Coles final project reflection:

The part of the project that taught me the most about data structures was the handling of our arraylist of pieces. The first problem was in accessing the arraylist, because it was stored in the ChessComponent but was needed in almost every class including the pieces themselves. We tried making it a static variable, but that did not work, and neither did a static getter class. Instead we had to call setter methods to update copy lists for each piece. The biggest problem with pieces was in our avoidCheck method, where we had to make a move in order to see if it would keep the player in check. To avoid messing up the original pieces, we would create arraylists to store the pieces and reassign pieces. The reason this was so difficult is because the pieces themselves could not be directly copied over to a new arraylist, and other methods that were used in avoidCheck relied on the instance variable pieces, meaning instead of creating a proxy arraylist, we had to edit the pieces arraylist itself and bring it back to its previous state.

Something I learned from our project was how to interact between classes. Normally most of my work is done in just a couple of classes. For the chess code, it was much less clear than ever which processes should be done in which class. For instance, the best place for testing if a king is in check is in the king’s class itself, because to move a king the method to see a spot is in check is necessary, however to test for checkmate, an almost identical situation, we needed to use the ChessComponent because among other reasons it matters whose turn it is to move which is not accessible in King. Other examples are the moves for castling, promotion, and en passant. These three moves are all possible moves by certain pieces, the king for castling and the pawn for the other two, but the methods for them had to go in chessComponent instead of the piece classes because they require certain things such as the previous move, drawing modifications, and changes to the pieces class. On the other hand, a move that is very similar, a pawn going up two on its first move, is actually best placed in the pawn class. All this is to say that I had to struggle with the distribution of information and the connections between classes.

The hardest thing about this project was probably the lack of visual progress for a majority of the time. Kargil and I were dedicated to not wasting time on creating temporary visuals such as printing each position on the board and trying to make moves through the terminal, because they would have to be replaced by very different code and we always knew we would be pressed for time. We started the project by coding where a piece could move, then we moved on to organizing them in an arraylist and updating position and removing captured pieces. Once we finished that, we tried making our own GUI for a starting screen but then shifted into creating a component for the board itself. Once we had the board created, it was not until we implemented reading mouse clicks, calculating squares and movement, and the highlighting of possible moves that we could finally see our work paying off. Of course, that momentous advancement also allowed us to see many flaws in our code that we had to fix before we had an even semi-functional game. The whole time I struggled to imagine the board or how certain interactions would happen, and Kargil and I were prone to drawing out scenarios on paper to determine many factors such as reading a square from a pixel and deciding how far to highlight along a pieces reach. However, I do not regret our choice of order because when we began we had no clue how to start the GUI components, whereas we had a much better idea of the actual gameplay of chess and how we wanted to code that so we got off to a faster start this way.

I have no doubt that the coolest part about our code is the ai player. The seemingly impossible idea of coding a player with intelligence was so far out that going into the project I always expected to just web scrape one from online. However, the challenge of coding an ai was the most fun part about this project, and although the impact is probably barely visible, I loved debating how to value each move for a piece so the computer could determine what moves were the best. Despite a depth of only two, I am surprised at how well the ai plays, and it never misses taking an unguarded piece which is an advantage over many real players. When I show my code to other people, it is always exciting for them to challenge the computer and to see how it reacts. The best touch is how it prints the value of each move as it calculates it, so the player can watch the computer calculating which move is its best.

Finally, if I were to do this project again, the biggest change I would make is in our handling of helper methods. We had some helper methods that were extremely useful, such as a method to find the piece on a certain square, however some methods we assumed would not be very important and very much regretted. The most important lacking helper methods are a comparator between two pieces and a general method for each pieces movement. These two methods would have saved us a lot of time if we had thought of them from the beginning.