Matthew Fong

Real-time Project 1 Documentation – Pictionary

**Site’s Purpose:**

For project 1, I decided to make a very simple Pictionary game with canvas and websockets. Players have to guess the word that the drawer is drawing. If a player guesses correctly, they will receive a point and will receive a random word to draw. The first person to three points wins and a new game will start again.

**Websocket use:**

For this project, I am using socket.io. My websockets are used in connecting players to multiple rooms of Pictionary games. When clients connect to the server, they are being stored in a room that is keyed inside a rooms object. A Pictionary Game begins when there are four players connected in a room. Before then, players are placed in a lobby/waiting state, where the players can freely draw and clear their canvas and chat with each other. Once four consecutive clients have joined, they are placed in a room and the next four will join a different room and so on. When a room has four players/clients connected, the game will start. A drawer will be selected by random and given a word to draw. During a game, only the drawer can draw on the canvas and clear the canvas if necessary. Likewise, the ‘guessers’, or so to speak, can only type in the chat to guess the word that the drawer is drawing. When a user draws to the canvas, their coordinates (previous X, previous Y, destination X, and destination Y), along with other canvas properties (linewidth, strokeStyle) are sent to the server. The server then updates the user’s coordinates on the server-side depending on a lastUpdate time property and the server emits that data is everyone in the room. The ‘guessers’ send their message to the server and the server actively checks to see if any of those messages are in fact the drawer’s word. If so, the user will be rewarded a point and is declared the new drawer for the next round and given another random word. The server then emits all necessary feedback to all the users in the room. Once a user gets three points, they are declared the winner and a new game starts again.

**What went right:**

Implementing the core features and gameplay mechanics for the necessity of a Pictionary Game was both very insightful and great practice in understanding the concepts of websockets. I believe the three main components of my project that I spent most, if not all my time on were drawing, messaging, and multiple rooms. Although it is expected in a Pictionary game, I wanted the experience for the players to be as realistic as possible. Therefore during a game, only the drawer can draw on the canvas and only the ‘guessers’ can chat. Implementing multiple rooms also took quite a bit of time. Structurally, the rooms object on the server is keyed by room numbers and the values are the four players in each room. I decided that four players was required to play a game and that if one disconnects, the game will end.

**What went wrong:**

Instead of continuing a game if a user disconnects from a room, I opted to just ending the game and forcing the players to reconnect to a new room to start a new game instead. This was definitely another way to handle disconnects but I would have liked to explore the other option, which involved picking a new drawer if a user disconnects, especially if the drawer disconnects and allowing the players to continue the game even with three players in a room. However, if there are only two players, then it is a bit redundant to continue, so having a room of at least three players would be the ideal route to go if I went in that direction of implementing rooms instead.

**Future Improvements:**

If I were to continue working on this project or had more time, there are a few more features I would have liked to included. For one, I would have liked for the server to announce who the drawer was each round to all the ‘guessers’ in the room. Additionally, I would like to continue adding more visual feedback for the players on the page. Rather than displaying all server and client feedback in the chatbox, I would have liked to move some feedback around, such as displaying the random word for the drawer in the canvas instead. I would also like to have some pop-up animations for when a guesser types a word in the chat. I would also like to add more drawing options (colors, tools, etc.) and continue working on CSS positioning and the overall page layout/design. Furthermore, there is also a minor bug where the ‘guesser’s can draw little lines/specs on their canvas during a game; however this is not sent to the server and is not reflected on other players’ canvases in the room. Nonetheless, I would like to spend more time investigating this issue. Finally, I would consider adding more words to my random words array, or even using an external random words API.

**Above and Beyond:**

One above and beyond features I implemented in my project was the server handling two different states when users were connected. Before a game starts, users who are connected in a room are placed in a lobby/waiting state where all connected users may freely draw and communicate with each other as they wait for the fourth user to connect. Once a fourth user connects, the server switched handling cases and begins to handle drawing and message commands during the game state, restricting the drawer to drawer-only features and ‘guessers’ to message-only features. I felt this added a small but unique feature to my app overall and definitely made my app a bit more versatile in terms of functionality. In terms of the coding required, it definitely added more monitoring and safe-checking since my server is dealing with two different states of the game. Another area I focused on for above and beyond was player expectations and app-friendly features. When a user types, they can simply press the enter key to send their message rather than being forced to click on the send button every time. Clicking the button will force the user to lose focus on the message box, forcing them to have to click back into it to regain focus which can drastically slow down gameplay. Likewise, if the user does decide to click the send button to send their messages, once they click back into the message box, their previous message will clear so that they do not have to manually erase their message to start typing a new one again. Finally, I feel my CSS and overall page layout offers a very simple yet appropriate and professional design to my overall app.