* Let's examine out moveStone function from line 249
* We'll say that the initialization on line 249 takes A milliseconds
* We'll also say that the initialization on line 251 - 254 takes B milliseconds
* Then cout on line 256 takes C milliseconds
* Now from line 257 to 271 we have a do while, this will take N \* (line 259 - 271)
* on line 259 we call function getPosition (line199) which will take A1 + B1 + N \* (C1 + D1 + E1 + F1 + G1 + H1 + I1) + J1, but let's simplify this to D + N\*E by adding everything together.
* Then on line 262 in the if statement the check takes F milliseconds and the execution of the code in either if or else takes G milliseconds.
* Finally on line 271 we have to check another statement which will takes us H milliseconds.
* That will leave us with: A + B + C + N \* (D + N\*E + F + G)
* Let's add A + B + C and just call it A for simplicity's sake.
* A + N \* (D + N\*E + F + G)
* Let's do the same for D + F + G and call it D
* A + N \* (D + N\*E)
* Now we'll simplify it to A + D\*N + E\*N^2
* Cout on line 273 takes F milliseconds
* 274 calls getPosition again (line199) which will take G + N\*H milliseconds
* And from line 277 to 281 we have an if statement that we'll say takes I seconds if we pass it (if we do not pass the if statement our function will return void and stop running)
* So far We've got: A + D\*N + E\*N^2 + F + G + N\*H + I
* From line 283 to 339 we have a massive if - if else - else statement with 5 possible outcomes. However if we get outcome number 5 our function will end so we will ignore that possibility for now. That leaves us with 4 similar possible outcomes.
* In either of these 4 outcomes we enter another if statement where our function stops running if we hit the else. We'll assume the user entered a valid move, in which case either of the 4 outcomes will take K + L milliseconds with the original check(s) for the statement on line 283 taking J milliseconds.
* Now we have: A + D\*N + E\*N^2 + F + G + N\*H + I + J + K + L
* Let's cheat a little more for readability's sake and call I + J + K + L just I (we add all of them together of course)
* A + D\*N + E\*N^2 + F + G + N\*H + I
* Now we have a final if statement on line 341, with all the previous error checking in place the outcome of this will always be true, and the check itself will take J milliseconds.
* The execution of line 343 to 347 will take K milliseconds altogether
* And that leaves us just with function checkAdjecent being called on line 349. This is a long function (line 122) which has a lot of if else statements which are all within one for loop. So let's say this function takes N\*L milliseconds.
* Now we have reached the end of our move function after taking A + D\*N + E\*N^2 + F + G + N\*H + I + J + K + N\*L milliseconds.
* Let's try to simplify this a little bit:
* A + D\*N + E\*N^2 + F + G + N\*H + I + J + K + N\*L
* = A + F + G + I + J + K + D\*N + N\*H + N\*L + E\*N^2
* = A + F + G + I + J + K + (D + H + L)\*N + E\*N^2
* Our most dominant factor is E\*N^2
* This means my move function has order N squared or O(N^2).
* If my user would not make any mistakes and enter no invalid moves, all of the N's except N\*L would be 1, which would leave me with the simplified:
* A + D + E + F + G + H + I + J + K + N\*L which would give me O(N)
* But unfortunately my user may make errors, so I am still stuck with O(N^2)