Az MVC tervezési minta

Az MVC egy népszerű szoftvertervezési minta, amely segít az alkalmazások szerkezetének áttekinthetővé tételében. Célja, hogy a különböző részek – adatkezelés, megjelenítés és vezérlés – elkülönüljenek egymástól.

Három részre bontja az alkalmazást: a *Model* kezeli az adatokat, a *View* mutatja a felhasználónak, a *Controller* pedig összekapcsolja a kettőt. Így könnyebb külön dolgozni a részeken, új funkciókat hozzáadni, hibákat javítani, és jobb felületeket készíteni. Az MVC rendszerezetté teszi az alkalmazást és javítja a minőségét.

Az MVC modellt előnyeit, filozófiáját és működésének gerincét a következő fogalmak mutatják be:

Model: A Model az alkalmazás adatait és üzleti logikáját kezeli. Felelős az adatok tárolásáért, érvényességük ellenőrzéséért, valamint a változások kezeléséért, így biztosítva, hogy az alkalmazás helyesen működjön.

View: A View a felhasználó számára látható felületet jelenti, amely az adatokat prezentálja. Nem tartalmaz üzleti logikát, célja csupán az információk megjelenítése és a felhasználói élmény biztosítása.

Controller: A Controller kezeli a felhasználói interakciókat, például gombnyomásokat vagy űrlapbeküldéseket. Összekapcsolja a Modelt és a View-t, meghatározva, hogy milyen műveletek történjenek a felhasználói input hatására.

Üzleti logika: Az üzleti logika a Model része, és az alkalmazás fő működését határozza meg. Ez az, ami az adatok feldolgozását és a döntések végrehajtását szabályozza.

Szeparáció elve: Az MVC egyik alapelve, hogy a feladatokat elkülönítve kezeljük. Ez megkönnyíti a karbantartást, a fejlesztést, és lehetővé teszi, hogy az egyes részek függetlenül cserélhetők vagy módosíthatók legyenek.

Adat- és eseményközvetítés: A Model és a View közötti frissítések kezelése biztosítja, hogy a felhasználói felület mindig aktuális adatokat mutasson. Ez lehet automatikus vagy kézi, és segít a szinkronizáció fenntartásában.

Router / navigáció: Különösen webes alkalmazásokban a router irányítja a bejövő kéréseket a megfelelő Controllerhez, így biztosítva a helyes adat- és felületkezelést.

Állapotkezelés: Az állapotkezelés a Model aktuális állapotának nyomon követését jelenti, hogy a View mindig a legfrissebb adatokat jelenítse meg a felhasználónak.

Tesztelhetőség: Az MVC szétválasztja a logikát, a felületet és az interakciókat, így könnyebb egység- és integrációs teszteket készíteni, és biztosítani a szoftver megbízhatóságát.

II. Lanterna manuális képernyőkezelés vs. GUI2 a vezérlés és MVC kontextusában

A Lanterna könyvtár a Java világában terminál-alapú grafikus felületek készítésére szolgál. Két fő megközelítést kínál az input és a megjelenítés kezelésére: az egyik a manuális Screen alapú kezelés, a másik pedig a GUI2 keretrendszer.

1. Manuális Screen-kezelés

A manuális megközelítésnél a fejlesztő közvetlenül a Screen és TextGraphics osztályokon keresztül rajzolja ki a felületet, és az inputot is saját ciklusában olvassa:

```
while(running) {
renderScreen();
KeyStroke key = screen.readInput();
updateState(key);
}
```

Ebben a modellben a controller mindenért felelős: a képernyő kirajzolásáért, a billentyűzetolvasásért és a felhasználói input feldolgozásáért.

Előnyei:

- Teljes kontroll: minden pixel és karakter a fejlesztő kezében van.
- Egyszerű, könnyen átlátható kisebb játékoknál.

Hátrányai:

- A controller túlterhelt lesz, mert a logika, az input és a render vezérlés mind ugyanott van.
- MVC-szempontból a view és az input logika összefonódik, ami csökkenti a moduláris fejleszthetőséget.
- Bonyolult a GUI elemek, fókuszkezelés és ablakok kezelése, különösen újrafelhasználható menük vagy állapotok esetén.

2. GUI2 alapú megközelítés

A GUI2 a Lanterna magasabb szintű keretrendszere, amely beépített ablakokat, panelek és komponenseket biztosít, és saját input-kezelő eseményciklussal rendelkezik. A MultiWindowTextGUI vagy BasicWindow komponensek maguk kezelik: A billentyűzet eseményeket, a fókusz váltását a komponensek között és a komponensek vizuális állapotának frissítését is

A fejlesztőnek már nem kell ciklusokat írni vagy manuálisan figyelni a billentyűleütéseket; a renderer deklaratív módon létrehozza a gombokat és paneleket, a controller pedig csak a logikát kezeli, például hogy a kiválasztott menüpont melyik AppState-t váltja.

Előnyei:

- A controller könnyű és tiszta marad: csak a logika és az állapotkezelés a feladata.
- Az MVC elv jobban érvényesül: a view deklaratív és izolált, a controller csak az eredményeket kezeli.
- Egyszerű a fókusz és a navigáció kezelése, a gombok, menük, ablakok újrafelhasználhatók.
- Automatikusan kezeli a képernyő frissítést, az inputot és az események sorrendjét.

Hátrányai:

- Kevesebb finom kontroll a terminál “alacsony szintű” részletei felett.

- Nagyobb keretrendszer overhead, ami egyszerű játékoknál felesleges lehet. Az eseményvezérelt modell miatt a logikai hibák okozhatnak rejtett problémákat, ha a controller és a callback-ek nincsenek pontosan összekapcsolva.

Kapcsolat az MVC-vel:

GUI2 esetén az MVC tisztábban érvényesül a játék állapotának követése, az automatikusan kezelt , komponensekre bontott renderelés. A controller logika pedig tisztábban marad a saját vonzáskörzetében.

Összegzés:

Manuális Screen esetén a controller túlterhelt, az MVC csak félig valósul meg. GUI2-vel az input és a render automatikus, a controller csak logikát kezel, a view deklaratív, és az MVC érvényesül. GUI2 előnyös a bővítésekhez, új menük és ablakok kezeléséhez, míg a manuális Screen egyszerű, de könnyen áttekinthetetlenné válik.