**Battleship:**

The project consists of 5 files:

* 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.
* Battleship.css: CSS file to make the buttons and grid pieces look pretty.
* 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.
* 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.
* 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.

Battleship.html:

* Battleship.html implements the web template for the game.
* It has separate "windows" that are loaded on the page along with corresponding javascript functions, the windows are id'd by the following, "shipNumPick", "gameOver", "gameOver", "transition", "p2View", "newGame".
* The main function that is called is the createUI(numberOfShips) function which is a javascript function in the GraphicsUI.js. The way this works is first the web page is loaded with 5 buttons, labeled 1-5, which then pass the number value of that button to the createUI function.
* Depending on the current status of javascript files the html hides and shows the windows with the information that needs to be included from the javascript files. (ex, number and location of ships on board, hit and miss of attacks, instructional message, etc.)
* The script calls the javascript files (GraphicsUI.js, ShipPlacement.js, and MainGame.js)

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.

GraphicsUI.js:

* GraphicsUI.js builds the visual user interface and handles everything the players directly interact with.
* The main function is createUI() which takes in an integer representing the number of ships being used as a parameter and adds the ships and boards to each player’s window.
  + 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.
* Whenever a tile is clicked, the function parseTileClick() is called and will call an appropriate function from MainGame.js based on the game state.
  + parseTileClick() takes in a string representing the tile ID as a parameter which should be in the format described above.
* The function moveShip() takes a ship and moves it to the location of a tile of a board when that board is visible.
  + It takes in three parameters:
    - a string representing the ID of the ship div element which should be in the format [player]-[ship type], i.e. “p1-1TileShip”, “p2-5TileShip”, etc
    - a string representing the ID of the tile, which should be in the format described above
    - a boolean representing if the ship is horizontal or vertical (true if vertical, false if horizontal)
* The function setTileState() 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
    - 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)
* The function switchWindow() hides all of the windows except the one specified in the parameter.
  + The parameter is a string representing the ID of the window div element and should be one of the following: “p1View”, “p2View”, “gameOver”, “shipNumPick”, “transition”, and “newGame”.
* updateTransitionText(text), this function 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), this function 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(), this function is a helper function that calls the switchWindow() function passing in the current transitionTarget as the parameter.
* setGameOverText(), this function receives a string parameter “text” and sets the text to the GameOver window. It gets called when certain win conditions are met.
* setInstruction(text, player), 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.
* Global variables: p1Ships, p2Ships, gameState, currentWindow, transitionTarget

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.
* initializeShipPlacement(\_numShips) is 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,
* hoverCell(cell) is 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) 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.
* Several functions in ShipPlacement.js make use of the moveShip and setShipProperties functions in GraphicsUI.js to render previews and final placements of ships.
* Variables:
  + numShips is the total number of ships per player. Is set in initializeShipPlacement and remains the same for the rest of the game.
  + shipsRemaining is the 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 is a 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 is a pointer to either p1Ships or p2Ships in GraphicsUI.js, depending on who is placing ships.

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, MainGame, the function 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, which displays text that instructs the player on what to do
* Player Turns
  + The turn of each players 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
  + Each time a successful is made in guessCell(), the function updateHitCounter() updates a player’s total number of hits
  + When a player’s total number of hits reaches the maximum number of hits, the game is over
  + endGame() calls setGameOverText(text) and switchWindow(text) in GraphicsUI, which displays game over messages to the user
* Miscellaneous
  + Several functions in MainGame.js are not actively used but useful for testing
  + printCoordinateArray() can be used to print all the coordinates a player’s array of ships occupies