1. Overview of the assignment + My goals in completing it

The first CM1210 (Object Oriented Java Programming) coursework involves two projects. The first is to create an algorithm to generate a magic square for a given positive odd integer; the second is to create a command line game that requires the user to solve a shuffled magic square for a given positive odd integer. My goals for the first project are to create a working magic square generator that first prompts the user to input a positive odd integer, then generates the magic square of size n * n in a 2D array format. My goals for the second project are to be able to successfully shuffle a requested magic square n * n times, allowing the user to reconstruct the shuffled magic square through input, a win condition that alerts the user that they have won, as well as displaying the number of moves made to achieve the rearrangement, and finally a prompt on whether the user wants to restart or exit the game. Throughout this coursework, I aim to improve my knowledge of the Java programming language, familiarizing myself with the Java reserved keywords and various methods to fully utilize Java's capabilities.

2. A description of my solution design, including assumptions made, algorithms used, etc.

Project one was straightforward, the public class "MagicSquare" consisted of 3 methods and a main method. The key takeaways are the methods for generating and printing the magic square. The algorithm for generating the magic square was provided in the assessment document and represents a recursive algorithm that places numbers in a specific pattern; this algorithm ensures all rows, columns, and diagonals contain values from 1 to n. However, the use of modulus wasn't provided in the assessment document. The modulus (... % n) is a key component to ensure the values within the new coordinates for the values within the n-sized magic square are placed within the square's boundaries, essentially wrapping around any values that would exceed the square's dimensions. At the end, my solution was clear, and it delivered all the functionalities required.

Project two, on the other hand, required me to modify the code from project one and adding more methods to achieve the goals of project two. The key takeaways are the methods for shuffling the magic square, user interface, and validating user input. To create the function for shuffling the magic square, I utilized the Switch Case statement, which led to me understanding the advantages of using the Switch Case statement over if/else statements. I assumed that making the user interaction method would be the most challenging, and I wasn't mistaken. I needed various trial and error attempts to piece together different components to functionalize this method. Since I've declared the user's input as an integer, inputting anything else that wasn't an integer would output an error; That's where I learnt about the Try Catch statement, yet another useful statement to handle unexpected errors to prevent program execution failures. Making this method has also strengthened my knowledge on while loops, if/else statements, switch case statements, and manipulating different variables. Within the user interaction method, I assumed I required a win condition statement, where if the user's move creates a magic square, the win condition will respond and prompt the winning message to the user.

3. A discussion of my completed solution to the assignment, including the scope of solution, quality of the solution, interesting results, difficulties, overcome, enhancements delivered...

Finally, the functionalities of my program meet the requirements displayed in the assessment document. My program allows users to create magic squares in any positive odd integer size they desire. Furthermore, it gives users a challenge to solve a reshuffled version of the magic square they created by inputting moves in a specified format and rearranging the values until the square is solved. After solving the magic square, the game will notify the user on the total moves they made.

As for enhancements delivered, I've added additional details to the game, which makes the users feel more inclusive. I've added an ascii art, organized line breaks, and multiple dialogue lines to introduce and explain the objective of the game to the user clearly, and I will have a dialogue line printed each time the user makes a move. When the user makes an invalid movement (e.g. going past the boundaries, input values too big, wrong format), a message for that error will display. Furthermore, I've added a win condition where if the user matches their magic square correctly, a win statement will appear (from the win condition method), asking the user to select if they want to restart or exit the program. I initially dealt with the problem of the win condition only recognizing the generated pattern for the correct magic square, and not the other possible correct patterns; At the end I solved the problem by implementing for loops and if statements to create a checker for whether the rows, columns, and diagonals, respectfully all added up to the same amount (thus the magic square has been achieved). Although, there is a notable limitation to my program that I wish to further improve if given a second opportunity. Every single mistake the user makes will have its own error prompt, however, when the user is prompt to make a move, if the user only inputs a positive number, nothing occurs, if they do the same, nothing occurs, and until they've inputted for the third time, then the error or move shows; This is the only input issue that I'm struggling with to solve, I attempted to utilize the scanner hasNext method, but failed to apply it to the Try Catch statement in the user interaction method.

4. A discussion of my software test methodology. How did I ensure that my solution does what it's meant to do?

My software test methodology was precise and thoughtful to ensure the program functioned correctly. Firstly, I ensured that the generated magic square was correct and that the shuffled magic square was different each time I executed the program. Secondly, user interaction operates smoothly. I have considered all cases where the user's input would lead to an error, thus leading the program to malfunction. Whether it'd be a different numerical value to that of a positive odd integer such as negative numbers, or letters and special characters; The program will then display an error message for whichever the specific error is, and the user will be asked to try again. Thirdly, to ensure that the switching between values during a move is correct, I ran multiple debugging tests (within Visual Studio Code) to ensure the values are utilized and switched correctly throughout the program. Finally, to test whether my program works for magic squares of all sizes, I attempted to solve the magic squares for positive odd integers 3, 5, and 7, and all times it generated the right magic square, and when the shuffling meets the exact same arrangement as the right magic square, the user wins. Although there are limitations (as highlighted previously) to this program, it overall achieves what I and the assessment aimed for, thus I am satisfied with what I made.