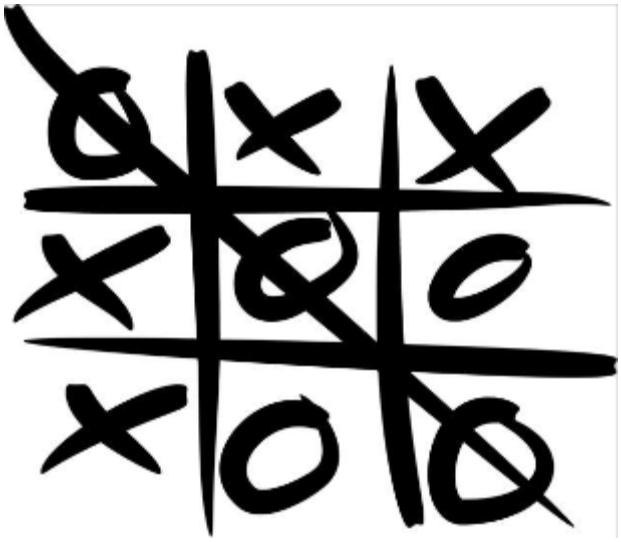
TIC TAC TOE Game



Student Name	Student Number
Bozhidar Lenchov	995959431
Shahriar Ninad	993690537
Kush Dua	996081957
Harkanwal Sidhu	999785996

1. General application architecture

The application generally follows the 3-tier architecture, where the web application rendered on a browser is the presentation layer, the java servlets (and helper classes) form the business logic layer and the Google Appengine Datastore form the data layer.

The User Interface is handled through jsp pages, where some UI logic is coded in Java and mixed with HTML to provide a dynamic page. We have used CSS and JQuery to make the UI consistent across different pages. For some pages (e.g play.jsp), we used JQuery to make AJAX calls to the web server and manipulate page elements. For some pages (e.g view_games.jsp and selectplayer.jsp), clicking a button (e.g “Start New Game”, “Accept”, “Return to Game”) does a HTTP Post to a servlet which handles the request, does some processing based on the type of request (e.g “starting a new game” or “accepting a game invite”) and redirects the user to an appropriate page (e.g return to game request redirects the player to the play.jsp page where he/she can resume a game). The Datastore queries are done by different helper classes (e.g Helper.java) and the servlets call the functions from the helper

classes to aid their processing.

1.1 Storage Model

We have two Entities - UserPrefs storing user details (user object, number of games played/won/lost/drawn, etc) and TTTGame storing game state (current board, past moves, player User objects, winner, game status (active, accepted), etc). Kinds are presented in more details below:

UserPrefs:

Name	Kind	Strategy (if applicable)
userEmail	String	Identity (primary key)
errorMessages	String	
successMessages	String	
user	User	
loggedIn	boolean	
rating	int	current rating of user
GamesWon	int	number of games won by user
GamesDrawn	int	number of games drawn by user
GamesLost	int	number of games lost by user

TTTGame:

Name	Kind	Strategy (if applicable)
gameId	String	Identity (primary key)
user1	User	user who initiated the game
user2	User	user who was invited to the game by user1
isActive	boolean	indicates if the game is active

isAccepted	boolean	indicates if the invited user accepted the game
isRejected	boolean	indicates if the invited user rejected the game
contentsOfBoard	String	
nextTurnUser	User	indicates whose turn it is
winner	int	Who was the winner - int with values 1 or 2 for user 1 or 2 respectively, or -1 for draw
boardHistory	ArrayList<String>	

1.2 Memcache Usage

We used memcache for saving user data which might not change often (e.g. each user's active and completed games and a game's current state (assuming time between user moves is significant and not real-time)). We use JCache to achieve this in the relevant Helper class and GameContents methods using these lists. As a future improvement, instead of storing/invalidating these game status lists, we could maintain a synchronized datastore image in this distributed memory cache, and populate it only on app startup. However, due to time limitations (for testing and ensuring proper app code synchronization) this is not done in this version of the app. Caching Active/Invited/Past game lists and game contents should however yield substantial improvement in the app speed.

1.3 Gameplay

To begin a game, a user has to find another player (e.g. search through user list). For future improvements, integration with autocomplete plugin (e.g. Select2 available at http://ivaynberg.github.io/select2/#basics) is possible. When the game is created by our StartGame servlet, the first move is decided based on the players' ratings. Ratings are used in such a way that stronger players have less of a chance to go first if they are playing against a weaker opponent. If players are evenly matched, they will have a 50-50 chance of starting the game. The user starting the game is redirected to the play page if it is their turn, or to the page to View Games if it is the other player's turn (if they accept the invitation). The gameID reflects the invitation user's userID as well as the current time.

In the actual game page, synchronous JavaScript calls are issued every 10 seconds whenever a user is inside a game. They are used to update board contents, only if it is not the current user's turn (e.g. wait until other player makes a move; emulate real-time play). Game logic is implemented in JavaScript and calls to update game state are sent to the GameContents servlet

as appropriate. If synchronizing with server hangs, the user can either refresh the gameplay page through their browser, or go back to View Games and reenter the game from there.

2. Instructions for accessing the application

Navigate to http://ece1779-group1-project2.appspot.com/ and click the Login link. Once logged in, you can navigate to “View Games” page to start a new game, return to a game in progress, accept/reject games that other players invited you to, or do a rematch with a previous player you have played with. You can also get your game statistics from the “Statistics” page.

Since the game only supports 2-player games and no AI, you need at least two different google accounts to test the application. You can sign into the application from two different browsers using two different google accounts, and then you can play against yourself.

3. AppEngine credentials
We have added delara@cs.toronto.edu as an admin for the app.

Thank you and enjoy our creation :)
-Shahriar, Bozhidar, Kush and Harkanwal