Offline szituációs játék informatikai rendszerek oktatásához

Korom Szilárd[1], Illés Zoltán[2]

[1]korom.szilard@inf.elte.hu, [2]illes@inf.elte.hu
ELTE IK

Absztrakt. Már számos informatikai fogalom, tananyagelem oktatására született olyan offline, a valóságban eljátszható módszertan, mely segíti a diákokat a megértésben. Például rendezési algoritmusokra, számrendszerekre vagy adatstruktúrákra. Az alábbiakban egy olyan szituációs játék kereteit kívánjuk bemutatni, mely a szoftverrendszerekre, hálózati kommunikációra fókuszál úgy, hogy különböző programozható eszközök együttes működését kell szimulálni. Reményeink szerint a játék izgalmas módon mozgatja meg a diákokat, miközben nehéz, de annál fontosabb informatikai fogalmak megértését segít az általános és középiskolás korosztályban. A játék egy okos otthon szimulációja tantermi környezetben, mindenféle informatikai eszköz használata nélkül.

Kulcsszavak: CSUnplugged, számítógép nélkül, okosotthon, informatikai rendszerek, hálózati kommunikáció, offline játék

1. Bevezetés

A számítógép nélküli oktatás lényege, hogy a diákok játékosan szerezhessenek valós tapasztalatot a lényeges informatikai fogalmakról. A módszerrel szembeni legfontosabb követelmények, hogy ne kelljen hozzájuk semmilyen digitális eszköz, illetve, hogy minimálisak legyenek az előkövetelmények. A lényeg tehát, hogy olyan problémákat kelljen eljátszaniuk, melyek megoldásához nagyrészben a saját intuícióikra hagyatkoznak.

Ezen játékok „központján" [1] rengeteg példát találhatunk, összesen több mint 20 nyelvre lefordítva. A „*CSUnplugged*" kifejezés az 1990-es évekre megy vissza. Az első komoly kötet Tim Bell, Mike Fellows és Ian Witten szerzők által írt "Computer Science Unplugged: Off-line activities and games for all ages" kötet volt [2]. A játékok közül sok magyar nyelven is elérhető [3]. A témában számos cikk is készült az évek folyamán [4, 5, 6], melyek nagyon változatos témákat fednek le az informatikán belül. A rendezési algoritmusok például egyáltalán nem könnyűek, azonban létezik több igen egyszerű, tantermi körülmények között is eljátszható gyakorlata [7]. Például, ha a diákokat megkérjük, valószínűleg igen hamar sorba tudnak állni magasság szerint. A tényleges algoritmus valószínűleg nem tudatosul bennük, de ha megfigyelik a saját tevékenységüket, a használt módszer formalizálható. A rendezési algoritmusok többsége pedig a valóságban is eljátszható, így a tanár felvethet, körbeírhat módszereket, instrukciókkal terelgetheti a diákokat, hogy máshogyan is kipróbálják, hogyan lehetne sorba állni.

Az offline játékoknak mára már kiterjedt szakirodalma van. Természetesen tantárgyakon átívelve is alkalmazható [8]. Természetesen nem kell/lehet mindent lecserélni ilyenre, de mindenképpen a tananyagba illeszthető. Ez egyébként egyes tankönyvek esetében már meg is történt [9, 10], illetve kódolással foglalkozó tanulást támogató rendszerek is átvették, mint például a code.org [11]. Persze az, hogy sokan használják, nem jelenti azt, hogy előnyös is. Mára már azonban kijelenthetjük, hogy a módszer bizonyítottan hatékony [12].

2. A szituációs játék ismertetése

A szituáció az, hogy egy tanteremben vagyunk a diákokkal és elképzeljük, hogy kiépítünk egy okosotthon projektet, ahol az informatikai eszközök, a vezérlők, szenzorok, átviteli eszközök, átjárók és megjelenítési megoldások (monitorozás) mind-mind diákok. Például a terem különböző pontjain szeretnénk mérni a hőmérsékletet, akkor oda kell állnia a diáknak, aki „méri" a hőmérsékletet (valóságban kitalál egy számot, ami hozzávetőlegesen reális), egy másik diák a kapcsolónál áll és a lámpát irányítja, megint másik diák a táblára tudja felírni az értékeket. A játék célja, hogy minél hatékonyabb felosztást és kommunikációs formákat találjanak ki a rendszer működtetésére. A kulcs az, hogy a rendszer bővítésével, bonyolításával milyen módon kell a különböző „komponenseket módosítani". Miközben a feladat mélyül és a diákok új és új megoldásokat találnak ki, új szereplőket vonnak be a játék egy szoftver architektúrájává válik át. A diákok fejében persze nem (feltétlenül) fogalmazódik meg a konkrét informatikai analógia. Erre a folyamatos reflexió és a tanári terelgetés enged teret. Például a diákok által eljátszott szerepek elemzésével megválaszolható, hogy ezt hány program, hány hardver tudná megvalósítani.

3. Informatikai háttér

Először be kívánjuk mutatni, hogy konkrétan milyen informatikai eszközök és ismeretek szükségesek ahhoz, hogy ténylegesen leimplementáljunk egy megoldást a fent említett problémára. A bemutatás nem teljeskörű, s nem részletekbe menő, hiszen az kívül esik a cikk keretein, de egy átfogó képet kívánunk adni.

3.1 Szenzorok/vezérlők

A szenzorok és vezérlők olyan alacsony-szintű informatikai eszközök melyek képesek jelet fogni, vagy küldeni a valós világban. Például hőmérséklet, fény vagy távolság érzékelő szenzor, de ide értünk egy egyszerű LED-et, vagy kapcsolót, esetleg motort is. Ezen a szinten nem programozható célszámítógépekről van szó, ezeknek az eszközöknek szükségük van egy közvetlen vezérlőre.

3.2 Helyi feldolgozás

Az előző réteg vezérlőiről van szó, melyek elsősorban M2M kommunikációra alkalmasak, vagyis vagy folyamatos futás mellett biztosítanak valamiféle értéket, vagy egy irányított kérésre cselekednek (pl.: kapcsolnak fel egy lámpát, küldenek vissza adatok).

Ezen a szinten beszélhetünk áramköri megoldásokról is, hiszen a perifériákkal valahogyan össze kell kötni a vezérlőt. Természetesen „ready-to-use" eszközök is a rendelkezésünkre állnak GPIO-val ellátva (Általános célú be- és kimeneti kapcsolatok). A legelterjedtebb ezek közül, melyek oktatási célokra is felhasználható az Ardunio és a Raspberry Pi számítógépek. Előbbivel beágyazott-rendszereket, utóbbival magasabb-szintű, asztali számítógép működéséhez nagyon hasonló termékeket lehet készíteni.

3.3 Hálózat és internet, adattovábbítás

A kommunikációs réteg legfontosabb kérdése, hogy mit jelent a kommunikáció. Az IoT eszközök esetében a hálózati protokoll adja meg a választ. A leggyakoribbak talán:

- MQTT vagy AMQP
- HTTP vagy REST

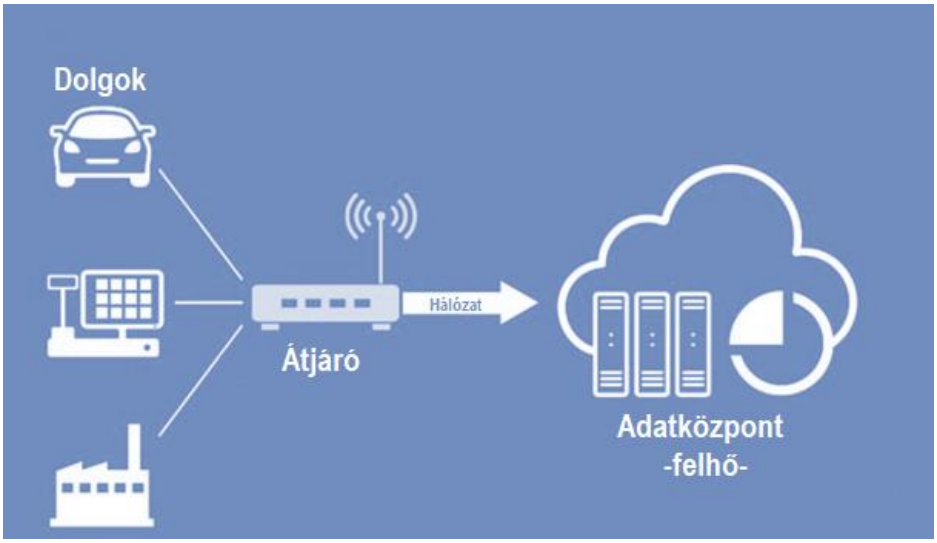
3.3.1 Az MQTT és az AMQP protokoll

Ezeket a protokollokat (illetve ezt a típust) azért is tartjuk fontosnak kihangsúlyozni, mert a valós életbeli kommunikáció talán erre, a publish-subscribe mintára hasonlít a leginkább. Ez a protokoll

feliratkozásokról és publikálásokról szól. Az ötlet az, hogy aki az üzenetet továbbítja nem foglalkozik azzal, hogy ki fogja fogadni az üzenetet, mondhatni csak elkiáltja magát, s aki feliratkozott (aki figyel) majd megkapja, s feldolgozza az üzenetet. A működési elv az, hogy van egy központi szoftver (úgynevezett broker), aki fogadja a publikált adatokat és elküldi azoknak, akik az adott fajta üzenetre feliratkoztak. Ez tehát merőben eltér a kérés-válasz struktúrától, hiszen aki az adatot küldi, az ezt nem kérésre teszi, ráadásul a kliens nem is tudja, hogy azt ki fogja fogadni (és hogy aki akarta, az ténylegesen megkapta-e). Általánosságban ez a sokeszközös világban igen elterjedt, így az okosotthon megvalósításoknál is. Például egy hőmérséklet érzékelő szenzornak nem feltétlenül kell tudnia róla, hogy ki kíváncsi az adatra. A szenzor továbbítja egy központi alkalmazásnak (az okosotthon vezérlőnek), amiktől adatokat kérnek a felhasználók egy webes felületen, vagy egy mobil alkalmazás segítségével.

3.4 Általános IoT architektúra

Egy valós IoT megoldás általános architektúrája tehát az 1. ábrán látható:

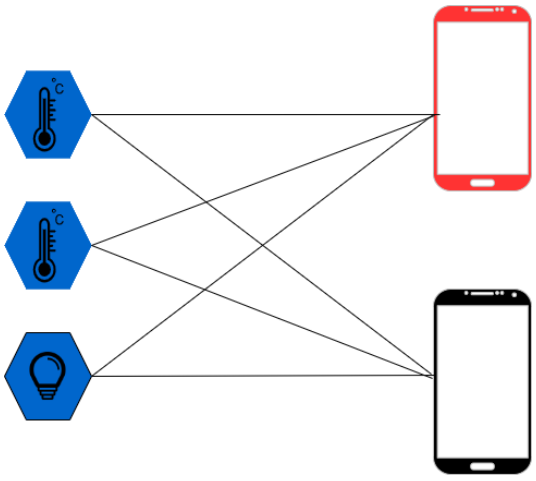


1. ábra: IoT megoldások általános architektúrája

3.5 Konkrét architektúra

A továbbiakban azt szeretnénk bemutatni, hogy a mi konkrét okosotthon megoldásunknak az architektúrája hogyan nézhet ki. Nem kívánjuk teljesen részletezni őket, hiszen nem ez a cikk fő fókusza. Természetesen más megoldások is elképzelhetőek. A példák azért fontosak, mert tulajdonképpen ezeket szeretnénk eljátszatani a diákokkal. A játék során pedig a diákoktól azt várjuk el, hogy ezeket tervezzék meg a saját tapasztalatok alapján.

3.5.1 Okosotthon architektúra szerver nélkül



2. ábra: Okosotthon architektúra szerver nélküli változata

Ebben az esetben összesen 3 alkalmazást kell készítenünk:

- A hőmérséklet érzékelők beágyazott rendszerek, hiszen csak adatot mérnek és küldenek tovább. Alacsonyszintű program fut rajtuk.
- A megvilágítás szintén beágyazott rendszer, hiszen csak egy egyszerű kérést kell kiszolgálnia és kezelnie. Alacsonyszintű program fut rajtuk.
- Mobil alkalmazás, mely képes feltérképezni a hálózaton elérhető eszközöket, képes velük kommunikálni. Mivel a mobil eszközön van operációs rendszer, nem alacsonyszintű programról van szó. Megjelenéssel itt foglalkozni kell.

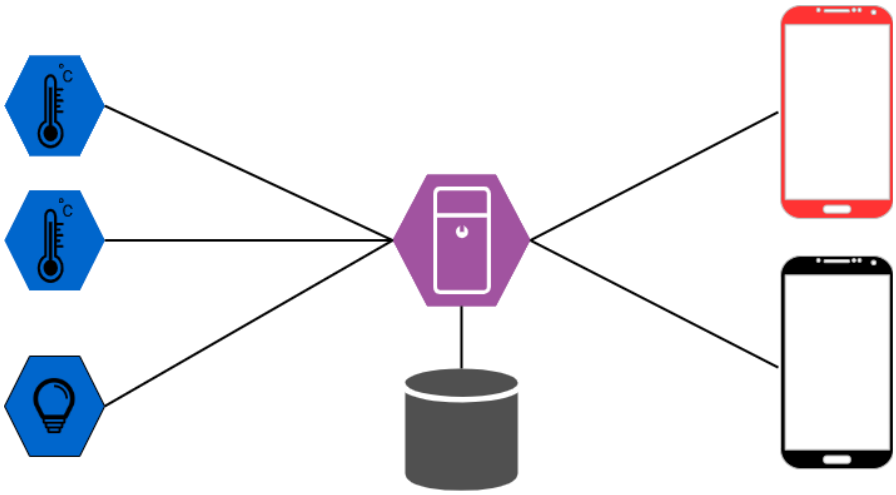
Ezzel a megoldással találkozhatunk a leggyakrabban a normál felhasználói környezetben. Előnye, hogy a megvalósítás olcsó és rugalmas, hiszen nem igényli egy szerver működtetését és fenntartását, integrálása könnyű. Számos olyan eszközzel találkozhatunk a piacon, amely megvétel után egy alkalmazás letöltését követően már működőképes és használható.

Hátránya, hogy a rendszer nem integrált. A valóságban ez azt jelenti, hogy ahányféle eszközt veszünk, annyiféle alkalmazást kell letöltenünk. Az eszközökkel való kommunikáció erősen korlátozott, ha sok klienst szeretnénk működtetni, más problémákba is ütközhetünk. Ha komplexebb megoldást keresünk, ahol már az eszközök együttes működése szükséges döntési logikával, akkor az architektúra alkalmazhatatlan (például azt szeretnénk, hogy a hőmérséklet függvényében kapcsoljon be a fűtés).

Mivel az eszközök teljesen különállóak, annyiféle alkalmazást kell fejlesztenünk ahányféle kliensünk van. Ha például szeretnénk webes felületen számítógéppel elérni az eszközöket, fejlesztenünk kell egy olyan alkalmazást, mely minden lehetséges komponenssel tud kommunikálni. Ha egy újfajta szenzort építünk a házba, azt megint külön kezelni kell.

További hátrány, hogy a szerver hiányában minden egyes eszköznek magának kell gondoskodnia a biztonságról. Az IoT világ egyik legnagyobb problémája a biztonság megfelelő fenntartása [13].

3.5.3 Okosotthon architektúra szerverrel



3. ábra: Okosotthon architektúra szerveres változata

Ebben az esetben 4 alkalmazást kell készítenünk. A korábbi hármat, valamint egy szervert. A szerverprogram képes a szenzorokkal és vezérlőkkel kommunikálni, döntéseket hozni, valamint adatot kiszolgálni és tárolni (például egy adatbázisban), megjelenés nem szükséges. A szerverprogram semmiképpen nem alacsonyszintű, hiszen több szálon is kommunikálnia kell és változatos feladatokat el kell látnia.

Előnye, hogy megfelelő tervezés esetében nagymértékben függetleníthető attól, hogy kitől kap és ad adatot. A rendszer tehát skálázható és könnyen bővíthető újfajta eszközökkel.

A rendszer továbbá biztonságos, hiszen a szenzoroknak csak egyetlen eszközzel kell kommunikálniuk, mondhatni, hogy elfedi az alsó réteget a struktúra.

Hátránya, hogy mind a készítése, mind a fenntartása költséges, hiszen az eszközök egyáltalán nem függetlenek egymástól. Úgy kell megválasztani minden egyes komponenst, hogy azok alkalmasak legyenek egy saját szervermegoldással való integráláshoz. Szükség van továbbá egy szervergépre, melyen a programunk fut, melynek állandóan elérhetőnek kell lennie.

4. A játék leírása

Az informatikai rendszerek működésének eljátszási problémája, hogy egy program, egy valós eszköz merőben eltér attól, hogy mi emberek a valóságban hogyan cselekednénk. Ha például kijelölünk valakit, aki a terem egy általunk választott pontján méri a hőmérsékletet, majd az eredményt fel kell írnia a táblára, akkor valószínűleg ezt úgy oldaná meg, hogy kitalál egy számot, majd odasétál a táblához és felírja. Informatikai rendszerek analógiájában azonban annál sokkal egyszerűbb megoldások is léteznek, minthogy építsünk egy programot és robotot, mely képes odasétálni és még rajzolni is. A probléma felvetése azonban nem hasztalan, hiszen ez rámutat az informatikai és hétköznapi gondolkodás különbségére.

A fő probléma tehát az, hogy a kommunikáció és az érzékelés merőben más gépek esetében. Például a diáknak nem tudunk olyan instrukciót adni, hogy ne lássa, hallja a többieket, de mégis eltudjon olvasni bizonyos üzeneteket.

A végső cél tehát hogy a diákokkal, mint szereplőkkel egy olyan rendszert építsünk fel, mely egy valós okosotthon informatikai megoldását modellezi.

4.1 Módszertan

A tervezést a diákoknak kell csinálniuk, s bonyolultsága fokozatosan mélyül, rétegről rétegre. A módszer alapja a probléma felvetése, majd kérdésekkel való terelgetése. Egy lehetséges kérdéssorozatot mutat be az 1. táblázat.

Felvetés - javítás	Várt kimenet	Játék	Informatikai kapcsolat
Milyen szereplőkre van szükségünk, ha szeretnék a terem 2 sarkában mérni a hőmérsékletet, szeretnék a lámpát kapcsolgatni, s szeretnénk, hogy egy felhasználó a táblánál tudja ezt vezérelni, mintha a táblán lévő rajz lenne maga a vezérlő alkalmazás?	Valaki megtervezi hogyan nézzen ki az alkalmazás (felrajzolja a táblára). A hőmérsékletmérők 1-1 diákok, valaki a kapcsolónál áll. Létezik felhasználó, aki megnyomja a táblán lévő rajzolt gombot, vagy elkiáltja magát, ha adatra van szüksége.	Mindenki nézi, hogy a táblánál lévő felhasználó mit szeretne (megnyom egy gombot, kér valamit), s külön-külön reagálnak rá.	Kérés-válasz protokollok (pl.: http). Több eszköz létezik különböző céllal. Grafikus megjelenést is tervezni kell. Létezik hálózat. Felvethető a peer-to-peer kapcsolat.
A probléma az előző rendszerrel, hogy valójában a gombot nem láthatják az érzékelők/vezérlők és valójában beszélni sem tudnak.	Kijelölnek egy új diákot, aki az alkalmazást vezérli és továbbítja a kérést a vezérlők felé, illetve felrajzolja, ha érkezik információ.	Ha egy felhasználó megnyomja a gombot, már a vezérlő diák beszél és csak a kijelölt „célszközzel" és ugyanez fordítva.	Szerver-kliens kapcsolat, de még adatbázis nélkül. Publish-subscribe alapú kommunikáció.
Folyamatosan szeretnénk látni a hőmérsékletet.	Az érzékelő diákok folyamatosan hangosan mondják mi van náluk.	Az eredmény túl káoszos, nehezen kivehető, ráadásul a rajzoló diák nem tudja követni.	Adatovábbító réteg, gateway, queue. A folyamatos adatmondogatás felel meg az MQTT/AMQP protokolloknak.
Oldjuk meg, hogy ne legyen hangzavar, ne akadjanak össze az információk.	Kijelölnek adattovábbító diákokat, akik viszik az információt az érzékelők és a monitorozó diák között.	Az adattovábbítók viszik az információt, így nincs hangzavar. Ha egyszerre érnek oda, sorba kell állniuk.	Nagyon szigorú architektúrális és kommunikációs szabályok, a zaj kiszűrésére és a hatékony adattovábbításra. Minden egyes komponensnek nagyon jól behatárolt feladata van.
Több vezérlő és felhasználó legyen (akár egy tényleg okosotthon projektnél).	A diákok kettéosztják a táblát és bevonnak egy új vezérlő embert, adattovábbító embereket.	A kommunikáció lelassul, mert a szenzoroknak és kapcsolóknak több adatot kell szolgáltatni.	

1. táblázat: Egy lehetséges menete a játéknak

Konkrét útmutatót a játékkal kapcsolatban nehéz adni, mert az egyes felvetések attól függenek, hogy a csoport a korábbira milyen megoldást ad. Elképzelhető például, hogy a diákok már a legelején egy elég jóltervezett struktúrát adnak:

- Van vezérlő ember
- Vannak adattovábbítók
- Van rajzoló ember
- A kommunikáció fegyelmezetten és a szabályok szerint történik

Persze ebben az esetben is lehet még fejleszteni, új komponenseket illeszteni a rendszerbe, de a folyamat menetét akár meg is fordíthatjuk, hogy direkt egy rossz megoldás eljátszását kérjük tőlük s azt, hogy fogalmazzák meg miért nem működött ez a megoldás.

4.2 Hangsúlyok

A feladat eredeti formájában a hálózati kommunikációra és a szoftveres architektúrákra helyezi a hangsúlyt. Ez persze módosítható. A lényeg tehát az, hogy a tanár milyen kérdéseket tesz fel, merre terelgeti a diákokat:

- **Algoritmusok**: Pontosan milyen „logikát" kell az egyes diákoknak (szenzorok, kapcsolók, vezérlők, rajzolók) megvalósítaniuk?
- **Hardverek**: Pontosan milyen hardverekre lenne szükségünk, ha ezt ténylegesen meg szeretnénk csinálni? Mik a követelmények a hardverekkel szemben, mit kellene tudniuk?
- **Szoftverrendszer**: Milyen szoftvereket kellene írnunk, hogy ezt megvalósítsuk? Hogyan kezeljük szoftverszinten a felmerülő problémákat (adatvesztés, hibás adat, feltorlódó kérések)? Milyen jellegűek legyenek a megvalósítások, milyen paradigmákat kellene használni (pl.: konzolos, webes, mobilos, ablakos)? Szükség van-e operációs rendszerre az egyes hardvereken? Ha igen, milyenre? Ha nem, miért nem?
- **Adatok**: Milyen adatok tárolására van szükség? A tárolást és az adatok menedzselését ki végezze? Milyen adatcsomagok menjenek át a kommunikáció során? Ezek az adatcsomagok hogyan nézzenek ki?
- **Programnyelvek**: Milyen programozási nyelvek lennének a hasznosak a megvalósítás során? Mely programozási környezetekkel lehetne a különböző komponenseket megvalósítani? Milyen programozási paradigmákat használhatunk?

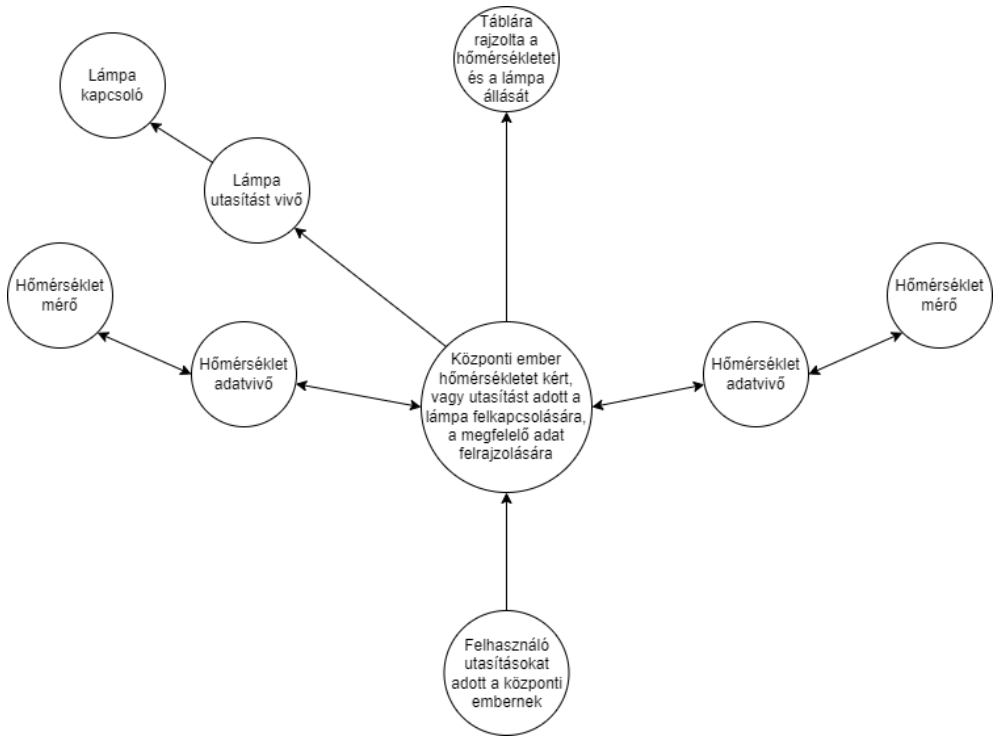
5. Éles tesztelés

5.1 A történtek leírása

A játékot kipróbáltuk az Eötvös Lóránd Tudományegyetem első féléves tanárszakos hallgatóival. A foglalkozáson 14 hallgató volt jelen és 60 percet vett igénybe. A foglalkozás végén egy kérdőívet is kitöltöttek.

A játék ebben a formában nyilván nem a legtermészetesebb, hiszen a tanárszakosok már informatika érettségivel rendelkeznek, tisztában vannak a játék által bemutatott fogalmakkal. Ettől függetlenül az ott nyert tapasztalat hasznos volt. A hallgatóktól egyszerre kértük a részvételt, mint „diákok", s mint megfigyelő „tanárok". A foglalkozás folyamán folyamatosan reflektáltunk a saját munkánkra és értékeltük mennyire volt hasznos az adott szekció.

A játék a szituáció bemutatásával, a feladat ismertetésével kezdődött. A közbeavatkozásunk nélkül a hallgatók nagyjából a következő architektúrát találták ki és valósították meg (minden kör egy embernek felel meg):



4. ábra: Hallgatók által kitalált első architektúra

A tervezéstől egészen odáig, hogy ténylegesen eljátszották és késznek tekintették a projektet nagyjából 20 perc telt el. Ezalatt közös ötletesés, illetve megbeszélés, finomítgatás történt, majd a tényleges játék. Nyilván az életkor és a szituáció miatt is nagyon fegyelmezettek és összeszedettek voltak, mely például egy általános vagy középiskolában nem várható el.

A foglalkozás közben nagyon kevés szerepünk volt. Amikor 1-1 fázissal készen voltak, adtunk egy következő fejlesztési lehetőséget, illetve tanulságot vontunk le velük közösen. (Hatékony volt-e így? Lehetne-e máshogy? stb.) Hogy konkrétan mi történt, a 2. táblázatban látható.

Felvetés - javítás	Tényleges kimenet	Tanulság	Informatikai kapcsolat
Lépjen be a játékba egy új felhasználó.	Az előzőkhez képest az az érdekes jelenség történt, hogy amikor a két felhasználó majdnem ugyanakkor kérte a lámpa felkapcsolását, akkor a központi embernek várnia kellett, mire visszaért a megfelelő adattovábbító ember.	A játék során ők problémának élték ezt meg, hiszen a valóságban itt hosszú másodpercek teltek el, de megbeszéltük, hogy persze lassabbak vagyunk, mint ahogyan valóban mennek az adatok, de gépek esetében is történik ilyen feltorlódás, amit kezelni kell.	http (elsősorban a böngészéshez hasonlítottuk); szerver-kliens kapcsolat; 4 programot kellene írnunk, ha ténylegesen meg akarnánk valósítani.
Legyen még egy megjelenítő (rajzoló).	Itt a központi ember már túl volt terhelve, nem tudta ellátni az összes feladatát, mert túl sok mindenkivel kellett volna egyszerre kommunikálnia.	A két rajzolást nem tudja egy ember csinálni még akkor sem, ha ugyanazt kell felrajzolni. A hallgatók körében felmerült, hogy ki kellene vezetni a központi embert a rendszerből.	Queue, megjelenítés teljes elkülönítése a szervertől.
Oldjuk meg adattovábbító ember nélkül (hangos beszéddel).	Sokan beszéltek egyszerre, de finomítgatások révén működött a rendszer.	A hallgatók érzékelték, hogy a probléma a kommunikáció. 1-2 valós teszt után nagyon szigorú szabályokat hoztak arra, hogy mikor ki beszélhet, illetve mindenkinek más hangszínt adtak, hogy meg tudják különböztetni ki mondta. Utána levontuk a tanulságot, hogy valós szoftverek esetében is valahogyan erősen el kell különíteni az üzeneteket, kell küldeni egy azonosító adatot is, hogy tudjuk honnan érkezett a hőmérséklet.	Hálózati protokollok; adatcsomagokra vonatkozó szabványok (adatleíró nyelvek); publish-subscibe protokoll.
Kérés nélkül folyamatosan lássuk a hőmérsékleteket a táblán.	A két hőmérséklet érzékelő ember hangosan mondogatta egyszerre az adatokat. Külön embert kellett behívni, aki csak a hőmérsékleteket írta folyamatosan.	Az előző szabályokat betartva elkülöníthetőek egymástól az adatok.	Többszálúság

2. táblázat[1]: Éles teszt alatt történtek

Felvetés - javítás	Tényleges kimenet	Tanulság	Informatikai kapcsolat
Oldjuk meg kettő központi emberrel.	Ténylegesen eljátszani nem tudtuk, mert nem volt annyi ember, amennyire szükség lett volna.	Feleslegesen növeli a kommunikációk és a felelősségek számát.	-
Oldjuk meg központi ember nélkül.	Megvalósult, sok hallgató feladat nélkül maradt.	A publish-subscibe mód ebben az esetben hasznosabb, mint a kérés-válasz szerinti. A rendszer rugalmasabb lett.	-

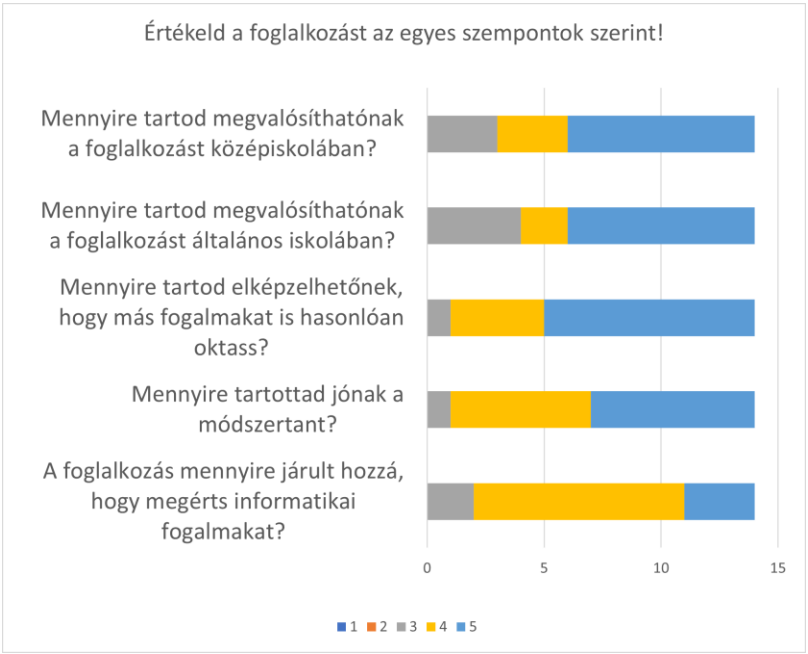
2. táblázat[2]: Éles teszt alatt történtek

A foglalkozás összességében szerintünk jól alakult és a várt eredményt hozta. A felmerülő problémák elsősorban architekturális vagy hálózati kommunikációs problémák voltak, s mindegyik analógiába állítható valamely informatikai fogalommal. A hallgatók (bár nem tudatosan), de szoftverarchitektúrát terveztek és igen kritikus, gyakori problémákkal szembesültek és adtak rá megoldást.

Nem utolsó sorban pedig a foglalkozás nagyon jókedvűen telt. A hallgatóknak együtt kellett dolgozniuk, közösen kellett megoldást találniuk és csak jól összehangolt munkával tudtak megoldást adni. Az egyes szituációk pedig gyakran viccesek voltak.

5.2 A kérdőívre adott válaszok

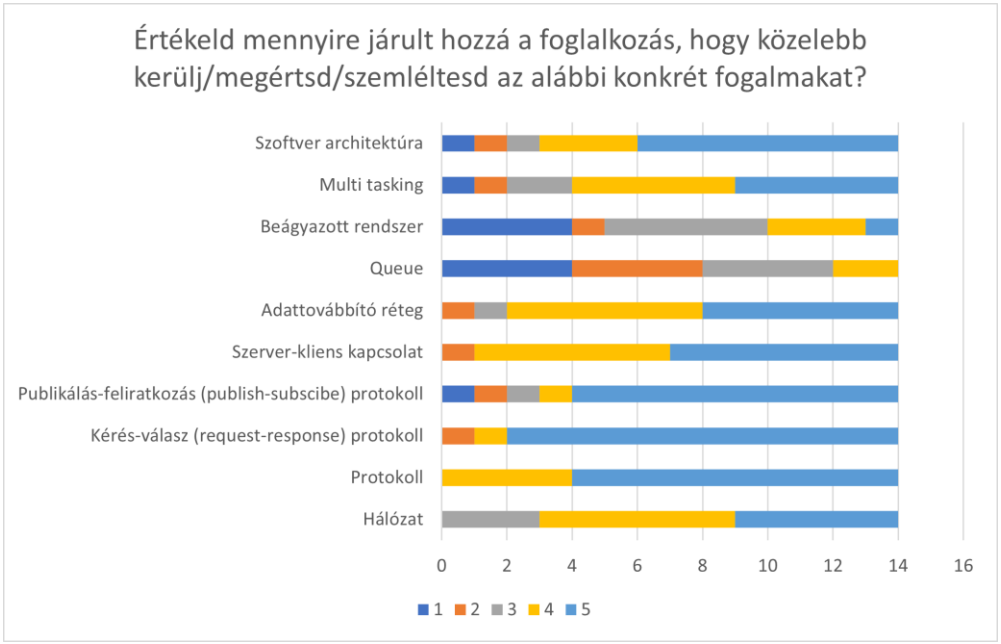
A következőkben a kérdőívben feltett kérdésekre adott válaszok láthatók.



5. ábra: Értékeld a foglalkozást az egyes szempontok szerint!

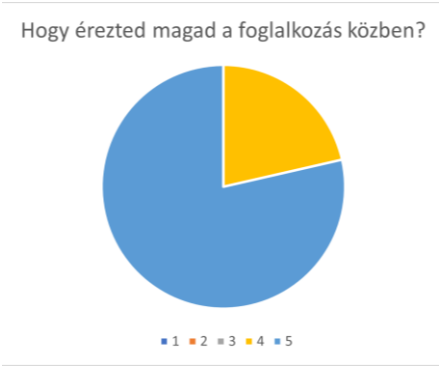
Az 5. ábrán látható diagram alapján a tanárszakos hallgatók jónak, kiválónak értékelték a foglalkozást. Örülünk a megerősítésnek, hogy általános és középiskolában is érdemes kipróbálni a módszert. Sajnos ez most kimaradt, de mindenképpen meg kellene ezt csinálni a téma teljessége kedvéért több szinten, több csoportban.

Már a 4-es jót jelent, azért nem lehet elmenni az utolsó kérdés eredményei mellett, ahol csupán hárman válaszoltak 5-össel; elképzelhető, hogy azért, mert már tisztában voltak a fogalmak nagyrészével, de természetesen az is lehet, hogy azt gondolják, nem ez a leghatékonyabb módja a megértetésnek. Ezzel egyet is tudunk érteni. A módszertan szerintünk sem a leghatékonyabb, de a legszemléletesebb. A szoftverarchitektúrák vagy hálózati kommunikációk általános iskolában való oktatása egyáltalán nem triviális és ritkán valósul meg. A módszertan egy új alternatívát kínál erre a problémára.



6. ábra: Értékeld mennyire járult hozzá a foglalkozás, hogy közelebb kerülj/megértsd/szemléltesd az alábbi konkrét fogalmakat?

A 6. ábra eredményei alapján kijelenthető, hogy a hallgatók szerint a foglalkozás általában hozzájárult különböző informatikai fogalmak megértéséhez. A hálózati protokollokkal kapcsolatos kérdések különösen jól szerepeltek, ami azért lényeges, mert a foglalkozás alatt mi erre kívántuk helyezni a hangsúlyt. A „queue" fogalma maradt csak alul, ami érthető, mert éppen csak megemlítésre került, különösen nem beszéltünk róla.



7. ábra: Hogy érezted magad a foglalkozás közben?

8. ábra: Összességében milyennek értékeled a foglalkozást?

A 7. és 8. ábrán az látható, hogy a hallgatók összességében jól érezték magukat a foglalkozáson, és azt hasznosnak találták.

6. Konklúzió

Az informatikai gondolkodást számítógép nélkül is lehet oktatni és demonstrálni. A cikkben bemutattunk egy olyan szituációs játékot, melyet általános és középiskolás korosztályban is alkalmazni lehet, s a szoftverrendszerek, valamint a hálózati kommunikáció tervezésére helyezi a hangsúlyt. A gyakorlat azért is lehet hiánypótló, mert ezeket a fogalmakat nagyon nehéz demonstrálni.

A módszer magában hordoz módosítási lehetőségeket (melyek közül párat mi is bemutattunk), így a tanár egy-egy kiemelt fogalomhoz, a korosztályhoz és az előképzettséghez tudja igazítani a játékot.

Irodalom

1. Tim Bell, Ian H. Witten, Mike Fellows: *CS Unplugged* https://classic.csunplugged.org/documents/books/english/CSUnplugged_OS_2015_v3.1.pdf (utoljára megtekintve: 2021.11.19)
2. Bell, T., Witten, I.H., Fellows, M.: *Computer Science Unplugged: Off-Line Activities and Games for All Ages (Original Book)* (1999)
3. Erdősné Németh Ágnes: *Számítógép nélküli tevékenységek a számítástudomány alapjainak bemutatására és az algoritmusok tanításakor* In: Szlávi, Péter; Zsakó, László (szerk.) INFODIDACT 2017 (Budapest, Magyarország: Webdidaktika Alapítvány) – ISBN 978-615-80608-1-3, Paper: No 6, 10 p. (2017) http://konferenciak.inf.elte.hu/infodidact/InfoDidact17/Manuscripts/ENA.pdf (utoljára megtekintve: 2021.11.30.)
4. Duncan, C., Bell, T.: *A pilot computer science and programming course for primary school students.* In: WIPSCE 2015, pp. 39–48 (2015)
5. Yvon Feastery, Luke Segarsz, Sally K. Wahbay, and Jason O. Hallstromy: *Teaching CS Unplugged in the High School (with Limited Success)*, ITiCSE 2011, Darmstadt
6. Erdősné Németh Ágnes: *Teaching Graphs for Contestants in Lower-Secondary-School-Age.* In: OLYMPIADS IN INFORMATICS 11: 1 pp. 41–53., 13 p. (2017)

7. Bernát Péter: *The Methods And Goals Of Teaching Sorting Algorithms* In: Public Education In ACTA DIDACTICA NAPOCENSIA 7: 2 pp. 1-10., 10 p. (2014)
8. Bell, T., Rosamond, F., Casey, N.: *Computer science unplugged and related projects in math and computer science popularization*. In: Bodlaender, H.L., Downey, R., Fomin, F.V., Marx, D. (eds.) The Multivariate Algorithmic Revolution and Beyond: Essays Dedicated to Michael R. Fellows on the Occasion of His 60th Birthday. LNCS, vol. 7370, pp. 398–456. Springer, Heidelberg (2012).
9. Clarke, B.*: Computer Science Teacher*. British Computer Society, Swindon (2017)
10. Hazzan, O., Lapidot, T., Ragonis, N.: *Guide to Teaching Computer Science: An Activity-Based Approach*. Springer, London (2011)
11. Code.org lesson: *Lesson 4: Dance Party: Unplugged* https://curriculum.code.org/hoc/unplugged/4/ (utoljára megtekintve: 2021.11.19)
12. Bell T., Vahrenhold J. (2018) CS Unplugged—How Is It Used, and Does It Work?. In: Böckenhauer HJ., Komm D., Unger W. (eds) *Adventures Between Lower Bounds and Higher Altitudes. Lecture Notes in Computer Science*, vol 11011. Springer, Cham. https://doi.org/10.1007/978-3-319-98355-4_29
13. Hassan W.H. *Current research on internet of things (iot) security: A survey. Comput.* Netw; 148:283–294 (2019)