Tervezési minták egy OO programozási nyelvben

MVC, mint modell-nézet-vezérlő minta

Az objektumorientált programozás lényege, hogy a kódbázist nem egy darab egybefüggő hosszú kód alkotja, hanem több, egymással együttműködő külön objektum. Az objektumoknak különböző struktúrájuk és felelősségi köreik vannak, ettől sokkal jobban átlátható, követhető, bővíthető és javítható lesz a kódbázis.

A tervezési minták ennek a helyes, gyors és strukturált használatára adnak egy stabil guidelinet, aminek követése alapján hamarabb és rendezettebben hozhatunk létre jól működő architektúrát.

Ezek elve mentén épül fel a repoban megtalálható játék is. A struktúra több packageből áll, amik alatt funkciójuk alapján külön lettek rendezve az objectek a jobb átláthatóság és kezelhetőség érdekében.

Maga a user interaction és kommunikáció is a **Mainben** történik, amely itt a nézet és részben vezérlő szerepét is betölti. Mivel konzolos játékról van szó, itt a **Main** hibrid szerepet tölt be. Kiírja a menüt, megjeleníti a táblát, de ugyanakkor beolvassa a parancsokat és továbbítja a játéklogika felé.

Ezt a **Game** package szolgáltatja**,** ahol a **Game** a fő játéklogikáért, többek közt lépések ellenőrzéséért, játékosváltásért, szabályokért felel. A **WinChecker** pedig a győzelmi logika ellenőrzését szolgálja.

A játék állapotát a **Board package** - állapotot leíró modellréteg alatt megtalálható komponensek kezelik. A **Board** osztály, ami magát a táblát és annak a celláit kezeli, a **Cell** ami egyetlen cellát és annak tartalmát reprezentálja és **Position,** amely a koordináták kezeléséért felel. Ez a réteg a modellek adatoldala.

A játékban használt szimbólumokat az **Enums** alatti **Symbol** enum szolgáltatja, amely világosan elkülöníti az X, O és üres mező reprezentációját. Ezek az értékek a **Board** és **Game** működéséhez nélkülözhetetlenek.

A játékos adatait a **User** package alatti **Player** osztály tárolja. Ez tartalmazza a játékos nevét, szimbólumát, pontszámát és általános információkat. A játék logikájába nem szól bele, de fontos része az adatszerkezetnek és a leaderboard működésének is.

A final bosst a **Bot** package alatt megtalálható **RandomBot** nyújtja. Mr Robot mindig egy véletlenszerű, ám szabályos lépést választ. A Bot interface segítségével ugyanakkor bármikor komolyabb ellenfelet adhatunk a kódbázishoz. Nincs szükség a meglévő logika módosítására, elég egy új bot osztályt írni. Ez jól mutatja a Strategy pattern lényegét, mivel a bot viselkedését külön objektumban kezeljük, és a Game csupán a stratégia interfészével dolgozik.

Az utolsó struktúrális elem a **DB** package. Megjelenik a Repository minta, amiben motort a **ScoreRepo** szolgáltatja, amely az adatbázis műveletekért felelős. Elkülöníti az adatbázis logikát a fő rendszertől. Feladatai közé tartozik a tábla kreálása (ha még nem létezik), új győzelmek hozzáadása illetve a rendezett highscore lista lekérdezése. Egyik fő komponense, a **ScoreRecord** egyszerűen tárolja a játékos nevét és győzelmeinek számát, nincs benne logika csak adat. A rétegek fölött található a **ScoreService** amelyik összefogja a teljes leaderboard működését. Eldönti melyik győzelmek kerüljenek rögzítésre, leveszi a terhet a **Main** és game osztályról - Separation of Concerns és Dependency Inversion elvén.

Összességében a teljes program rétegzett, jól elkülönített felelősségi körökre épül, így kitűnően bővíthető, tesztelhető és karbantartható.