Battleship – Online Group

Anthony Bliss

Hannes Ziegler

Leoncio Hermosillo

Marco Reyes

Matthew Rodriguez

Michael Hawara

Stephanie Peacock

Steven Contreres

Description

Our group, the Online Battleships group, implemented the Battleship game as a PHP-driven web application, hosted on the XAMPP server.

Rules

To play battleship, the rules are as follows:

- An account must be created with a valid email address and password. The account holds wins, losses, and potential games in progress for the user's future reference.
- A new game may be started at any time, or a previous game can be loaded and continued.
- Both players, in this case this A.I., are given 5, 4, two 3's, and 2 block ships (5 total). Players can place their ships horizontally and vertically, not diagonally, and must be within the board's span.
- Each player takes a turn selecting a spot on the board to "attack" and whether the opponent has a ship there determines whether the attack was a hit or a miss. A player wins by sinking all their opponents' ships before getting them sunk.
- A common strategy of this game is when an opponent's ship is hit, the player will check in a '+ 'formation to see which way the ship is pointing and sink it.
- There exists another strategy that's based on the size of the ships that haven't been sunk yet which is to hit on potential areas that the ship could be through deduction and reasoning. Ships will generally not be placed on the outskirts of the board, so it's a good strategy to attack on the diagonals of the board and then space 2-3 blocks from each previous attack for the preceding attack. This increases the chances of a player hitting another nearby ship, typically one of higher size due to the span of each attack.

GitHub

https://github.com/hansintheair/BattleshipWeb

Team responsibilities & Major Contributions

Anthony Bliss

(original git project Player-Gameplay)

- Implemented "Placement Phase" for ship placement.
- Implemented first iteration of player gameplay and "easy" AI.
- DB troubleshooting and some fixes

Steven Contreres

(original git projects Account, Logins, Accounts_Logins_Player

- Update User Login
  - Add registering a new account
- Created User Meenu
  - Added update/delete Account
  - Link to Game Menu
  - Added Viewing Account Info
- Created Admin Menu
  - Separating admin from user
  - Added Update Account
  - Added View User, Delete User, Modify User, Logging Out
- Documentation, testing

Leoncio Hermosillo –

(original git projects aiGameplay – dumb AI and less dumb AI)

- Documentation, testing, visual CSS changes.

Stephanie Peacock – User/Menus/Consolidation

(original git projects Account, Logins, Accounts_Logins_Player, Battleship folders)

-

Marco Reyes – AI Gameplay

(original git projects aiGameplay/AI ShipPlacement)

- Documentation and Testing

Matthew Rodriguez

(original git projects aiGameplay – smart AI)

- Update/upgrade AI algorithm
  - Added the smart AI and Dumb AI capabilities(playerComp.js)
- Register regex
  - Added regex for the email and password

Michael Hawara

(original git projects are Player-Gameplay and Battleship Gameboard Test Bed folders)

- Documentation and Testing

Hannes Ziegler – Database Mgmt/access/retrieval

(original git projects Account, Admin_User_Player folders)

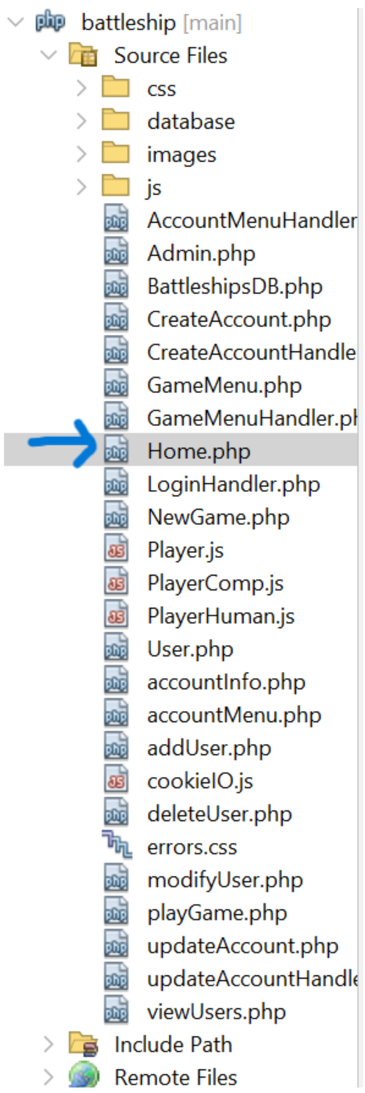
- Project Co-manager
- SQL Database setup/design
- Data storage IO
  - Cookie IO (cookieIO.js)
  - SQL database IO (BattleshipsDB.php)
  - JavaScript general load/save functions controller (Game.js)
  - Various endpoints for getting/setting data on the SQL backend (calls through BattleshipsDB)
    - SaveGameHandler.php
    - LoadGameHanlder.php

        ▪ CreateAccountHandler.php
        ▪ LoginHandler.php
- Initial (very rough) setup for Player.js to allow save/load of state.
- "Create an account" and "Login" home pages.
- GitHub repository setup and management.
- Documentation for versions and concepts used, as well as several items in the final project structure section (those related to database/cookie IO, homepage login and create user, and load/save game state).

Documentation

This project is located on GitHub at [GitHub - hansintheair/BattleshipWeb]()

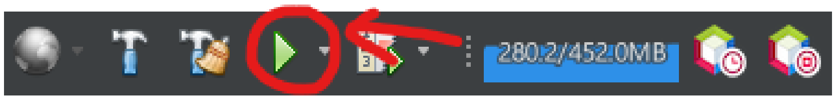
Getting Started



```
battleship [main]
  Source Files
    css
    database
    images
    js
    AccountMenuHandler
    Admin.php
    BattleshipsDB.php
    CreateAccount.php
    CreateAccountHandle
    GameMenu.php
    GameMenuHandler.pl
    Home.php
    LoginHandler.php
    NewGame.php
    Player.js
    PlayerComp.js
    PlayerHuman.js
    User.php
    accountInfo.php
    accountMenu.php
    addUser.php
    cookieIO.js
    deleteUser.php
    errors.css
    modifyUser.php
    playGame.php
    updateAccount.php
    updateAccountHandle
    viewUsers.php
  Include Path
  Remote Files
```

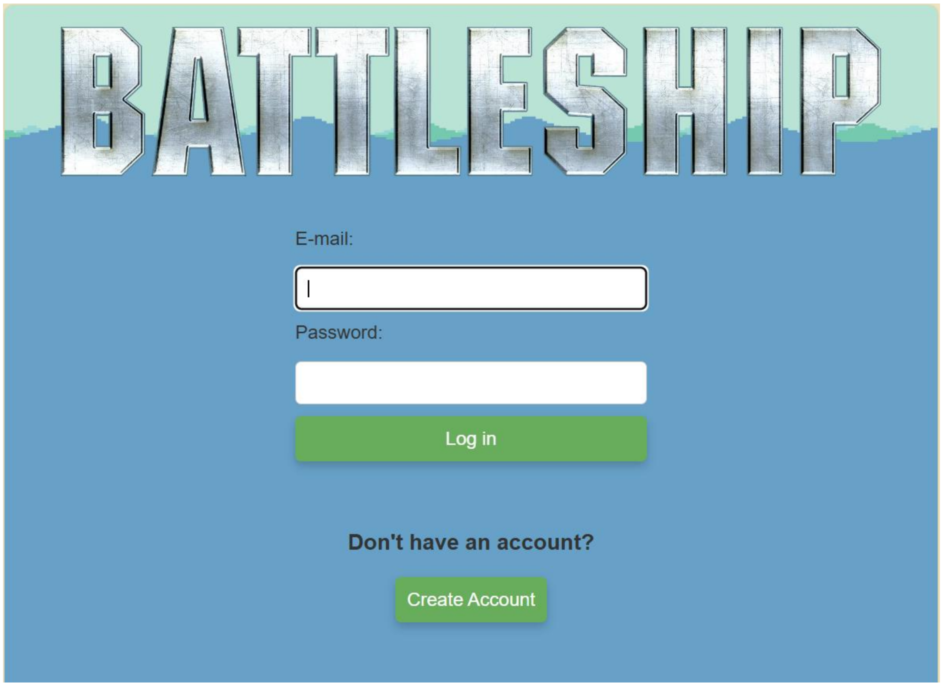
The project repository readme document contains detailed instructions for initial setup to configure the project within your environment.

After setting everything up, you can open the project in NetBeans and navigate into the source files. As seen to the left, click on *Home.php* to see the main driving code.



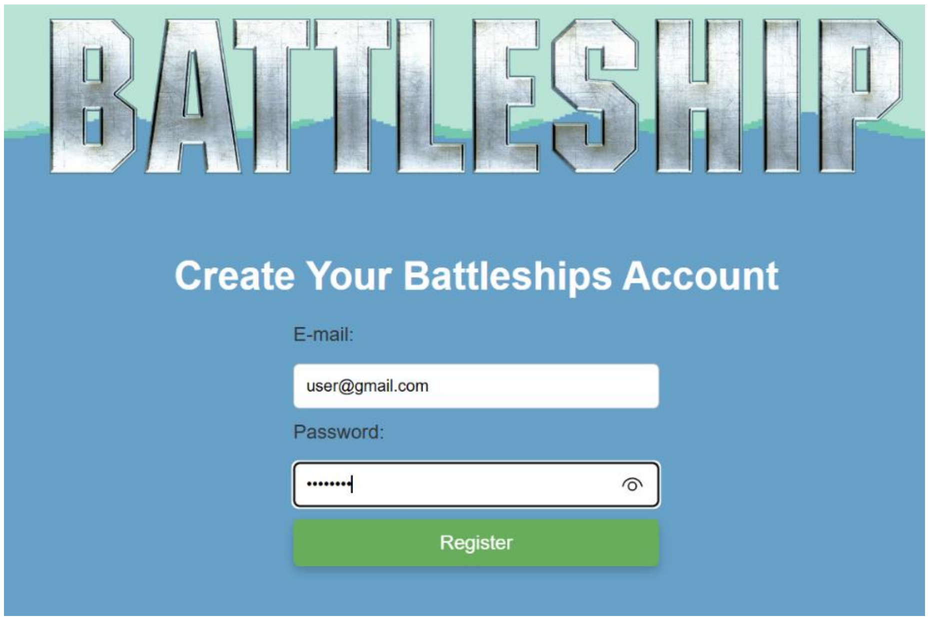
Click on the green run button to run the program!

Examples:



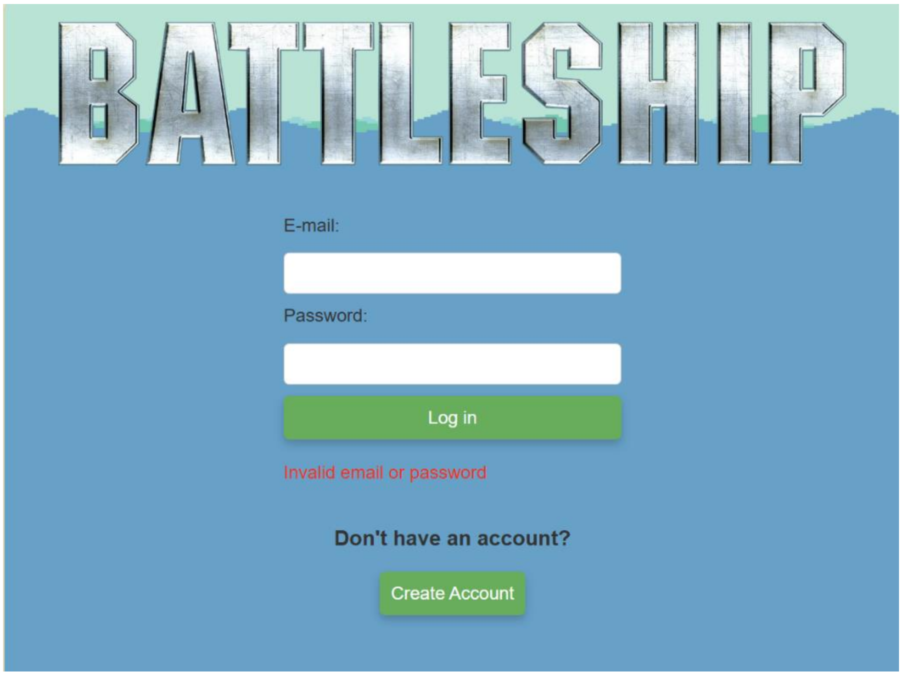
Upon running the code, the user will be presented with the choice to login with their email and password, or alternatively, they can choose to create an account.

Create a user account:



Upon clicking the “Create an Account” button, the user is prompted to enter an e-mail and secure password.

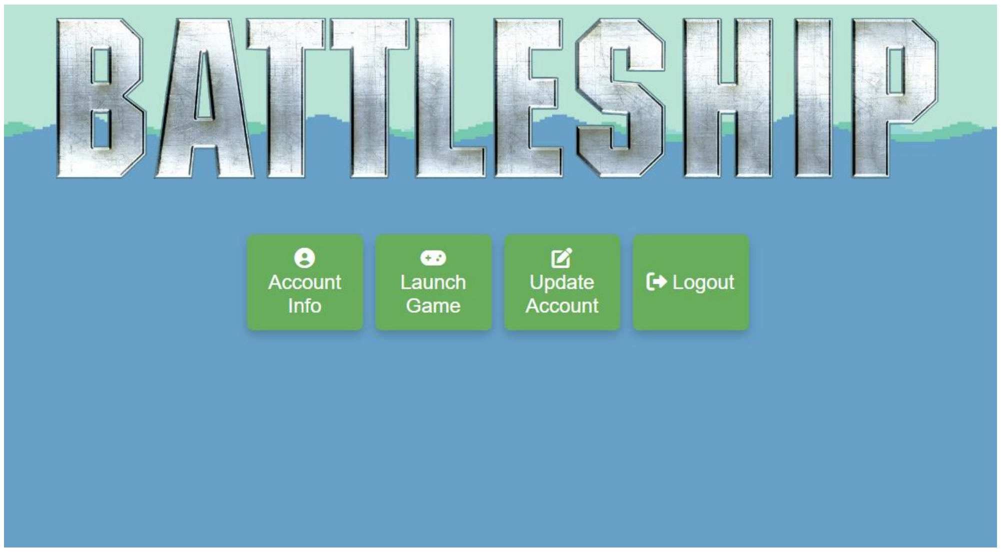
Login – Incorrect Password:



Invalid email or password

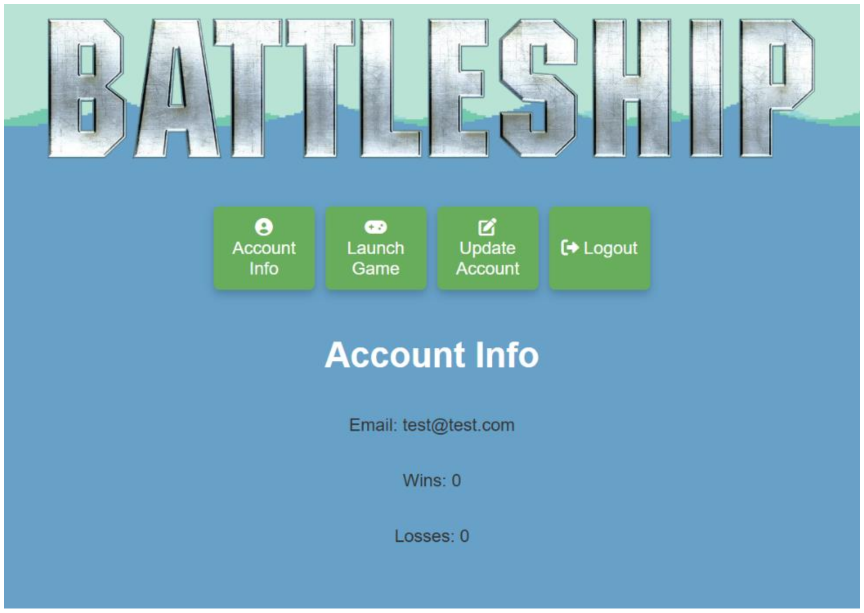
As shown in the image above, when a user enters an incorrect password, or email, they are denied access to their account, if it exists.

Login – Valid Email and Password:



Upon successfully logging in, the user is greeted with four separate menus.

View User Account:



Account Info

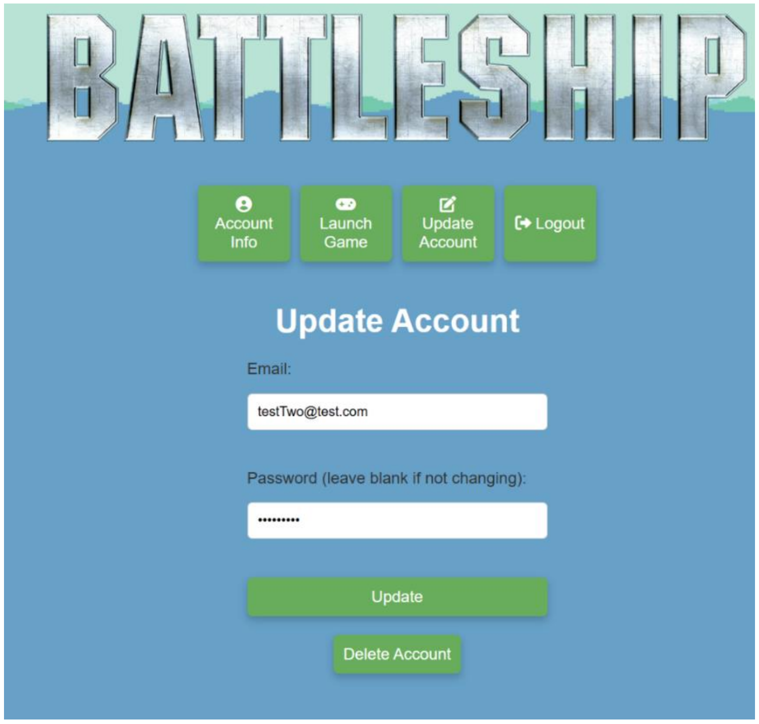
Email: test@test.com

Wins: 0

Losses: 0

When the "Account Info" button is clicked, the user's email, wins, and losses are displayed before them.

Edit User Account:



Update Account

Email:

testTwo@test.com

Password (leave blank if not changing):

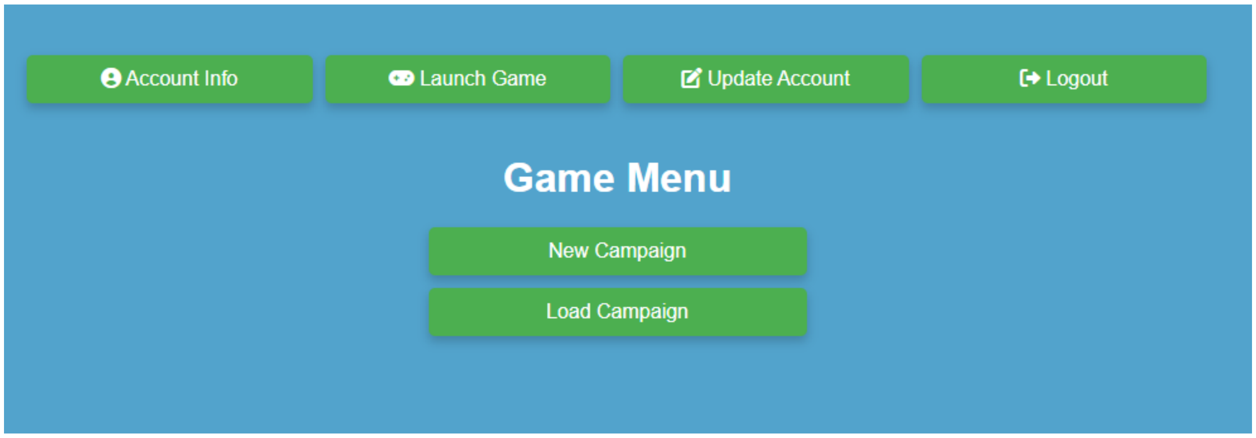
•••••••••

Update

Delete Account

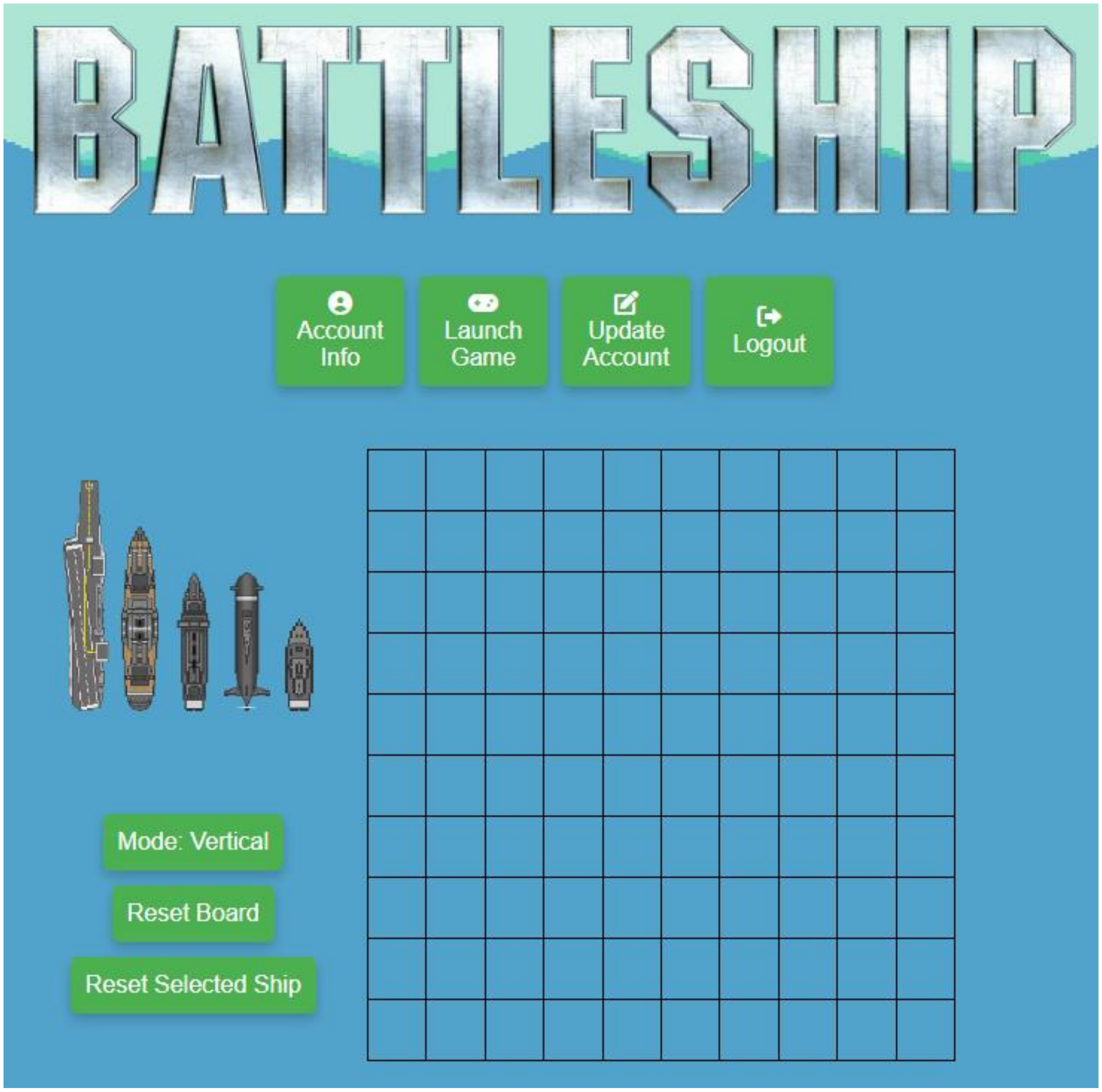
The picture to the left depicts a user updating their account. First, they update their email from *test@gmail.com* to *testTwo@gmail.com*. Additionally, they choose to change their password as well.

Play a game:



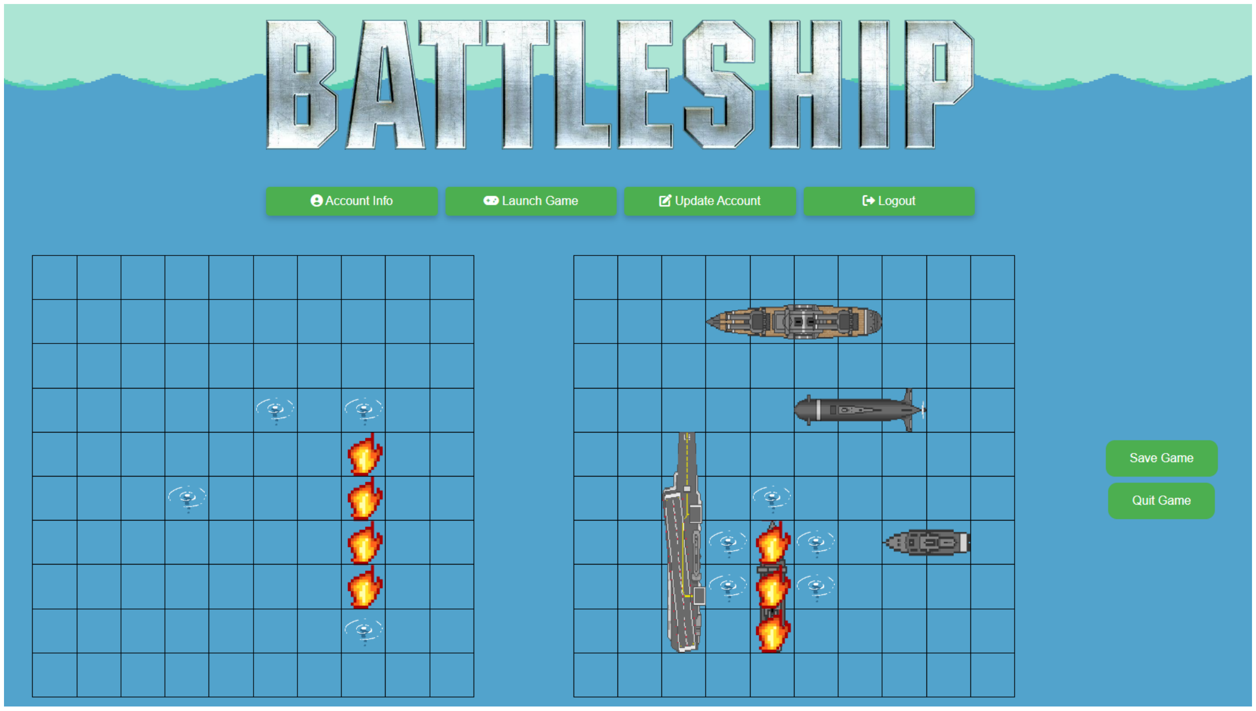
When the "Launch Game" button is pressed, there are two options given in the Game Menu. The options are new campaign, and load campaign. A user can only ever own a single game.

New Campaign:



When “New Campaign” is selected, the user is prompted to select the layout of their 5 different ships and can put them vertically or horizontally. They can also reset their entire board or undo the current ship they’ve placed.

Playing the game:

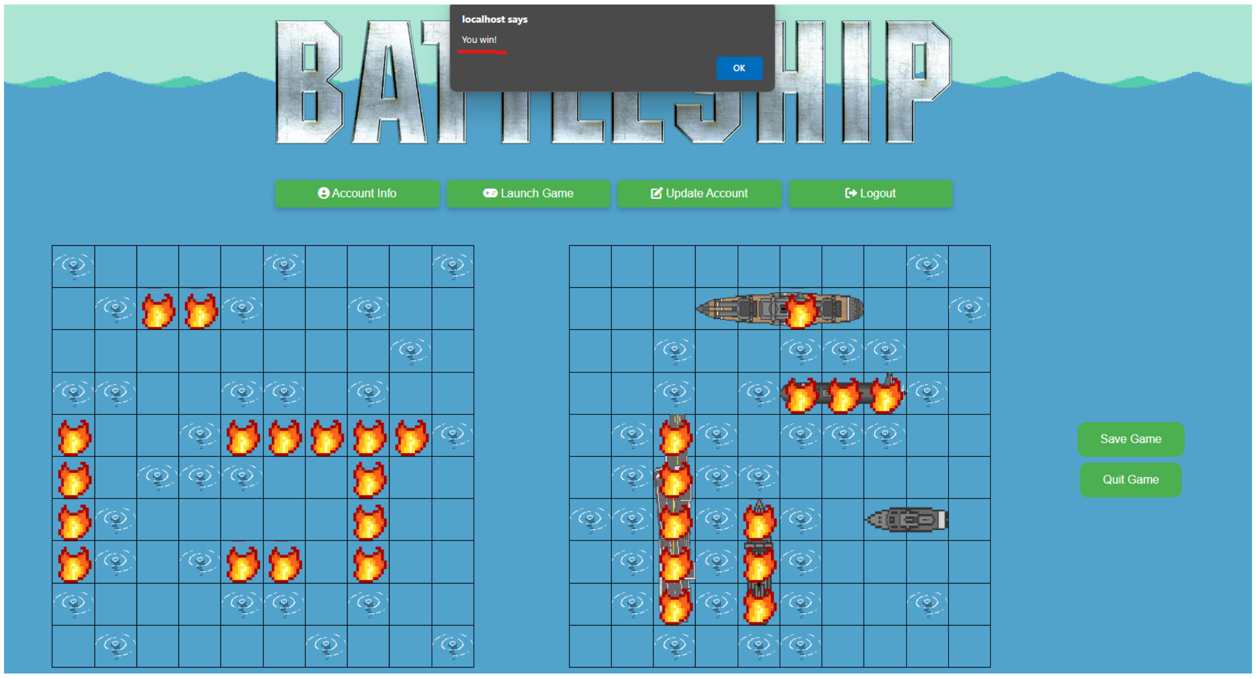


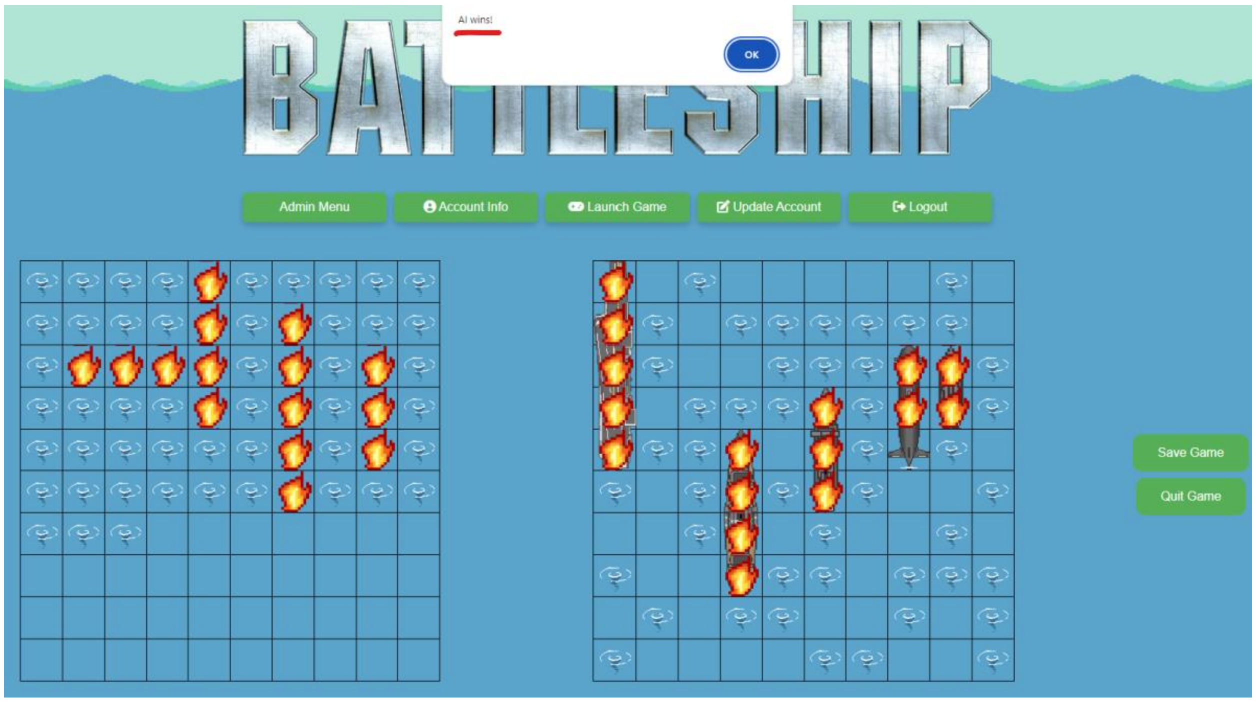
To play the game, you click on the cell you want to attack and the A.I. will choose a cell on your board to attack. The left board contains the user's shots while the board on the right contains the A.I's shots. Once a ship is hit, a fire '.gif' will play. If it's a miss, a ripple effect '.gif' will play.

Game state is tracked by a cookie. You can permanently save the game to your account at any time by clicking Save Game. This stores the game's state in an SQL table with a record associated to your account. This will overwrite the previous save game. If you have never saved a game, it will create a new record for your account in the SQL table.

You can quit the game at any time by clicking Quit Game or navigating to another page.

Winning the Game

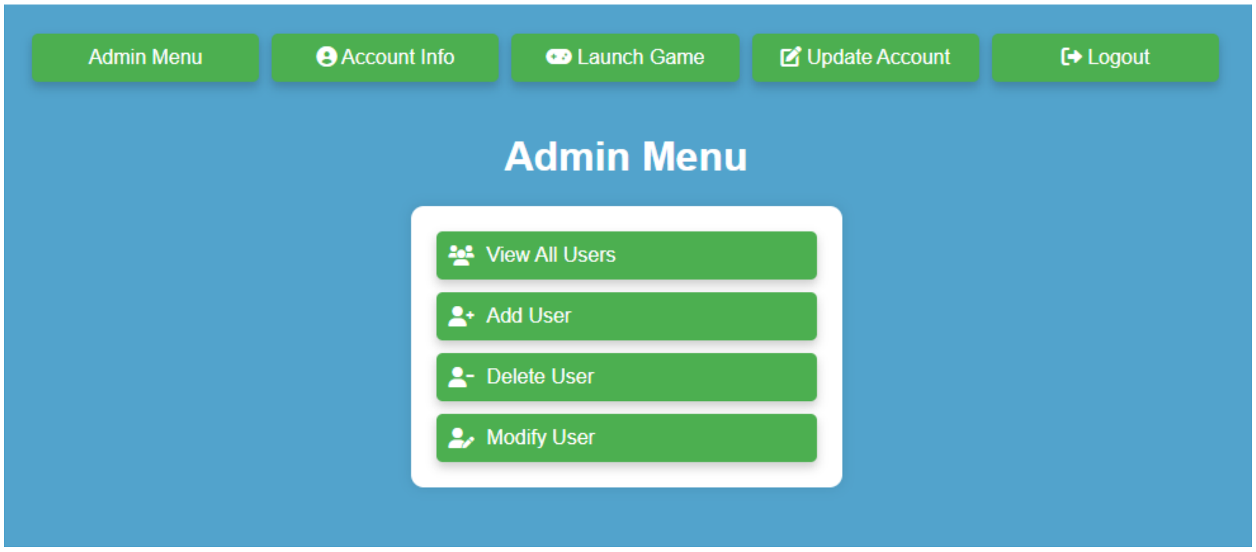




After a game has been won, a prompt is displayed saying who won the game and if the win is in the user's favor, it is recorded in their account.

Admin menu:

If you are logging in as an Administrator, you will see an additional Admin Menu to the left of Account Info.



From here you can view all users, as well add, delete, or modify users. You will not be able to make changes to your own account from this menu.

Final project Structure

- AccountMenuHandler.php
  - This controller receives requests from users allowing the user to update, change or deletion of account with UpdateAcount.PHP handling it accordingly.
- Admin.php
  - This model/view, after admin status is checked for valid admin, displaying the menu option for the administrator allowing the admin to modify users' info either updating account information, deleting users, viewing all users or launching a game.
- BattlshipsDB.php
  - This backend database IO model allows for various interactions with the SQL database. It is a wrapper for mysqli and instantiating the class is all that is needed to connect to the database, the necessary setup and information is privately held in the class. To use, you first instantiate an instance, then use connect to start an IO session. There are functions for interacting with user account data and saved game data. Once processing is completed, call disconnect to end the session.
- CreateAccount.php
  - This view (web page) contains UI for creating a new account and calls the CreateAccountHandler endpoint to create a new account in the SQL

    - database. Input validation occurs in the view, and again in the backend before writing to the database.

- CreateAccountHandler.php
    - This controller (endpoint) receives a request to create a new account from CreateAccount.php and handles it accordingly. First, it validates the inputs. If valid, it will write the new account using the BattleshipsDB class. If the inputs are invalid, a session token is used to communicate this to the CreateAccount view and it returns to the CreateAccount view. The password is hashed for secure storage.
- GameMenu.php
    - The game menu view provides options to start a new campaign or load an existing campaign. When loading an existing game, the saved game pertaining to the user id is first loaded from the database by making a request to LoadGameHandler through an event handler bound to onclick. The data is then saved into a cookie and the view is redirected to playGame.php which reads the game state from the cookie.
- GameMenuHandler.php
    - This endpoint redirects to the relevant views depending on which button was clicked in the GameMenu view.
- Home.php
    - This view (web page) contains UI for logging in. It is the Home page or landing page of our Battleship game website. It calls the LoginHandler endpoint to validate the login credentials and redirect to the User menu.
- LoadGameHandler.php
    - This endpoint retrieves game state data for a particular game (by user id as only one game is associated with each account and is stored in the database table with an id matching the user id of that account) from the SQL database and sends it back to the requestor.
- LoginHandler.php
    - This controller (endpoint) receives a request to log in from Home.php and handles it accordingly. First, it validates the username and password match a record in the database. The password stored in the SQL database is decrypted to check the match. If the credentials match an account, the session token is set, and we redirect to the User.php. Otherwise, an error session token is set, and we redirect back to Home.php, where the error token is used to relay the login failure message to the user.
- NewGame.php

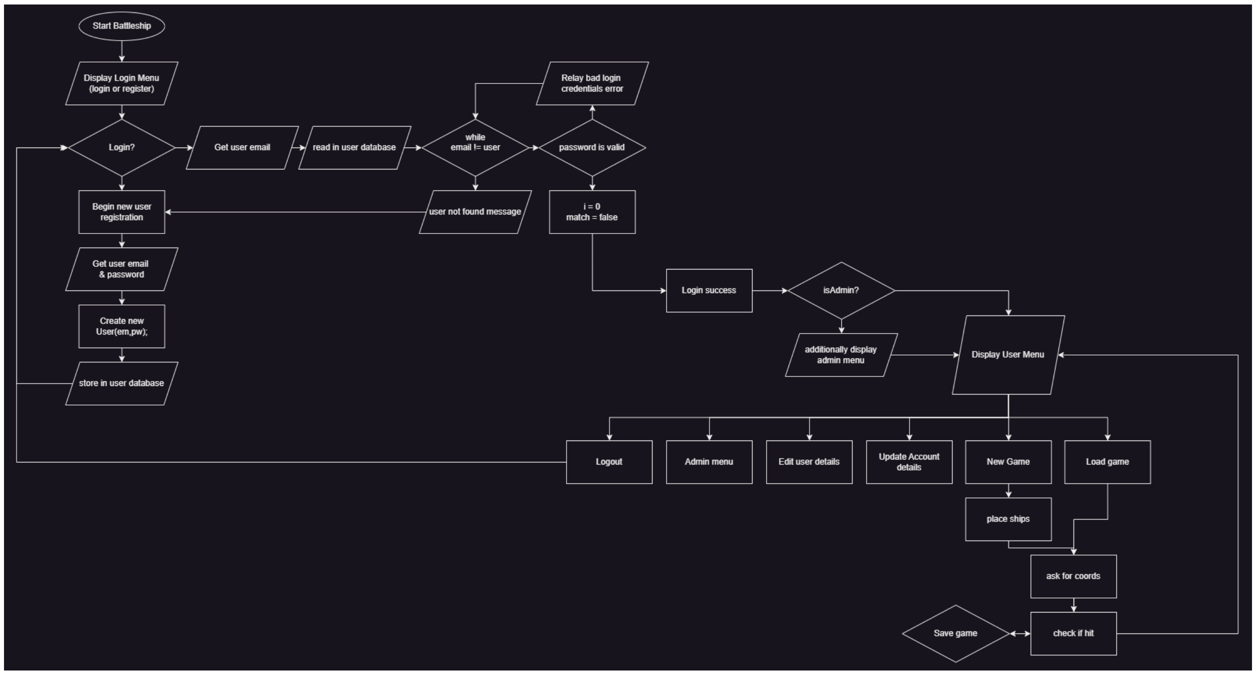
  - NewGame displays the “placement phase” of the game. It allows the player to select a ship with their mouse and click a cell to place the ship on the board. The Mode button allows them to swap between horizontal and vertical placement. The player can choose to reset the selected ship or reset the entire board if they want to move their ships. A confirm button displays after all five ships are placed, allowing them to proceed to the gameplay against AI.
- Player.js
  - Player.js contains all the data structures needed for ships, boards, shots, locations, and health. It has several functions that dynamically generate HTML tables based on the contents of the arrays. Player.js has all the functions necessary for the “placement phase”, such as validation, bounds checking, button functionality, and the logic to select the correct cell to place a ship based on the users’ click. Player.js also contains the functions to save and load the arrays from JSON. Lastly, it has functions to check win conditions and handle the end game state.
- PlayerComp.js
  - PlayerComp extends the player class, which represents a computer-controlled player in a game. It manages the ship placements and shooting logic depending on the mode that is in (Dumb mode or Smart mode). Smart mode uses a checkerboard pattern while the dumb mode shoots at random cells. The AI tracks hits and employs a search strategy to locate and sink ships. The class also handles the game state.
- PlayerHuman.js
  - PlayerHuman extends the player class. PlayerHuman has the logic for the way the player shoots at the enemy board. A player clicks on a cell on the enemy table to fire. The “hits” and “misses” are tracked to progress the game. Hits will decrement the health of the ship that was hit. This class also assists with ending the game, by alerting the player if all enemy ships were sunk and “stopping” the game.
- Game.js
  - This controller handles common game loop interactions, mainly loading and saving game state to and from cookies and the SQL database.
- User.php
  - This model/view, after admin status is checked for valid admin, displaying the menu option for all non-administrator allowing the users to modify only their own accounts, either updating email/password, or deleting account,

viewing wins/losses record and lunching a game. AcountMenu.php, updateAccountPHP, playGamePhP handling logic.

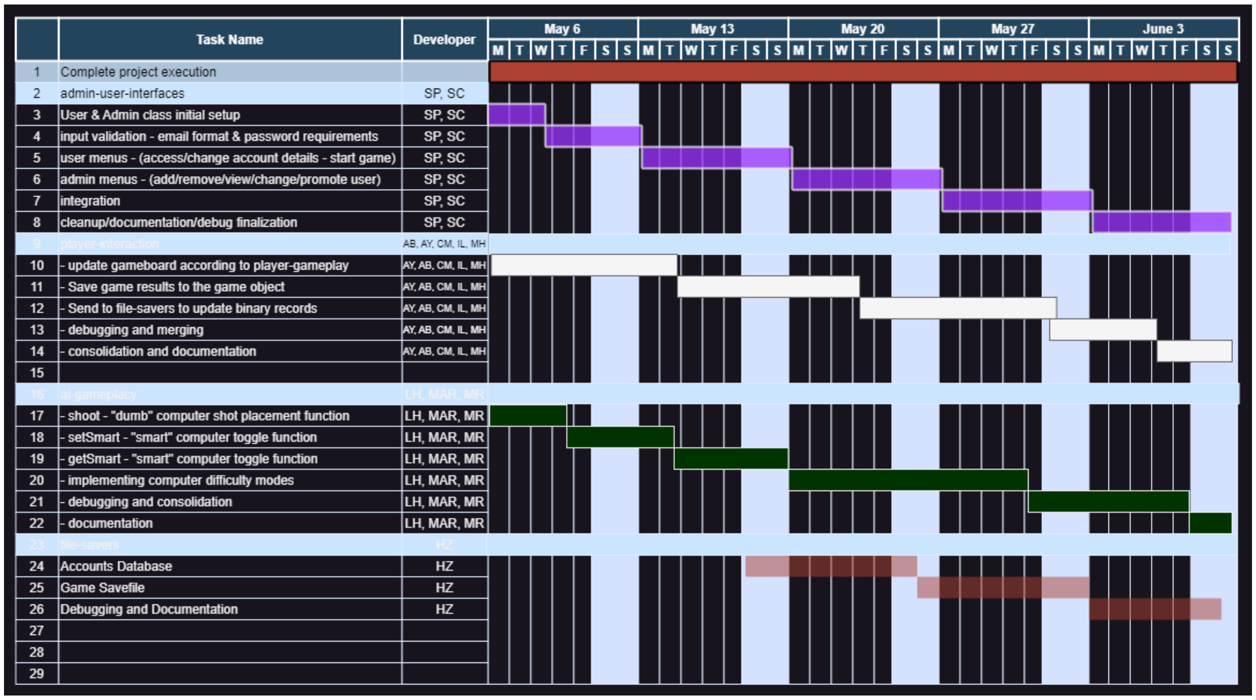
- accountInfo.php
  - This controller queries the database by searching form email address of user that is currently logging in and displaying email address and wins/loss records
- accountMenu.php
  - This model/view prompts user for option to change email address, update password or delete account with accountMenuHandler handling the interacting with the database.
- addUser.php
  - In here the admin is giving option to add a new user with the same logic when a new user registers account. It checks that email address is in correct format and then queries database for existing user, informing admin if there is an account with matching email. Along with that, the password is checking if meets requirements and if both are meeting the new email and password are added to account with 0 wins/loss, not set as an admin and email hashed for security.
- cookieIO.js
  - This IO model handles interactions with cookies. It allows writing to cookie (or create if not already existing), reading from a cookie, and deleting a cookie (by marking it expired).
- deleteUser.php
  - This controller interacts with databases for checking if the user supplied email address is in database and if user exists deleting user from the database.
- modifyUser.php
  - This Model/view prompts the admin if they want to update a specific user's email address, password, admin status, wins or losses. Based on what option is chosen it queries the database with BattleshipBD handles the updating of user accounts with correct function.
- playGame.php
  - PlayGame handles the display of the gameplay. It loads ship placements from JSON/cookies, creates the boards, and starts the game loop. The game loop in playGame controls how the player and AI takes turns. PlayGame also has save game and quit game buttons.
- updateAccount.php

    - This view prompts users to either update email address, password or delete account, while displaying successful updates or deletions.
- updateAcountHandler.php
    - This controller handles the updating and deleting of accounts, checking the user email meet specified requirements along with checking passwords meets requirements and informing user if criteria is meet before update database.
    - When a user deletes the account, they are prompted to enter the correct password for verification for deleting and the database is queried for matching password and account removed from base after confirmation.
- viewUsers.php
    - This controller queries the database from all registered users and displays it onscreen for view.
- battleships.sql
    - The database schema

Flowchart Diagram



Project Timeline (GANTT)



	Task Name	Developer
1	Complete project execution	
2	admin-user-interfaces	SP, SC
3	User & Admin class initial setup	SP, SC
4	input validation - email format & password requirements	SP, SC
5	user menus - (access/change account details - start game)	SP, SC
6	admin menus - (add/remove/view/change/promote user)	SP, SC
7	integration	SP, SC
8	cleanup/documentation/debug finalization	SP, SC
9	player-interaction	AB, AY, CM, IL, MH
10	- update gameboard according to player-gameplay	AY, AB, CM, IL, MH
11	- Save game results to the game object	AY, AB, CM, IL, MH
12	- Send to file-savers to update binary records	AY, AB, CM, IL, MH
13	- debugging and merging	AY, AB, CM, IL, MH
14	- consolidation and documentation	AY, AB, CM, IL, MH
15		
16	ai-gameplacy	LH, MAR, MR
17	- shoot - "dumb" computer shot placement function	LH, MAR, MR
18	- setSmart - "smart" computer toggle function	LH, MAR, MR
19	- getSmart - "smart" computer toggle function	LH, MAR, MR
20	- implementing computer difficulty modes	LH, MAR, MR
21	- debugging and consolidation	LH, MAR, MR
22	- documentation	LH, MAR, MR
23	file-savers	HZ
24	Accounts Database	HZ
25	Game Savefile	HZ
26	Debugging and Documentation	HZ
27		
28		
29		

Versions

Below outlines the various versions of the project.

0.1 Groundwork – database setup, accounts, and home page

hansintheair/BattleshipWeb at battleship/0.1 (github.com)

This version is all groundwork. It sets up the GitHub repository and our initial database model. It also contains the home (login) and creates account pages and basic functionality without any styling. It does account creation and login validation but stops short of defining UI on successful login.

0.2 Menus

hansintheair/BattleshipWeb at battleship/0.2 (github.com)

This version fleshes out the user and admin experiences. The home and create account pages are now styled, and there are now user and admin menus which are mostly styled. All menus are fully functional except for anything related to gameplay.

0.3 Basic Gameplay

hansintheair/BattleshipWeb at battleship/0.3 (github.com)

This version introduces basic gameplay functionality and storage of game state in cookies. You can set up the board (place ships) and take shots. The game is not yet fully playable as AI is still missing.

1.0 First Release

hansintheair/BattleshipWeb at battleship/1.0 (github.com)

This version represents a fully developed website for the battleship game. A game of battleship is now fully playable with AI implemented. The game can be permanently saved under your account and loaded up again later. There are still a few bugs, but the site is fully operational, and the remaining bugs could be resolved if development continued. We would probably need another week of polish to make it truly shippable (pun intended).

Main

hansintheair/BattleshipWeb (github.com)

Any additional work (fixes/features/etc.) between noon June 9th and midnight June 9th are in this branch (main). Main is the development branch, and it may or may not be unstable. It represents our most recent up-to-date version of the project.

Concepts used

We used the following concepts in our composition of the online-battleship game.

MVC

The stack we used to create this project naturally enforces the MVC paradigm. Views are implemented with HTML and CSS. Controllers are implemented with JavaScript (either in separate files or often within the page) and PHP endpoints. Models are implemented with a PHP class backed by an SQL database. The model is leveraged by the PHP endpoints.

Object Oriented

The PHP model for SQL database interactions is implemented as a class that wraps myslqi. The class is designed to self-contain all the details required to connect to the battelship SQL database, and the all the client code must do is instantiate the class and then start a session with by calling connect. The client code can then leverage any of the implemented SQL IO operations and when done, call disconnect to end the session.

The gameplay is implemented using JavaScript classes. The Player class serves as the base class and contains methods and data common to both subclasses, which are

PlayerHuman and PlayerComp. PlayerHuman and PlayerComp inherit (extend) from Player and implement additional methods and data required to make gameplay work for these player types.

Inheritance

The PlayerHuman and PlayerComp (AI) classes are implemented with inheritance. The PlayerComp class inherits common members from Player and implements methods specific to AI gameplay. Likewise, the PlayerHuman class inherits common memebers from Player and implements methods specific to human gameplay.

SQL database

The SQL database schema was created using OpenOffice. The schema is simplistic as there is not much complexity in the data storage requirements. There are two tables, entity_users, and entity_games. Because we decided that a user can only maintain a single saved game, there is a one-to-one relationship between each user and each game record. For this reason, each game record is associated with each user by reusing the user's unique id. This avoids requiring an xref table to link games to a user. We still implemented them as separate tables in case we had time to allow multiple save games later, however, we did not manage to find the time, so this simple database structure is all that is needed to support the Battleships website and game.

The game state is stored in JSON string format in two varchar fields (p1 for Player 1, and p2 for Player 2) with a character limit of 3000 characters. 3000 characters is probably double what is needed in each field, but we left extra space if we found we needed to store more data later during development.

Cookies

Cookies are used as an in-between storage for game state. This is to enable easily passing game state between pages. The actual game state is stored in the format of a JSON. The JSON stores the contents of each player object relevant to the game's state. Cookies are only kept for the use session and destroyed (set to current time -3600) upon logout.

Sessions

Session tokens are used to pass messages from PHP endpoints back to the HTML view. These are generally messages relaying whether the endpoint request succeeded or failed (and if it failed, why).

Securing pages

User account data/pages are secured behind a user account login which requires a matching username and password. The user's password is securely stored as a hashed string in the server-side SQL database. Upon a login request to a server-side PHP endpoint, we verify that the username and password match those on file before we initialize a login.

We use Session tokens to enforce login security. When a user logs in, the user email and id are stored in a session token. If the token is not set to a valid user, anyone trying to access a page that requires being logged in will be redirected to the login home page. The session is destroyed upon logout, ensuring that another person using the PC does not access the account by entering a URL for the user menu.

Serialization/Deserialization of game state to JSON

The Player object defines toJSON and fromJSON methods responsible for serializing and deserializing the player state to and from a JSON.

Form Validation and Regular Expressions

Form validation is performed through built-in input types, client and server-side regular expressions, and server-side data verification.

The first layer of defense is using the appropriate input type for input fields in the forms. HTML forms support various input types with useful behaviors for their specific type of input, such as email and password input types. These can be marked required fields with the required attribute, and they also support additional custom validation through specifying a regular expression pattern. We use both to prevent invalid form data from being sent to the server-side PHP endpoint where it will be processed.

On the server, we receive the pre-validated data and validate it again with regular expressions to ensure nothing fell through the cracks. We also perform additional validation against the database if required (for example, in the case of logging in we locate the user account using the submitted e-mail and decrypt the stored password to compare it to the password received from the client-side request.

Any problems with the form inputs are relayed back to the user through session tokens which are then handled on the client after redirecting back to the caller. Only if everything checks out do we perform the requested action on the database. In some cases, we also use a session token to communicate success back to the caller.

User-Admin

The project implements user and admin roles, where users and admins have separate privileges.

- Users may:
  a. Play games
     - Create, load, and save (overwrite existing) game
  b. View and edit their own profile
- Admins may:
  a. Can do anything a user can, and:
  b. View and edit all user accounts

Server-Client

We used the technology stack HTML/CSS/JavaScript/PHP/SQL. The server-client relationship of web pages is well defined for the technology stack we used. Server-side operations are performed in PHP and SQL. Client-side operations are performed in JavaScript and viewed from HTML/CSS pages. Communication between client and server is accomplished through POST requests and by setting session tokens.

Not completed / Known issues / Missing features

The 1.0 version of the Battleship website and game are fully functional, but some bugs and missing features remain incomplete. Below is a list of goals that we did not have the time/resources to complete by the due date.

Bugs:

- When loading a game, shots are not displayed on the board. The shots are accurately recorded and loaded from the saved game in storage, but we are missing logic to display them as this logic was placed in the PlayerHuman and PlayerComp methods that place the shot on a board as they are taken, rather than a renderAllShots type method like we have for renderAllShips. There was an attempt to refactor and get this done, but there were technical difficulties that proved too great for the limited time we had left.
- The AI will not always "finish off" a ship that it has detected. However, it usually does the job, and it can beat a human player as I found out the hard way. This bug may be seen as a "feature" to not make the game too difficult to beat.

Features (mostly polish):

- Communicate who’s turn it currently is
- Communicate when a ship has been sunk