**Battleship-KAZOO  
  
Specifications  
  
  
Methods:  
  
1. InitializePlayer**

**Signature: void InitializePlayer(player \*p)**

**Requires: p: A valid pointer to a player object is initialized**

**Effects:**

**Initializes the player's grids (game grid, smoke grid, and opponent grid)**

**Set all ships as not sunk**

**Resets game variables (e.g., radar count, smoke count, and ability unlock)  
  
No return value**

**2. CreateGrid**

**Signature: void CreateGrid(char grid[10][10])**

**Requires: char grid [10][10]: A valid 10x10 grid array is initialized**

**Effects:**

**Initializes every cell in the grid to ~ (water)  
No return value**

**3. PlaceShip**

**Signature: void PlaceShip(player \*player, int isBot)**

**Requires: player: A valid pointer to a player object is initialized (the one that is placing ships)**

**isBot: An integer to determine if the player is human or bot (0 for human player, 1 for bot).**

**Effects:**

**Places four ships of varying sizes on the grid (Carrier, Battleship, Destroyer, Submarine)  
Ensures all ships are placed within bounds and without overlap  
Updates grid with the ship placement  
For bots, ship placement is randomized.  
No return value**

**4. Fire**

**Signature: void Fire(player \*opponent, int row, char column, int diff, player \*self)**

**Requires:**

**opponent: A valid pointer to the opponent’s player object (the one receiving the attack).**

**row: an integer specifying the target row (1-based index).**

**column: a character specifying the target column (A-J).**

**diff: An integer specifying difficulty that affects the display (0 for easy, 1 for hard).**

**self: A valid pointer to the current player object(the one that’s doing the attacks).**

**Effects:**

**Updates the opponent's grid with a hit (\*) or miss (o).**

**Checks for ship damage and sinks ships when all parts are hit.  
  
Handles logic for sinking ships, updating bot states.  
Prints “Hit!” or “Miss!”**

**Unlocks abilities (artillery and torpedo) based on sunk ships.**

**5. RadarSweep**

**Signature:int RadarSweep(player \*opponent, int row, char column, player \*self)**

**Requires:**

**opponent: A valid pointer to the opponent’s player object(the one being scanned)**

**row: An integer specifying the center row (1-based index)**

**column: A character specifying the center column (A-J)**

**self: A valid pointer to the current player object(the player scanning)**

**Effects:  
Consumes a radar sweep**

**Scans a 2x2 area around the specified location on the opponent’s grid.  
Prints either “No enemy ship(s) found” or “Enemy ship(s) found” accordingly**

**Returns 1 if a ship is found in the area; otherwise, returns 0.**

**6. SmokeScreen**

**Signature: void SmokeScreen(player \*self, int row, char column)**

**Requires:**

**self: A valid pointer to the current player object.(the player deploying the smoke screen)**

**row: An integer specifying the center row (1-based index)**

**column: A character specifying the center column (A-J)**

**Effects:**

**Deploys a smoke screen over a 2x2 area on the grid.**

**Marks the area with ~ on the smoke grid**

**Decrements the player’s smoke count  
No return value**

**7. Artillery**

**Signature: void Artillery(int row, int col, player \*opponent, player \*self, int diff)**

**Requires:**

**row: An integer specifying the target row (1-based index).**

**col: An integer specifying the target column (0-based index).**

**opponent: A valid pointer to the opponent’s player object (the player being attacked)**

**self: A valid pointer to the current player object**

**diff: An integer (0 for easy, 1 for hard) affecting the display**

**Effects:**

**Fires at a 2x2 area centered around the specified location**

**Updates the opponent's grid with hits or misses and handles ship sinking logic  
Prints the number of hits and whether any ships are sunk**

**Marks the artillery as used for the player  
No return value**

**8. Torpedo**

**Signature:**

**void Torpedo(player \*opponent, char target, int diff, player \*self)**

**Requires:**

**opponent: A valid pointer to the opponent’s player object(the one being attacked)**

**target: A character specifying the row (1-9) or column (A-J) to target.**

**diff: An integer (0 for easy, 1 for hard).**

**self: A valid pointer to the current player object.**

**Effects:**

**Fires a torpedo at the specified row or column.**

**Updates the opponent’s grid with hits or misses.  
Prints the number of hits and whether any ships are sunk**

**Marks the torpedo as used for the player.**

**9. UpdateProbabilityMap**

**Signature: void UpdateProbabilityMap(player \*bot)**

**Requires:**

**bot: A valid pointer to the bot’s player object(the bot that’s updating its probability map)**

**Effects:**

**Updates the bot's probability map based on known hits, misses, and remaining ships based on the bots logic   
No return value**

**10. BotMakeMove**

**Signature: void BotMakeMove(player \*bot, player \*human, int diff)**

**Requires:**

**bot: A valid pointer to the bot’s player object (the bot making the move)**

**human: A valid pointer to the human player’s player object**

**diff: An integer (0 for easy, 1 for hard)**

**Effects:**

**Makes a move for the bot based on probability, unlocked abilities, and remaining abilities as well  
Prints the Bots action to the console   
No return value**

**11. deploySmokeScreenOverShip**

**Signature:void deploySmokeScreenOverShip(player \*bot)**

**Requires:**

**bot: A valid pointer to the bot’s player object (the player using the smoke screen)**

**Effects:  
Selects one of the unsunk ships randomly and after looking at all possible positions for the selected unsunk ship chooses a random position**

**Deploys a smoke screen by calling SmokeScreen over a unsunk ship on the bot’s grid.**

**No return value**

**12. selectHighestProbabilityCell**

**Signature: void selectHighestProbabilityCell(player \*bot, int \*targetRow, int \*targetCol)**

**Requires:**

**bot: A valid pointer to the bot’s player object.**

**targetRow: Pointer to an integer where the selected row index will be stored.**

**targetCol: Pointer to an integer where the selected column index will be stored.**

**Effects:**

**Selects the cell with the highest probability after the call to updating the probability map  
Updates targetRow and targetCol with the the chosen cell  
No return value**

**13. checkForSunkShips**

**Signature: void checkForSunkShips(player \*bot, player \*human)**

**Requires:**

**bot: A valid pointer to the bot’s player object(bot that’s checking opponents ships)**

**human: A valid pointer to the human player’s player object(whose ships are being checked)**

**Effects:  
Handles changes in number of sunk ships**

**Checks if any ships have been sunk by the bot**

**Unlocks abilities (artillery and torpedo) as required  
No return value**

**14. selectHighestProbabilityArea**

**Signature:void selectHighestProbabilityArea(player \*bot, int \*targetRow, int \*targetCol)**

**Requires:**

**bot: A valid pointer to the bot’s player object (the bot making a selection)**

**targetRow: Pointer to an integer where the target row will be stored**

**targetCol: Pointer to an integer where the target column will be stored**

**Effects:**

**Identifies the 2x2 area with the highest cumulative probability from the bot’s map.  
Updates targetRow and targetCol with the chosen area.**

**Does not return a value.**

**15. decideRowOrColumn**

**Signature: int decideRowOrColumn(player \*bot, char \*target)**

**Requires:**

**bot: A valid pointer to the bot’s player object.**

**target: A pointer to a character where the chosen row or column will be stored.**

**Effects:**

**Decides whether to target a row or column for torpedo attacks based on probability.  
Updates target with the chosen row or column.**

**Returns 1 for row targeting, 0 for column targeting.**

**16. compare\_strings**

**Signature:int compare\_strings(const char \*str1, const char \*str2)**

**Requires:**

**str1 and str2: are valid strings to compare**

**Effects:**

**Compares the two strings   
Returns 1 if they are equal, otherwise returns 0.**

**17. handle\_move**

**Signature:void handle\_move(player \*current\_player, player \*opponent, int diff)**

**Requires:**

**current\_player: A valid pointer to the current player object (whose making the move)**

**opponent: A valid pointer to the opponent’s player object**

**diff: An integer (0 for easy, 1 for hard).**

**Effects:**

**Handles the player’s move by processing commands like Fire, Radar, Smoke, Artillery, or Torpedo.  
Prints the correct messages for each action  
No return value**

**18. displayGrid**

**Signature: void displayGrid(char grid[10][10], int diff)**

**Requires:**

**grid: A valid 10x10 grid array.**

**diff: An integer (0 for easy, 1 for hard).**

**Effects:**

**Displays the grid with different visibility rules based on the difficulty level.  
Prints the grid to the console   
No return value  
  
20. isValidPlacement  
  
Signature: int isValidPlacement(player \*player, char column, int row, char orientation, int size);  
Requires:** **player: Pointer to the player placing the ship.**

**column: The starting column for placement (e.g., 'A')**

**row: The starting row for placement.**

**orientation: The orientation of the ship ('H' or 'V')**

**size: The size of the ship**

**Effects:**

**Validates whether the ship can be placed at the specified location.**

**Returns 1 if valid, 0 otherwise**

**19. main**

**Signature:int main()**

**Requires: None.**

**Effects:**

**Initializes the players and the game.**

**Handles turns and checks for game-over conditions.**

**Frees allocated memory at the end.**