https://doi.org/10.36007/4492.2023.225

TAPASZTALATOK ÉS NEHÉZSÉGEK AZ ÁLTALÁNOS ISKOLAI FELADATOK KÉSZÍTÉSE SORÁN STACK FELÜLETEN

VONTSZEMŰ Miklós[1]

ABSTRAKT
This article is about the analysis of the difficulties encountered in creating an online collection of problems and the experience gained in solving them, as part of an ongoing mathematics project. It describes the project, its objectives and the implementation process. The aim was to create random parameter and self-assessment tasks for each mathematical topic, using an online interface. The majority of the tasks in the exercise book were created in STACK. In solving the problems encountered in the creation of these tasks, certain methods were identified which could be usefully applied in the future. In this paper, some of the main experiences have been summarised and illustrated for specific tasks.

KEYWORDS
mathematics, problems, problem solving, STACK

BEVEZETÉS

A 1/0386/21 számú VEGA projekt kialakulásához az az egyszerű kérdés vezetett, hogy vajon miért eredményesek, vagy nem eredményesek a diákok matematikából. A kérdés még aktuálisabb lett a pandémia ideje alatti elektronikus oktatás során. A matematika eredményesség felmérésére egy online feladatgyűjtemény készült. Ebben a felmérésben kapóra jön, hogy tapasztalat alapján a matematika oktatásban igény volna egy online tanagyagra, ami feladatgyűjteményt is tartalmaz. Az igény létezését alátámasztja egy felmérés is, melyben a szlovákiai magyar iskolák matematika tanárait kérdezték meg [1]. Így az elsődleges hangsúly egy feladatgyűjtemény létrehozására került, mégpedig egy olyanra, ami parametrizált. A parametrizálás, vagyis a változók használata azért kapott kiemelt figyelmet, hogy ezáltal az egyes feladatok többször használhatóvá váljanak. A feladatokból tetszőleges tesztsor is előállítható, amit a rendszer azonnal kiértékel. A parametrizálás által egyedi feladatokat kell megoldania a felhasználónak, így házi feladatnál, vagy tesztelésnél a megoldás nem másolható. A projekt most ér az éles tesztelés fázisába, eddig a feladatbank fejlesztése kapott teret. A célkorosztály az általános iskola felső tagozata, vagyis az 5-9 évfolyamok. Az egész felső tagozat matematikai töltete, évfolyamokra és témakörökre bontva került kidolgozásra, ami a jövőben akár bővíthető. Törekedtünk a felbontást a Szlovákiában elfogadott ISCED alapján létrehozni [3]. A feladatgyűjtemény egy online felületen került kivitelezésre, ami a Selye János Egyetem szerveréről üzemel. A matek.ujs.sk oldal erre a célra lett létrehozva és moodle keretrendszerben működik. Azon belül egy STACK nevű software-t használtunk. A STACK rövidítés a „System for Teaching and Assessment using a Computer algebra Kernel“. A fel-

[1] Mgr. Vontszemű Miklós, Matematika Tanszék, SJE Komárom, vontszemum@ujs.sk

adatok létrehozásánál STACK-et használtuk a legtöbb feladat kialakítása során. Kivételt képeztek főként a geometria feladatok, mivel azokra a Geogebra appletek sokkal alkalmasabbak. Két fő ok, ami miatt a STACK mellett döntöttünk: randomizálás, vagyis a véletlenszerű változók kezelése, illetve a kiértékelés, helyesség ellenőrzésének beállításai voltak. Ezek mellet több érv is szólt a STACK mellett, nem utolsó sorban a kompatibilitása a moodle keretrendszerrel [4]. Ezen matematikai feladatok online integrálása során fellépő tapasztalatokról lesz szó a következőkben.

Feladatok létrehozása

A cél az volt, hogy egy komplex feladatgyűjteményt hozzunk létre, ami olyan feladatokból áll, melyek azonnal kiértékelésre kerülnek, ezáltal alkalmasak a gyakorlásra és a tesztelésre egyaránt. Fontos volt továbbá a véletlenszerű változók használata, hogy egy feladat többször is megoldható legyen. A feladatok STACK-ben való létrehozásához minimális programozási ismeret szükséges, ami az egyszerű feladatoknál nagyjából a változók deklarálásában és a kiértékelés elágazásaiban merül ki. Fontos megemlíteni, hogy a STACK a Maxima programnyelvet használja, és a LaTeX-ből ismert matematikai kifejezések többsége is alkalmazható ebben a környezetben [5].

Egy új STACK feladat létrehozásánál az első lépés a változók létrehozása. Változó lehet bármilyen betű vagy kifejezés, utána kettőspont, majd a kívánt érték pl.: *a:1;* vagy *eredmeny:5;* a változók után a sort pontosvesszővel zárjuk. Több lehetőségünk van arra, hogy véletlenszerű értéket adjunk egy változónak. Az egyik leggyakrabban használt a *rand(x)*, ahol a rand az angol random (véletlen) szóból és a zárójelben egy tetszőleges x értékből áll. Ekkor az *a:rand(x)* változó egy véletlen értéket kap, mégpedig 0 és x között.

A változók létrehozása után következeik maga a feladat szövege. A szövegben a változó megjelenítése kapcsos zárójelben, két „@” jel közé írva történik, pl.: {@a@}. Ekkor a feladat megjelenítésénél a már kigenerált érték fog szerepelni. Az „[[input:ans1]]” az első beviteli mező, ahová a választ várja a feladat. Ebből több is szerepelhet egy feladaton belül a megfelelő számozással. Hasonló módon a beviteli mezőkhöz tartozik egy „[[validation:ans1]]”, értelmezési mező, ami kiírja, hogy miként értelmezte a beviteli mező választ, ami hasznos lesz a későbbiekben.

Ezen kívül minden feladathoz tartozik egy visszajelzés: „[[feedback:prt1]]” ami azonnali értékelést hajt végre, így a felhasználó az ellenőrzés után rögtön látja a válasza helyességét. Továbbá, ha több választ tartalmaz a feladat, akkor megadható, hogy részleges helyesség esetén hány pontot kapjon a kitöltő. Az utolsó teendő egy feladat létrehozásánál, a helyes válasz/válaszok megadása. A válasz lehet konkrét érték, képlet, de lehet a fentiekben meghatározott változó is. Az akár több választ váró feladatok esetén a megfelelő ans1, ans2, … fülekben adhatjuk meg a helyes választ, itt további beállítással is találkozunk, mint például a beviteli mező hossza, típusa, és így tovább.

Egyszerűbb műveletek

Eddigi ismeretek tudatában egyszerű műveleteket tartalmazó feladatokat könnyedén létre lehet hozni. Ilyen feladatok alkotják például az ötödik osztály kezdő fejezetét is: az összeadás, illetve kivonás témakörét. Az összeadást tartalmazó feladat létrehozása talán a legegyszerűbb, nem igényel hosszú szöveget sem, csak a minden feladatnál használandó változók létrehozását, melyek random, azaz véletlenszerűen generálódnak. Szükség van tehát két változóra a véletlenszerű összeadandókhoz, majd egy harmadik változó lesz az eredmény, amit a megfelelő összeadás művelettel kiszámoltatunk. Végül a kiértékelésnél a rendszer ellenőrzi, hogy ezt az eredményt adta-e meg a felhasználó. A használt változók például a:rand(9);

b:rand(9), c:a+b; ahol a és b 0 és 9 közti számok a c pedig az összegük. Az eredménynél a helyes válasz a c értéke. Az első dolog, amit általában ajánlott kiküszöbölni, hogy ne álljon elő az az eset, hogy nulla plusz valami, vagy nulla plusz nulla, stb. Ezt könnyedén elkerülhetjük a a:rand(8)+1 használatával, ahol a generált érték már nem 0 és 8 között, hanem 1 és 9 közöti egész szám lesz. Mivel egyszerű feladatról van szó, így akár több ilyen feladat is kerülhet egy lapra. Ekkor annyiban változik a feladat, hogy több változóra, beviteli mezőre, és válaszmegadásra lesz szükség. Segítség lehet a több változó közti eligazodásnál ha a megoldásokat egységesen jelöljük, pl.: megoldás1:a+b vagy m1:c+d . Tetszőleges számú összeadás is kerülhet egy lapra, amit egyszerre ellenőriz majd a rendszer. Ha négy összeadást szeretnénk, célszerű lehet akár az utolsó kettőt valamiben módosítani, ami nem sokkal nehezebb, de mégis más feladat, pl. használunk több számjegyű számokat. Ekkor a megfelelő változókat kétszámjegyűnek generáljuk, pl. e:rand(89)+10 által 10 és 99 közti érték kerül majd ki.

Az első kiküszöbölendő probléma a kivonásnál a negatív eredmény, mivel 5.osztályban még nem kerül bevezetésre a negatív szám, ezért a kivonás csak úgy történhet, hogy a nagyobból vesszük el a kisebb értéket. Ez megfelelő változók használatával meg is oldható, akár több módon is. Például az első változó legyen kétjegyű a második pedig egy, így sosem fordul elő, hogy az első kisebb. Megoldás lehet még, ha két kétjegyű szám különbségére vagyunk kíváncsiak, hogy a tartományokat megfelelően állítjuk be a:rand(29)+40 és b:rand(9)+10 ekkor m:a-b biztosan nem lesz negatív, legrosszabb esetben is nulla, mivel a 40 és 69, b pedig egy 10 és 40 közötti szám. Megoldható továbbá olya módon is, hogy az első értéket a második érték valahányszorosaként adjuk meg pl.: b:rand(9)+10, a:b*rand(4)+1. Látható, hogy egy probléma több féle módok is megoldható, igyekezzünk mindig a legmegfelelőbben kiválasztani. A későbbi évfolyamban, ahol már bevezetésre kerül a negatív szám fogalma, már pont, hogy olyan feladatokra is szükség van, ahol nagyobb értéket kell kivonni, a kisebből és itt fordítva használnánk a fent említett megoldások valamelyikét.

A szorzásnál, hasonló a helyzet, mint az összeadásnál, az osztásnál viszont megint problémába ütközhetünk, a törtek ismeretének hiányában. Tehát a véletlengenerálást úgy kell megoldani, hogy az első érték osztható legyen a másodikkal maradék nélkül. Általában két random érték generálásánál ez aligha teljesülne. Ismét több lehetőség van a kiküszöbölésre, de felesleges például addig generálni értékeket, amíg nem áll elő megfelelő eset. Az ilyen, vagy hasonló esetekben tökéletesen alkalmazható a „visszafelé gondolkodás” módszere. Vagyis, az eredményt választjuk véletlen generált értéknek majd ezt beszorozzuk a második véletlen változóval, és az így kapott érték kerül az előre, amit osztunk a második változóval, így biztosan egész eredményt kapunk, ráadásul már meg is van az eredményünk egyetlen változóval, így nagyon egyszerű leellenőrizni. A változók például az m:rand(8)+1; b:rand(8)+1); a:b*m. Ez a módszer az összes eddigi feladatnál is használható lett volna, kiemelten a kivonásnál és az osztásnál célszerű használni.

Az eddig említettek, csak a legalapvetőbb feladatok voltak, de mivel véletlen változókkal van dolgunk ezért már ezeknél az eseteknél is előállhatnak nem várt helyzetek, ezért kell átgondoltan meghatározni a feladatok változóit. A szorzásnál nem célszerű a 0-át benne hagyni a lehetséges értékek között, míg az osztásnál nem is szabad, mivel tudjuk, a nullával való osztás nincs értelmezve, és a rendszer is hibát jelezne ebben az esetben.

Eddigi ismeretekből egy konkrét, egyszerű műveleteket tartalmazó feladat létrehozása így nézne ki lépésenként:

1. Változók deklarálása

A változókat a fentebb említettek szerint adtuk meg, ügyelve főként a kivonásra és az osztásra (a zárójelben lévő szöveg csak magyarázat):

a:rand(9)+10; (első összeadandó, kétjegyű)

b:rand(8)+1;	(második összeadandó, egyjegyű)
m1:a+b;	(első megoldás az összeg, kétjegyű)
c:rand(9)+10;	(kisebbítendő, kétjegyű)
d:rand(8)+1;	(kivonandó, egyjegyű)
m2:c-d;	(második megoldás a különbség)
e:rand(8)+1;	(első tényező, egyjegyű)
f:rand(8)+1;	(második tényező, kétjegyű)
m3:e*f;	(harmadik megoldás a szorzat)
m4:rand(8)+1;	(negyedik megoldás a hányados, egyjegyű egész szám)
g:rand(8)+1;	(osztó, egyjegyű)
h:m4*g;	(osztandó)

Fontos, hogy a visszafelé gondolkodás miatt a g és az m4 hamarabb kerüljön kigenerálásra, mint a h, ha előtte szerepelne, hibát jelezne a rendszer.

2. A feldat szövegének beírása

Számítsd ki!
\({@a@}+{@b@} = \) [[input:ans1]]
\({@c@}+{@d@} = \) [[input:ans2]]
\({@e@}*{@f@} = \) [[input:ans3]]
\({@g@}/{@h@} = \) [[input:ans4]]
[[validation:ans1]]
[[validation:ans2]]
[[validation:ans3]]
[[validation:ans4]]

3. A helyes válaszok megadása

A megoldásokat kell a megfelelő helyre beírni, vagyis az ans1 be az m1-et, ans2-be az m2-t, és így tovább, ahol az ans1 az első válasz, és az m1 a helyes megoldás.

További feladatköröknél fellépő problémák

A tizedes számokat használó feladatoknál tisztázandó kérdés egy online feladat megoldásánál, hogy az eredménynél az értékben ponttal vagy vesszővel jelezzük a tizedespont/vessző helyét. Ez tapasztalatból a köztudatban sem tisztázott pontosan. Ennek megoldása lehet, hogy a feladatok szövegében jelezzük, hogy a válaszban ponttal jelezze a felhasználó a tizedespont helyét. Sok feladatnál ez időigényes és a sokadik ilyen feladatban már a felhasználót is zavarhatja, ha például már tizedszer olvassa ezt a szöveget. Ezért inkább a visszajelzést használjuk ki, ez a funkció benne van a feladatokban. A „validation” rész abban az esetben, ha a felhasználó nem ponttal, hanem vesszővel írná a tizedesszámot, akkor jelzi, hogy az értéket nem tudja elfogadni és vessző helyett pontot vár. A lényeges h mindezt az ellenőrzés előtt teszi, így nem a beírt válasz után tudja meg a felhasználó, hogy nem elfogadott a válasz típusa. A tizedes számok összeadása és kivonása, nem igényel különösebb odafigyelést, viszont a szorzás és osztás már igen. Itt arra érdemes figyelni, hogy a véletlenül generált érték ne legyen túl nagy vagy túl kicsi. Még ha nem túl valószínű, hogy a két érték az adott intervallum szélső tartományaiból kerülne ki, van rá esély, így érdemes a megfelelő tartományból generálni a változókat.

Kerekítési feladatokat megadni egyszerű. Az online felületen való megjelenítése, illetve ellenőrzésénél viszont figyelni kell néhány dologra. Például míg egész számokat szeretnénk kerekíteni, addig nem találkozunk olyan esettel, amikor véletlenszerű tizedes számot generálunk. Ha például egy random háromjegyű számot osztunk százzal, akkor az egy tizedes szám lesz. A feladatnál megadjuk értelemszerűen, hogy hány tizedesjegyre kerekítse a felhasználó,

és itt előállhat az az eset, hogy a véletlen szám háromjegyű ugyan, de nullára végződik, és ilyenkor a kerekítendő szám hiába 2,60 a megjelenítésben is valószínűleg 2,6 ként fog szerepelni, és ezt egy tizedesjegyre kerekítve önmagát kapjuk, ami túl egyszerű feladat lenne. Ezt kiküszöbölendő nem árt más módon generálni a számokat, hogy ne végződhessen 0-ra. Ezt egy plusz lépéssel, egy 1-9 terjedő érték hozzáadásával megtehetnénk ugyan, de akkor egy if függvény segítségével kellene vizsgálni, ha nullára végződik a generált érték, akkor adná csak hozzá az értéket. Ez megoldaná a problémát, de használjunk egy gyorsabb megoldást. Előállíthatnánk a számot helyiértékesen is, úgy hogy rand(9)+1*100 + rand(9)+1*10+ rand(8)+1. Sőt még ezt is megírhatnánk egyszerűbben, mivel csak az utolsó szám a fontos, ekkor generálnánk egy háromjegyű számot, szoroznánk tízzel és hozzáadnánk egy számot 1-9-ig: (rand(899)+100)*10+rand(8)+1. Ezek a módszerek garantálják, hogy nem fog nullát tartalmazni a véletlen szám. Most, hogy már a kerekítendő értéket tisztáztuk, választhatunk, hogy melyik helyiértékre szeretnénk kerekíteni: tízesek, százasok, tizedek, századok, stb. Ezt megtehetjük fixen, manuálisan beírva, vagy szintén generálva, bár az utóbbi kicsit több munkával jár. A lényeg viszont az eredmény ellenőrzése, amelyhez jó, ha megismerünk egy új függvényt, mégpedig a round(x) függvényt, amely pont a kerekítésre szolgál. Pontosabban az x értékhez legközelebbi egész számot adja meg. Itt egy komolyabb gondba ütköztünk, mivel a rendszer nem ismeri a kerekítés megfelelő helyiértékre parancsot. Ezért önmagában a round függvény nem adna jó eredményt különböző helyiértékekere való kerekítésnél. Ezért bonyolultabb megoldást alkalmaztunk, az x értéket elosztjuk megfelelő helyiértékkel, így kerekítjük, majd visszaszorozzuk, így megfelelően fogja kerekíteni: m:round(float(x/10))*10; a float függvény azért szükséges mert így az x/10 értéke tizedes szám lehet, amit jelezni kell.

Maga a véletlen generálás nagyon hasznos a számunkra, de nem mindig a legelőnyösebb, néha célravezetőbb kiszűrni bizonyos eseteket. Ilyen eset például az oszthatósági tartományok témaköre, ahol fontos lehet, hogy egy szám ne végződjön valamilyen konkrét értékre, vagy a prímszámokat tartalmazó feladatok, ahol prímszámokat generálni nem is lenne egyszerű feladat. Ahhoz, hogy prím értéket adjunk egy változónak hasznos bevezetni a lista függvényt. Egy lista megadása a következő módon történik: maga a változó utána kettőspont és szögletes zárójelben az elemek vesszővel való felsorolásával. Íme egy példa: x:[2,3,5,7]. Az oszthatósági feladatok nagyrésze a már említett visszafelé haladás módszerével jól generálható, de akad például a prímtényezős felbontás feladat létrehozásánál egy újabb szempont, ez pedig az azonos nehézségű feladatok létrehozása. Az összeadásnál még nincs akkora nehézségkülönbség mondjuk az 5+5 és a 18+27 között, de prímtényezős felbontás feladatánál, ha a felbontandó érték teljesen véletlenszerű, akkor például a 8 gyorsabban felbontható, mint az 51, ha valaki nem tudja, hogy az 51 prímszám. A lista segítségével prímszámokat már tudunk generálni, viszont hasonló nehézségű feladatokat kicsit nehezebb. Eltérő paraméterű feladatoknál az azonos nehézség továbbra is elvárható. Tehát egyazon feladatnál a paraméter változásával az azonos megoldási idő a cél. Olyan feladatoknál, mint a prímtényezős felbontás elvégzése, a változók közti kis eltérés esetén is fennál a megoldási idő nagyobb mértékű változása. Attól, hogy tudjuk minden összetett szám felírható prímszámok szorzataként, valamely számok prímtényezős felbontását tovább tarthat felírni. Gondoljunk csak bele, hogy generálunk három prímszámot egy és száz között, ha a háromból, kettő már elég nagy, a feladat megoldása több időt vesz igénybe. Ehelyett a 2, 3 illetve 5 prímszámok valamilyen hatványon szereplő produktját vesszük, ami a tagok összeszorzását jelenti. Ha a hatványt is limitáljuk mondjuk háromig, akkor egész hasonló nehézségű feladatokat kaphatunk.

A következő dilemma a tizedes szám vagy tört válasz elfogadása volt, például 0.5 vagy 1/2. Hasonló a helyzet, mint a tizedesvessző vagy tizedespont esetében, itt sem a szövegben leírva figyelmeztetjük a felhasználót a válaszadás formájára. Ehelyett megadható az elfogadható

válasz típusa, ahol beállíthatjuk, hogy ne fogadjon el tizedes számot válasznak. Ezt a válaszok beállításánál a „forbid float" fület igen-re változtatva tehetjük meg. Ekkor a „validation" tizedes szám esetleges beírásánál jelzi a felhasználónak, hogy nem elfogadott választípus és tört formában kéri a választ, mindezt ismételten az ellenőrzés előtt.
Felmerül továbbá a tört válasz megadásának módja. Az alapértelmezett tört válasznál a számláló és nevező közé a „/" jel kerül, például: 2/3. Erre fel is hívja a „validation" a figyelmet, viszont előfordulhat, hogy bonyolultabb törtkifejezéseknél nem így szeretnénk látni a választ. Erre több megoldást találhatunk, egyik lehetséges ilyen, hogy a tört számlálóját és nevezőjét két válaszmezőként értelmezzük. Kiíratásnál ez annyit tesz, hogy egymás alá írva és egy (tört)vonalat húzva a két beviteli mező közé a felhasználó annyit lát, hogy tört formájú a válasz és külön kell beírni a számlálót és nevezőt. A feladat válasz részénél pedig a két válasz értékét elosztva egymással ellenőrizzük le helyességét. Törtkifejezések létrehozása akár a LaTeX-ből is ismert „frac" funkció alkalmazásával történik, melynek két paramétere van amit kapcsos zárójelek közé kell írni: frac{1}{2}; ahol az első zárójel a tört számlálója, a második pedig a tört nevezője lesz, vagyis ez az 1/2-et jeleníti meg. A zárójelek közé kifejezések vagy polinomok is kerülhetnek. Mint minden törtet tartalmazó feladatnál, itt is ügyelnünk kell arra, hogy a nevező nem lehet nulla, mivel a nullával való osztás nincs értelmezve a valós számhalmazon. Míg egyszerű nevezőjű törteket generálunk, addig ez nem jelent különösebb gondod. Ellentétben, ha bonyolultabb nevezőjű törtet generálunk, amiben esetlegesen ismeretlen is szerepel, mindig ügyeljünk, hogy a nevező értéke ne legyen nulla.
Halmaz műveleteket tartalmazó feladatokat, az eddigi ismeretek használatával akár létre is tudnánk hozni, a generált értékeket a szövegben kapcsos záróljelben jelenítenénk meg. A nehézség ezután jönne, hogy határozzunk meg egy műveletet, akár az uniót, akár a metszetet. Illetve fennáll a veszélye, hogy két egyforma érték kerül egy halmazba, ami hiba lenne. Az ismeretek bővítésére van szükség a feladat létrehozásának megkönnyítése érdekében. Az első a halmaz, mint változó deklarálása, ami a következő módon történik: A:{1,2,3,4}; vagy B:{a,b,c}. Mint látható a halmazt a közismert nagy betűvel adjuk meg és az elemei lehetnek konkrét értékek, vagy változók is, amiket kapcsos zárójelben felsorolunk. A következő újdonság, a függvények használata, mégpedig a halmazműveletek függvényei, például az unió: m1:union(A,B); vagy a metszet: m2:intersection(A,B). A további függvények is megtalálhatók mint a kivonás: m3:setdifference(A,H); ami tökéletesen használható a komplementer számítására is, ahol a H az alaphalmazt jelenti. Eredménynek kapcsos zárójelben felsorolt értékeket vár, és nem számít a sorrend, tehát a {1,3} és a {3,1} mindkettő helyes megoldásnak számít. Látható, hogy a halmaz, mint változó és a függvények ismerete nélkül bonyolult és hosszadalmas lett volna a feladat létrehozása, ebből is adódik, hogy érdemes először utánanézni, hogy létezik e megfelelő megoldás a Maxima nyelvben.
Zárójeleket és abszolútértéket tartalmazó feladatoknál, az első gondolat után elvetésre került a változókon kívüli randomizálás, vagyis a műveleti jelek, illetve a zárójelek pozíciójának véletlenszerű változtatása, mivel nagymértékben bonyolítja az eredmény meghatározását. Ezért, ha elsőre jó ötletnek tűnik is az ilyen mértékű véletlenszerűen generált feladat, inkább törekedni kell az egyszerűségre. Egy ilyen mértékű parametrizált feladat létrehozása lehetne egy külön projekt. Másrészt azért is került elvetésre, mert úgy nem biztosított az egyenlő nehézségű feladat a felhasználók számára. Mint már korábban is említésre került, fontos szem előtt tartani, hogy bár célunk a megismételhető, és több felhasználó által számolható feladatok generálása, az is szempont, hogy legyen azonos a nehézség. Mivel maradtunk a véletlenszerű, de egyszerű vonalon ezért a műveleti jelek és a zárójelek pozíciója egyszer került rögzítésre. Ennél a feladatnál igyekeztünk a fixen rögzített zárójeleket logikusan elhelyezni, lefedve a legtöbb különböző esetet.

Akármilyen gondosan is tervezzük meg a feladatokat, előállhat olyan eset, ahol valamit elnéztünk, nem gondoltuk át megfelelően, vagy csak változtatni szeretnénk a paraméteren. Ezért törekedjünk úgy létrehozni a feladatot, hogy lehetőleg egy-egy megfelelő változó átírása elegendő legyen. Főként a megoldás beírásánál nem javasolt a megoldás képletét beírni, helyette magát a változót megadni.

Nehézségek és tippek összefoglalása

Nehézségek:

- adott célkorosztály figyelembevétele
- a válasz megfelelő típusa
- bizonyos feladatok parametrizálása, megfelelő random tartomány
- nehézségazonos feladatok
- feladatok javítása vagy módosítása

Tippek:

- megfelelő parametrizálás
- feladatok létrehozásánál „visszafelé gondolkodás” módszere használni
- egyszerűségre törekvés
- átlátható, könnyen javítható, módosítható feladatokra való törekvés
- nehézségi különbségek kiküszöbölése
- helyiértékes gondolkodás
- megfelelő függvény használata
- optimalizásál

BEFEJEZÉS

Matematikai parametrizált feladatok létrehozása online felületen nem annyira egyszerű, mint amennyire elsőre tűnhet. Egyszerre kell átgondolni, hogy mit szeretnénk egy feladattól, és hogy hogyan tudjuk azt a legmegfelelőbben létrehozni. A parametrizálás nem kell, hogy a bonyolultságot is jelentse. A cél, minél egyszerűbben létrehozni egy feladatot úgy, hogy többször is megoldható legyen ugyan azon felhasználó számára. A feladat így lesz alkalmas a gyakorlásra, vagy akár a felmérésre. Az eddig sorra vett szempontok segíthetnek matematikai STACK feladatok létrehozásánál, de akár más felületen is alkalmazhatóak ezek a módszertani megfigyelések. A STACK egyszerre nyújtott segítséget a moodle-ban való átfogóbb feladatok létrehozásánál, és állított kihívások elé. Összességében a feladatok létrehozásánál szerzett tapasztalatok hasznosak lehetnek általános matematikai feladatok tervezésénél és kivitelezésénél is.

Köszönetnyilvánítás

Ez a tanulmány a 1/0386/21 számú *Analýza dôvodov úspešnosti/neúspechu študentov v matematike s dôrazom na elektronické dištančné vzdelávanie* (*A diákok eredményesség/eredménytelenség okainak vizsgálata a matematikában az elektronikus, távolléti oktatásra való tekintettel)* című VEGA projekt támogatásával készült.

Irodalomjegyzék

[1] Csiba, P., Svitek, Sz. (2020) The Presence and Future of Teaching Mathematics online - Focus on Hungarian Language in Education in Slovakia. DisCo, 25-33.
[2] Gall, M. (2014) Experience with STACK in teaching of mathematics. ZAMAT
[3] ISCED2 Matematika: https://www.statpedu.sk/files/articles/dokumenty/inovovany-statny-vzdelavaci-program/matematika_nsv_2014.pdf
[4] Sangwin, C. (2010) Who uses STACK? A report on the use of the STACK CAA system
[5] STACK: https://docs.stack-assessment.org/content/2019-STACK-Guide.pdf