OOD EXPLAINED

The design of the game is shown in the UML diagram along with the help of CRC cards. However, there are several classes which must be implemented to encapsulate some objects to gives an OOD definition to the program.

Two of the main features of the OOD are:

1. How the tanks are going to be placed?

Sol: - As it can be seen in the UML diagram, the board is designed as a 2-D array where each index of the matrix will of type CELL which is also implemented in the model. This board is a grid made up with these kinds of cells and the cells have certain members which will store the information about that particular index of the board. It has a member status which will tell if this particular cell has been engaged in one of the tanks or not. If it has, then it will be given name of that tank. Along with these features, each cell will have a list of its neighbours which will help the algorithm to place the tanks on the board in the first place. While placing the tanks, any vacant cell is chosen at random and the rest 3 cells must be chosen such that they are not engaged in any other tank and they are not already given to this tank. After engaging, the USER class will have this information stored about a particular tank with the help of the name of the tank. All the tanks will have their own name and it will be shown on the fortress map for the user once he/she hits any cell of any tank. The classes involved in this part is obviously Board, Cell and Tank.

2. How the user's move is handled?

Sol: - Once the user is prompted to enter which cell he/she wants to target, that string is taken in and forwarded to a user which is an object of the USER class. So, every time the system takes in the information about the next step, the user object knows about it. It uses a class called SHOT ANALYZER which is dependent on cell and that very cell is in a particular tank or it is vacant. The user analyzes the shot as if the targeted cell is vacant or not, if it is related to a tank or not, if it has already been shot or not. In this case, it uses the information about a particular cell. If a cell belongs to a particular tank, it will update the damage done to that tank with the help of *getShotAnalysis ()* function in the tank class. After retrieving the information about the shot, the state of cell is changed according to the original state of the cell.

a) If that cell was a part of any tank, it is hit now and the damage that tank can produce is reduced accordingly and that particular cell has ' ' as its name.
b) If that cell was not a part of any cell, then the user is told that this shot misses every tank and there is no decrement in the damage each tank can produce. But the state of that particular cell is changed anyways.

If a cell is targeted again, the shot analyzer will straight away say that this cell has already been targeted once and hence there will be no need to check that cell again. This will help improving efficiency too.
The classes involved are Cell, Shot Analyzer, Tank and the User (to some extent).