**Carta**

A lo largo del juego una carta cualquiera se encuentra en solo uno de tres estados posibles:

* Oculta: aun no se ha hecho click en ella y por tanto solo se puede ver el reverso de la carta.
* Mostrada/Visible/Seleccionada: ha sido clicada, se muestra su contenido y se le aplica un estilo especial para resaltar que ha sido seleccionada.
* Bloqueada/Resuelta/Inactiva: el jugador ha encontrado la pareja de esta carta y ya no puede interactuar más con ella.

En el siguiente diagrama se muestran las transiciones válidas entre estos estados:



Para implementar esta funcionalidad he usado tres métodos y dos atributos booleanos “oculta” y “bloqueada”. Cada estado de la carta se corresponde además con un estilo definido en un archivo css:

Pantalla de computadora con letras

El contenido generado por IA puede ser incorrecto.

Texto

El contenido generado por IA puede ser incorrecto.

Captura de pantalla de computadora

El contenido generado por IA puede ser incorrecto.

Interfaz de usuario gráfica

El contenido generado por IA puede ser incorrecto.

En caso de que la operación tenga éxito, es decir la carta cambie de estado, estos métodos devolverán verdadero, en caso contrario devolverán falso. Esta decisión la tomé para evitar tener que acceder a los atributos de estado de un objeto Carta desde una clase externa y así encapsular más mi código. Todas las comprobaciones necesarias relativas al estado de la carta se llevan a cabo dentro del propio método.

Me he inspirado en métodos de la interfaz Collection de Java como “add()” o “remove()” que devuelven un booleano para indicar si la operación se ha llegado a realizar o no.

Para resolver la problemática de emparejar cartas e identificar las parejas he usado un atributo “idPareja” que es utilizado por el siguiente método para comprobar si dos cartas son pareja:

Una captura de pantalla de un celular

El contenido generado por IA puede ser incorrecto.

El hecho de que el id de la pareja se guarde dentro del objeto hace redundante que se añada un atributo del mismo nombre al elemento HTML.

El resto de atributos y métodos de la clase Carta están relacionados con el HTMLImageElement que contiene la imagen de la carta y sus propiedades, tales como “src”, “id” y “class”. El constructor es además usado para crear el elemento HTML.

**Cartas**

Además en el transcurso del juego solo puede haber dos cartas como máximo en el estado Mostrada/Visible, para imponer esta restricción he utilizado un atributo de clase o estático que lleva la cuenta del número de cartas visibles y que resulta de gran importancia para el funcionamiento de la clase Juego.

Interfaz de usuario gráfica, Texto, Chat o mensaje de texto

El contenido generado por IA puede ser incorrecto.

Este atributo es actualizado cada vez que una carta cambia de estado, es decir, cada vez que se llama a uno de los métodos de instancia de la clase Carta.

**Contador**

Al principio implementé el contador como una clase instanciable. Sin embargo, esto me estaba dando bastantes quebraderos de cabeza puesto que en algún punto siempre acababa con varias instancias de la clase Contador, todas ellas con la misma referencia al elemento HTML que uso para mostrar la información y actualizandolo a la vez.

Para evitar errores decidí implementar el contador como una clase ‘estática’, ya que de todas maneras solo va a haber un único contador por juego.

De igual manera, por cada elemento html que debía manejar, he creado una clase que guardase la referencia al elemento en un atributo estático y que realizase todos los cambios pertinentes a través de métodos estáticos. Al fin y al cabo tal y como he estructurado mi código html solo hay un único contador, un único contenedor de vidas, un único contenedor de cartas, …, no tiene mucho sentido poder hacer varios objetos diferentes de estas clases.