**EVENTS TO EMIT and LISTEN FROM THE CLIENT SIDE (PUBLIC ROOM)**

**1. join\_game**

**Description**: **“joins a public room according to the gameType(3, 4, 5)”**

const **mainPlayer** = {

name: ‘playername’,

…other player properties (optional)

}

Once the player is ready and initialized, you need to emit the **join\_game** eventtogether with the mainPlayer details and also the type of the room.

*socket.emit(****‘join\_game’****, mainPlayer, gameType)*

gameType is a number and should be one of these values:

for 3x3 games: 3

for 4x4 game: 4

for 5x5 game: 5

**What the server does on receiving this event?**

Upon receiving this event the server creates a new player with the given player details. It also assigns the player a new/existing room. The server tries to find an existing room with empty slots depending on two/three/four player matches. If there is no proper room slot, then a new room is created appropriately.

If the player was created and a room was joined/created successfully, then the following events are emitted from the server that the client should listen to:

**a) player\_registered**

This event is fired after a successful **join\_game** event, and it returns the player details including important details like socketId, roomId, and turn.

socket.on(‘player\_registered’, player => {

/\*

player contains all the important details that you should store in the client end (update the player variable on the client end by adding new properties like socketId, roomId).

\*/

})

**b) player\_joined**

This event is broadcasted to all the players in the room whenever anyone joins the room including the player himself. You can listen to this event on the client end and provide real-time status like 2 out of 4 players joined.

Socket.on(**‘player\_joined’**, (players, playerJoined) => {

/\*

players is an array that are now currently in

the room. Use **players.length** to find total players.

**playerJoined** contains all the details of the player joining the game.

\*/

})

**c) game\_started**

Socket.on(**‘game\_started’**, room => {

})

This event is emitted when the room is filled and all the required players have joined. At this stage, the game is ready to be played. This event is emitted by the server to all the clients in that room.

You can use this event to initialize certain things in the game but don’t allow anyone to mark on the XYOT board until the “**turn**” event is received.

**d) turn**

This event is emitted by the server every time on a move of a player. The purpose of this event is to announce all the players whose move it is currently. When the game starts, the **turn** event is also emitted at that time.

You can implement it similar to the client side JS code below:

socket.on('turn', playerTurn => {

    // it will receive the player with that given turn...

    if (playerTurn.socketId === mainPlayer.socketId && playerTurn.turn) {

        mainPlayerTurn = true

        statusDisplay.innerHTML = 'Your turn...'

    } else {

        mainPlayerTurn = false

        statusDisplay.innerHTML = `Waiting for ${playerTurn.name}'s turn...`

    }

})

**2. play\_turn**

This event should be emitted from the client end when a player makes a mark on the board. Along with the event, also send the player details and the board position where it was marked.

Socket.emit(‘play\_turn’, mainPlayer, gridIndex)

Where grid index is one of

0 1 2

3 4 5

6 7 8

From the xyot board. For three/four players, it is the same logic. The events ‘turn’ and ‘turn\_played’ are emitted to the client once the server receives the ‘play\_turn’ event.

**a) turn\_played**

Once the play\_turn is processed successfully by the server, the server broadcasts the other players about the turn played along with the position where the mark was made by that player.

Socket.on(‘turn\_played’, (player, gridIndex) => {

/\*

other players will listen to this and update

the client UI. REMEMBER THAT ‘turn\_played’ EVENT will not be received by the player that made the mark on the board. It is broadcasted to other players only and not the player himself who made the turn. It is a little confusing to understand at first.

\*/

})

**b) game\_won**

Server sends this game won event when the game is won by a player.

Socket.on(‘game\_won’, winnerPlayer => {

})

**c) game\_draw**

Similar to game won event but called when the game is a draw.

Socket.on(‘game\_draw’, () => {

})

**d) game\_over**

Listen to the game over event to detect if the game is over.

socket.on('game\_over', (result, player) => {

    statusDisplay.textContent = `Game Over. ${result.message}`

    gameOver = true

})

**3. timeout**

**Description**: Runs every second on every tick of the timer. Every player is given 10 seconds to make a move within that time. If the timer reaches the 9th second and player doesn’t make a move then the robot does it.

Socket.on(‘timeout’, time => {

// will be called every second where time is 1-10 seconds.

// should be used to update/show the timer to the player.

// time is same for everyone in the room.

}

**4. player\_left**

This is broadcasted to all in the room when a player leaves the game. Only useful to alert other playes that the player has left the game and that robot will play on behalf of that player.

Socket.on(‘**player\_left’**, playerWhoLeft => {

/\* playerWhoLeft will contain all the properties of the player who left. \*/

}

**EVENTS TO EMIT FROM THE CLIENT SIDE (PRIVATE ROOM)**

Most of the events in public room like game\_start, turn\_played, etc. all work in private room too. There are only a few extras that is required for private rooms which are mentioned here.

**1. create\_room**

Emit this event from the client when a new room needs to be created.

Socket.emit(‘create\_room’, mainPlayer, gameType)

Where gameType should be one of 3, 4, or 5 for 3x3, 4x4, 5x5 board respectively.

**2. join\_room**

Emit this event from the client when the player wants to join a private room using the room code.

Socket.emit(‘join\_room’, mainPlayer, roomCode)

**Events to listen on the client side (private room)**

**1. room\_created**

Once the server creates the room, this event will run. You can listen to it similar to this:

Socket.on(‘room\_created’, (player, roomCode) => {

})

**2. room\_joined**

If a player joined the room successfully, then the room joined event is fired.

Socket.on(‘room\_joined, (player, roomId, room) => {

})

**Room API**

To find whether a room exists or to get a room details from the server using roomId, then Send a GET request to

GET <http://socketserver.xyot4.com/rooms/:roomId>

Where :roomId is the id of the room you wish to search for.

It will return a json response:

{

**message**: ‘room found’,

**room**: { … the room object with that roomId }

}

Useful to check if the roomId to join in private room exists and alert the player if it doesn’t exist.

If room is not found it sends a 404 JSON response

{

**message**: ‘room not found’

}

MESSAGE

When playing online match players will be able to send predefined messages to other players in the room.

Client Socket Message

socket.emit(‘message’, ‘message to broadcast’, playerWhoEmittedTheMessage)

socket.on(‘message’, broadcastedMessage, playerWhoEmittedTheMessage) => {})

Server Message event

socket.on(‘message’, (messageToBroadcast, playerWhoEmittedTheMessage) => {

socket.to(roomId).emit(‘message’, ‘message to others in the room’, playerWhoEmittedTheMessage)

})