Team Quoll Tic Tac Toe System Documentation

Lucie Mugnier | Amanda Munoz | Yan Olsheveskyy | Z Zialcita

The system is implemented by using html and css to the render the visual aspects then these skeletons are populated using javascript which calls on php scripts which then communicate to the database through four specialized php classes that correspond to the four tables of the tttdatabase. WAMP was used as the main server but MAMP worked reasonably well in conjunction.

[INSTALLATION]

The TTT System is a trio of html pages manipulated by several Javascript, PHP, and CSS scripts. In order to run it:

1. In order the establish the system the user must have WAMP/MAMP/LAMP/XAMP on their computer.
2. Establish the database by copy and pasting the following lines of code into the MySQL terminal

    **As a note if the following lines of code do not work one common problem is that the defaults are not set, you can set them yourself by using ALTER TABLE. Also success has been achieved by handtyping all lines into the terminal.*

```
        CREATE DATABASE tttdatabase;

        USE tttdatabase;

        CREATE TABLE playerinfo (username VARCHAR(20), password VARCHAR
    (30), email VARCHAR (30), onlinestatus VARCHAR(7) DEFAULT “offline”, ingame
    TINYINT(1) DEFAULT 0, PRIMARY KEY (username));

        CREATE TABLE playerstats (username VARCHAR(20), rank INT DEFAULT 0,
    wins INT DEFAULT 0, losses INT DEFAULT 0, ties INT DEFAULT 0, total INT DEFAULT
    0, standing INT DEFAULT 0, FOREIGN KEY (username) REFERENCES
    playerinfo(username));

        CREATE TABLE friendslist (username VARCHAR(20), friend VARCHAR(20),
    FOREIGN KEY (username) REFERENCES playerinfo (username), FOREIGN KEY
    (friend) REFERENCES playerinfo(username));

        CREATE TABLE tttgame (playerX VARCHAR(20), playerO VARCHAR(20),
    moves VARCHAR(9) DEFAULT “---------”, gametype VARCHAR (6), score
    VARCHAR(10) DEFAULT “0,0”, activeplayer VARCHAR(20), notice varchar(7) DEFAULT
    “request”, FOREIGN KEY (playerX) REFERENCES playerinfo(username), FOREIGN
```

KEY (playerO) REFERENCES playerinfo(username), FOREIGN KEY (activeplayer) REFERENCES playerinfo(username));

GRANT ALL ON tttdatabase.* TO 'user'@'localhost';

SET PASSWORD FOR 'user'@'localhost' = PASSWORD('userpassword');

3. Copy provided Quoll_tttcode.zip into //www folder or corresponding folders on systems beside WAMP.
4. You are now set up to access login.html via the localhost.

[GLOSSARY]

current player : whichever of two opposing players of tic-tac-toe from which the code perspective is aligned. For example, if the current player has lost it means that the opponent has won and the current player is executing some form of the youLose() function.

friendslist: One of the four tables in tttdatabase. This table keeps track of who is friended with who. See **[DATA]** section further below for elaboration.

index page: The main page of the site-- including the gameboard, friend list, leaderboard, menu bar, game banners, and user stat banners.

playerinfo: One of the four tables in tttdatabase. This table keeps track of existing users for login purposes. See **[DATA]** section further below for elaboration.

playerstats: One of the four tables in tttdatabase. This table keeps track of a player's current statistics. See **[DATA]** section further below for elaboration.

tttdatabase: A mySQL database split into four tables, all of which are responsible for holding user and game stats.

tttgame: One of the four tables in tttdatabase. This table keeps track of all current games. See **[DATA]** section further below for elaboration.

[SYSTEM DETAIL EXPLANATIONS]

General Site Appearance Issues:

Issue: What mechanism determines the overall appearance of the login page?
Solution: CSS. The corresponding file loginstyle.css works with all div objects on the page.

Issue: What mechanism determines the overall appearance of the registration page?
Solution: CSS. The corresponding file regstyle.css works with all div objects on the page.

Issue: What mechanism determines the overall appearance of the index page?
Solution: CSS. The corresponding file indexstyle.css works with all div objects on the page.

Site Login Feature Issues:

Issue: When logging in, a user must submit an existing username and password. What mechanism checks to see it a set of credentials are registered with the system?
Solution: The tttlogin.php script starts off by making an object called $player using the PHP playerinfo class (represented by playerinfo.php). In a later if statement, $player is asked to call on its (playerinfo.php provided) “validLogin” function. “validLogin” takes the username and password supplied through submission, using mySQL commands to compare them to all entries in the playerinfo table.

Issue: What mechanism prevents a user from logging in if they supply the wrong credentials?
Solution: If the “validLogin” function cannot find an entry containing both the submitted username and password, then it returns false to the tttlogin.php script’s if statement. This makes the script skip into the else branch of the if statement, which will only alert the user of their failed login attempt and reload the login page.

Issue: When a user correctly logs in, their session is set and they are sent to the index page. What mechanism is responsible for this?
Solution: If the “validLogin” function returns true to tttlogin.php, then that means it has found the existing user with the correct password. tttlogin.php will then run the if statement, which establishes the session, sends the user to the index page, establishes who the current user and marks them as online by making $player access its “getPlayer” and “putOnline” functions.

Site Registration Feature Issues:

Issue: In order to sign up, a user must supply a unique username, a password, and an email address. If these requirements are met, they will be registered and allowed to login. What mechanism stores these credentials?
Solution: Within register.html, the form object calls tttreg.php on submission. The tttreg.php script starts off by making an object called $player using the PHP playerinfo class (represented by playerinfo.php). It then asks $player to access its (playerinfo.php provided) “exists” function, along with supplied username. The “exists” function checks to see if there is an entry in the playerinfo table that already contains the username, returning true or false based on the result.

If false, then there is no existing user with that username, thus $player is asked to access its (playerinfo.php provided) “createPlayer” function with the supplied username, password, and email address. The “createPlayer” function uses mySQL commands to insert that info as an entry in the playerinfo table. Then it creates a new object called $stats using the PHP playerstats class (represented by playerstats.php), and has $stats access its “createPlayerStats” method with the supplied username, which uses mySQL commands to insert a new entry into the playerstats table with the username.

Issue: If a user supplies a non-unique username, however, their registration will be denied. What mechanism checks for this?
Solution: If the “exists” function actually returns true in the above scenario, it means that the supplied username already exists in the system. So the first part of the if statement is triggered, the system alerts the user to their failed registration attempt, and re-sends them to the register page to try again.

<u>Site Index Feature Issues:</u>

Issue: What mechanism fills a user’s friend list?
Solution: In generateFriendsList.php the unique friendslist for each username is created by calling the construct from friendslist.php. The friendslist is displayed to the user in sideBar.js.

Issue: What mechanism fills the leaderboard?
Solution: generateLeaderboard.php updates the standings for all players then retrieves the top ten which is then displayed using the generateLeaderboard() function in sideBar.js.

Issue: On both the leaderboard and friend list, online users are shown with their names darker than those of offline users. What mechanism checks for who is online and adjusts name colors as necessary?
Solution: Within the sideBar.js script are the functions “generateFriendsList” and “generateLeaderboard”, both of which contain if statements that call on the isOnline.php script. Based on the information provided by isOnline.php, css script is generated which will then be implemented when the div is created.

Issue: On the leaderboard list, there are little plus symbols next to each name. Clicking on the symbol will add the corresponding user to the friend list. What mechanism adds the new friend?
Solution: Within the sideBar.js script, inside the “generateLeaderboard” function, there’s an embedded on-click JQuery function that calls on the addFriend.php script with the username that was next to the cross symbol clicked. The addFriend.php script is responsible for making sure that the friend a user attempting to add is not themself, and has not already been added, only adding the friend when these cases have been cleared.

Issue: On the friend list, there is also a search bar where one can type in another user’s name and add them to the friend list manually. To make things a little simpler, from two letters onward, the search bar will auto-suggest existing users. What mechanism does the auto-suggesting?
Solution: This is done with a combination of the JQuery autocomplete widget (http://code.jquery.com/ui/1.10.3/jquery-ui.js) and the autocomplete.php script. The former is

implemented in sideBar.js, and triggers the latter by sending it the user's typed query. autocomplete.php then uses mySQL commands to echo back a list of usernames that are similar to the query, which the widget then makes appear as suggestions in the search bar.

Issue: Once a name is chosen in the friend search bar and the enter key is pressed, the selected user is automatically added to the friend list. What mechanism adds the new friend?
Solution: There is a javascript function defined in sideBar.js that overrides the traditional submit function for the form that actually contains the add new friend feature. In the new .submit handler, it adds the friend to the list and then impedes the form from doing a default refresh because it is unnecessary.

Issue: Each name on the friend list also has a small cross symbol next to it. Clicking on this will remove the user in question from the friend list. What mechanism is responsible for removing the friend?
Solution: By clicking the cross the click handler for the cross img div is activate which in turn activates removeFriend.php. The entry in the database corresponding to the current user with the specified friend is found and the friendslist entry is updated.

Issue: How are the top ten players for the leaderboard determined?
Solution: The top ten players are determined by standing. In the code, whenever the getTopTen() function from the playerstats class is called, an updateStandings() is called first thereby assigning new standings and returning the top ten with updated standings.

Issue: What mechanism gets and displays a user's stats?
Solution: For both in-game and not-in-game states of the index page, the statpage.js script is responsible for handling the user's stats. Within the script is a function that calls on the returnPlayerStats.php and supplies the user's username. The returnPlayerStats.php takes the username and returns all stats stored in the database (wins, losses, ties, total, rank, and standing) in the form of an array which the javascript then uses to populate the player stats area.

Issue: What mechanism displays the extended version of a user's stats?
Solution: On clicking the arrow located within the user's stat banner, statpage.js steps in with a series of jquery commands working to hide currently displayed starts, banners and the gameboard, and replacing them by displaying an expanded form of the user banner and more detailed version of the user's statistics. Clicking on the arrow a second time will switch the hide and display commands around in order to set all divs and displayed info back to their previous states.

Issue: The logout button is located in the upper right corner of the index page. When clicking this, the user's session is destroyed and they are determined to be no longer online. What mechanism handles this?
Solution: When the logout button is clicked, the tttlogout.php script gets called. The objects $player (using the playerinfo.php PHP class) and $game (using the ttgame.php PHP class) are

made. The script has $player run its “getPlayer” method in order for it to keep track of the current user, and subsequently has it run its “putOffline” method which uses a mySQL command to update the user’s row in the playerinfo table, marking them as offline. The script has $game to run its “getGame” method to keep track of any game that may or may not be happening currently-- if there is a game happening, $game has to run “updateNotice” in order to let the game know that one of its players is quitting early, otherwise nothing has to happen.

<u>Gameplay Issues:</u>

Issue: What mechanism renders the gameboard?
Solution: The gameboard is actually a collection of div objects set up in the index.html script-- so when the index page is loaded, a blank gameboard is already set up. Each of the nine squares is its own div object, identified by two classes, the first is a plain box class and the second defines which box it is (b1 - b9).

Issue: What mechanism draws the moves made?
Solution: tttboard.js is responsible for drawing a move that was made.When the user clicks on one of the boxes(in index.html it corresponds to div classes) the attribute is set from false to true and based on who is active player either O or X is drawn and the database is updated.

Issue: When a user starts “Vs. Random” Mode, the system randomly pairs that user with another user who is both online and not already in a game. What mechanism selects the opposing player?
Solution: The opposing player is selected based on a combination of a playerinfo class function and another php script. The playerinfo class returns a list of all players who are online and not in a game. Another php script takes this list and randomly selects a player, making sure that it is not the current user.

Issue: What mechanism decides who is X and who is O?
Solution: When any game is started, which in turns calls on the startGame.php script. Inside the script is a function that makes use of a random number generator. Depending on whether the generator returns 0 or 1, the $game object calls on the (tttgame.php supplied) “beginNewGame” function with a variation in the parameter order. Whichever of the two players’ usernames comes first becomes X, while the second becomes O.

Issue: What mechanism determines whose turn is it?
Solution: When a game is started, it is randomly decided who is playerX and who is playerO. Whomever is playerX is then also assigned as the active player. The active player field in the tttgame table is what keeps track of whose turn it is. A game can start where player X is not the active player if it is a rematch game, which will not update the active player but merely let who lost be the first player.

Issue: In order to make a move, a user has to click on the square they want to make their mark

in. What mechanism determines where they clicked?
Solution: There are click handlers in the tttboard.js function. This is where the double class assignment to the boxes makes the code simpler because whenever a box is clicked, there is a reaction and the reactionary function can be fed the second class which determines exactly which box it was that was clicked.

Issue: What mechanism prevents a user from making a move when it is not their turn?
Solution: This is the first check whenever a user clicks on a box. The play() function calls isActivePlayer.php which returns whether the current player is also the active player. If false, an error message is displayed otherwise it allows the user to play where they clicked, if it was a valid move.

Issue: What mechanism keeps tracks of all the moves played?
Solution: When player makes a move, updateMoves.php updates the game's corresponding entry in the tttgame table-- its' "moves" field will be edited with with X or O depending on who is activePlayer.

Issue: A game is considered done when there is a tie, one user has won, or one user has abruptly quit out. What mechanism determines when any of these situations happen?
Solution: There is a function within the tttboard.js script called "noticeListener" that calls on the "getNotice.php" script. The notices are set when a player wins, ties, or quits a call to updateNotice.php allows the javascript to set the notice to whatever is required. The listener allows the other player who did not win, quit, or initiate a tie to respond to these events. The function checkWin() checks after every play to see if there has been a tie or if any of the win conditions have been met (checks rows, columns, and diagonals for all of one symbol). If a win or tie is found then code is executed to set the notice and inform the other player of a win or tie.

Issue: In a series of games against the same opponent, there are banners just above the gameboard displaying the current score. What mechanism keeps track of the score?
Solution: If both players approve of a rematch then the score is updated with either a writeTietoScore.php or writeWintoScore. This modifies the score variable in the tttgame table.

Issue: It is only in "Vs. Random Player" Mode that the system distributes the appropriate amount of points to the users' stats on a game's completion. What mechanism determines the points and changes both users' stats as necessary?
Solution: The check for this is in the php code itself. When delegateWin.php, delegateTie.php, or delegateLoss.php the code first makes sure that the type of game that is being played is registered as "random" otherwise it does not change the player's stats.

Issue: A user has the option of playing against the AI. What mechanism is responsible for the implementation of the AI?
Solution: The AI player algorithm is stored seperately in the ai.js script. The AI performs primarily via the "drawO" function. The function is called when it is the AI's turn and there are still

remaining squares on the board-- the function selects a square, and marks the appropriate cell in the array storing all previously made moves. If the chosen square, and thus the cell, has already been marked, then “drawO” is called recursively until an empty square is chosen.

[DATA]

We created our database using the MySQL that comes with WAMP/MAMP/etc.. The database we created is called “tttdatabase” and has four different tables: playerinfo, playerstats, friendslist, and tttgame.

Table 1: Columns from playerinfo table

Field	**Type**	**Null**	**Key**	**Default**
username	VARCHAR (20)	NO	PRIMARY	
password	VARCHAR (30)	YES		NULL
email	VARCHAR (30)	YES		NULL
onlinestatus	VARCHAR (7)	YES		Offline
ingame	TINYINT(1)	YES		0

Table 1 refers to the playerinfo table which contains five fields username, password, email, onlinestatus, ingame. Username is the primary key because every username is unique but also so that other tables can connect via Foreign keys. Onlinestatus is used to keep track of whether the player is logged in or not. The ingame field is used to track whether or not a player is currently playing a game which is useful when it comes to managing the tic-tac-toe board and making sure other players cannot then start a game with a user that is already “ingame.”

Table 2: Columns from playerstats table

Field	**Type**	**Null**	**Key**	**Default**
username	VARCHAR (20)	YES	FOREIGN KEY REF playerinfo(username)	NULL
rank	INT(11)	YES		0
wins	INT(11)	YES		0
losses	INT(11)	YES		0
ties	INT(11)	YES		0
total	INT(11)	YES		0
standing	INT(11)	YES		0

The player stats table is linked to the playerinfo table by using the username field as a foreign key. We chose to create a separate table for playerstats based on how infrequently the data in playerinfo and playerstats are jointly referenced. It is faster to have the stats info in a separate table then by over encumbering the playerinfo table. The fields are self-explanatory except for the difference between rank and standing. Rank is based off of a point system that awards 1 point for a tie, 3 points for a win, and subtracts 1 point for a loss therefore multiple people can have the same rank. Standings are entirely unique; they are calculated based on comparing ranks with further levels of comparison in the order of wins, losses, ties, total, and then alphabetically. The standing is assigned when generating the leaderboard.

Table 3: Columns from friendslist table

Field	**Type**	**Null**	**Key**	**Default**
username	VARCHAR (20)	YES	FOREIGN KEY REF playerinfo(username)	NULL
friend	VARCHAR (20)	YES	FOREIGN KEY REF playerinfo(username)	NULL

The friendlist table implements a buddy system. Each entry is for a user that is friends with another user. This is not necessarily a mutual friendship and there are no “friend requests” in the code implementation.

Table 4: Columns from tttgame table

Field	**Type**	**Null**	**Key**	**Default**
playerX	VARCHAR (20)	YES	FOREIGN KEY REF playerinfo(username)	NULL
playerO	VARCHAR (20)	YES	FOREIGN KEY REF playerinfo(username)	NULL
moves	VARCHAR (9)	YES		“---------“
gametype	VARCHAR (6)	YES		NULL
score	VARCHAR (10)	YES		“0,0”
activeplayer	VARCHAR (20)	YES	FOREIGN KEY REF playerinfo(username)	NULL
notice	VARCHAR(7)	YES		“request”

The tttgame table holds the record of the games. Each game is a temporary entry because games are deleted upon completion. playerX and playerO are assigned randomly but the activeplayer will always start off as whoever is playerX. When playing successive games, a game may start where the starting player is not playerX and this is because the game is never deleted the activeplayer just continues to alternating making the loser the first to play in the next game. The moves field starts off with a completely empty board, where an empty spot is denoted with a “-”. The nine characters of the moves field represent the nine boxes of the tic-tac-toe board row by row as shown in table 5. The score is represented separated by a comma where the number before the comma is the score for playerX and the number after the comma is the score for playerO. The notice field is used for communicating between players when one user’s code requires action from the other player. For instance, if one player wins then notice is set to “loss” or “lossr” if the winner wishes to play again then the loser’s code responds to this in their code. In the implementation, the notice can have a value of “loss”, “lossr”, “tie”, “tier”, “quit” and the default “request”.

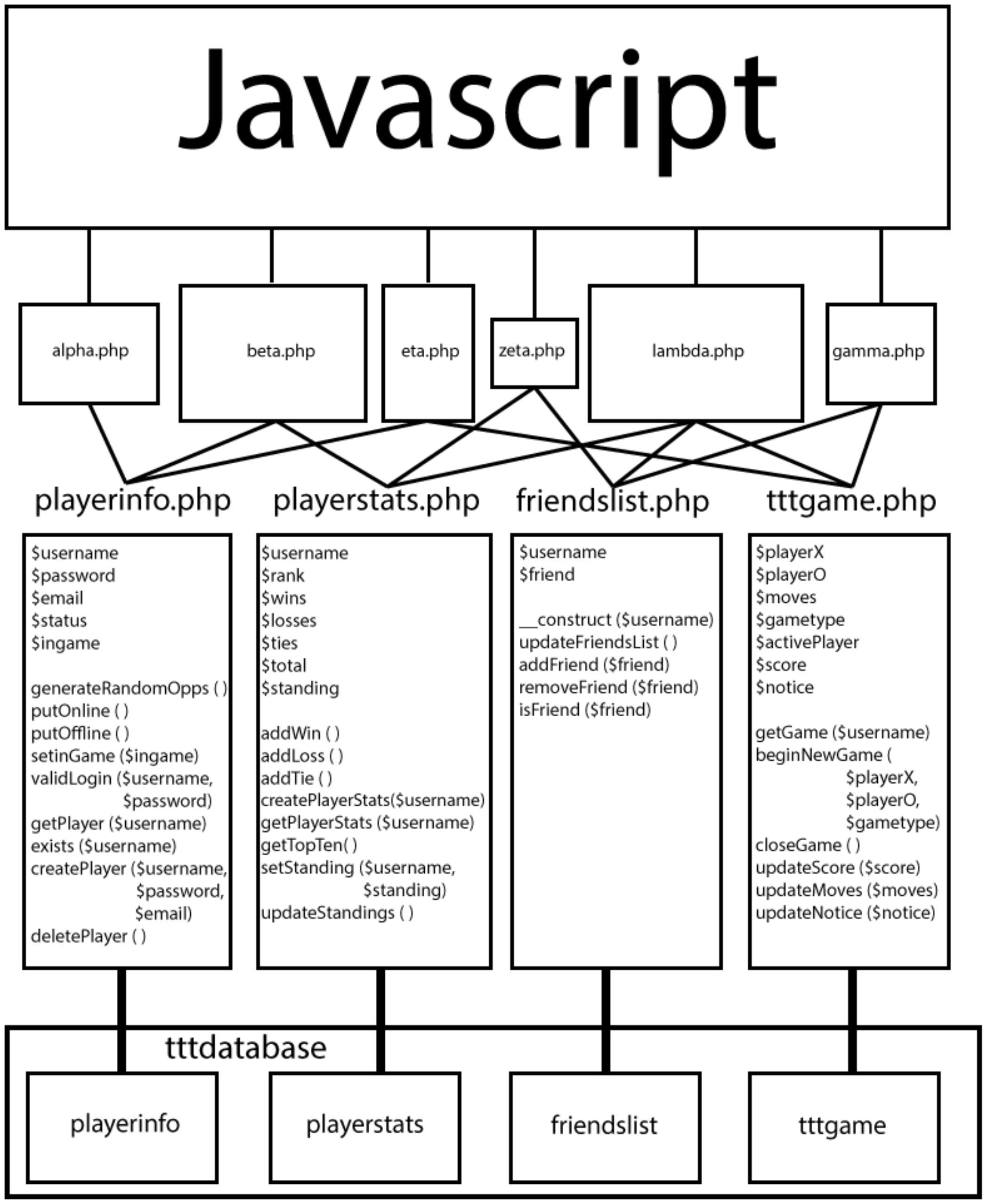
Table 5 : Moves representation on the tic-tac-toe board

0	1	2
3	4	5
6	7	8

[SERVER COMMUNICATION]

As a server we used WAMP/MAMP/etc. software. A single computer was used as the main server and others could connect using the ip address. The server computer had all of the code and the MySQL database set up as described in **[DATA]**. Then, four php classes (corresponding to the four tables) were implemented to allow easy access to the database. These scripts took care of all database queries while other php scripts served essentially as javascript extension functions for accessing the database.

Diagram 6: Server Side Code Organization



```
$username
$password
$email
$status
$ingame

generateRandomOpps ( )
putOnline ( )
putOffline ( )
setinGame ($ingame)
validLogin ($username,
            $password)
getPlayer ($username)
exists ($username)
createPlayer ($username,
              $password,
              $email)
deletePlayer ( )
```

```
$username
$rank
$wins
$losses
$ties
$total
$standing

addWin ( )
addLoss ( )
addTie ( )
createPlayerStats($username)
getPlayerStats ($username)
getTopTen( )
setStanding ($username,
             $standing)
updateStandings ( )
```

```
$username
$friend

__construct ($username)
updateFriendsList ( )
addFriend ($friend)
removeFriend ($friend)
isFriend ($friend)
```

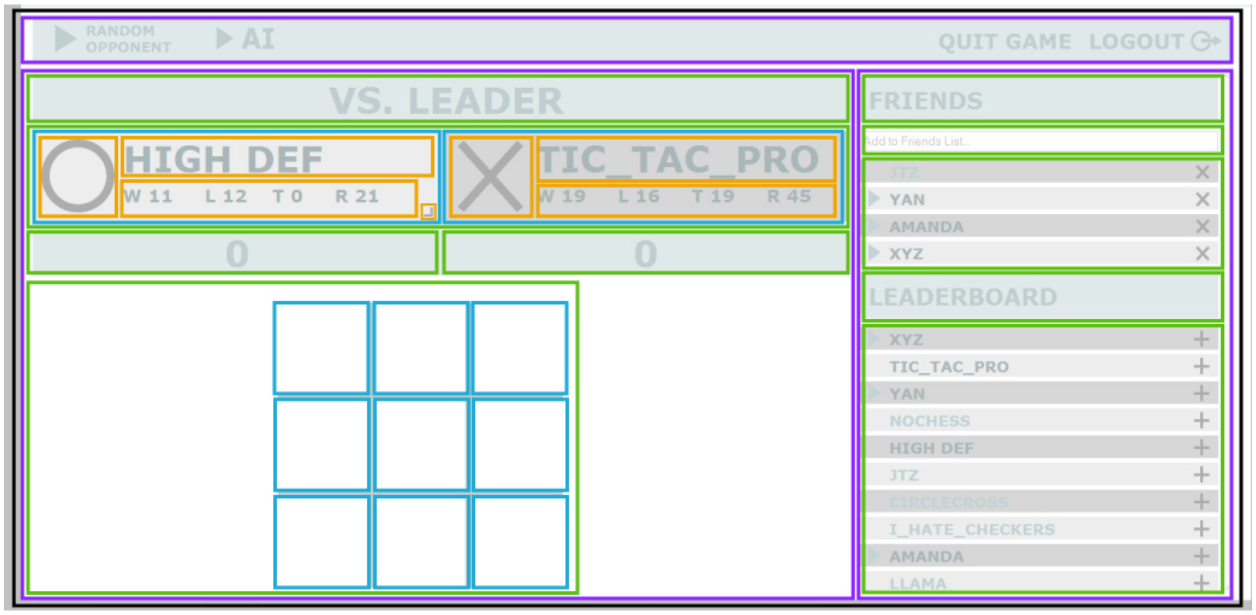
```
$playerX
$playerO
$moves
$gametype
$activePlayer
$score
$notice

getGame ($username)
beginNewGame (
        $playerX,
        $playerO,
        $gametype)
closeGame ( )
updateScore ($score)
updateMoves ($moves)
updateNotice ($notice)
```

The diagram above shows the basic layout of the code from the perspective of the server. As mentioned above, the tttdatabase holds four tables each communicate directly and independently to four php classes. Included are the public variables and relevant functions of each class. These classes then communicate with multiple php scripts that use functions and variables from the provided classes to write and read from the database. The secondary php scripts are then called from the javascript which acts on the information given. When communicating between the javascript and php scripts we encoded everything into JSON notation. However there are php scripts that have an automatic function (e.g. setting ingame to 0) which are just called by the javascript but no information is received back to be processed. For a more detailed understanding of each function, please see inline comments in the corresponding file.

**names of secondary php scripts (alpha.php, beta.php, etc.) are fictitious. Any resemblance to real files, active or inactive, is purely coincidental.*

<u>[SYSTEM DIAGRAMS]</u>

Diagram 1: Index.html div hierarchy



The html code for the main page (index.html) is organized into 5 levels of divs. The outer layer (#gameContainer) is in black and used mainly for resizing erroneous css percentages to actually fit the screen. Next there are the three divs in purple: the menu bar (#menuBar), side bar (#sideBar), and main board (#playerBoard). The menu bar is not divided any further and the images that serve as buttons are appended in menuBar.js.

The side bar has 5 divs of equal depth in green: a banner to denote friends (#friendsBanner), a div which is actually a form for adding friends (#addFriendForm), the div where the friends list is placed (#friendsList), another banner to denote the Leaderboard section

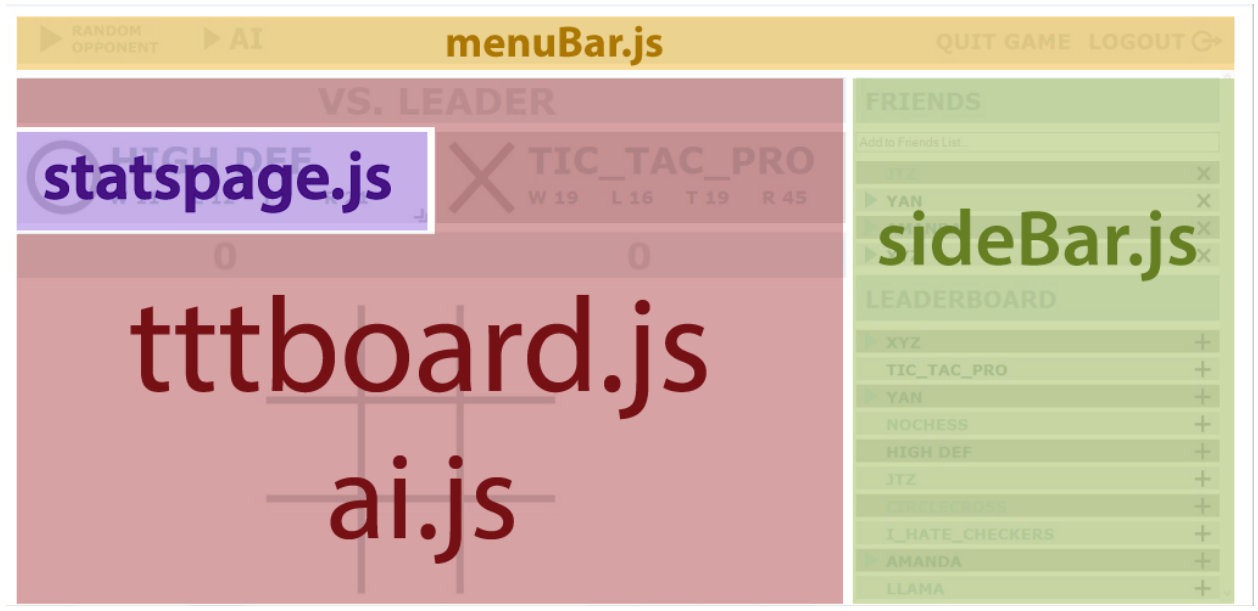
(#leaderboardBanner), and finally a div where the actual leader board is displayed (#leaderboard).

Next the player board has several layers of divs. In green, is the Banner for displaying game type--vs. Friend, vs. AI, vs. Leader, or vs. Random-- (#gameTypeBanner), the div for holding the banners for each player (#playerBanners), the two divs for holding the players' respective score (#player1scoreBanner, #player2scoreBanner), and the gameboard (#gameboard). Each player banner in blue (#player1Banner, #player2Banner) holds information specific to the player, the player 1 banner is more extensive because it is also responsible for displaying the stats page. Therefore, in the player 1 banner there are shown divs in orange for displaying the x or o assignment (#player1_X_O), the username (#player1name), the summated stats (#player1stats) which holds a "pre" div so that the html rendering would include spacing, and an img div (.extendarrow) to show the clickable area to display the stats page--the image is actually appended in statpage.js. All of these areas are replicated in the player 2 banner area in orange except for the arrow to show the stats page (#player2_X_O, #player2name, #player2stats). Then there are several hidden children in the #player1Banner for displaying the stat page. The four divs which are all actually pre divs wrapped in divs include one for displaying the player name (#player1statsPageName), another for displaying the stat names (#player1statsPage)--win, tie, lose, standing--and a separate div for displaying the number values of these stats (#player1statsPage2), and finally a div for displaying the rank of the player (#player1statsPageRank). The game board holds the 9 divs in blue that serve as the nine boxes of tic-tac-toe. Each div has two classes : a general (.box) class and another class for specifying the specific box (.b1 - .b9). We included two classes because it makes handling click events much easier.

The divs are all styled using indexstyle.css. In general, the divs are all named using ids and the imgs are all referenced using classes. No images are appended in index.html but in the corresponding Javascript classes. Also very little text is provided, only for divs that have a constant text value. This index.html script is the essential model for our system.

**div elements are referenced using their JQuery equivalent (#something for divs referenceable with ids, .something for divs referenceable with classes)*

Diagram 2 : Division of Javascripts on the Main Page

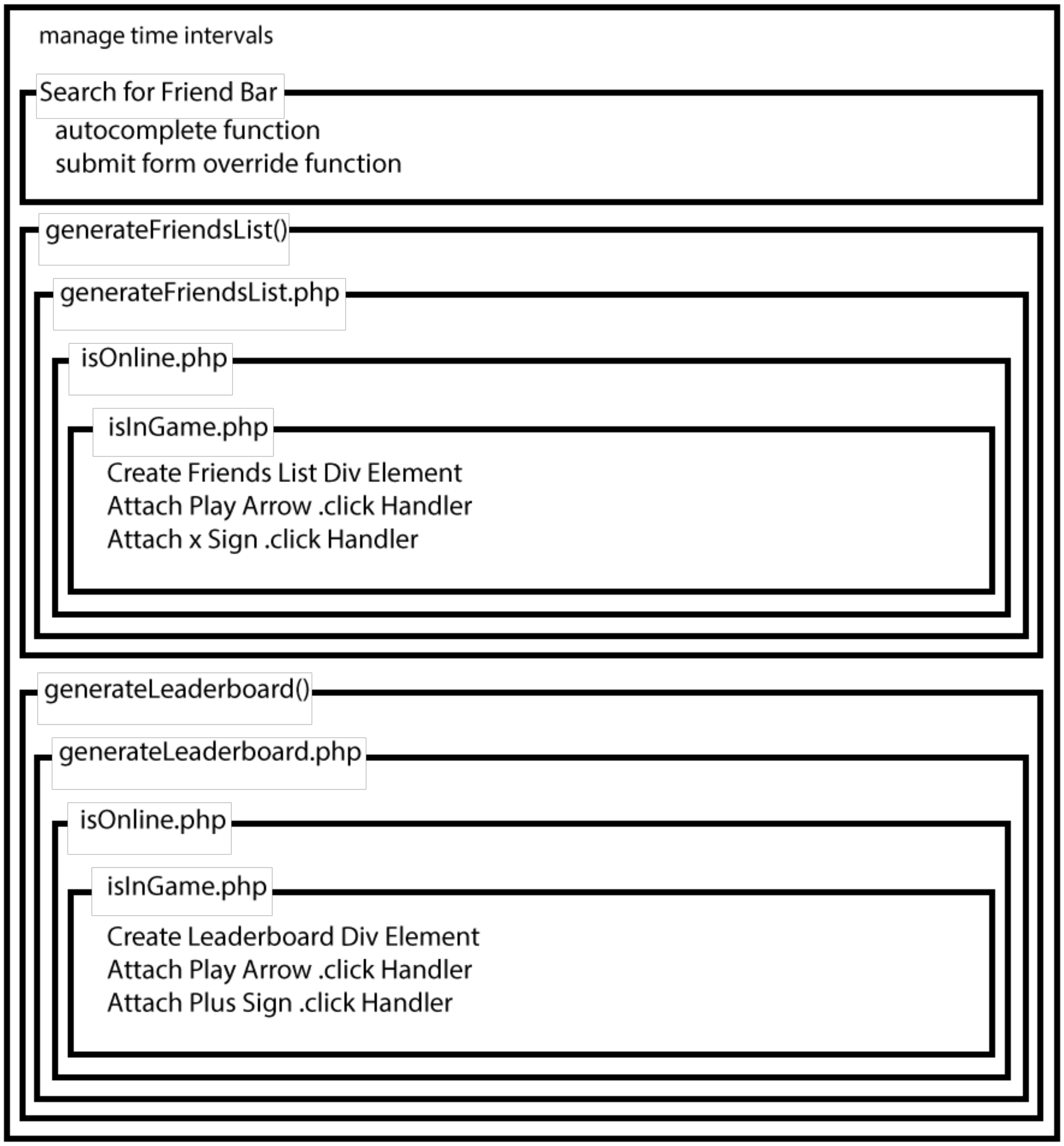


The main page of tic-tac-toe is divided as shown above. There is some crossover where the quit game and play vs. ai click handling is done by tttgame.js and ai.js, respectively. Otherwise there is little to no crossover. The area in red is mainly controlled by tttboard.js, when playing against anything but the ai. When playing against the ai, ai.js takes effect and populates and controls the area in red.

The menu bar javascript has the task of appending all images then handling some of the click functions. There are four images in the menu bar, one to play a random opponent, one to play the ai, another to quit a game and the last to logout. The quit game img is actually hidden until made visible by tttboard.js which also handles the click event. ai.js handles the click event for the img corresponding to vs. ai. menuBar.js handles the click functions for playing vs. a random opponent and the logout imgs. Therefore, our “buttons” throughout the code are actually imgs or divs paired with click functions.

The side bar has two major functions: the first is to manage the friends list and the second is to manage the leaderboard.

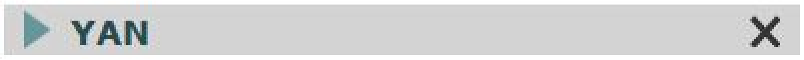
Diagram 3 : Explanation of sideBar.js


manage time intervals
Search for Friend Bar
autocomplete function
submit form override function
generateFriendsList()
generateFriendsList.php
isOnline.php
isInGame.php
Create Friends List Div Element
Attach Play Arrow .click Handler
Attach x Sign .click Handler
generateLeaderboard()
generateLeaderboard.php
isOnline.php
isInGame.php
Create Leaderboard Div Element
Attach Play Arrow .click Handler
Attach Plus Sign .click Handler

The above diagram illustrates the nested relationship in the sideBar.js file. Making sure the current user's friends list and displayed leaderboard are up to date is handled by a series of set time intervals. Because these intervals impact the performance of the site negatively, there was a mostly reactionary approach. Instead of having faster time loops, we chose longer ones but the lists will still be kept up to date by the .click functions. Changes to the friends list are going to happen when a user adds or removes a friend. When these buttons are clicked, the generateFriendsList() function is called and the list is updated. The slower loops will catch when a player goes offline, comes online, or begins playing a game.

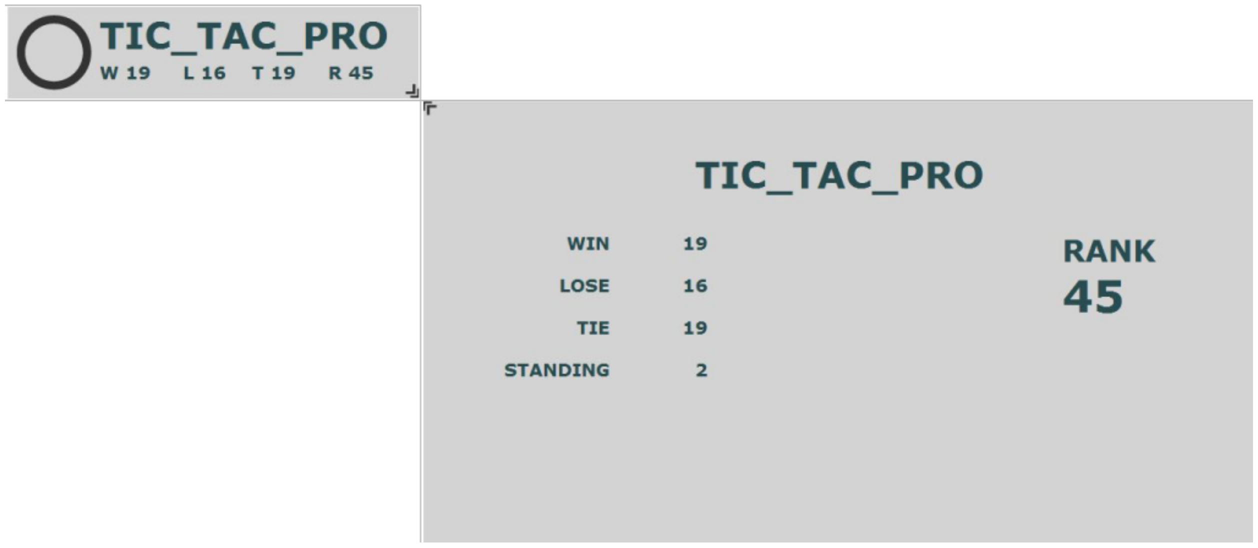
The friendslist and leaderboard div elements are created in very similar ways. generateFriendslist.php and generateLeaderboard.php both return a JSON array of the corresponding list and the rest of the code essentially styles it. isOnline.php determines what color the text of the username will take on: a lighter blue for offline and a much darker blue for online. isInGame.php determines whether there will be a play arrow displayed, obviously the current player cannot start a game with someone who is already playing one and therefore the option is removed. The actual creation of the div element results in the following html code.

```
        <div class = “friendentry” style = “background-color : #D3D3D3”>
                <img class = “activearrow” data-opponent = “Yan” style = “” src =
“img/activearrow_light.png”>
                Yan
                <img data-friend = “Yan” class = “x” stc = “img/x_light.png”>
        </div>
```



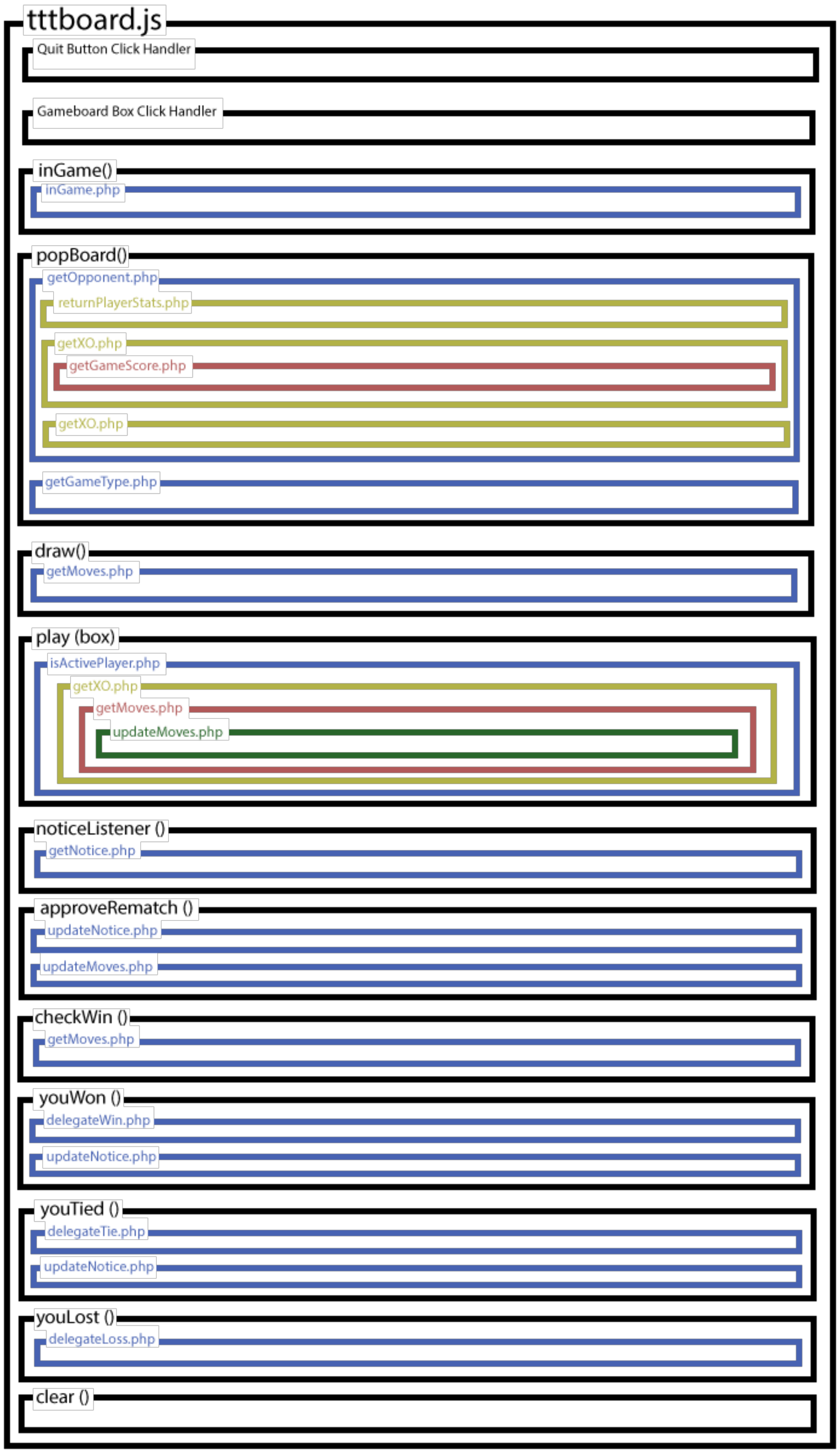
The styling is done mainly with indexstyle.css. The functions also alternate between lighter backgrounds and darker backgrounds for easy distinction between successive divs. The function to generate the leader board functions similarly to the function to generate the friends list; they both follow the same model.

Diagram 4 : Example of statsPage.js



The stats page does not use any complicated code. There is a lot of repetition but this is because in order to display the stats page a majority of the #playerboard must be hidden. The stats page is shown whenever the extension arrow is clicked in the player 1 banner. The code uses animation to expand the div then fill it with previously defined but previously hidden.

Diagram 5 : Explanation of tttboard.js



tttboard.js has possibly the most complex code out of all of the javascript files because it has the most complex uses. The main functions and their uses are listed below.

inGame()

This function is originally set at the beginning of the document to check to see if a player is in-game. This feature corresponds to the fact that when a game commences it only inserts a new row into the tttgame table and sets both players to in-game. Therefore, this function serves as a listener for when the player enters a game so that it can begin reading from the database.

popBoard()

This function is used to populate all player board elements except the player 1 banner. It does show the player 2 stats, the x and o assignments for both players, the gametype, and the score.

draw()

This draws the gameboard based on any moves that have been recorded in the database.

play(box)

This is the function associated with clicking on any of the 9 divs that make up the gameboard. It makes sure a move is valid then registers it if true. First, it checks to see if the current player is the active player, otherwise it outputs an error message. Secondly, it checks to see if the move is valid where the code uses the data-occupied attribute of each box element if the attribute is true then the move is invalid, otherwise it continues. It has now established that the move is entirely valid so it figures out whether the current player is x or o then writes the appropriate symbol and location to the database.

noticeListener()

This is what the code uses to check for changes in the game state based on what the other player is doing. It is the most direct communication between the two players. In the above diagram this code is over-simplified because, in fact, there are many case statements based on the options that notice can be (“loss”, “lossr”, “tie”, “tier”, “quit”, “approve”, “reject”). Since the notice is based on another players action there is no option for win because the current player will always execute the youWon() function without need of input from the opponent.

approveRematch()

This code is executed multiple times throughout the document and it was easier and clearer to make a separate function. If a rematch is approved, this is the code that sets up a new game.

checkWin()

This function checks the current board state versus the typical win and tie conditions for tic-tac-toe

youWon()

This function is more complex than what is described in the above document. In addition to what is in the above diagram, it checks to see if the player would like to play again if so then the player is set into a “waiting approval” state, waiting for the opponents response.

youTied()

This function is tricky because a tie applies equally to both players yet one must initiate the actions for the other. A better understanding of what happens to the player that begins the tie state, see the first several lines of checkWin() that pertain to checking for a tie scenario.

youLost()

This code is entirely reactionary to the opponent's win scenario. As such, it does not touch the notice field and only delegates a loss to the stats of the current player if the gametype is "random" which applies to all changes of stats.

clear()

clears all entries made by popBoard(), returning them to their original, blank states.

<u>[LIST OF PHP SCRIPTS AND THEIR FUNCTIONS]</u>

addTietoScore.php
- includes tttgame.php
- arguments: none
- modifies score variable : adds one to both sides
- echos new score

addWintoScore.php
- includes tttgame.php
- arguments: none
- modifies score variable : adds one to current player's side
- echos new score

getGameScore.php
- includes tttgame.php
- arguments: none
- modifies nothing
- echos (string, JSON) score

getGametype.php
- includes tttgame.php
- arguments: none
- modifies nothing
- echos (string, JSON) gametype

tttlogout.php
- includes tttgame.php, playerinfo.php
- arguments: none
- modifies onlinestatus : puts player offline, if in tttgame sets notice to "quit"
- echos nothing

quitGame.php
- includes tttgame.php
- arguments: none
- modifies notice : sets to "quit"
- echos nothing

getRandomPlayer.php
- includes playerinfo.php
- arguments: none
- retrieves list of random opponents, selects one at random that is not the current player
- echos (string, JSON) random player

updateNotice.php
- includes tttgame.php
- arguments: notice=newNotice
- updates notice to new value
- echos nothing

getNotice.php
- includes tttgame.php
- arguments: none
- modifies nothing
- echos (string, JSON) notice

exitGame.php
- includes playerinfo.php
- arguments: none
- sets inGame to 0
- echos nothing

autoselect.php
- includes: none
- arguments: searchTerm
- goes through playerinfo table and gets all usernames similar to search term
- echos (array, JSON) list of matching usernames

delegateLoss.php
- includes tttgame.php, playerstats.php
- arguments: none
- if user is playing a random player, adds loss to stats
- echos nothing

delegateTie.php
- includes tttgame.php, playerstats.php
- arguments: none
- if user is playing a random player, adds tie to stats
- echos nothing

getOpponent.php
- includes tttgame.php
- arguments: none
- returns the name of player that is playing against the current user
- echos (string, JSON) opponent name

inGame.php

- includes playerinfo.php
- arguments: none
- returns true if player is in-game false otherwise
- echos (bool, JSON) ingame

endGame.php

- includes tttgame.php, playerinfo.php
- arguments: none
- deletes game from tttgame table, set ingame of both players to 0 a.k.a. false
- echos nothing

delegateWin.php

- includes tttgame.php, playerstats.php
- arguments: none
- if playing against a random opponent, add win to stats
- echos nothing

startGame.php

- includes playerinfo.php, tttgame.php
- arguments: {type : “gametype”, opponent : “opponent”}
- starts new tttgame, generates either 0 or 1 randomly for choosing who is playerX, then sets both players’ ingame to 1
- echos nothing

isInGame.php

- [redundancy] -- deal with

getMoves.php

- includes tttgame.php
- arguments: none
- either returns false if the tttgame has been deleted otherwise returns a JSON array of the stored moves
- echos (array, JSON) moves

switchActivePlayer.php

- includes tttgame.php
- arguments: none
- switches the active player (see tttgame.php)
- echos nothing

updateMoves.php

- includes tttgame.php
- arguments: {moves : “x-o-----x”}
- sets moves equal to new moves string value
- echos nothing

getXO.php

- includes tttgame.php
- arguments: {username : “username”}
    - if no username is specified, assumes current player as username
- returns either “x” or “o” depending on whether the “username” is equal to playerX or playerO
- echos (string, JSON) “x” or “o”

isActivePlayer.php

- includes tttgame.php
- arguments: none
- checks to see if the current user is the active player, returns “no game” if there is no tttgame associated with the user
- echos (bool or string, JSON) true/false/”nogame”

returnPlayerStats.php

- includes playerstats.php
- arguments: {username : “username”}
    - if username=”this” assumes current user as “username”
- gets all playerstats for specified user
- echos (array, JSON) [username, wins, losses, ties, total, rank, standing]

isOnline.php

- includes playerinfo.php
- arguments: {friend : “username”}
- returns whether “username” is online or offline
- echos (bool, JSON) true/false

generateLeaderboard.php

- includes playerstats.php
- arguments: none
- updates the standings for all players then get the top ten players based on the new standings
- echos (array, JSON) top ten players

addFriend.php

- includes friendslist.php
- arguments: {friend : “newFriend”}
- if the “newFriend” is not the current player or already friends with the current player, they are added as a new friend
- echos nothing

getTopTen.php

- [same thing as generate leaderboard.php] tttReg.php

- includes playerinfo.php
- arguments: {username: “username”, password: “password”, email: “email”}
- checks if username already exists-- adds into as entry to playerinfo if false, rejects registration attempt if true
- echos message alert and javascript window change if registration failed

tttlogin.php

- includes playerinfo.php
- arguments: {username: “username”, password: “password”}
- checks playerinfo table to see if username and password are valid, then either logs user in or rejects login attempt
- echos message alert and javascript window change if login failed

removeFriend.php

- includes friendslist.php
- arguments: {friend : “username”}
- deletes “username” from current player’s friends list
- echos nothing

generateFriendsList.php

- includes playerinfo.php
- arguments: none
- gets the friends of the current player
- echos (array, JSON) friend list