Budapesti Műszaki és Gazdaságtudományi Egyetem
Villamosmérnöki és Informatikai Kar
Automatizálási és Alkalmazott Informatikai Tanszék

Szita Ádám

FELÜGYELŐ RENDSZER FEJLESZTÉSE ÚJGENERÁCIÓS MOBIL OKTATÓJÁTÉKOKHOZ

KONZULENSEK

Dr. Forstner Bertalan
Szegletes Luca

Tartalomjegyzék

Összefoglaló

A napjainkban piacra kerülő mobil eszközöket és ezek újabb és újabb funkcióit már-már nagyon nehezen tudjuk követni, a folyamatos fejlesztéseknek és bővítéseknek hála mostanra már aligha létezik, olyan része az életünknek melyben ne tudnának ezek a készülékek segítségünkre lenni. Nagy kereslet van jelenleg a minél többet tudó okostelefonok iránt, melyek egyre gyorsabbak, egyre több tárhellyel rendelkeznek és egyre szebb képet adnak. A tanulási módszerek és technikák ennek a technológiának köszönhetően nagymértékben korszerűsíthetőek, modernizálhatóak, hogy azok minél hatékonyabbak lehessenek.

Elvégzett munkám egy projekt része, mely az előbbi két területet próbálja megcélozni úgy, hogy tulajdonképpen egy olyan eszközt, rendszert valósítsunk meg mobil technológiával, mely tanulási nehézségekkel küzdő gyerekek számára kínál egy újfajta módszert. Célunk az, hogy a játszva tanulás lehetőségét biztosítsuk nekik a mai piaci forgalomban is megtalálható eszközökkel, így például Android platformú tablet-ekkel. Játék közben a rendszerünkkel a játékosnak nem csak a játékkal történő interakcióját kívánjuk rögzíteni, hanem az aktuális mentális állapotát is vizsgáljuk, mely befolyásolni tudja magát a játékmenetet is. Egy fejre könnyen feltehető headset méri a játékos agyhullámait, és ad visszajelzést a játéknak (biofeedback), mellyel akár a játék nehézsége is befolyásolható.

A koncepció összetettsége miatt szükség van egy felügyelő rendszerre is, ami információt szolgáltat a játék és a játékos aktuális állapotáról egy tanárnak, mentornak. Ilyen információ többek között a játékszintű események, a felhasználó érzelmei, pillanatkép az aktuális játékállapotról, vagy akár az EEG headset-ből kinyert egyéb adatok. Munkám során létrehoztam egy olyan felügyelőalkalmazást mely alkalmazkodni tud bármilyen a rendszerünkhöz írt oktatójátékhoz és hatékonyan ad képet egy felügyelőnek a pillanatnyi helyzetről. Ezen kívül a rendszer egyéb komponenseinek megtervezésében és fejlesztésében is részt vettem (és jelenleg is részt veszek), melyeket majd szintén részletezni kívánok.

Abstract

Nowadays the market is flooded with ever-renewing mobile devices, whose new functions and features may seem hard to follow. Due to the intense development and great investments in this field, one can hardly find a part of everyday life where these devices cannot be at our aid. There is a substantial demand for smart phones, which come with new and new features, more processing speed, more storage and better looking displays. With the help of this technology learning methods and techniques can also be excessively modernized, so that they become even more efficient.

Currently I take part in a project, in which these two scientific fields are targeted in a way that we are developing a system on mobile technology, which can be used as a tool to support a new method of learning for children with learning disorders and difficulties. We aim at providing the opportunity of learning through playing with the help of devices currently available on market for example Android-based tablets. During the gameplay we not only intend to record the interactions between the player and the game, but we also would like to examine the actual mental state of the player, which can also be used to alter game scenarios. An easy-mountable headset is placed on the player that measures brain activity, and may influence the difficulty of the current game. This is called biofeedback.

Because of the great complexity that comes with this method there is a need for a supervisor system that provides information about the current state of both the player and the game to a teacher or mentor. This information may consist of various types of data such as the current feelings of the player, or details about the current game session. The work I carried out for this project so far includes the creation and development of a supervisor system that effectively monitors the game and the player. Furthermore I have taken part in the initial planning of the whole architecture and developed other smaller parts of it, which I am going to provide details about.

1 Bevezetés

A mobiltechnológia manapság már olyan nagy piaci részesedéssel bír, hogy egyre több és több cég valamint intézmény próbálja meg ezen új eszközök segítségével fejleszteni saját üzletvitelét. Az ok egyszerű: napjainkban már megfizethető áron kaphatunk kézhez olyan mobilkészülékeket, melyek funkcionalitásának sokrétűsége szinte már egy számítógépével is felérhet. A kompaktság pedig tovább bővíti a technológia előnyeit, hiszen egy kis telefont vagy egy tabletet bármikor magunknál tarthatunk, és bárhova magunkkal vihetjük. Mivel ez egy meglehetősen felkapott informatikai ágazatnak számít, jelentős mértékű fejlesztési munka támogatja jelenleg, ezért rendkívül gyakran jönnek ki piacra az egyre gyorsabb és bővebb funkciójú eszközök.

Nem szabad megfeledkezni azonban a hátrányokról sem, ugyanis fejlesztői szemszögből nézve egy jó minőségű szoftver elkészítése ilyen mobil eszközökre koránt sem triviális feladat. A legnagyobb akadály, amibe az ember beleütközik, az az asztali számítógépekkel szembeni teljesítménykülönbsége e készülékeknek. Bár jelenleg már kapható több (kettő vagy akár négy) magos processzorral szállított eszköz is, azért még mindig jócskán elmaradunk egy PC képességeitől, mind feldolgozási idő, mind pedig rendelkezésre álló memória területén. Tovább nehezíti dolgot az a tény, hogy tulajdonképpen nincsen végtelen kapacitású energiaforrásunk, hiszen a készülékek mobil voltából adódik, hogy idejük nagy részében akkumulátorukról vannak táplálva. Ezen pedig csak tovább ront, hogy egyre nagyobb kijelzők és processzorok természetesen mindig egyre többet is fogyasztanak.

A mobil platformokra való fejlesztés tehát nem egyszerű munka, azonban kifizetődő, mert a lefejlesztett újabb és újabb alkalmazásokkal egyre több funkció ellátása szakadhat el az otthoni asztali számítógépen való elvégzéstől, legyen az szórakozás, munka, internetes bankolás vagy egyéb információelérés.

Bár a legtöbb fejlesztés még inkább egyszerűbb alkalmazások és szórakozást nyújtó játékokra koncentrálódik, érdemes lehet erőfeszítéseket komolyabb, akár különböző tudományos területek együttműködését is megkövetelő alkalmazások, illetve alkalmazásrendszerek tervezésébe és fejlesztésébe fektetni. Így például egy rendkívül érdekes, és még annál is inkább jótékony beruházás lehet olyan oktatóprogramok

elkészítése mobil platformokra, melyek tanulási nehézségekkel küzdő gyermekek számára kínálnak egy új, modern módszert fejlesztésük támogatásához.

Sajnos tény, hogy napjainkra jelenetős nagyságúra tehető az olyan gyermekek száma, akik különböző neurális működésbeli hiányosságok miatt speciális igényű oktatásra, illetve fejlesztésre szorulnak. Az ilyen tanulási nehézségek közé sorolható például a dyslexia is, mely nagyban megnehezíti a speciális oktatási igényűek számára a normál módú oktatásba való beilleszkedést, illetve annak tempója tartását.

Néhány intézmény pont arra szakosodott, hogy az emiatt hátrányban szenvedőeket felkarolja és speciális módszereket alkalmazó oktatást, felzárkóztatást és fejlesztést biztosítson számukra. Ezen feladatok típusa sokrétű lehet, léteznek a logikus gondolkodást elősegítő tevékenységek, melyben például több kategória elemeit kell a gyermekeknek csoportosítaniuk. Ezek megvalósítása egészen eddig többnyire papír és toll, illetve más modern technológiát nem igénylő eszközökkel történt. Természetesen ez bonyolultabb adminisztrációt von maga után, valamint kevésbé informatív visszajelzéseket képes csak nyújtani. A mi célunk, hogy karöltve az olyan tanulási nehézségekkel (is) foglalkozó intézményekkel, mint például a Meixner Alapítvány, új fejezetet nyithassunk e speciális fejlesztésre kialakított feladatok alkalmazásában: a játékos feladatokat mobil technológia felhasználásával, modern táblagépeken tálaljuk a gyermekeknek.

Ezzel élvezetesebbé és könnyebben kezelhetővé válik a tanulás a speciális feladatok alkalmazásával, hiszen egyrészt nincs másra szükség csak egy tabletre, melyen sokkal látványosabban és információgazdagabban ábrázoltathatjuk az elvégzendő feladványokat, másrészt pedig jobban leköthetjük a gyermekek figyelmét, mint például a régebbi módszert képviselő egyszerű ceruza és papír alkalmazásával.

Ez a koncepció már önmagában is rendkívül hasznosnak hangzik, mi azonban nem kívánunk itt megállni és ennyivel beérni, szeretnénk ugyanis bővíteni azt az információmennyiséget, melyet a gyermek a feladat elvégzésével szolgáltat: agyhullámok mérésével a belső mentális állapotot is vizsgálni szándékozzuk. Ezt két fő ok miatt is hasznos megtenni. Az első, hogy az agy állapotának folyamatos rögzítése és a feladat elvégzésével való párosítása után később adatbányász módszerek felhasználásával egyfajta tanulási modellt lehet kialakítani. Egy ilyen modell alapján továbbfejleszthető és célzottan bővíthető a speciális oktatási feladatok repertoire-ja.

A másik ok a manapság egyre felkapottabb terület, a biofeedback használata a feladatok játékmenete alatt. A biofeedback tulajdonképpen az alany fiziológiai jellemzőinek (agyi aktivitás, szívritmus, stb.) információként való visszacsatolása úgy, hogy ezek befolyásolják az alany jövőbeli tevékenységét. Mivel a mi esetünkben a játékos tanulást szeretnénk befolyásolni, olyan mintákat keresünk az agyi aktivitásban, amelyekből következtethetünk, hogy milyen mértékben kellett gondolkodásra és logikus következtetésre használni az agyat. Ezt az információt aztán visszacsatolva a játékba befolyásolhatjuk a következő játékpálya nehézségét, vagy egyéb játékot leíró paramétereket, hogy az minél inkább a tanulás hasznára váljon.

A következőkben bemutatom az erre a koncepcióra fejlesztett, az Automatizálási és Alkalmazott Informatikai Tanszék által indított projektet, mely az InnoLearn nevet viseli. Ismertetem a projekt feladatainak megosztását a fejlesztők között, és bemutatom a projekt komponenseit. Továbbiakban röviden ismertetem a felhasznált technológiákat és elméleti ismereteket. Részletesen kitérek az általam tervezett és fejlesztett részekre, végigjárva a szoftverfejlesztési folyamat tervezési fázisát, majd végül a kivitelezés során tapasztalt nehézségeket és érdekességeket is prezentálom.

2 Az InnoLearn projekt

A projektet az Automatizálási és Alkalmazott Informatikai Tanszéken a 2012/13-as tanév őszi félévében indítottuk el. Eredetileg Önálló laboratórium 1 illetve 2 tárgyak témájaként, jelenleg pedig Diplomatervezés keretein belül dolgozom tovább a projekten. Bár a projekt egyes alegységeinek prototípusai már működőképesek, pillanatnyilag is folyik még a fejlesztés és a tesztelés az elkészített elemeken. A projekt aktuális állapota és elérhető funkciói már többször is bemutatásra kerültek a partnerintézmények számára, és minden alkalommal elnyerte a tetszésüket az általunk végzett munka. Folyamatosan fogadjuk az intézmények javaslatait, kéréseit, továbbá az általuk biztosított oktatójátékok elkészítése is a napokban végzett munka részét képzi.

Két partnerintézményt emelnék ki, akikkel együttműködve dolgozunk az InnoLearn projekten:

- Meixner Alapítvány: Az alapítvány tanulási nehézségekkel (például dyslexia) küzdő gyerekek felzárkóztatásával foglalkozik, játékos feladatokkal fejleszti a gyerekeket, hogy megelőzze, illetve kezelje e diszfunkciók tüneteit. Számukra önálló, de kontrollált feladatgyakoroltatásra készítjük a rendszert.
- Zeneakadémia: Egy a Zenepedagógiai Tanszékkel közös TÁMOP[1] projekt keretén belül, gyerekkortól kezdve egészen a haladó hangszeres tanulásig nyújt az InnoLearn rendszer ritmus-érzék fejlesztő és egyéb zenei oktatójátékokat

2.1 Az architektúra

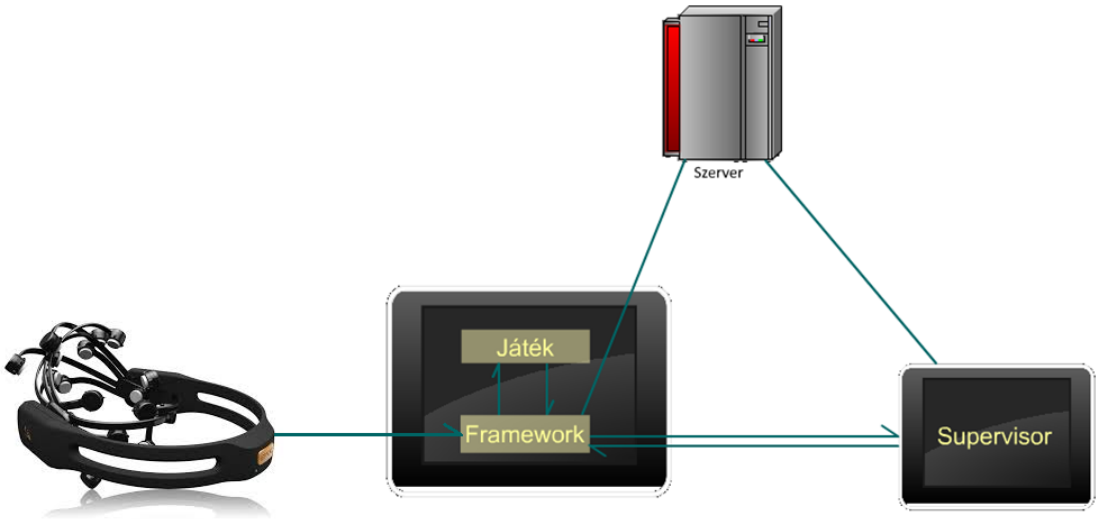
Az általunk fejlesztett projekt alkalmazásrendszere a következő komponensekből áll:

- Keretrendszer (Framework)
- Felügyelő (Supervisor)

[1] Társadalmi Megújulás Operatív Program

- Webes szerver
- A speciális, fejlesztő oktatójáték
- Szenzorok

Ezek közül a keretrendszer és a játék közös, a felhasználó (gyermek) táblagépén fut, a szerver egy Windows Azure alapú felhőbeli szolgáltatás, a felügyelőrendszer pedig egy mentor vagy tanár által használt táblagépen fut. A szenzorok közül pillanatnyilag egy EEG mérőberendezést használunk. Ez egy agyhullámok mérésére szolgáló, könnyedén fejre helyezhető eszköz, mely később majd bővebb bemutatásra fog kerülni. Ezek együttműködése és egymáshoz való viszonya a 2-1 ábrán figyelhető meg.



2-1 Ábra

A jelenleg megvalósított változatban a mobil technológiát az Android platform adja, így Android alapú alkalmazásként kerültek megvalósításra mind a keretrendszer, játékok és a felügyelő egység is. Emiatt a fejlesztés túlnyomó részben Java nyelven folyik, eltekintve a keretrendszeren futó natív (C++) kódrészletektől, valamint a szervertől, mely .NET alapokra támaszkodik.

2.1.1 A keretrendszer és az oktatójáték

A keretrendszer tulajdonképpen egy köztes rétegként szolgál a projektben résztvevő többi komponens között, hiszen (ahogy az a 2-1 ábrán is jól látszik) ő látja el az információközvetítő szerepét. A framework tulajdonképpen egy Android service, ami

a játékos tabletén a háttérben futva adatrögzítést és kommunikációt végez, mégpedig eseményalapú információkezelést valósít meg.

Tervezésétől kezdve azt tartottuk szem előtt, hogy minél absztraktabb módon valósítsuk meg a funkcionalitását, hogy képes legyen többfajta játékot is kezelni. A játékoknak meg kell valósítaniuk a keretrendszer által diktált interfészt, mellyel lehetővé tesszük, hogy a framework befolyásolhassa a játékot, valamint adatokat kaphasson tőle. Így például a játék indítását, nehézségének állítását a framework végzi, és a játék által generált általános valamint játék-specifikus információkat is fel tudja dolgozni. Ehhez a játék elején végbemenő regisztrálási folyamat szükséges, melyről bővebben a felügyelőrendszer bemutatásánál írok (hiszen ott is ugyanezt a koncepciót kellett követni).

A játék eseményein túl fogadja még a keretrendszer az EEG eszköz által szolgáltatott agyi aktivitást leíró eseményeket is. Az EEG eszköz USB-n keresztül van a táblagéphez illesztve, és natív forráskód állít elő a nyers adataiból a keretrendszernek értelmezhető java objektum szintű eseményeket. Az előfeldolgozást követően másodperc gyakorisággal veszi a szenzoreseményeket, melyeket tárolás mellett arra is felhasznál, hogy a játékot befolyásolja, továbbá a felügyelőrendszernek továbbítsa.

Egy játékmenet után a játék során rögzített információt HTTPS kapcsolaton keresztül fogja továbbítani a felhőben elhelyezkedő szerver felé, ahol az adatok végleges perzisztálásra kerülnek.

2.1.2 Felügyelő alkalmazás

A felügyelőrendszer egy pedagógus által használt táblagépen futó Android alkalmazás, melynek felelőssége az alábbiakra terjed ki:

- A játék irányítása
- Esetleges hibás események elvetése végleges rögzítés előtt
- Külső események hozzáadása a játékmenet adataihoz
- Játékos felügyelete: játékbeli aktuális előrehaladás, játékos mentális állapota

A supervisor egy olyan felületet biztosít, mellyel könnyedén nyomon követhető a futó játék által küldött információhalmaz. Léteznek előre bejegyzett események,

melyeket a keretrendszer akkor továbbít a játék felől, ha például a játék elindult, új játékszintet kezdett meg, szint végéhez ért, stb. Ezeken az egyszerűbb eseményeken felül, csakúgy, mint a framework esetén képes a felügyelőalkalmazás előre nem definiált, játék-specifikus esemény fogadásra is. Mivel egy ilyen eseményhez nem tudni előre, hogy milyen adatokat szállít, ezért a játék megkezdésekor lezajlik egy regisztrációs időszak, amikor is a keretrendszer a már hozzá beregisztrált játékeseményeket továbbítja a supervisor számára.

A felügyelő képes továbbá parancsok küldésére is, melyekkel a játékot irányíthatja, és felülbírálhatja a pedagógus a keretrendszer eseményeit. Ezek a parancsok szintén két csoportra, előre bejegyzett valamint dinamikus, azaz futási időben beregisztrált típusokra oszthatóak.

Az egyik legfontosabb funkciója a supervisor alkalmazásnak az nem más, mint a játékból érkező pillanatkép kirajzolása a grafikus felületre. Talán nincs is annál egyszerűbb módja a játékos előrehaladásának felmérésére, mint rátekinteni arra a képre, amely előtte is megjelenik. Ezt azért hasznos távolról, egy táblagépről, és nem pedig a gyermek válla fölött megtenni, mert így nem befolyásolja a mérés és a játék kimenetelét a játékosban manifesztálódó megfigyelés-tudat.

A felsorolt funkciókon kívül képes még a supervisor egy könnyen értelmezhető ábrán összegyűjtve megjeleníteni azt is, hogy megfelelő jelerősséggel érkeznek-e az EEG headset-ből származó adatok. Ez nagy segítség lehet a felügyelő tanárnak, hogy a gyermek fején mely elektródákat kell még igazítania a jobb jel-zaj viszonyú feszültségértékek vételéhez.

2.1.3 Szerver

Az adat áramlásának szempontjából projektünk jelenlegi állása szerint a szerver komponens képzi az utolsó állomást, ahol a játékmenet alatt nyert összes információ – többek között játékesemények, EEG nyers adatok, játékos teljesítménye – végleges tárolásra kerül. A szerverben tárolt adatokat egy későbbi felhasználásra, az adatbányászat által kinyert statisztikai információk és modellek felállításra tartjuk majd meg. Mivel itt potenciálisan hatalmas volumenű adatról van szó, ezért kezdetektől fogva a számítási felhőben való implementálás útját preferáltuk. A Microsoft Windows Azure nevű szolgáltatására esett a választás, melyben így egy cloud szolgáltatásként fut az InnoLearn szerver.

A szerver így elérhető interneten, és a keretrendszer egy játékmenet befejezése után – amennyiben van éppen internetkapcsolat – HTTPS kapcsolaton keresztül eléri azt, és feltölti az adatokat, melyek egy SQL adatbázisban kerülnek perzisztálásra. További funkcionalitása a szervernek, hogy általa végezi el a projekt többi komponense az autentikációt. A tárolt adatok megtekintésére, valamint a felhasználók, intézmények adminisztrációjára a szerver egy webes felületet nyújt, melyre belépve grafikus felületen vizsgálhatóak meg ezen információk.

2.2 Munkamegosztás

A csapatban négyen dolgozunk szoftverfejlesztőként, segítségünkre egy docens és két PhD hallgató van, akikkel hetente tartunk konzultációkat. A projekt indítása utáni kezdeti időszakban mind, közösen vettünk részt az alkalmazásrendszer megtervezésében, a szükséges komponensek és közöttük lévő kapcsolatok azonosításában. Továbbá ekkor határoztuk meg a főbb technológiai irányokat is a későbbi implementálási fázishoz.

Az adatstruktúra kezdeti kialakítása után a következő felbontásban osztottuk el magunk között a további tervezési és implementálási munkákat:

- Ketten együtt dolgozzák ki a keretrendszer és a Meixner-féle játékok részletes terveit és implementálják prototípusát.
- Mivel az EEG eszköz illesztése Android platformú táblagéphez bonyolult lépés, egy harmadik társam dedikáltan ezzel foglalkozik.
- A felügyelőrendszer, valamint a szerver tervezése és megvalósítása az én feladatom, mindkét komponens szükséges funkcióit implementálni kell, továbbá illeszteni őket a keretrendszerhez. Jelenleg e komponensek fejlesztése mellett a keretrendszerhez készítek egy olyan modult, mely az EEG-ből beérkező agyi aktivitási adatok alapján befolyásolja a játék következő szintjének nehézségét.

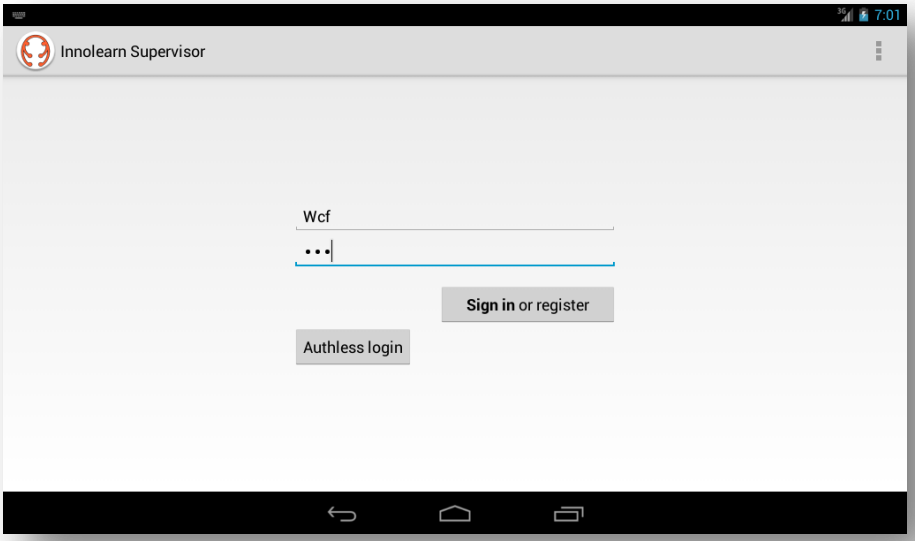
A fenti főbb pontokon kívül a felmerülő kisebb játékfejlesztési és egyéb elvégzendő feladatokat megosztjuk egymást közt.

2.3 A prototípusok

Ebben a bekezdésben nagyvonalakban bemutatom az általam elkészített komponensek prototípusainak néhány képernyőképét. A következő fejezetekben ezeket részletesen bemutatom a tervezési fázistól végigvezetve.

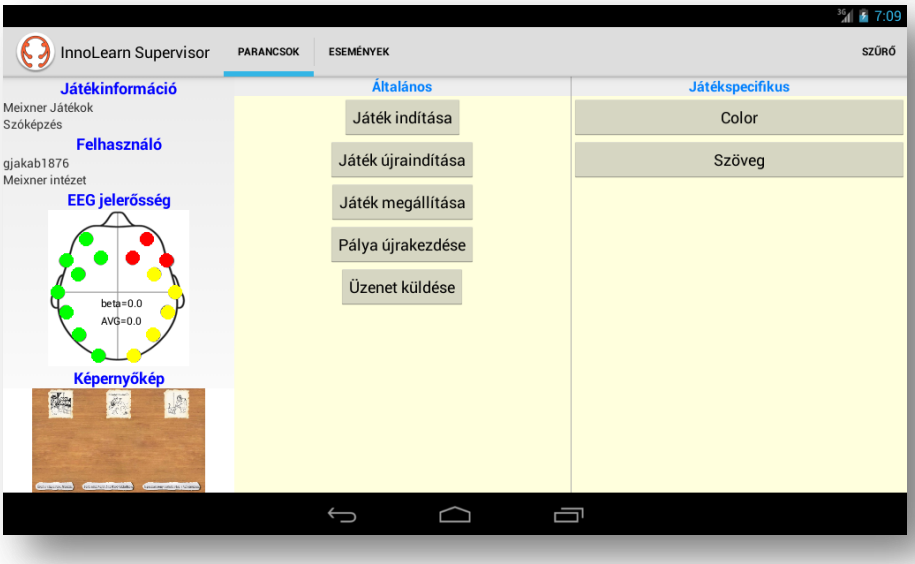
2.3.1 Felügyelőrendszer

Az alkalmazást elindítva a 2-2 képen látható bejelentkező képernyő fogadja a tanárt/mentort. Itt az *Authless login* gomb tesztelési célokat szolgál, szerverhez való csatlakozás és autentikáltatás nélkül is továbbengedi a tesztelőt.



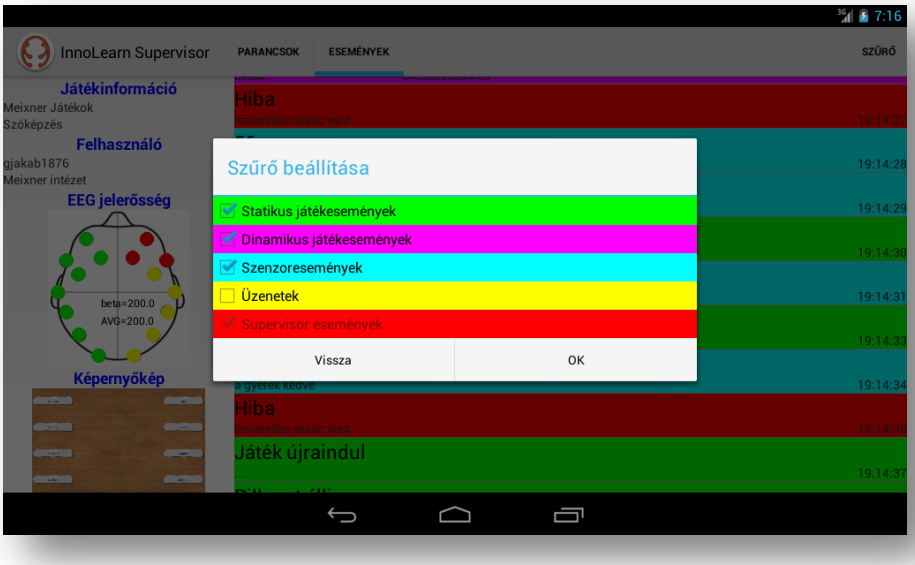
2-2 Bejelentkező képernyő

Sikeres belépés után várakozó állapotba kapcsol a rendszer: a keretrendszer csatlakozási kérelmére várunk. A játékos táblagépén az EEG vevőegység csatlakozásakor elindul a keretrendszer, így csak arra van szükség, hogy a játékot elindítsa. Ekkor a keretrendszer csatlakozik a várakozó supervisor-hoz, és a 2-3 ábrán látható főképernyő tárul a felügyelést végző tanár elé. Itt információ jelenik meg a játék aktuális állapotáról, és a *parancsok* fül alatt megtalálhatóak a játékmenetet befolyásoló parancsküldő vezérlők. Az *események* fül alatt a keretrendszerből érkező események (játék által generált információk, játékos állapotát leíró adatok) listája jelenik meg. Ez látható a 2-4 képen, melyen az események listájához elkészített szűrő beállítás ablaka is megfigyelhető.



2-3 Főképernyő

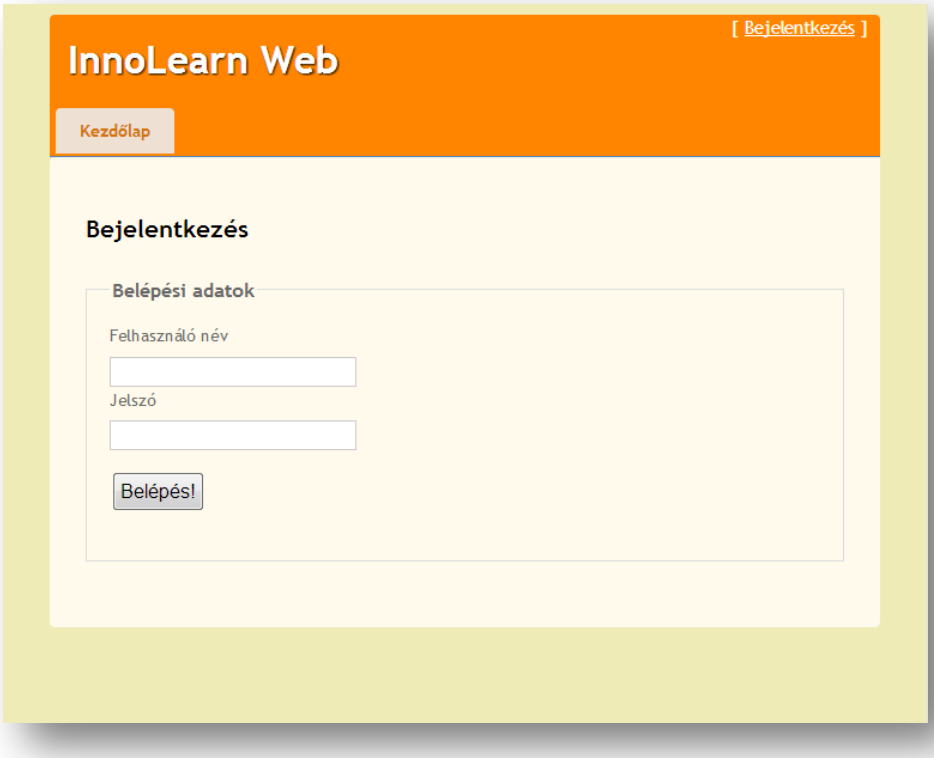
A főképernyőn megtekinthető a bal alsó sarokban a játékos aktuális tevékenysége is, a keretrendszer ugyanis a futó játékról másodpercenkénti gyakorisággal pillanatképet küld a supervisor számára. A képen egy Meixner játékállapot látható éppen. A kis képre tapintva a pillanatkép teljes képernyős ablakban jön elő.



2-4 Eseménylista

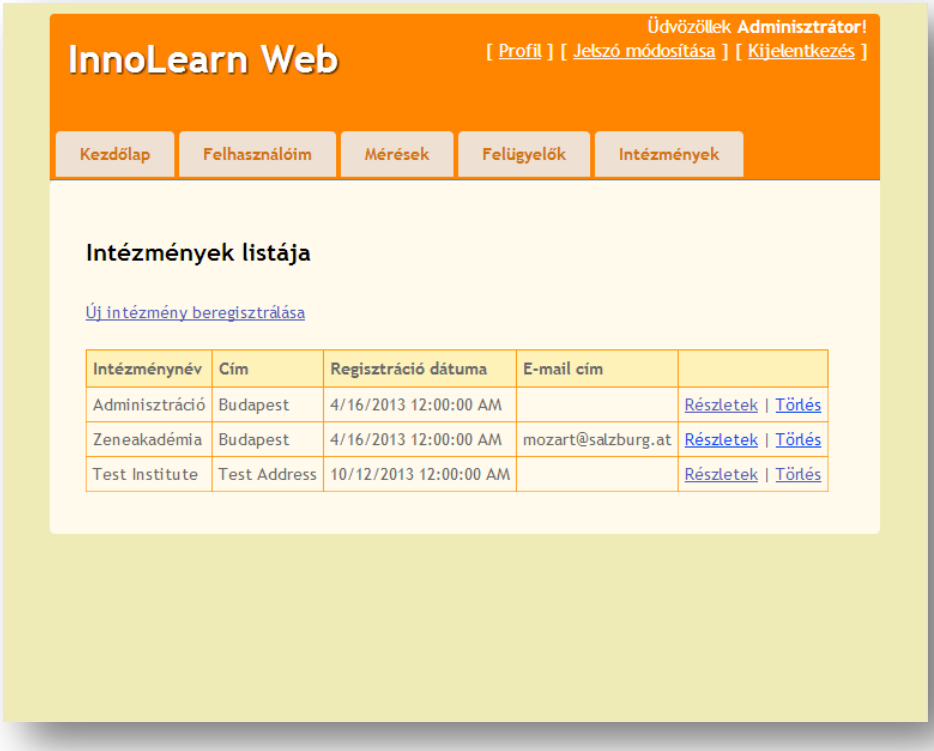
2.3.2 Szerver

A szerver alkalmazás webes része a 2-5 képen látható bejelentkező oldallal tárul a felhasználók elé. Belépés után a belépett felhasználó jogosultsági szintjének megfelelően (lásd 4.2.1 fejezet) jelennek meg az elérhető funkciók. A fő funkciók közé az intézmények, felügyelők, felhasználók menedzselése, valamint a felhasználókhoz társított feltöltött játékmenet adatok megtekintése sorolható.



2-5 Szerver beléptető weblap

A 2-6 képen az látszik, ahogy az Admin felhasználó belépése után a regisztrált intézmények kezelőfelületén listázva látja az intézmények adatait. Itt lehetőség van új intézmények felvételére, adataik módosítására, valamint intézmények törlésére. A felügyelők és felhasználók kezelőfelületén hasonló funkcionalitással rendelkezik a szerver. A feltöltött játékmenet adatok a *Mérések* menüpont alatt érhetőek el. Az egyes mérésekhez a szerverre perzisztált események listája is megtekinthető.



2-6 InnoLearn web funkciók

3 Felhasznált technológiák és ismeretek

Ebben a fejezetben rövid ismertetőt adok azokról a technológiákról és szabványokról, amelyeket a keretrendszer és a szerver tervezéséhez és implementálásához kellett használni. Továbbá dióhéjban bemutatom az általunk használt EEG eszközt, amellyel a biofeedback-et megvalósítjuk. Ehhez kapcsolódóan ismertetem azokat a biológiai és jelfeldolgozási tényeket, amelyekre támaszkodtam a fejlesztés során.

3.1 Mobil technológia – Android

Ahogy azt már az eddigiekben is említettem, azok a projektben megvalósított alkalmazások, amelyek táblagépekre készültek, Android platformon implementáltak.

3.1.1 Választás okai

Az egyik legfontosabb érv az Android által nyújtott mobil technológia választása mellett, hogy az Android operációs rendszer nagymértékben platform-független. Versenytársaival szemben egyszerűbben kapcsolhatóak hozzá az olyan külső eszközök, mint a mi esetünkben az EEG eszköz, nem szükséges a Google-től extra licenszet kérni, elég csupán az illesztési feladatot kivitelezni.

Megfelelő szintű SDK és NDK támogatás adott, és a fejlesztésben az is sokat segíthet, hogy a rendszer debütálása óta eltelt idő alatt rengeteg dokumentáció, know-how és egyéb probléma-megoldási leírások is készültek.

3.1.2 A platform

Dióhéjban összefoglalva az Android egy unix kernel felett működő operációs rendszer, mely mobil eszközökön nyújt egy rétegekből álló szoftver stack-et. Jelenleg a Google fejleszti és folyamatosan jelennek meg a rendszer újabb verziói.

Az Android platformra készült alkalmazások saját virtuális gépükön a Dalvik VM-en futnak. Az alkalmazásokat Java programozási nyelvben készítjük el, esetleg kiegészítve natív, C++ kódrészletekkel, melyek teljesítményoptimalizálási szempontból használatosak.

3.1.3 Hálózati kommunikáció

Az Android operációs rendszert használó mobil eszközök szinte kivétel nélkül támogatják a vezeték nélküli helyi hálózat (WLAN, vagy WiFi), valamint a Bluetooth hálózati adatátviteli megvalósításokat. Az InnoLearn projekt keretében egyelőre a WLAN megoldást használjuk a komponensek közti kommunikáció létrehozására.

Játékmenet alatt – mivel az eszközök mind egy légtéren belül fognak működni – a keretrendszer és a felügyelőrendszer ugyanazt a hálózatot fogja használni, így egymáshoz képest ők helyi elérhetőségben vannak. Közöttük a kommunikáció a következőképpen tagolódik:

- Felderítési szakasz: UDP használatával
- Kiépített kapcsolati szakasz: TCP használatával

Android platformon lehetőség van mindkét szállítási rétegbeli kommunikációs protokollt használni, ezeket a java.net package-ben implementált osztályok segítségével használhatjuk fel saját céljainkra. A lényegi különbség a TCP és UDP protokollok között a következő:

	TCP	**UDP**
Sorrendhelyesség	A TCP csomag fejlécében sorszám biztosítja a csomagok megérkezésének sorrendjét.	Az UDP nem vállal garanciát a sorrendhelyes csomag-továbbításra
Megbízhatóság	Garanciát vállal az elküldött csomagok megérkezésére	Nem biztos, hogy megérkeznek az UDP által csomagolt adatrészletek
Overhead[2]	Nagy, egyrészt a TCP fejléc mérete (minimum 20 byte), másrészt a nyugtázás miatt	Kicsi, az UDP egy úgynevezett lightweight pro-tokoll
Broadcast[3]	Nem támogatott	Támogatott

[2] Az összes hálózatra kerülő adat nem hasznos része (például fejlécmezők)

[3] Csomagok elküldése az alhálózat összes aktív végpontjára

A broadcast módon küldött csomagokra azért van szükség, hogy a kapcsolatfelvételt a kommunikáló felek IP címeinek előre való tudása nélkül is kivitelezhessük. Ilyenkor a küldő egy rövid üzenetet küld, amit a fogadó fél megkapva megismeri a küldő IP címét, és így kezdeményezheti a TCP kapcsolat kiépítését. A supervisor által megvalósított kommunikációkiépítést majd a 4. fejezetben ismertetem részletesebben.

3.2 .NET

A projekt szerver komponense .NET alapon megvalósított alkalmazás, mely a Microsoft által üzemeltetett Windows Azure felhőben fut. Röviden bemutatom azokat a technológiákat, amelyekre az elkészített felhőszolgáltatásom támaszkodik.

3.2.1 MVC

A webes felület MVC 3.0-val készült el, mely egy olyan webes alkalmazási keretrendszer, ami az MVC (model-view-controller) szoftverfejlesztési mintán alapszik. A szoftver ilyen rendszerű szervezésében három rétegre oszlanak fel a program funkciói:

- View: megjelenítésért felel, a Model-ből kér információt, melyet a felhasználó számára jelenít meg
- Model: leírja az adatstruktúrát, tárolja az információt
- Controller: általa módosítható a tárolt információ, a Model-t értesíti a változtatásokról

A weblapok címét megadva egy úgynevezett Router dönti el, hogy mely Controller-t kell meghívni, és az milyen View-val, azaz melyik weblappal térjen vissza. A lapok sablonból történő generálását Razor megjelenítési motor végzi.

3.2.2 Entity framework

A szerver egy szintén Windows Azure-ban működő SQL szervert használ az adatok tárolására és lekérdezésére. Az objektum-relációs leképezést az Entity framework végzi. Ez úgynevezett *DB First* módon – azaz a táblák meta adatait a már meglévő adatbázisból kiemelve– C# osztályokat generál az egyes táblákhoz, az osztály

példányait pedig a táblák egy-egy sorának felelteti meg. A leképzett osztályhierarchia fogja képviselni a modellt az MVC mintában.

3.2.3 WCF

A WCF, azaz a Windows Communication Foundation[4] szolgáltatás-orientált alkalmazások elkészítéséhez biztosít egy közös kommunikációs keretrendszert. Segítségével definiálhatjuk az adatátvitel paramétereit a végpontok között. A kommunikált információ adatstruktúráját egy adatszerződésben, úgynevezett DataContract-ban kell rögzíteni. Továbbá meg kell adni egy, vagy több Binding-ot is, mely leírja, hogy miként kommunikálhatnak a felek (például egyszerűen socket-ekkel TCP csatornán, vagy HTTP-n), és definiálni kell a szolgáltatás elérhetőségi címét.

A projekt keretein belül WCF segítségével vannak a szerver olyan funkciói kiajánlva, melyeket távolról, keretrendszerről és felügyelőalkalmazásról kell tudni elérni. A kommunikációban JSON (JavaScript Object Notation) tölti be az adatszállító adatformátum szerepét.

3.3 Elektroenkefalográfia

Az InnoLearn projektet úgy terveztük, hogy a későbbiekben többféle szenzort is hozzáilleszthessünk. Az elkészített absztrakt perifériakezelő eseményeit a keretrendszer kapja meg és dolgozza fel. A korábbi fejezetben már ismertettem, hogy jelenleg egy EEG (elektroenkefalográf) mérőberendezést illesztettünk a keretrendszerre. Pillanatnyilag fejlesztés alatt áll egy szívritmus mérő eszköz illesztése is.

3.3.1 Emotiv epoc

A projektünkhöz egy Emotiv epoc nevű EEG eszközre esett a választásunk megfizethetősége, és az általa nyújtott funkciók miatt. Ez a berendezés tulajdonképpen egy headset, melyet egyszerűen lehet a játékos fejére helyezni. Előnye a többi hasonló kategóriájú EEG mérőeszközzel szemben, hogy használatához nem szükséges zavaró, sűrű kontaktanyaggal bekenni a felhasználó fejét az elektródák helyén. A headset-en lévő elektródavégződéseken műanyag foglalatok helyezkednek el, melyekbe speciális szivacsszerű vezetőket kell becsavarni. Ezeket csupán egy erre a célra biztosított kontaktfolyadékkal kell megnedvesíteni, mely mellett az Emotiv epoc kényelmes viseletet nyújt.

A viselő neurális aktivitását 16 csatornán[1] méri az eszköz a 0,2 és 45 Hz közötti frekvenciatartományban. A mérést végző elektródák a fejen nemzetközi szabványban[4] megadott pozíciókon helyezkednek el. (Ez azért is hasznos, mert az EEG-t használó kutatások dokumentációi is ezekre a csatornákra hivatkoznak.) Ebből kettőt használ referenciaként, hogy a többi 14 csatorna potenciálkülönbség-jeleit továbbíthassa. A berendezés 8400 μV peek-to-peek feszültségtartományban mér, és 14 biten, digitálisan küldi tovább a mért értékeket (ez 0,51μV felbontást jelent).

Az eszköz vezeték nélküli: akkumulátorral ellátott és rádiós közegben továbbítja a mért feszültségértékeket egy USB-s vevőegységnek. Ezt a vevőegységet illesztjük a játékos táblagépéhez, mely a keretrendszer alkalmazásnak küldi az adatokat.

3.3.2 Felhasznált jelek, adatok

Az EEG headset-tel mért agyhullámokat frekvenciájuk szerint az alábbi kategóriákba[2] sorolhatjuk:

- Delta hullámok: 0,1 és 3,5 Hz között
- Théta hullámok: 3,5 és 8 Hz között
- Alfa hullámok: 8 és 12 Hz között
- Béta hullámok: 12 és 30 Hz között, ezen belül
    - Alacsony béta: 12 és 15 Hz között
    - Közép béta: 15 és 18 Hz között
    - Magas béta: 18 és 30 Hz között
- Gamma hullámok: 30 Hz felett

Ezek közül a delta hullámok sávjában akkor tapasztalható neurális aktivitás, amikor az alany éppen mély álomban alszik (és nem REM fázisban tartózkodik éppen). A théta hullámok félálomban jelentkeznek, illetve felnőtteknél előfordulhat éber állapot mellett rendszertelenül is. E frekvenciatartomány felett már mindenképpen aktív, ébrenléti állapotban van az agyunk. Az alfa hullámok 50 μV amplitúdó alatt figyelhetőek meg pihenő állapotban, csukott szem mellett. Ilyenkor kevés külső ingert

[4] A 10-20-as rendszer, alfanumerikus kódokat társít az elektródák pozícióihoz

dolgozunk fel, ha viszont kinyitjuk a szemünket, vagy hirtelen külső stimulálás történik, akkor átkerülünk a béta hullámok frekvenciatartományába. (Ezt nevezzük deszinkronizációnak, ilyenkor egy alacsonyabb amplitúdójú, de magasabb frekvenciájú jel vált le egy alacsonyabb frekvenciájú jelet.) Nyitott szem mellett az agyunk 13 Hz feletti agyhullámokat generál, melyek amplitúdójából következtethetünk az alany aktuális agyi terhelésének mértékére. A gamma hullámok az információ magas szintű feldolgozásának illetve információrészek összekötésének aktív folyamatára utalnak.

A framework applikáció szenzorillesztő könyvtára generál olyan eseményeket, amelyek a béta hullámok nagyságát szállítják a keretrendszernek, illetve az tovább a felügyelő alkalmazásnak. Ezt úgy érjük el, hogy a megfelelő csatornákon beérkező jelek Fourier transzformáltját képezzük és a kapott spektrumból a béta hullámok tartományában összeadjuk a komponensek amplitúdóit. Erről a folyamatról részletesen az 5. fejezetben lehet olvasni.

A béta hullámok mérésén kívül a projektben felhasználjuk még az úgynevezett P300 jel[3] detektálását is a futó játék befolyásolására is. A P300 jel tulajdonképpen egy minta, pozitív irányú potenciálváltozás a mért agyhullámok jelében. A jel megjelenése (ami körülbelül 400 ms késleltetéssel érkezik) arra utal, hogy az agy felismer egy már korábban eltárolt információt. Tehát e minta detektálásával következtethetünk a tanulási folyamat megtörténtére.

4 A felügyelő rendszer és a szerver megtervezése

Az InnoLearn Supervisor és Server megtervezésénél első sorban olyan szempontokat kellett figyelembe vennem, mint például az együttműködésre tervezés és teljesítményorientáltság. Ebben a fejezetben ismertetem előbb a felügyelőrendszer, majd a szerver alkalmazás fejlesztésének tervezési fázisait.

4.1 Felügyelőrendszer

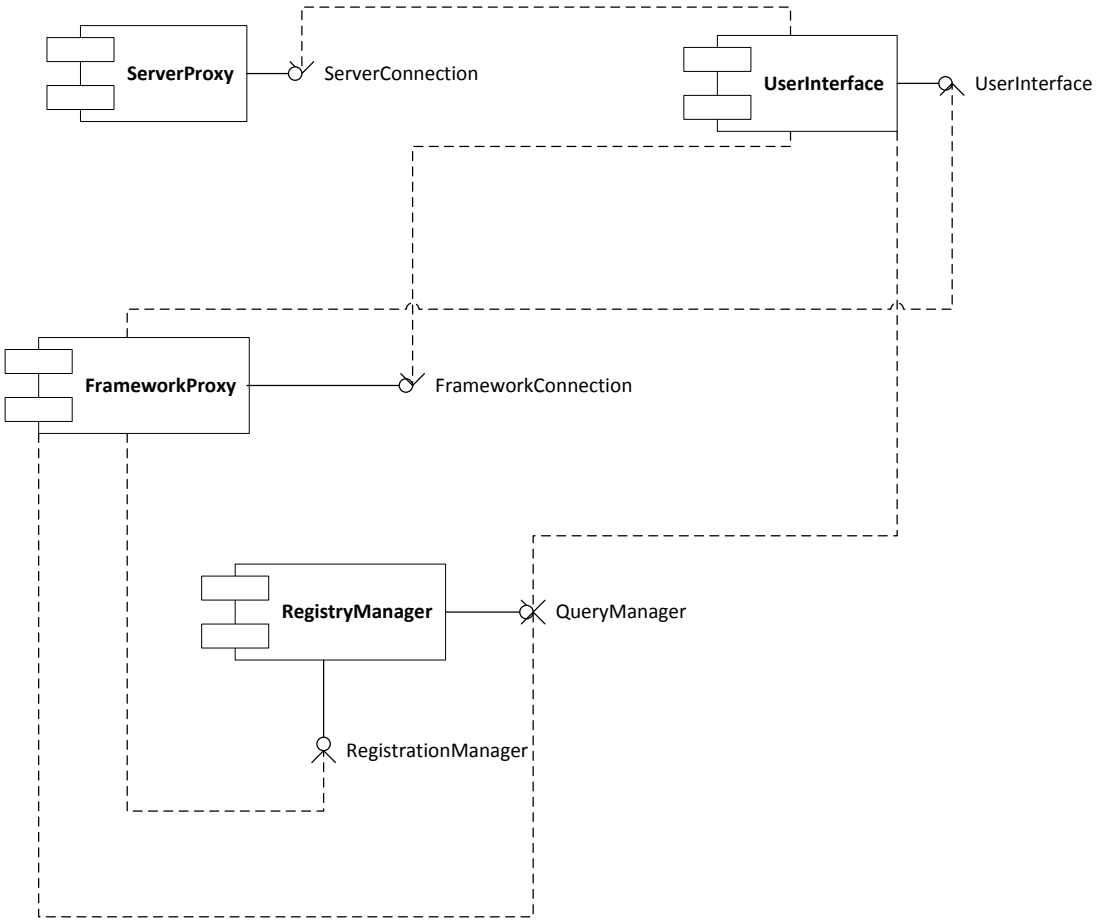
A felügyelőrendszer jelenleg egy keretrendszerhez való csatlakozást támogat, de úgy terveztem meg, hogy a későbbiekben képes legyen majd több keretrendszerhez is kapcsolódni, és a tanár így több felügyelt diákja előrehaladását is figyelemmel követhesse. Egy több keretrendszerből érkező pillanatképgyűjtő felületen a későbbiekben erre is lehetőség lesz.

Fontos szempont továbbá, hogy a supervisor funkciókat megvalósító egységek cserélhetőek, illetve újrafelhasználhatóak legyenek. Így többféle grafikus felhasználói felületet is meg lehet valósítani hozzá, továbbá a keretrendszer fejlesztése és módosítása nem követeli meg a felügyelőrendszer teljes újraírását, hanem csupán egy-egy moduljának cseréjét.

Az ergonomikus grafikus felhasználói felület kialakítása nagy hangsúlyt kap, mivel az az alkalmazás célja, hogy a felügyelést végző tanár viszonylag gyorsan tudjon reagálni az alkalmazás által biztosított interaktív felület segítségével a játékmenetben történő eseményekre. Gyors beavatkozásra lehet ugyanis szükség, ha a játékmenet során olyan esemény generálódik, mely nem releváns a későbbi adattárolás céljából, illetve annak információtartalmát helytelenül befolyásolná. Ezzel valósul meg első körben a zajszűrés.

4.1.1 Statikus struktúra

A supervisor felépítését a 4-1 komponensdiagram szemlélteti.



4-1 Supervisor komponensdiagram

A felügyelőrendszer komponensei az alábbi feladatok ellátásért felelősek:

- ServerProxy: Kommunikáció lebonyolítása az InnoLearn szerverrel, komment feltöltése befejezett játékmenethez
- FrameworkProxy: Kapcsolat létrehozása és fenntartása a keretrendszerrel, parancsok küldése és események fogadása illetve dekódolása
- RegistryManager: Játék-specifikus események beregisztrálása, továbbá események meta-információinak szolgáltatása

- UserInterface: Felhasználói felület komponenseinek egységes ellátása adatokkal

Az egyes komponensek interfészeiken definiálják funkcióikat, melyeknek megvalósítását a komponens belsejében működő osztályok elfedve végzik. A komponensek így csak kiajánlott végpontjaikon kommunikálhatnak egymással, ami növeli az újrafelhasználhatóságot.

Az InnoLearn projektben előre egyeztetett szabvány szerint, az általam fejlesztett felügyelőrendszer alkalmazásban is minden komponenshez tartozik egy inicializáló osztály, mely példányosításakor létrehozza az általa kezelt komponens kapcsolatait a többi egységhez.

4.1.1.1 RegistryManager

Mivel a *RegistryManager* komponens meta-információt kezel, érdemes volt az általa megvalósított funkciókat két interfészben kiajánlani: adat beregisztrálási funkciók és lekérdező funkciók a 4-2 és a 4-3 táblázatok alapján lettek szeparálva.

RegistrationManager	
Metódus	*Funkció*
`registerGameInfo(GameInfo object) : bool`	A hívó komponens kérheti a játékmenet adatainak registry-be történő betöltését
`registerUserInfo(UserInfo object) : bool`	A hívó komponens kérheti a játékos adatainak registry-be történő betöltését
`registerEvent(TypeDescriptor object) : bool`	Játék-specifikus esemény meta-adatait leíró objektum adható meg vele a registry-nek
`registerCommand(TypeDescriptor object) : bool`	Játék-specifikus parancsot leíró adatok tárolása kérhető vele

4-2 RegistrationManager interfész

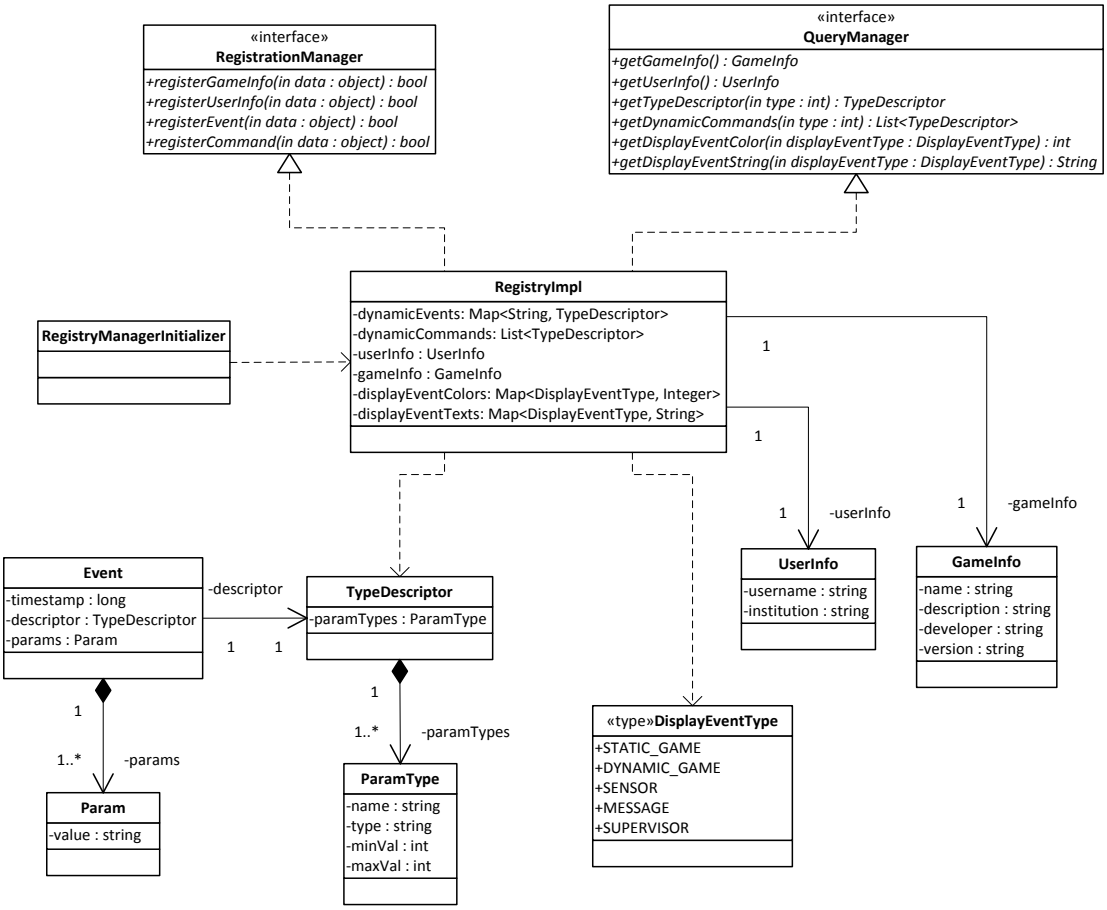
A *RegistrationManager* interfészt csak a *FrameworkConnectionProxy* komponens használja a játékmenet eseményregisztrációs szakaszában.

QueryManager	
Metódus	*Funkció*
`getGameInfo() : GameInfo`	Az aktuális játékmenet adatait adja vissza

`getUserInfo() : UserInfo`	A keretrendszert használó felhasználó adatait adja vissza
`getTypeDescriptor(String name) : TypeDescriptor`	A megadott nevű *TypeDescriptor*-ral tér vissza
`getDisplayEventColor(DisplayEventType eventType) : int`	Eseménytípus alapján lekérhető, hogy milyen színnel jelenjen meg az esemény a felhasználói felületen
`getDynamicCommands() : List<TypeDescriptor>`	Visszaadja a bejegyzett játék-specifikus parancsokat
`getDisplayEventString(DisplayEventType eventType) : String`	Eseménytípus alapján lekérhető az esemény típusának neve

4-3 QueryManager interfész

A *QueryManager* metódusait egyrészt a *FrameworkProxy*, másrészt a *UserInterface* komponens használja, felhasználói felületen megjelenítendő adatokat és a megjelenítés paramétereit kérdezi le vele.



4-4 RegistryManager osztálydiagram

A 4-4 osztálydiagram a komponens belő felépítését ábrázolja. Az interfészeket megvalósító osztály a *RegistryImpl*, amely a játékmenet információit kulcs-érték párok struktúrájában tárolja. A játék-specifikus események és parancsok *TypeDescriptor* osztálypéldányokkal írhatóak le. Ezekre nevük alapján hivatkozhatunk, és az ábrán is látható módon, több leíró paramétert tartalmazhatnak, melyek típusa és értéktartománya is egyenként definiálható. Erről még bővebben az 5.2.2 pontban lehet olvasni.

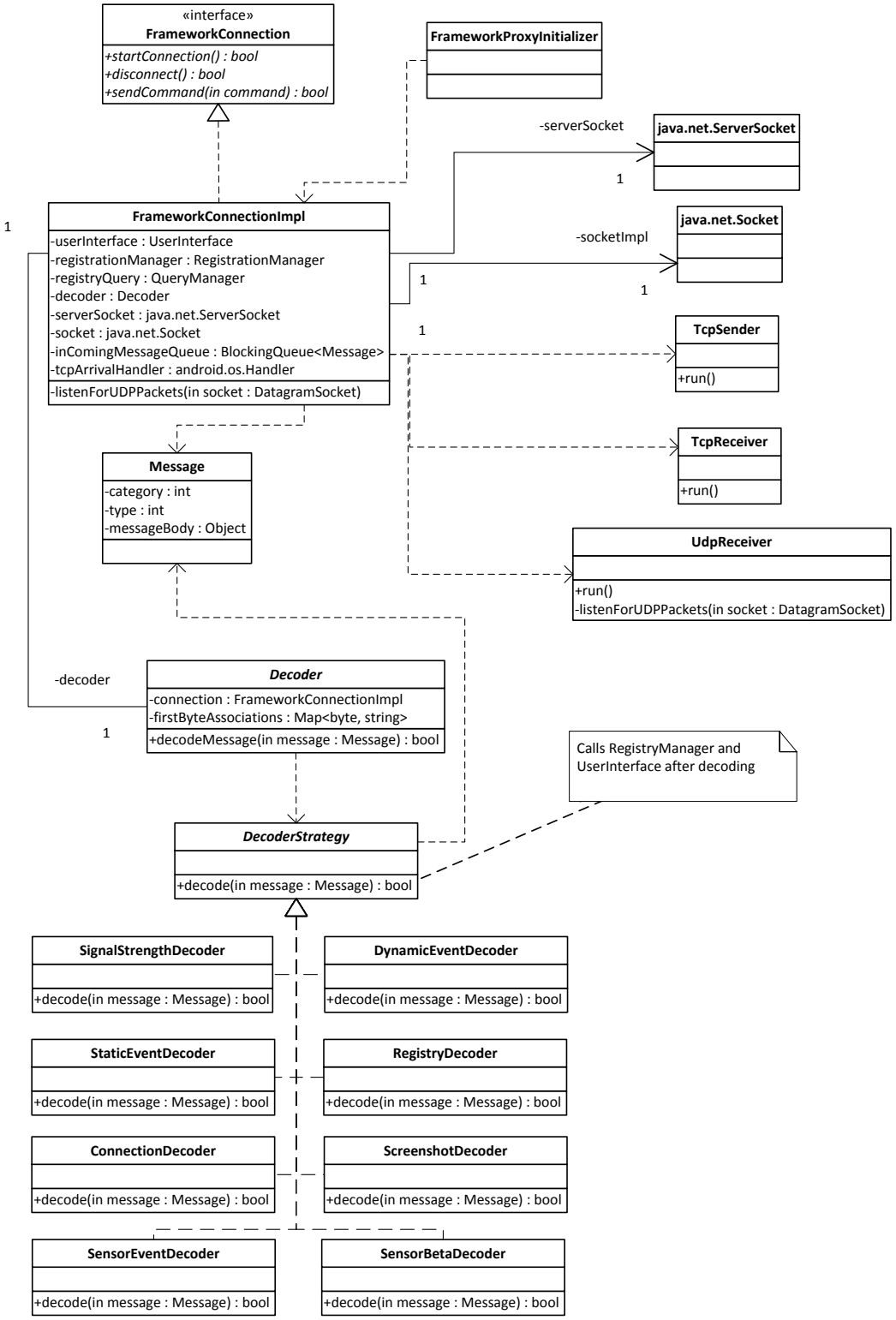
4.1.1.2 FrameworkProxy

A *FrameworkProxy* komponens menedzseli a keretrendszerrel történő kommunikációt. Az általa megvalósított interfész a *FrameworkConnection*, melynek metódusai a 4-5 táblázat tartalmának megfelelőek.

FrameworkConnection	
Metódus	*Funkció*
`sendCommand(Message message) : bool`	Parancs küldése a framework-nek a megadott üzenet objektummal
`startConnection() : bool`	Hálózati kapcsolat kiépítése a felügyelő alkalmazás és a keretrendszer között
`disconnect() : bool`	Framework – Supervisor kapcsolat biztonságos bontása

4-5 FrameworkConnection interfész

E komponens a 3.1.3 pontban bemutatott hálózati protokollok (TCP és UDP) felhasználásával kezeli a keretrendszerrel való összeköttetést. Belső működésének központja a *FrameworkConncetion* interfészt megvalósító *FrameworkConnectionImpl* osztály, ahogy az a 4-6 FrameworkProxy osztálydiagramon is megfigyelhető. A belépési pont a *startConnection* metódus, mely létrehozza az UDP és TCP socket-eket kezelő belső privát osztályokat. Ezek új szálon indítják el az adatkommunikáció lebonyolítását. Az *UdpReceiver* felel a beérkező kapcsolatnyitási kezdeményezés fogadására, majd a kiépített kapcsolatot és a bejövő adatforgalmat a *TcpReceiver* osztály példánya kezeli. Parancsküldés esetén *TcpSender* objektumot hozunk létre, mely elküldi a felügyelő által kiadott parancsot. Minden adatot (UDP-vel történő kapcsolat felépítésen kívül) *Message* objektumba csomagolunk, mely a keretrendszer és a felügyelőalkalmazás egy közös függősége. Eltárolja a küldött objektumot, és annak típusáról is továbbít információt. Ezt a 4.1.2 pontban részletezem.



4-6 FrameworkProxy osztálydiagram

A beérkező *Message* objektumot előbb egy *Decoder* objektumnak adjuk át, hogy a megfelelő dekódolási stratégiát alkalmazza. A *Decoder* strategy szoftvertervezési mintát valósít meg, így a különféle üzenettartalmakhoz mind külön feldolgozóosztály

definiált. A dekódolt üzenet tartalma ezután a felhasználói felületet reprezentáló *UserInterface* komponensnek kerül elküldésre.

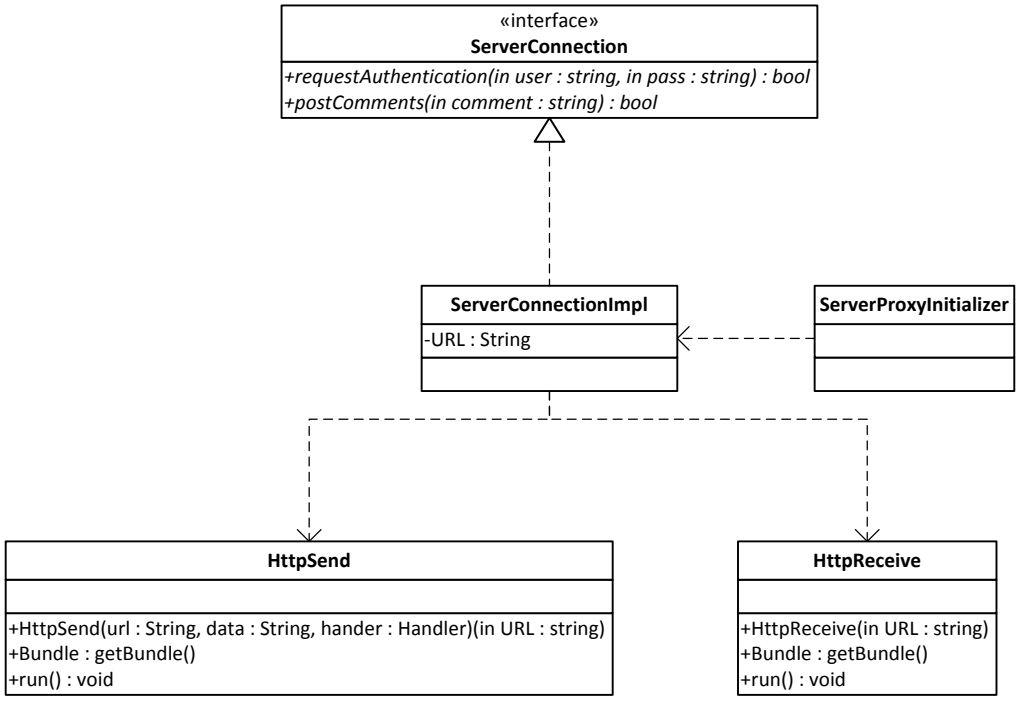
4.1.1.3 ServerProxy

A szerver és a felügyelőrendszer közti kapcsolatért felelős *ServerProxy* komponens HTTP kapcsolatot épít ki, majd ezen küld illetve fogad adatokat. Az általa megvalósított interfész metódusok a 4-7 táblázatban vannak listázva.

ServerConnection	
Metódus	*Funkció*
`requestAuthentication(String user, String pass) : bool`	Bejelentkeztetés adott felhasználói névvel és jelszóval
`postComments(String comment) : bool`	A játékmenet végén megírt kommentek elküldése szerverre

4-7 ServerConnection interfész

A komponens által nyújtott szolgáltatások közül jelenleg csak a *requestAuthentication* metódust használom. Az alkalmazás indulásakor egy beléptető képernyő fogadja a felhasználót, ahol megadhatja a felhasználónevét és jelszavát, melyeket a program ennek a metódusnak ad át (jelenleg tesztelési megfontolások miatt van opció a felületen autentikálás nélküli belépésre is).



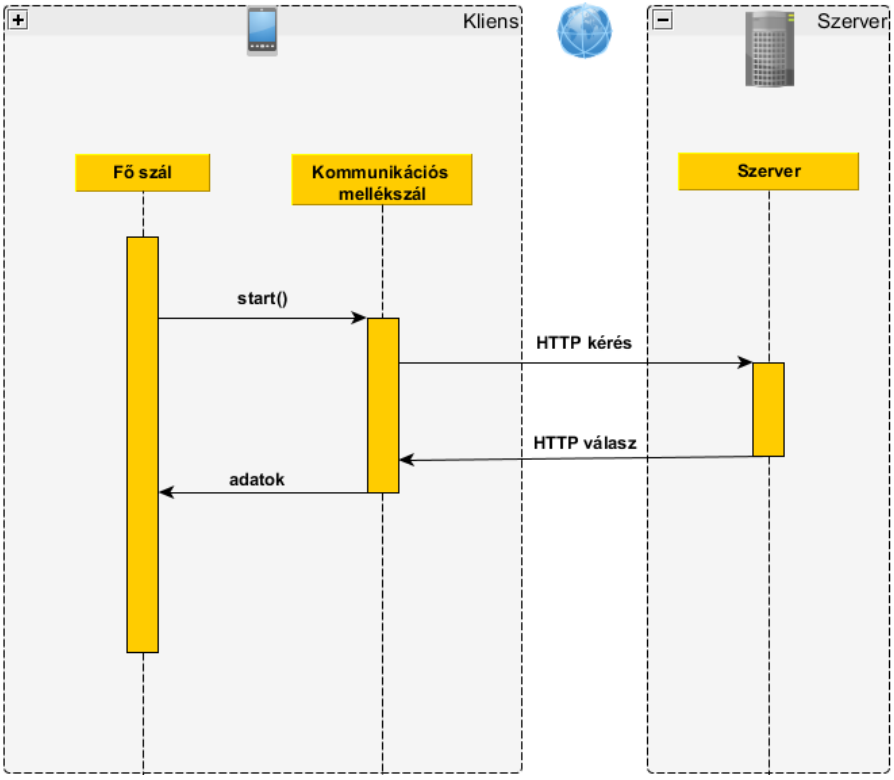
4-8 ServerProxy osztálydiagram

Az üzleti logikája a komponensnek a *ServerConnectionImpl* osztályban van megvalósítva, mely két hálózati működést lebonyolító osztályra, a *HttpReceive* és *HttpSend*-re támaszkodik. Ezek egymáshoz való elhelyezkedése a 4-8 osztálydiagramon látható.

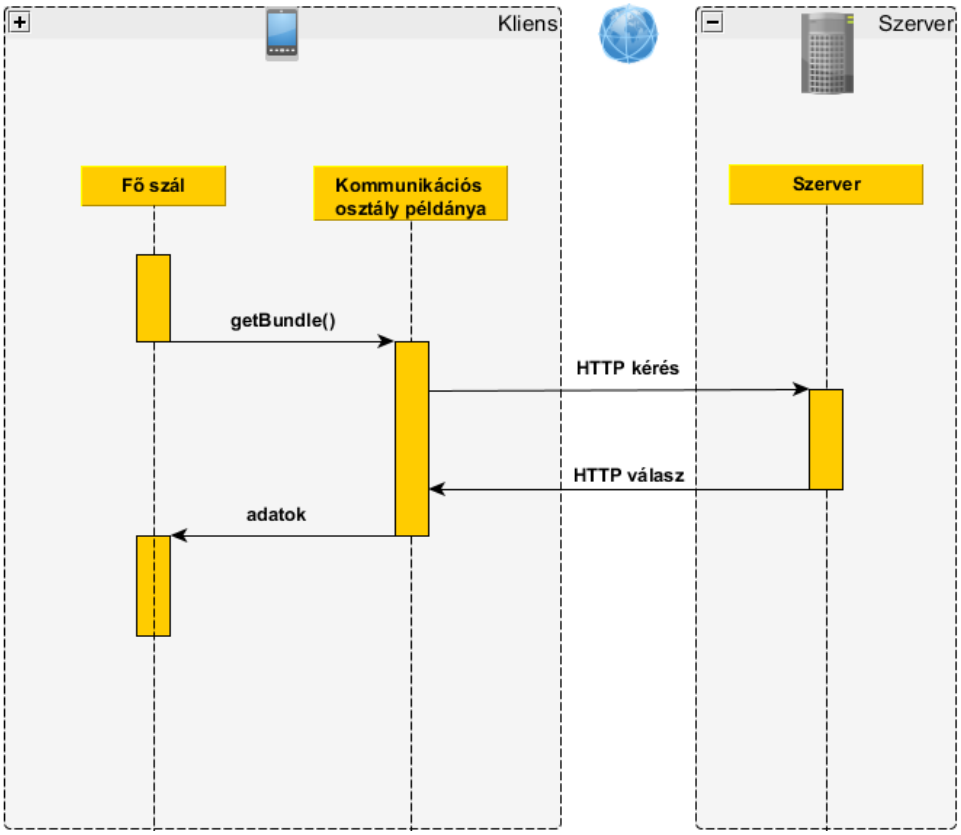
A *HttpReceive* osztály GET metódusú HTTP kérések küldéséért és válaszuk fogadásáért felelős. Ilyenkor adatot az URL paraméterlistáján adunk át. POST metódus használatához a *HttpSend* áll rendelkezésünkre, a neki megadott adatokat egy POST kérés body részébe helyezve küldi el. Ezen felül mindkét osztály szinkronitás szempontjából kétféle módon tud működni:

- Mivel a *Thread* osztályból származnak ezért külön szálon indítva őket aszinkron kommunikációt hajtanak végre, a megadott *Handler* objektumon tudnak visszajelezni, ha megérkezett a HTTP válasz.
- Szinkron módon úgy használhatjuk őket, hogy a *getBundle* metódusukat hívjuk meg példányosításuk után.

A kétfajta működés szekvenciái közül a szinkron a 4-9 ábrán, az aszinkron pedig a 4-10 ábrán figyelhető meg.



4-9 Aszinkron kommunikáció [5]



4-10 Szinkron kommunikáció [5]

4.1.1.4 UserInterface

A supervisor felhasználói felületéért felelős komponense a *UserInterface*. A *FrameworkProxy* komponens dekódolás után ennek a komponensnek az interfészét használja, hogy az a beérkező adatokat, eseményeket megjelenítse. Az interfész leírása a 4-11 táblázatban olvasható.

UserInterface	
Metódus	*Funkció*
`pushDisplayEvent(DisplayEvent displayEvent) : void`	Megjelenítendő esemény küldése a komponensnek
`pushUserInfo(UserInfo userInfo) : void`	Játékos adatait leíró objektum átadása felületen való kiíratásra
`pushGameInfo(GameInfo gameInfo) : void`	Játékmenet adatait leíró objektum átadása felületen való kiíratásra
`pushScreenshot(Bitmap screenshot) : void`	Keretrendszerből érkező pillanatkép kirajzoltatása
`pushSignalStrength(SignalStrength signalStrength) : void`	EEG eszköz jelerősségét leíró objektum átadása kirajzoláshoz

`pushBeta(Double beta) : void`	EEG headset-ből érkező, agyhullámok alapján számított béta érték átadása
`pushDynamicCommand(TypeDescriptor typeDescriptor) : void`	Játék-specifikus parancs átadása, hogy megfelelő vezérlőket rajzolhasson ki a komponens
`pushConnectionAddress(String address) : void`	Keretrendszer helyi hálózati címe (több keretrendszer használatához)

4-11 UserInterface interfész

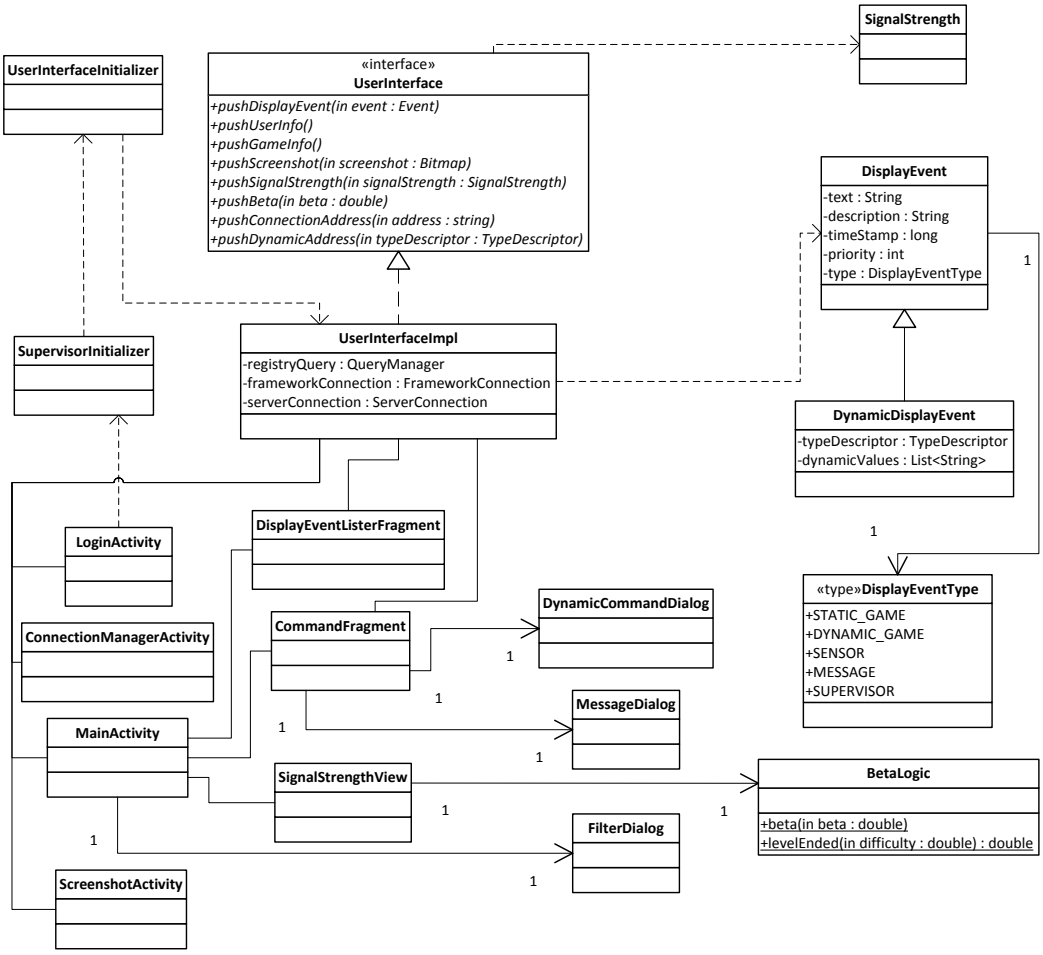
A többi komponenshez hasonlóan itt is egy *UserInterfaceImpl* osztály tartalmazza a kiajánlott metódusok megvalósítását. A 4-12 osztálydiagramon látható módon az Android Activity-k mind referenciát tartanak erre az osztályra, mely szintén referenciákkal rendelkezik a felület elemeiről. Ilyen módon a *UserInterfaceImpl* routing feladatot lát el az adatfolyam szempontjából, hiszen a beérkező eseményeket és objektumokat ő adja át a megfelelő grafikus vezérlőelemeknek.

A következő Activity-ket használtam fel a felügyelőrendszer megvalósításához:

- LoginActivity: Kezdőképernyő, a felügyelőnek itt kell autentikáltatnia magát felhasználónév és jelszó megadásával.
- ConnectionManagerActivity: Pillanatnyilag még használaton kívüli, itt lehet majd választani aktív keretrendszer kapcsolatok közül.
- MainActivity: Főképernyő, a játék és a játékos aktuális állapota követhető nyomon vele, listázza a beérkező eseményeket, vezérlőin keresztül parancs küldhető a keretrendszernek.
- ScreenshotActivity: Pillanatképnek dedikált Activity, teljes képernyőn mutatja a játékos aktuális tevékenységét.

A *LoginActivity* továbbá a felügyelőrendszer indulásakor az alkalmazás belépési pontjaként is szolgál. Ezért az ő felelőssége példányosítani a *SupervisorInitalizer* osztályt, mely aztán majd minden komponenst inicializál, és az összeköttetéseket

létrehozza a referenciák állításával. Sikeres belépés után a *ConnectionManagerActivity* töltődik be, ezután pedig a *MainActivity* jelenik meg. A *MainActivity* felülete két részből áll, egy fix bal oldali sávból, ahol a felhasználó, játék és EEG jelerősség információk vannak kirajzolva, és egy dinamikus tartalmú jobb oldali részből. Ez utóbbi lehetséges tartalmait az Android Fragment osztályából származtattam le, hogy futásidőben cserélhető legyen. A *DisplayEventListerFragment* valósítja meg az események listázását, melynek tartalmát egy erre a célra fejlesztett szűrővel lehet szűrni. A *CommandFragment* segítségével pedig választhat a felügyelést végző tanár, hogy milyen parancsot küldjön a keretrendszernek.



4-12 UserInterface osztálydiagram

A felbukkanó vezérlőfelületeket az Android Dialog osztályával valósítottam meg, így a már említett szűrő beállításait is egy ilyen, *FilterDialog* nevű osztályimplementáció rajzolja ki. A parancsküldő fragment-ben két további Dialog implementációt találunk. Lehetőség van üzenetet írni a keretrendszernek, ami a *MessageDialog* segítségével történik, továbbá a játék-specifikus parancsok küldéséhez

DynamicCommandDialog-ot használtam. Ez utóbbi a dinamikus parancshoz tartozó *TypeDescriptor*-t előbb elkéri a *RegistryManager* komponens *QueryManager* interfésze segítségével, majd a benne megadott paramétertípusok és értékkészletek kiolvasásával testre szabja a megjelenítendő Dialog ablak vezérlőit. Ez egy rendkívül hatásos módszer, hiszen így bármilyen játék bármilyen parancsának beállító felületét egy osztállyal ki tudjuk rajzoltatni és nem kell újabb fajta parancsokhoz a supervisor bináris forrását módosítani.

Az EEG headset jelerősség adatait a *SignalStrength* osztály példányai írják le. Ebben az egyes csatornákon mért jelerősség, valamint a mérőeszköz akkumulátorfeszültsége is tárolásra kerül. A szállított információ megjelenítését a *SignalStrengthView* osztály végzi, amely a *MainActivity* fix bal oldali sávjában helyezkedik el. Ez jeleníti meg továbbá a béta értéket is, melynek előszűréséhez és átlagolásához a *BetaLogic* osztály nyújt neki segítséget.

4.1.2 Kommunikáció

A keretrendszer és a felügyelőrendszer közti kommunikáció közös adatkontraktus meglétét igényli. A korábban leírtak szerint a kapcsolat UDP socket segítségével jön létre, a forgalom lebonyolítása pedig TCP socket-tel történik. *Message* objektumokat küld a két fél egymásnak, melyre a felügyelőrendszer egy közös library projektből hivatkozik. Ez így azért hasznos, mert kommunikált adatoktól és azok típusaitól függetlenül csak egy osztálytól fog függeni mind a keretrendszer, mind a felügyelőalkalmazás. Az osztály az alábbi tagváltozóival definiált:

- type : byte
- content : Object

A *content* tölti be a hasznos teher szerepét, mely többféle típust is felvehet. A típusra azonban szükség lesz a fogadóoldalon, hogy megfelelő osztálypéldánnyá cast-olhassuk az üzenet tartalmát. Erre ad támogatást a *type* tagváltozó, mely egy közösen, előre rögzített értékkészletből vehet fel értékeket. Az ezt leíró osztály a közösen elkészített *MessageCodes* nevű osztály, mely a *Message* osztállyal együtt egy úgynevezett Android library projektből van hivatkozva, hogy mindkét fél elérhesse.

A típusok leírására jelenleg elég 1 byte (ez 256 lehetőséget jelent), és figyelembe véve, hogy ennek elküldése overhead-ként jelentkezik a kommunikációban, ezért ezt minél jobban minimalizáltuk.

A típus byte felépítése a 4-13 táblázatban látható módon hierarchiát követ. Az első 4 bit választja ki az üzenettípus kategóriáját, az utolsó 4 pedig ezen belül megadja a konkrét típust.

Referencia	Byte	Leírás
Kommunikáció		
DISCOVER	0x00	Keretrendszer UDP felderítő csomagja
HELO	0x01	TCP kapcsolat létrejötte után mindkét fél küldi
ACK	0x02	Felügyelőrendszer válasza DISCOVER-re
FIN	0x03	TCP kapcsolat bontása
FRAMEWORK_VERSION	0x03	Keretrendszer verziója
Statikus események és parancsok		
SCREENSHOT	0x10	Az keretrendszer pillanatképet küldött a játékból
GAME_START	0x11	Felügyelő játékindító parancsa
GAME_PAUSE	0x12	Felügyelő játékmegállító parancsa
GAME_RESTART	0x13	Felügyelő játék újraindító parancsa
GAME_FINISHED	0x14	Keretrendszer jelzi a játékmenet végét
LEVEL_RESTART	0x15	Játékszint-újraindítási parancs
MESSAGE	0x16	Felügyelő szöveges üzenete a játékosnak
Regisztráció		
EVENT	0x20	Játék-specifikus eseményleíró objektum érkezett
REWARD	0x21	Játék-specifikus, jutalomleíró objektum érkezett
COMMAND	0x22	Játék-specifikus parancsleíró objektum érkezett
USER_INFO	0x23	Felhasználó adatai érkeztek
GAME_INFO	0x24	Játékadatok érkeztek
Dinamikus esemény		
DYNAMIC_EVENT	0x30	Beérkező játék-specifikus esemény
Protokoll hiba		
PROTOCOL_ERROR	0x40	Hibás típuskód – mindkét fél küldheti
Ismeretlen hiba		
UNKNOWN_ERROR	0x50	Ismeretlen hiba történt – mindkét fél küldheti
Szenzoresemények		
SENSOR_EMOTION	0x60	Játékos aktuális neurális állapotáról érkezett információ
SENSOR_CAMERA	0x61	Kameraesemény érkezett
SENSOR_SIGNAL_STRENGTH	0x62	EEG headset jelerősség leíró objektum érkezett
SENSOR_BETA	0x63	Béta neurális aktivitás érték érkezett

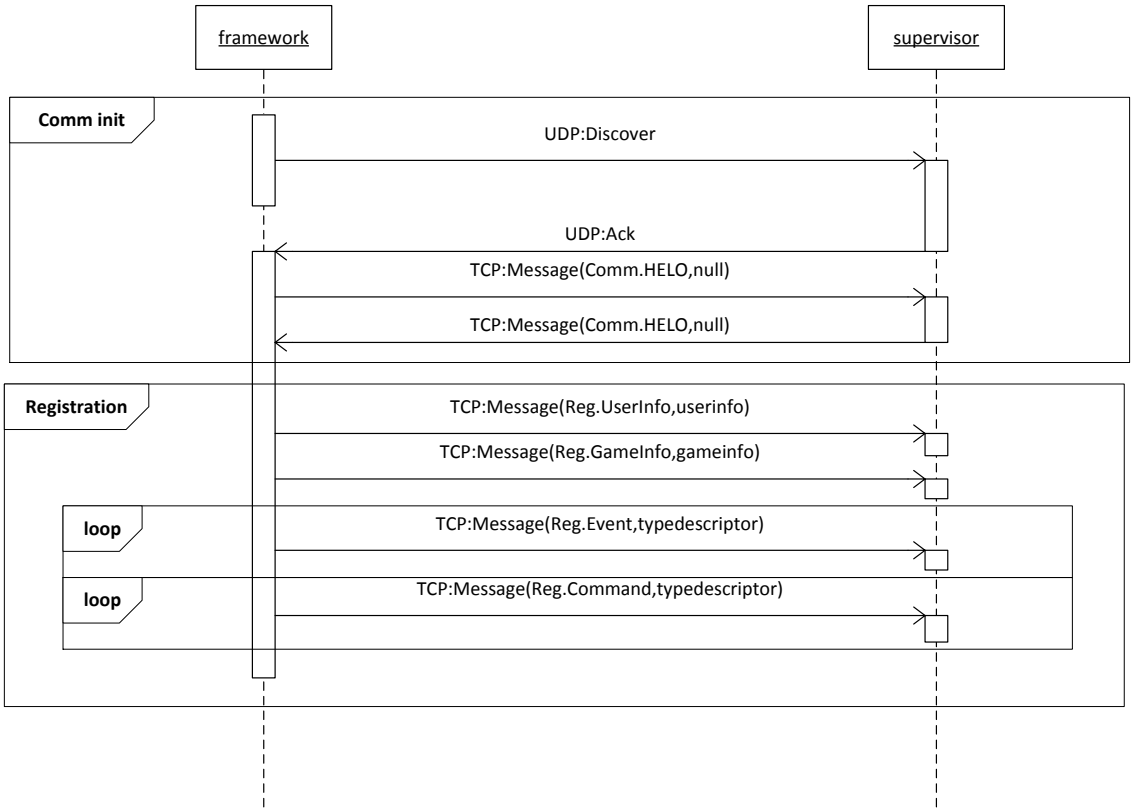
Parancsok		
DYNAMIC_COMMAND	0x70	Játék-specifikus parancs küldése
SCREENSHOT_MODIFY	0x71	Pillanatkép kis-nagy felbontás közötti váltás kérése
INGNORE_EVENT	0x72	Esemény figyelmen kívül hagyásának kérése

4-13 Protokoll típuskódok

A felsorolt típuskódoknak jelenleg csak egy részét használja a projekt.

4.1.3 Viselkedés – dinamikus struktúra

Az alkalmazás belépési pontja a *LoginActvity*. Bejelentkezés után a *ConnectionManagerActivity* indul el, és a *FrameworkProxy* komponens megkezdi az UDP socket-en a várakozást a keretrendszer jelentkezésére. Ha beérkezik a megfelelő UDP csomag, a felügyelő visszaküld egy UDP csomagot arra a címre, ahonnan a felderítő üzenet jött. A framework ezután TCP socket-et nyit a supervisorhoz. A kommunikáció a 4-14 szekvencia diagramnak megfelelő módon zajlik le.



4-14 Szekvencia diagram

Az üdvözlő üzenetek elküldése után belépünk a regisztrációs fázisba. Itt elküldi a keretrendszer a felügyelő alkalmazásnak a játékosról és a játékról megjelenítendő információit, továbbá elküldi az játék-specifikus eseményhez tartozó *TypeDescriptor* objektumokat. Ezek után megtörténik a játék-specifikus parancsok továbbítása is, és supervisor oldalon ezek az információk tárolódnak a *RegistryManager* komponensben.

A regisztrációs folyamat a háttérben zajlik, az előtérben a felhasználó elé tárul a *MainActivity*, melyen megjelennek az érkező meta adatok. A felügyelő tanár ezután a játék indítása gombbal indíthatja el a keretrendszer játékmenetét, és megindul a folyamatos adatküldés és- fogadás:

A keretrendszer által küldött adat a *FrameworkImpl* osztály *TcpReceiver* által képviselt szálban érkezik meg. *Message* objektumra való cast-olás után bekerül az üzenet egy várakozási sorban, és az adatfogadó szál egy Handler-n keresztül értesít, hogy új üzenet érkezett. Ez azért fontos, mert elképzelhető, hogy hálózati forgalmi okok miatt több üzenet torlódva érkezik meg és hirtelen sok adatot kell feldolgozni. A Handler ezután leveszi az új üzenetet a sorból és átadja a *Decoder*-nek értelmezésre. A *Decoder* az üzenet típus byte-jának első 4 bitje alapján eldönti, hogy milyen kategóriájú üzenetről van szó, majd a második 4 bit felhasználásával a megfelelő stratégiát alkalmazza az értelmezéshez. Ha ez például egy béta értéket tartalmazó üzenet, akkor a *SensorBetaDecoder* fogja megkapni az üzenetet.

A megfelelő dekódoló kiválasztása után, az átadott üzenet tartalmát a helyes típusú objektumra cast-oljuk. A dekódereknek referenciájuk van a *UserInterface* komponensre, és ezen keresztül átadják az értelmezett üzenettartalmat a megfelelő metódusnak. A *UserInterface* komponensbe kerülve az implementáció már rendelkezik referenciával a komponens grafikus felhasználói felületet megvalósító objektumaira, és metódustól függően átadja a megjelenítendő tartalmat valamelyik Activity-nek, vagy Fragment-nek.

A másik irány lényegesen egyszerűbb: a felügyelő tanár által kiadott parancsnál, gomblenyomás után hozom létre azt a *Message* objektumot, amit majd a keretrendszerre küldök. A *FrameworkProxy* komponens *sendCommand* metódusának átadva, létrejön egy küldő kommunikációs szál – a *TcpSender* példánya – és az aktív socket kapcsolaton keresztül elküldésre kerül a parancs.

4.2 Szerver

Pillanatnyilag az InnoLearn Server már elérhető a projekt többi komponensének számára, azonban még mindig folyamatos fejlesztés tárgyát képzi. Az alapkoncepció a tervezésnél az volt, hogy egy olyan, később bármikor bővíthető szerveralkalmazást hozzak létre, mely támogat egyszerű, de a projekt szempontjából jelenleg fontos funkciókat. A két fő funkció az adminisztrációs felület biztosítása, illetve a táblagépekről is elérhető adatfeltöltő szolgáltatás.

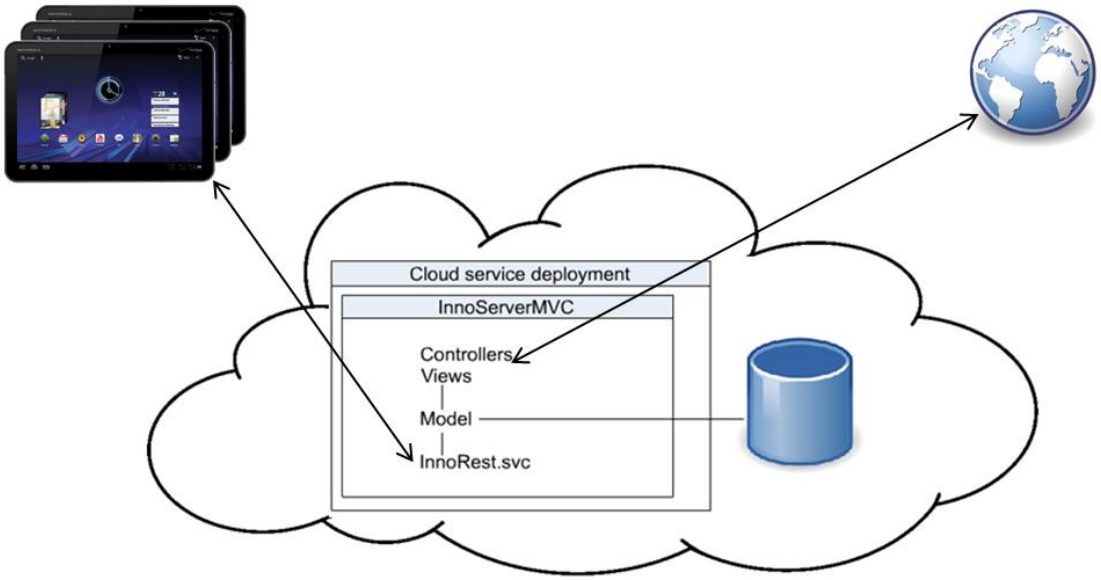
A jövőben szeretnék egy WPF-es klienst is elkészíteni hozzá, amelyen a feltöltött adatok lennének megtekinthetőek grafikonos formában.

4.2.1 Felépítés

A szerver egy ASP.NET alapú MVC 3.0 alkalmazás. A felhőbe kiszervezett szolgáltatás adatforrása egy szintén Windows Azure-on belül található SQL szerver, melyre a játékmenetek adatait tölti fel a keretrendszer. Az adatok így csak a cloud service-en keresztül érhetőek el.

Az MVC architektúra modelljét Entity Framework generálta le az adatbázissémából (a sémáról részletesebben a 4.2.2 bekezdésben lehet olvasni). Véleményem szerint ez a legtisztább módja az adatbázistáblák objektumokba való leképezésének. Ha közben változtatni akarunk az adatbázison, akkor csak újra kell generáltatni a modellt, hogy szinkronban legyen a sémával.

A generált modellnek több felhasználója is van, az egyik az a REST alapú szolgáltatás, amely lehetővé teszi a táblagépek távoli csatlakozását a szerverhez. A másik két modellt használó komponens a Contoller-ek és View-k halmaza, melyek az adminisztrációs felületért felelősek. Egy tanár által megnézni kívánt játékmenet adatok az általuk létrehozott weblapokon jelennek meg. A szerver komponenseinek ilyen módszerű szervezése a **Hiba! A hivatkozási forrás nem található.** képen figyelhető meg.



4-15 Szerver architektúra

A weblapon történő bejelentkeztetéshez az MVC 3.0 által generált *AccountModels* osztályait használom az InnoLearn projektre testre szabottan. Az adminisztrációban háromféle felhasználói jogkört különböztetek meg elérhető funkciók szerint csökkenő sorrendben:

- Admin: minden adminisztratív funkciót használhat, beregisztrálhat új intézményeket, létrehozhatja és módosíthatja a felügyelőket és felhasználókat (játékosokat)
- InstituteAdmin: egy intézményhez tartozó adminisztrátori poszt, a saját intézményéhez tartozó felügyelőket és felhasználókat menedzselheti
- Supervisor: saját felhasználóit menedzselheti csak, megtekintheti a hozzájuk tartozó játékeredményeket, adatokat

A felhasználók számára nem biztosított az adminisztrációs felület elérése – így őket külön táblába is szerveztem – azonban a WCF szolgáltatásba be tudnak lépni a keretrendszer által, hogy játékmenetük adatai feltöltésre kerülhessenek a szerverre.

A fentiek alapján a 4-16 táblázatban összefoglaltaknak megfelelően látják el funkcióikat a Controller-View párok.

Controller/View	Szükséges jogosultsági szint	Funkció
Home	Bárki	Az adminisztrációs felület főoldala
Account	Bárki	Belépés- és regisztráció funkciókat valósítja meg
Institute	Admin	Listázza a tárolt intézményeket és adatait, felületet biztosít ezek módosításához illetve új intézmény hozzáadásához
Supervisor	InstituteAdmin	Felügyelők listáját mutatja, létrehozható és módosítható vele a tanárok/mentorok adatai
User	Supervisor	Felhasználók kezeléséhez nyújt menedzsment felületet
GamePlay	Supervisor	A felhasználó egyes játékmenetei tekinthetőek meg vele
Event	Supervisor	A játékmenethez tartozó, keretrendszer által feltöltött események és azok adatait mutatja meg

4-16 Szerver controller/view fukciói

A felhőben futó WCF szolgáltatás a következő végpontokon várja a beérkező kéréseket (relatívan az innolearn.cloudapp.net/InnoRest.svc címhez):

- /SupervisorLogin: a felügyelőrendszer az indítása után ide küldi el a belépéshez szükséges felhasználónevet és jelszót
- /UserLogin: a keretrendszer játékindítás előtt ezen a címen tudja a felhasználót bejelentkeztetni
- /UserLogout: keretrendszer explicit kijelentkeztetést kérhet session ID alapján

- /GamePlayUpload: játékmenet felöltési végpont a bejelentkezett felhasználóhoz tartozó játékmenet összes adatának elküldéséhez.

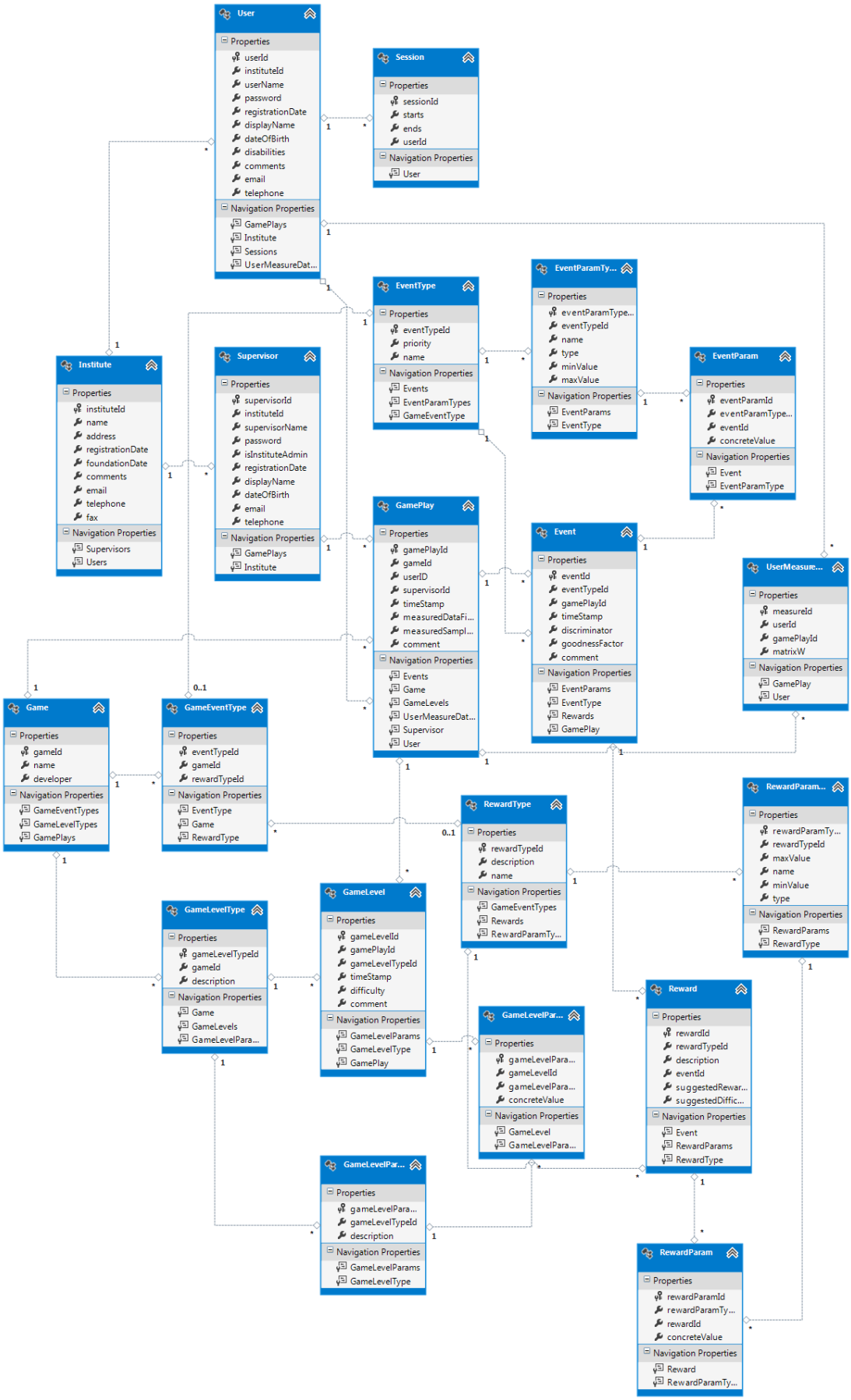
A fenti végpontok mindegyikén POST metódussal várja a szerver a kéréseket, melynek body részében JSON adatstruktúra az elvárt.

A WCF szolgáltatás szintén az Entity Framework által generált modellt, az *InnoModel.edmx*-et használja fel. Működéséhez szükséges azonban még az is, hogy tudja, milyen adatokat várhat és küldhet. Ezért érdemes kiszervezni egy külön osztályba a DataContract-okat. A szerver projektben így a *DataContracts* osztály tartalmazza azokat az osztályokat, melyek nevei [DataContract]-tal, tagváltozói pedig [DataMember]-rel annotáltak. Ilyenek például az elkészített *LoginRequest, LoginResponse* osztályok is.

4.2.2 Adatstruktúra

A szerveren alkalmazott adatbázisséma tulajdonképpen a keretrendszeren is kurrens sémának a kiegészítése az adminisztrációs feladatok ellátásához szükséges táblákkal és kapcsolatokkal. Az SQL szerver adatbázisában található táblák szövevényes hálózata a 4-17 ábrán tekinthető meg.

A User tábla reprezentálja a játékmenetekben résztvevő felhasználókat. Hozzá kapcsolódóan egy Session nevű táblát készítettem el, mely a bejelentkezésben nyújt segítséget: felhasználói belépés után egy session kulcs generálódik, amelyet elévüléséig (jelenleg 1 hónap alapértelmezetten) használhat a keretrendszer játékmenet adatok feltöltésére. Egy felhasználó – és a felügyelő (Supervisor) is – egy intézményhez (Institute) kapcsolható a projekt pillanatnyi állása szerint, és több játékmenet (GamePlay) tartozhat hozzá. A GamePlay eseményekből áll (Event), melynek paramétereire és adataira a többi tábla segítségével hivatkozik.



4-17 Szerver adatbázisséma

5 A megvalósítás

Ebben a fejezetben az implementációs fázis érdekesebb részeiről lehet olvasni, továbbá az elkészített komponensek prototípusainak tesztelését mutatom be.

5.1 Béta érték mérése

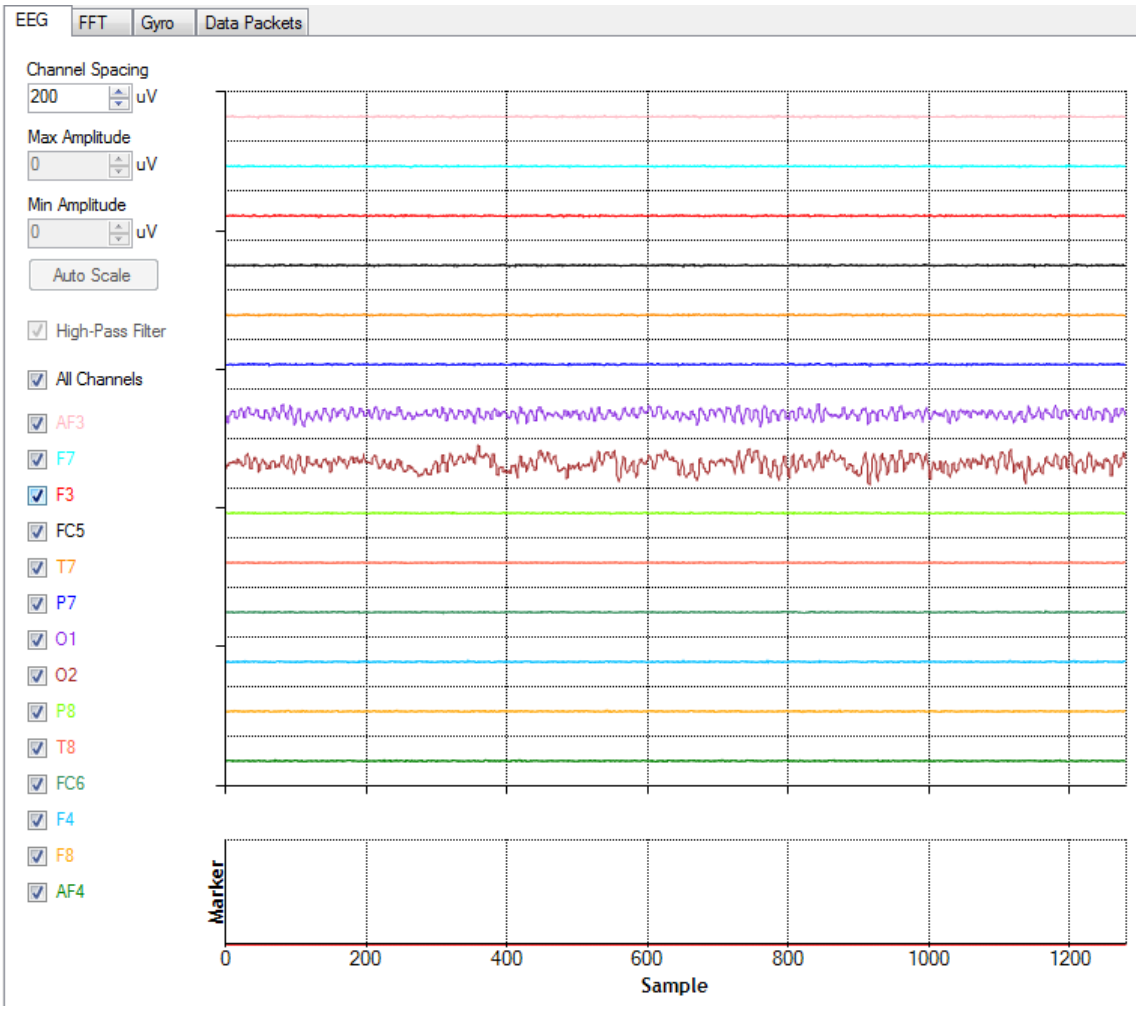
Jelenleg a legtöbb fejlesztési munkát az úgynevezett béta érték felhasználhatóságának vizsgálatára fordítom. A 3.3.2 részben bevezetett béta frekvenciatartományú agyhullámok mérésből származó adatot szeretném a játékmenet alatt megvizsgáltatni egy algoritmussal. A cél az, hogy ezzel befolyásolni lehessen a következő játékszint nehézségét. (Azaz egy ajánlást adjon a nehézségi szintre, mivel a végső cél az, hogy több szenzorból illetve adatforrásból érkező információ aggregált összességéből állapítsuk meg a szükséges nehézséget.

5.1.1 Alapkoncepció

Tudományos kutatások[6] alapján a fej megfelelő pontjaira elhelyezett elektródákról érkező megfelelő frekvenciatartományba eső jelek erőssége pozitívan korrelál az alany aktuális neurális terhelésével.

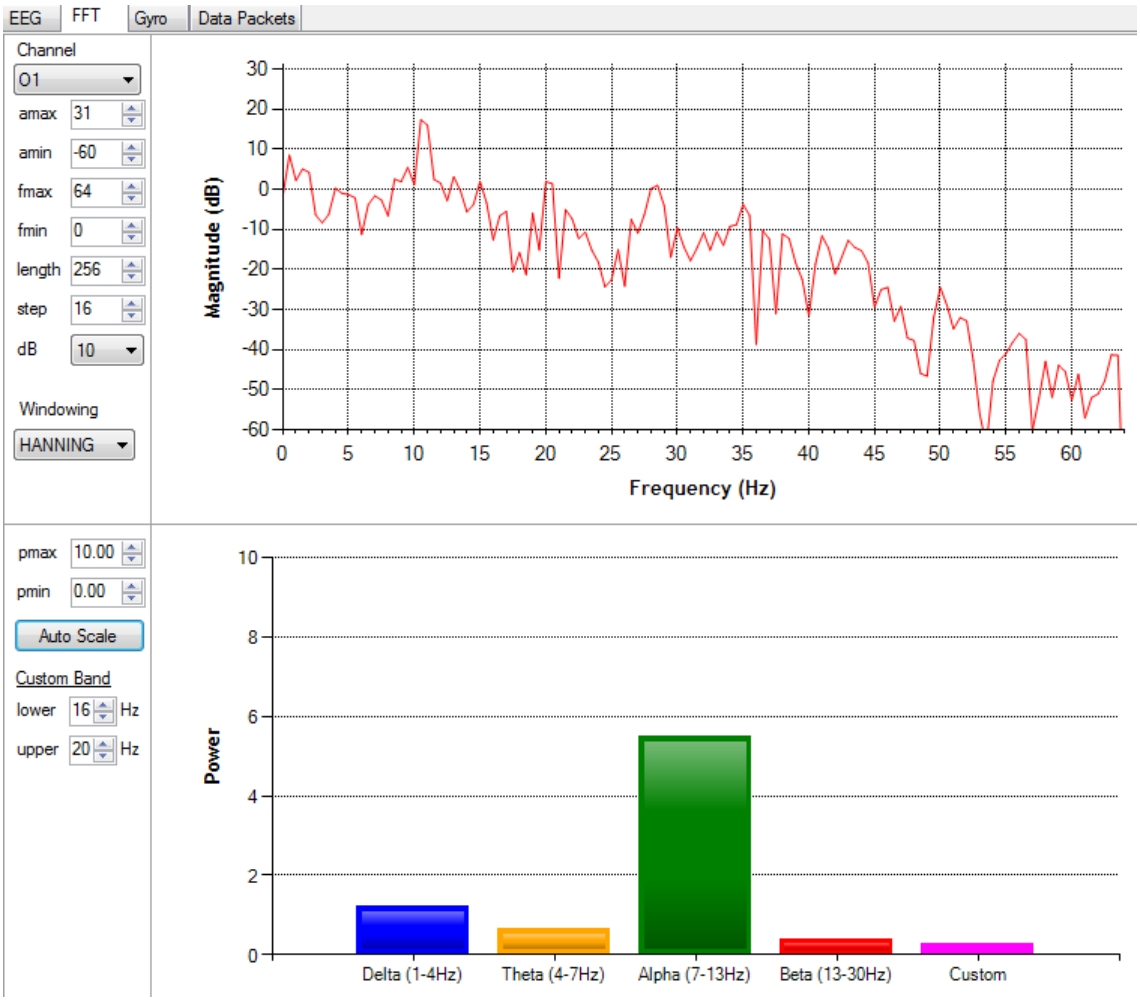
Ehhez természetesen azt feltételezzük, hogy az egyszerű illetve összetett feladatok elvégzésekor manifesztálódó kognitív állapotok lebonthatóak olyan komponensekre, mint például figyelem, memóriaterhelés, és mentális terhelés. Az EEG által mért terhelés növekszik a szükséges memória felhasználás növekedésével, problémamegoldás alatt és elemező érvelés során.

Az 5-1 képen megtekinthető az Emotiv Epoc EEG eszközünkkel mért jelhalmaz, mely mérés során az O1 és O2 csatornáknak megfelelő elektródák kerültek bekalibrálásra. Ezek a csatornák az agy nyakszirti lebenyéből származó agyhullámait továbbítják. Ebből az adatból – pontosabban az O1 csatornán érkező jelből – Fast-Fourier transzformációval kinyert spektrum az 5-2 ábrán látható.



5-1 Mért EEG jelek O1 és O2 csatornán

A képeken látható értékeket saját magamon mértem csukott szem mellett, pihenő, ellazuló állapotban. Az O1 jel spektrumán jól látszik, hogy az alfa frekvenciatartományban magasabb jelerősséget mérhetünk, mint a béta tartományban, amely megfelel az elvárásainknak (lásd 3.3.2 bekezdés.) A Custom nevű oszlop a 16 és 20 Hz közötti tartományt mutatja – körülbelül a béta tartomány közepe – melyet jelenleg a keretrendszer szenzorkönyvtára is szolgáltat. Ezt az adatot látjuk tehát a felügyelőalkalmazás főablakában is, ami azért hasznos, mert a felügyelést végző tanár számszerű érték alapján tud meggyőződni arról, hogy a távoli táblagépet használó játékos figyelemmel kíséri-e a játékmenet feladatait.



5-2 EEG O1 csatorna spektruma

5.1.2 A nehézségek

Az 5.1.1 pontban az O1 és O2 csatornák jeleinek felhasználásával mutattam meg a béta érték méréséből származó adat játéknehézséget befolyásoló koncepcióját. Jelenleg az implementációban ezeket a csatornákat használjuk fel a spektrum előállításához. A gond az, hogy az O1 és O2 csatorna az agy nyakszirti lebenyének aktivitását méri. Az innen származó béta tartománybeli jelerősség-értékek kevésbé korrelálnak a neurális terheléssel, mint ha azokat homloklebenyen mérnénk.

Az F- (azaz a frontális, a homloklebenyen elhelyezkedő mérési pontok) csatornákon mért jelek felhasználhatóságát azonban rontja az a tény, hogy a zajterhelés ezeket a csatornákat jobban befolyásolja, mint a fej tarkó részén mért jeleit. A zajok olyan nem mentális terhelésből származó hasznos jelformák, mint például a pislogás, izmok megfeszítése, stressz, idegesség, vagy szorongás.

A fentieken kívül minden csatorna jel-zaj értékét rontja az alany fejmozgása, amely befolyásolja az elektróda érintkezését a fejjel. A mérések során többször előfordult, hogy már kisebb fejmozdítás esetén is egy-egy pillanatra tízszeres nagyságú bétaértéket mértem a felügyelőrendszer grafikus felületén. Ezek nyilvánvalóan fals adatok, melyekre egy intelligens szűrést kell alkalmazni.

5.1.3 Az algoritmus

Az algoritmus kidolgozása pillanatnyilag folyamatban van, egyelőre egy csatornáról érkező jeleket használok fel. A teszteléshez nagy segítséget nyújt, hogy a kapott mért értékeket a felügyelőalkalmazáson meg tudom tekinteni. Az elkészült modult majd a keretrendszerbe kell beépítenem, de tesztelési megfontolásokból, és mivel a kurrens béta értéket egyébként is meg kell jeleníteni a supervisor felületen, az algoritmust magába foglaló *BetaLogic* osztályt a felügyelőrendszer részeként dolgozom ki.

Az eddig elkészült implementáció egyszerű funkcionalitást valósít meg. Az alapgondolat az, hogy a folyamatosan beérkező bétaértékeket a játékmenet alatt játékszintenként átlagoljuk. Ekkor az első szint után még nem tudunk ajánlást adni, de eltároljuk a befejezett szint nehézségét és hozzátartozó béta átlagot. A következő játékszint végén már össze tudjuk hasonlítani, hogy az előző szint óta hogyan változott a nehézség és a bétaátlag, azaz a játékos mentális terhelése. Így már ajánlást tehetünk, annak érdekében, hogy a játék az egész játékmenet alatt megfelelő szintű neurális tevékenységet igénylő feladatokat adjon a felhasználónak.

Az aktuális átlag új beérkezett értékkel való befolyásolása előtt egy szűrést hajtok végre a zajos adatok kiküszöbölése érdekében. Az első N mérési adat után kialakult átlagot nem engedi az algoritmus olyan adatokkal befolyásolni, melyek az aktuális átlagtól egy bizonyos intervallumon kívül esnek. Ezen intervallumot, és az N számot is mérési eredményekre támaszkodva határoztam meg. Pillanatnyilag a játékszint elindulása utáni első 10 másodpercig nem veszem figyelembe az érkező béta értékeket, mert kell egy kis idő, míg az érkező adatok helyes értéket vesznek fel (az FFT komponens is pufferrel való működése miatt)

5.2 Játék-specifikus parancsok vezérlőfelülete

A supervisor komponens elkészítésének egyik legnagyobb kihívása – a rendszer terveinek elkészítésén kívül – annak a grafikus felületnek a megvalósítása volt, amely akkor tárul a felügyelést végző pedagógus elé, mikor játék-specifikus parancsot szeretne elküldeni.

5.2.1 A probléma

Az InnoLearn projektben egy olyan rendszert akarunk létrehozni, mely nagymértékben játék-független. A keretrendszerre elkészített játékok által használt események és parancsok egyéniek, előre nem ismertek. Ez így azért rendkívül hasznos, mert nem kötjük meg a játékfejlesztő kezét, így kevesebb korlátozással kell élnie, hogy a framework-höz illeszthesse az alkalmazását.

A feladat nehézségét az adja, hogy a fentebb leírt tények miatt az előre megtervezett, Android layout fájlban leírt grafikus elemekkel szemben, az e parancsokhoz kirajzolandó vezérlők paramétereit csak futási időben tudjuk meg. Emiatt programkódból, dinamikusan kell létrehozni a különböző View elemeket.

5.2.2 Megoldás Dialog osztályban

Összetett paraméteres, többvezérlős parancsok küldése előtt meg kell tudni jeleníteni a felhasználó számára a beállító szerveket. Ezt Android platformon Dialog osztály segítségével érdemes megvalósítani, amely felugró ablakként jelenik meg az aktuális Activity fölött. A Dialog-ból leszármaztatott *DynamicCommandDialog* osztály konstruktorában átveszi a *FrameworkConnection* interfészt (hogy majd ennek a segítségével küldhesse el a parancsot) és egy dinamikus parancsot leíró *TypeDescriptor* objektumot. A *TypeDescriptor* a következő módon működik:

A dinamikus esemény vagy parancs tartalmaz referenciaként egy ilyen *TypeDescriptor* példányt, továbbá eltárolja egy String tömbben a paraméterek értékeit (ezek többféle típus String-gé alakított formátuma). A hozzátartozó *TypeDescriptor*-ban tárolt paraméterek tömbjében sorrendhelyesen szerepelnek az értékekhez a *ParamType* objektumok. A *ParamType*-ban eltároljuk az adott paraméter nevét, típusát (például int, String, long, double stb.), egy minimum- és egy maximumértéket, melyekkel az értékkészletet határozzuk meg.

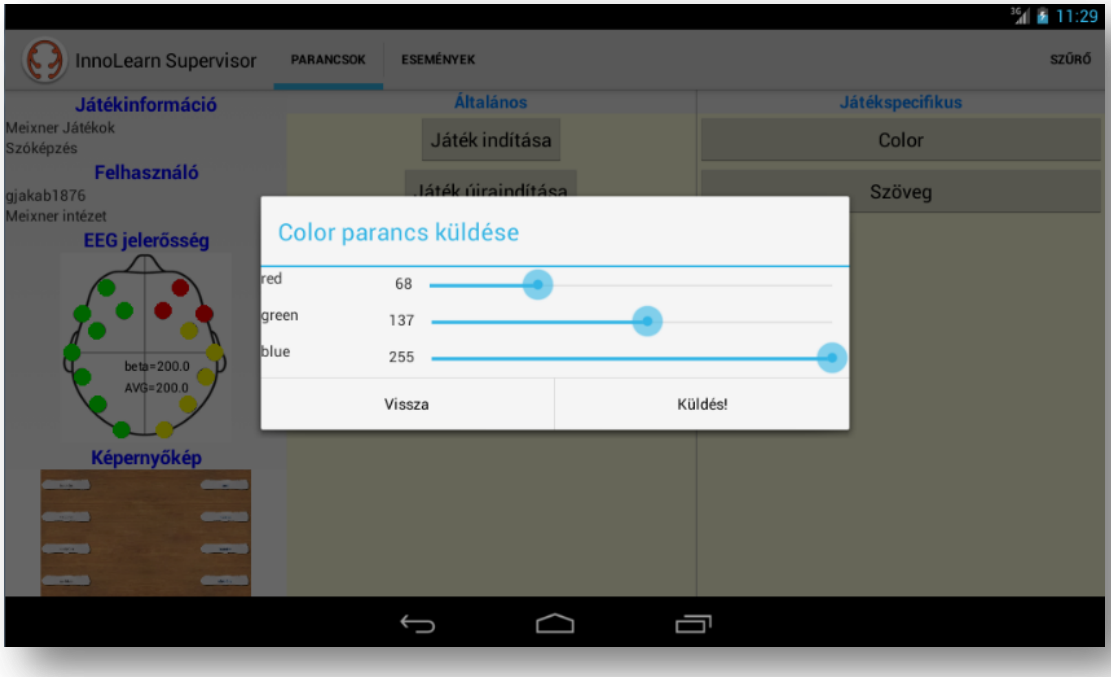
Ilyen struktúrában leírt paraméterekhez úgy érdemes vezérlőt rajzoltatni, hogy azt szám típusok esetében csúszkát (Android Seekbar), szöveg típus esetében pedig szövegbeviteli mezőt veszünk fel a felületre. Csúszka esetén beállítjuk, hogy mik a szélsőértékek az értékkészletet illetően, továbbá intelligens beosztást készítünk, hogy a csúszka hosszában lineárisan elosztva reprezentálják a csúszka pozíciói az értékeket. A csúszkát 0-100 közötti beosztással hozom létre, és az aktuális pozíció által reprezentált értéket megjelenítem a felügyelő tanárnak egy SeekBar mellett elhelyezett TextView segítségével.

Az *onCreateDialog* metódusban létrehozom a megjelenítendő View elemeket, az OK és mégse gombokat, továbbá meghívom az *inflateDialogControls()* függvényt, melybe kiszerveztem a futtatási időben való felület-összeállítást. Ennek átadom paraméterként azt a konténert (LinearLayout példány), amit ki akarok vele töltetni. Ebben a függvényben ciklussal végigjárom a *TypeDescriptor paramTypes* listáját, és az előző bekezdésben leírtak alapján létrehozom a szükséges Widget-eket, és hozzáadom a konténerhez, valamint egy *widgets* nevű tagváltozó listájához is hozzáadom a referenciát.

Ezzel azonban csak az egyik irányt oldottam meg, az OK gombra való érintés után még végig kell járni a felületeken kirajzolt kezelőszerveken, és összegyűjteni egy küldésre kész objektumba a beállított értékeket. Ehhez egy OnClickListener-t kell írni, mely ciklusban végignézi közös indexváltozóval a *paramTypes* és a *widgets* listákat:

```
.setPositiveButton(R.string.dialogOK,
   new DialogInterface.OnClickListener() {
      @Override
      public void onClick(DialogInterface dialog, int id) {
         values.clear();
         for (int i = 0; i<paramTypes.size(); i++){
            Class<?> classType = paramTypes.get(i).getTypeClass();
            View widget = widgets.get(i);
            if (classType.equals(String.class)) {
               values.add(((EditText)widget).getText().toString());
            }else{
               values.add(generateStringValue((SeekBar)widget, paramTypes.get(i)));
            }
         }
         DynamicEvent commandEvent = new CommandEvent(Calendar.getInstance()
                                    .getTimeInMillis(), null, typeDescriptor
                                    .getName(), values);
         Message msg =new Message(MessageCodes.Command.DYNAMIC_COMMAND, commandEvent);
         boolean result = frameworkConnection.sendCommand(msg);
         if (!result)
            Toast.makeText(getActivity(), "Sikertelen parancsküldés!",
                           Toast.LENGTH_SHORT).show();
      }
   })
```

A *generateStringValue()* függvénynek, a neki átadott SeekBar és *paramType* kinyeri a SeekBar alapján a megfelelő értéket, és a *paramType*-nak megfelelő típusra cast-olja. Az elkészített implementáció az 5-3 képen látható. Itt egy Color nevű tesztparancsot paraméterezhetünk, mely a jól ismert RGB struktúrával rendelkezik.



5-3 DynamicCommandDialog

5.3 Felügyelőrendszer tesztelése

A supervisor alkalmazás fejlesztése során implementált funkciók helyes működését verifikálni kell. Android alkalmazásról lévén szó, első körben JUnit tesztek juthatnak az ember eszébe, mivel a Java programozási nyelvben készült el a program. Az Android SDK kínál speciálisan Android JUnit tesztelésre is lehetőséget[5], azonban ezzel egy olyan több komponensből álló projektben, mint az InnoLearn projekt, nem fedjük le a komponensek együttműködésének ellenőrzését, az integration tesztet.

[5] Az android.test.runner könyvtárat be kell tölteni az AndroidManifest.xml-ben, ezek után rendelkezésünkre áll az Android Test Instrumentation, melynek megadhatjuk a teszteket tartalmazó package nevét.

5.3.1 Összetett rendszer, bonyolult tesztelés

A bemutatott projektben láthattuk, hogy több különböző hardverelem, különböző programozási nyelveken megírt szoftver, és többfajta protokoll felhasználásával működik együtt. Bár a felügyelőrendszer tervezése során arra törekedtem, hogy jól elkülöníthető komponensekből álljon, azért az implementációt mégsem lehet teljesen elszeparáltan komponensenként véghezvinni. Az alkalmazás két legfontosabb funkciója, az adatkapcsolat kezelése a keretrendszerrel, és a megjelenítés így időben részben átlapolva készült el. Mivel a keretrendszer a felügyelőalkalmazás fejlesztésekor még szintén csupán fejlesztési fázisában tartott, nem lehetett tesztelni a felügyelő helyes működését.

A helyes megoldás így az volt, hogy az előrehaladás érdekében létrehozzak egy mock[6] keretrendszert, mellyel szimuláltathatom az adatforrást, amitől az eseményeket fogadja a felügyelőrendszer.

5.3.2 A keretrendszer mock-olása

Az *InnolearnSupervisorTester* projekt egy egyszerű Android alkalmazás, mellyel a supervisor alábbi funkciói tesztelhetőek:

- Hálózati kapcsolat kiépítése
- Adatátvitel a kiépített kapcsolaton mindkét irányban
- Közös *Message* és *MessageCodes* osztályok tesztelése
- Pillanatkép – Bitmapátvitel tesztelése
- Játék-specifikus események és parancsok regisztrációja

A tesztelő alkalmazás felhasználói felületén lehetőség van IP cím megadásával is kapcsolódni a felügyelőrendszerhez, továbbá UDP broadcast módú csomagküldéssel szimulálható a felderítési funkció működése is. E kezelőfelületet az 5-4 ábrán lehet megtekinteni. Lehetőség van még a felületen a képküldés állítására is, erre szolgál a hivatkozott ábrán látható PicSend gomb.

[6] Olyan objektumok vagy komponensek, melyek szimulálják az igazi objektumok viselkedését. Használatuk hátránya, hogy folyamatos karbantartást igényelnek (hogy követhessék az igazi objektumok fejlődését), és rosszabb tesztlefedettséget érhetünk el velük.

E tesztprojekt elkészítése rendkívül hasznosnak bizonyult, hiszen olyan hibákra hívta fel a figyelmem, mint a képküldésnél jelentkező OutOfMemoryError, mely hibakeresés után arra volt visszavezethető, hogy a Java-ban implementált ObjectOutputStream-en bizonyos időközönként *reset()* metódust kell hívni, hogy eltávolítsa a már nem használt memóriában tartózkodó objektumokat.

5-4 InnolearnSupervisorTester

6 Összefoglalás

6.1 Értékelés

A projektben való szerepvállalásom során több komponens szoftverfejlesztési folyamatát járom végig. A folyamatnak még nincs vége, hiszen az implementációs fázis a különböző intézményekkel való együttműködés előrehaladásával még mindig tart, azonban úgy gondolom, hogy a projekt által megfogalmazott célért dolgozni rendkívül kifizetődő. A különböző rendszerkomponensek együttműködése mindig is érdekelt, és véleményem szerint ebben a projektben kellő szakmai kihívással találhatja magát szemben az ember. Egy ilyen kaliberű fejlesztési projektben rengeteg, egyetemen tanult ismeret köszönt vissza, a teljes kép kialakításához szerteágazó tudásbázisra van szükség. Elég, ha csak a tervezésre gondolunk, hiszen a felügyelőalkalmazás statikus struktúrájának elkészítése is meglehetősen bonyolult feladatnak bizonyult. Úgy gondolom, hogy a rendelkezésre álló technológiákat hatékonyan sikerült beépíteni, felhasználni mind a supervisor, mind az InnoLearn szerver elkészítése során. Továbbá mind a tervezés, mind az implementáció során hasznos volt különös körültekintéssel eljárni, hogy a projekt komponensrendszerébe jól illeszkedjenek az általam kidolgozott alrendszerek is.

A mobil technológia felhasználását egy izgalmas, új megközelítésbe helyeztük, hiszen a biofeedback alapokon működő rendszerek még aligha elterjedtek, így érdekes kutatási témában van szerencsém részt venni. A projekt célja talán nem is lehetne nemesebb, mint speciális oktatási igényű gyermekek számára segítséget nyújtani az InnoLearn rendszerrel, amely forradalmasíthatja az oktatás több területén alkalmazott módszereket. Az elkészített és bemutatott komponensek prototípusai partnerintézményeinknél rendkívül jó fogadásban részesültek, növekvő érdeklődésben részesítve a projektünket. A projekt során továbbá lehetőségem van olyan eszközök megismerésére és olyan tudományterületekbe való belekóstolásra, mint például az elektroenkefalográffal történő neurális aktivitás mérése. A felsoroltak miatt véleményem szerint ez a munka nagyban hozzájárul szakmai fejlődésemhez, és tapasztalataim gazdagításához.

6.2 Továbbfejlesztés irányai

Mivel az elkészített komponensek csupán prototípusok, ezért folyamatos fejlesztés alatt állnak még, újabb és újabb funkciók beépítését tervezzük az InnoLearn projektbe.

A projekt keretein belül jelenleg az 5.1 fejezetben ismertetett béta érték alapján történő nehézségi szint beállító algoritmus fejlesztése van első helyen az elkészítendő illetve fejlesztenivaló funkciók listáján. Itt érdemes azt vizsgálni, hogy – megfelelő szűrők alkalmazása után – milyen más csatornákról érdemes a jeleket felhasználni, illetve ezeket milyen frekvenciatartományban vizsgáljam. A neurális terhelés mérése nem triviális feladat, több változó használatát is megköveteli majd a minél pontosabb játéknehézség szabályozórendszer kifejlesztése.

A felügyelőrendszer egyik legsürgetőbb újítása a több keretrendszerhez való csatlakozási lehetőség implementálása lehetne. Az alkalmazás tervezése során ügyeltem rá, hogy az elkészített tervek ne zárják ki a jövőben történő ilyen irányú bővítést. Később érdemes lehet a felhasználói felület megreformálásán is elgondolkozni, hogy a felügyelést végző tanárok és mentorok minél ergonomikusabb kezelőfelületet kaphassanak kézhez. Ehhez természetesen elengedhetetlen a velük történő konzultálás.

Szerver oldalon, azaz inkább adattárház oldalon, már utaltam rá, hogy célszerű lenne diagramszerű adat statisztikákat készíteni, és ezekhez egy WPF alapú klienst is szolgáltatni. Fontos lehet még továbbá a szerver teljesítményének mérése és optimalizálása, hogy a játékmenetek végén robbanásszerűen nagy mennyiségű adatfelöltést jól tudjuk kezelni.

Irodalomjegyzék

[1] Emotiv Epoc Technical Specifications
http://emotiv.com/upload/manual/EPOCSpecifications.pdf
(2013. okt)

[2] The science of brainwaves – The langauge of the brain,
http://www.nhahealth.com/science.htm
(2013. okt.)

[3] Brain fingerprinting
http://en.wikipedia.org/wiki/Brain_fingerprinting
(2013. okt.)

[4] What Is Windows Communication Foundation
http://msdn.microsoft.com/en-us/library/ms731082.aspx
(2013. okt.)

[5] Szita Ádám: Android kliens fejlesztése elosztott workflow rendszerhez - Adatfeldolgozó- és kommunikációs alrendszer megvalósítása

[6] Chris Berka et al.: EEG Correlates of Task Engagement and Mental Workload in Vigilance, Learning, and Memory Tasks