A programozás elektronikus oktatásának kérdései

HABILITÁCIÓ TÉZISFÜZET

Király Sándor

Eötvös Loránd Tudományegyetem
Informatikai Kar
Budapest, 2020

1. Tartalomjegyzék

2. Bevezetés

Napjainkban az IT ipar vonzó karrier lehetőséget tud ajánlani a szoftverfejlesztők számára, miközben a felsőoktatásból kikerülő diákok nagy része nem rendelkezik olyan képességekkel és kompetenciákkal, melyeket az IT vállalatok elvárnak. Ráadásul a kikerülők száma sem elegendő az IT vállalkozások számára, hiszen nagymértékű a lemorzsolódás is. Például az ELTE Informatikai Karán 2010 és 2016 között átlagosan a beiratkozott hallgatók 44,5%-a diploma nélkül hagyta el az egyetemet (Takács et al., 2017). A jelenség és okainak feltárására már több kutatás született, melyekből kiderül, a problémát több tényező okozza (Ulriksen et al., 2010). Például a **nem megfelelő előképzettség** és az **alacsony motiváció**.

Tapasztalataink szerint azok a hallgatók, akik már középiskolában elsajátították a számítógépes programozást, az első tanévben sokkal könnyebben boldogulnak a programozással kapcsolatos tárgyakban az egyetemeken, így több idejük marad más tárgyak, például a matematikai tárgyakra is. Azaz a probléma egyik megoldása lehet, hogy az informatika iránt érdeklődő középiskolások megtanulják a programozást még a középiskolában, így lehetőségük lesz emelt szintű érettségi vizsgát tenniük. Ehhez kapcsolódó érdekes adat, hogy az ELTE Informatikai Karának 2016-os évfolyamán a diákok 25%-a nem tett emelt érettségi vizsgát (Takács et al., 2017).

Miért ilyen nehéz a programozás? Az egyik ok az, hogy a programozás tanulása nemcsak egy nyelv szintaktikájának és szemantikájának az elsajátítását jelenti, hanem az algoritmizálási ismeretekét is. Nem véletlen, hogy a felsőoktatásban külön tantárgy foglalkozik a programozási nyelvekkel és az algoritmizálással is, ellentétben a középiskolákkal, ahol az alacsony óraszámok miatt többnyire a középszintű érettségire történő felkészítés dominál, melynek teljesítéséhez nincs szüksége a diákoknak programozási ismeretekre. Miközben sok tehetséges diákot érdekel a programozás, de ez tanári segítség nélkül sokszor nehezen megy, a legtöbb középiskolában a programozás oktatására nincs elegendő kapacitás. Erre az egyik megoldás a programozás online oktatása, amelyet már 2014-ben javasoltunk (Király, 2014).

Bár rendelkezésre állnak különböző könyvek, jegyzetek magyarul, valamint elérhetők angol nyelvű kurzusok is, de ez az út nem járható a többség számára. Olyan portálok mint a CodeAcademy, CodeSchool, Khan Academy, Coursera, Udacity vagy a Code Avengers, melyek különböző programozási nyelveket (is) tanítanak interaktív formában, angolul, több magyar diák számára is hasznosak lehetnek. Sokaknak azonban ezek a kurzusok túl nehezek, illetve nem tudnak még annyira jól angolul.

Ezért kutatásunk célja az volt, hogy olyan elektronikus környezeteket hozzunk létre, melyek lehetővé teszik a fiatalok számára akár a teljesen önálló programozás tanulást, miközben megkedveltetik velük a kódolást is.

Ennek eredményeként a **memooc.hu** portálon (amely a Miskolci Egyetem portálja) nemcsak C, hanem Java programozási nyelvet is lehet tanulni magyarul (Kusper et al, 2016a, 2016b). Ehhez csak be kell iratkozni az általam készített **Programozási nyelvek alapjai**, valamint ennek folytatása, az **Objektumorientált programozá**s kurzusokra. Amikor a két kurzus elérhetővé vált, már közel 100 diák tanult C# nyelven programozni az általunk készített **kodolosuli.nejanet.hu** URL-en elérhető tananyagból, majd később már a bárki által ingyenesen elérhető **kodolosuli.hu** portálon.

A magyar informatika oktatásának legfontosabb szabályozója a Nemzeti alaptanterv (Nat) (Török és Király 2017). A 2020-as Nat-ban már a 3-4. évfolyamon megjelenik a „Robotika és a kódolás

alapjai", az 5-8. évfolyamon pedig az „Algoritmizálás és blokkprogramozás" témakör, melyek segíthetnek megalapozni a középiskolai programozást. A középiskolai kerettantervekben pedig szerepel a formális programozási nyelv használata a 9-11. évfolyam számára. A változások évek múlva eredményezhetik azt, hogy minden végzett középiskolás tud valamilyen szinten programozni, illetve rendelkezik fejlett algoritmizáló képességgel.

A következő fejezetekben az alábbi tézisekkel kapcsolatos eredményeket mutatom be:

1. tézis: Egy már gamifikált programozási tananyag hatékonysága is javítható oktatási célú játékokkal.

2. tézis: Az automatikus programkiértékelők nemcsak a programok tesztelésére, ellenőrzésére használhatók, hanem a tanulási folyamatot is felgyorsíthatják, ha a tesztesetek célját, a tesztadatokat és a megoldási útmutatót is elérhetővé teszik.

3. tézis: Egy online programozási kurzusban használt programkiértékelő rendszer megvalósítható úgy is, hogy négy aspektus szerint is képes legyen a beküldött kódok ellenőrzésére.

4. tézis. Olyan programkiértékelő rendszer is készíthető, amely GUI osztályokat tartalmazó program ellenőrzésére is alkalmas.

3. Elektronikus környezet online programozáshoz

Napjainkban az online oktatás egyik legnépszerűbb változatát a MOOC (*Massive Open Online Course*) kurzusok jelentik, melyek egy újfajta megközelítést kínálnak a távoktatás és az online oktatás területén. A MOOC kurzus egy online tanítási program, amely nem limitált számú résztvevő számára biztosítja a tradicionális kurzus anyagokat weben keresztül. Egy jellemző MOOC tanfolyam videó előadásokból, olvasóanyagból és a hallgatók számára könnyen elérhető tesztekből áll, valamint interaktív fórumokat biztosít a tanulók számára, így támogatva a tudásmegosztást, a csoportmunkát és a kommunikációt a tanulók, a tanárok és a tanítást támogató személyek számára, ezzel is fejlesztve a nem kognitív funkciókat (Arefin et al., 2015; McDowell et al., 2002; Teague & Roe, 2008). A programozás tanulása azonban többet jelent, mint oktató videók nézése, tájékoztató anyagok olvasása vagy tesztek kitöltése. Ez a tanulási folyamat algoritmikus gondolkodást, probléma megoldási képességeket igényel, ráadásul hosszú ideig tartó folyamat (Muratet et al., 2011). Különbséget kell tenni a programozási ismeretek (például képes valaki megállapítani, hogyan működik egy „while" ciklus) és a programozási stratégiák (képes valaki „while" ciklust használni) között. A diákok számára nehézségeket okozhat a programozási utasítások (feltételek, ciklusok stb.) együttes használata. Winslow (Winslow, 1996) megfigyelte, hogy azok a hallgatók, akik megértik az egyes programozási utasítások szintaxisát és szemantikáját, néha nem tudják szoftverkóddá alakítani, felhasználni azokat.

A programozás tanulása tehát megköveteli a hallgatóktól, hogy gyakorlati készségeket szerezzenek (Robins et al., 2003). Következésképpen az online környezetben történő programozási tananyagoknak kódolási feladatokat kell tartalmazniuk, valamint biztosítani kell az elkészült programok kiértékelését, azokról gyors visszajelzés küldését (Vihavainen et al., 2012).

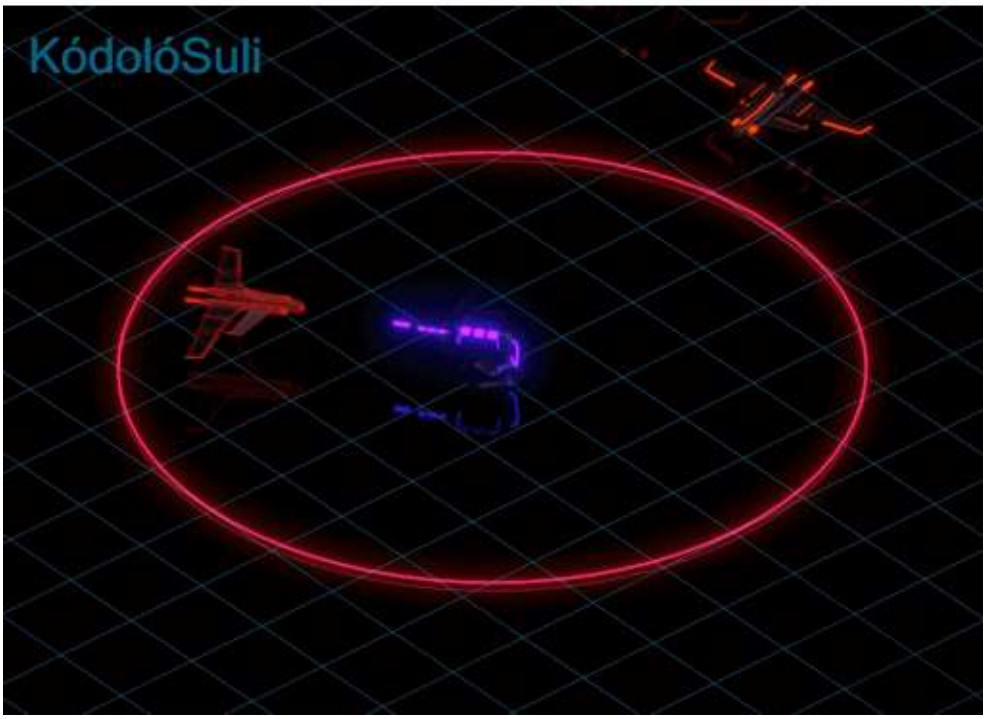
Egy, a fentieknek eleget tévő kurzus készítéséhez szükség van egy tartalomkezelő rendszerre *(Learning Management System*, LMS). Ilyen például a Moodle, a Listmos, a Canvas, a Talent Learning Management System vagy a MeMOOC-ban használt Open edX. **Megoldást jelenthet saját LMS készítése is, ahogyan az a kodolosuli.hu portál esetében történt.**

A felsorolt LMS-ek rögzítik a felhasználók tevékenységeit, ezek elemzésével a tananyagok hatékonysága növelhető. Saját LMS készítése esetében azonban figyelni kell arra, hogy az elemzéshez szükséges felhasználói interakció rögzítve legyenek (Király, 2016b).

Online környezetben a tanulók teljesítményét több olyan tényező is befolyásolhatja, amely a tanulók egyéni szokásaitól, sajátosságaiból erednek. Ilyenek például a tanulók képessége figyelmük fenntartására, belső motivációjuk a tanulásra (Faragó, 2015; Király, 2016a). Ezen tényezőket online környezetben is tudjuk befolyásolni, hiszen lehetőségünk van az elkötelezettség növelésére, amely meghatározható úgy, mint a hallgató kognitív folyamata, aktív és érzelmi részvétele a tanulási folyamatban (Pellas, 2014; Wolf, 2007; Clark and Mayer, 2011). Az elkötelezettség mindhárom faktora (kognitív, viselkedéses és az érzelmi) növelhető, amennyiben a tananyag szövege nemcsak szakmailag korrekt, hanem jó stílusú, személyes hangvételű (Héjja-Nagy, 2015). A szöveg gondolkodásra késztető hatását érdekes példák, jó metaforák, olykor meghökkentő és provokáló felvételek, kérdések, ellentmondások, meglepő, sok esetben humoros fordulatok beillesztése segítheti elő (Király, 2015; Király, 2016b).

A MeMOOC kurzusokban például már a fejezetek címei is meglepőek, figyelmet felkeltőek, például: „Üdvözlet a mátrixban", „Mindig függünk valamitől", „Válaszút elé érkeztünk", „Legyen saját osztályod!", „Mi a mai menü?".

Az elektronikus környezet hasznossága szignifikáns hatással van a tanulók megelégedettségére, ezért törekedtünk arra, hogy a tartalom a valós életben is alkalmazható legyen (Joo et al., 2013). Ennek megfelelően gyakoriak a mindennapi életből vett gyakorlati feladatok, valamint a játékokkal kapcsolatos feladatok is, mint például egy tolatóradar működtetése, egy bolygóra való leszállás vezérlése vagy a legközelebbi tárgy (defibrillátor) megtalálása. Az eldöntés tételét kell alkalmazni akkor is, ha a feladatot így fogalmazunk meg: „Adott egy pont, valamint egy kör középpontja és sugara, döntsük el, hogy a pont a kör belsejében van-e", de mennyivel „izgalmasabb" a feladatot akkor megoldani, ha az így szól: „Ismerjük egy lézerágyú hatótávolságát, valamint az érkező ellenséges objektum koordinátáit. Döntsük el, hogy érdemes-e tüzelni az ágyúval!" (1. ábra). Ha mindehhez még egy videót is készítünk, ahogyan ez a kodolosuli.hu portál kezdő oldalán is látható (ha a három videó közül éppen ez jelenik meg). Természetesen a szöveg stílusát erősen befolyásolja az is, hogy mely korosztálynak készítjük.



1. ábra: Egy feladathoz készített animáció a gamifikáció részeként

A hatékony e-learning környezet kialakításakor törekedni kell arra, hogy a tanuló érezze: a tanár, az instruktor jelen van, elérhető, azaz a **kognitív jelenlét** érzése valósuljon meg (Joo et al., 2013). A MeMOOC kurzusokban ezt biztosította a Vitafórum, Mentorálás és a Hírek funckiók, a kódolósuliban pedig a Disqus rendszer.

Mindkét környezet **harmadik típusú e-learning környezetnek**, az irányított felfedezés környezetének tekinthető, hiszen jellemző rájuk a **magas pszichológiai és viselkedéses elköteleződés és nagymértékű interaktivitás** (Clark et al., 2011).

3.1. A tartalom

Kijelenthető, hogy a számítógépes programozás tanítása különbözik a többi tantárgy tanításától, hiszen a programozási nyelvek szintaxis-szabályokkal rendelkeznek, amelyekre emlékeznie kell annak, aki programozó akar lenni, ráadásul a programozónak algoritmizáló képességgel is rendelkeznie kell. Ezért a programozás oktatása során kiemelt szerepe van az algoritmizáló képesség fejlesztésének, amelyhez az első, és egyben legfontosabb lépés a programozási tételek megismertetetése, valamint a használatukra épülő feladatok megoldása (Szlávi és Zsakó, 2004). Ennek oka, hogy a gyakorlatban előforduló feladatok túlnyomó része visszavezethető ezekre a tételekre, ráadásul az összetett programozási tételeket is vissza lehet vezetni elemi tételekre (Fekete et al., 2019; Szlávi et al., 2019). Például a másolás, kiválogatás, szétválogatás, metszet, unió a sorozatszámításra vezethetők vissza, amit pedig egy egyszerű összegzésből kaphatunk (Zsakó et al., 2017).

Egy kezdőknek szóló programozási tananyagnak a nyelvtől[1] függetlenül körülbelül ugyanazokat a témaköröket kell tartalmaznia, függetlenül attól, hogy a célunk egy OOP-s tárgy megalapozása (például a MeMOOC kurzus), az érettségi programozás részére történő felkészítés (kódolósuli), vagy akár a Nemzetközi Informatikai Diákolimpiára való felkészítés (Király, 2011). Ennek

[1] Imperatív nyelvek esetén

megfelelően a figyelem felkeltését szolgáló bevezető részek után az egyes fejezetek a kódolósuliban (Balla és Király, 2017):

- A programozás lépései
- Változók használata és a nyelv típusa
- I/O műveletek
- Operátorok
- Elágazások
- Iteráció
- Tömbök
- Szöveges típusok használata
- Fájlok használata
- Alprogramok
- Listák
- Struktúrák, osztályok
- Kivételek kezelése

A fenti fejezetcímek a kódolósuliban találhatók, de a MeMOOC kurzus is körülbelül ezeket a fejezeteket tartalmazza, kiegészítve egy olyan fejezettel („A laboratórium"), amely csak a programozási tételekkel foglalkozik. Mindkét kurzusban találhatók olyan fejezetek, amelyben csak olyan feladatok találhatók (már elméleti felkészítés nélkül), melyek megoldásához a megfelelő programozási tételt, tételeket kell megtalálnia és alkalmaznia a tanulóknak. Mindkét portálon a kurzusok tartalmazzák az egyes fejlesztői környezetek (Visual Stúdió, SharpDevelopment, CodeBlocks, NetBeans) használatához szükséges információkat példákon keresztül.

Az Objektumorientált programozás (MeMOOC) fejezetei az OOP-hez kötődnek:

- Legyen saját osztályod
- Az öröklés és következményei
- Osztályok és a közöttük lévő kapcsolatok
- Generikus osztályok és a Collections
- Visszahívást kérek (Callback)

A kurzus „ékköveit" jelentik a

- GUI készítése
- Mi a mai menü?
- Egy kis grafika

fejezetek a hozzájuk tartozó oktatási célú játékokkal. **Ezek a részek rendkívül látványosak, alkalmasak az ösztönzésre és az elkötelezettség növelésére**.

Ennek megfelelően minden, általunk készített kurzusban a diák megismeri, valamint képes lesz felismerni és használni ezeket a tételeket. A legegyszerűbb feladatoktól haladunk a bonyolultabb problémák megoldásig. Miközben tudjuk, hogy az, hogy a tanulók mennyi ideig tudják figyelmüket fentartani olyan tényezőktől is függ, mint például a motiváció, érzelem, élvezet és a napszak (Bunce, Flens and Neiles 2010), azért hogy fentartsuk a motivációt, a tanulási egységek feldolgozásának időtartama nem haladja meg a 10 percet.

4. Játékosítás (gamification) és a „komoly játékok” (serious games)

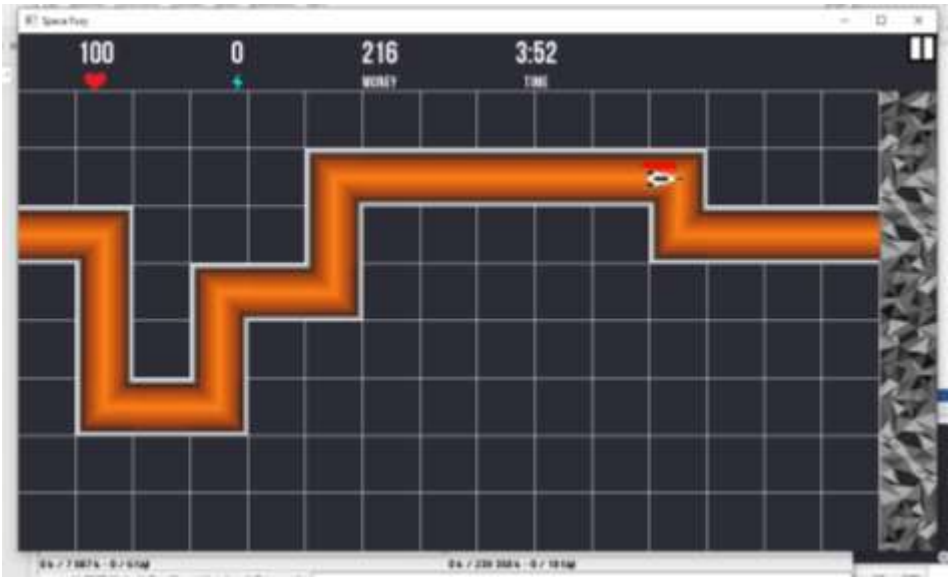
A „Gamifikáció (*gamification*), komoly játék (*serious game*), edutaintment (*education* és az *entertaintment* szavak összevonásából származó, szórakoztatva tanítás) – egymáshoz nagyon közel álló fogalmak, melyek egy folyamat leírására vonatkoznak: hogyan lehet a játék mechanizmusait hagyományosan nem játékos környezetben az eredeti folyamatok sikerének növelése érdekében. A gamification kifejezést Nick Pelling használta először (Pelling, 2011). Azt jelenti, hogy játékelemeket használunk „nem-játék” kontextusban az ember és a számítógép közötti elkötelezettség fokozása (Deterding et al., 2011), valamint a diákok oktatási környezetben való motiválása érdekében (Tsai et al., 2016). A gamifikáció olyan elemeket használ, amelyek támogatják a belső és külső motivációt, például díjakat (jutalmakat) kínálnak (Surendeleg et al., 2014). Ezen kívül lehetőséget kínál a szabályok, az érzelmek és a társadalmi szerepek kipróbálására (Lee et al., 2011).

Mivel egy online kurzusban a tanár személyes ösztönző, motiváló szerepe nem érvényesül, így a **gamifikáció szinte kötelező eleme egy programozást tanító kurzusnak**. A játékok kiválóan teljesítenek, ha az önkéntesség, az interakció és a személyre szabottság mértékét kell növelni. A játékelemek vonzóak, mivel képesek kielégíteni az alapvető kompetenciára és autonómiára vonatkozó pszichológiai igényeket (Fotaris et al., 2015; Li et al., 2013). Online környezetben is fontos a hallgatói kognitív részvétel fokozása (Wolf, 2011), amelyet például játékelemek alkalmazásával lehet megvalósítani. Ráadásul egy gamifikált tananyagban érvényesül a folyamatos értékelés elve, sok esetben támogatja az egyéni tanulási útvonalakat, ugyanakkor nem tartalmaz negatív értékelést.

Ugyanakkor érdemes megemlíteni, hogy a szkeptikusok szerint a tanulási vágy játékvággyal való helyettesítése semmiképpen sem lehet az oktatási szféra célja hosszabb távon. Az is erős ellenérv, hogy a jelvények, szintek és rangok dinamikus rendszere lényegét tekintve (és az újdonság elmúltával) csupán csak egy újabb, kicsit bonyolultabb jegyrendszer.

4.1. Gamifikáció a Kódolósuli és a MeMOOC projektekben

A gamifikáció fent említett előnyös tulajdonságai miatt programozási kurzusainkat játékossá tettük. Szándékosan eltekintettünk a kitűzőktől, jutalmaktól, a pontoktól és az előrehaladási folyamat kijelzésétől. Ez utóbbitól azért, mert a tanulás során mindenki láthatja, hogy hol tart a tanulási folyamatban, melyik fejezet dolgozza fel éppen, valamint azt, hogy mennyi van még hátra. A ranglistákkal az a gond, hogy a tapasztalataink szerint a legjobb N diákot motiválja, a többieket inkább frusztrálja a ranglista. Ugyanakkor azok a diákok, akik teljesítik a kurzus minden egyes elemét és feladatát, letölthetnek egy játékprogramot Android és Windows platformokra (2. ábra). Bizonyos fejezetek teljesítése után a felhasználók információkat és képernyőképeket kapnak a letölthető, jutalmazását jelentő programról. Természetesen a portál biztosít azonnali visszajelzéseket és elismeréseket (Király és Balla, 2016a).



2. ábra: Az egyik letölthető program képernyőképe

Ezek az elemek tekinthetők a gamification 1.0-ás verziójának, vagy más szóval tradicionális gamifikációnak. A következő változatban a diákok már játszhatnak, ráadásul olyan játékokkal, amelyek nemcsak motiválnak, hanem oktatási tartalommal is rendelkeznek. A cél az, hogy a játékélmény oktatási tapasztalattá alakuljon át (Garris et al., 2011). Az informatika bevonulása a mindennapjainkba, sőt, minden „másodpercünkbe" egy új kommunikációs teret hozott létre (Benczúr 2003), ennek megfelelően már a játékosítás új szintjére is jellemzőek a **közösségi média, a közösségek, valamint az animációk** használata is. A kodolosuli.hu már a nyitó oldalon animációkkal próbálja vonzani a programozást tanulni szándékozó diákokat. A portálba integráltuk a Disqus rendszert (3. ábra), amely alkalmas közösségek építésére, elérhetők a közösségi médiák is, egészen pontosan a FaceBook és a Twitter (Király és Balla, 2016b).

3. ábra: A kódolósuliba integrált Diqus rendszer közösség építésére

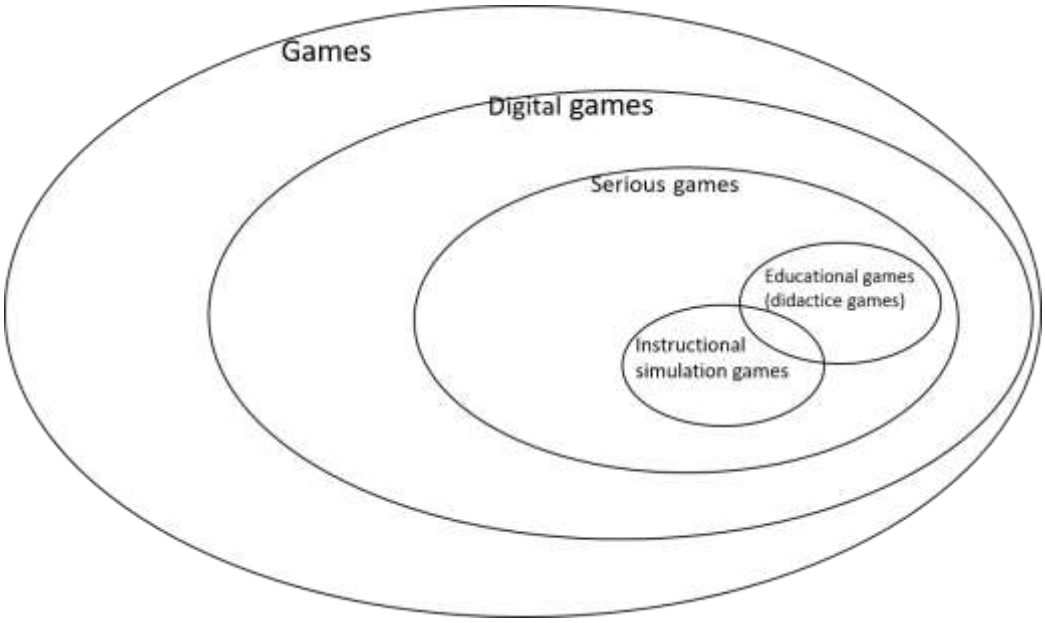
A MeMOOC rendszerben nem volt előírás a gamifikálás, viszont 15-20 olyan játékot kell fejleszteni, amelyek segítik a tananyag megértését.

4.2. Serious games

Számos játéktípus létezik, amely képes lehet növelni egy programozási kurzus hatékonyságát. Digitális játékok azok, amelyek valamilyen informatikai eszközön futnak (PC, táblagép, okostelefon, stb.), a komoly játékok pedig olyan digitális játékok, amelyeket azzal a céllal hoztak létre, hogy szórakoztassa a játékost és elérjen legalább egy további célt, például tanítson vagy segítsen az egészséget megőrizni (Dörner et al., 2016).

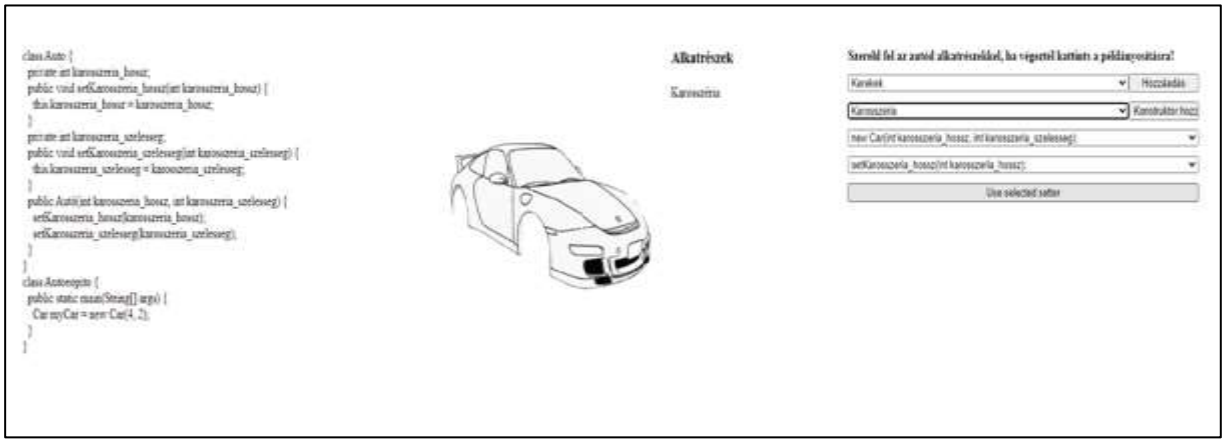
Ennek megfelelően ezek a játékok a programozás tanulásának ösztönzésére is használhatók (Frankovic et al., 2018). Például a Disney Minnie Dizz földjét fedezi fel (Dörner et al., 2016), ahol a kisgyermekek fejleszthetik a problémamegoldó készségeket, ilyen továbbá a CodeCombat, amely egy játék alapú számítástechnikai program, ahol a hallgatók valódi kódot írnak, és látják, hogy karaktereik valós időben reagálnak. Nyilvánvaló, hogy ezen játéktípusok alkalmasak arra is, hogy a programozás tanulását ösztönözzék (Frankovic et al., 2018).

Az általunk fejlesztett játékok tehát a serious games csoportba tartoznak, viszont **kifejezetten didaktikai célból** hoztuk ezeket létre. Az ilyen típusú játékokat hívjuk oktatási (*educational)* vagy didaktikai (*didactic*) játékoknak. Definiálhatjuk ezeket úgy mint interaktív, versengést lehetővé tevő játékok meghatározott tanulási eredményekkel, amelyek lehetővé teszik a diákok számára, hogy szórakozzanak a tudás megszerzése során. Céljuk nem csupán a szórakozás, hanem oktatási elemet is tartalmaznak (Blumberg et al., 2013). A játékok többségére igaz, hogy a felhasználók a megadott utasításokat követve szimulálhatják a számítógépes programozási kódszerkezetek működését, amelyek megkönnyítik a kódok működésének megértését. A definíció alapján (Sauvé, Renaud Kaufman és Marquis (2007)) ezek a játékok nemcsak oktatási játékok, hanem oktatási szimulációs játékok (*Instructional Simulation Games*) is (4. ábra). A tanulók gombok, szövegdobozok és csúszkák segítségével futtathatják vagy leállíthatják a kísérletet, amely a valóság egyszerűsített változata, és a szabályok keretén belül megváltoztathatják a jelenségek paramétereit.



4. ábra: Játékok típusai

A Game-based Learning (GBL, játék alapú tanulás) a játékokkal történő tanulás folyamata és gyakorlata. Célja a tanulás javítása, a tanulás hatékonyságának növelése. Portálunkat nem tekinthetjük sem pusztán játék alapú tanulásnak (GBL), sem csak gamifikált portálnak. A gamifikált tananyagunkat bővítettük „komoly játékokkal", melyek oktatási játékoknak és oktatási szimulációs játékoknak tekinthetők.
A MeMOOC két kurzusában összesen 34 játékot készítettem, mindegyiket a Java nyelv oktatására. A kódolósuliban ebből 19-et használtunk fel, mindegyiknek elkészítve a C++ és a C# nyelvi változatát is. Valamennyi a HTML5 és a JavaScript nyelvek felhasználásával készült. A játékok fajtáit és szerepét több tudományos közleményben is megjelentettük (Király és Balla, 2020a; Király és Balla 2016a; Király, Balla és Kusper 2016).
Ezek nagy része oktatási szimulációs játék, de implementáltunk olyan játékokat is, amelyek a memorizálást segítik (memória kártyák, tábla játékok, akasztófa), és olyanokat is, amelyek azt „mérik", hogy a tanuló mennyire értette meg a tananyagot. A következő ábra egy olyan játék pillanatnyi helyzetét mutatja, melynek során a diák egy autó osztályt épít fel, listaelemek kiválasztásával, közben látja az épülő kódot és az autót is (5. ábra).



5. ábra: Autó osztály építése az elemek kiválasztásával

A játékok a következő tulajdonságokkal rendelkeznek:

- A tanulókat bevonják a rendszerbe, interakciókra (6. ábra) késztetik őket (Khaleel et al., 2015).
- Kihívás elé állítják a diákokat, ugyanakkor igazodnak az addig elsajátított tudáshoz, így nem túl könnyűek és nem is túl nehezek.
- Egyértelmű utasításokat tartalmaznak a szabályokról, a játék menetéről.
- Gyors visszajelzést nyújtanak a folyamatos kommunikáció fenntartása érdekében aktuális státuszukról, saját állapotukról és viselkedésükről (lásd 5. és a 6. ábrát).
- Felkeltik a kíváncsiságot, újszerűek, így
- ezek a játékok nemcsak elősegítik a motivációt, hanem gyorsabb megértést is biztosítanak (Domínguez et al., 2013).

6. ábra: Egy eljárások használatát bemutató alkalmazás. A sárga kör mutatja az éppen végrehajtott utasítást, a szám a változó értékét. Cél a feladatban kértek alapján a megfelelő paraméterek beállítása.

Ezek pedig éppen a játékos tanulási környezetben való részvétel legfontosabb tulajdonságai.

Vihavainen és társai szerint (Vihavainen et al., 2014), az oktatási játékok bevezetése a programozási kurzusokba átlagosan 10,8% -kal növeli az átadási arányt. Mi is kíváncsiak voltunk, hogy a kifejlesztett játékok hozzáadása portálunkhoz javíthatja-e annak hatékonyságát.

1. tézis: Egy már gamifikált programozási tananyag hatékonysága is javítható oktatási célú játékokkal.

4.3. Kísérlet a játékok hatásosságáról

15 darab játékot elhelyeztünk a tananyag különböző részeibe, mindegyiket úgy, hogy utána biztosan legyen két olyan feladat, amely az adott játékhoz kapcsolódik. Az LMS-ben tárolt adatokat

felhasználva megvizsgáltuk 200 véletlenszerűen kiválasztott felhasználó esetében, hogy a játékok után lévő két feladatot meg tudták-e oldani első vagy második kísérletre. Ők képezték a **kísérleti csoportot**. Az eredményeiket összehasonlítottuk 200 olyan felhasználóval, akiknek nem volt lehetőségük a játékokkal játszani, ők voltak a **kontroll csoport** tagjai. A két csoport között életkorban és a nemek összetételében alig volt eltérés.

A felmérés eredménye alapján mind a 15 játék esetében a kísérleti csoport tagjai átlagosan jobb eredményeket értek el a feladatok megoldásában, mint a kontroll csoport tagjai a páros t-próba alapján (5%-os szignifikancia szint mellett). **Az első tézist alátámasztó felmérés eredményét tudományos folyóiratban publikáltuk** (Király és Balla, 2020a). A kódolósuli használói szívesen játszanak a játékokkal, többen pihentetőnek tartják őket, sokan a játék után értik meg teljesen az előtte lévő lecke anyagát.

5. Programkiértékelő rendszerek (Grading systems)

Az egyszerűbb programokat sem könnyű ma már kézi értékeléssel ellenőrizni, a programszöveg vizsgálatával. A kézi értékelés másik esete a tesztelés, melynek során az előre elkésztett tesztadatokra futtatjuk le a programokat. Egy programozási versenyen ez a fajta értékelés rendkívül sokáig tart, elektronikus környezetben pedig kivitelezhetetlen, hiszen a számítógépes programozási kurzusokban interaktív, dinamikus, online kódolási gyakorlatokat és összetettebb programozási feladatokat kell biztosítani akár több száz tanuló számára, a beküldés ideje pedig a nap bármely időszakára eshet. Ugyanakkor biztosítani kell azt, hogy a diákok azonnali visszajelzést kapjanak a feltöltött kódok helyességéről.

5.1. Elkülönült, URL-el azonosítható értékelő rendszerek

Az értékelő rendszerek egyik nagy csoportját képezik az önállóan, elkülönült rendszerként működők, melyeket elsősorban versenyek lebonyolítására használnak, illetve feladatbankkal együtt gyakorlásra, önálló haladásra használhatók. Ilyen a Debreceni Egyetemen készült **progcont.hu** (Aszalós, Kósa és Pánovics, 2017), valamint az ELTE rendszere, a **biro.inf.elte.hu**. Ez utóbbit használják a Nemes Tihamér és az OKTV versenyeken is, a **mester.inf.elte.hu** változata pedig a közoktatásban, az előbb említett versenyekre való felkészítést segíti.

2. tézis: Az automatikus programkiértékelők nemcsak a programok tesztelésére, ellenőrzésére használhatók, hanem a tanulási folyamatot is felgyorsíthatják, ha a tesztesetek célját, a tesztadatokat és a megoldási útmutatót is elérhetővé teszik.

A kódok gyors kiértékelésére a megoldást az automatikus, online értékelés jelenti, melynek menete a következő (Horváth Győző, Horváth Gyula és Zsakó László, 2017):

- a feltöltött programot az értékelő rendszer elmenti, a beadás tényét rögzíti;
- az értékelő rendszer lefordíttatja a beadott programot, sikertelen fordítás esetén a rendszer átirányítja a fordító üzenetét a beküldőnek;
- a rendszer lefuttatja a kódot az előre elkészített tesztesetekre, védett környezetben (sandbox), korlátozva a program számára az erőforrásokat (időlimit, merevlemezre írás, stb.);
- a korlátok megsértése vagy futási hiba esetén az értékelés véget ér, a beküldő erről értesítést kap;
- az ellenőrző program megvizsgálja a beküldött program kimeneteit az egyes tesztesetekre;

- az előre rögzített pontozási rendszer alapján a rendszer pontokat ad a programra, melyről értesíti a beküldőt.

Egy ilyen értékelő rendszer működéséhez a bemeneti és a kimeneti adatok szerkezetét specifikálni kell. Azt a készítők döntik el, hogy csak a tökéletes és hatékony megoldásra adnak pontot (ACM stílus), a tökéletes, de nem a leghatékonyabb megoldásra is adnak pontot (IOI stílus), a részben helyes megoldásra is adnak pontot (Nemzeti versenyek stílusa, például USACO, COCI, UVa) (Erdősné és Zsakó, 2018).

Olyan feladatok esetében, ahol minden tesztesethez csak egyetlen megoldás létezik, elég megvizsgálni, hogy a beküldött program pontosan az elvárt kimenetet adja-e. Nehézséget okoz, ha egy adott tesztesethez több kimenet is tartozhat. Ebben az esetben, az ilyen feladatokhoz el kell készíteni azt a programot, amely a beküldött program kimenetéről eldönti, hogy az helyes-e.

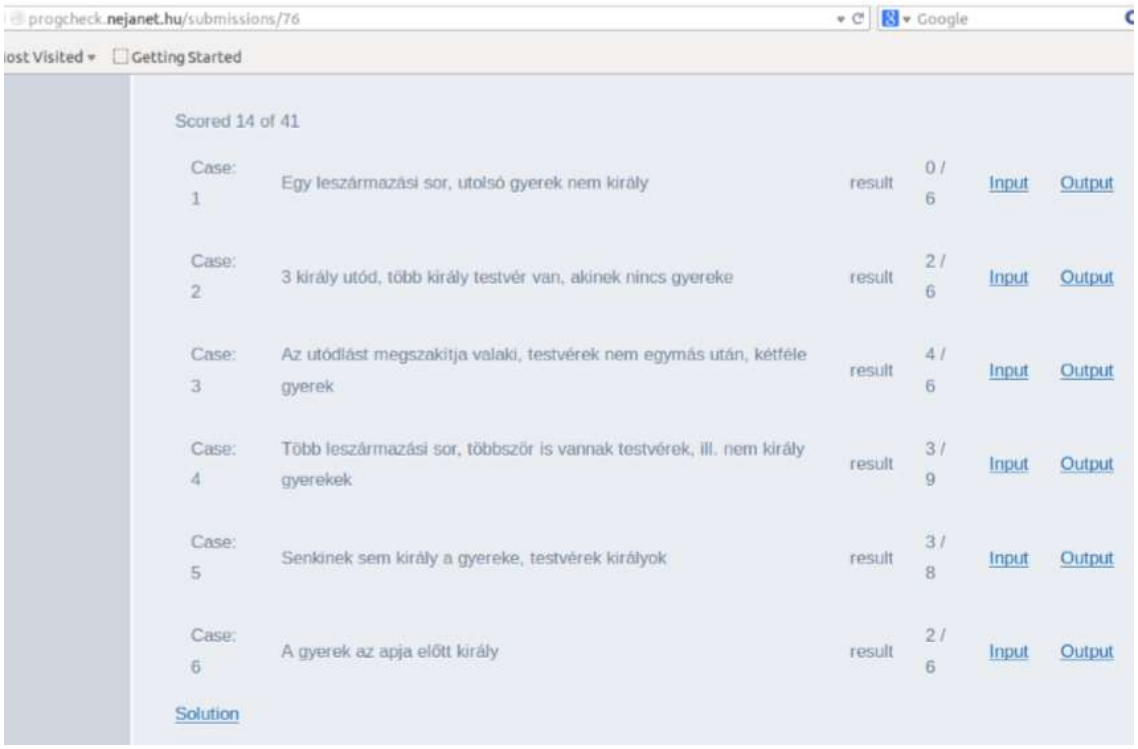
Ezek a rendszerek a weben keresztül elérhetők, objektív értékelést biztosítanak, használatukkal időt lehet megtakarítani, valamint utólagos kiértékelést, statisztikák készítését teszik lehetővé.

Ebbe a csoportba tartozik az általunk készített **progcheck.nejanet.hu** is, melyet 2014-ben készítettünk, 2015-ben pedig az eredményeinket publikáltunk. Ekkor már a **biro.inf.elte.hu**-t használhattuk a Nemes Tihamér versenyeken, a **mester.inf.elte.hu** még készülőben volt. A rendszerrel a célunk az volt, hogy a programozói versenyekre és a „bíró" használatára való felkészülést segítsük, a diákoknak ne kelljen várniuk a tanári válaszra, valamint akár saját feladataink megoldására készített és beküldött programokat is ellenőrizni tudjuk. Az ellenőrzés feladatát a rendszer leveszi a tanár válláról, miközben a forráskódokat bármikor megtekintheti, valamint az is látható, ki és mikor töltött fel megoldást, hány alkalommal.

Az akkor ismert rendszerekhez képest az általunk készített az alábbi, tanulást támogató tulajdonsággal rendelkezik:

- Nemcsak az egyes tesztesetekre adott pontokat írja ki, hanem azt is, hogy az egyes tesztesetek milyen célból készültek, mit akartunk tesztelni (7. ábra)
- Ha valamelyik tesztesetre a diák nem kap pontot, a teszt fájlt a rendszer letölthetővé teszi (versenyek esetében ez a funkció nem volt aktív)
- Nem maximális pontszám esetén a rendszer elérhetővé tesz egy pszeudó kódban megírt, magyarázatokkal ellátott megoldást

Az értékelőrendszer fentiekkel történő kiegészítése az én ötletem volt, a megvalósítás is az én terveim szerint valósult meg.



7. ábra: Egy beküldött program kiértékelésének eredmények a progcheck.nejanet.hu-n

A rendszerbe nem töltöttünk fel feladatokat (bár megoldható lett volna), melynek oka az, hogy a versenyzők tanulási sebessége eltérő, máshol tartanak a felkészülésben, valamint nem tartottuk szerencsének, hogy olyan feladatokat próbáljanak megoldani, melyek megoldásához szükséges ismeretek még nem állnak rendelkezésükre. Így minden versenyzőm személyre szabottan kapott feladatokat, a haladási tempójának megfelelően.

A rendszert csak saját diákjaim felkészítésre használtam, amely évenként 4-5 új diákot jelentett. Egyöntetű volt a véleményük azzal kapcsolatban, hogy óriási könnyebbséget jelentett a tesztesetek letölthetősége, és mivel hiba esetén azt is megkapták, hogy azzal a tesztesettel éppen mit teszteltünk, így a tesztelési képességeik is gyorsabban fejlődtek. A rendszert használók közül kiemelkedik Sály Márió 17. helye az OKTV-n 2017-ben, valamint Zsemberi Dániel 36. helye 2018-ban.

A letölthető tesztesetek, megoldási javaslatok felgyorsították a tanulási folyamatot, testre szabták a diákok felkészülését a versenyekre. **Ez alátámasztotta a 2. tézist (Király és Székely, 2015).**

A rendszert az egyetemi oktatásban is használtam programozási kurzusokon beadandó programok értékelése során. Itt megmutatkozott a rendszer azon hátránya, hogy a kódminőséget nem ellenőrzi, valamint a kód eredetiségét is nehéz garantálni (Horváth Győző, Horváth Gyula és Zsakó László, 2017). A diákok nehezen szokták meg a szigorú specifikációs követelményeknek való megfelelést.

A középiskolások számára meghirdetett programozási versenyeken is használtam a rendszert. Akik már dolgoztak hasonló rendszerrel, ők könnyen alkalmazkodtak ehhez az értékelési formához, a többiek sokszor „elvárták”, hogy részletezzük a versenyen, mit jelent pontosan a „Hibás kimenet!”. Verseny módban azonban a rendszer nem teszi letölthetővé a tesztadatokat.

5.2. LMS-be integrált programkiértékelők

Az online értékelők másik csoportját alkotják azok, melyeket egy online tanulási környezetben integráltak, egy LMS-hez kapcsolódnak. Egy ilyen rendszerben akár több ezer diák is tanulhat egyszerre. Automatikus kiértékelő nélkül a tanároknak, valamint a tanár asszisztenseknek kell a

programokat futtatniuk, és manuálisan kiértékelniük egy ellenőrzési lista alapján a beküldött kódokat. Ugyanakkor a kezdő programozónak azonnali visszacsatolást kell biztosítania a rendszernek a program minőségéről, akár a sokadig feltöltés esetén is. Az is előfordulhat, hogy a tanuló egyébként helyes megoldását nem fogadják el az ellenőrzést végzők, és az is, hogy a diák gépén helyesen működő változat a tanár gépén nem fut le megfelelően. Ennek megfelelően, programozást oktató portál nem képzelhető el automatikus programkiértékelő rendszer nélkül (Hundley and Britt, 2009).

A programkódok kiértékelésének négy aspektusa van (Király, Nehéz és Hornyák, 2017):

1) szintaktikai aspektus,
2) kódolási konvenciók aspektusa,
3) strukturális aspektus,
4) működési logika aspektusa.

3. tézis: Egy online programozási kurzusban használt programkiértékelő rendszer megvalósítható úgy is, hogy négy aspektus szerint is képes legyen a beküldött kódok ellenőrzésére.

Kezdő programozók számára a helyes kódolási stílus megtanulása alapkövetelmény. A programozási kurzusokhoz kapcsolt kiértékelőknek garantálniuk kell, hogy a diák megtanulja a kódolási konvenciókat: megfelelő változó nevek, megjegyzések elhelyezése a kódban, stb. Ennek megfelelően egy ilyen rendszernek képesnek kell lennie elemeznie a kódot sorról sorra. Az olyan programozási kurzusok esetében, amelyek a nyelv grafikus osztályainak használatát kérik a kódolás során, a kiértékelőknek ezek használatát is tudniuk kell ellenőrizni. Ez volt az oka annak, **hogy mind a Kódolósuli mind a MeMOOC projektben különböző kiértékelési módszereket használtunk.**

A kódolósuliban kezdetben a feladatok specifikációja annyira szigorú, hogy a feladat megoldásának egy lehetséges módja van. Ezért megoldható a soronként ellenőrzés, melynek során a tanuló kódját össze tudjuk hasonlítani az elvárt kóddal. Ha például a „switch" utasítás akarjuk gyakoroltatni a diákkal, akkor a feladat előírja, hogy csak ennek használatával fogadjuk el a kódot, vagy egy „for" ciklust gyakoroltató feladat esetében sem engedjük meg például „while" ciklus írását.

Amikor a tanulók már elsajátították a megfelelő programozási stílust, és megfelelő a tudásszintjük, akkor teljes kódokat, futtatható programokat kell írniuk. Ezek tesztelése, ellenőrzése hasonló módon történtik, mint a progcheck.nejanet.hu esetében. A diák feltölti a kódot, a kiértékelő védett környezetben lefordítja, hiba esetén a fordítási hibát kapja vissza a diák, egyébként megtörténik a futtatás. Itt minden programnak 5 másodperc alatt kell lefutnia, egyébként az értékelő leállítja a beküldött program futását.

A MeMOOC projektben a kiértékelő rendszerünk az edX egy külső alrendszere. Amikor egy programozási kurzushoz tervezünk kiértékelőt, az alábbiakat kell figyelembe venni (Király, Nehéz és Hornyák, 2017):

- A kiértékelőnek le kell futtatnia a kódot. Többnyire nem elegendő, ha a kapott kódot mint szöveget összehasonlítja az elvárt kódszöveggel.
- A kiértékelőnek a diákok kódját egymástól elkülönítetten kell futtatnia.
- Egy kártékony vagy egy rosszul megírt kód nem befolyásolhatja a MOOC rendszer működését, így ha Alice egy végtelen ciklust tartalmazó kódot tölt fel, a MOOC

rendszernek továbbra is képesnek kell lennie a válaszadásra. Ha pedig Bob kódja a merevlemezt akarja törölni, akkor ez a kód nem futhat le a MOOC rendszerben.

- A hatékonyság nagyon fontos tényező. Felhasználók ezrei küldhetik be a kódot egyidejűleg. A rendszernek ezért **sorba kell állítania a kéréseket**, és a lehető leggyorsabban kell válaszolnia.
- A kiértékelőnek támogatnia kell a virtuális környezetet. Ha a MOOC rendszert egy UNIX alapú rendszeren akarjuk futtatni, de Windows programozást oktatunk, a külső kiértékelőnek lehetőleg egy virtuális környezetben kell futnia.
- A biztonság kiemelkedően fontos. Egy MOOC rendszerben a diákok krediteket kapnak a diplomához vezető úton, így nem lehetnek képesek megváltoztatni a beküldött kódjukra kapott értékelést.
- A kiértékelőtől kapott válasznak egyértelműen jeleznie kell, hogy a kód elfogadható-e. Hiba esetén támogatnia kell a kód javítását.
- A kiértékelőnek mindig biztosítania kell a névtelenséget, ne legyen tudható, hogy az értékelt kód Alice vagy Bob kódja.

5.3. Programok kiértékelése a MeMOOC rendszerben

Az általunk készített kiértékelő rendszer az edX-hez kapcsolt külső rendszer, amely szolgáltatásként fut, az edX platformtól függetlenül is telepíthető. Az edX egy XQueue nevű interfészen keresztül kommunikál a kiértékelővel. Elküldi a tanuló inputját a kiértékelőnek, majd aszinkron módon vár a kiértékelő eredményére és visszajuttatja azt az LMS-be. A beküldések egy üzenetsorban tárolódnak mindaddig, amíg a kiértékelő el nem kéri azt feldolgozás céljából. A külső kiértékelő rendszeresen lekérdezi az XQueue-t. Amikor a kiértékelés megtörtént, a választ visszaküldi az XQueue számára egy RESTful interfészen keresztül. Az XQueue ezután válaszol az edX LMS-nek. Így működik a kiértékelő rendszerünk, amely tehát aszinkron módon veszi ki (*pull*) az üzeneteket a sorból és teszi vissza a válaszokat (*push*) (8. ábra).

8. ábra: A külső kiértékelő működése

A feltöltött kódok nem egy mappaszerkezetben találhatók mint a progcheck esetében, hanem adatbázisban. Az edX alapértelmezésben támogatja nagy fájlok feltöltését egy Amazon S3 bucket-ben és az URL-jét tárolja ezeknek a lokális adatbázisban. Több ezer beküldött fájl esetén ez a megközelítés nagyon hasznos. A hátránya az, hogy az Amazon S3 kereskedelmi szolgáltatás, ami növeli a fenntartási költségeket, és külső függést okoz a szolgáltatásban. A MeMOOC-ban egy lokális MySQL adatbázist használunk a beküldött fájlok tárolására. A mi rendszerünk tartalmaz egy üzenetsort (RabbitMQ), amely a következőképpen működik (Hornyák és Király, 2015):

(1) Kap egy üzenetet az XQueue sortól. (2) A RabbitMQ kiolvassa az adatot az adatbázisból, majd (3) serializálja azt. Ezután (4) a MeMOOC rendszer küld egy HTTP üzenetet a külső kiértékelőnek és (5) várakozik a válaszra. Közben a küldő kiértékelő (6) inicializálja a virtuális környezetet azért, hogy ott (7) lefordíthasson (vagy értelmezhessen) és (8) futtathasson egy unit tesztet, amely ellenőrzi a beküldött kódot. A program outputja egy fájlban lesz tárolva, melynek neve véletlenszerűen generált karakterekből áll.

A következő, 9. ábra mutatja a rendszerünk működését.

9. ábra: A beküldött program kiértékelésének menete

A megvalósított kiértékelő rendszerünk nemcsak programokat, hanem metódusokat is tud tesztelni unit tesztek futtatásával. Ehhez a JUnit `assertEquals` metódusát használhatjuk, a teszt eredményét JSON string-ben adja vissza, amely tartalmazza, hogy a beküldött kód helyes volt-e, a pontot és egy tetszőleges üzenetet.

A rendszerben a programozási feladatokat egy belső XML-ben lehet megadni. Ráadásul ebben a fájlban lehetőség van Python kódok írására is, ami lehetővé teszi **dinamikusan változó feladatok** írását.

A következő, 10. ábrán látható egy feladat olyan formában, ahogyan a tanuló látja, alatta pedig az XML változat a Python kódokkal:

10. ábra: Felül a diák által látott feladat, alul a mögötte lévő dinamikusan változtatható feladat kódja

Ennek megfelelően a MeMOOC rendszer képes

- ugyanazt a kódolási feladatot adni minden felhasználónak,
- ugyanazt a kódolási feladatot adni véletlenszerűen generált értékelési paraméterekkel,
- véletlenszerűen adni egy kódolási feladatot.

A Python kód feldolgozása után természetesen a változók helyére konstans értékek kerülnek.

A MeMOOC rendszerben használt kiértékelő program a kiértékelő rendszerek mind a négy aspektusa szerint képes volt a beküldött kódok ellenőrzésére, ami nemzetközi viszonylatban is ritkaságnak számít.

A **szintaktika aspektus** azt jelenti, hogy a diák kódját le kell futtatni, illetve átadni egy értelmezőnek (*interpreter*), ennek megfelelően a kódnak a használt nyelv szintaktikai szabályait kell követnie.

A **kódolási konvenció** szempontja annak ellenőrzésére szolgál, hogy a hallgatók betartották-e az általunk előírt kódolási útmutatást. A MeMOOC projekt esetében ennek egy haladó módszerét használtuk a kódolási stílus ellenőrzésére. A **Checkstyle** egy nagyon jól konfigurálható kódstílus-elemző eszköz, amely segíti a programozókat, hogy olyan Java kódot írnak, amely egy előre definiált (Sun vagy Google) standard kódoláshoz kapcsolódik. Modulok olyan halmazát határozza meg, amelyek képesek megvizsgálni a gyakori stílushibákat, például

- osztályok, metódusok és attribútumok helytelen név konvencióit;
- kötelező fejlécek jelenlétét;
- nem megfelelő importálást;
- hatókör-módosítókat;
- utasításblokkokat;
- hosszú sorhosszakat.

A következő, 11. ábrán látható kód a helytelen kapcsos zárójel használatot mutatja be. A Google Java Style Guide (2016) definiálja a megfelelő zárójel használatot, a kód megsért ezt. A CheckStyle-t futtatva az észreveszi a hibát és jelzi azt.

```
public class Person
 {  // Violates 4.1.2 point of [25]: No line break before the opening brace.
   private Date birthDay;
   public Date getBirthDay()
   { // the same formatting problem
     return birthDay;
   }
   public void setBirthDay(Date birthDay) {
     this.birthDay = birthDay;
   }
 }
```

```
Starting audit...
[WARN] Person.java:2:1: '{' at column 1 should be on the previous line. [LeftCurly]
[WARN] Person.java:5:3: '{' at column 3 should be on the previous line. [LeftCurly]
Audit done.Audit done.
```

11. ábra: Felül a beküldött kód, alul az értékelés eredménye

A **strukturális aspektust** akkor használjuk, amikor a tanulók már képesek komplex adatokat, struktúrákat létrehozni, a kiértékelőnek pedig ezeket kell tudnia ellenőrizni. Például a feladat szerint a diáknak egy `Person` osztályt kell deklarálnia, amelyben az egyik mező neve szöveg típusú, neve `name`, valamint implementálnia kell egy `getName` metódust, amely visszaadja a személy nevét. Az osztálynak van még egy `integer` típusú `age` nevű mezője is. A kiértékelőnek ellenőriznie kell, hogy

- a `Person` osztály deklarálva lett-e,
- deklarálva lett-e az `age` mező,
- létezik-e `getName` metódus.

Az átalunk készített kiértékelő a JUnit-ot fogja használni az ellenőrzéskor. Megpróbálja betölteni a `Person` osztályt dinamikusan. Ha az nem létezik, akkor egy a `ClassNotFoundException`-t dob, amely el lesz küldve a beküldő számára. Majd ellenőrzésre került a `getName` metódus is. Ha az nem létezik, akkor egy `assertion failure` keletkezik, azaz a unit teszt sikertelen. Ha a metódusnak paramétereket kell átadni, akkor a `person.getMethod` hívásakor azokat is át kell adni. A következő, 12. ábra mutatja azt is, hogyan ellenőrizzük az `age` mezőt. A kód megkapja a `Person` osztály mezőit, a kapcsolódó unit teszt hibát ad, ha az `age` nem található.

Az **működési logika aspektusa** esetén a rendszerünk a standard JUnit teszteket használta a program a működési logika ellenőrzésére.

```
@Test
 public void testRetirementAge() {
  Person p = new Person();
  p.setAge(60);
  assertEquals(p.getAgesUntilRetirement(), 5);

  p.setAge(70);
  assertEquals(p.getAgesUntilRetirement(), -1);
 }
```

12. ábra: Példa unit tesztre

A feladat azt kéri, hogy hozzunk létre egy függvényt, amely meghatározza, hogy egy személynek hány évet kell még dolgoznia nyugdíjba vonulásáig (65 éves koráig). Annak a kódnak a tesztelésére, amely megvalósítja a fenti logikát, a 12. ábrán látható unit tesztet lehet futtatni (szándékosan egyszerűsítve a kiírást egyetlen számra).

Először a MeMOOC, majd később a Kódolósuli projektben került sor olyan programokat kiértékelő rendszer létrehozására, amely nemcsak a feltöltött programokat tudta tesztelni, hanem képes volt a kiértékelő rendszerek mind a négy aspektusa szerint ellenőrizni a kódot, alátámasztva a 3. tézist.

4. tézis. Olyan programkiértékelő rendszer is készíthető, amely GUI osztályokat tartalmazó program ellenőrzésére is alkalmas.

Habár a parancssori interfész (CLI) alkalmazások többnyire elegendőek a programozási készségek fejlesztésére, a grafikus felhasználói interfészek (GUI) az alkalmazásokban növelhetik a diákok motivációját és a felhasználói élményt (Staubitz et al. 2015). Az általunk kifejlesztett rendszerben a GUI-t tartalmazó Java programok értékelésére a *Mockito* framework-öt használtunk[2]. Tegyük fel, hogy egy programnak három checkbox elemet kell a felülethez hozzáadnia, valamint implementálnia három eseménykezelőt! A *Mockito* rendszerben a tesztet úgy kell implementálni, hogy megvizsgáljuk a három checkbox objektum példányosítva van-e, és az `addItemListener` metódus meg lett-e hívva háromszor. A külső GUI létrehozásához Mockito és *Powermock-module-JUnit* installálására volt szükség a szerver oldalon. A tesztelő programban a következő módon kell ellenőrizni az osztály létezését: az értékelő programunk megpróbál hivatkozni egy objektumra, amelyet a diák programjában létre kellett hozni (checkbox), és ha az nem létezik egy *try-catch* blokkban, az értékelő elkapja a hibát. Az elkapott kivétel alapján a külső értékelő üzenetet küld a diáknak arról, hogy mi hiányzik a programjából. Az üzenet javaslatokat is tartalmazhat a hiba megoldásához.

A negyedik tézist megvalósító MeMOOC kurzusra több, mint 300 diák regisztrált, és dolgozta fel az egyes leckéket.

Az ilyen módon elkészített értékelő alkalmas GUI programok ellenőrzésére is, alátámasztva a negyedik tézist.

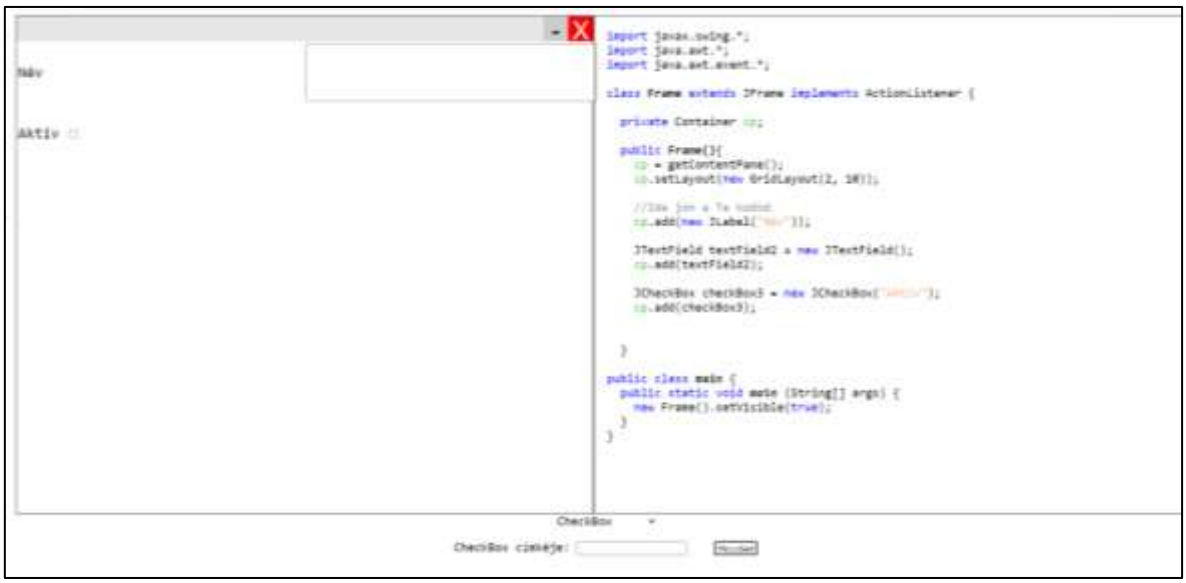
6. Adaptivitás online környezetben

..."adaptivitáson a pedagógiában olyan alkalmazkodva fejlesztést/fejlődést értünk, amely egyaránt tekintettel van az egységesség és a differenciáltság szempontjaira, akár az egyének, akár a csoportok, akár az intézmények igényei felől közelítünk." (Nádas, 2010) Azaz olyan pedagógiai

[2] Mockito testing framework, 2016, [online] Elérhetőség: https://mockito.org/

tevékenységrendszerről van szó, amely arra törekszik, hogy egyszerre vegye figyelembe az oktatási rendszer valamennyi szereplőjének az igényeit. Úgy értelmezhetjük mint a változás-reflexió-tanulás/innováció fogalmak dinamikus kölcsönhatása (Rapos, Gaskó, Kálmán és Mészáros, 2011).

Az elméletben régen létező és szorgalmazott adaptív pedagógia a digitális eszközökkel a valóságban is realizálódhat. Az olyan rendszerektől, mint a memooc.hu vagy a kodolosuli.hu elvárható, hogy adaptívak legyenek. Jelen pillanatban azonban ezek a rendszerek mindenkinek egységes módon biztosítják a tananyag-közvetítési, haladási, értékelési, és a reflexiós lehetőségeket. Ahogyan az korábban bemutatásra került, a MeMOOC programozási kurzusaiban van lehetőség az adaptivitásra a kódolási feladatok megoldásában. A kódolósuli esetén is megmutattuk, hogyan tervezzük az adaptivitás megvalósítását a feladatok megoldása során (Balla és Király, 2020a, Balla és Király, 2020b). A megoldás részét képezi az oktatási játékok könnyített változatának felhasználása is. Ezen változatok esetén a diáknak kevesebb és egyszerűbb feladatai vannak a játék során, valamint sok esetben nem kell kódot sem gépelnie, elegendő a megfelelő kódsor menüből történtő kiválasztása, ahogyan az a következő, 13. ábrán is látható.



13 . ábra: Frame építése komponensek hozzáadásával. A kiválasztott komponens megjelenik a Frame-en, a megfelelő utasítások pedig a kódhoz lesznek hozzáadva.

A teljesen adaptivitás elérése rendkívül munkaigényes feladat, a tananyag, valamint az LMS nagymértékű átalakítását is igényli. Megvalósítása egy Moodle rendszer használata esetén is rengeteg munkát igényel a fejlesztőktől (Király, 2016c). A munka első lépése a rendszereket használó tanulók kezdeti paramétereinek (életkor, előismeret, tanulási célok, stb.) azonosítása. További mérésekre van szükség a felhasználók digitális környezettel folytatott interakciókról: mennyi időt töltöttek egy lecke elolvasásával, egy feladat megoldásával, hányszor javították a megoldást, milyen sorrendet követtek, stb. A kontextuális adatgyűjtési lehetőséggel kapcsolatban vannak és növekvő jelentőséget tulajdonítanak az adatbányászati technológiáknak (Hülber, 2015).

Kutatási tevékenységünket folytatva a kódolósuli további fejlesztésében az adaptivitás megteremtése lesz a következő feladatunk.

7. Összegzés

Ebben a dolgozatban a PhD megszerzését követő időszak kutatásainak egyik szeletét mutattam be. Azokkal a kutatásokkal foglalkoztam ezen keretek között, amelyek kapcsolódnak a programozás elektronikus oktatásához szükséges e-learning környezet és a megfelelő tananyag kialakításához, valamint a feladatok megoldásait jelentő kódok kiértékeléséhez.

Kutatásaink alapján azt mondhatom, hogy szükség van olyan, bárki által a weben elérhető, magyar nyelvű, önállóan is teljesíthető programozási kurzusokra, amelyek lehetővé teszik a programozás elsajátítását. Mivel a programozás oktatás megköveteli, hogy a tanulók gyakorlati készségeket is szerezzenek, ezért szükség van a feladatok megoldását jelentő kódok ellenőrzésére.

Az általunk készített online programozási kurzusok olyan elektronikus környezetben teljesíthetők, amelyek motiválók, jellemző rájuk a viselkedéses elköteleződés és a nagymértékű interaktivitás. A feladatok gondolkodásra késztető, játékos, a mindennapi életből vett gyakorlati feladatokat, valamint játékos feladatokat tartalmaznak.

Minden kurzushoz a feltöltött kódok ellenőrzésére alkalmas kiértékelő rendszereket készítettünk. Ezek nemcsak a kódok szintakszisának és működési logikájának ellenőrzésére használhatók, hanem a megoldáshoz szükséges struktúrák használatát és a kódolási konvenciók betartását is tudják ellenőrizni. Sőt, akár Java GUI osztályok használatának ellenőrzése is megoldott.

A kurzusokhoz olyan oktatási célú játékokat fejlesztettem ki, amelyek nemcsak motiválnak, hanem oktatási tartalommal is rendelkeznek, így a játékélmény oktatási tapasztalattá alakulhat át, ugyanakkor növelik az elkötelezettség mindhárom faktorát.

Az Objektumorientált programozás MeMOOC kurzusra 2015 szeptembere óta **337**-en regisztráltak. A többségük még diploma nélküli volt, de 156-an már BSc, MSc vagy Phd fokozattal rendelkeztek. A Programozási nyelvek alapjai kurzusra **955**-en regisztráltak a 2015 szeptemberi nyitás óta. 403-an voltak diplomások és 28 általános iskolás diák is regisztrált a kurzusra. A kodolosuli.hu-nak több, mint **700** regisztrált felhasználója van. Többen közülük ugyan nem végzik el a teljes kurzust, csak bizonyos fejezeteket dolgoznak fel egy kiválasztott programozási nyelvből, ugyanakkor többen két nyelv esetében is feldolgozták a leckék nagy részét.

Szándékunkban áll az ez irányú kutatásokat folytatni. A kódolósuliban szeretnék adaptív környezetet megvalósítani, először a feladatmegoldásban, majd a tananyag megfelelő átalakításával, tovább növelve ezáltal a tanulási folyamat hatékonyságát.

8. Irodalomjegyzék

8.1. A kitekintés szakirodalmi hivatkozásai

Arefin, A. S. (2015): Pedagogy of Computer Programming: *An Interactive and Collaborative Learning Approach*. Macquarie University Postgraduate Certificate of Higher Education EDCN 871 Final Project

Aszalós, L., Kósa, M. and Pánovics, J. (2017): *Renewal of ProgCont and Talent Management in High Schools*. InfoDidact 2017. https://people.inf.elte.hu/szlavi/InfoDidact17/Abstracts/ALKMPJ_abs_eng.pdf

Benczúr A. (2003): *A kommunikáció fejlődése és az információs forradalom*. Természet világa, Külön szám, pp. 74–79.

Bunce, D.M., Flens, E.A. & Neiles, K. Y. (2010): *How long can students pay attention in class? A study of student attention decline using clickers*. Journal of Chemical Education, vol. 87, no. 12, pp. 1438–1443. doi: 10.1021/ed100409p

Blumberg, F., Debby E. Almonte, Jared, S. Anthony, and Hashimoto, N. (2013): *Serious Games: What Are They? What Do They Do? Why Should We Play Them*. In: Dill, K.E. (Hrsg.): The Oxford Handbook of Media Psychology, Oxford et al. 2013. S. 334-351. doi: 10.1093/oxfordhb/9780195398809.013.0019

Clark, R.C., Mayer, R.E. (2011): *E-learning and the science of instruction*. Pfeiffer, San Francisco, 2011

Deterding, S., Khaled, R., Nacke, L., Dixon, D. (2011): *Gamification: Toward a definition*. In: Proceedings of Chi 2011, Workshop Gamification: Using Game Design Elements in Non-Game Contexts. 6-9.

Domínguez, A., Saenz-de-Navarrete, J., Luis de-Marcos, Fernández-Sanz, L., Pagés, C., Martínez-Herráiz, J. (2013): *Gamifying learning experiences*. Practical implications and outcomes. El-sevier, Volume 63, April 2013, pp 380–392.

Dörner, R., Göbel, S., Effelsberg, W., Wiemeyer, J. (2016): *SeriousGames- Foundations,Concepts and Practice*. 1st ed.; Springer International Publishing: Basel, Switzerland. doi:10.1007/978-3-319-40612-1

Erdősné, N. Á, Zsakó, L.(2018): *Grading Systems for Algorithmic Contests*. Olympiads in Informatics, 2018, Vol. 12, 159–166©2018 IOI, Vilnius University. DOI: 10.15388/ioi.2018.13. https://ioinformatics.org/journal/v12_2018_159_166.pdf

Faragó, B. (2015): *Tanulói aktivitás, aktív tanulás és tevékenység online környezetben* In: Papp-Danka, Adrienn; Lévai, Dóra (szerk.) Interaktív oktatásinformatika, 19-33.

Fekete, I., Gregorics, T., Kovácsné, P. K., Veszprémi, A. (2019): *Programming Theorems and Their Applications*. 2019 / 17 / 2 (5) - DOI: 10.5485/TMCS.2019.0466 - p. 213-241

Fotaris, P., Mastoras, T., Leinfellner, R., Rosunally, Y. (2015): *Who want to be Pythonista? Using Gamification to Teach Computer Programming*. 7th International Conference on Education and New Learning Technologies, Pages: 2611-2619.

Frankovic, I., Hoic-Bozic, N., Nacinovic Prskalo, L. (2018): *Serious Games for Learning Programming Concepts*. International Conference The Future of Education 8th Edition Florence, Italy 28-29 June 2018 ISBN: 978-88-3359-020-2 ISSN: 2384-9509.

Garris, R., Ahlers, R., Driskell, J.E. (2014): *Games, motivation and learning, A research and practice model*. Simulation and Gaming, 33(4), 441-467

Héjja-Nagy, K. (2015): *Tanulási stratégiák és a tanulói aktivitást befolyásoló egyéni feltételk online környezetben*. In: Papp-Danka, Adrienn; Lévai, Dóra (szerk.) Interaktív oktatásinformatika p. 33-49.

Horváth Győző, Horváth Gyula és Zsakó, L. (2017): *A Bíró és a Mester. Az online értékelés szerepe a programozásoktatásban*. INFOÉRA 2017.

Hundley, J. & Britt, W.(2009): *Engaging students in software development course projects*. The Fifth Richard Tapia Celebration of Diversity in Computing Conference. Intellect, Initiatives, Insight, and Innovations (TAPIA '09), ACM, New York, pp. 87–92.

Hülber, L. 2015): *Az adaptív online környezet lehetőségei az egyén fejlesztésében*. In: Papp-Danka, Adrienn; Lévai, Dóra (szerk.) Interaktív oktatásinformatika, p.138-151.

Joo, Y.J., Joung, S. Kim, E.K.(2013): *Structural Relationships among E-learners' sense of Presence, Usage, Flow, Satisfaction, and Persistence*. Educational Technology and Society, 16(2), 310-324., 2013.

Khaleel, F. L., Sahari, N., Meriam, T.S., Ismail, A. (2015): *The study of gamification application architecture for programming language course*. ACM IMCOM 2015 - Proceedings. Association for Computing Machinery, Inc, 2015. a17.

Lee, J., & Hammer, J. (2011): *Gamification in education: what, how, why bother?* Acad. Exch. Q. 15 (2), 1–5.

Li, C., Dong, Z., Untch, R.H., and Chasteen, M. (2013): *Engaging Computer Science Students through Gamification in an Online Social Network Based Collaborative Learning Environment*. International Journal of Information and Education Technology, Vol. 3, No. 1, February 2013

McDowell, C., Werner, L., Bullock, H., & Fernald, J. (2002): *The effects of pair-programming on performance in an introductory programming course*. SIGCSE '02 Proceedings of the 33rd SIGCSE technical symposium on Computer science education: Pages 38-42.

Muratet, M., Torguet, P., Viallet, F. & Jessel, J.P. (2010): *Experimental feedback on Prog&Play: a serious game for programming practice*. In: L. Kjelldahl and G. Baronosk (Eds.), EUROGRAPHICS 1–8.

M. Nádas, M. (2010): *Adaptív nevelés és oktatás*. Magyar Tehetségsegítő Szervezetek Szövetsége. Budapest. 2010

Pellas, N. (2014): *The influence of computer self-efficacy, metacognitive self-regulation and self-esteem on student engagement in online learning programs*. Evidence from the virtual world of Second Life Computers in Human Behaviour, 35, 157-170, 2014.

Pelling, N. (2011): *The (short) prehistory of "gamification", Funding Startups (& other impossibilities)*. Available at: https://nanodome.wordpress.com/2011/08/09/the-short-prehistory-of-gamification/

Rapos, N., Gaskó, K., Kálmán, O, Mészáros, Gy. (2011): *Az adaptív-elfogadó iskola koncepciója*. Oktatáskutató és Fejlesztő Intézet. Budapest. 2011. http://mek.oszk.hu/13000/13021/13021.pdf

Robins, A.,Rountree,J.&Rountree,N.(2003): *Learning and teaching programming: A review and discussion*. Computer Science Education, vol. 13, no. 2, pp. 137–172.

Sauvé, L., Renaud, L., Kaufman, D., & Marquis, J. S. (2007): *Distinguishing between games and simulations: A systematic review*. Educational Technology & Society, 10 (3), 247256.

Staubitz, T., et al. (2015): *Towards practical programming exercises and automated assessment in Massive Open Online Courses*. 2015 IEEE International Conference on Teaching, Assessment, and Learning for Engineering (TALE), IEEE, New York, pp. 23–30

Surendeleg, G., Murwa, V., Yun & H.K., Kim, Y.S. (2014): *The role of gamification in education–a literature review. Contemporary*. Engineering Sciences Vol 7 No 29–32, pp 1609-1616.

Szlávi, P., Zsakó, L., and Törley, G. (2019): *Programming Theorems Have the Same Origin*. Central-European Journal of New Technologies in Research, Education and Practice, 1(1), 1-12. https://doi.org/10.36427/CEJNTREP.1.1.380

Szlávi, P., Zsakó, L. (2004): *Módszeres programozás: programozási tételek*. Mikrológia 19, ELTE TTK Informatikai Tanszékcsoport

Takács, R., Ilyés, E., Bagyura, G., Megyesi, A., és Horváth Zoltán (2017): *Lemorzsolódás csökkentése az alapozó tárgyakat támogató tanulásmódszertan segítségével az ELTE Informatikai Karán*. Informatika

a felsőoktatásban 2017 konferencia, Debrecen, 2017. augusztus 29–31. https://www.inf.unideb.hu/sites/default/files/upload_documents/if_2017_konferencia_kiadvanya_20180117.pdf

Teague, D., & Roe, P. (2008): *Collaborative learning: towards a solution for novice programmers*. Paper presented at the Proceedings of the tenth conference on Australasian computing education-Volume 78.

Tsai, M.-J., Huang, L.-J., Hou, H.-T., Hsu, C.-Y., Chiou, G.-L. (2016): *Visual behavior, flow and achievement in game-based learning*. Computers & Education. 98, pp 115–129. doi: 10.1080/00461520.2015.1122533

Ulriksen, L., Madsen, M. M. and Holmegaard, H. (2010): *What Do We Know about Explanations for Drop out/Opt out among Young People from STM Higher Education Programmes?* Studies in Science Education 46:2, 209-244, DOI: 10.1080/03057267.2010.504549

Zsakó, L., Törley, G., Szlávi, P. (2017): *Az összetett programozási tételek is egy tőről fakadnak*. http://real.mtak.hu/78931/1/ZsLTGSzP.pdf 2017

Vihavainen, A., Luukkainen, M. & Kurhila, J. (2012): *Multi-faceted support for MOOC in programming*. Proceedings of the 13th Annual Conference on Information Technology Education, ACM, New York, pp. 171–176.

Winslow, L.E. (1996): *Programming pedagogy – A psychological overview*. SIGCSE Bulletin, Vol. 28, pp 1722.

Wolf, M., (2011): *The digital divide: arguments for and against Facebook, Google, texting, and the ages of social network*. Jeremy P. Tarcher/Penguin, New York. 34-37., 2007

8.2. A tézisfüzetben közvetlenül felhasznált saját közlemények jegyzéke

Balla Tamás és Király Sándor (2017): *Online programozás a kódolósuliban*. In: Szlávi, Péter; Zsakó, László (szerk.) INFODIDACT 2017, Budapest, Magyarország: Webdidaktika Alapítvány, (2017) Paper: 8

Balla Tamás és Király Sándor (2020a): *Enhancing learning efficiency after analysing the users' results in a gamified learning portal for computer programming education*. Absztrakt kötet. https://icai.uni-eszterhazy.hu/2020/?abstracts

Balla Tamás és Király Sándor (2020b): *Enhancing learning efficiency after analysing the users' results in a gamified learning portal for computer programming education*.Central-European Journal of New Technologies in Research, Education and Practice, Vol. 2, N o. 2. Közlésre elfogadva.

Hornyák Olivér és Király Sándor (2015): *Open edX kurzusok speciális követelményei és lehetőségei*. In: Ollé, János (szerk.) I. Nyílt Oktatás Konferencia Absztraktkötet, Eger, Magyarország: EKF Líceum Kiadó, (2015) pp. 15-16., 2 p.

Király Sándor (2011): *How to teach computer programming if our goal is the International Olympiad in Informatics*. Teaching Mathematics and Computer Science 9: 1 pp. 13-25., 13 p. (2011)

Király Sándor (2014): *Tehetséggondozás az informatikában csökkenő óraszámok mellett*. In: Kis-Tóth, Lajos (szerk.) Agria Media 2014, ICI 13, ICEM 2014: Információtechnikai és Oktatástechnológiai Konferencia és Kiállítás. 2014. október 8–10

Király Sándor (2015): *Tanulás támogatása digitális környezetben*. In: Hülber, László (szerk.) Digitális pedagógus konferencia 2015, Budapest, Magyarország: ELTE Pedagógiai és Pszichológiai Kar, (2015) p. 13

Király Sándor (2016a): *Tanulás támogatása digitális környezetben*. OKTATÁS-INFORMATIKA 2016: 1 pp. 29-40., 12 p. (2016)

Király Sándor (2016b): *How to Implement an E-learning Curriculum to Streamline Teaching Digital Image Processing*, ACTA DIDACTICA NAPOCENSIA 9: 2 pp. 13-22., 10 p. (2016)

Király Sándor (2016c): *Adaptív oktatás a moodle segítségével*. https://folyoiratok.oh.gov.hu/uj-kozneveles/adaptiv-oktatas-a-moodle-segitsegevel

Király Sándor és Balla Tamás (2016a): *Code School: a gamified learning portal for computer programming education*. In: Veronika, Stoffová; Zsakó, László; Szlávi, Péter (szerk.) New methods and technologies in education and practice: Proceedinhs of XXIX. DIDMATTECH 2016, Budapest, Magyarország: ELTE Informatikai Kar, (2016) pp. 89-94., 6 p.

Király Sándor és Balla Tamás (2016b): *Gamification a Programozás tanításában*. In: Hülber, László (szerk.) I. Oktatásszervezési és Oktatásinformatikai Konferencia [elektronikus dok.]: 2016-február 5-6.: absztraktkötet, Eger, Magyarország: Líceum Kiadó, (2016) pp. 22-22., 1 p.

Király Sándor és Balla Tamás (2020a): *The effectiveness of a fully gamified programming course after combining with serious games in a computer programming portal,* ACTA DIDACTICA NAPOCENSIA 13: 1 pp. 65-76., 12 p. (2020). DOI: 10.24193/adn.13.1.7

Király Sándor, Balla Tamás és Kusper, Gábor (2016): *Code School: Learn to code by practicing*. In: Emil, Vatai (szerk.) 11th Joint Conference on Mathematics and Computer Science, CEUR-WS.org, (2016) pp. 136-143., 7 p.

Király Sándor, Nehéz, Károly és Hornyák, Olivér (2017): *Some aspects of grading Java code submissions in MOOCs*, RESEARCH IN LEARNING TECHNOLOGY 25 Paper: 1945, 16 p.

Király Sándor és Székely, Szilveszter (2015): *How to use our own program evaluation system to streamline teaching computer programming*, TEACHING MATHEMATICS AND COMPUTER SCIENCE 13: 1 pp. 73-80., 8 p. (2015)

Kusper et al. (2016a), Kusper, Gábor ; Havasi, Gábor ; Király, Sándor ; Kocsis-Baán, Mária ; Nehéz, Károly ; Hornyák, Olivér ; Mileff, Péter: *Introducing MeMOOC and Recent Results*. In e-Learning at University of Miskolc, In: Darco, Jansen; Lizzie, Konings (szerk.) MOOCs in Europe: Overview of papers representing a collective European response on MOOCs as presented during the HOME conference in Rome November 2015: Papers 'WOW! Europe embraces MOOCS', Heerlen, Hollandia: European Association of Distance Teaching Universities (EADTU), (2016) pp. 75-78., 4 p

Kusper et al. (2016b), Kusper Gábor, Percze Gábor, Kovács László, Nehéz Károly, Radványi Tibor és *Király Sándor*: *A MeMOOC online informatikai egyetem koncepciója*, In: NIIFI (szerk.) Networkshop 2016, Budapest, Magyarország: Nemzeti Információs Infrastruktúra Fejlesztési Intézet (NIIFI), (2016) p. CD

Török Balázs és Király Sándor (2017): *Nemzetközi, nemzeti oktatás-informatikai szabályozások*, In: Kerülő, Judit; Jenei, Teréz; Gyarmati, Imre (szerk.) XVII. Országos Neveléstudományi Konferencia: Program és absztrakt kötet, Nyíregyháza, Magyarország: MTA Pedagógiai Tudományos Bizottság, Nyíregyházi Egyetem, (2017) pp. 117-117., 1 p.