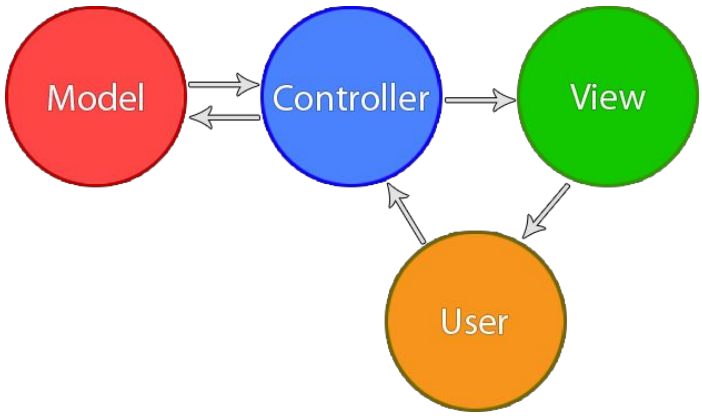
MVC tervezési minta idle játék esetén

A Model-View-Controller tervezési minta

A Model-View-Controller egy szoftvertervezési minta. Segít meghatározni és elkülöníteni a szoftver egyes részeit, feladatkörök szerint csoportosítva. Akkor alkalmazható, ha jól elkülöníthető az adatréteg, az üzleti logika és a felhasználói felület, így részei a Model (modell) réteg, a Controller (vezérlő) réteg és a View (nézet) réteg.



Model

A modell vagy adatréteg része minden az alkalmazást leíró adat és az alkalmazás perzisztálható (elmenthető) állapota. Az adatok egyszerű osztályokként a memóriában vagy adatbázisban tároltan vannak jelen.

További lehetséges funkcionalitások

- adat validálás (egy adatleíró csak megadott értékeket vehet fel, pl. játékos HP nem lehet negatív)
- egyedi adatlekérdezések (pl. az alkalmazás számára nem fontos, viszont ember számára csak adott formában olvasható táblázatok mutatása)
- automatikus adatbázis műveletek (pl. frissítés)

Controller

A Controller a vezérlési logika, összeköti a Model-t és a View-t. Reagál a View inputokra, felel az alkalmazás állapotváltozásáért és a működés alapján módosítja a modelt és frissíti a View elemeket.

View

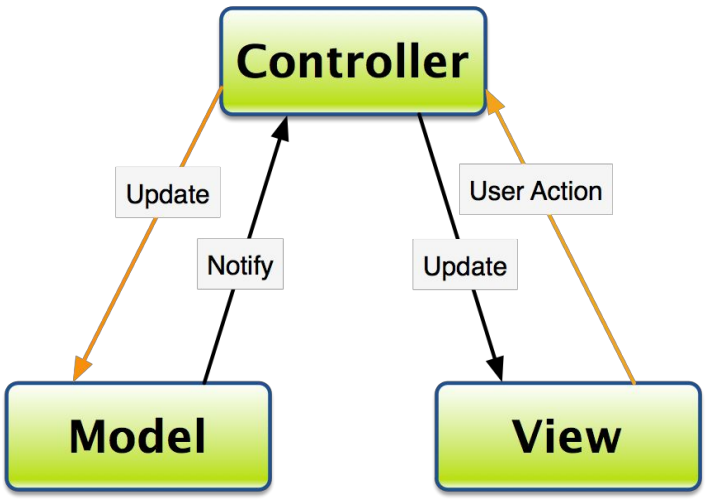
A view réteg interface modulok összessége, melyek összekötik a felhasználót az alkalmazással.

- Minden ami az alkalmazás állapotát a felhasználó által befogadható formában megjeleníti (akár képi, akár hang, akár más pl. gamepad rezgetés)
- Olyan elemek (pl. UI gomb kezelés), melyek képesek felhasználói inputok kezelésére

Működés

MVC-ben a felhasználói input, annak értelmezése és a visszajelzés a következők szerint vezethető végig:

1. A felhasználótól input érkezik a View-ba
2. A View értesíti a Controllert a felhasználói interakcióról
3. A Controller a Model figyelembevételével feldolgozza az inputot és szükséges esetben módosításra szólítja fel a Modelt.
4. Opcionálisan, ha a Model változott, arról értesíti a Controllert
5. A Controller a változásokat figyelembe véve értesíti a View-t az új állapotról
6. A View megjeleníti az aktuális állapotot
   (opcionálisan lekérdezést is intézhet a Model felé)



Előfordulhat, hogy az alkalmazás állapota a nézetektől függetlenül változik (pl. mert telik az idő), ekkor a fentiekhez hasonló módon, viszont a 3. ponttól kezdve követendő a logika.

Előnyök

- Jól szeparált feladatkörök
- Modularitás
- Bővíthetőség

Helye az architektúrában

Működése hasonló a háromrétegű architekturális minta működéséhez (adatelérési réteg - üzleti logika - megjelenítés).
https://hu.wikipedia.org/wiki/T%C3%B6bbr%C3%A9teg%C5%B1_architecht%C3%BAra

Szoftver-architektúrákra általánosan jellemző, hogy logikai mélység szerint más és más tervezési mintákat érdemes alkalmazni, így az MVC egyes rétegein belül további fejlesztési mintákat is alkalmazhatunk. Pl. ha egy játékbeli karakter a View réteg része, mert csak a játék állapotát mutatja, annak attól még lehet saját AI-ja, tehát egy saját kis MVC-ből is felépülhet.

MVC alkalmazása idle játékra

Model elemek

GameData

A statikus, a játék során nem változó játék adatok, a játékot leíró információk.
Pl. a szörnyeknek és a boss-nak mekkora sebzése van.

A GameData elemeinek szerkesztéséhez érdemes egy külső eszközt használni pl. adatbáziskezelő, saját editor, vagy Google Docs XLS, így elkülönülhet a játékleírók módosítása a játék tényleges módosításától. Ekkor természetesen szükség van egy további eszközre, amely a GameData-t a játék számára megfelelő állapotba hozza (pl. fájlba írja).

GameState

Dinamikus adatok, a játék játékos (és idő) szerinti aktuális állapota. ~Játék mentés.
Pl. a játékos 20-as szintű és 500 coinja van.
Fontos, hogy tartalmazzon verzió-információt.

A model réteg elemei Unity-ben

A model réteg elemei leginkább szimpla adatobjektumokként (POCO) jelennek meg - függvények nélküli, csak változókat tartalmazó osztályok formájában.
Pl. Player class: name, health és level attribútumokkal.

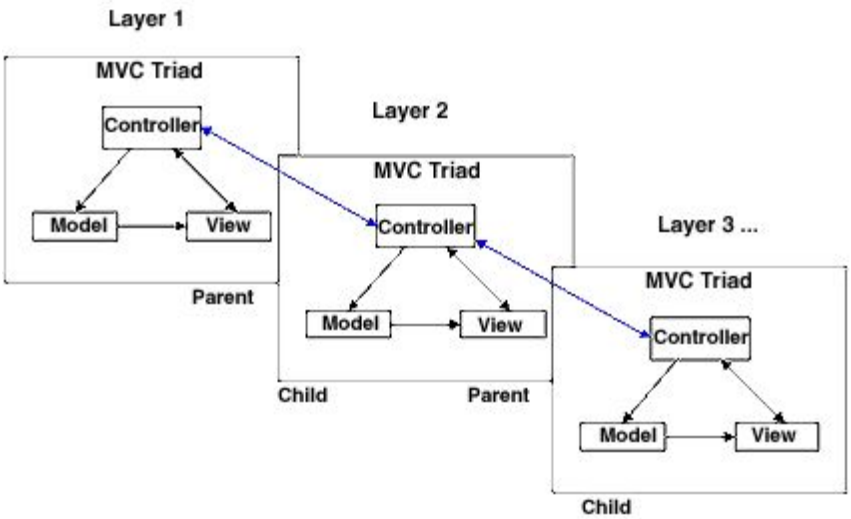
Előfordulhat azonban, hogy az alapfunkcionalitást (read/write) kiterjesztjük közvetett állapot-lekérdező és módosító függvényekkel, a könnyebb kezelhetőség érdekében.
Pl. GetExperienceForNextLevel() - azaz mennyi XP kell a következő szint eléréséhez.

Controller elemek

Minden ami összeköti a Model-t a View-val. Ide tartoznak a játékmechanika vezérlők, a játéklogika, a user input és az idő kezelés.
Pl. tapelés, currency kezelés, upgrade-k, prestige szint, achievement kezelés.

Controllerek kapcsolata

A controllerek szintjén jön létre az egyes mechanikák kapcsolata. Ha egy-egy mechanikát külön-külön rétegként képzelünk el (pl. currency kezelés réteg és karakter réteg), akkor a különböző rétegek model és view elemei nem érik el a másik réteg elemeit, csak a controllerek kommunikálnak egymással.
Pl. a játékos rányom a karakter ablakon az upgrade gombra, akkor a karakter controller erről értesül, szól a currency controller-nek, hogy változott a pénz mennyiség a karakter ára alapján, erre a currency controller módosítja a currency model állapotot, majd OK-t küld a karakter controllernek, így az módosítja a karakter model-t és megjelníti a karakter UI-on, hogy egy szinttel magasabb lett a választott karakter.



Eseménykezelés

A Controllerek közötti kapcsolat lehet beégetett, direkt 1-1 kapcsolat, azonban így elveszítenénk a szeparálhatóságot és laza csatolást, ezért érdemesebb esemény-alapú kapcsolatokat felépíteni. Az egyes controllerek tehát eseményeket definiálnak, amelyre bármely másik controller feliratkozhat. Csak addig van feliratkozva egy controller egy másik eseményeire, ameddig feltétlenül szükséges.
A C#-os event-delegate rendszer megfelelő megoldást biztosít erre.
https://unity3d.com/learn/tutorials/modules/intermediate/scripting/events

View elemek

A játék aktuális állapotát mutató nézet elemek, melyek kezelhetik a (felhasználói) inputokat.

UI

Felhasználói felület UGUI-val. Pl. panelek, tabok, gombok stb.
Két fő UI kategória

- Screen-space UI: a kameráktól és a játéktértől független helyzetű UI elemek, melyek általában mindig ugyanott találhatók. Pl. pause gomb a jobb felső sarokban.

- World-space UI: játékbeli objektumok-, játéktér- vagy kamera beállítástól függő helyzetű UI elemek. Pl. HP csík egy egység felett.

http://unity3d.com/learn/tutorials/topics/user-interface-ui

GFX

Minden grafikai tartalom, ami nem UI vagy effekt. (GFX == graphics)
Típus szerint

- 2D sprite-ok, billboardok
- 3D modellek (mesh fbx-ként)
- Textúrák
- Materialok
- Animációk

Játéklogika szerint

- Távoli háttérelemek, pl. égbolt
- Közeli környezeti elemek, pl. domborzat, panelházak
- Közeli objektumok, pl. asztal, pad
- Interaktív játékelemek, pl. powerupok, felvehető itemek
- Karakterek

Particle Systems

A részecske-rendszerek látványos effekteket jelenítenek meg.
Céljuk

- valamely játék állapot bekövetkeztének jelzése
  pl. ahova a játékos tappelt ott kis szikrák keletkeztek
- valamely játékállapot fennállásának jelzése
  pl. ha meg van mérgezve a karakterünk, akkor zöld füst veszi körül
- díszítő, vizuális hatás
  pl. animáció közben a kard suhintását fénycsóva kíséri

Audio

Az audio is a játék állapotát megjelenítő nézet.
Audio típusok

- SFX (hangok, hangeffektek)
- zene
- beszéd

MVC elemek helye Unity-ben

- Unity scene-ben GameObject hierarchia
- Az MVC elemek a hierachiába kerülnek, csoportosítva (GameObject lehet csoportosítási pont)
- A mechanikák scriptek formájában komponensekként csatlakoznak a GameObjectekhez
- Figyeljünk az egyes alrendszerek, betöltések, kapcsolat-felépítések sorrendjére, a boot folyamatra!
- Ha szükséges, oldjuk meg kódból, de inkább kerüljük a scene-en belüli direkt összeköttetéseket prefabek között!