Multiplayer Connect-4 made with Phaser and Socket.io

1. `Npm install` for installing the dependencies.
2. `Npm run start` to start the server.
3. Game is located on `localhost:3000`.

Documentation

Multiplayer Connect-4 was made with Phaser (v3.15.1), Express and Socket.io. The game allows two players to join a server and play together. When a player joins, they will be assigned to a player one (black) and see the gameboard, however are unable to play until a second player joins. When the second player (white) joins, player one will be allowed to make the first turn. Players alternately place discs on a vertical board, which has seven hollow columns and six rows. The player who is the first to form a horizontal, vertical, or diagonal line of four of one's own discs wins the game.

Server Logic (server.js)

The `connection` event listener contains three event listeners on the `socket` object that listen for events from the client. `dropDisc`, `replayGame` and `disconnect`, which I will explain in the following. I am aware that it is wise to put the listening events outside from the connection listener separately, so the listeners would not be registered every time the sockets reconnect, however I couldn't find a way that would make the code work the other way.

Whenever a player joins, they will be registered in the `players` list. Dependent on the current `players` size I will determine player one and player two. After player two joined the game player one will always start the game with their turn. Since only two players are allowed to be in the room, all subsequently joining players will get kicked out by the server. Additionally, I optionally hide the gameboard and putting a text, showing that the room is full. If one of the two players decides to leave the match, another player can join. For that reason, I always update the `players` list whenever a player joins, leaves, disconnects, or gets kicked out. If for that reason a player gets disconnected from the game, independently of the current player role of the last remaining player, they should be assigned to the new player one. I think that this way it is rather easy to keep an overview who each player is and it's not important if we turn from player one to player two or the other way around.

When the `dropDisc` event is triggered, `updateGameBoard` will check if the move is valid and updates the gameboard. I will send this gameboard to all clients by emitting `updateGameBoardVisually`, which then individually updates the current Board anew visually. Afterwards I will check if via `checkWinCondition` the winning condition is met. If not, I will check with the `turnCounter` if I already have used all the 42 turns, which would mean that the gameboard is full and therefore concluding a draw. If a draw isn't the case, I will set the `next turn`, by the opposite client. Usually `broadcast.emit` would emit to all clients except the one client I was listening to. However, since I am only having two clients, it is certainly the opposing one.

*Note: Since I didn't emit events to a specific client and don't use rooms, by using `io.emit`, all clients would have been notified, even clients who are not participating in the game. That is why I limited the actual game to exactly two players. *

When a condition in `checkWinCondition` is met, I will emit `gameOver` with the winning `playerID`. The players now have the chance to replay the game, which would use `resetGame` and similar to `nextTurn` if player one clicks on replay, player 2 will start the game and the other way around. I thought about randomizing it, but I think it is better that the players can decide in that way, who actually starts the game. For aesthetic reason, I will also update the text from "Player X won the game" to "Waiting for other player to drop disc".

Thought: My idea was, that the client should only send the information of the disc's location, while the server checks if everything works out. There was also the option to let the client have their own gameboard updated from the server, which can be regularly checked with the server's one. Since both versions didn't solve the problem, that if the server hypothetically crashed, to resume the game, it didn't matter to which of them I sticked to.

Client Logic (game.js)

For the game objects I thought about a group for the gameboard via grids, a group for the discs via images of black and white circles and a button for quitting and replaying the game. There was also a lot of text coordinating the current states of the game for each player. Whenever I dropped a disc with `pointerdown` or received the turn, I had to always call the children of the group one by one and activated or deactivate the grids. That is why I put these repeating methods as an own function `disableGridWithOptionalMessage`. I could also use a boolean to decide if I enable or disable the grid, but I liked it that way.

We also had the buttons and grid highlighted with `pointerover` and `pointerout`. As I also use that quite often, I thought about putting it in a `pointerEvent` function.

Whenever the player clicks on a certain slot row, they send a `dropDisc` with the picked `slotRow` to the server. Subsequently the server sends a `updateGameBoardVisually` event to all other clients. The `updateGameBoardVisually` event listener updates the grid with the dropped disc.

Solving Possible Errors

We are also looking for situations in which a player tries to put a disc, even though this certain row is already full. Before I check the vertical position of the disc through the server after the player decided for a slot. I also will initiate `slotCol` with -1, so I know if no free space was found. If this is the case, I will let the same player redo his turn once again, additionally mentioning that this slot is already full.

I was thinking what happens if the whole gameboard is used but still no winner is determined. That is why I am counting the total turns during this session. If it reaches 46 and no winner was determined, I am sure that the game ends with a draw and emit `gameOver` with the `playerID = 0`, which is interpreted as no player won.

Since only the server holds the gameboard and I didn't want to have another gameboard except in the `updateGameBoardVisually` function, I decided to declare it as a constant `GAMEBOARDLENGTH`.

Whenever the player clicks on replay, the server crashes or a player leaves, the whole game will be reset. There are no save states for these scenarios. Also, by clearing the gameboard whenever it gets updated, I can assure that both players will have the same game board during all times.

Encountered Problems

I was struggling to decide how I should implement the game. For example, I started with the update method or initiated the grid one by one before changing it and having all grids initiated in a loop, to lessen and clean the code.

Depending on the version of Socket.io or the declaration, I had different sets of methods available, which were not explained in the official Socket.io website. The site was even showing codes which I couldn't use, independently of the version. I had to look in Stackoverflow to find methods, which would be fitting to that particular function. Additionally some function weren't reliable, as for example they didn't update as fast as similar methods as for example `sockets.length` compared to `Object.keys(players).length`

There is also the `The AudioContext was not allowed to start` warning, which is apparently showing because Phaser is trying to create a audio context automatically when the game boots, while the Chrome browser is

blocking it Autoplay policy in Chrome. I also tried this in windows explorer, with the same warning. I am assuming that all current browsers are having similar behaviours.