UNIVERSIDAD POLITÉCNICA DE TECÁMAC

UNIVERSIDAD POLITÉCNICA DE TECÁMAC

Docente: Emmanuel Torres Servín

Integrantes:	**Matriculas:**
• Alemán Pérez Natali Joselin	1321124050
• Ángel Velasco Marco Joel	1321124015
• De la Cruz Medina José Eduardo	1321124069
• De León Carbajal Isaac	1321124054
• Díaz García Karla Faviola	1321124001

Grupo: 5322IS

Batalla Naval

Índice

Explicación de la maqueta mockups y su importancia

Un mock-up es un prototipo hecho antes del desarrollo del trabajo. Sirve para transformar ideas en funcionalidades y ayuda al cliente a exteriorizar y comprender lo que necesitamos, dicho de otra manera un mockup no es más que un montaje que realizan los diseñadores gráficos y diseñadores web**,** para mostrar a sus clientes cómo quedarán sus diseños impresos en alguna superficie.

Al trabajar con un mock-up, el cliente expone ante el profesional qué es lo que pasa por su cabeza, y consigue comprender mejor el nivel de complejidad del trabajo, lo cual también facilita la comunicación y desarrollo del proyecto luego de la contratación, además permite que el producto sea testeado a un costo mucho menor.

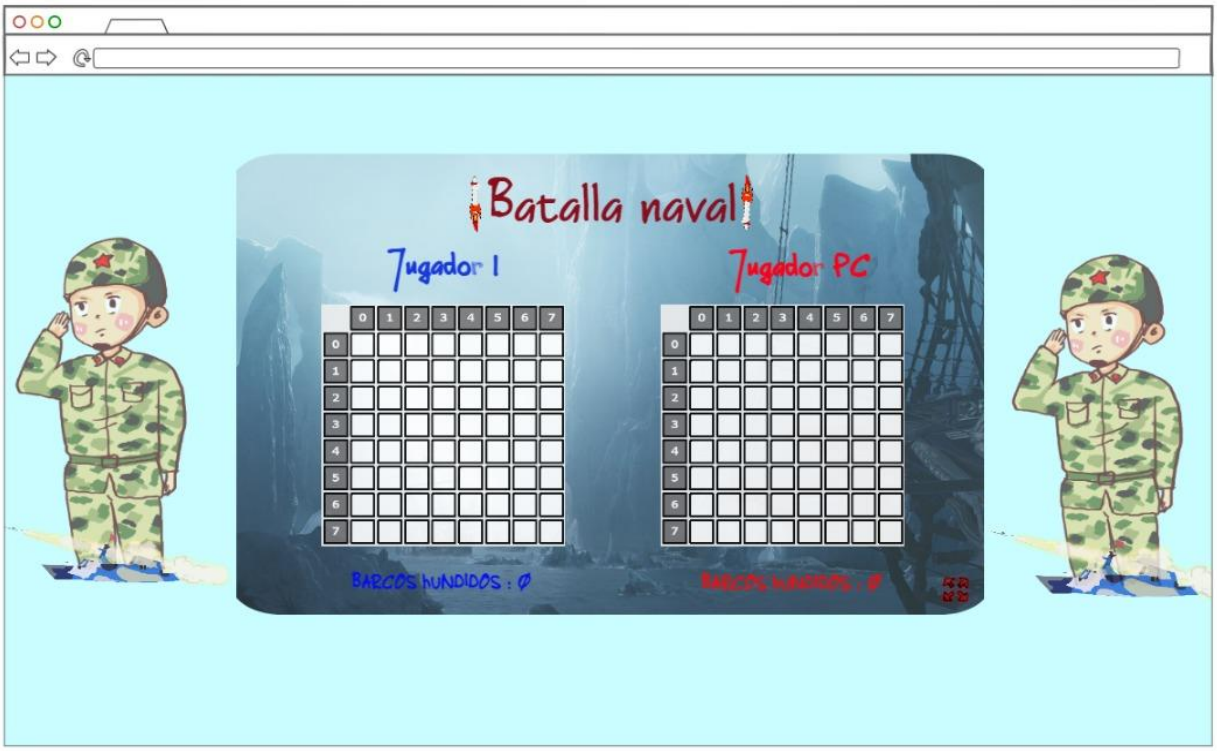
Es una pieza fundamental porque al momento de reunirnos con el cliente realizamos la estructura del diseño del sitio o aplicación web, permitiendo tener una idea más clara de cómo van a estar distribuidos los elementos, como la barra de navegación, los titulares, las imágenes, los párrafos. etc.

El valor y tiempo para ajustes en esa etapa es enorme, y esto puede evitarse pasando por una fase de test a través del mock-up.

Mockups (Batalla Naval)

Inicio de juego

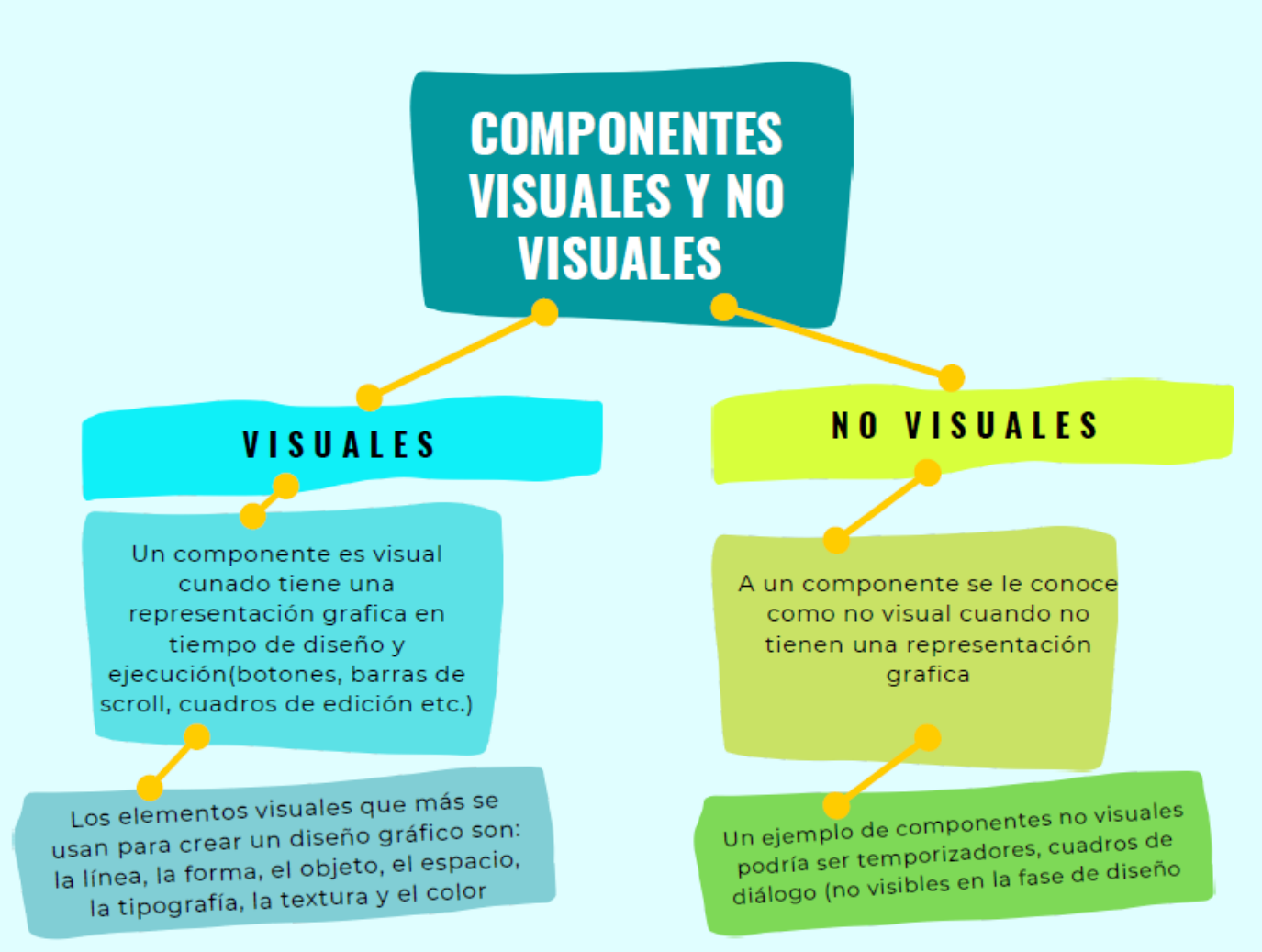
Tablero de juego



Lista de componentes visuales y no visuales.

Componentes	
Componentes visuales	**Componentes no visuales**
Botones	Tamaño de los paneles
Etiquetas	
Imágenes	Color de los paneles
Iconos	
Barras de scroll	

Mapa conceptual de componentes visuales y no visuales



Juego Batalla Naval

Reglas del juego

- Tablero de n tamaño.
- Cada jugador tiene 5 barcos.
- El barco puede ocupar cualquier posición y no se debe encimar.
- Dimensión (x, y).
- Se muestra la posición de los tiros.
- No se puede tirar dos veces en la misma posición.
- El tamaño del barco es de una posición.
- El juego termina cuando un jugador ya no tenga barcos.

Documentación

Clase Juego

En la clase juego se guardan las posiciones de los barcos en un arreglo. Así mismo para los tableros se ocupan arreglos bidimensionales. Una vez ingresado el nombre del usuario podrá definir el tamaño de los tableros y la posición en la que desea colocar sus barcos, pero se debe tomar en cuenta que el tablero tiene las dimensiones de (x, y).

```
class Juego {

    constructor() { }

    jugar() {
        //Nombre de los jugadores
        let nombre1;
        let nombre2;
        let ganador = ""; //Cuando alguien gane, se guardará su nombre

        //Posiciones de los barcos de los jugadores
        let posBarcos1 = null;
        let posBarcos2 = null;

        //Cantidad de barcos destruidos por los jugadores
        let puntos1 = 0;
        let puntos2 = 0;

        //Crear los tableros
        let tablero1 = this.crearTablero(tamanoTablero);
        let tablero2 = this.crearTablero(tamanoTablero);
        //Ingresar los barcos
        alert("Ingresar los barcos de " + nombre1 + "\n");
        posBarcos1 = this.pedirPosiciones(cantBarcos, false);
        console.log("El jugador " + nombre1 + " ha ingresado sus barcos");
        this.mostrarTablero(tablero1);
```

El uso de **while** consiste en que mientras haya un jugador, se otorgarán los turnos al jugador y al bot tirando uno a la vez, así mismo cada vez que un jugador hunda un barco se le sumará un punto a ese jugador. Cada vez que los jugadores acaben de tirar se mostrarán tableros actuales y el lugar donde tiraron.

```
while (ganador == "") { //Mientras no haya un ganador
    //Turno del jugador 1
    if (this.realizarTurno(nombre1, tablero1, posBarcos2, false)) { //Si el jugador atina el ataque
        puntos1++; //Sumarle un punto
    }

    //Turno del jugador 2
    if (this.realizarTurno(nombre2, tablero2, posBarcos1, true)) { //Si el jugador atina el ataque
        puntos2++; //Sumarle un punto
    }

    if (puntos1 == cantBarcos && puntos2 == cantBarcos) { //Si ambos jugadores terminaron los barcos del otro en el mismo turno
        ganador = "empate";
    } else if (puntos1 == cantBarcos) { //Si el jugador 1 alcanzó los puntos necesarios
        ganador = nombre1;
    } else if (puntos2 == cantBarcos) { //Si el jugador 2 alcanzó los puntos necesarios
        ganador = nombre2;
    }
```

Se piden posiciones para poner los barcos del jugador y guardarlas en un arreglo, y en caso de que las posiciones estén ocupadas también se mostrará que esa posición no está disponible (no se pueden colocar dos barcos en la misma posición), de lo contrario si está disponible es turno del otro jugador para registrar sus posiciones. Se utiliza el método **enteroRamdom()** para generar las posiciones del bot.

```
        err = false;
        nums = [this.enteroRandom(1, cantBarcos), this.enteroRandom(1, cantBarcos)]; //Obtener dos numeros al azar

        if ((nums[0] < 1 || nums[0] > tamanoTablero) || (nums[1] < 1 || nums[1] > tamanoTablero)) { //Si alguno de los numeros está fuera del rango
            err = true; //Reintentar
        }
    } while (err);

    pos = new Posicion(nums[0], nums[1]);
}

if (pos != null) { //Si la posicion es valida
    for (let i = 0; i < cont; i++) { //Revisar el resto de posiciones
        if (posiciones[i].equals(pos)) { //Si la posicion ya esta ocupada
            if (!esBot) { //Si es un turno del jugador
                alert("La posicion " + pos.toString() + " ya esta ocupada"); //Sólo avisar si es el jugador
            }
            error = true;
            break; //Salir del for
```

Si el ataque es válido, se actualizará el tablero y con el método **ataque.toString()** (de la clase Posición) se muestra la posición en consola que el usuario intentó atacar o atacó correctamente.

```
if (tablero[ataque.x][ataque.y] == atacado || tablero[ataque.x][ataque.y] == destruido) {
    alert("Esa posicion ya habia sido atacada");
    console.log(nombre + " volvió a atacar la posición " + ataque.toString());
    return false; //Indicar que el ataque no alcanzó objetivo
} else { //Si la posicion no habia sido atacada
    for (let i = 0; i < posEnemigo.length; i++) {
        if (posEnemigo[i].equals(ataque)) { //Si el enemigo tiene un barco en la posicion atacada
            alert("El ataque fue exitoso");
            console.log(nombre + " destruyó un barco en la posición " + ataque.toString());
            tablero[ataque.x][ataque.y] = destruido; //Actualizar el tablero
            return true; //Indicar que el ataque fue exitoso
        }
    }
```

El método **crearTablero()** sirve para generar un arreglo bidimensional del tamaño que el usuario haya especificado. Los valores iniciales del arreglo representan casillas no atacadas.

```
crearTablero(tamano) {
    let tablero = [];

    for (let x = 0; x < tamano; x++) { //En todo lo largo
        let aux = [];

        for (let y = 0; y < tamano; y++) { //En todo lo alto
            aux.push(noAtacado); //Agregar un caracter de casilla no atacada
        }

        tablero.push(aux);
    }

    return tablero;
}
```

Clase Posicion

```
class Posicion {
    x = 0;
    y = 0;
```

Se declara un método estático que se llama **pedirPosicion(),** donde se le pide al usuario colocar la posición que él desea en el tablero, retorna el objeto Posición en la posición que fue ingresada. Se usa el parámetro de maxPos para la posición máxima que se puede elegir de acuerdo con el tamaño del tablero, solicitado al usuario anteriormente.

Posteriormente se encuentran varias condiciones **if**, que evalúan si la posición es válida de acuerdo con su tablero. Tales condiciones son que debe de ser un número, deben de ser dos números correspondientes a las coordenadas **x** & **y,** debe de estar dentro del rango del tablero. Si estas condiciones no se cumplen entonces no devolverá ninguna posición dentro del tablero, junto con alertas indicando su error.

En caso de cumplirlas, se devuelve la posición deseada**.**

```
    static pedirPosicion(maxPos) {···
    }
```

Otro método estático es **separarStringEnNumeros** usando los parámetros de texto y separador, así como retornar un arreglo con los datos.

Para separar un String, vaciando los valores separados en un arreglo**.**

Si el elemento no es un número no se retornará nada, si el carácter no es un separador se agrega a la variable de tipo string.

```
static separarStringEnNumeros(texto, separador) {···
}
```

Lo siguiente es un **constructor** que se usa para construir una posición a partir de dos números enteros, usando **this** que hace referencia a los atributos **x** & **y** para la posición en el tablero como un plano cartesiano.

```
constructor(x, y) {···
}
```

Después, el método **toString(),** retorna la forma en que se mostrara las posiciones de los jugadores de la siguiente manera: **(6, 7).**

```
toString() {···
}
```

Por último, el método **equals** recibe un parámetro, **pos**, siendo esta la posición a comparar. La función tendrá como objetivo comparar dos posiciones y saber si apuntan a la misma casilla, retornando valores booleanos. true si los valores de las posiciones son iguales y false si no lo son.

```
equals(pos) {···
}
```