Technical Tips for a Successful Project

The first step to understanding any new programming environment is to spend time reviewing the existing code – in this case, the sample player (**TemplatePlayer**). Where does it get control? What does it do? How does it do it? These are the questions you can user to start learning about the programming environment with the sample player program installed by default.

The second step is to begin adding optimizations. In the game of battleship, there are 3 key things which can be optimized. They are:

1. Keeping track of your guesses such that you do not guess the same location multiple times. As we play a game, we want to check any guess against the list of already guessed values, especially when the guess is created randomly. Think about how you might represent the board and track your guesses. Then, ensure that your random guesses have not already been guessed. If they have, continue getting random numbers until you get a hit. Or you might consider thinking about how to apply your random number to only the remaining guesses.

2. When you get a "hit", your next guess should be focused on sinking the ship, i.e., "localized" to the where the hit occurred. (For simplicity, let's call this the "sink optimization".) The challenge with this situation becomes processing all the "hits" completely in a region. One can think about this as ensuring you get a ring of misses around a hit. But what happens when you get a second hit? Techniques such as recursion OR data structures like queues or lists may be helpful in helping keep track of hits which have not been complete processed. Oh, and the guess optimization from #1 above will come in handy too.

3. Traversing the remaining free space on the board in a predictable fashion when all previous localized hits have been handled. (Let's call this the "free space optimization".) One way to explore various "free space optimization" algorithms is to think about the type of ship-hiding strategies players may use. Ships may be grouped versus spaced. They may be clustered or in a line or not connected. They may be located near one side, in the middle, or generally distributed. Think back to your strategy when playing the real game. If you knew what kind of boat distribution philosophy your opponent had, how would you search for

those boats? This optimization is about translating your search strategy to a computer algorithm.

Now, let us think for a moment about team strategy. If this competition was scored individually, then most of your time should be spent trying to find an optimal algorithm for traversing all board types. However, given that this is a team competition where points accrue for the class, the class should focus on diversity of algorithms.

To explore this concept a little further, let's look closer at the various "free space optimizations" which are possible. How many ways can you "traverse" a board? Here's a starting list:

- Top-to-bottom
- Left-to-right
- Outside-to-inside
- Diagonally

Can you think of others? How about the "opposites" such as "bottom-to-top"? Don't forget those. Oh, and does the starting place matter? Yes! I can start at the top-left or the top-right sides. Will both perform the same for a given board? Nope! Likewise, how will similar algorithms perform on the same board? Near identically.

Keep all these concepts in mind when selecting your "free space optimization" algorithm. Pick one that's different than your neighbor. The winning team will have the greatest diversity of algorithms.