Minijuegos con socket.io

Estructura previa de un multijugador.

Para cualquier **minijuego multijugador con Socket.IO**, incluso uno muy simple, es importante establecer una **base común de gestión de jugadores y comunicación** antes de implementar la lógica del juego.

1. Identificación básica del jugador

Antes de permitir interacción en el juego, cada cliente debe identificarse.

Recomendación mínima

- Al conectar, el cliente envía un **nombre o alias** elegido localmente.
- El servidor lo registra junto con su `socket.id`.

Cliente (JS):

```
const socket = io();
const playerName = prompt("Ingresa tu nombre:");
socket.emit("registerPlayer", playerName);
```

Servidor (Node.js):

```
const players = {};

io.on("connection", (socket) => {
  socket.on("registerPlayer", (name) => {
    players[socket.id] = { name, score: 0 };
    io.emit("playerList", Object.values(players));
  });

  socket.on("disconnect", () => {
    delete players[socket.id];
    io.emit("playerList", Object.values(players));
  });
});
```

Ventajas:

- Permite mostrar nombres en el ranking o tablero.
- Facilita la administración y comunicación (por ejemplo, avisar quién se unió o salió).

Opcional:

- Guardar un token o `sessionId` en `localStorage` para reconexiones.

2. Gestión de conexión / desconexión

Antes del juego, debes decidir **qué pasa cuando un jugador se conecta o desconecta**.

Buenas prácticas

- Enviar al cliente la lista de jugadores actuales (`playerList`) al entrar.
- Notificar a todos cuando alguien se conecta o se va.
- Opcionalmente, limpiar su estado (puntuación, posición, etc.) en el servidor.

Ejemplo de eventos:

```
// Servidor
io.emit("playerJoined", { name });
io.emit("playerLeft", { name });
```

Cliente:

```
socket.on("playerJoined", ({ name }) => showMessage(`${name} se ha
unido`));
socket.on("playerLeft", ({ name }) => showMessage(`${name} se ha ido`));
```

3. Sincronización inicial del estado del juego

Antes de empezar la partida (o cada ronda), todos los jugadores deben tener el mismo estado base.

Qué incluir

- Tiempo restante (si hay temporizador global).
- Marcadores o estado actual.
- Configuración de la partida (por ejemplo, dificultad o palabra actual).

Servidor:

```
const gameState = {
  round: 1,
  timer: 30,
  started: false,
  players: {}
};
```

Al conectar un nuevo jugador:

```
socket.emit("syncState", gameState);
```

4. Control del flujo del juego

Antes del primer evento de juego (`click`, `guess`, etc.), necesitas:

- Un **evento de inicio global** (`startGame` o `nextRound`).
- Lógica centralizada en el servidor para evitar trampas (el cliente no decide resultados).
- Broadcast del estado tras cada acción relevante (`updateScoreboard`, `gameOver`, etc.).

5. Estructura recomendada del servidor (mínima)

```
io.on("connection", (socket) => {
  // 1. Registro
  socket.on("registerPlayer", handleRegister);

  // 2. Eventos de juego
  socket.on("playerAction", handleAction);

  // 3. Sincronización
  socket.on("requestState", sendGameState);

  // 4. Desconexión
  socket.on("disconnect", handleDisconnect);
});
```

Esto separa responsabilidades y mantiene el servidor extensible.

6. Extras opcionales útiles

Función	Propósito	Implementación típica
Rate limiting	Evita spam de eventos	Ignorar eventos si llegan demasiado rápido
Reunión de jugadores en salas	Soporte para múltiples partidas simultáneas	`socket.join(roomName)`
Persistencia	Guardar puntajes entre rondas	Usar Redis o JSON temporal
Heartbeat/ping	Detectar desconexiones lentas	Intervalos de confirmación `ping`/`pong`

Ideas para juegos:

Debatir un poco y añadir las que se nos ocurran en clase.

Construcción Colaborativa (Pixel Art / Mural Global)

Concepto:
Todos los jugadores comparten un lienzo (píxeles o bloques). Cada jugador puede pintar o colocar un bloque cada X s.
El resultado es una obra colectiva.

Eventos Socket.IO:

- `placePixel` → jugador modifica una coordenada.
- `canvasUpdate` → sincroniza cambios con todos.
- `cooldown` → controla el intervalo entre acciones.

Extras:

- Guardado del mural final.
- Colores limitados por jugador o equipo.

Clicker Multiplayer

Concepto:
Todos contribuyen a una meta común (por ejemplo, acumular “energía”). Cada acción individual aumenta un contador global visible.

Eventos Socket.IO:

- `action` → jugador aporta puntos.
- `updateProgress` → broadcast con progreso global.
- `goalReached` → resetea y muestra estadísticas.

Extras:

- Cooperativo puro o con equipos.
- Efectos visuales tipo “barra de energía del mundo”.

Tap the Target (Objetivos Globales)

Concepto:
Aparecen objetivos en el tablero de todos; el primero en hacer clic suma puntos. Todos los clics se sincronizan globalmente.

Eventos Socket.IO:

- `spawnTarget` → servidor crea un nuevo objetivo.
- `hit` → jugador que hace clic primero.
- `scoreUpdate` → broadcast con ranking.

Extras:

- Varias zonas de aparición simultánea.
- Pérdida de puntos por clics erróneos.