**Testing strategies:**

for main functions test edge cases normal cases and invalid input.

**Attack.c**

**1.**Fire**Function**

**Test Case 1: Valid hit on a ship**

* **Input:** inputC = "B2", grid with a ship at B2.
* **Expected Output:** outputMsg = "Hit!", grid updated with HIT at B2.

**Test Case 2: Missed shot**

* **Input:** inputC = "C3", grid with water at C3.
* **Expected Output:** outputMsg = "Miss", grid updated with MISS at C3.

**Test Case 3: Already hit coordinate**

* **Input:** inputC = "A1", grid with HIT at A1.
* **Expected Output:** outputMsg = "This coordinate has already been hit."

**Test Case 4: Invalid coordinate (e.g., non-existent grid cell)**

* **Input:** inputC = "Z9" (out of bounds).
* **Expected Output:** outputMsg = "Invalid coordinates!"

**Test Case 5: Input format incorrect**

* **Input:** inputC = "12A" (wrong format).
* **Expected Output:** outputMsg = "Invalid coordinates!"

**2.**performRadarSweep**Function**

**Test Case 1: Enemy ship detected**

* **Input:** inputC = "C3", player has radar sweeps left, and a ship exists within the radar area.
* **Expected Output:** outputMsg = "Enemy ships found.", player->sweepsLeft decremented by 1.

**Test Case 2: No enemy ships detected**

* **Input:** inputC = "E5", player has radar sweeps left, no ships in the radar area.
* **Expected Output:** outputMsg = "No enemy ships found.", player->sweepsLeft decremented by 1.

**Test Case 3: No radar sweeps remaining**

* **Input:** inputC = "F7", player->sweepsLeft = 0.
* **Expected Output:** outputMsg = "No more radar sweeps allowed! You lose your turn."

**Test Case 4: Invalid coordinate**

* **Input:** inputC = "X10" (out of bounds).
* **Expected Output:** outputMsg = "Invalid coordinates!"

**3.**applySmokeScreen**Function**

**Test Case 1: Valid smoke screen application**

* **Input:** inputC = "A1", player has used fewer smoke screens than the opponent's sunk ships.
* **Expected Output:** outputMsg = "Smoke screen applied.", player->smokeGrid updated.

**Test Case 2: Exceeding smoke screen limit**

* **Input:** inputC = "D4", player->usedsmokes >= countSunkShips(opponent).
* **Expected Output:** outputMsg = "You cannot use more smoke screens than the ships you've sunk!"

**Test Case 3: Invalid coordinate**

* **Input:** inputC = "J11" (out of bounds).
* **Expected Output:** outputMsg = "Invalid coordinates!"

**Test Case 4: Input format incorrect**

* **Input:** inputC = "AA" (wrong format).
* **Expected Output:** outputMsg = "Invalid coordinates!"

**4.**Artillery**Function**

**Test Case 1: Valid artillery usage**

* **Input:** inputC = "B3", player->prevSunk = 1, and ships exist in the target area.
* **Expected Output:** outputMsg = "Hit!", opponent's grid updated with HIT.

**Test Case 2: Artillery used without sinking a ship in the previous round**

* **Input:** inputC = "C4", player->prevSunk = 0.
* **Expected Output:** outputMsg = "Artillery can only be used in the round right after sinking an opponent's ship!"

**Test Case 3: All misses**

* **Input:** inputC = "E7", no ships in the target area.
* **Expected Output:** outputMsg = "Miss!"

**Test Case 4: Invalid coordinate**

* **Input:** inputC = "K1" (out of bounds).
* **Expected Output:** outputMsg = "Invalid coordinates!"

**5.**Torpedo**Function**

**Test Case 1: Valid torpedo usage**

* **Input:** inputC = "A5", opponent has 3 or more sunk ships, and ships exist in the target row/column.
* **Expected Output:** outputMsg = "Hit!", opponent's grid updated with HIT.

**Test Case 2: Torpedo used without sinking three ships**

* **Input:** inputC = "F6", countSunkShips(opp) < 3.
* **Expected Output:** outputMsg = "Torpedo can only be used after sinking the opponent's third ship!"

**Test Case 3: All misses**

* **Input:** inputC = "H9", no ships in the target row/column.
* **Expected Output:** outputMsg = "Miss."

**Test Case 4: Invalid coordinate**

* **Input:** inputC = "W2" (out of bounds).
* **Expected Output:** outputMsg = "Invalid coordinates!"

**Test Case 5: Input format incorrect**

* **Input:** inputC = "22A" (wrong format).
* **Expected Output:** outputMsg = "Invalid coordinates!"

**CalcProbs.c**

**1. CheckHitOrMiss**

**Test Case 1: Cell is a hit**

* **Input:  
  grid = [["-", "-", "-"], ["-", "H", "-"], ["-", "-", "-"]]  
  target = [1, 1]**
* **Expected Output: 1**
* **Reasoning: The cell at [1][1] contains HIT, so the function should return 1.**

**Test Case 2: Cell is a miss**

* **Input:  
  grid = [["-", "-", "-"], ["-", "M", "-"], ["-", "-", "-"]]  
  target = [1, 1]**
* **Expected Output: 1**
* **Reasoning: The cell at [1][1] contains MISS, so the function should return 1.**

**Test Case 3: Cell is neither hit nor miss**

* **Input:  
  grid = [["-", "-", "-"], ["-", "-", "-"], ["-", "-", "-"]]  
  target = [1, 1]**
* **Expected Output: 0**
* **Reasoning: The cell at [1][1] does not contain HIT or MISS, so the function should return 0.**

**2. CalcCutoffProb**

**Test Case 1: Valid target**

* **Input:  
  player = {probabilityGrid filled with zeros, valid ship bounds}  
  target = [4, 4]**
* **Expected Output: 1**
* **Expected Behavior: The probability for the cell [4][4] should increase based on the ship lengths and available space.**

**Test Case 2: Target out of bounds**

* **Input:  
  player = {valid data}  
  target = [-1, 0]**
* **Expected Output: 0**
* **Reasoning: Target is invalid; the function should return 0 without modifying the grid.**

**Test Case 3: All ships sunk**

* **Input:  
  player = {all ships sunk}  
  target = [3, 3]**
* **Expected Output: 1**
* **Expected Behavior: Since all ships are sunk, no probabilities will be calculated, and the probability for [3][3] remains 0.**

**3. InitalizeCutOffProb**

**Test Case 1: Standard case**

* **Input:  
  player = {probabilityGrid filled with zeros, ship sizes: [5, 4, 3, 2]}  
  target = [3, 3]**
* **Expected Output: 1**
* **Expected Behavior: Probabilities for [3][3] should increase based on ship sizes and the space available.**

**Test Case 2: Target out of bounds**

* **Input:  
  player = {valid data}  
  target = [10, 10]**
* **Expected Output: 0**
* **Reasoning: Target is invalid; the function should return 0 without modifying the grid.**

**4. CalcOverlapProb**

**Test Case 1: Cell surrounded by hits and misses**

* **Input:  
  player = {grid with HITs and MISSes near [4][4]}  
  target = [4, 4]**
* **Expected Output: 1**
* **Expected Behavior: Probabilities for [4][4] should adjust based on the surrounding HIT and MISS cells.**

**Test Case 2: Empty grid**

* **Input:  
  player = {empty grid}  
  target = [3, 3]**
* **Expected Output: 1**
* **Expected Behavior: No adjustments for surrounding cells; probabilities for [3][3] remain unchanged.**

**5. UpdateSurroundingProbabilities**

**Test Case 1: Single ship left**

* **Input:  
  player = {1 ship not sunk, grid without hits or misses}  
  target = [5, 5]**
* **Expected Output: 1**
* **Expected Behavior: Probabilities for the region around [5][5] update based on the remaining ship's size.**

**Test Case 2: Multiple ships left**

* **Input:  
  player = {multiple ships not sunk, grid partially marked with hits and misses}  
  target = [3, 3]**
* **Expected Output: 1**
* **Expected Behavior: Probabilities update, taking into account the overlap and remaining ship lengths.**

**6. UpdateRegionProbabilities**

**Test Case 1: Valid rectangular region**

* **Input:  
  player = {valid data}  
  target = [2, 4, 2, 4]**
* **Expected Output: 1**
* **Expected Behavior: Probabilities for all cells in the specified region are updated.**

**Test Case 2: Region out of bounds**

* **Input:  
  player = {valid data}  
  target = [-1, 3, -1, 3]**
* **Expected Output: 0**
* **Reasoning: Region is invalid; function should return 0.**

**Test Case 3: Region contains sunk ships**

* **Input:  
  player = {region contains sunk ships, valid data}  
  target = [1, 5, 1, 5]**
* **Expected Output: 1**
* **Expected Behavior: Cells that belong to sunk ships should be skipped, and others should be updated.**

**Coordslib.c**

**1. CoordToIndex**

**Purpose: Convert coordinate strings (e.g., "AAA001") to numerical indices.**

* **Test 1: Valid input with a simple numeral system (e.g., coords="A0", startingChar='A', endingChar='Z', shift=0).**
* **Test 2: Valid input with a multi-character system (e.g., coords="AA0", startingChar='A', endingChar='Z', shift=0).**
* **Test 3: Input contains out-of-bound characters (e.g., coords="A!").**
* **Test 4: Invalid input length (e.g., start=0, end=0).**
* **Test 5: Edge of range (e.g., coords="Z9" with system startingChar='A', endingChar='Z').**

**2. GetCoordSplitIndex**

**Purpose: Split the row and column numeral systems in a string.**

* **Test 1: Input with equal numeral systems (e.g., coords="AB12").**
* **Test 2: Input with long row and short column coordinates (e.g., coords="AAA1").**
* **Test 3: Input with invalid separator (e.g., coords="ABC!").**
* **Test 4: Only column numeral input (e.g., coords="1234").**

**3. IsCoordValid**

**Purpose: Validate that a segment of the coordinates is within the bounds of a numeral system.**

* **Test 1: Valid segment (e.g., coords="AAB", start=0, end=3, startingCoord='A').**
* **Test 2: Segment contains an out-of-bound character (e.g., coords="AA!").**
* **Test 3: Segment with partial bounds (e.g., coords="AAB", start=0, end=2).**

**4. IndexWithinRange**

**Purpose: Check if a numerical index falls within the grid size.**

* **Test 1: Index in bounds (e.g., index=5, GRIDSIZE=10).**
* **Test 2: Index out of bounds (e.g., index=-1 or index=15).**

**5. alloc\_ArrayCoordsFromUserCoords**

**Purpose: Allocate memory for and return array indices based on user input coordinates.**

* **Test 1: Valid coordinates (e.g., coords="A1").**
* **Test 2: Invalid coordinates with out-of-bound characters (e.g., coords="A!").**
* **Test 3: Valid multi-character coordinates (e.g., coords="AA12").**
* **Test 4: Coordinates leading to out-of-range indices (e.g., coords="ZZ99" with GRIDSIZE=10).**

**6. alloc\_GridAreaFromInput**

**Purpose: Allocate memory for grid area boundaries based on starting coordinates, orientation, and dimensions.**

* **Test 1: Valid input with horizontal orientation (startingCoords="A1", orientation="h", width=3, height=2).**
* **Test 2: Valid input with vertical orientation (startingCoords="A1", orientation="v", width=3, height=2).**
* **Test 3: Invalid starting coordinates (startingCoords="A!").**
* **Test 4: Out-of-range grid bounds (startingCoords="Z9", width=10, height=10).**
* **Test 5: Memory allocation failure (simulate with invalid malloc).**

**Inputlib.c**

**1. StringToEnumIndex**

**Test Case 1: String found in the array**

* **Input:  
  str = "quit"  
  arr = {"start", "quit", "next"}  
  arrLen = 3**
* **Expected Output: 1**
* **Description: The function should return the index 1 as "quit" is the second string in the array.**

**Test Case 2: String not found in the array**

* **Input:  
  str = "stop"  
  arr = {"start", "quit", "next"}  
  arrLen = 3**
* **Expected Output: -1**
* **Description: The function should return -1 because "stop" is not present in the array.**

**2. alloc\_IntegerToNumeral**

**Test Case 1: Convert small integer to custom numeral system**

* **Input:  
  i = 27, maxInt = 1000, startingChar = 'A', endingChar = 'Z'**
* **Expected Output: "AB"**
* **Description: In a base-26 numeral system, 27 corresponds to "AB".**

**Test Case 2: Convert larger integer to custom numeral system**

* **Input:  
  i = 701, maxInt = 1000, startingChar = 'A', endingChar = 'Z'**
* **Expected Output: "ZZ"**
* **Description: The largest two-character numeral in this system, 701, corresponds to "ZZ".**

**3. alloc\_GetCoordsFromIndices**

**Test Case 1: Generate grid coordinates**

* **Input:  
  row = 3, col = 4, MaxInt = 100, startingCoordinate\_1 = 'A', startingCoordinate\_2 = '1', endingCoordinate\_2 = '9', endingCoordinate\_1 = 'Z', coord1\_shift = 0, coord2\_shift = 0**
* **Expected Output: "D4"**
* **Description: The column index 4 corresponds to "D", and the row index 3 corresponds to "4".**

**4. next**

**Test Case 1: Extract first word from a string**

* **Input:  
  input = "fire artillery torpedo"**
* **Expected Output:  
  Word: "fire"  
  Remaining Input: "artillery torpedo"**
* **Description: The function should extract "fire" as the first word and update the input pointer to the rest of the string.**

**Test Case 2: Extract word when input has leading spaces**

* **Input:  
  input = " smoke screen deployed"**
* **Expected Output:  
  Word: "smoke"  
  Remaining Input: "screen deployed"**
* **Description: The function should skip the leading spaces, extract "smoke", and update the input pointer.**

**5. nextInt**

**Test Case 1: Extract integer from the string**

* **Input:  
  input = "123 456 789"**
* **Expected Output:  
  Integer: 123  
  Remaining Input: "456 789"**
* **Description: The function should extract 123 as the first integer and update the input pointer to the rest of the string.**

**6. alloc\_Input**

**Test Case 1: User enters valid input**

* **User Interaction:**
  + **Prompt: "Enter your command:"**
  + **User Input: "fire"**
* **Expected Output: "fire"**
* **Description: The function should display the prompt, capture the user's input, and return "fire".**

**Test Case 2: User input exceeds max length**

* **User Interaction:**
  + **Prompt: "Enter your command:"**
  + **User Input: "abcdefghijklmnopqrstuvwxyz" (length exceeds MAXINPUTLENGTH)**
* **Expected Output: "abcdefghijklmnopqrstu" (truncated to fit MAXINPUTLENGTH)**
* **Description: The function should truncate input to the maximum allowed length.**

**1/2**

**1. StringToEnumIndex**

**Test Case 1: String found in the array**

* **Input:  
  str = "quit"  
  arr = {"start", "quit", "next"}  
  arrLen = 3**
* **Expected Output: 1**
* **Description: The function should return the index 1 as "quit" is the second string in the array.**

**Test Case 2: String not found in the array**

* **Input:  
  str = "stop"  
  arr = {"start", "quit", "next"}  
  arrLen = 3**
* **Expected Output: -1**
* **Description: The function should return -1 because "stop" is not present in the array.**

**2. alloc\_IntegerToNumeral**

**Test Case 1: Convert small integer to custom numeral system**

* **Input:  
  i = 27, maxInt = 1000, startingChar = 'A', endingChar = 'Z'**
* **Expected Output: "AB"**
* **Description: In a base-26 numeral system, 27 corresponds to "AB".**

**Test Case 2: Convert larger integer to custom numeral system**

* **Input:  
  i = 701, maxInt = 1000, startingChar = 'A', endingChar = 'Z'**
* **Expected Output: "ZZ"**
* **Description: The largest two-character numeral in this system, 701, corresponds to "ZZ".**

**3. alloc\_GetCoordsFromIndices**

**Test Case 1: Generate grid coordinates**

* **Input:  
  row = 3, col = 4, MaxInt = 100, startingCoordinate\_1 = 'A', startingCoordinate\_2 = '1', endingCoordinate\_2 = '9', endingCoordinate\_1 = 'Z', coord1\_shift = 0, coord2\_shift = 0**
* **Expected Output: "D4"**
* **Description: The column index 4 corresponds to "D", and the row index 3 corresponds to "4".**

**4. next**

**Test Case 1: Extract first word from a string**

* **Input:  
  input = "fire artillery torpedo"**
* **Expected Output:  
  Word: "fire"  
  Remaining Input: "artillery torpedo"**
* **Description: The function should extract "fire" as the first word and update the input pointer to the rest of the string.**

**Test Case 2: Extract word when input has leading spaces**

* **Input:  
  input = " smoke screen deployed"**
* **Expected Output:  
  Word: "smoke"  
  Remaining Input: "screen deployed"**
* **Description: The function should skip the leading spaces, extract "smoke", and update the input pointer.**

**5. nextInt**

**Test Case 1: Extract integer from the string**

* **Input:  
  input = "123 456 789"**
* **Expected Output:  
  Integer: 123  
  Remaining Input: "456 789"**
* **Description: The function should extract 123 as the first integer and update the input pointer to the rest of the string.**

**6. alloc\_Input**

**Test Case 1: User enters valid input**

* **User Interaction:**
  + **Prompt: "Enter your command:"**
  + **User Input: "fire"**
* **Expected Output: "fire"**
* **Description: The function should display the prompt, capture the user's input, and return "fire".**

**Test Case 2: User input exceeds max length**

* **User Interaction:**
  + **Prompt: "Enter your command:"**
  + **User Input: "abcdefghijklmnopqrstuvwxyz" (length exceeds MAXINPUTLENGTH)**
* **Expected Output: "abcdefghijklmnopqrstu" (truncated to fit MAXINPUTLENGTH)**
* **Description: The function should truncate input to the maximum allowed length.**