1.Bevezetés

2.Felhasznált Technológiák

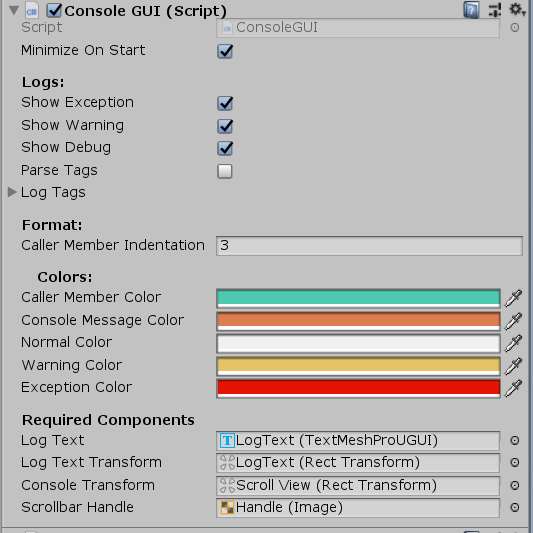
A szakdolgozat elkészítéséhez több keretrendszert is felhasználtam. Kliens oldalon első sorban a Unity3D játékmotort az alkalmazás alapjaként, Vuforia motort tárgyak felismeréséhez, valamint SqLite adatbázist a szerverről lekérhető adatok tárolására gyorsabb adatelérés, illetve offline üzem esetére. A szerver egy ASP .NET keretrendszert felhasználva készült REST API, amely MySql adatbázist használ. A kliens és a szerver is a Dapper micro ORM-et alkalmazza, hogy az adatbázisból lekért adatokat könnyen kezelhető objektumokká alakítsa. Az API emellett a Dapper egy bővítményét, a Dapper Extensions-t használja fel, mely képes az alapvető SQL műveletekhez kódot generálni típus alapján. JSON objektumok kezeléséhez a kliens, illetve az ASP .NET keretrendszer beépítve használja a JSON .NET keretrendszert. Az alkalmazás két része C# nyelvben íródott. Az alábbiakban ezeket a technológiákat részletesen ismertetem.

2.1. Unity3D

A Unity3D az egyik, ha nem legismertebb játék motor napjainkban. Népszerűségét többek között köszönheti árazásának, a szerkesztő művész barát kialakításának, a támogatott platformoknak, valamint, hogy játékfejlesztés legfőbb technikai kihívásait (fizika, képalkotás stb.) átvállalja a keretrendszer felhasználójától mégis nagyon flexibilisen használható. Az virtuális, kiterjesztett, illetve kevert valóság alkalmazások fejlesztésében is a Unity a legnépszerűbb választás, mivel mobil ezeket az alkalmazások főként mobilokra fejlesztik.

2.1.1. GameObject-ek és komponensek

A keretrendszer flexibilisségének és a kódok újrahasználhatóságának oka a kódok rendszerezésére választott megközelítés. Minden entitást a játéktérben egy GameObject reprezentál. Egy GameObject tulajdonképpen csak egy szervezési egység, egy tároló, amely a játéktérben futtatható kódokat a MonoBehaviour-öket vagy másként komponenseket magában foglalja. A GameObject-en keresztül a komponensek egymást is képesek elérni. Alapvető komponens például a Transform, mely az objektum helyzetét, forgását, skálázott méretét határozza meg illetve kezeli a hierarchiában alatta elhelyezkedő Transform-okat és minden objektumon kötelezően szerepel. A GameObject-ekhez tetszőleges számú komponenst adhatunk így könnyedén személyre szabva annak viselkedését. A komponensek publikus, illetve „Serializefield” attribútummal ellátott adattagjainak a szerkesztőből adhatunk értéket, így könnyedén az adott helyzetre lehet szabni viselkedésüket. Fontos megjegyezni, hogy nem képes mindent szerializáni, ezeket nem a mezőket nem jeleníti meg a szerkesztő, ilyenek például a property-k, list-ek, dictionary-k stb.



A képen a ConsoleGUI komponens beállításai láthatóak.

2.1.2. MonoBehaviour szerkezete

Ahhoz, hogy a kódunkat egy GameObjecthez tudjuk csatlakoztatni annak a MonoBehaviour osztályból kell öröklődnie. Egy MonoBehaviour életciklusa során különböző események következnek be, ezekre reagálva tudjuk a kódunkat futtatni. Ilyenek például az Awake, mely közvetlenül a komponens létrejötte után hívódik meg, a Start mely az ezt követő képkocka előtt vagy az Update, amely minden képkocka kirajzolásakor. A Unity az adott eseményekhez tartozó függvények aláírását keresi az osztályunkban, hogy megtudja hívni azt. Ha egy ilyen nevezetes függvényt használunk, azt a Visual Studio kék kiemeléssel jelzi.

