270 BattleShip Documentation:

**CoordToIndex**

* **Purpose**: Converts user-input coordinates into indices suitable for array access.
* **Parameters**:
  + coords[] (char): Raw coordinate string input by the user.
  + start (int): Starting index for the coordinate substring.
  + end (int): Ending index for the coordinate substring.
  + startingChar (char): ASCII starting character for numeral conversion.
* **Returns**: int - Converted index based on the numeral system.
* **Details**: This function splits coordinates into two numeral system components (e.g., ‘AAA001’ into ‘AAA’ and ‘001’) and converts each part into an integer index. The numeral system for rows and columns can be customized with any sequence of ASCII symbols in a sequential order (e.g., ‘A’, ‘B’, etc.).

**GetCoordSplitIndex**

* **Purpose**: Finds the split index between row and column components in user-input coordinates.
* **Parameters**:
  + coords[] (char): Raw coordinate string input by the user.
* **Returns**: int - Index in the coordinate string where row and column components split.
* **Details**: Assumes valid input. Identifies the index separating two numeral system components by comparing against the character range of the row’s numeral system.

**IsCoordValid**

* **Purpose**: Validates if a coordinate component lies within specified bounds.
* **Parameters**:
  + coords[] (char): User-input coordinate string.
  + start (int): Start index of the component.
  + end (int): End index of the component.
  + startingCoord (char): Starting ASCII character of the numeral system.
* **Returns**: bool - True if the coordinate component is valid; False otherwise.

**IndexWithinRange**

* **Purpose**: Checks if an index value is within the bounds of the grid size.
* **Parameters**:
  + index (int): The index to be verified.
* **Returns**: bool - True if within range; otherwise False.

**alloc\_ArrayCoordsFromUserCoords**

* **Purpose**: Allocates memory for array coordinates derived from user-input coordinates.
* **Parameters**:
  + coords[] (char): Raw coordinate string from user input.
* **Returns**: int\* - Dynamically allocated array of size 2 containing i and j array indices; NULL if the coordinates are invalid or out of bounds.
* **Details**: The function splits the coordinates into row and column numeral system components, then calculates indices for each. If valid, memory is allocated for a 2-element array with i and j indices.

**alloc\_GridAreaFromInput**

* **Purpose**: Allocates memory for boundaries of a specified area on the grid based on orientation and dimensions.
* **Parameters**:
  + startingCoords[] (char): Starting grid coordinate in user input.
  + orientation[] (char): Orientation of the area ('horizontal' or 'vertical').
  + width (int): Width of the area.
  + height (int): Height of the area.
* **Returns**: int\* - Dynamically allocated 4-element array containing boundary indices (i0, i1, j0, j1).
* **Details**: (i0, j0) is the starting coordinate; (i1, j1) is the ending coordinate. Depending on the orientation, width and height are assigned to either the row or column range. Adjusting orientation flips these assignments.
* **Attack function:**

Input: char points[] (which is the coordinates) - and a variable named turn  
after the attack turn will be incremented by one, and a function named CheckTurn will be added to check.

Task 3: Create the main driver:

Functions:

1. StartGame() ---> Asks the user to input any key to start the game
2. SelectGameMode() ---> Asks the user to select gamemode. There are two game-driver functions, one for each game mode.  
   Picking a game mode will call the respective function which will handle gameplay accordingly.
3. Quit() ---> Stops the program and quits the exe file.

Inside the PVP-Driver Function:

1. SetPlayerNames() ---> Function that sets and saves player names
2. SetupPlayerGrid() ---> Function that sets up a player's grid by allowing him to place his ships. (Needs to be called twice in another parent function)

Common function between two game modes:

1. RunTurn(int oponentNumber) ---> A function that contains all behavior for player attacks and gameplay moves. Takes an integer argument and performs whatever move inputted by the user on the player at the index oponentNumber in an array of players. This allows for scalability, if we wanted to add more players later (the current player could maybe pick which player to attack...), or even for PVE we just fill the player array with the only player in game and give it index 0, and AI takes index 1.

In-game Functions:

1. NextPlayer():
   1. A function that switches turn to the next player in the player list.
2. Attack(Player player, int attackIndex):
   1. A function that runs the specified attack by the player. Using a switch-case operation, it calls the right function for the attack on the target player’s grid.