Contributors: Walker White, Benjamin Montanez, Matthew Gin

Overview

The final app will consist of a tic-tac-toe game. The game will contain both singleplayer and multiplayer elements. For singleplayer interaction, the app will have an AI that will play games against the player. For multiplayer interaction, players will face off against each other. Each player will have an account that keeps track of their lifetime number of wins, losses, and draws. The leaderboard will display the scores of the top 10 accounts, as well as the overall score of the AI.

Features like the game state, account data, and AI moves will all occur on the server. Player moves and login info will be generated on the client side, but will be sent to the server via an API. Game state and win status will be sent to the client from the server via another API.

Backend

Updating the game state will be handled by a series of POST requests made by the client. The POST request will include the player's move, and the server response will indicate whether that is a valid move, and the updated game state. Examples of a valid move might be ‘A2’, while invalid moves could involve the player attempting to move to a board space that is already occupied, attempting to move when it is not their turn, or attempting to move for the other player.

Example POST request:

```
fetch("http://example.com/456e7890-e89b-12d3-a456-426614174001/123e4567-e89b-12d3
-a456-426614174000", {
  method: "POST",
  body: JSON.stringify({
    move: "A2"
  }),
  headers: {
          "Content-type": "application/json; charset=UTF-8"
  }
});
```

The leaderboard will be retrieved from the server via a GET request. However, the client will not be responsible for notifying the server of an update to the leaderboard. This will be handled server-side when the server recognizes a finished game state.

Player identification will be handled by assigning a UUID to each user account. The server will store the account name strings, and their associated UUID in a mongodb database. When a player logs in, they will be redirected from http://example.com/login to http://example.com/UUID. When the player makes their post request to the server, seeking to update the game state, this UUID will be used to ensure that the player's requests only affect that player's game state.

When a player joins a match, their match will be assigned another UUID, representing the game's state, and indicating to the server which game state to send to the client. For example, the player will join a game vs the AI, the server will generate a UUID for the game room, and redirect the player to that room. The player would be redirected from http://example.com/playerUUID to http://example.com/playerUUID/gameUUID. The server will determine whether the user is one of the one (or two) game players. If they are, the player will be able to make moves and update the game state. If they are not, the player will be able to view the game being played, although without the ability to make moves.

Because the game's UUID is a unique identifier of that game to the server, it can be used to generate shareable links for each game lobby. When the user navigates to http://example.com/gameUUID, they will be directed to the login page. The game UUID will be stored as a query parameter like so: http://example.com/login?gameUUID=<gameUUID> . Once they are logged in, if there are less than two players in the lobby, the user will become a player. If there are already two or more players, the user will become a spectator.

The database's primary key for each account will be their string username. Associated with that username will be several pieces of information. It will include their UUID and their number of wins, losses, and cats.

There will be an additional database that will store all game states, as well as which players are playing them, and whose turn it is. Game requests will be handled asynchronously by Node.js's native concurrency handling.
Because tic tac toe is a solved game, there will be two versions of the AI. The hard version will be impossible to beat, and will be hardcoded to always win or cat. There will also be an easy version of the AI that will make random moves, except when a win or loss is imminent, in which case it will make the winning move or block the opponent's winning move.

At the very least, the server will require the modules 'express', 'mongoose', and 'uuid', although more modules may prove necessary as time progresses.

Timeline (11/22 - 12/13):

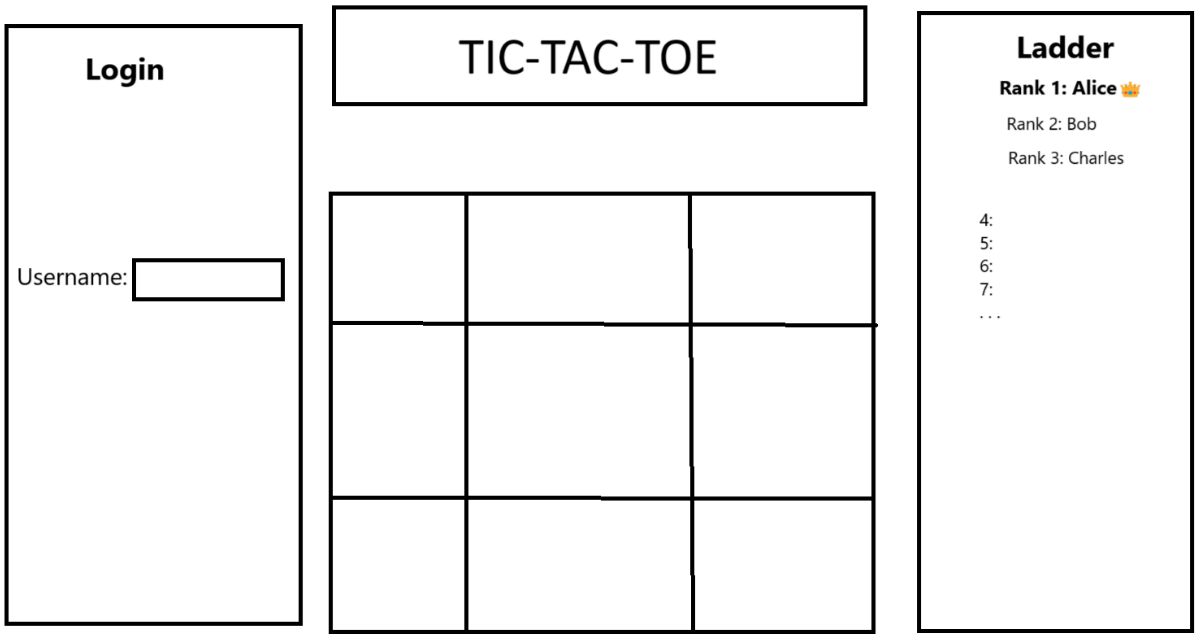
Week 1
NodeJS Server (to handle interactions between the user, database, and server): 10 hours
Users Home Page/Game Page: 5 hours
MongoDb Database (to contain information regarding the user and the leaderboard): 5 hours

Week 2
NodeJS server (cont.): 5 hours
tic-tac-toe game logic: 10 hours
CSS for Home Page/Game Page: 5 hours

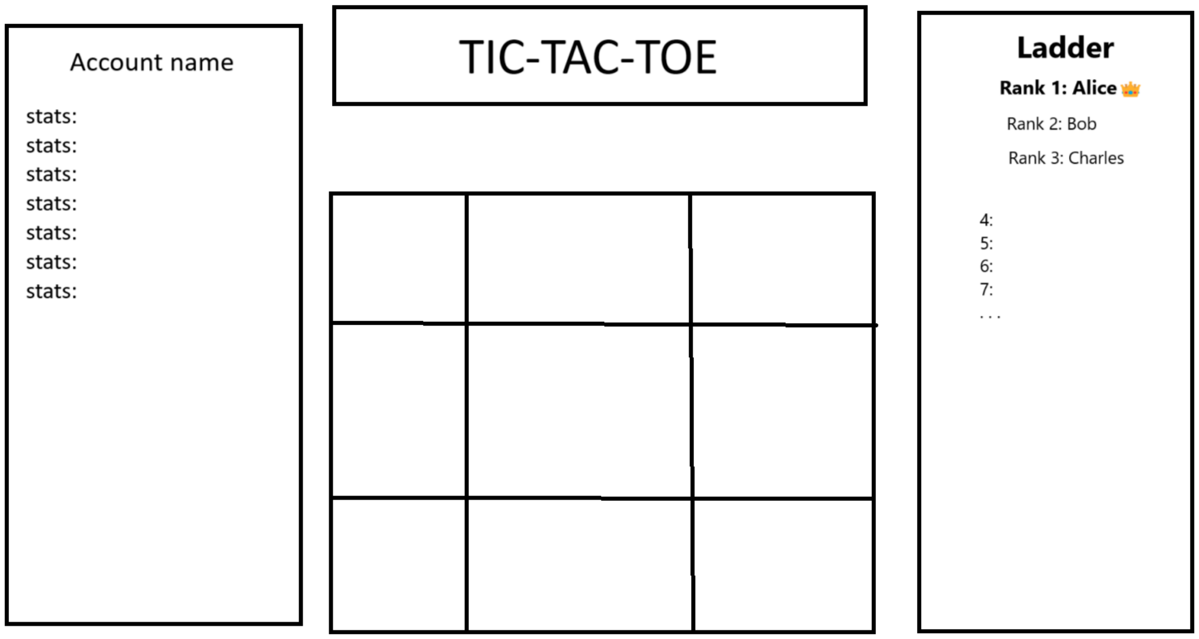
Week 3
tic-tac-toe game rendering: 10 hours
tic-tac-toe AI: 10 hours

Frontend

Page when user has not yet logged in:



Page when user has logged in:



Stats will include:
Total games played
Wins
Losses
Winrate
Elo
Rank