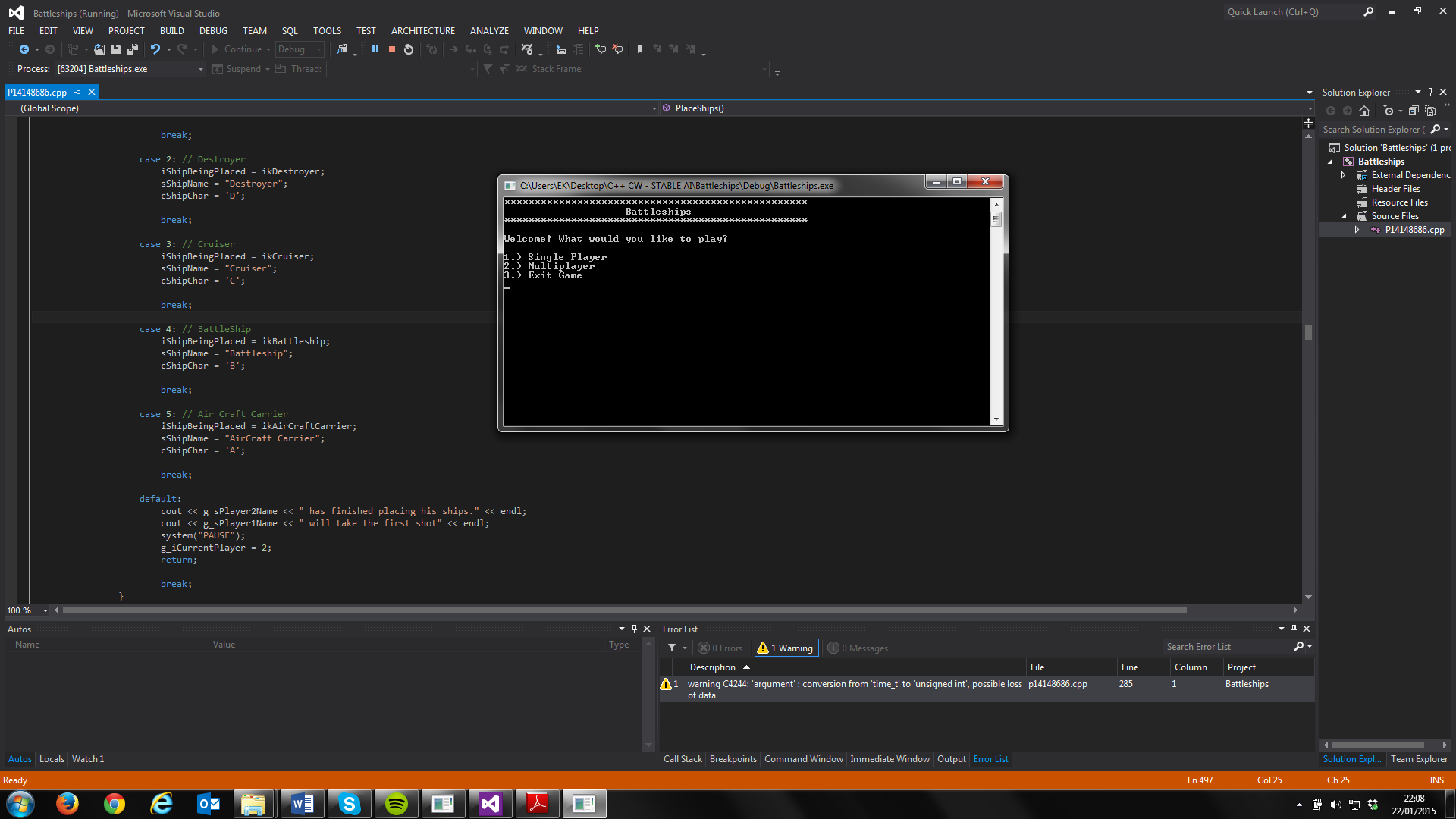
Battleships Report

Battleships was a challenging game to program in C++. The majority of the challenge came in the form of allowing the users and the AI to place ships where they want. This is mostly due to the complexity of having so many checks to perform for each move. The program has to check that each ship (of various lengths) are within the borders of the grid and also if they are overlapping with each other. Another challenge to this system is that the ships are also rotatable adding an extra layer of complexity. This was quite a hurdle to overcome and while it was difficult I also found it educational and rewarding.

Firstly when starting my program I decided to use four 2 dimensional arrays as opposed to one 3 dimensional array for the sake of readability. I felt it would be possible (and would use less lines of code) to use a single 3D array to store everything by using the first value in the array as a toggle between the players grid, players hit grid, enemy grid and the enemy hit grid but I felt that this was too pointlessly complicated of an idea which would only make my code more difficult to read and understand.

I created a constant global integer called “g\_kiGridSize” and used it in order to ensure all of my arrays were of the correct size and avoid potential overflow problems during for loops, for example when my grids are draw in the “drawBoard()” function.

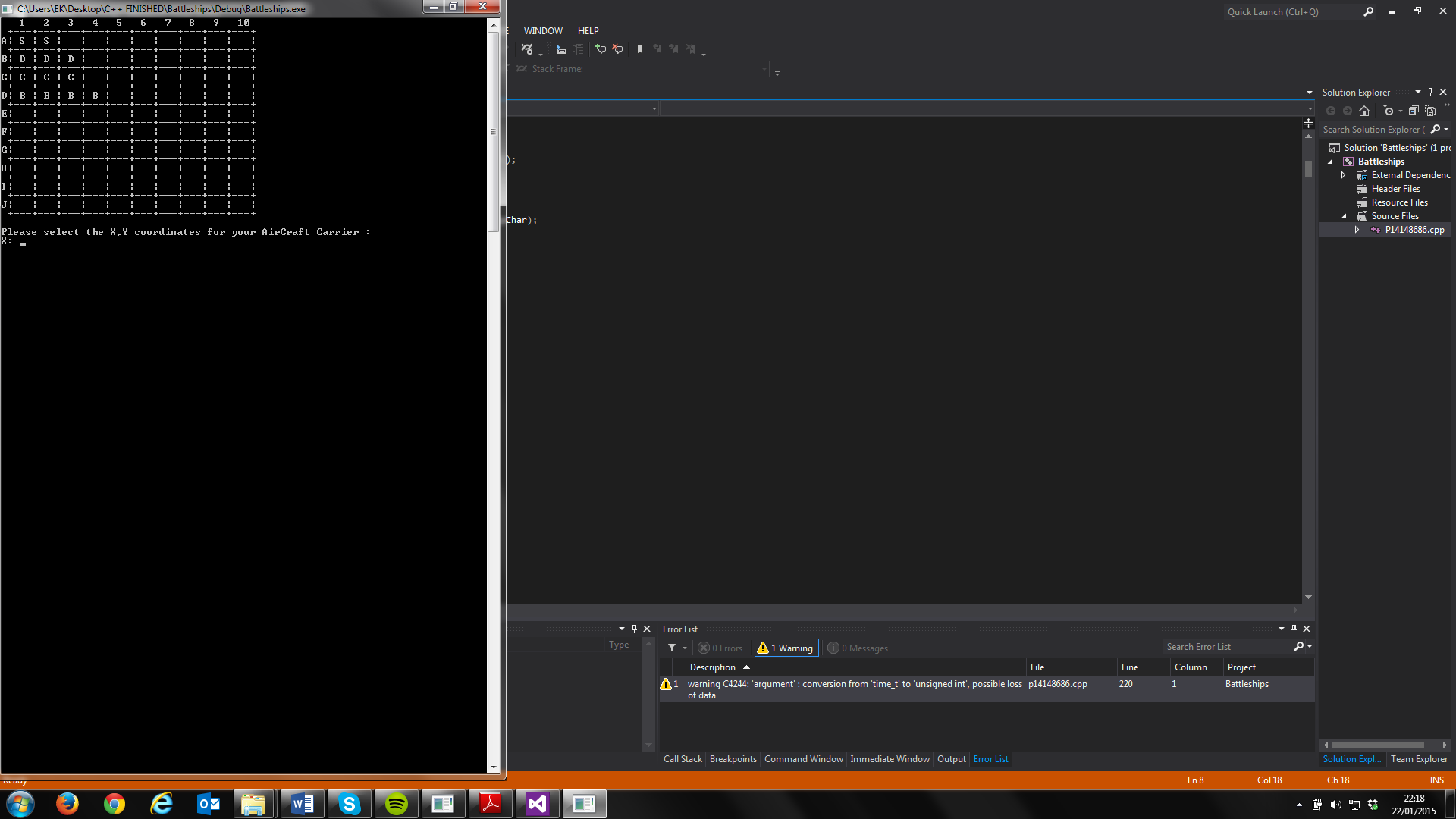
Once all of my arrays were created I decided my first course of action would be to create a main menu so my player would choose between playing “Single Player”, “Multiplayer” and exiting the game. Single Player would be against an AI opponent whereas multiplayer would be against a second player. I created a subroutine for the main menu using basic “cout” and “cin” commands and made an effort to make the menu itself to look appealing. While I would have liked to attempt a graphical interface at this point I simply lacked the knowledge of how to create it and decided to stick with systems that I know and understand and potentially think about adding graphics to the system if I still have time towards the deadline and the text-based version is finished.



When the user picks either the single or multiplayer mode I ask them for the names of the player(s). This is so I can alternate the players in both the game modes by telling them who’s turn it is. If there are two players the game asks for two names, however if there is only one player the game will only ask for the humans player’s name.

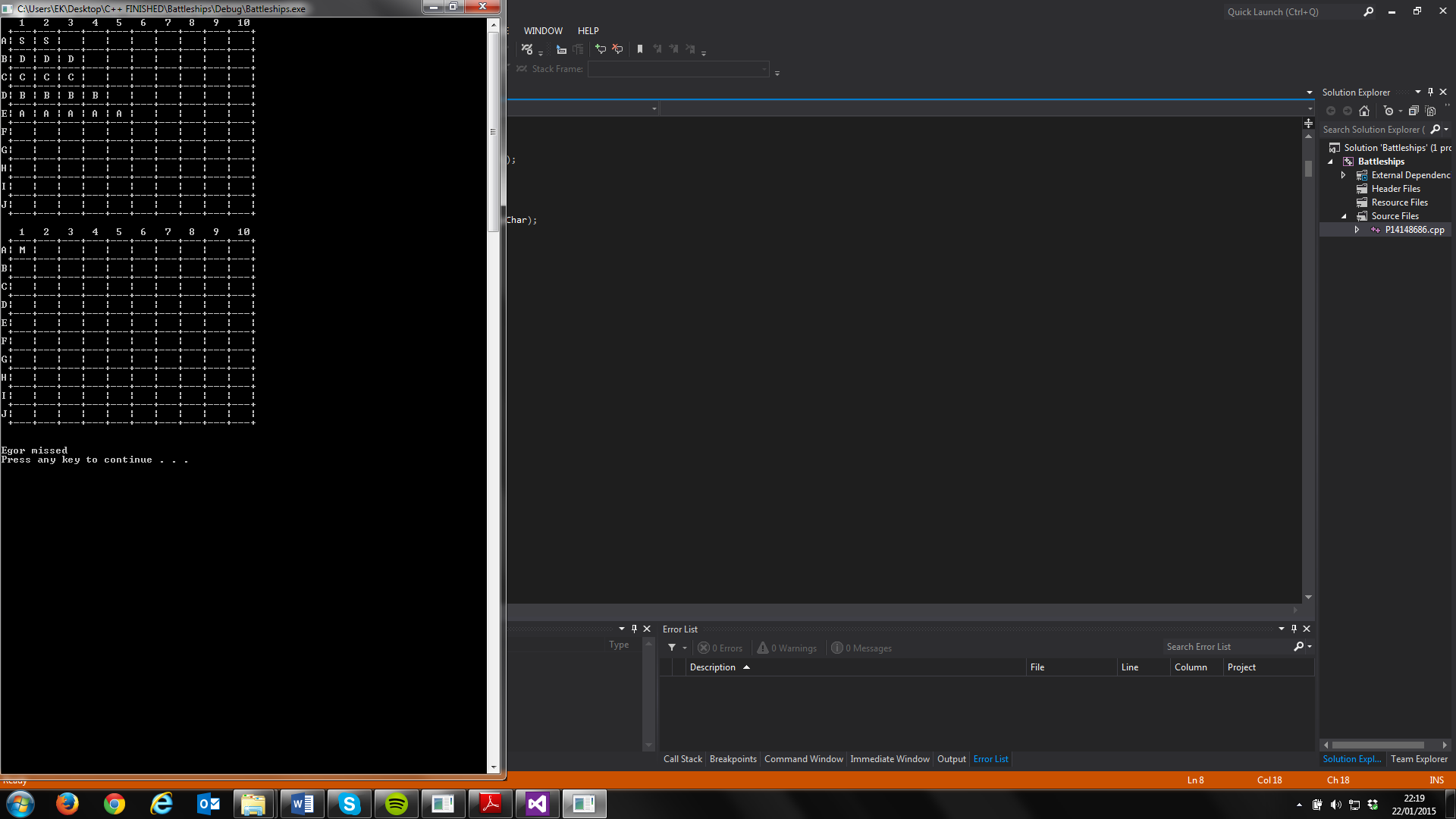
Once names are provided my game draws the first player’s board using the “drawBoard” subroutine. This subroutine requires a single parameter which is the array which will be loaded into the board; this means that by simply changing the parameter of the function I am able to draw any of the four boards in the game to the screen. I use this often throughout my code.

The “PlaceShips()” subroutine took most of my time. I was by far what I found the most challenging out of this coursework. I had to accommodate for both players aswell as the possibility of one of the players being an AI. The way I make the ship placement work is by use of ‘for’ statements. In simple terms one of my ‘for’ statements cycle though each ship type changing the variables which represent their names, length and the character which represents them. Then I let the user pick an origin point for their rotation and check that that point had not already had a ship placed in it. Then from that origin point the user must pick a rotation which uses a ‘for’ loop to check the validity of that location. Firstly, I use a variable called I blank check which checks if the number of positions needed to for the ship are available in that direction. If the check encounters either a ship or the end of the grid in either location then the game will refuse the user the right to place a ship in that location and give the player and error. The game will then ask the user for another location. Once all the ships are entered into valid positions then the second player or AI will place their ships .When the AI places the ships it performs a similar action except that the X, Y and rotation values are randomised instead of chosen. If the AI gets an invalid location it will try again until a valid position and rotation is selected. The difference between the AI placing ships and the second player placing ships is that the AI has no display and simply shows a message saying that all the ships have been placed. Once all the ships are placed by either a player or AI or both players.



The “PlayGame” subroutine controls the gameplay one all of the ships have been placed. Firstly, it calls “switchCurrentPlayer()” which switches the player following this the screen is cleared and a message is shown so the players know who’s turn it is. One the turn has been established the “PlayerShooting()” subroutine is run.

“PlayerShooting()” is mostly rather simple. It loops as long as the player(s) or AI keeps getting consecutive ship hits. Firstly the both the array storing the current player’s ship locations and an array storing where the current player has shot and hit are drawn. The player will then select a location and the program will validate this location and ensure it is within the boundaries of the grid. Once both a valid X and Y value are given then the program will check that that place has not been shot at already, if it has then the game will ask for another set of X and Y values. If it has not then the ship that the player has hit will be sent to the “ShipDestroyed()” subroutine which will minus “health” the ship that was hit. If that ships health becomes ‘0’ (which is represented by an array for each player) then a message will be shown to that player telling them the ship has been destroyed. If all the players’ ships are destroyed this way then a victory message will be shown to the current player. The “PlayerShooting()” subroutine will mark ship hits with a X and send a “HIT!!” message to the current player. It will also mark misses with the ‘M’ character and then make “bHit” equal false ending the loop and switching the player.



There are a number of things I would like to improve in my program if I had more time or was to do it again. Firstly, I feel like I have used too many global variables. While my 4 arrays need to be global because of the difficulty of referencing arrays and the fact that they are used almost everywhere throughout my code I could have likely limited the number of other global variables by removing global variables such as “g\_iCurrentPlayer” and replacing them with local and often referenced variables. If would also have liked to decrease the amount of repetitive code specifically my ship rotation selection which I use many times. Originally my code had many repeats of my ship loop subroutine but instead improved it by adding the “SetShipType” subroutine which I use throughout my code to decrease repetition by using referencing in order to permanently change the variables in my code. I would also have liked to allow the user to enter any order of character or number without crash as it is rather annoying when a character is accidentally entered as an integer and the game crashes; I would have done this using string manipulation if I had more time.

Overall, I have found this project to be helpful in advancing my skills and generally educational.