

## Specifications: (sectioned somewhat for better readability)

### Game Setup and Initialization Functions:

- **char set\_game\_difficulty();**
  - *// Requires: None*
  - *// Effects: Prompts user to select game difficulty level, returns difficulty char ('E', 'H')*
- **void initialize\_player(Player\* player);**
  - *// Requires: Valid Player pointer not NULL*
  - *// Effects: Initializes player's turn, boards, ships, and the number of different moves they are allowed and their sunken ships counter.*
- **void initialize\_board(char board[GRID\_SIZE][GRID\_SIZE]);**
  - *// Requires: Valid board array of GRID\_SIZE x GRID\_SIZE*
  - *// Effects: Populates board with initial empty/default state*
- **void initializeBotPlayer(Player \*bot);**
  - *// Requires: Valid Player pointer for bot*
  - *// Effects: Initializes player's turn, boards, ships, and the number of different moves they are allowed and their sunken ships counter by calling initialize\_player(Player \*player) and calls initHuntQueue(HuntQueue \*queue) to initialize the bot's queue.*
- **void startGame(Player \*currentPlayer, Player \*opponent, char game\_difficulty);**
  - *// Requires: -human != NULL: Valid pointer to the 1st player*  
*- bot != NULL: Valid pointer to the 2nd player*  
*- game\_difficulty is a valid difficulty level ('E', 'H')*  
*- Both players are initialized and have their ships placed on their respective boards*
  - *// Effects: - Modifies player stats by calling selectMove()*  
*- Continues game loop until one player sinks all opponent's ships*  
*- Prints the winner's name at game conclusion*  
*- Displays final board states for both players*

### Board and Display Functions:

- **void displayBoard(Player \*player);**
  - *// Requires: Valid Player pointer with initialized board*  
*- The GRID\_SIZE constant should be defined.*
  - *// Effects: Outputs to the console the player's game, showing ship positions*
- **void display\_opponent\_grid(char board[GRID\_SIZE][GRID\_SIZE], char game\_difficulty);**
  - *// Requires: Valid board array, valid game difficulty*
  - *// Effects: Outputs to the console the opponent's grid based on current game difficulty*

## Ship Placement Functions:

- **void placeShips(Player \*player);**
  - *// Requires: - Valid Player pointer*  
*-The player's ships array should be initialized with Ship structs.*
  - *// Effects: No return value; modifies player's ship placements by prompting for valid positions and places ships by calling placeShipOnBoard*
- **int checkShipOverlap(Player \*player, Ship \*ship, int startRow, int startCol, char orientation);**
  - *// Requires: Valid player, ship, start coordinates, orientation*
  - *// Effects: Checks if ship placement would overlap with existing ships. Returns 1 if valid placement, 0 otherwise.*
- **void placeShipOnBoard(Player \*player, Ship \*ship, int startRow, int startCol, char orientation);**
  - *// Requires: Valid player, ship, start coordinates, orientation*
  - *// Effects: Places ship on player's board at specified location*

## Player and Move Management Functions:

- **void selectMove(Player \*attacker, Player \*defender, char game\_difficulty);**
  - *// Requires: Valid player pointers, game difficulty*
  - *// Effects: Allows player to choose and execute a move against opponent*  
*If the attacker is a bot, it calls botSelectMove and switches players.*  
*For human players:*  
*Prompts the player for a move input in the format: MoveType Coordinate.*  
*Validates and processes different move types (Fire, Torpedo, Artillery, Radar, Smoke).*  
*Executes the selected move and switches turns when valid.*  
*Tracks and updates special moves.*
- **void selectBotCoordinate(Player \*bot, Player \*opponent, int \*x, int \*y, char moveType);**
  - *// Requires: Valid Bot and opponent Player pointers, pointers for x/y coordinates*
  - *// Effects: Bot selects optimal attack coordinates based on game strategy*
- **char selectBotMoveType(Player \*bot);**
  - *// Requires: Valid bot Player*
  - *// Effects: returns best move type('T','A','R','S','F') for bot based on current game state*
- **void botSelectMove(Player \*bot, Player \*human, char game\_difficulty);**
  - *// Requires: Valid player pointers*
  - *// Effects: -Selects a move type for the bot (Fire, Artillery, Torpedo, Radar, or Smoke).*  
*-Selects coordinates for the move based on the selected move type.*  
*-Executes the selected move by calling move execution functions*

*-Modifies the bot's resources (e.g., artillery, torpedoes, radar, smoke screen*

## Move Execution Functions:

- **void FireMove(Player\* attacker, Player\* defender, int x, int y, char game\_difficulty);**
  - *// Requires: Valid player pointers, target coordinates, game difficulty*
  - *// Effects: Executes standard fire attack at specified coordinates*
- **void ArtilleryMove(Player\* attacker, Player\* defender, int x, int y, char game\_difficulty);**
  - *// Requires: Valid player pointers, target coordinates, game difficulty*
  - *// Effects: Executes artillery attack at specified coordinates*
- **void TorpedoMove(Player\* attacker, Player\* defender, int x, int y, char game\_difficulty);**
  - *// Requires: Valid player pointers, target coordinates, game difficulty*
  - *// Effects: Executes torpedo attack at specified coordinates*
- **void RadarMove(Player \*attacker, Player \*defender, int x, int y);**
  - *// Requires: Valid player pointers, coordinate*
  - *// Effects: Reveals information about opponent's grid in a specific area*
- **void SmokeMove(Player \*attacker, int x, int y);**
  - *// Requires: Valid attacker Player, coordinates*
  - *// Effects: Creates a smoke screen to obscure ship positions*
- **void markAffectedArea(int x, int y, char moveType, char orientation);**
  - *// Requires: Valid coordinates, move type, and orientation*
  - *// Effects: Marks grid cells affected by a specific move type*
- **void HitOrMiss(Player \*attacker, Player \*defender, int x, int y, char movetype, char orientation, char game\_difficulty);**
  - *// Requires: Valid Player pointers for attacker and defender*
    - *Attack coordinates (x, y) must be within game grid (0-9)*
    - *movetype must be a valid attack type*
    - *orientation must be a valid orientation*
  - *// Effects:*
    - *Updates defender's grid with hit (\*) or miss ('o') markers*
    - *Tracks individual ship damage*
    - *Increments ships sunk counter*
    - *Displays opponent's grid*
    - *Displays hit/miss message*
    - *Grants artillery weapon when first ship sinks \* - Grants torpedo weapon when third ship is sunk*

*For bot , adds adjacent unexplored cells to potential targets*

## Utility and Validation Functions:

- **void playerswitch(Player \*attacker, Player \*defender);**
  - *// Requires: Two valid Player pointers*

- *// Effects: Swaps roles between attacker and defender players*
- **int is\_fire(char\* moveType);**
  - *// Requires: valid pointer to a char*
  - *// Effects: Case insensitive.*

*returns 1 if moveType was fire, 0 otherwise*
- **int is\_artillery(char\* moveType);**
  - *// Requires: valid pointer to a char*
  - *// Effects: Case insensitive.*

*returns 1 if moveType was artillery, 0 otherwise*
- **int is\_torpedo(char\* moveType);**
  - *// Requires: valid pointer to a char*
  - *// Effects: Case insensitive.*
- **int is\_radar(char\* moveType);**
  - *// Requires: valid pointer to a char*
  - *// Effects: Case insensitive.*

*returns 1 if moveType was radar, 0 otherwise*
- **int is\_smoke(char\* moveType);**
  - *// Requires: valid pointer to a char*
  - *// Effects: Case insensitive.*

*returns 1 if moveType was smoke, 0 otherwise*
- **int is\_equal(char\* str1, char\* str2);**
  - *// Requires: valid pointers to the two strings*
  - *// Effects: Case insensitive.*

*returns 1 if the two strings are equal, 0 otherwise*
- **void displayAvailableMoves();**
  - *// Requires: None*
  - *// Effects: Outputs list of available moves directly to the console*
- **void clear\_screen();**
  - *// Requires: None*
  - *// Effects: Clears the console/terminal screen*
- **int column\_to\_index(char column)**
  - *// Requires: None*
  - *// Effects: Converts alphabetic column to numeric index*
- **int isShipSunk(Ship \*ship);**

- *// Requires: Valid ship pointer*
- *// Effects: Determines if a specific ship has been completely destroyed*
- **void HitOrMissMessageDisplay(int movesuccess);**
  - *// Requires: None*
  - *// Effects: Displays message indicating attack outcome*
- **int isBot(Player \*player);**
  - *// Requires: Valid Player pointer*
  - *// Effects: returns 1 if player is the Bot, 0 otherwise*

## BOT Strategy Functions:

- **void addAdjacentUnexploredCells(Player \*bot, Player \*opponent, int x, int y);**
  - *// Requires: Valid Bot and opponent Player*
  - *// Effects: Adds unexplored adjacent cells to Bot's tracking queue*
- **void findVulnerableRegions(Player \*bot, int \*bestX, int \*bestY);**
  - *// Requires: Valid Bot Player and coordinates pointers*
  - *// Effects: Identifies most vulnerable areas on Bot's grid. Sets \*bestX and \*bestY to the chosen coordinates; If a vulnerable area is found, it will store the coordinates of that area.*
- **void findDenseClusterOrRandom(Player \*bot, int \*bestX, int \*bestY);**
  - *// Requires: Valid Bot Player and coordinates pointers*
  - *// Effects: Identifies most dense areas on Bot's grid. Sets \*bestX and \*bestY to the chosen coordinates; If a dense cluster is found, it will store the coordinates of that cluster; otherwise, it will store a random coordinate.*
- **float calculateUnexploredPercentage(Player \*bot);**
  - *// Requires: valid Bot Player*
  - *// Effects: Calculates the percentage of unexplored cells on the opponent's grid. Returns A floating-point value representing the percentage of unexplored cells ('~' cells) on the grid.*
- **int calculateVulnerabilityScore(Player \*bot);**
  - *// Requires: valid Bot Player*
  - *// Effects: Calculates the vulnerability score of the bot's ships based on the number of unhit cells in partially damaged ships. Returns An integer value representing the bot's vulnerability score.*
- **int calculateProtectionScore(Player \*bot, int x, int y);**
  - *// Requires: valid Bot Player*
  - *// Effects: returns the protection score of a certain 2x2 area (x and y are starting point)*
- **float calculateRadarThreshold(Player \*bot);**
  - *// Requires: valid Bot player*

- *// Effects: Calculates the threshold for when the bot should consider using a Radar Move. Returns a floating-point value representing the threshold for the bot to use radar.*

## Utility String Functions:

- **void stringcopy(char\* dest, char\* src);**
  - *// Requires: Valid source and destination string pointers*
  - *// Effects: Safely copies string from source to destination*
- **void to\_lowercase(char\* src, char\* dest);**
  - *// Requires: Valid source and destination string pointers*
  - *// Effects: Converts source string to lowercase in destination*

## Hunt Queue Management Functions:

- **void initHuntQueue(HuntQueue \*queue);**
  - *// Requires: Valid HuntQueue pointer*
  - *// Effects: Initializes hunt queue to empty state*
- **int isHuntQueueEmpty(HuntQueue \*queue);**
  - *// Requires: Valid HuntQueue pointer*
  - *// Effects: Checks if hunt queue contains no elements. Returns 1 if empty, 0 otherwise.*
- **void enqueueHunt(HuntQueue \*queue, int x, int y);**
  - *// Requires: Valid HuntQueue, coordinates*
  - *// Effects: Adds coordinates to hunt queue*
- **void dequeueHunt(HuntQueue \*queue, int \*x, int \*y);**
  - *// Requires: Valid HuntQueue, coordinate pointers*
  - *// Effects: Removes and returns top coordinates from queue*
- **int isHuntQueueFull(HuntQueue \*queue);**
  - *// Requires: Valid HuntQueue pointer*
  - *// Effects: Returns 1 if full, 0 otherwise.*