|  |  |  |
| --- | --- | --- |
| **Hrs** | **Date** | **Work Completed / Discussion** |
| 0.75 | Jan 16 | I created basic graphics for the game in PowerPoint. I struggled to find high-quality images of chess pieces. Once obtaining them, I used recoloring and clipart to create the different versions and alterations that appear in the game. |
| 2.5 | Jan 17 | I began formulating the movement mechanisms for regular chess pieces. I finished and debugged the pawn and I’m working on the others. Next, I will create the GUI and then add my game’s special enhancements.  My biggest challenge was organizing my program to avoid redundancy and make it easy to edit and debug in the future. For instance, I tried several different ways to store the location of pieces on the board. At first, I made each square a class-defined object, but eventually settled on organizing the pieces into an array. Each piece also stores its own location. |
| 1 | Jan 20 | I completed and debugged the movement mechanics for the rest of the pieces. Each piece required a slightly different technique, but I figured them all out. My only concern is that the movement mechanics may be incompatible with other elements of the program (which will be added later). |
| 1.25 | Feb 6 | I established the turn-order mechanism, in which one color moves after the other. I added a log to keep track of the moves made in a game. I also added the capability for special moves, such as pawn hops and castling.  I am making some minor modifications to the rules of chess. For instance, I am making it optional to move, and removing the concepts of check, checkmate and stalemate. The game will end as soon as one side’s king is captured (yes, you can blunder your king). |
| 2.25 | Feb 9 | I created a function that, after every turn, checks to ensure that both kings are still on the board. Then I created the actual (graphical) game board and pieces. Currently, they can only be moved by typing moves into the terminal. This part was uncharted territory for me, so there was a bit of a learning curve, but I managed to figure it out quickly.  I ran into a problem, however. The Pygame display grid has its origin at the upper-left corner, while a chess board has its origin at the lower-left, so currently the board appears inverted. The conversion needs to be handled carefully in order to avoid confusion – I will do this next time. |
| 4 | Apr 27 | Accounting for the difference in coordinate grid systems between chess and Python was a simple task, but it is already causing some confusion. For simplicity, the program will primarily use the chess coordinates, only switching to Python coordinates when manipulating the display.  I also spent significant time with the drag-and-drop interface, which required a partial rehaul of the gameplay mechanism. I reorganized the images of the chess pieces into a dictionary, which made accessing them far easier. I also learned how to better use and manipulate the gameplay loop. Surprisingly, the drag-and-drop worked well after very little debugging. |
| 2 | May 3 | I created the advanced mechanics and the buttons and graphics necessary to implement them. PowerBoxes will randomly spawn throughout the chessboard, which players can activate by capturing them with their pieces. They come in three different rarities (Regular, Rare, Legendary); the rarer the box, the more powerful its effect. Upon capturing, a player will be given a choice between two Powers of the appropriate rarity. |
| 2.5 | May 5 | I created the game clock and Pause, Restart, Help and Quit buttons. The greatest challenge was the clock, which involved numerous functions that I had not used before. I learned how to create custom events, setting a “Tick” event to advance the clock ten times per second. I also learned how to display text on the screen. It took me some time to position the clock and figure out how to use the font I desired. |
| 0.75 | May 7 | Unfortunately, I found several bugs that had to be fixed before I can continue. The main problem was the whenever I picked up a piece, the clock stopped running. When I fixed this, several new bugs appeared, such as unusual blinks and flickers. In retrospect, this was a good exercise in bug fixing. I had to first study and understand the specifics of the problem, and then comb through the code to figure out what was causing it. |
| 9 | May 18-21 | First, I reorganized my program a little to make it more readable. For instance, I turned some repetitive procedures (such as changing whose turn it is) into defined functions.  Then, I created a mechanism for promotions that allows players to select which piece they are promoting into. This took significantly longer than expected, requiring me to reconfigure certain parts of my program. I needed to get the main gameplay loop to interact with other code. I designed a dictionary to store the location and function of all on-screen buttons, which helped significantly. Every time the mouse is pressed, the program can loop through the button dictionary to see if any buttons were activated.  Finally, I added PowerBoxes. I realized that programming each different Power could take dozens more hours, so I opted for a simplified version. I developed the mechanisms for spawning PowerBoxes, storing their locations, and checking when they’re tripped. Currently, if a player activates a Box, then they are simply granted the opportunity to move again. |

**Total Hours: 26**

**What I could’ve done better:** I decided to start with a simplified version of the final product and gradually add complexity. This was probably a misguided approach, since every increasing layer of complexity required increasingly substantial modifications to the existing code. The program quickly grew disorganized and unwieldy, and I even forgot how some parts of it work. I tried to move some of my function and class definitions to external modules (which could be imported into the main executable file), but realized that this would require a complete overhaul. I probably could have written more efficient or comprehendible code (if I knew how), but my biggest problem was failing to plan and outline. If I was going to write a 5000-word paper, I wouldn’t just “wing it;” the same should apply to a 1000-line program. Additionally (much like Elon Musk, whose biography I’m currently reading), I underestimated the difficulty of the task at hand, failing to foresee the countless problems and bugs I would encounter. In engineering, a problem is ALWAYS harder than it looks.

**What I learned:** The main thing I learned is that I have a lot left to learn. If I started this project again from scratch, I could probably finish it in two-thirds or half the time. An experienced programmer could probably do it in three to five hours. Besides not planning, I probably made numerous other “rookie mistakes” that I’m not even aware of. While the project proved more difficult than I expected, I’m confident that, with proper training, I can accomplish much greater tasks. After all, professional software engineers write programs with millions of lines of code. This project provided a valuable glimpse into the world of advanced computer science, which will likely be my college major (or one of them).

**Future work:** I want to keep going with this project, and maybe eventually post it on Github or something, but the deadline passed and so I’m handing it in. I’ll probably continue it on my own over the summer, making it more presentable and adding new features and complexity.