T. Kelly, A. Kuang, M. Lee, J. Sizoo, and M. Swartz (Group 14)
Dr. David Johnson
EECS 448

Project 2 – Code Documentation

Battleship Software

1. **Battleship.css**: CSS file to make the buttons and grid pieces look pretty.
2. **Battleship.html**: the html file the user will open in the browser. This contains several divs (referred to in code as "sections" or "windows"), used for different stages of the game.
3. **GraphicsUI.js**: general I/O interface. Contains functions to move and modify the DOM elements, as well as handlers to take input events and pass the necessary information to the proper backend file.
4. **ShipPlacement.js**: backend logic for the ship placing stage of the game. Checks for valid placement of ships, and starts the main game when placement is complete.
5. **MainGame.js**: backend logic for the main ("attacking") phase of the game. Handles player guesses, checks for hits, and ends the game when a player wins.
6. **AI.js**: backend logic for AI functionality. Includes ship placement and easy, medium, and hard game modes.

---

1. Battleship.css

- Battleship.css implements the browser visuals as outlined and setup in the html file.
- The html div's characteristics and properties were set and stored in this file.

---

2. Battleship.html

- Battleship.html implements the web template for the game.
- There are separate windows that are loaded on the webpage. All windows are shown in Table 1 below. Each button is connected to a JavaScript function from GraphicsUI.js.
- The status of the JavaScript files can affect the html file: number and location of ships on board, hit and miss of attacks, instructional message, etc.

Table 1. Battleship.html

Div ID	Description
shipNumPick	Shows 5 buttons for the user to select the number of ships. The user can have 1 to 5 ships. Each button calls setNumShipsChoice() when clicked.
specialShot	Shows 3 buttons for the user to select the number of special shots. The buttons are for 1, 2, and 3 special 3x3 shots. Each button calls setSpecialShot() when clicked. The function is defined in GraphicsUI.js.
chooseOp	Shows 2 buttons for the user to select the opponent: human or AI. Clicking on either button will call setOpponent(). Clicking on AI will show the "selectDifficulty" div.
selectDifficulty	This div is hidden unless the user chooses an AI opponent in the "chooseOp" div. Shows 3 buttons: easy, medium, and hard. Clicking on a button will call setDifficulty().
startGame	Shows 1 button: start game. Clicking it will call createUI().
gameOver	Displays the game over message
p1View	Shows the player 1 view, which includes instruction text, a button to rotate a ship during placement, and a special shot button. Clicking the rotate ship button will call rotateShip(). Clicking the fire special shop button will call initSpecShot().
p2View	Same as "p1View" but for player 2.
transition	Displays the player transition message and a next player button. Clicking the button calls handleTransition().
newGame	Shows 1 button to reset the game. Clicking this button will reload the window, which effectively restarts the game.

3. GraphicsUI.js

- GraphicsUI.js builds the visual user interface and handles everything the players directly interact with.
- There are four boards with the following IDs: “p1HomeBoard”, “p1AttackBoard”, “p2HomeBoard” and “p2AttackBoard”. Each tile of the boards has the following template for its ID: [coordinate][player][board], e.g. “a01p1HomeBoard”, “j10p2AttackBoard”, etc.

Table 2. GraphicsUI.js	
Function	Description
setNumShipsChoice(ships)	Sets the number of ships as selected by the user. Called by clicking the number button in the UI start menu.
setSpecialShot(count)	Sets the number of special shots for both users. Called by clicking the number button in the UI start menu.
setOpponent(op)	Sets the opponent type: human or AI. Called by clicking an opponent button in the UI start menu.
setDifficulty(chooseDifficult)	Sets the difficulty of the AI: easy, medium, or hard. Called by clicking a difficulty button in the UI start menu.
showDifficulty()	Displays the AI difficulty options for the user.
createUI()	Builds the UI for the players. First, it checks if all necessary buttons have been clicked. Then it hides all start menu options. Then it draws the player boards and ships. Calls setBigShotHover() and initializeShipPlacement().
drawGrid(gridId, gridClass)	Creates and return a grid for the player gameboard. This includes the board title, a-j labels, and 1-10 lables, and a 10x10 ship tiles.
setBigShotHover()	Enables mouseover hover effects for the attack boards.
drawShips(numberOfShips, player)	Creates and returns an inventory box. The box includes all ships to be placed.
getNeighborCells(cell)	Returns an array of cells that surround the parameter cell. There are 9 or fewer neighboring tiles.
parseTileClick(tile)	Called when the user clicks on a tile. The function will call the appropriate handler for the current game phase/turn. Takes a string parameter that represents the tile ID; the format is [coordinate][player][board], e.g. “a01p1HomeBoard”.
initSpecShot()	Changes the bool state when a special shot is fired.
rotateShipButton()	Calls the rotate button in the html file.
parseTileHover(tile)	Is called when the user hovers over a tile. Calls the corresponding handler for the game phase.
moveShip(shipId, tileId, isVertical)	Moves the ship to the destination given in the parameters: a string representing the ID of the ship div element which

	should be in the format [player]-[ship type], i.e. “p1-1TileShip”, a string representing the ID of the tile, which should be in the format described above, and a boolean representing if the ship is horizontal or vertical (true if vertical, false if horizontal)
setShipProperties(shipId, opacity, color)	Sets the opacity and color of a ship.
setTileState(tileId, isHit)	Changes the appearance of a tile depending on its hit status. Sets the state of a tile when it is targeted to either targeted but not hit (when there is no ship present in that tile) or target and hit (when there is a ship). It takes in two parameters: a string representing the ID of the tile which should be in the above format and a boolean representing if there is a ship present in that tile or not (true if there is a ship and false if there is not a ship)
drawHitMark(tileId)	Creates a X mark to be placed over a tile.
switchWindow(windowId)	Displays one of the four windows: p1View, p2View, gameOver, and shipNumPick.
updateTransitionText(text)	Sets the text shown in the transition window. It has a parameter which is a string of text that whoever is calling it wants to be displayed in the HTML div element with the id = "transitionText", this is used to update the user of a hit or miss on a turn then to instruct the user to pass the computer to the opposing player.
updateTransitionTarget(windowId)	Sets the transition target. It has a parameter of a string that sets the value of that string to the transitionTarget string. This function works with the handleTransition() function that uses the updated state that the updateTransitionTarget() just updated.
handleTransition()	Switches the window and draws an X if a hip is hit. This is a helper function that calls the switchWindow() function passing in the current transitionTarget as the parameter.
setGameOverText(text)	Sets text for the game over window. It receives a string parameter “text” and sets the text to the GameOver window. It gets called when certain win conditions are met.
hideElement(id)	Hide the html element
setInstruction(text, player)	Check for the current player and update the text. This function has two parameters, "text" is a string of text and player is an int associated to whichever player's turn it is. This function works by checking if the player value passed

	is 1 or 2, if it's player 1 it grabs the dom element by id "p1InstructionText" if player 2 it grabs "p2InstructionText" then sets the content of that tag to be the text that you pass into the function.
Variable	Description
p1Ships	List of Player 1's ships
P2Ships	List of Player 2's ships
gameState	Tracks the game state (i.e. p1Place, p2Turn, numShipSelection)
numShipsChoice	Number of ships for a user.
numShipsChosen	Bool that is true after the user selects how many ships to play with.
p1SpecShot	Number of special shots that Player 1 has
p2SpecShot	Number of special shots that Player 2 has
specialShotChosen	Bool that is set to true after the user selects how many special shots to play with.
opponent	Tracks if the opponent is human or AI
opponentChosen	Bool that is true after the user selects the opponent to play with.
difficulty	Tracks the difficulty of the AI: easy, medium, or hard.
difficultyChosen	Bool that is true after the user selects the AI difficulty to play with.
fireSpecShot	True when the special shot is activated
currentWindow	Tracks the current window for the UI
transitionTarget	Tracks the target window for the UI
columnLabelAlphabet	Stores the column coordinate alphabet list.
aiHitTileID	Tracks the AI hit cell.

4. ShipPlacement.js

- ShipPlacement.js contains functions and logic for handling the stage of the game when players are placing ships. It is initialized by createUI() in GraphicsUI.
- ShipPlacement assumes the p1Ships and p2Ships lists are empty and will populate these lists as the players place their ships.
- Several functions in ShipPlacement.js make use of the moveShip and setShipProperties functions in GraphicsUI.js to render previews and final placements of ships.

Table 3. ShipPlacement.js	
Function	Description
initializeShipPlacement(_numShips)	Called after the user decides how many ships to play with. It switches to player 1's view, updates the global variables to the correct initial values.
initializeP2Placement()	Helper function for player 2 ship placement that is called internally.
hoverCell(cell)	Used to render a preview of the current ship being placed, coloring it red if the location is invalid. The cell parameter is a full DOM element ID, in a form like "c04p1HomeBoard".
rotateShip()	Flips the isVertical property of the current ship being placed (see the variable nextShip). It is called by the "Rotate Ship" button in the p1View and p2View windows of the game.
attemptShipPlace(cell)	This function is the handler for the user clicking a tile on their own board. If the placement is invalid, the function does nothing. Otherwise, it places the ship, adds the ship to the appropriate ship list, and moves on to placing the next ship. If no more ships need to be placed, it moves on to player 2 ship placement or uses the transition window to move to player 1's first turn. The format of the cell parameter is the same as in hoverCell.
placeShip(ship)	Helper function to place a ship and update the ship list
isShipValid(ship)	Returns true if the ship placement is at a valid location, returns false otherwise.
initializeShip(_length, _topLeft = "a01", _isVertical = false)	Creates and returns a ship object.
initializeTestFunctions()	Create test functions for ship placement.
getShipID(length)	Returns the ID of the ship: p#+length+TileShip
Variable	Description

numShips	Total number of ships per player. Is set in initializeShipPlacement() and remains the same for the rest of the game.
shipsRemaining	Number of ships the current player still needs to place. It is used to determine both what the length of a ship should be, as well as when the player is done placing ships.
nextShip	JavaScript object of the form {length: int, topLeft: string, isVertical: boolean}. nextShip.topLeft is a full DOM ID, e.g. “b10p1HomeBoard”, describing the location of the topmost/leftmost tile of the ship.
isP2	Tracks whether player 1 or player 2 is currently placing their ships.
shipList	Pointer to either p1Ships or p2Ships in GraphicsUI.js, depending on who is placing ships.

5. MainGame.js

- MainGame.js' role is to provide the backend logic during the main part of the game, e.g., when users are attacking each other's battleships
- Initialization
    - MainGame code assumes that the users have placed their battleships on the board
    - At the top of the code, some global variables are declared (explained below) as well as two empty boards (10x10 arrays) that are filled with zeroes. These boards are used to represent the board a player is attacking (explained below)
    - Using the ship arrays filled out by ShipPlacement.js, createCoordinateArray() creates an array of tiles (e.g. "e04", "e05") that is associated with each ship, which is used for scanning the boards and determining whether a player's guess is a hit.
    - initializeGame() calls the setInstruction() function located in GraphicsUI.js, which displays text that instructs the player on what to do
- Player turns
    - If the user chose to play against an AI opponent, player 2 is controlled by the AI within the softeware. If the user chose to play against a human component, player 2 is open to human control.
    - The turn of each player is controlled by a global variable called "turn", which takes on two values depending on whether it's player 1's or player 2's turn. The "turn" variable is used to indiscriminately access an array containing two empty boards declared in initializeGame(), one of which represents player 1's board that they are guessing on, the other of which player 2's
    - Each turn, a player's click calls the function guessCell(cell), which scans the coordinate arrays of each player's ships to determine whether the cell is a hit or miss guessCell(cell) checks for repeat guesses using isGuessed(cell), if a cell has already been guessed, nothing happens, and the player must click a different cell for the game to progress
    - guessCell(cell) calls a function callSetTileState(cell, isHit), this function is responsible for reconstructing the tileID's and calling setTileState(tileID, isHit) in GraphicsUI.js, visually updating the board the player is attacking. In guessCell(cell), the "cell" passed to the function has the format <column><row><player><attack or home board>, for example "e04p1HomeBoard", this string is truncated in guessCell() then rebuilt in updateGuessedBoard(cell, isHit) to make parsing the ship arrays simpler.
    - updateGuessedBoard(cell, isHit) is responsible for updating the board a player is attacking—each player's guessed board is initially filled with zeroes (default/not guessed). If the cell guessed by a player is a hit, that cell of the guessed board is changed to a 1, while a miss changes that cell to a 2
    - After a guess has been made the function switchTurns() is called, which changes the turn index and the gameState to reflect the other player, and calls switchWindow(text) and updateTransitionTarget(text) to change the game's visuals
- End Game
    - The end of the game is controlled by three global variables: p1Hits, p2Hits, and maxHits.

- The value of maxHits depends on the number of ships on the board: if there are 5 ships, for example, the maximum number of hits a player can get is 5+4+3+2+1 = 15.

Table 4. MainGame.js	
Function	Description
initializeGame()	Initialized the game according to the player's ship number and ship placement. Creates coordinate arrays and sets instruction text.
guessCell(cell)	Scans all components of the ship array to determine whether a guessed tile is a hit or miss. If the guess is a hit, the hit counter, player boards, and transition text is updated. Checks if the game is over. If the guess is a miss, update the board and transition text. Lastly, switch turns.
guessCells(cells)	Similar to guessCell(), but adapted for special shots.
checkSunk(ship)	Returns true of the ship is sunk, false otherwise.
isGuessed(cell)	Checks if a cell has been guessed or not.
updateGuessedBoard(cell, isHit, ship)	Update the board with hit or missed shots.
callSetTileState(cell, isHit)	Creates a tileID. The ID is passed to GraphicsUI.js through setTileState().
switchTurns()	Switches the turn index, updates the game state, and updates the visuals.
updateHitCounter()	Update the total hit count for a player.
isOver()	Checks if the game is over. Game ends when a player has hit all ships.
endGame()	Ends the game by switching to the gameOver window and updates the text.
wait(ms)	Stalls code execution.
createEmptyBoard()	Creates an empty 2D array for a player board.
createCoordinateBoard()	Helper function to sink ships. Creates a board array containing coordinates.
createCoordinateArray(shipArray)	Takes a players ship array as a parameter. Turns the array info of top left, orientation, and length for each ship into board coordinates.
printCoordinateArray(shipArray)	Prints the coordinants that the shipArray parameter occupies.
nextChar(c)	Increments a character. Used for creating coordinate arrays and string concatenation.
Variable	Description
turn	Tracks whose turn is it: player 1 (0) or 2 (1).
p1Hits	Number of ship hits. Starts at 0.

p2Hits	Number of ship hits. Starts at 0.
maxHits	Maximum number of hits possible. Used to track how close players are to ending the game.
targetShip	Tracking variable for medium AI
targetLoci	Tracking variable for medium AI
p1GuessedBoard	Array of player boards.
p2GuessedBoard	Array of player boards
arrGuessedBoard	Array containing both player 1 and 2 boards. Accessed using arrGuessedBoard[turn]

6. AI.js

- The role Ai.js is to provide the backend logic for the AI. The AI played the role of Player 2 in the main game.
- Placing ships
    - The AI uses a looping function to iterate through all ships. The row, column, and orientation are generated randomly. Each ship placement coordinate is checked for a valid location.
- Easy – fire randomly every turn.
    - Generate the row and column coordinate at random. Check for valid coordinate and then fire.
- Medium – fires randomly until hit, then fires orthogonally in adjacent spaces until it hit/sinks a ship.
    - First, the AI generates a valid random coordinate position to fire at.
    - Once the AI hits a ship, the next turns maze walks around the hit location (up, right, down, left) using recursion. Continue until hit. Repeat until the ship is sunk.
    - After ship is sunk, continue to fire randomly.
- Hard – the AI knows where all the ships are and lands a hit on every turn.
    - The AI has access to the other players ship placement grid.
    - Cycle through each location until all ships are sunk.

Table 4. MainGame.js

Function	Description
selectMode()	Calls the appropriate AI difficulty function.
randomBool()	Helper function for ship placement. Returns a bool true or false at 50% probability.
autoShip(length)	Generates a new valid ship and places it. Helper function for ship placement.
easyAI()	Checks that random cell is valid before returning the cell.
generateCell()	Generates a random cell coordinate.
randomInt(min = 1, max = 10)	Generates a random number between min and max.
mediumAI()	Call easyAI() if the AI has not hit a ship or if the ship was sunk on the previous turn. Otherwise, update tracking variables and guess4d(). Returns the cell coordinate from guess4d().
guess4d()	Finds the next orthogonal position available to hit a ship. If the boundary is exceeded, reset the orientation, and offset. Call recursively until a valid cell is generated.
changeDirection()	Changes orient to up → right → down → left → up. New orientation depends on previous orientation.
getNextChar(col, offset)	Helper function that is used to get the next column.
getPrevChar(col, offset)	Helper function that is used to get the previous column.

charToInt(char)	Convert character number to an integer.
hardAI()	Iterate over player 1's ship array and hit each ship.
Variable	Description
orient	Tracks the up/right/down/left orientation for the medium AI maze walking.
offset	Used by the medium AI for cell guessing.
oldP2Hits	Tracks hits.
newAIShip	When true, the orientation and offset are reset in medium AI.