*Owen Lindsey*

*Professor Demland, David*

*CST-201 Battleship*

*10/20/2024*

**Pseudo code for battleship**

*Pseudo code solution for PlaceCruiser:*

**METHOD PlaceCruiser(board, row, column, cruiser)**

**IF cruiser is vertical THEN**

**IF cruiser is upwards-oriented THEN**

**SET offsets to: (0,0), (-1,0), (-2,0)**

**ELSE**

**SET offsets to: (0,0), (1,0), (2,0)**

**END IF**

**ELSE**

**IF cruiser is left-oriented THEN**

**SET offsets to: (0,0), (0,-1), (0,-2)**

**ELSE**

**SET offsets to: (0,0), (0,1), (0,2)**

**END IF**

**END IF**

**FOR EACH offset in offsets**

**SET newRow to row + offset.Row**

**SET newColumn to column + offset.Column**

**IF newRow or newColumn are out of bounds OR board[newRow, newColumn] is not empty THEN**

**RETURN false**

**END IF**

**END FOR**

**FOR EACH offset in offsets**

**SET newRow to row + offset.Row**

**SET newColumn to column + offset.Column**

**SET board[newRow, newColumn] to occupied**

**END FOR**

**RETURN true**

**END METHOD**

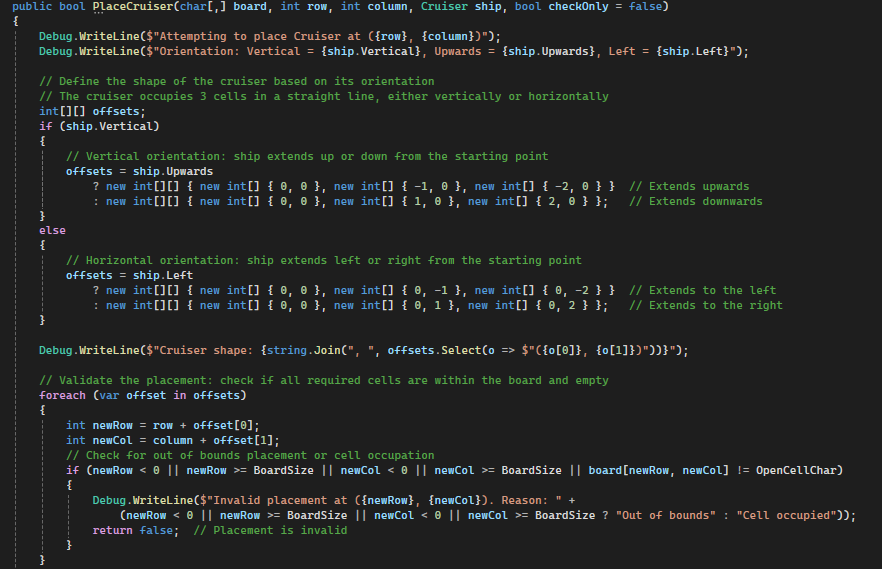
**Test Case for PlaceCruiser:**  
*Summary of PlaceCruiser Test Requirements:*

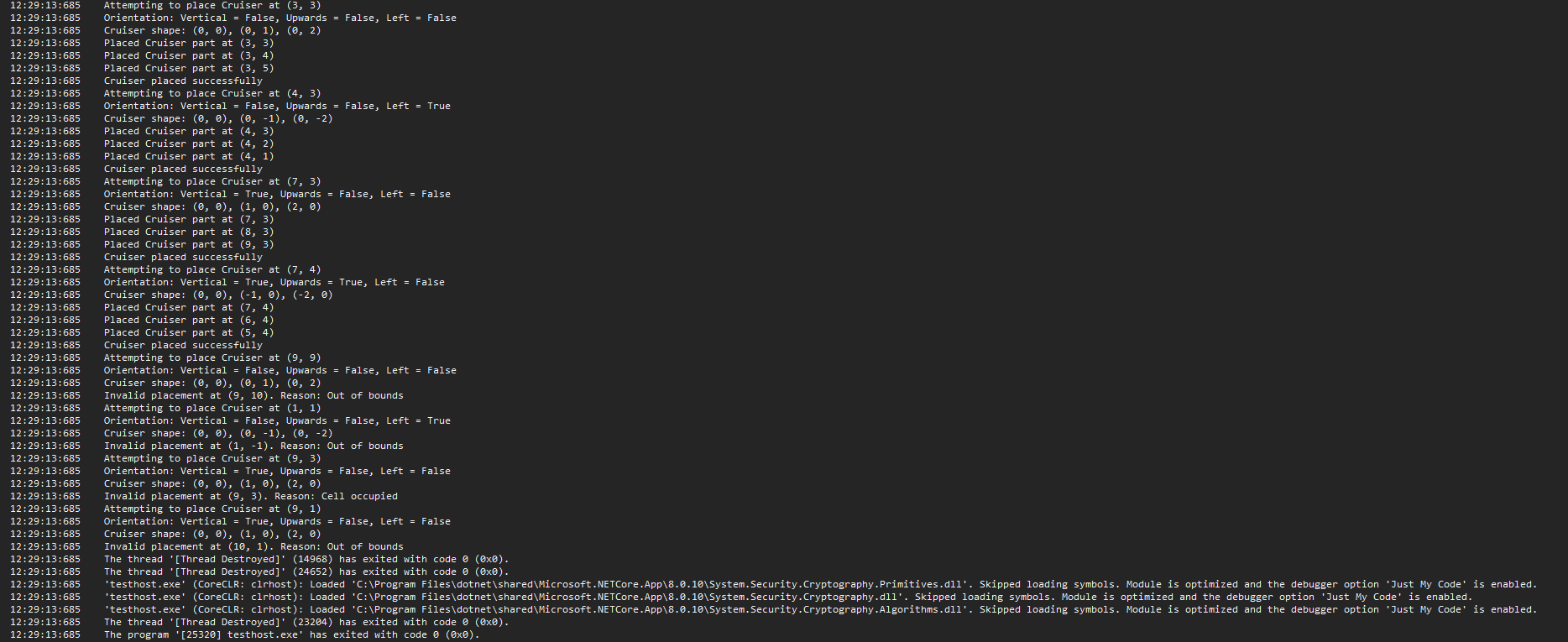
1. **Horizontal Placement:**
   1. Standard (right-oriented): Place at (3,3), occupying (3,3), (3,4), (3,5)
   2. Left-oriented: Place at (4,3), occupying (4,3), (4,2), (4,1)
2. **Vertical Placement:**
   1. Standard (downward): Place at (7,3), occupying (7,3), (8,3), (9,3)
   2. Upward: Place at (7,4), occupying (7,4), (6,4), (5,4)
3. **Invalid Placements:**
   1. Out of bounds: Attempt to place at (9,9) horizontally
   2. Out of bounds: Attempt to place at (1,1) horizontally left-oriented
   3. Out of bounds: Attempt to place at (9,3) vertically downward
   4. Out of bounds: Attempt to place at (9,1) vertically upward

Why **PlaceCruiser** Passes All Test Cases:

1. **Flexible Orientation:** 
   1. The method correctly handles all four orientations (horizontal right/left, vertical up/down) by adjusting the offsets based on the cruiser's properties (Vertical, Upwards, Left).
2. **Boundary Checking:** 
   1. Before placing the ship, it checks if all required cells are within the board boundaries. This ensures it catches all out-of-bound placement attempts.
3. **Collision Detection:** 
   1. It verifies that all required cells are empty (**OpenCellChar**) before placement, preventing overlap with existing ships.
4. **Atomic Placement:** 
   1. The method only places the ship if all checks pass. If any check fails, it returns false without modifying the board.
5. **Correct Cell Marking:** 
   1. When placement is valid, it correctly marks all three cells with **ShipCellTaken**.
6. **Return Value:**
   1. It returns true for successful placements and false for any invalid placement attempts, matching the test expectations.
7. **Preservation of Board State:** 
   1. For invalid placements, the board remains unchanged, which is tested by subsequent successful placements in the same test method.

The PlaceCruiser method's careful validation of placement coordinates, orientation-based offset calculation, and all-or-nothing placement strategy ensure it meets all the test requirements. Its ability to handle various orientations and edge cases demonstrates its robustness in dealing with different scenarios presented in the test cases.  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
**Test Case for PlaceCruiser:**  
*Screenshot of method PlaceCruiser:*



**Test Case for PlaceCruiser:**  
*Screenshot of Debug output of PlaceCruiser test case:*  


**Test Case for PlaceCruiser:**  
*Screenshot of success output of PlaceCruiser test case:*  
