ['b', 5], ['
b', 6], ['b', 7], ['b', 8], ['c', 1], ['c', 2], ['c', 3], ['c', 4], ['c', 5], ['c', 6], ['c', 7], ['c', 8], ['d', 1], ['d', 2], ['d', 3], ['d', 4], ['d', 5],
['d', 6], ['d', 7], ['d', 8], ['e', 1], ['e', 2], ['e', 3], ['e', 4], ['e', 5], ['e', 6], ['e', 7], ['e', 8], ['f', 1], ['f', 2], ['f', 3], ['f', 4], ['f', 5]
, ['f', 6], ['f', 7], ['f', 8], ['g', 1], ['g', 2], ['g', 3], ['g', 4], ['g', 5], ['g', 6], ['g', 7], ['g', 8], ['h', 1], ['h', 2], ['h', 3], ['h', 4], ['h',
5], ['h', 6], ['h', 7], ['h', 8]]
current square: ['b', 3]     new square: ['c', 5]
piece found
King at ['c', 5]
no piece
no piece
no piece
no piece
no piece
no piece
no piece
no piece
legalmoves: [['b', 5], ['d', 5], ['c', 4], ['b', 4], ['d', 4], ['c', 6], ['b', 6], ['d', 6]]
                MOUSE
clicking grid[['a', 1], ['a', 2], ['a', 3], ['a', 4], ['a', 5], ['a', 6], ['a', 7], ['a', 8], ['b', 1], ['b', 2], ['b', 3, white king], ['b', 4], ['b', 5], ['
b', 6], ['b', 7], ['b', 8], ['c', 1], ['c', 2], ['c', 3], ['c', 4], ['c', 5], ['c', 6], ['c', 7], ['c', 8], ['d', 1], ['d', 2], ['d', 3], ['d', 4], ['d', 5],
['d', 6], ['d', 7], ['d', 8], ['e', 1], ['e', 2], ['e', 3], ['e', 4], ['e', 5], ['e', 6], ['e', 7], ['e', 8], ['f', 1], ['f', 2], ['f', 3], ['f', 4], ['f', 5]
, ['f', 6], ['f', 7], ['f', 8], ['g', 1], ['g', 2], ['g', 3], ['g', 4], ['g', 5], ['g', 6], ['g', 7], ['g', 8], ['h', 1], ['h', 2], ['h', 3], ['h', 4], ['h',
5], ['h', 6], ['h', 7], ['h', 8]]
current square: ['c', 5]     new square: ['b', 5]
piece's square = ['c', 5]     clicked square = ['b', 5]
black king moved to ['b', 5]
GRID BEFORE UPDATING = [['a', 1, white bishop], ['a', 2], ['a', 3], ['a', 4], ['a', 5], ['a', 6], ['a', 7, black pawn], ['a', 8, black knight], ['b', 1], ['b'
, 2], ['b', 3, white king], ['b', 4], ['b', 5], ['b', 6], ['b', 7, black pawn], ['b', 8], ['c', 1, white knight], ['c', 2], ['c', 3], ['c', 4], ['c', 5, black
 king], ['c', 6], ['c', 7], ['c', 8, black knight], ['d', 1, white pawn], ['d', 2], ['d', 3], ['d', 4], ['d', 5], ['d', 6], ['d', 7], ['d', 8, black rook], ['
e', 1, white pawn], ['e', 2, white pawn], ['e', 3], ['e', 4], ['e', 5], ['e', 6], ['e', 7], ['e', 8, black knight], ['f', 1, white knight], ['f', 2], ['f', 3]
, ['f', 4], ['f', 5], ['f', 6], ['f', 7, black pawn], ['f', 8, black knight], ['g', 1, white queen], ['g', 2], ['g', 3], ['g', 4], ['g', 5], ['g', 6], ['g', 7
, black pawn], ['g', 8, black rook], ['h', 1, white knight], ['h', 2], ['h', 3], ['h', 4], ['h', 5], ['h', 6], ['h', 7, black pawn], ['h', 8, black bishop]]
    piecesquare is ['c', 5]
moved index = 20, captured index = 12


['c', 5, black king]
```

For a long time I was struggling with a bug where eventually attempting to move a piece would crash the game. This was because when I was defining a new variable by referencing the BLANKGRID, it would end up changing the actual BLANKGRID, which needed to stay blank for other parts of the program to work. This happened randomly however, so I still don't know why it changed BLANKGRID in some cases but not others.

The 2 biggest bugs that currently still reside in the game both involve the pawns. Firstly I did not have time to implement promotion, so they are essentially useless when they hit the last ranks.



Black to Move

Moving black pawn on ['f', 8]

No Winstreak

Starting Black Material: 26

Starting White Material: 25

Legal Moves: [['f', 6]]

However, while it would've been easy to fix, I did not have the time. We can see that on its first move, the pawn is able to jump over other pieces, unlike what it should do.

One of the major bugs that I did solve beforehand was with setting the board. After a while the win streak will be so high that there will not be enough spots to generate all the pieces on the black side, earlier the game would just crash (for reference look at gameplay_crash.mp4 [warning all videos have loud sounds]). But now, it will simply disregard each extra piece. Allowing the player to get past winstreaks of 5 or 6.

I've been a part of all 3 game jams as well as the Expo held by Clark IGDA this semester and while I don't regret any part of those, in each of those instances I was on my team as an artist. It is something I enjoy, however I want to be able to get good at programming in order to do my own projects. This is kind of what I wanted to achieve with this project. I've tried making a game before as a final project however it didn't go too well because I had not learned enough of the language to properly do anything (i didn't even know about for loops at the time). I feel like I've properly exceeded that goal through this project. I had to learn how to use pygame, classes and inheritance, global variables, references, etc. And now I have a fully interactable game that I've been the sole developer for. Granted it's not exactly at a stage where anyone can simply download and play, but I’m still proud of where I’ve gotten in the project, and the progress I’ve made both with the project and the class.

All Tutorials references:
https://www.youtube.com/watch?v=vnd3RfeG3NM
https://www.youtube.com/watch?v=4TfZjhw0J-8
https://www.youtube.com/watch?v=Lw8ndzcWFFw
https://www.youtube.com/watch?v=kO8Ae3bz0kc
https://www.digitalocean.com/community/tutorials/understanding-class-inheritance-in-python-3
https://pybit.es/articles/python-subclasses/
https://stackoverflow.com/questions/576169/understanding-python-super-with-init-methods