**MVC dolgozat javítása**

--- Kádár Kristóf ---

(Szürke színű kód az eredeti, sárga a javított!)

***Controll.js :***

***1.***

new JatekTer(lista); 🡪 this.jatekTer = new JatekTer(lista);

Az első változatnál minden egyes kattintásnál újra létrejön a teljes játékterület, míg a második változatban csak a tartalom frissül a már létező JatekTer példányban.

***2.***

… 🡪 this.jatekTer.empty();

Nem hozunk létre új játékteret minden alkalommal, hanem töröljük ürítjük és frissítjük a meglévőt.

***3.***

this.copyModell.ellenorzes(); 🡪 this.copyModell.ellenorzes(*event*.detail);

A második változat pontosabb, mert figyelembe veszi a kiválasztott kártya indexét az ellenőrzéskor.

***Modell.js:***

***1.***

előre kész lista 🡪 generált lista (30% miatt)

A statikus, előre kódolt elrendezés unalmassá válhat, mivel a játékosok megtanulják a kártyák elhelyezkedését.

***2.***

if(index === 3) 🡪 if (this.lista[*index*] === '👺')

Az index kódolása korlátozza a kártyák ellenőrzésének lehetőségeit. Így az ellenorzes metódust paraméterezetté tettük, hogy bármelyeik kártyát le tudja ellenőrizni.

***3.***

console.log("Elvitt az ördög!") 🡪 return "Elvitt az ördög!";

Ez eslő módszer nem ad vissza értéket, hanem közvetlenül a konzolra írja a találatokat. A második egy Stringet ad vissza, amely később még felhasználható és változtatható lesz.

***4.***

#index 🡪 …

Egy privát mező (#index) van definiálva, de a megadott módszerekben nem használjuk. Felesleges.

***JatekTer.js:***

***1.***

this.osszerakHTML(); this.szuloElem.empty();

this.szuloElem.empty(); 🡪 this.osszerakHTML();

A this.szuloElem.empty(); hívás a konstruktorban, az elemek hozzáadása előtt történik. Ez azt biztosítja, hogy a tartalom törlése mindig előbb történik meg, mielőtt új elemeket adunk hozzá.

***2.***

this.szuloElem.append("szöveg"); 🡪 this.szuloElem.text(*szoveg*);

A második verzió újrahasznosíthatóbb, mivel lehetővé teszi a különböző szövegek beállítását anélkül, hogy módosítani kellene a metódust.

***Kartya.js:***

***1.***

*const* felepites = `<div class="kartya"></div>`

this.szuloElem.append(felepites) 🡪

this.kartyaElem = $(`<div class="kartya"></div>`);

        this.szuloElem.append(this.kartyaElem);

A kártya HTML struktúráját közvetlenül a felepites változóba menti, amely nem elérhető más metódusokból. Ez a kód nem fut le, mivel a JSX szintaxis nem megfelelő a JavaScriptben (hiányzik a $ jel). A másodikban a kártya elem jQuery-vel jön létre, és elmenti az osztály adattagjaként (this.kartyaElem), így később is hozzáférhet a kártyához.

***2.***

this.szuloElem.on("click", () => {

const e = new CustomEvent("kattint", {detail: this.#index});

window.dispatchEvent(e);

felepites.append(this.#lista[this.#index]);

});

🡪

this.kartyaElem.on("click", () => {

const e = new CustomEvent("kattint", { detail: this.#index });

window.dispatchEvent(e);

this.kartyaElem.text(this.#lista[this.#index]);

this.kartyaElem.addClass("felforditott");

if (this.#lista[this.#index] === '👺') {

$(".kartya").off("click");

} });

Az első az eseménykezelő a szuloElem-re van beállítva, ami azt jelenti, hogy bármelyik kártyára kattintva a szülőelem eseménykezelője aktiválódik, nem csak az adott kártyáé. A felepites változót nem használjuk helyesen, mivel nem elérhető a esemenyKezelo() metódusban.

A másodikban az eseménykezelő közvetlenül a kártya elemére van beállítva, így pontosan azt a kártyát célozza meg, amelyikre kattintottak. Különböző reakciót ad vissza, attól függően, hogy a kártya ördög (👺) vagy szellem (👻). Ha az ördög kártyára kattintunk, letiltja az összes kártya kattintási eseményét.