SWARS

A 2-player networked space battle game.

The Deliverables (Stuff You Must Turn In)

Step 1 (24%)

Write [] = swars(*player*). *player* must be 1. Draw a "ship" wherever the mouse is positioned inside the window, such that the ship will move with the mouse.

Next, modify swars such that the ship will no longer follow the mouse. Instead, when the right mouse button is clicked, the ship will move in a straight line from the ship's present position to the location the mouse was when the right mouse button was clicked. The ship will halt when it reaches the spot where the right mouse button was clicked. The speed of movement is up to you, but should be slow enough that we can watch it move bit by bit across the screen. It shouldn't be so slow as to be boring, though, and this speed can be adjusted in the subsequent steps to make game play more interesting. Typing q while the mouse is in the window will cause the program to exit. You don't need to delete the window, but the ship should no longer respond to mouse clicks or motion after typing q.

Step 2 (25%)

Improve swars such that *player* can be 1 or 2. Running swars(1) will display a window, while running swars(2) will display a second window. swars(2) could also be run on another computer on the same network, displaying a window on that second computer. Two ships will be displayed in both windows. Player 1's ship should be mostly red, while player 2's ship should be mostly yellow. You may change their shape if you want, or add some other coloration (like an outline color), but keep your ships mostly this color. Player 1's ship should respond to right mouse clicks in player 1's window, while player 2's ship should respond to right mouse clicks in player 2's window.

I recommend you start by getting player 1's ship to display in both windows, but just display player2's ship in player 2's window. Both ships will respond to right mouse clicks as described above. Make sure both instances of swars can be entered and exited without causing a crash. After you get all this working, then add the ability of player 1's window to display player2's ship, such that all the requirements in the previous paragraph are now fulfilled.

Step 3 (25%)

Both ships can fire torpedoes at each other. Clicking the left mouse button fires torpedoes from the player's own ship and each mouse click fires a new torpedo. The torpedo travels in a straight line from the ship to the mouse's location at the time the left button was clicked. The torpedoes don't do any damage and they just pass right through a ship or another torpedo if a collision occurs. A torpedo which moves off screen should be deleted, which means it is no longer displayed and no memory is used to store it. Notice that torpedo movement is different than ship movement: ships stop when they reach the location where the right mouse button was clicked, but torpedoes just keep going until they go offscreen. Both windows show both ships and all the torpedoes.

Step 4 (25%)

The torpedoes from step 3 kill an opponent's ship if they hit it. Notice that your opponent might try to dodge your torpedoes. You must get a torpedo into the same location at the same time as your opponent to score a hit. Since it's unlikely, using floating point computation, that your torpedo will be in exactly the same place at the same time as your opponent's ship, use a "kill radius", which is just a circular zone around your opponent's ship. If you manage to get a torpedo in this zone, your opponent is destroyed. Destroying an opponent gains a player one point. Destroyed opponents disappear and respawn at a random location on the playing surface, while the player who was not destroyed maintains his previous location and is able to continue moving and firing torpedoes. Hitting your opponent 10 times causes you to win the game and both programs should exit. The score should be clearly displayed on both windows at all times.

Peer Evaluation (1%)

You get 1% by turning in your peer evaluation of yourself and your teammates. Furthermore, the weighted sum of your Step 1, Step 2, Step 3, Step 4 and Peer Evaluation grades is multiplied by the average peer evaluation you receive from members of your group. The Peer Evaluation is just a number from 0 to 100 for yourself and each of your teammates. Everyone will turn in a Peer Evaluation to their own Blackboard Project Peer Evaluation assignment.

<u>Program Requirements</u>

Communication between programs will be done using fprintf and fscanf. This isn't the best way to do it and is even a little ugly. However, this isn't a computer science class but a programming class for engineers. fscanf and fprintf are much more useful for engineers than using sockets, which is one way a computer scientist might prefer to communicate between the two players. Interested students can learn more here:

https://www.mathworks.com/help/instrument/communicate-using-tcpip-server-sockets.html

Truths, Facts, Hints, Suggestions, Etc.

1. Most of the communication between programs is one way. For example, each player tells the other player where he is located. For one-way communication, the easiest thing is for one party to open a file with "w" and the other with "r" and just leave it open until the game is over. The only requirement is to call frewind() after the read or write is done to reposition the file pointer for the next read or write.
2. A small amount of communication might be two-way, which means that a program will read the file some of the time and write to it some of the time. The easiest solution here is to open it with 'w' or 'r', write or read it, then immediately close it. Make sure both programs won't be trying to write to the file at the same time or you'll have problems!!!! Note that since player1 and player2 are running asynchronously, in many cases it might be impossible to guarantee only one program will have a file opened for writing. If you can't guarantee this, then don't use a file for two-way communication. One-way is safer and easier.
3. You don't need an if statement to detect if the mouse has moved. The mouse position will be

updated automatically as the mouse is moved.

4. If a mouse button was pressed, a function will be called automatically and that function can set a global variable.
5. If a key on the keyboard was pressed, a function will be called automatically and that function can set a global variable.
6. You can open a file in one program with 'w+' or 'w' and in the other with 'r' and keep it open, writing and reading using 'w+' (or writing with 'w') and reading with 'r'. The file doesn't need to be closed after a read or write, just rewound. This doesn't have to be done synchronously.
7. Octave won't run newly created files unless you type : `path (path)` in the command window or restart Octave.
8. It does not work to have a file open with 'w+' from 2 programs and read/write from both programs. The best way to do this is to close the file after and reopen it before each file access.
9. I strongly recommend you limit global variable use to detecting mouse moves, mouse clicks and keys pressed.
10. If you add functionality beyond left clicks to shoot, right clicks to move and 'q' to quit, this should be displayed on the screen when the user enters the game.

Grading for Steps 1, 2, 3 and 4

25% Code Reusability (don't cut & paste code, use functions)
25% Documentation (standard function documentation plus commenting within the code)
25% Specification (does the program actually do what it's supposed to do)
25% Efficiency (correct choice of for/while/vectorization, if/switch, good design decisions)

Thoughts

One way I save a lot of code is by having both players use the same function, swars. swars has a concept of my ship and my opponent's ship. Also, it thinks about my torpedoes and the opponent's torpedoes. There's also my score and the opponent's score. If you saw my code, you'd see variables like my_file_pos (the file identifier that contains the position of my ship) and opponent_torp_list (a list of my opponent's torpedoes). In this way, one function can play both player 1 and player 2. Player 1's opponent is player 2 and player 2's opponent is player 1. This idea can flow naturally out of the pseudocode or flow chart you'll write. Sure, this requires some if statements or switch statements, but not doing it can roughly double the amount of code you must maintain. So this isn't a requirement, but you ought to at least consider it. It's easier to maintain a 500 line program than a 1000 line program (and mine is nowhere near 500 lines long).

You'll have a loop, but use as few loops as possible. Whenever you create a loop, ask yourself, "Can I come up with a vectorized way to do this?" Vectorized code is harder to write, but it's faster and shorter! It might actually be easier to write the 1000 line program with lots of loops than the 500 line program using vectorization, but do you really want to try to fix all the bugs in the bigger program? Also, don't you want the fastest program possible? You might have to carefully craft vectorized code, and perhaps even reread parts of your textbook, but it will be worth it to get short, fast, reliable code. Again, my program is much, much less than 500 lines!