TECNOLÓGICO NACIONAL DE MÉXICO
INSTITUTO TECNOLÓGICO DE TIJUANA

SUBDIRECCIÓN ACADÉMICA
DEPARTAMENTO DE SISTEMAS Y COMPUTACIÓN

SEMESTRE:
Agosto - Diciembre 2025

CARRERA:
Ingeniería en Sistemas Computacionales

MATERIA:
Patrones de diseño

TÍTULO ACTIVIDAD:
Examen unidad 4 y 5

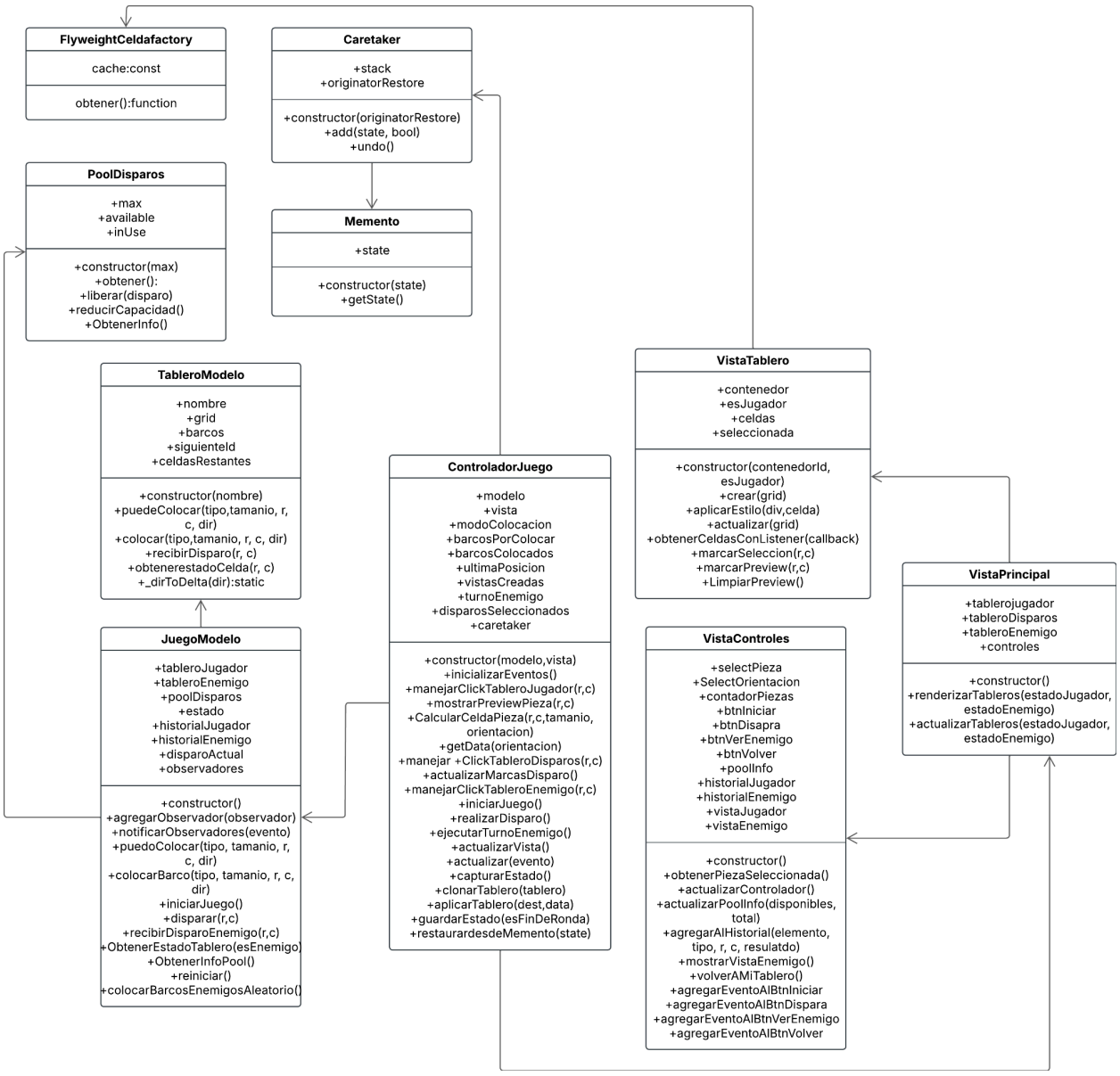
Alumno:
Hernandez Vazquez Cristian 21211964

NOMBRE DEL MAESTRO(A):
Maribel Guerrero Luis

Fecha

11/12/2025

UML


FlyweightCeldafactory
cache:const
obtener():function
Caretaker
+stack
+originatorRestore
+constructor(originatorRestore)
+add(state, bool)
+undo()
PoolDisparos
+max
+available
+inUse
+constructor(max)
+obtener():
+liberar(disparo)
+reducirCapacidad()
+ObtenerInfo()
Memento
+state
+constructor(state)
+getState()
TableroModelo
+nombre
+grid
+barcos
+siguienteId
+celdasRestantes
+constructor(nombre)
+puedeColocar(tipo,tamanio, r, c, dir)
+colocar(tipo,tamanio, r, c, dir)
+recibirDisparo(r, c)
+obtenerestadoCelda(r, c)
+_dirToDelta(dir):static
ControladorJuego
+modelo
+vista
+modoColocacion
+barcosPorColocar
+barcosColocados
+ultimaPosicion
+vistasCreadas
+turnoEnemigo
+disparosSeleccionados
+caretaker
+constructor(modelo,vista)
+inicializarEventos()
+manejarClickTableroJugador(r,c)
+mostrarPreviewPieza(r,c)
+CalcularCeldaPieza(r,c,tamanio, orientacion)
+getData(orientacion)
+manejar +ClickTableroDisparos(r,c)
+actualizarMarcasDisparo()
+manejarClickTableroEnemigo(r,c)
+iniciarJuego()
+realizarDisparo()
+ejecutarTurnoEnemigo()
+actualizarVista()
+actualizar(evento)
+capturarEstado()
+clonarTablero(tablero)
+aplicarTablero(dest,data)
+guardarEstado(esFinDeRonda)
+restaurardesdeMemento(state)
VistaTablero
+contenedor
+esJugador
+celdas
+seleccionada
+constructor(contenedorId, esJugador)
+crear(grid)
+aplicarEstilo(div,celda)
+actualizar(grid)
+obtenerCeldasConListener(callback)
+marcarSeleccion(r,c)
+marcarPreview(r,c)
+LimpiarPreview()
VistaPrincipal
+tablerojugador
+tableroDisparos
+tableroEnemigo
+controles
+constructor()
+renderizarTableros(estadoJugador, estadoEnemigo)
+actualizarTableros(estadoJugador, estadoEnemigo)
JuegoModelo
+tableroJugador
+tableroEnemigo
+poolDisparos
+estado
+historialJugador
+historialEnemigo
+disparoActual
+observadores
+constructor()
+agregarObservador(observador)
+notificarObservadores(evento)
+puedoColocar(tipo, tamanio, r, c, dir)
+colocarBarco(tipo, tamanio, r, c, dir)
+iniciarJuego()
+disparar(r,c)
+recibirDisparoEnemigo(r,c)
+ObtenerEstadoTablero(esEnemigo)
+ObtenerInfoPool()
+reiniciar()
+colocarBarcosEnemigosAleatorio()
VistaControles
+selectPieza
+SelectOrientacion
+contadorPiezas
+btnIniciar
+btnDisapra
+btnVerEnemigo
+btnVolver
+poolInfo
+historialJugador
+historialEnemigo
+vistaJugador
+vistaEnemigo
+constructor()
+obtenerPiezaSeleccionada()
+actualizarControlador()
+actualizarPoolInfo(disponibles, total)
+agregarAlHistorial(elemento, tipo, r, c, resulatdo)
+mostrarVistaEnemigo()
+volverAMiTablero()
+agregarEventoAlBtnIniciar
+agregarEventoAlBtnDispara
+agregarEventoAlBtnVerEnemigo
+agregarEventoAlBtnVolver

Código

```
const FlyweightCeldaFactory = (function () {
  const cache = {};

  function obtener(key) {
    if (cache[key]) return cache[key];
    const fly = {
      aplicar(el, estado) {
        el.className = 'celda';
        if (estado === 'buque') el.classList.add('barco');
        if (estado === 'submarino') el.classList.add('submarino');
        if (estado === 'hit') el.classList.add('hit');
        if (estado === 'miss') el.classList.add('miss');
        if (estado === 'seleccion') el.classList.add('seleccion');
      }
    };
    cache[key] = fly;
    return fly;
  }

  return { obtener };
})();

```

```
class Caretaker {
  constructor(originatorRestore) {
    this.stack = [];
    this.originatorRestore = originatorRestore;
  }

  add(state, esFinDeRonda = false) {
    this.stack.push(new Memento({ ...state, esFinDeRonda }));
  }

  undo() {
    if (this.stack.length <= 1) {
      alert('No hay más estados anteriores para deshacer.');
      return;
    }

    // Queremos encontrar el ANTERIOR fin de ronda
    // Empezamos desde length - 2 (antes del estado actual)
    let idx = this.stack.length - 2;

    // Buscar hacia atrás un estado marcado como fin de ronda
    while (idx >= 0) {
```

```
      while (idx >= 0) {
        const currentState = this.stack[idx].getState();
        if (currentState.esFinDeRonda) {
          // Encontramos un fin de ronda anterior, retroceder a este punto
          this.stack = this.stack.slice(0, idx + 1);
          this.originatorRestore(currentState);
          return;
        }
        idx--;
      }

      // Si no encontramos ningún fin de ronda (estamos en fase de colocación),
      // retroceder al estado anterior
      this.stack.pop();
      if (this.stack.length > 0) {
        const previous = this.stack[this.stack.length - 1];
        this.originatorRestore(previous.getState());
      } else {
        alert('No hay más estados anteriores para deshacer.');
      }
    }
  }

```

```
class Memento {
  constructor(state) {
    this.state = state;
  }

  getState() {
    return this.state;
  }
}

```

```
class PoolDisparos {
  constructor(max) {
    this.max = Math.max(0, max);
    this.available = [];
    this.inUse = new Set();
    for (let i = 0; i < this.max; i++) {
      this.available.push({ id: i, r: 0, c: 0, resultado: null });
    }
  }

  obtener() {
    if (this.available.length === 0) return null;
    const disparo = this.available.pop();
    this.inUse.add(disparo);
    return disparo;
  }

  liberar(disparo) {
    this.inUse.delete(disparo);
    if (this.available.length < this.max) {
      this.available.push(disparo);
    }
  }

  reducirCapacidad() {
    // Cuando el enemigo golpea, reducimos la capacidad máxima del pool
    if (this.max > 0) {
      this.max--;
    }
  }

  obtenerInfo() {
    return {
      disponibles: this.available.length,
      enUso: this.inUse.size,
      total: this.max
    };
  }
}
```

```
class ControladorJuego {
  constructor(modelo, vista) {
    this.modelo = modelo;
    this.vista = vista;

    // Estado del controlador
    this.modoColocacion = true;
    this.barcosPorColocar = { buque: 2, submarino: 1 };
    this.barcosColocados = { buque: 0, submarino: 0 };
    this.ultimaPosicion = null;
    this.vistasCreadas = false;
    this.turnoEnemigo = false;
    this.disparosSeleccionados = new Set(); // Set de coordenadas seleccionadas

    // Inicializar Caretaker para Memento
    this.caretaker = new Caretaker((state) => this.restaurarDesdeMemento(state));

    this.actualizarVista();
    this.inicializarEventos();
    this.guardarEstado(); // Estado inicial

    // Agregar listener para Ctrl+Z
    document.addEventListener('keydown', (e) => {
      if (e.ctrlKey && e.key.toLowerCase() === 'z') {
        e.preventDefault();
        this.caretaker.undo();
      }
    });
  }

  inicializarEventos() {
    // Eventos del tablero del jugador (colocación)
    this.vista.tablerojugador.obtenerCeldasConListener((r, c) => {
      this.manejarClickTableroJugador(r, c);
    });

    // Preview al pasar el mouse (solo en modo colocación)
    this.vista.tablerojugador.contenedor.addEventListener('mouseover', (e) => {
      if (!this.modoColocacion) return;

      const celda = e.target.closest('[data-r][data-c]');
      if (celda) {
        const r = parseInt(celda.dataset.r);
        const c = parseInt(celda.dataset.c);
        this.mostrarPreviewPieza(r, c);
      }
    });

```

```
50      this.vista.tablerojugador.contenedor.addEventListener('mouseout', () => {
51        if (this.modoColocacion) {
52          this.vista.tablerojugador.limpiarPreview();
53        }
54      });
55
56      // Eventos del tablero de disparos
57      this.vista.tableroDisparos.obtenerCeldasConListener((r, c) => {
58        this.manejarClickTableroDisparos(r, c);
59      });
60
61      // Eventos del tablero enemigo (vista)
62      this.vista.tableroEnemigo.obtenerCeldasConListener((r, c) => {
63        this.manejarClickTableroEnemigo(r, c);
64      });
65
66      // Eventos de controles
67      this.vista.controles.agregarEventoAlBtnIniciar(() => this.iniciarJuego());
68      this.vista.controles.agregarEventoAlBtnDispara(() => this.realizarDisparo());
69      this.vista.controles.agregarEventoAlBtnVerEnemigo(() => this.vista.controles.mostrarVista
70      this.vista.controles.agregarEventoAlBtnVolver(() => this.vista.controles.volverAMiTablero
71
72      // Observador del modelo
```

```
73        this.modelo.agregarObservador(this);
74      }
75
76      manejarClickTableroJugador(r, c) {
77        if (!this.modoColocacion) return;
78
79        const { tipo, orientacion } = this.vista.controles.obtenerPiezaSeleccionada();
80        const tamanios = { buque: 2, submarino: 3 };
81        const tamanio = tamanios[tipo];
82
83        // Verificar si aún se pueden colocar barcos de este tipo
84        if (this.barcosColocados[tipo] >= this.barcosPorColocar[tipo]) {
85          alert(`Ya colocaste todos los ${tipo}s`);
86          return;
87        }
88
89        // Intentar colocar el barco inmediatamente
90        if (this.modelo.puedeColocar(tipo, tamanio, r, c, orientacion)) {
91          if (this.modelo.colocarBarco(tipo, tamanio, r, c, orientacion)) {
92            this.barcosColocados[tipo]++;
93            this.guardarEstado(); // Memento
94            this.actualizarVista();
95
```

```
96          // Verificar si completó la colocación
97          const totalColocados = this.barcosColocados.buque + this.barcosColocados.submarino;
98          const totalRequerido = this.barcosPorColocar.buque + this.barcosPorColocar.submarino;
99
100         if (totalColocados === totalRequerido) {
101           this.vista.controles.habilitarBotones(true, false);
102           alert('Has colocado todas las piezas. Presiona "Iniciar Juego" para comenzar.');
103         }
104       }
105     } else {
106       alert('No se puede colocar allí (fuera de rango o sobreposición).');
107     }
108   }
109
110   mostrarPreviewPieza(r, c) {
111     const { tipo, orientacion } = this.vista.controles.obtenerPiezaSeleccionada();
112     const tamanios = { buque: 2, submarino: 3 };
113     const tamanio = tamanios[tipo];
114
115     // Verificar si se puede colocar
116     if (this.barcosColocados[tipo] >= this.barcosPorColocar[tipo]) {
117       this.vista.tablerojugador.limpiarPreview();
118       return;
```

```
119     }
120
121     const celdas = this.calcularCeldasPieza(r, c, tamanio, orientacion);
122     const valido = this.modelo.puedeColocar(tipo, tamanio, r, c, orientacion);
123     this.vista.tablerojugador.marcarPreview(celdas, valido);
124   }
125
126   calcularCeldasPieza(r, c, tamanio, orientacion) {
127     const celdas = [];
128     const delta = this.getDelta(orientacion);
129     for (let i = 0; i < tamanio; i++) {
130       const rr = r + delta.dr * i;
131       const cc = c + delta.dc * i;
132       celdas.push({ r: rr, c: cc });
133     }
134     return celdas;
135   }
136
137   getDelta(orientacion) {
138     switch (orientacion) {
139       case 'horizontal': return { dr: 0, dc: 1 };
140       case 'vertical': return { dr: 1, dc: 0 };
141       case 'diag-dr': return { dr: 1, dc: 1 };
```

```
      case 'diag-ur': return { dr: -1, dc: 1 };
      default: return { dr: 0, dc: 1 };
    }
  }

  manejarClickTableroDisparos(r, c) {
    if (!this.modoColocacion && this.modelo.estado === 'jugando' && !this.turnoEnemigo) {
      const key = `${r},${c}`;

      // Verificar si ya fue disparado
      if (this.modelo.tableroEnemigo.grid[r][c].disparado) {
        alert('Ya disparaste en esa casilla');
        return;
      }

      // Verificar disparos disponibles
      const infoPool = this.modelo.obtenerInfoPool();
      if (infoPool.disponibles <= 0) {
        alert('No tienes disparos disponibles');
        return;
      }

      // Alternar selección
      if (this.disparosSeleccionados.has(key)) {
```

```
        this.disparosSeleccionados.delete(key);
        this.vista.tableroDisparos.marcarSeleccion(-1, -1);
      } else {
        // Limitar a los disparos realmente disponibles
        const disparosPendientes = infoPool.disponibles - this.disparosSeleccionados.size;
        if (disparosPendientes <= 0) {
          alert(`Solo puedes seleccionar ${infoPool.disponibles} casillas`);
          return;
        }
        this.disparosSeleccionados.add(key);
      }

      // Actualizar UI
      this.actualizarMarcasDisparos();
      this.actualizarVista();
    }
  }

  actualizarMarcasDisparos() {
    // Verificar que existan celdas
    if (!this.vista.tableroDisparos || !this.vista.tableroDisparos.celdas || this.vista.tableroDisparos.celdas.length === 0) {
      console.warn('tableroDisparos o celdas no disponibles');
      return;
    }
```

```
190
191     console.log('Actualizando marcas. Disparos seleccionados:', Array.from(this.disparosSeleccionados));
192
193     // Limpiar marcas previas
194     this.vista.tableroDisparos.celdas.forEach(c => {
195       if (c) c.classList.remove('disparar-seleccionado');
196     });
197
198     // Marcar disparos seleccionados con efecto visual
199     this.disparosSeleccionados.forEach(key => {
200       const [r, c] = key.split(',').map(Number);
201       const idx = r * 10 + c;
202       if (idx >= 0 && idx < this.vista.tableroDisparos.celdas.length) {
203         const celda = this.vista.tableroDisparos.celdas[idx];
204         if (celda) {
205           celda.classList.add('disparar-seleccionado');
206           console.log(`Marcada celda [${r},${c}] con disparar-seleccionado`);
207         }
208       }
209     });
210   }
211
212   manejarClickTableroEnemigo(r, c) {
213     if (!this.modoColocacion && this.modelo.estado === 'jugando' && !this.turnoEnemigo) {
```

```
212   manejarClickTableroEnemigo(r, c) {
214       // Redirigir al tablero de disparos
215       this.manejarClickTableroDisparos(r, c);
216     }
217   }
218
219   iniciarJuego() {
220     const totalColocados = this.barcosColocados.buque + this.barcosColocados.submarino;
221     const totalRequerido = this.barcosPorColocar.buque + this.barcosPorColocar.submarino;
222
223     if (totalColocados < totalRequerido) {
224       alert('Debes colocar todas tus piezas antes de iniciar.');
225       return;
226     }
227
228     this.modoColocacion = false;
229     // Colocar barcos enemigos aleatoriamente
230     this.modelo.colocarBarcosEnemigoAleatorio();
231     if (this.modelo.iniciarJuego()) {
232       this.vista.controles.habilitarBotones(false, true);
233       this.guardarEstado(true); // Marcar fin de fase de colocación
234       this.actualizarVista();
235     }
236   }
237
```

```
238   realizarDisparo() {
239     if (this.modelo.estado !== 'jugando' || this.turnoEnemigo) return;
240
241     if (this.disparosSeleccionados.size === 0) {
242       alert('Selecciona al menos una casilla para disparar');
243       return;
244     }
245
246     const resultados = [];
247     const orden = Array.from(this.disparosSeleccionados).sort();
248
249     // Disparar en todas las casillas seleccionadas
250     orden.forEach(key => {
251       const [r, c] = key.split(',').map(Number);
252       const resultado = this.modelo.disparar(r, c);
253
254       let mensaje = `(${r + 1},${c + 1}) - `;
255       if (resultado.estado === 'fallo') {
256         mensaje += 'Fallo';
257       } else if (resultado.estado === 'impacto') {
258         mensaje += `¡Impacto en ${resultado.tipo}!`;
259       } else if (resultado.estado === 'hundido') {
260         mensaje += `¡${resultado.tipo} hundido!`;
261       }
262       resultados.push(mensaje);
```

```
263       this.vista.controles.agregarAlHistorial('historialJugador', 'Disparo', r, c, resultado.estado);
264     });
265
266     this.disparosSeleccionados.clear();
267     this.actualizarVista();
268     alert('Tus disparos:\n' + resultados.join('\n'));
269
270     // Verificar si ganó
271     if (this.modelo.estado === 'fin') {
272       alert('¡Has ganado!');
273       this.guardarEstado(true);
274       return;
275     }
276
277     // Turno del enemigo
278     this.turnoEnemigo = true;
279     setTimeout(() => this.ejecutarTurnoEnemigo(), 1000);
280   }
281
282   ejecutarTurnoEnemigo() {
283     alert('Turno del enemigo');
284
285     // El enemigo dispara tantas veces como celdas le queden ocupadas
```

```
286     const disparosEnemigo = this.modelo.tableroEnemigo.celdasRestantes;
287     const mensajes = [];
288
289     // Obtener celdas no disparadas
290     const celdasDisponibles = [];
291     for (let r = 0; r < 10; r++) {
292       for (let c = 0; c < 10; c++) {
293         const celda = this.modelo.tableroJugador.grid[r][c];
294         if (!celda.disparado) {
295           celdasDisponibles.push({ r, c });
296         }
297       }
298     }
299
300     // Mezclar aleatoriamente
301     for (let i = celdasDisponibles.length - 1; i > 0; i--) {
302       const j = Math.floor(Math.random() * (i + 1));
303       [celdasDisponibles[i], celdasDisponibles[j]] = [celdasDisponibles[j], celdasDisponibles[i]];
304     }
305
306     // Disparar y contar impactos
307     let impactosEnemigo = 0;
308     for (let i = 0; i < Math.min(disparosEnemigo, celdasDisponibles.length); i++) {
309       const { r, c } = celdasDisponibles[i];
```

```
310       const resultado = this.modelo.recibirDisparoEnemigo(r, c);
311
312       // Si fue impacto, reduce los disparos del jugador
313       if (resultado.estado === 'impacto' || resultado.estado === 'hundido') {
314         impactosEnemigo++;
315         // Reducir la capacidad máxima del pool de disparos del jugador
316         this.modelo.poolDisparos.reducirCapacidad();
317       }
318
319       let mensaje = `(${r + 1},${c + 1}) - `;
320       if (resultado.estado === 'fallo') {
321         mensaje += 'Fallo';
322       } else if (resultado.estado === 'impacto') {
323         mensaje += `¡Impacto en tu ${resultado.tipo}!`;
324       } else if (resultado.estado === 'hundido') {
325         mensaje += `¡Hundió tu ${resultado.tipo}!`;
326       }
327
328       mensajes.push(mensaje);
329       this.vista.controles.agregarAlHistorial('historialEnemigo', 'Disparo enemigo', r, c, resultado.estado);
330     }
331
332     this.actualizarVista();
333     alert('Disparos del enemigo:\n' + mensajes.join('\n'));
```

```
334
335     // Verificar si perdiste
336     if (this.modelo.estado === 'fin') {
337       alert('¡Has perdido!');
338       this.guardarEstado(true);
339       return;
340     }
341
342     // Devolver turno al jugador
343     this.turnoEnemigo = false;
344     // GUARDAR ESTADO FIN DE RONDA: después de que jugador y enemigo han disparado
345     this.guardarEstado(true);
346     const infoPool = this.modelo.obtenerInfoPool();
347     alert(`Es tu turno nuevamente. Tienes ${infoPool.total}/${infoPool.total} disparos disponibles`);
348   }
349
350   actualizarVista() {
351     const estadoJugador = this.modelo.obtenerEstadoTablero(false);
352     const estadoEnemigo = this.modelo.obtenerEstadoTablero(true);
353
354     // Renderizar tableros solo la primera vez
355     if (!this.vistasCreadas) {
356       this.vista.renderizarTableros(estadoJugador, estadoEnemigo);
357       this.vistasCreadas = true;
```

```
358     } else {
359       this.vista.actualizarTableros(estadoJugador, estadoEnemigo);
360     }
361
362     // Actualizar contador de piezas
363     const barcosFaltantes = this.barcosPorColocar.buque - this.barcosColocados.buque;
364     const submarinosFaltantes = this.barcosPorColocar.submarino - this.barcosColocados.submarino;
365     this.vista.controles.actualizarContador(barcosFaltantes, submarinosFaltantes);
366
367     // Actualizar info del pool
368     const infoPool = this.modelo.obtenerInfoPool();
369     // disparosDisponibles = total - seleccionados (lo que queda por usar en esta ronda)
370     // totalDisparosActual = capacidad máxima (disminuye cuando enemigo golpea)
371     const disparosDisponibles = infoPool.total - this.disparosSeleccionados.size;
372     const totalDisparosActual = infoPool.total;
373     this.vista.controles.actualizarPoolInfo(disparosDisponibles, totalDisparosActual);
374
375     // Habilitar botones según estado
376     if (this.modoColocacion) {
377       this.vista.controles.habilitarBotones(true, false);
378     } else {
379       this.vista.controles.habilitarBotones(false, true);
380     }
381
```

```
381
382     // RE-APLICAR MARCAS VISUALES A CASILLAS SELECCIONADAS
383     this.actualizarMarcasDisparos();
384   }
385
386   // Implementar interfaz de Observador
387   actualizar(evento) {
388     console.log('Evento del modelo:', evento);
389     switch (evento.tipo) {
390       case 'barco_colocado':
391         this.actualizarVista();
392         break;
393       case 'juego_iniciado':
394         alert('El juego ha iniciado. Selecciona tus casillas en la tabla de disparos para disparar');
395         break;
396       case 'disparo_realizado':
397         this.actualizarVista();
398         break;
399       case 'juego_finalizado':
400         const mensajeFinal = evento.data.ganador === 'Jugador' ? '¡Has ganado!' : '¡Has perdido!';
401         alert(mensajeFinal);
402         break;
403       case 'juego_reiniciado':
404         this.modoColocacion = true;
```

```
405           this.barcosColocados = { buque: 0, submarino: 0 };
406           this.actualizarVista();
407           break;
408       }
409     }
410
411     // Métodos para patrón Memento
412     capturarEstado() {
413       return {
414         modelo: {
415           tableroJugador: this.clonarTablero(this.modelo.tableroJugador),
416           tableroEnemigo: this.clonarTablero(this.modelo.tableroEnemigo),
417           estado: this.modelo.estado,
418           historialJugador: [...this.modelo.historialJugador],
419           historialEnemigo: [...this.modelo.historialEnemigo],
420           poolMax: this.modelo.poolDisparos.max,
421           poolDisponibles: this.modelo.poolDisparos.obtenerInfo().disponibles
422         },
423         controller: {
424           modoColocacion: this.modoColocacion,
425           barcosColocados: { ...this.barcosColocados },
426           turnoEnemigo: this.turnoEnemigo,
427           disparosSeleccionados: Array.from(this.disparosSeleccionados)
428         }
```

```
    };
  }

  clonarTablero(tablero) {
    return {
      nombre: tablero.nombre,
      grid: tablero.grid.map(fila => fila.map(cell => ({ ...cell }))),
      barcos: tablero.barcos.map(b => ({ ...b, celdas: b.celdas.map(c => ({ ...c })) })),
      siguenteId: tablero.siguenteId,
      celdasRestantes: tablero.celdasRestantes
    };
  }

  aplicarTablero(dest, data) {
    dest.nombre = data.nombre;
    dest.grid = data.grid.map(fila => fila.map(cell => ({ ...cell })));
    dest.barcos = data.barcos.map(b => ({ ...b, celdas: b.celdas.map(c => ({ ...c })) }));
    dest.siguenteId = data.siguenteId;
    dest.celdasRestantes = data.celdasRestantes;
  }

  guardarEstado(esFinDeRonda = false) {
    this.caretaker.add(this.capturarEstado(), esFinDeRonda);
  }
```

```

  restaurarDesdeMemento(state) {
    this.aplicarTablero(this.modelo.tableroJugador, state.modelo.tableroJugador);
    this.aplicarTablero(this.modelo.tableroEnemigo, state.modelo.tableroEnemigo);
    this.modelo.estado = state.modelo.estado;
    this.modelo.historialJugador = [...state.modelo.historialJugador];
    this.modelo.historialEnemigo = [...state.modelo.historialEnemigo];

    // Restaurar pool
    if (state.modelo.poolMax !== undefined) {
      this.modelo.poolDisparos = new PoolDisparos(state.modelo.poolMax);
      // Simular disparos usados
      const disparosUsados = state.modelo.poolMax - state.modelo.poolDisponibles;
      for (let i = 0; i < disparosUsados; i++) {
        this.modelo.poolDisparos.obtener();
      }
    }

    this.modoColocacion = state.controller.modoColocacion;
    this.barcosColocados = { ...state.controller.barcosColocados };
    this.turnoEnemigo = state.controller.turnoEnemigo || false;
    this.disparosSeleccionados.clear();
    if (state.controller.disparosSeleccionados) {
      state.controller.disparosSeleccionados.forEach(key => this.disparosSeleccionados.add(key));
    }
```

```
478         this.actualizarVista();
479         this.actualizarMarcasDisparos();
480       }
481     }
482
```

```
2   document.addEventListener('DOMContentLoaded', () => {
3     // Crear instancias del patrón MVC
4     const modelo = new JuegoModelo();
5     const vista = new VistaPrincipal();
6     const controlador = new ControladorJuego(modelo, vista);
7
8     // El controlador ya maneja toda la interacción
9     console.log(' Juego Battleship - Patrón MVC iniciado');
10  });
11
```

```
2   const TAMAÑO = 10;
3
4   /**
5    * TableroModelo - Gestiona la lógica del tablero
6    */
7   class TableroModelo {
8     constructor(nombre) {
9       this.nombre = nombre;
10      this.grid = Array.from({ length: TAMAÑO }, () =>
11        Array.from({ length: TAMAÑO }, () => ({
12          tiene: null,
13          barcoId: null,
14          disparado: false
15        }))
16      );
17      this.barcos = [];
18      this.siguenteId = 1;
19      this.celdasRestantes = 0;
20    }
21
22    puedeColocar(tipo, tamanio, r, c, dir) {
23      const pasos = [];
24      const delta = TableroModelo._dirToDelta(dir);
25      for (let i = 0; i < tamanio; i++) {
26        const rr = r + delta.dr * i;
```

```
      const cc = c + delta.dc * i;
      if (rr < 0 || rr >= TAMAÑO || cc < 0 || cc >= TAMAÑO) return false;
      if (this.grid[rr][cc].tiene) return false;
      pasos.push({ r: rr, c: cc });
    }
    return true;
  }

  colocar(tipo, tamanio, r, c, dir) {
    if (!this.puedeColocar(tipo, tamanio, r, c, dir)) return false;
    const id = this.siguenteId++;
    const delta = TableroModelo._dirToDelta(dir);
    const celdas = [];
    for (let i = 0; i < tamanio; i++) {
      const rr = r + delta.dr * i;
      const cc = c + delta.dc * i;
      this.grid[rr][cc].tiene = tipo;
      this.grid[rr][cc].barcoId = id;
      celdas.push({ r: rr, c: cc });
    }
    this.barcos.push({
      id,
      tipo,
      tamanio,
```

```
      celdas,
      hits: 0,
      hundido: false
    });
    this.celdasRestantes += tamanio;
    return true;
  }

  recibirDisparo(r, c) {
    const cell = this.grid[r][c];
    if (cell.disparado) return { estado: 'repetido' };
    cell.disparado = true;
    if (cell.tiene) {
      const barco = this.barcos.find(b => b.id === cell.barcoId);
      barco.hits++;
      this.celdasRestantes--;
      if (barco.hits >= barco.tamanio) {
        barco.hundido = true;
        return { estado: 'hundido', tipo: barco.tipo };
      }
      return { estado: 'impacto', tipo: barco.tipo };
    }
    return { estado: 'fallo' };
  }
```

```
75
76      obtenerEstadoCelda(r, c) {
77        return this.grid[r][c];
78      }
79
80      static _dirToDelta(dir) {
81        switch (dir) {
82          case 'horizontal': return { dr: 0, dc: 1 };
83          case 'vertical': return { dr: 1, dc: 0 };
84          case 'diag-dr': return { dr: 1, dc: 1 };
85          case 'diag-ur': return { dr: -1, dc: 1 };
86        }
87        return { dr: 0, dc: 1 };
88      }
89    }
90
91    /**
92     * JuegoModelo - Orquesta el estado general del juego
93     */
94
95    ////////////////////////////////////
96    class JuegoModelo {
97      constructor() {
98        this.tableroJugador = new TableroModelo('Jugador');
99        this.tableroEnemigo = new TableroModelo('Enemigo');
```

```
    this.poolDisparos = new PoolDisparos(0);
    this.estado = 'colocacion'; // colocacion, jugando, fin
    this.historialJugador = [];
    this.historialEnemigo = [];
    this.disparoActual = null;
    this.observadores = [];
  }

  agregarObservador(observador) {
    this.observadores.push(observador);
  }

  notificarObservadores(evento) {
    this.observadores.forEach(obs => obs.actualizar(evento));
  }

  puedeColocar(tipo, tamanio, r, c, dir) {
    return this.tableroJugador.puedeColocar(tipo, tamanio, r, c, dir);
  }

  colocarBarco(tipo, tamanio, r, c, dir) {
    const resultado = this.tableroJugador.colocar(tipo, tamanio, r, c, dir);
    if (resultado) {
      this.notificarObservadores({ tipo: 'barco_colocado', data: { tipo, r, c } });
```

```
124       }
125       return resultado;
126     }
127   ////////////////////////////////////////
128     iniciarJuego() {
129       if (this.tableroJugador.barcos.length === 0) return false;
130       this.estado = 'jugando';
131       // Crear pool basado en celdas ocupadas del jugador
132       this.poolDisparos = new PoolDisparos(this.tableroJugador.celdasRestantes);
133       this.notificarObservadores({ tipo: 'juego_iniciado' });
134       return true;
135     }
136
137     ////////////////////////////////////Object pool
138     disparar(r, c) {
139       const resultado = this.tableroEnemigo.recibirDisparo(r, c);
140       const disparo = this.poolDisparos.obtener();
141
142       if (disparo) {
143         disparo.r = r;
144         disparo.c = c;
145         disparo.resultado = resultado.estado;
146         this.historialJugador.push(disparo);
147
```

```
148         // El jugador siempre recupera el disparo, sin importar si acierta o falla
149         this.poolDisparos.liberar(disparo);
150       }
151
152       this.notificarObservadores({
153         tipo: 'disparo_realizado',
154         data: { r, c, resultado }
155       });
156
157       if (this.tableroEnemigo.celdasRestantes === 0) {
158         this.estado = 'fin';
159       }
160
161       return resultado;
162     }
163
164     recibirDisparoEnemigo(r, c) {
165       const resultado = this.tableroJugador.recibirDisparo(r, c);
166
167       this.notificarObservadores({
168         tipo: 'disparo_enemigo',
169         data: { r, c, resultado }
170       });
171
```

```
172     if (this.tableroJugador.celdasRestantes === 0) {
173       this.estado = 'fin';
174     }
175
176     return resultado;
177   }
178
179   obtenerEstadoTablero(esEnemigo = false) {
180     const tablero = esEnemigo ? this.tableroEnemigo : this.tableroJugador;
181     return {
182       grid: tablero.grid,
183       barcos: tablero.barcos,
184       celdasRestantes: tablero.celdasRestantes
185     };
186   }
187
188   obtenerInfoPool() {
189     return this.poolDisparos.obtenerInfo();
190   }
191
192   reiniciar() {
193     this.tableroJugador = new TableroModelo('Jugador');
194     this.tableroEnemigo = new TableroModelo('Enemigo');
195     this.estado = 'colocacion';
```

```
196     this.historialJugador = [];
197     this.historialEnemigo = [];
198     this.notificarObservadores({ tipo: 'juego_reiniciado' });
199   }
200
201   colocarBarcosEnemigoAleatorio() {
202     const colocarTipo = (tipo, tam, intentosMax = 200) => {
203       let intentos = 0;
204       while (intentos < intentosMax) {
205         const r = Math.floor(Math.random() * TAMAÑO);
206         const c = Math.floor(Math.random() * TAMAÑO);
207         const dirs = ['horizontal', 'vertical', 'diag-dr', 'diag-ur'];
208         const dir = dirs[Math.floor(Math.random() * dirs.length)];
209         if (this.tableroEnemigo.puedeColocar(tipo, tam, r, c, dir)) {
210           this.tableroEnemigo.colocar(tipo, tam, r, c, dir);
211           return true;
212         }
213         intentos++;
214       }
215       return false;
216     };
217     colocarTipo('buque', 2);
218     colocarTipo('buque', 2);
219     colocarTipo('submarino', 3);
```

```
219     colocarTipo('submarino', 3);
220   }
221 }
222
```

```html
<!doctype html>
<html lang="es">
<head>
  <meta charset="utf-8" />
  <meta name="viewport" content="width=device-width,initial-scale=1" />
  <title>Battleship</title>
  <link rel="stylesheet" href="Styles.CSS" />
</head>
<body>
  <header>
    <h1>Battleship</h1>
  </header>

  <section id="contenedor-controles">
    <div class="control">
      <label>Seleccionar pieza:</label>
      <select id="select-pieza">
        <option value="buque">Buque (2)</option>
        <option value="submarino">Submarino (3)</option>
      </select>
    </div>
    <div class="control">
      <label>Orientación:</label>
      <select id="select-orientacion">
        <option value="horizontal">Horizontal →</option>
        <option value="vertical">Vertical ↓</option>
```

```
27            <option value="diag-dr">Diagonal ↘</option>
28            <option value="diag-ur">Diagonal ↗</option>
29          </select>
30        </div>
31        <div class="control">
32          <label>Piezas restantes:</label>
33          <span id="contador-piezas">Buques: 2 | Submarinos: 1</span>
34        </div>
35        <div class="control">
36          <button id="btn-iniciar" disabled>Iniciar Juego</button>
37          <button id="btn-ver-enemigo">Ver Tablero Enemigo</button>
38          <button id="btn-volver" style="display:none">Volver a mi tablero</button>
39        </div>
40      </section>
41
42      <main id="principal">
43        <section id="vista-jugador" class="vista">
44          <h2>Mi Tablero</h2>
45          <div class="tableros">
46            <div>
47              <div class="panel">
```

```
48                <h3>Tablero de Piezas</h3>
49                <div id="tablero-jugador" class="tablero"></div>
50              </div>
51            </div>
52            <div>
53              <div class="panel">
54                <h3>Tablero de Disparos</h3>
55                <div id="tablero-disparos" class="tablero"></div>
56                <div class="acciones">
57                  <span id="pool-info">Disparos disponibles: 0</span>
58                  <button id="btn-disparar" disabled>Disparar</button>
59                </div>
60              </div>
61            </div>
62          </div>
63          <div class="historial">
64            <h4>Historial (mis disparos)</h4>
65            <ul id="historial-jugador"></ul>
66          </div>
67        </section>
68
```

```
    <section id="vista-enemigo" class="vista" style="display:none">
      <h2>Tablero Enemigo (Vista)</h2>
      <div class="tableros">
        <div>
          <div class="panel">
            <h3>Tablero Enemigo</h3>
            <div id="tablero-enemigo" class="tablero"></div>
          </div>
        </div>
        <div>
          <div class="panel">
            <h3>Historial (disparos del enemigo)</h3>
            <ul id="historial-enemigo"></ul>
          </div>
        </div>
      </div>
    </section>
  </main>

  <footer>
    <small>Reglas: Coloca 2 Buques (2) y 1 Submarino (3). Orientaciones permitidas. Usa alertas para turnos y resultad
  </footer>
```

```
    <!-- Patrones de Diseño -->
    <script src="Memento/Memento.js"></script>
    <script src="Memento/Caretaker.js"></script>
    <script src="ObjectPool/PoolDisparos.js"></script>
    <script src="Flyweight/FlyweightCeldaFactory.js"></script>

    <!-- Arquitectura MVC -->
    <script src="Model.js"></script>
    <script src="View.js"></script>
    <script src="Controller.js"></script>
    <script src="Main.js"></script>
  </body>
  </html>
```

```css
:root{
  --celda-size:28px;
  --gap:8px;
}
*
{
  box-sizing:border-box;
  font-family:Segoe UI, Tahoma, sans-serif
}
body
{
  padding:16px;background:#f6f8fa;
  color:#122
}
header h1
{
  margin:0 0 12px
}
#contenedor-controles
{display:flex;
  flex-wrap:wrap;
  gap:12px;
  margin-bottom:12px
}
.control
{
```

```
    display:flex;align-items:center;gap:6px
}
.tableros
{
    display:flex;gap:20px;flex-wrap:wrap
}
.panel
{
    background: white;
    padding:8px;
    border-radius:6px;
    border:1px solid  #ddd;
    box-shadow:0 2px 4px rgba(0,0,0,0.1);
}
.tablero
{
    display:inline-grid;
    grid-template-columns:repeat(10, var(--celda-size));
    grid-template-rows:repeat(10,var(--celda-size));
    gap:2px;
    padding:6px;
}
.tablero.label-grid
{
    grid-template-columns: 24px repeat(10, var(--celda-size));
```

```
    grid-template-rows: 18px repeat(10,var(--celda-size));
  }
  .label
  {
    display:flex;
    align-items:center;
    justify-content:center;
    font-size:11px;
    color:#334;padding:2px
  }
  .celda
  {
    width:var(--celda-size);
    height:var(--celda-size);
    background:#e9eef3;
    border-radius:3px;
    display:flex;
    align-items:center;
    justify-content:center;
    cursor:pointer;
    font-size:11px
  }
  .celda:hover
  {
    outline:2px solid #bcd
```

```
77  }
78  .celda.barco
79  {
80    background: #a0c4ff
81  }
82  .celda.submarino
83  {
84    background: #8ee6b0
85  }
86  .celda.hit
87  {
88    background: #ff6b6b;
89    color: white
90  }
91  .celda.miss
92  {
93    background: #94a3b8;
94    color: white
95  }
96  .celda.seleccion
97  {
98    outline:3px solid #ffd166
99  }
100 .celda.disparar-seleccionado
101 {
```

```
    background: #e6f337 !important;
    outline: 2px solid #adb80b !important;
    box-shadow: 0 0 6px rgba(173, 184, 11, 0.4) !important;
    border-radius: 3px !important;
  }
  .celda.preview-invalido
  {
    outline:3px solid #ff6b6b;
    background:#ffcccc
  }
  .acciones
  {margin-top:6px;
    display:flex;
    gap:8px;
    align-items:center
  }
  .historial
  {
    margin-top:12px
  }
  ul
  {
    padding-left:18px
  }
  button
```

```
  {
    padding:6px 10px;
    border-radius:4px;
    border:1px solid #888;
    background:#fff;
    cursor:pointer
  }
  button:disabled
  {
    opacity:0.5;
    cursor:not-allowed
  }
  footer
  {
    margin-top:14px;
    color:#445
  }

```

```
class VistaTablero {
  constructor(contenedorId, esJugador = true) {
    this.contenedor = document.getElementById(contenedorId);
    this.esJugador = esJugador;
    this.celdas = [];
    this.seleccionada = null;
  }

  crear(grid) {
    this.contenedor.innerHTML = '';
    this.celdas = [];

    // Agregar clase para grid con etiquetas
    this.contenedor.classList.add('label-grid');

    // Crear etiquetas de columnas (X: 1-10)
    const labelVacio = document.createElement('div');
    labelVacio.classList.add('label');
    this.contenedor.appendChild(labelVacio);

    for (let c = 0; c < TAMAÑO; c++) {
      const labelCol = document.createElement('div');
      labelCol.classList.add('label');
      labelCol.textContent = (c + 1);
      this.contenedor.appendChild(labelCol);
```

```
27      }
28
29      // Crear filas con etiquetas (Y: 1-10)
30      grid.forEach((fila, r) => {
31        // Etiqueta de fila (Y)
32        const labelFila = document.createElement('div');
33        labelFila.classList.add('label');
34        labelFila.textContent = (r + 1);
35        this.contenedor.appendChild(labelFila);
36
37        // Celdas
38        fila.forEach((celda, c) => {
39          const div = document.createElement('div');
40          div.dataset.r = r;
41          div.dataset.c = c;
42          this.aplicarEstilo(div, celda);
43          this.contenedor.appendChild(div);
44          this.celdas.push(div);
45        });
46      });
47    }
48  ////////////////////////////////////////////////////////////////////////////
49    aplicarEstilo(div, celda) {
50      let estado = 'vacio';
```

```
51      if (this.esJugador && celda.tiene) {
52        estado = celda.tiene;
53      }
54      if (celda.disparado) {
55        estado = celda.tiene ? 'hit' : 'miss';
56      }
57
58      const fly = FlyweightCeldaFactory.obtener(estado);
59      fly.aplicar(div, estado);
60    }
61
62    actualizar(grid) {
63      grid.forEach((fila, r) => {
64        fila.forEach((celda, c) => {
65          const idx = r * TAMAÑO + c;
66          if (this.celdas[idx]) {
67            this.aplicarEstilo(this.celdas[idx], celda);
68          }
69        });
70      });
71    }
72
73    obtenerCeldasConListener(callback) {
74      // Usar delegación de eventos en el contenedor
```

```
    this.contenedor.addEventListener('click', (e) => {
      const celda = e.target.closest('[data-r][data-c]');
      if (celda) {
        const r = parseInt(celda.dataset.r);
        const c = parseInt(celda.dataset.c);
        callback(r, c);
      }
    });
  }

  marcarSeleccion(r, c) {
    if (this.seleccionada) {
      this.seleccionada.classList.remove('seleccion');
    }
    if (r >= 0 && r < TAMAÑO && c >= 0 && c < TAMAÑO) {
      const idx = r * TAMAÑO + c;
      this.seleccionada = this.celdas[idx];
      this.seleccionada.classList.add('seleccion');
    }
  }

  marcarPreview(celdas, valido = true) {
    // Limpiar preview anterior
    this.celdas.forEach(c => {
```

```
    this.celdas.forEach(c => {
      c.classList.remove('seleccion');
      c.classList.remove('preview-invalido');
    });

    // Marcar nuevas celdas
    celdas.forEach(({ r, c }) => {
      const idx = r * TAMAÑO + c;
      if (this.celdas[idx]) {
        if (valido) {
          this.celdas[idx].classList.add('seleccion');
        } else {
          this.celdas[idx].classList.add('preview-invalido');
        }
      }
    });
  }

  limpiarPreview() {
    this.celdas.forEach(c => {
      c.classList.remove('seleccion');
      c.classList.remove('preview-invalido');
    });
  }
```

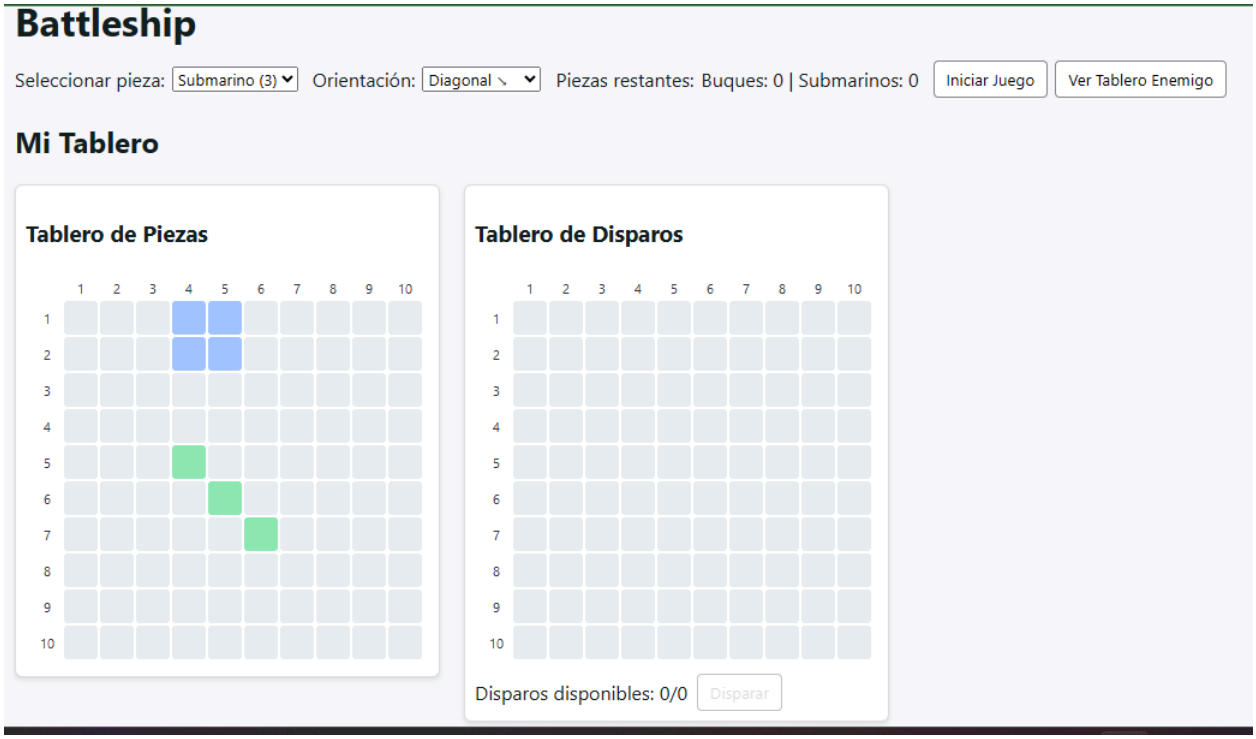
```
122  }
123
124  /**
125   * VistaControles - Gestiona los controles de la interfaz
126   */
127  class VistaControles {
128    constructor() {
129      this.selectPieza = document.getElementById('select-pieza');
130      this.selectOrientacion = document.getElementById('select-orientacion');
131      this.contadorPiezas = document.getElementById('contador-piezas');
132      this.btnIniciar = document.getElementById('btn-iniciar');
133      this.btnDispara = document.getElementById('btn-disparar');
134      this.btnVerEnemigo = document.getElementById('btn-ver-enemigo');
135      this.btnVolver = document.getElementById('btn-volver');
136      this.poolInfo = document.getElementById('pool-info');
137      this.historialJugador = document.getElementById('historial-jugador');
138      this.historialEnemigo = document.getElementById('historial-enemigo');
139      this.vistaJugador = document.getElementById('vista-jugador');
140      this.vistaEnemigo = document.getElementById('vista-enemigo');
141    }
142
143    obtenerPiezaSeleccionada() {
144      return {
145        tipo: this.selectPieza.value,
146        orientacion: this.selectOrientacion.value
```

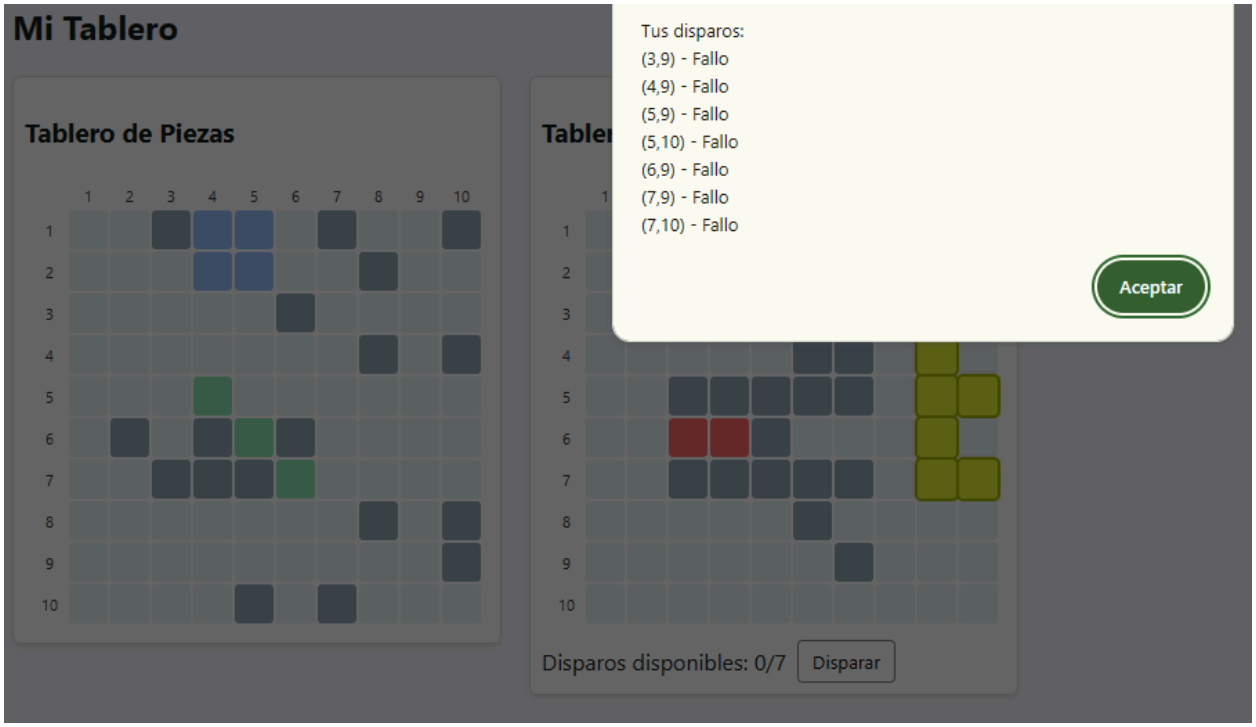
```
147     };
148   }
149
150   actualizarContador(barcos, submarinos) {
151     this.contadorPiezas.textContent = `Buques: ${barcos} | Submarinos: ${submarinos}`;
152   }
153
154   habilitarBotones(iniciar = false, disparar = false) {
155     this.btnIniciar.disabled = !iniciar;
156     this.btnDispara.disabled = !disparar;
157   }
158
159   ////////////////////////////////////
160   actualizarPoolInfo(disponibles, total) {
161     this.poolInfo.textContent = `Disparos disponibles: ${disponibles}/${total}`;
162   }
163
164   agregarAlHistorial(elemento, tipo, r, c, resultado) {
165     const li = document.createElement('li');
166     li.textContent = `${tipo}: (${r},${c}) - ${resultado}`;
167
168     if (elemento === 'historialJugador') {
169       this.historialJugador.appendChild(li);
170     } else {
```

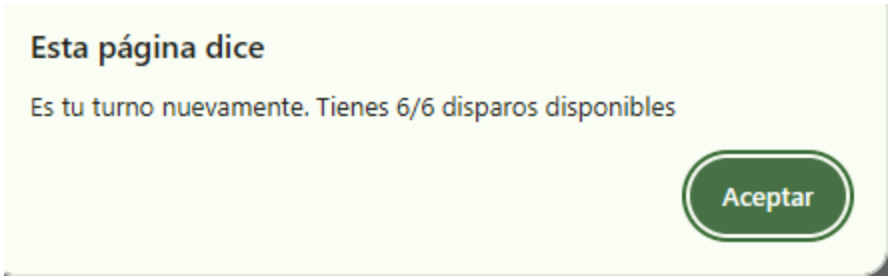
```
171         this.historialEnemigo.appendChild(li);
172       }
173     }
174
175     mostrarVistaEnemigo() {
176       this.vistaJugador.style.display = 'none';
177       this.vistaEnemigo.style.display = 'block';
178       this.btnVerEnemigo.style.display = 'none';
179       this.btnVolver.style.display = 'inline-block';
180     }
181
182     volverAMiTablero() {
183       this.vistaJugador.style.display = 'block';
184       this.vistaEnemigo.style.display = 'none';
185       this.btnVerEnemigo.style.display = 'inline-block';
186       this.btnVolver.style.display = 'none';
187     }
188
189     agregarEventoAlBtnIniciar(callback) {
190       this.btnIniciar.addEventListener('click', callback);
191     }
192
193     agregarEventoAlBtnDispara(callback) {
194       this.btnDispara.addEventListener('click', callback);
```

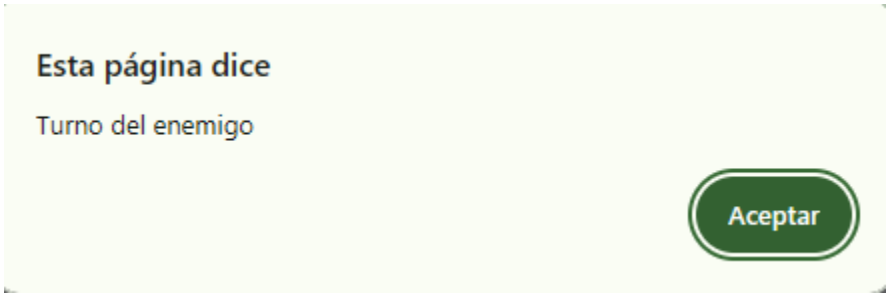
```
195     }
196
197     agregarEventoAlBtnVerEnemigo(callback) {
198       this.btnVerEnemigo.addEventListener('click', callback);
199     }
200
201     agregarEventoAlBtnVolver(callback) {
202       this.btnVolver.addEventListener('click', callback);
203     }
204   }
205
206   /**
207    * VistaPrincipal - Coordina todas las vistas
208    */
209   class VistaPrincipal {
210     constructor() {
211       this.tablerojugador = new VistaTablero('tablero-jugador', true);
212       this.tableroDisparos = new VistaTablero('tablero-disparos', false);
213       this.tableroEnemigo = new VistaTablero('tablero-enemigo', false);
214       this.controles = new VistaControles();
215     }
216
217     renderizarTableros(estadoJugador, estadoEnemigo) {
218       this.tablerojugador.crear(estadoJugador.grid);
219       this.tableroDisparos.crear(estadoEnemigo.grid);
220       this.tableroEnemigo.crear(estadoEnemigo.grid);
221     }
222
223     actualizarTableros(estadoJugador, estadoEnemigo) {
224       this.tablerojugador.actualizar(estadoJugador.grid);
225       this.tableroDisparos.actualizar(estadoEnemigo.grid);
226       this.tableroEnemigo.actualizar(estadoEnemigo.grid);
227     }
228   }
229
```

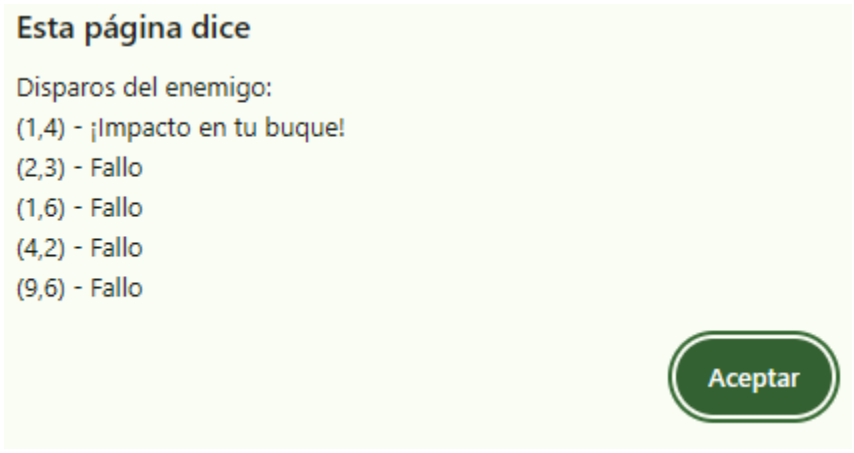
Capturas del programa funcionando

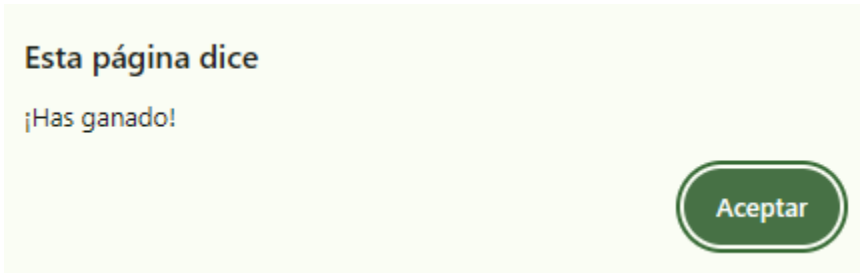





Esta página dice
Es tu turno nuevamente. Tienes 6/6 disparos disponibles
Aceptar


Esta página dice
Turno del enemigo
Aceptar


Esta página dice
Disparos del enemigo:
(1,4) - ¡Impacto en tu buque!
(2,3) - Fallo
(1,6) - Fallo
(4,2) - Fallo
(9,6) - Fallo
Aceptar

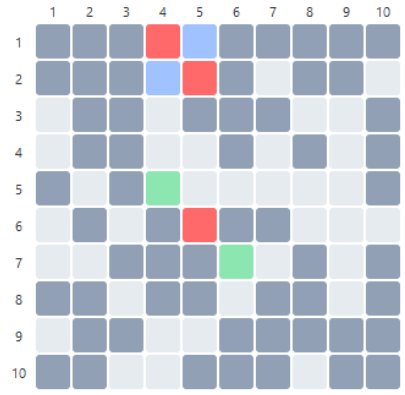

Esta página dice
¡Has ganado!
Aceptar

Battleship

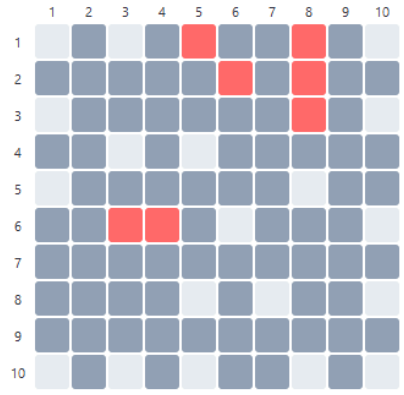
Seleccionar pieza: Submarino (3) Orientación: Diagonal ↘ Piezas restantes: Buques: 0 | Submarinos: 0 Iniciar Juego Ver Tablero Enemigo

Mi Tablero

Tablero de Piezas



Tablero de Disparos



Disparos disponibles: 4/4 Disparar

Historial (mis disparos)

- Disparo: (0,0) - fallo
- Disparo: (0,9) - fallo
- Disparo: (4,3) - fallo
- Disparo: (4,4) - fallo
- Disparo: (5,4) - fallo
- Disparo: (9,0) - fallo
- Disparo: (9,9) - fallo
- Disparo: (2,4) - fallo
- Disparo: (2,5) - fallo
- Disparo: (2,6) - fallo
- Disparo: (3,5) - fallo
- Disparo: (3,6) - fallo
- Disparo: (4,5) - fallo
- Disparo: (4,6) - fallo
- Disparo: (5,3) - impacto
- Disparo: (6,3) - fallo
- Disparo: (6,4) - fallo
- Disparo: (6,5) - fallo
- Disparo: (6,6) - fallo
- Disparo: (7,5) - fallo
- Disparo: (8,6) - fallo
- Disparo: (1,1) - fallo
- Disparo: (4,2) - fallo
- Disparo: (4,3) - fallo

Conclusión

La implementación del juego tipo Battleship utilizando los patrones de diseño Flyweight, Object Pool, Memento y el enfoque arquitectónico MVC permitió construir una solución eficiente, modular y fácil de mantener. El uso de Flyweight optimizó la creación de los tableros al reducir el consumo de memoria; Object Pool mejoró el rendimiento en la gestión de disparos al reutilizar objetos en lugar de generarlos continuamente; y Memento proporcionó una forma segura y estructurada de restaurar estados previos de la partida, facilitando la función de deshacer acciones. Integrado bajo la arquitectura MVC, el sistema separa claramente la lógica, la interfaz y el control, lo que favorece la escalabilidad y la organización del código. En conjunto, estos patrones permitieron desarrollar un proyecto sólido y alineado con buenas prácticas de ingeniería de software.