Constructor:

I first created three boards, one was to keep track of places on the game board and have everything drawn on. The other two was for each player in the game. I also created two lists for each of the players square coordinates that have been drawn. Also, there are two score variables. The first score variable is to hold the score scored on a specific turn, which the other variable I used to keep track of what each respective player score is. I also have the same for the multiplier to keep track what the multiplier was for the last turn and how it impacted the score earned that turn.

paintEvent:

I first created two for loops that will visit each position on the board and ultimately do all the drawing I need done. I created two if statements for 0 and 1, which correspond to player 1 and player 0, respectively. This will draw in the square where each player clicks on the game board. The following two if and elif statements tell the user who's turn it by displaying the players name in their color telling them to play their move. This is changed every time a click occurs in order to alternate the turns. What is also drawn in the for loops, is permanently appearing score boards with its respective multipliers right to its side to display how the game is being scored.

Alternate:

This function takes whoever's turn it and switches it to the other turn using a variable called self.__turn. This is how I incorporate the switch of turns and to make the game a two-player game.

mousePressEvent:

I take values for row and column using the position of the click and make the proper math equation in order to have it become an index that my list of board can interpret. I then use these two values for row and column and check to see if they are clicked on a part of the board and then if that positon on the board is either already taken by a 0 or 1 which shows a previous click. If the click passes these two conditions I set our counter back equal to 0 and alternate the turn of player. Then I use the corresponding player turn to either write a 0 or 1 depending on whos turn it is in the entire board and then the respective player board. At the end of this, I update the program with the new movement.

CheckSquare:

For the regular squares, I used a generic algorithm that just adds one to some positions in the implementation to check each position of the board for this specific square. For a 2x2, I know that there a total of 6 possible combinations in the first two rows and it could move anywhere in the board. To take this into account I subtracted 2 from the length of the board(8) and made two for loops that will check row and column, respectively with I and j. The first thing I will do is check to see if the four positions of the square has a 0 or 1 depending on whos turn it is in their

own respective player board. If this condition is met, I will then create a path that will first make sure that at least one of the four positions is not in the list for square coordinates and then add each coordinate to the list along with the score and a 1 to the variable counter. This will of course only run once since the second time around the four positions will all be in the coordinate list, so the else statement will draw four line segments from the positions. This way the score is only added once and the information to draw is also added, but not drawn yet. To draw I just call this function in paintEvent. Setting the pen to green and back to black after the lines have been drawn in the else statement. In order to do this for the following squares I just changed all the 1's to 2's in the indices' and the draw line information. And in the for statement I subtract one more making the -2 into -3. I continued to do this until it reached -7 and 8 meaning the largest possible generic square to be drawn. To take into account diamond squares I had to only use the odd numbered sides of squares since even numbered square are not possible to draw. I used the same concept as the genric square in terms of repeating the for loops. This time I first made sure that each position that was able to draw either a 3x3, 5x5, or 7x7 square was either all 0 or 1 and then I added the coordinate into the list in order to incorporate the points and counter addition then I used the else statement to draw the lines from position to position. This time to go from each len I subtracted 2 each time from the for loops and added one or two to the indices to incorporate the possible positon of the points on the board. The rotated squares for squares 4x4 to 7x7 is the following lines, which uses the same concept of checking the positions for either a 0 or 1 in each of the positions necessary to create a square then add the coordinates to the list in order to add the score once and its multiplier and then draw the square the following times its iterated. Again, you just add one or subtract one from each index to check for the proper square. I made all the player1 squares in the first half and then player0 in the second half in an attempt to have them organized in some way. At the end of the function I call upon the function Multiplier to deal with the Scoring part of the game.

Multiplier:

What this function does first, is check to see if the number of squares drawn was either zero or one and if that has happened then the multiplier will be set back to 1 and the score from the round is added to the score shown and then set back equal to zero for the next click. If the number of squares drawn is 2 or greater, then the multiplier is set equal to the counter variable which is shown on the score board and then the round score is multiplied by the counter variable and then added to the actual score. After, the round score variable is set back equal to zero. When a rotated square is drawn, the program uses this as if two squares have been drawn and gives the corresponding multiplication and multiplier. So let's say a 3x3 rotated square is drawn with a 2x2 generic square, the multiplier would be 3 and it would be 13 times 3 making the round score equal to 39 which is added to the actual score. This is a way that my program rewards for creating a rotated square.

Dazzle:

When opening the program, there are prompt boxes that ask for names for player0 and player1 to give the program more meaning and connection to the two players who are playing the game. I also, added a feature that looks at the scores as integers and determines who is winning and writes who is winning at the moment and if it is tied it has a neutral color saying the game is

tied. Also I added a feature that restarts the entire game if the space bar is hit in order to give a way to keep on playing the game after one game has ended.

Problems Encountered:

When trying to draw just the square on the game board. The square would draw and then the rest of the board was shaded in by the color of the square. To fix this I had to research how to take out a Brush color and have just the square draw. Also, I had figured out a method that would tell that a square was drawn but I could not figure out a way to draw the actual lines to draw the square and how to incorporate score into the program. To fix the draw issue, I had use inspiration from our project about ShapeDrawer and using the qp parameter to have information to draw rather than actually drawing the lines. For score, I had use an if statement that will only add the score once per square of those coordinates by having an empty coordinate plain and appending those coordinates so that the if statement will skip over it the following time and using else to draw the lines for the square.