# LightOn MVC

## MVC összetevők:

A képen szöveg, képernyőkép, Betűtípus, szám látható

Automatikusan generált leírás

1. **Model (Adatkezelő, Logika)**
   * **Feladata**: A Model felelős az alkalmazás belső logikájáért és adatainak kezeléséért. Az ábrán látható példa szerint ez tartalmazza azt az adatstruktúrát (egy tömböt), amely a lámpák aktuális állapotát reprezentálja: fel vannak-e kapcsolva, vagy le vannak kapcsolva.
   * **Tömb**: A tömb minden eleme egy-egy lámpa állapotát tárolja (pl. f az "off" vagyis kikapcsolt, l az "on" azaz bekapcsolt).
   * **Szerepe**: A Model tudja, hogy milyen állapotban van a rendszer, és képes az adatok módosítására (például a szomszédos lámpák állapotának változtatására). A Controller segítségével kommunikál a View-val.
2. **Controller (Irányítás, Közvetítés)**
   * **Feladata**: A Controller a közvetítő a Model és a View között. Felelős azért, hogy a felhasználói interakciókat kezelje, és a megfelelő változásokat végrehajtsa az alkalmazásban.
   * **Példányosítja a View-t és a Model-t**: A Controller létrehozza a View-t (felhasználói felületet) és a Model-t (adatkezelés).
   * **Kommunikáció**: Az események (például egy lámpa kattintásának érzékelése) továbbítása a View-ból a Model-be, ahol a változás történik, majd az új adatokat visszaküldi a View-nak, hogy az frissítse a kijelzést.
   * **Például**: Ha egy lámpára kattintasz, a Controller értesül az eseményről, majd frissíti a Model adatait, és új állapotot küld a View-nak.
3. **View (Megjelenítés)**
   * **Feladata**: A View felelős a felhasználói felületért és a megjelenítésért. HTML, CSS és JavaScript segítségével mutatja meg az adatokat a felhasználónak.
   * **HTML és CSS**: A View gondoskodik arról, hogy a lámpák megfelelően jelenjenek meg a kijelzőn, vizuálisan reprezentálva az állapotukat (pl. sárga színű lesz, ha be van kapcsolva).
   * **Lámpa JS osztály**: A View egy lámpa objektumot használ, amely létrehoz egy saját eseményt (például kattintásra reagál), és jelzi a Controllernek, hogy változtatás történt.
   * **Feladata a megjelenítés frissítése**: Amikor a Model-ben változik az állapot (például egy lámpa felkapcsol), a View frissíti a kijelzést.

**Mire jó az MVC?**

Az MVC minta előnyei közé tartozik:

* **Jobb karbantarthatóság**: A kód külön részekre van osztva, amelyek különböző felelősségi köröket látnak el. Könnyebb változtatni a megjelenésen (View), anélkül, hogy az adatkezelést (Model) megbolygatnánk.
* **Újrafelhasználhatóság**: A különböző rétegek (Model, View, Controller) önállóan újrahasználhatók más alkalmazásokban vagy más projektekben.
* **Könnyebb hibakeresés**: Mivel az egyes részek külön funkciókat látnak el, könnyebb megtalálni, ha valahol probléma van (pl. ha egy állapotfrissítés nem jelenik meg, akkor lehet tudni, hogy a hiba valószínűleg a View-ban van).

Összességében az MVC egy olyan struktúra, amely a fejlesztést és a karbantartást egyszerűsíti azzal, hogy a logikai elemeket, az adatkezelést és a megjelenítést elkülöníti egymástól.

A képen szöveg, képernyőkép, szoftver, Betűtípus látható

Automatikusan generált leírás

LightOn szerkezete:

## A képen szöveg, képernyőkép, képernyő, szoftver látható Automatikusan generált leírásJatekController

1. Az első két sor importálja a modellt és a nézetet (View), hogy azokkal dolgozhasson a vezérlő (Controller).
2. A JatekterController osztály felelős a játék irányításáért, összeköti a nézetet és a modellt.
3. A nézet és a modell is egy adott mérettel (meret) jön létre, amely jelen esetben 3.
4. Az eseménykezelő figyeli a "lampaKapcsolas" nevű egyéni eseményt, és amikor egy lámpa kapcsolása történik, akkor meghívja a modell megfelelő metódusait az állapotváltás érdekében.
5. Miután a modell frissítette az állapotot, a nézetnek visszaadja az aktuális állapotot, hogy vizuálisan is megjelenítse a változásokat.

A képen szöveg, képernyőkép, szoftver, képernyő látható

Automatikusan generált leírás

A képen szöveg, Multimédiás szoftver, szoftver, Grafikai szoftver látható

Automatikusan generált leírás

1. A #jatekTerTomb privát változóként van definiálva, ami a játéktér minden lámpájának állapotát tartalmazza (true a bekapcsolt, false a kikapcsolt állapot).
2. A konstruktorban a játéktér méretének megfelelően inicializáljuk a lámpák állapotát, kezdetben mindegyik ki van kapcsolva.
3. A szomszedokValtoztatasa(id) metódus gondoskodik arról, hogy a kiválasztott lámpa és annak közvetlen szomszédai (balra, jobbra, fölött, alatta) állapotot váltsanak.

## A képen szöveg, képernyőkép, szoftver, képernyő látható Automatikusan generált leírásJatekter

1. A #LampaLista privát változóként tárolja az összes lámpa objektumot, amelyek a játéktéren találhatók.
2. A konstruktor beállítja a játéktér méretét (CSS-ben is), és létrehozza az összes lámpa objektumot. Azokat a lámpa objektumokat a #LampaLista tömbbe menti.
3. A setLampakAllapota(tomb) metódus fogadja a modelltől kapott állapotokat (egy true/false tömb formájában), és frissíti az összes lámpa állapotát a játéktérben.

## Lampa

A képen szöveg, képernyőkép, szoftver, Multimédiás szoftver látható

Automatikusan generált leírás

A képen szöveg, szoftver, Multimédiás szoftver, Grafikai szoftver látható

Automatikusan generált leírás

1. A #allapot privát változó jelzi, hogy a lámpa be van-e kapcsolva (true) vagy ki van kapcsolva (false).
2. A #index privát változó tárolja, hogy a lámpa hol helyezkedik el a játéktérben (pozíciója).
3. A konstruktorban hozzáadjuk a lámpát a DOM-hoz, majd beállítjuk a kezdeti állapotát és eseménykezelőt adunk hozzá.
4. A setAllapot metódus frissíti a lámpa állapotát a vezérlő által adott értékek alapján.
5. A setSzin metódus megváltoztatja a lámpa színét az állapot szerint.
6. A #kattintasTigger egy egyéni eseményt hoz létre, amelyet a vezérlő felé jelez a lámpa kapcsolásakor.

## A képen szöveg, képernyőkép, szoftver, képernyő látható Automatikusan generált leírásMain

1. A $() függvény egy jQuery inicializáló, amely az oldal betöltése után hívódik meg, itt példányosítjuk a JatekterController osztályt, ami a program vezérlését végzi.
2. A kommentált részek olyan kódrészek, amelyek már áthelyezésre kerültek a vezérlő vagy a modell osztályokba. Az eseménykezelést most már a JatekterController és a modell végzi.
3. A #szomszedokValtoztatasa funkciót is áthelyeztük a modellbe, mivel az állapotváltoztatás a modell felelőssége.

## Index.html

A képen szöveg, Multimédiás szoftver, szoftver, Grafikai szoftver látható

Automatikusan generált leírás

A képen szöveg, képernyőkép, szoftver, képernyő látható

Automatikusan generált leírás

1. A .jatekter osztály a **grid** elrendezést használja, amely három egyenlő szélességű oszlopot hoz létre. A gap 10px-es távolságot biztosít az elemek között.
2. A .felkapcs osztályt a lámpák "bekapcsolt" állapotához használod, és biztosítod, hogy a háttérszín felülírja az alapértelmezett zöldet a !important használatával.
3. A h1 és .forma szabályok a szövegek stílusait határozzák meg, fehér színnel és középre igazítva.