

SZAKDOLGOZAT



MISKOLCI EGYETEM

Matematikai kvízjáték és kártyapakli készítő program

Készítette:

Téglás Réka

Programtervező informatikus

Témavezető:

Dr. Körei Attila

MISKOLC, 2021

MISKOLCI EGYETEM

Gépészszmérnöki és Informatikai Kar

Alkalmazott Matematikai Intézet Tanszék

Szám:**SZAKDOLGOZAT FELADAT**

Téglás Réka (WWSAMB) programtervező informatikus jelölt részére.

A szakdolgozat tárgyköre: matematikai kvízjáték, kártyapakli készítése**A szakdolgozat címe:** Matematikai kvízjáték és kártyapakli készítő program**A feladat részletezése:**

A szakdolgozati feladat fő célja egy matematikai készségfejlesztő kártyajáték létrehozását támogató alkalmazás fejlesztése Java környezetben. A játékmenet kitalálása és a kártyalapok megtervezése is a feladat részét képezi, az alkalmazás pedig legyen képes a kártyajáték lapjainak fizikai előállítására.

A kártyalapok megrajzolása mellett a L^AT_EX formátumot támogató szövegdoboz beágyazása is a fejlesztési feladat része, valamint a kitöltött kártyák adatbázisba mentését és a nyomtatási lehetőség alkalmazásba történő beépítését is meg kell oldani.

Témavezető: Dr. Körei Attila (egyetemi docens)**A feladat kiadásának ideje:**.....
szakfelelő

EREDETISÉGI NYILATKOZAT

Alulírott **Téglás Réka**; Neptun-kód: WWSAMB a Miskolci Egyetem Gépészmérnöki és Informatikai Karának végzős Programtervező informatikus szakos hallgatója ezennel büntetőjogi és fegyelmi felelősségem tudatában nyilatkozom és aláírásommal igazolom, hogy *Matematikai kvízjáték és kártyapakli készítő program* című szakdolgozatom saját, önálló munkám; az abban hivatkozott szakirodalom felhasználása a forráskezelés szabályai szerint történt.

Tudomásul veszem, hogy szakdolgozat esetén plágiumnak számít:

- szószerinti idézet közlése idézőjel és hivatkozás megjelölése nélkül;
- tartalmi idézet hivatkozás megjelölése nélkül;
- más publikált gondolatainak saját gondolatként való feltüntetése.

Alulírott kijelentem, hogy a plágium fogalmát megismertem, és tudomásul veszem, hogy plágium esetén szakdolgozatom visszautasításra kerül.

Miskolc,évhónap

.....

Hallgató

1.

szükséges (módosítás külön lapon)

A szakdolgozat feladat módosítása

nem szükséges

.....

dátum

témavezető(k)

2. A feladat kidolgozását ellenőriztem:

témavezető (dátum, aláírás):

.....
.....
.....

konzulens (dátum, aláírás):

.....
.....
.....

3. A szakdolgozat beadható:

.....

dátum

témavezető(k)

4. A szakdolgozat szövegoldalt

..... program protokollt (listát, felhasználói leírást)

..... elektronikus adathordozót (részletezve)

.....
..... egyéb mellékletet (részletezve)

.....

tartalmaz.

.....

dátum

témavezető(k)

5.

bocsátható

A szakdolgozat bírálatra

nem bocsátható

A bíró neve:

.....

dátum

szakfelelős

6. A szakdolgozat osztályzata

a témavezető javaslata:

a bíró javaslata:

a szakdolgozat végleges eredménye:

Miskolc,

a Záróvizsga Bizottság Elnöke

Tartalomjegyzék

Köszönetnyilvánítás	1
1. Bevezetés	2
1.1. Témaválasztás	2
2. Koncepció	3
2.1. Elméleti háttér: használt nyelvek, környezetek	3
2.1.1. Java	3
2.1.2. LaTeX	4
2.1.3. NetBeans	4
2.2. Az alapötletet adó játék bemutatása	5
2.2.1. Cardgame Toolkit	5
2.2.2. Legyen Ön is milliomos!	5
2.3. Specifikációk és követelmények	6
2.3.1. Felhasználói követelmények	6
2.3.2. Rendszerkövetelmények	8
2.3.3. Java, mint objektum orientált nyelv	10
2.3.4. Apache Maven Project	10
3. Tervezés	11
3.1. Folyamatábra	11
3.1.1. Folyamatábra felépülése implementálás közben	12
3.2. UML ábra	15
3.2.1. Use-case diagram	15
4. Megvalósítás	17
4.1. Az adatbázis	17
4.1.1. Relációs adatmodell	18
4.1.2. Adatbázis-kapcsolat létrehozása	18
4.2. A két játékrész közös osztályai – signinup csomag	19
4.2.1. Signinup és Signup osztály	19
4.2.2. SignupDBM osztály	20
4.2.3. SignupDBM osztály – LogIn()	20
4.2.4. SignupDBM osztály – Crypting()	21
4.3. Legyen Ön is milliomos! osztályai – game csomag	23
4.3.1. PreGame és Game osztály	23
4.3.2. Ranking és BarChart osztály	23
4.3.3. GameAlgorithms osztály	23
4.3.4. GameAlgorithms osztály – Randomizer()	25

4.3.5.	GameAlgorithms osztály – GetQuestion()	26
4.3.6.	GameAlgorithms osztály – LateXConverter()	27
4.3.7.	GameAlgorithms osztály – saveq()	28
4.3.8.	GameAlgorithms osztály – generate()	30
4.3.9.	GameAlgorithms osztály – playerhelp2randomizer()	30
4.4.	Kártyapakli készítő osztályai – deckCreate csomag	33
4.4.1.	FirstPg osztály	33
4.4.2.	TextEditor és DeckCreate osztály	33
4.4.3.	Selector osztály	33
4.4.4.	DBM osztály	34
4.4.5.	DBM osztály – Insert()	35
4.4.6.	DBM osztály – captureComponent()	35
4.4.7.	DBM osztály – Image()	36
4.4.8.	DBM osztály – CardLoader()	37
4.5.	Project Files - pom.xml	38
4.6.	Futtatható állományba generálás	39
5.	Tesztelés	41
5.1.	A játék menete	41
5.1.1.	Nyeremények	44
5.1.2.	Játékszabályok	44
5.1.3.	A játék vége	47
5.2.	Kártyapakli készítése	49
5.2.1.	Funkciók – Nyomtatás, Másolás, Frissítés	51
5.3.	Tesztfuttatás	54
5.3.1.	Tesztelés ismerősökkel	55
6.	Összefoglalás	57
Irodalomjegyzék		58

Köszönetnyilvánítás

Ezúton szeretném köszönetet mondani témavezetőmnek, Dr. Körei Attila egyetemi docensnek a hónapokon keresztül tartó segítségéért, és szakmai tudásáért, amelyek hozzájárultak szakdolgozatom elkészüléséhez.

Mindezek mellett hálás vagyok azoknak a Tanáraimnak, akik még segítséget nyújtottak a szakdolgozatom elkészítésében: köszönet Piller Imrének és Gégény Dávidnak az elméleti és gyakorlati segítségért; Lengyelné Dr. Szilágyi Szilvia Tanárnőnek a sok kvízkérdésért Analízis témakörben; valamint köszönet Dr. Olajos Péter egyetemi docensnek, amiért tanulmányaim során betekintést nyújtott a L^AT_EX szövegszerkesztőbe, és ezt a tudást fel tudtam használni a szakdolgozatom megírása közben.

Külön köszönetet szeretném mondani még Dr. Krizsán Zoltán egyetemi docensnek, hogy sok éves szakmai tudásával segítette a szakdolgozatom megvalósulását.

Végül pedig köszönetet mondanék a Családomnak, Barátaimnak, és különösen a Páromnak a rengeteg támogatásért, biztatásért és kitartásért minden félévem során, valamint a segítségükért a szakdolgozatom tesztelésében, ezáltal az elkészítésében.

Köszönöm mindenkinél!

1. fejezet

Bevezetés

1.1. Témaválasztás

A szakdolgozatom témája, a nyári szakmai gyakorlatom továbbfejlesztése volt, hiszen a Miskolci Egyetem Alkalmazott Matematikai Intézeti Tanszékén végeztem azt.

Az alap cél egy matematikai készségfejlesztő kártyajáték létrehozását támogató alkalmazás fejlesztése volt, Java környezetben. Pontosabban, egy olyan Java alkalmazás létrehozása, amely matematikai készségfejlesztő kártyajáték lapjainak előállítására használható. A kártyalapok megrajzolása mellett a L^AT_EX formátumot támogató szövegdoboz beágyazása is a fejlesztési feladatom része volt, valamint a kitöltött kártyák adatbázisba mentésének megoldása és a nyomtatási lehetőség alkalmazásba történő beépítése.

Mindezeket a kvíz kártyákat tovább gondolva, és továbbfejlesztve azon funkció megvalósítása jutott eszembe, hogy lehetőség legyen számítógépen is játszani ezzel az alkalmazással, kvízjáték formájában. Itt egy már meglévő – általam előállított – adatbázisból jönnek a Játékos elé a kérdések, amelyeket helyesen meg kell válaszolni, és így eljutni a főnyereményig.

Ezt a kvízjátékot alapvetően a jól ismert televíziós kvízjáték és műveltségi vetélkedő ihlette, a Legyen Ön is Milliomos!. Azonban az általam megvalósított játékban csak matematikai témajú kvízkérdések szerepelnek: egyszerű logikával kiszámítható kérdések, nagy százalékban az Analízis I. és II. című tárgyaim során tanult tananyagból, azonban vannak benne egyszerűbb Valószínűségszámítás és Numerikus analízis kérdések is.

2. fejezet

Koncepció

2.1. Elméleti háttér: használt nyelvek, környezetek

2.1.1. Java

A programom megírásához a Java nyelvre esett a választásom, hiszen ezzel a nyelvvel ismerkedtem meg a legrészletesebben egyetemi tanulmányaim során, az "Objektum Orientált Programozás" című tárgy keretein belül. Az alábbi fejezet írásakor a [1] [2] [3] forrásokra és jegyzetekre támaszkodtam.

A Java egy általános célú, teljesen objektum orientált programozási nyelv. Az 1990-es években a fő cél egy platform független programozási nyelv kifejlesztése volt, egy már létező nyelvből – a C++-ból. 1995-ben ez meg is valósult, létrejött a Java 1 platform, 1996-ban pedig a JDK 1.0. A nyelv támogatja a webes és az elosztott alkalmazások fejlesztését, valamint a több szalon futó programok fejlesztését, és mindezek mellett platform független (WORA: write once run anywhere).

A Java programok működéséhez szükség van egy telepített Java szoftverre az adott eszközön, egy **JRE**-re (*Java Runtime Environment = javat futtató környezet*), amely ingyenesen letölthető. A JRE-t egy másik, a program elkészítéséhez szükséges komponens tartalmazza, a **JDK** (*Java Development Kit = java fejlesztői csomag*). Én a programom elkészítése során **JDK 11**-t használtam.

A kinézetet Java Swing-el készítettem el, amely a Java részeként egy grafikus felhasználói felület elkészítésére szolgáló komponens. A Java Swing GUI, azaz a *grafikus felhasználói felület*, valójában komponensek halmaza. Ilyenek például az ablakok, panelek, nyomógombok, szövegdobozok, menüsorok, legördülő listák – csak hogy néhányat említsek azokból, amelyeket én is használtam az alkalmazásom megírása közben.

A legfontosabb külső library-k, amiket használtam a program megírásához:

1. Az *itextpdf-5.5.9*-re volt szükségem a kártyalapok PDF fájlba való mentéséhez.
2. A *jlatexmath-1.0.7* segítségével a program a LaTeX parancsokat tudja értelmezni, ami a matematikai kifejezések értelmezéséhez nélkülözhetetlen.
3. A *mysql-connector-java-8.0.19* tartalmazza az adatbázis kapcsolat létrehozásához illetve a műveletek elvégzéséhez szükséges JDBC driver-t.

Mivel Java Maven verziókezelőt használtam, így ezek mint függőségek kerültek a projektbe a `pom.xml` fájlba, amelyről a 4.5 alfejezetben írok.

2.1.2. LaTeX

Korábbi félévem során már megismerkedtem a L^AT_EX dokumentumleíró nyelvvel a "Bevezetés a T_EX szövegszerkesztésbe" című tárgy keretein belül, így a nyelv, és a technológia nem volt ismeretlen számomra. Köszönhetően ennek a tárgynak, az órán bemutatott [4] jegyzetre, valamint az órán ajánlott [5] irodalomra támaszkodtam a dolgozat megvalósítása során. Mindezek mellett pedig programtervező informatikus hallgatóként magát a szakdolgozatomat is L^AT_EX formátumban kell megírnom.

A L^AT_EX egy T_EX-en alapuló szövegformázó rendszer, egy jelölőnyelv, azaz szövegek annotációjára/kivonat készítésére szolgáló számítógépes nyelv. Ez a szövegszerkesztő az alapszintű szövegformázástól kezdve egészen a nyomdal felhasználásig tud létrehozni magas minőségen formázott dokumentumokat, műszaki tudományos, vagy egyéb jellegű iratokat.

A L^AT_EX egy leíró nyelv, amit meg kell tanulni a használatához. A parancsok angol nyelven vannak, és a nyelvet értők számára az összes parancs teljesen értelmes és logikus. Ezek mellett a L^AT_EX platformfüggetlen, valamint Windows operációs rendszerre a MiKTeX¹ a T_EX/L^AT_EX ingyenes változata.

2.1.3. NetBeans

Az Apache NetBeans² fejlesztő környezettel a tanulmányaim során nem találkoztam, azonban a játék elkészítése közben döntöttem úgy, hogy ezt fogom használni, nem pedig az Eclipse IDE-t. Mindennek az volt az oka, hogy úgy véltem, hogy a NetBeans több, átláthatóbb és modernebb komponensekkel rendelkezik a design terén. Elméleti hátterének az interneten jártam utána a [6] [7] weboldalakon.

Ez a fejlesztőkörnyezet szintén egy platformfüggetlen integrált Java fejlesztői környezet, azaz **IDE** (*Integrated Development Environment*). Legnagyobb előnye számomra a vizuális GUI tervezésben nyilvánult meg, hiszen fejlett Swing támogatásával gyorsan és rugalmasan van lehetőség az általunk elképzelt GUI megalkotására.

A NetBeans természetesen rendelkezik olyan funkciókkal, mint az Eclipse, gondolok itt a kényelmesebb és gyorsabb kódíráshoz tartozóan a nyelv kulcsszavainak különböző színekre való átszínezésére, amely átláthatóbbá teszi a kódot, vagy az automatikus kódgenerálásra és kódkiegészítésre, valamint a szintaktikai hibák jelzésére – ezekhez még van, hogy megoldási lehetőséget is kínál.

A szakdolgozatom írása során a legfrissebb változattal, a 12.3-al dolgoztam.

¹<https://miktex.org/download>

²<https://netbeans.apache.org/download/nb123/nb123.html>

2.2. Az alapötletet adó játék bemutatása

2.2.1. Cardgame Toolkit

A Cardgame Toolkit³-tel a nyári szakmai gyakorlatom során ismerkedtem meg. Az volt a feladatom, hogy ehhez az online kártyapakli készítő felülethez hasonló alkalmazást hozzak létre.



2.1. ábra. A Cardgame Toolkit logója.

A Cardgame Toolkit programmal különböző, felhasználók által készített paklikat lehet megnézni és letölteni, emellett saját kártyapakli létrehozására is van lehetőség, melyben magunk állíthatunk be különböző kinézeteket, valamint elrendezéseket a kártyalapok előlapján és hátlapján egyaránt. A kártyalapokra szövegeket vihetünk fel, és akár illeszthetünk be ábrákat, képeket is.

2.2.2. Legyen Ön is milliomos!

A Legyen Ön is milliomos!⁴ egy televíziós kvízjáték, egy műveltségi vetélkedő. 1998-ban sugározták először a műsort az Egyesült Királyságban, Magyarországon pedig 2000 elején debütált a műsor.



2.2. ábra. A Legyen Ön is milliomos! logója.

A játékos nagyon magas díjakat nyerhet, ha a 15 egymást követő feleletválasztós kérdésre jól felel. A fődíj Magyarországon kezdetben 25, később 40 millió Forint volt, jelenleg 50 millió Forint.

A játékban egyszerre csak egy játékos játszik, és a hangsúly a feszültségen van, nem pedig a gyorsaságon, ugyanis nincsen időkorlát a kérdések megválaszolására.

³<https://cardgametoolkit.com/card-style>

⁴https://rtl.hu/rtlklub/milliomos/cikk/2012/02/29/legyen_on_is_milliomos_jatekszabalyzat

2.3. Specifikációk és követelmények

Ebben a fejezetben a szoftver specifikációit szeretném definiálni, hiszen ez egy ellen- gedhetetlen fázis a szoftverfejlesztés folyamatában. Ezzel a lépéssel a szoftver funkciói, valamint az elvárások tisztázódnak, felhasználói és fejlesztői szempontból is.

Mivel korábbi féléveimben megismerkedtem a szoftvertervezés alapjaival, a "Szoftvertechnológiák" című tárgy keretein belül, így az órán használt [8] jegyzetre, valamint az [9] ajánlott irodalomra támaszkodtam, és egy, az interneten talált [10] jegyzetre.

2.3.1. Felhasználói követelmények

A felhasználói követelmények a funkcionális és nem funkcionális követelmények leírá- sából állnak, mindenek egyszerű nyelvezettel megfogalmazva.

A **funkcionális követelmények** a rendszer működését írják le, valamint a felhasználó által igényelt összes szolgáltatást. A következőkben két részre bontom ezeket a funkcionális követelményeket: arra, hogy a játék megkezdése előtt milyen szolgáltatá- sokra van szükség, valamint hogy a játék során milyen funkciók szükségesek.

Játék előtt:

- A Játékosnak szükséges regisztrálnia. A regisztrációhoz *felhasználónév*, *teljes név*, *jelszó*, és az egyetemi *kar* megadása szükséges.
- Regisztráció után belépés a játék "elő" felületére.
- A Játékosnak választania kell, hogy a *kvízjáték*, vagy a *kártyapakli készítő* felületét indítja-e el.

Játék közben:

- Kvízjáték:
 - Játékos felhasználónevének megjelenítése.
 - Menüsorban visszalépési és kilépési lehetőség.
 - Megnyerhető összegek listájának megjelenítése a kérdések sorszámozásával, valamint jelzés, hogy a Játékos éppen melyiknél tart.
 - A összeglistán kívül egy aktuális pénzösszeg jelző.
 - A Játékos segítségeinek megjelenítése.
 - A kérdés megjelenítése, valamint a négy válaszlehetőség, amelyek megjelölésével megy tovább a játék folyamata.
 - Ha a Játékos helyes válasz után továbbmegy, akkor következzen számára a következő kérdés.
 - Ha a Játékos helyes válasz után megáll, akkor a játék befejeződik, a szerzett összege pedig mentésre kerül.
 - Ha a Játékos helytelen választ ad, akkor az összege nem kerül mentésre, kivéve, ha túl volt már valamelyik szakaszkérdésen (5., 10.).
 - Ha a Játékos minden a 15 kérdésre helyesen válaszolt, a fődíjat megnyerte, a játék véget ért.
 - Ha a játék véget ért, új játék kezdetére legyen lehetőség.

- Kártyapakli:
 - Játékos felhasználónevének megjelenítése.
 - Menüsorban visszalépési és kilépési lehetőség.
 - Kártya színének kiválasztása az alapszínekből, vagy hexadecimális színkód-dal megadva.
 - Kártyalap kérdés oldalának kitöltése: kérdés megadása, matematikai kifeje-zés megadása L^AT_EXparancssal, ábra beszúrásának lehetősége.
 - Kártyapakli ID-jének megadása, valamint a kártyalap nevének megadása. A megadott pakli ID és kártyalap neve segítségével a kártyalap mentése.
 - Kártyalap szerkesztési és másolási lehetősége.
 - Kártyapakli nyomtatási lehetősége, amelyet ezáltal egy PDF fájlba generál a program.

A **nem funkcionális követelmények** az egész rendszerre vonatkozó tulajdon-ságokra koncentrálnak, valamint a szolgáltatások megszorításait tárgyalják és azok viselkedését. Itt esik szó például a termék hatékonyságáról, megbízhatóságáról, fel-használhatóságáról, hordozhatóságáról.

Felhasználhatóság:

- Felhasználóbarát felület, egyértelmű instrukciókkal.
- Bonyolultabb leírás, segédlet nem szükséges a használatához.

Hordozhatóság:

- Szinte minden személyi számítógépen használható az alkalmazás, Windows és Linux alapú operációs rendszereken is.
- Azonnal használható, nem igényel külön telepítést.

Hatókonysság:

- Memóriaigénye legyen reális a játék funkcióihoz képest.
- Relatív rövid válaszidő, néhány másodperces.

Környezeti követelmények:

- Az alkalmazás az adatok tárolására egy SQL alapú adatbázis-szertvert használ.

Működési követelmények:

- Átlagos gyakoriságú használat.
- Maximum néhány órás, átlagban rövid futás.

Fejlesztési követelmények:

- Java nyelv, NetBeans környezet.
- Objektum orientáltság, Java Maven verziókezelő segítségével.

2.3.2. Rendszerkövetelmények

A rendszerkövetelmények alatt a felhasználói követelmények – azaz a funkcionális és a nem funkcionális követelmények – egy részletesebb leírását értjük. Itt ugyanazon pontokon fogok végighaladni, amelyeket a felhasználói követelményekben megfogalmaztam, csak a rendszer szempontjából fogom tekinteni, hogy az adott pont hogyan is valósítható meg.

Magához az egész játék működéséhez szükséges egy "motor", amelyben minden olyan algoritmus és metódus meg van írva, amely a kód és a játék helyes futásához elengedhetetlen. Ezt a kódot főként a `GameAlgorithms` és a DBM osztályok tartalmazzák, a Belépés és Regisztrációhoz szükséges eljárások pedig a `SignupDBM` osztályban találhatóak.

Először is a **funkcionális követelményeket**, azaz a szolgáltatásokat, funkciókat részletezném.

Játék előtt:

- A Belépés és Regisztráció felület megvalósítása `JFrame` Form-ok segítségével (amely a `javax.swing.JFrame` osztály leszármazottja), rajta különböző Java Swing komponensekkel. Megvalósítás részletezése a 4.2 és az 5.1 alfejezetekben.
- A Belépés utáni játék "elő" felület megvalósítása, ahol a Játékosnak választani kell a két játékrész között, szintén `JFrame` Form segítségével. Ezek részletezése szintén az 5.1 alfejezetben található.

Játék közben:

- Kvízjáték:
 - A kvízjáték során, a GUI-val kapcsolatban felmerülő felhasználói követelmények megvalósításának egy része szintén a Java Swing segítségével.
 - Az összes kattintáshoz köthető tevékenység megvalósítása a `JButton` gombokhoz kapcsolódó `actionPerformed` metódusok segítségével.
 - Összegek kiíratása a Játékos számára a GUI felületre SQL-lekérdezésekkel az adatbázisból.
 - Játékos segítségeinek megjelenítése szintén `JButton` gombok használatával.
 - Kérdés és válaszok megjelenítése az adatbázisból SQL-lekérdezéssel, a matematikai kifejezések átkonvertálása \LaTeX formátumba, `jlatexmath` könyvtár felhasználásával.
 - A definiált algoritmusokkal a "motorból" (`GameAlgorithms`) a kérdések léptetése, Továbbmegy és Megáll funkció megvalósítása, valamint a szakaszkérdesek figyelése.

Részletes bemutatás a 4.3 és az 5.1 alfejezetekben.

- Kártyapakli:
 - A kártyapakli készítés során, a GUI-val kapcsolatban felmerülő felhasználói követelmények megvalósításának egy része szintén a Java Swing segítségével.

- Az összes kattintáshoz köthető tevékenység megvalósítása a JButton gombokhoz kapcsolódó `actionPerformed` metódusok segítségével.
- A kártyapakli készítés során az összes funkció és azok algoritmusai, metódusai a DBM osztályban legyenek definiálva.
- Kártyalapok mentése az adatbázisba pakli ID és kártyalap név alapján, egyszerű SQL `Insert` parancsal, amelybe a lapok minden komponens értéke belekerül.
- Kártyalap szerkesztési és másolási lehetősége SQL `Insert` parancsal.
- Kártyapakli nyomtatási lehetősége PDF fájlba azáltal, hogy egy SQL `Select` segítségével minden adat visszatöltődik a kártya készítéshez, és egy gomb lenyomásával lehet indítani a nyomtatást, azaz a PDF-be írást `PdfWriter` segítségével.

Részletes bemutatás a 4.4 és az 5.2 alfejezetekben.

A továbbiakban a **nem funkcionális követelményeket**, azaz az egész rendszerre vonatkozó követelményeket részletezem.

Felhasználhatóság: minden funkció egyértelmű a feliratok alapján; átlátható a rendszer.

Hordozhatóság: Bármilyen személyi számítógépen indítható az alkalmazás, amelyen található JRE.

Hatékonyúság: 50-200 MB közötti memóriaigénye legyen maximum, azáltal biztosítható a rövid válaszidő.

Környezeti követelmények: SQL alapú adatbázis-szerver használata az adatok tárolására, amely egy SSD alapú virtuális szerveren helyezkedik el, ezáltal szükséges az internetkapcsolat a játék használatához.

Működési követelmények: Működésének zökkenőmentesnek kell lennie átlagos gyakoriságú használat esetén, de akár viszonylag sűrű használat esetén is, rövid illetve valamivel hosszabb futási időben is.

Fejlesztési követelmények: Az egész alkalmazás megvalósítható Java nyelven, NetBeans környezetben Java Maven verziókezelő, valamint Java Swing támogatással, objektum orientált módon.

2.3.3. Java, mint objektum orientált nyelv

A Java egy általános célú, teljesen objektum orientált programozási nyelv. Ezt a munkám során úgy tudtam kihasználni, hogy törekedtem minden objektum orientált programozási alapelveit megvalósítani.

Először is semmilyen módon nem történt osztályon kívüli deklaráció, valamint azt is kihasználtam, hogy az osztály saját névteret alkot, azaz az *egységezés alapelve* megvalósult.

Az *adatrejtés alapelve* szintén megvalósult, olyan módon, hogy az osztályok adattagjai, valamint túlnyomó többségben a metódusok `private` hozzáférési módosítóval rendelkeznek.

A *polimorfizmus alapelve* oly módon valósult meg, hogy egy osztályban történt metódus túlterhelés (*Method Overloading*), ugyanis szükséges volt két ugyanolyan eljárás definiálása, csak más paraméter szignatúrával.

2.3.4. Apache Maven Project

Az Apache Maven egy szoftver menedzsment eszköz, amely segítségével egy Java alapú projektet könnyebben lehet kezelni. A legnagyobb előnye a build-folyamat automatizálása mellett, a projekt függőségeinek kezelése.

A projekt függőség kezelése azt jelenti, hogy a programozás során külső könyvtárat szükséges használnunk, azaz külső könyvtáraktól függ a projektünk. Ezek a külső library-k Java nyelv esetén `jar` fájlok. A Maven a `pom.xml` fájl segítségével automatikusan kezeli ezeket a függőségeket, ezáltal nem szükséges külön a `jar` fájlokat letöltenünk és hozzáadnunk a projektünkhez.

A Maven alapja a projektobjektum-modell, azaz a POM (*Project Object Model*). Ez egy XML fájl, amely tartalmazza a projekt menedzseléséhez szükséges beállításokat, konfigurációs információkat. Ebben a fájlból adhatjuk meg a szükséges függőségeket, valamint, hogy a Maven melyik könyvtár tárolókban (*Maven repository*) keresse azokat.

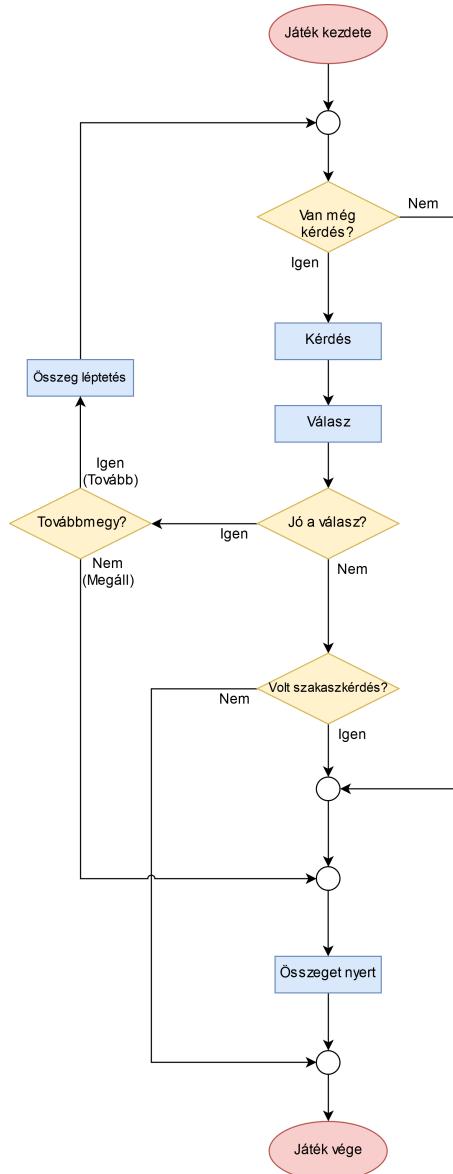
A `pom.xml` fájlról részletesebben írok a 4.5 alfejezetben.

Az Apache Maven projekt menedzsselő eszközzel már korábban találkoztam, azonban a pontosabb ismeretekért közvetlen az Apache Maven [11] weboldalán böngésztem, valamint a [12] [13] jegyzeteket is felhasználtam.

3. fejezet

Tervezés

3.1. Folyamatábra

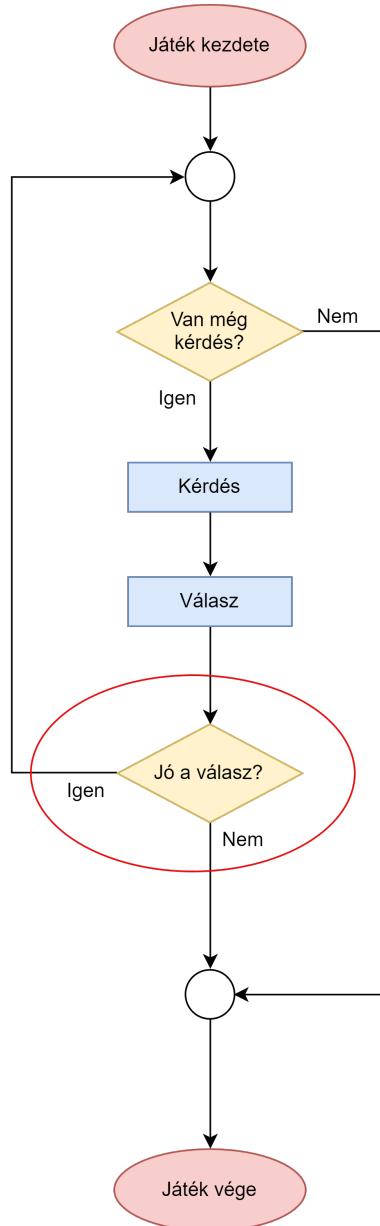


3.1. ábra. A program folyamatábrája.

3.1.1. Folyamatábra felépülése implementálás közben

A program megvalósítása közben természetesen lépésről lépére haladtam minden funkció megvalósításával. Itt most csak azokra a funkciókra gondolok, amelyek a folyamatábrán is látszódnak. Tehát ezalatt a *Jó válasz vizsgálatra*, a *Továbbmegy vagy megáll lehetőségre*, a *Szakaszkérdezések figyelésére*, és a *Nyeremény összegek léptetésére* gondolok.

1. lépés: Legelsőnek a legegyértelműbb és legfontosabb részt valósítottam meg, amely nem más, mint a *Jó kérdés vizsgálat*. Azaz, hogy a program olyan állapotban legyen, hogy sorban, egymás után érkezzenek a kérdések, és lehetőség legyen a 4 válasz-lehetőség közül választani, és ezt a program a választás után helyesen vizsgálja, hogy jó választ jelölt-e a Játékos vagy sem.

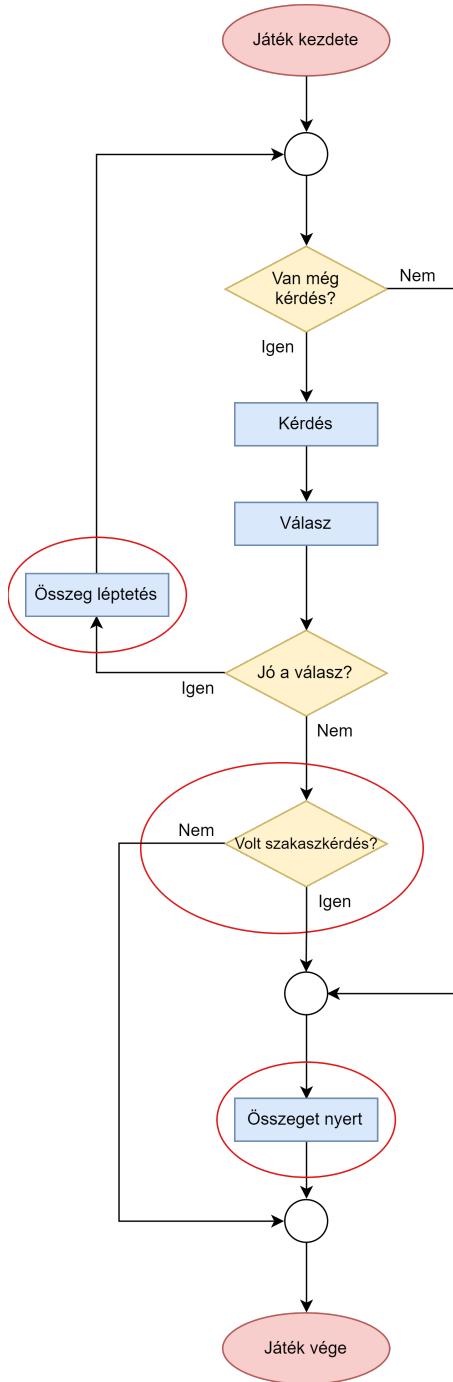


3.2. ábra. Folyamatábra 1. lépése.

2. lépés: A kérdések helyes vizsgálata után, a *Szakaszkérdések jelölése* és egyben a *Nyeremény összegek léptetése* volt a következő lépés.

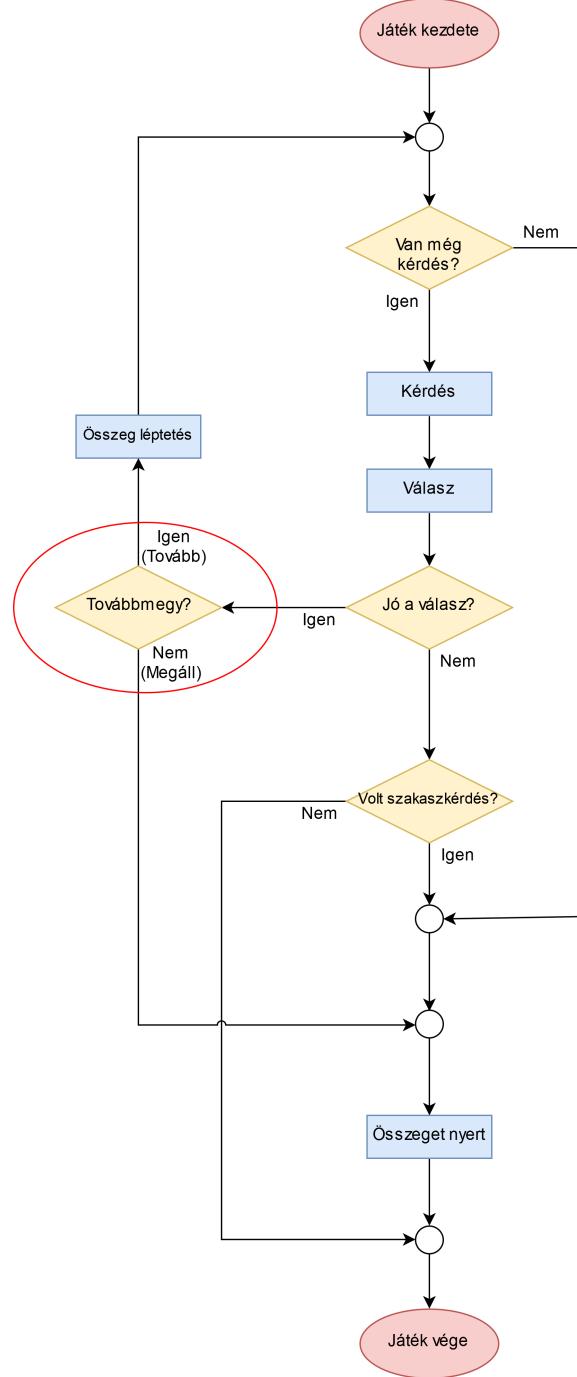
A szakaszkérdések jelölésével a Játékos, ha túl jut az 5. kérdésen, akkor minden esetben kap garantált nyereményösszeget.

Mindezek mellett pedig a Játékosnak folyamatosan növekszik a megnyerhető összeg nagysága, az összegek léptetésével – amelyet egyébként a játék felületén is lát, miközben játszik, hogy éppen melyik kérdésnél és milyen összegnél tart.



3.3. ábra. Folyamatábra 2. lépése.

3. lépés: Utolsó lépésnek hagytam azt, amiről úgy véltem, hogy a legbonyolultabb lesz a megvalósítása, amely a *Továbbmegy, vagy megáll* lehetőség, és ezáltal el is készült a program teljes folyamatábrája.



3.4. ábra. Folyamatábra 3. lépése.

A diagramokat a Diagram Editor¹ online weboldal segítségével készítettem el. Ezt a diagram szerkesztő ingyenes oldalt a korábbi féléveim során is használtam, mert nagyon áttekinthető és egyszerűen kezelhető a felület. Akár ki is lehet választani, hogy milyen fajta diagramot szeretnénk készíteni (pl. UML, ER diagram), de természetesen szinte bármilyen formát, alakzatot lehetőség van beilleszteni.

¹<https://www.diagrameditor.com/>

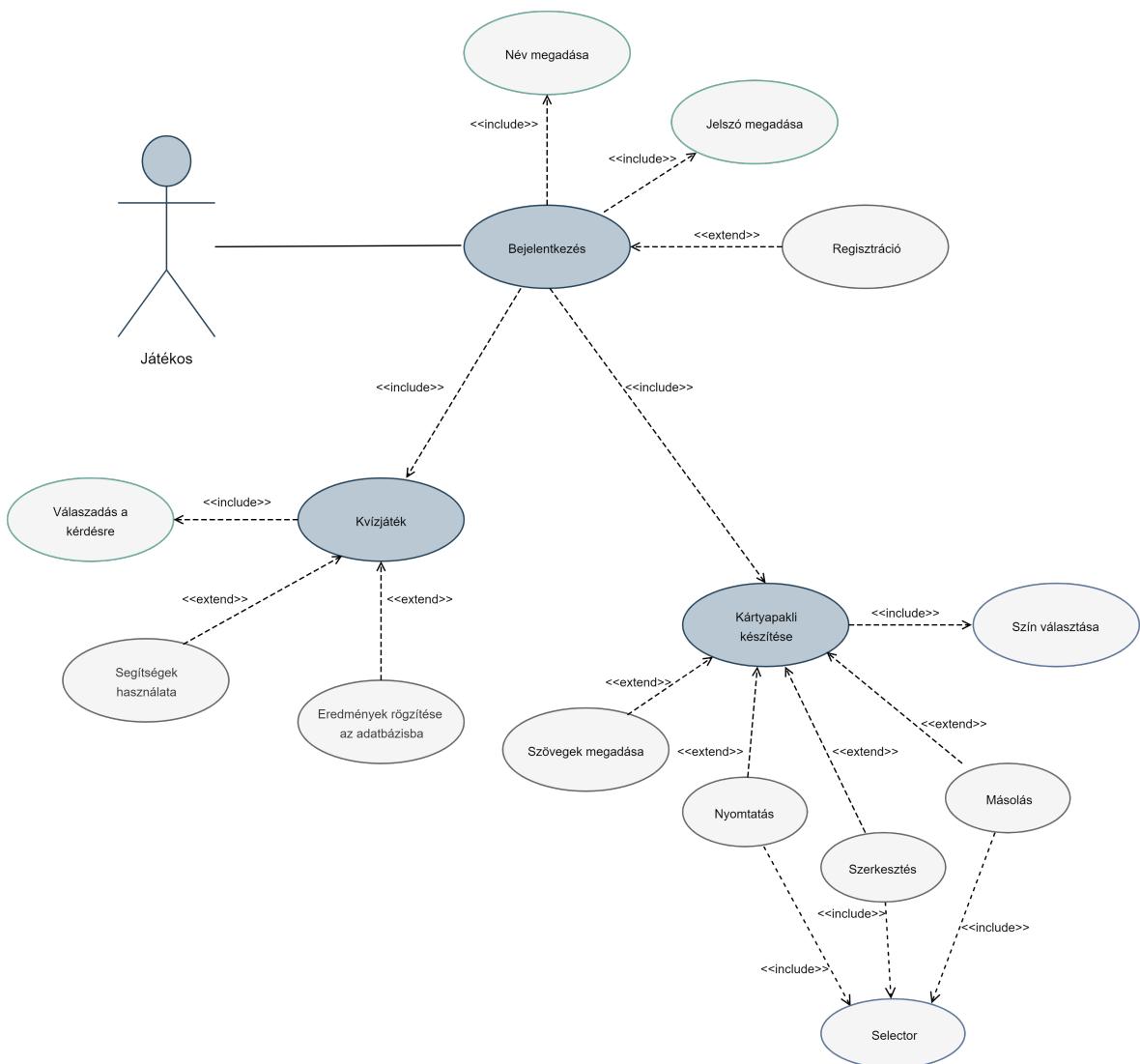
3.2. UML ábra

Az UML egy általános célú modellező nyelv, amely segítségével különböző diagramokat lehet megkotni, ezáltal szinte bármilyen programrendszer modelljét vizualizálni lehet. Habár ez egy grafikus nyelv, mégis vannak benne szintaktikai megkötések.

Az UML 2.0 13 különböző diagramtípust definiál, amelyeket modellezési fajta szerint két fő csoportra lehet osztani: *Struktúramodellezésre* (6 diagramtípus) és *Viselkedésmodellezésre* (3 diagramtípus), valamint a viselkedésmodellezésben belül egy alcsoport szerepel, a *Kölcsönhatás modellezése* (4 diagramtípus).

A következőkben a use-case, azaz a használati eset diagramot valósítottam meg.

3.2.1. Use-case diagram



3.5. ábra. Használati eset diagram.

Fentebb látható a program használati eset diagramja. Hárrom alapeleme van a use-case diagramnak: *használati eset (use-case)*, *aktor* valamint a *kapcsolat*. A *use-case* a felhasználó által látható funkció, az *aktor* a személy, aki a rendszerrel kapcsolatba kerül

(például a Játékos), a *kapcsolat* pedig a *use-case* és az *aktor* közötti viszonyrendszert határozza meg.

Használati esetek között háromféle kapcsolat értelmezhető: **include**, **extend** és **általánosítás** (öröklődés). Esetben csak az első két említett kapcsolat típus volt szükséges.

Az **include**, mint tartalmazás, nem más, mint amikor egy adott *use-case* végrehajtásakor, egy másik *use-case* feltétel nélkül minden szintén végrehajtódik. Erre a legjobb példa az ábrán látható "Bejelentkezés" használati eset, amelyből mint látható, négy **include** viszony is kiindul. Először is az ábrán felfelé elhelyezkedő kettőről beszélnek. Ezek minden azt reprezentálják, hogy a "Bejelentkezés" feltétlenül tartalmazza a "Név megadását" valamint a "Jelszó megadását", ahogyan ez valójában is így van. Ugyanis, ha bárhol be kell jelentkezni egy Felhasználónak, általában ahhoz egy felhasználónév, és a jelszó megadása szükséges. Az **include** kapcsolat jelölése minden szaggatott vonallal történik, a tartalmazó használati esettől a tartalmazott felé, valamint a nyílon fel kell tüntetni az «**include**» sztereotípiát.

A "Regisztráció", mint kibővítés (**extend**) azt reprezentálja, hogy a bővítő használati eset az alap funkciót kiegészíti, vagy annak valamilyen kivételes esete. Azaz a "Regisztráció" azért kibővítése a Bejelentkezésnek, hiszen ha egy Felhasználó nem tud belépni, mert nincsen még felhasználói fiókja, akkor lehetősége van "Regisztrálni", amely után már képes lesz bejelentkezni. Jelölése minden a kiegészítő *use-case* felől az alap funkció felé mutató szaggatott nyíllal történik, amelyen szerepelni kell az «**extend**» sztereotípiának.

A "Kvízjáték" és a "Kártyapakli készítés" azért a "Bejelentkezés" használati eset tartalmazottjai, mivel a két *use-case* csak akkor következhet be, ha a "Bejelentkezés" megtörtént.

A "Kvízjáték" tartalmazója a "Válaszadás a kérdésre" használati esetnek, hiszen a Játékosnak mindenkorban választ kell adni a kérdésre. A "Segítségek használata" és az "Eredmények rögzítése az adatbázisba" *use-case*-ek csak kiegészítői a "Kvízjáték"-nak, ugyanis a segítségek használata nem kötelező, valamint az eredmények rögzítése csak abban az esetben történik meg, hogyha a Játékos helytelenül válaszol, megáll, vagy éppen megnyerte a főnyereményt.

A "Kártyapakli készítése" használati esetnek négy kiegészítő funkciója is van, amelyek nem következnek be mindenkorban, csak ha a Játékos úgy dönt, hogy használja ezeket. Ugyanis nem kötelező a "Szövegek megadása" a kártyáknak, azonban a "Szín kiválasztása" igen, ezért is tartalmazza ezt a használati esetet a "Kártyapakli készítés".

A "Nyomtatás", "Szerkesztés" és "Másolás" kiegészítő funkciói a "Kártyapakli készítésnek", azonban mindenkorban használati esethez szükséges a "Selector" felület, ahol a Játékosnak ki kell kiválasztani a nyomtatni, másolni, vagy szerkeszteni kívánt kártyapaklit, és annak kártyalapját, ezért is mindenkorban tartalmazottja a "Selector" *use-case*.

A *use-case* diagramot szintén a Diagram Editor² online weboldal segítségével valósítottam meg, valamint az elméleti hátteret hozzá a [8] Szoftvertechnológiák tárgy órai jegyzetéből merítettem.

²<https://www.diagrameditor.com/>

4. fejezet

Megvalósítás

4.1. Az adatbázis

Annak érdekében, hogy az adatbázisban lévő adatok perzisztens módon legyenek tárolva, úgy döntöttem, hogy a legjobb megoldás bérálni egy kis teljesítményű virtuális szervert. Virtuális szerverem elhelyezésére a VPS4You¹, magyar szerver szolgáltatót választottam, hosszas keresés után. Egy SSD alapú **VPS** (*Virtual Private Server = Virtuális dedikált szerver*) szolgáltatást választottam, 512 Mb RAM-al és 5 Gb SSD tárhellyel.

A dedikált IP-címmel rendelkező virtuális szerverre az Ubuntu Linux olyan disztribúciója van telepítve, amelyhez nem tartozik felhasználói felület (UI), csak karakteres. Ezen operációs rendszeren fut a MySQL adatbázis szerver, amelyen a programom futása szempontjából szükséges adatok vannak.

A Putty² program segítségével, SSH kapcsolaton keresztül telepítettem fel a MySQL szervert a VPS-re, a `sudo apt install mysql-server` parancssal. A telepítés után beléptem a virtuális gépen a MySQL szerverbe a `sudo mysql -u root -p` parancssal, majd létrehoztam egy felhasználót a következőképpen:

```
create user 'treka'@'%' identified by 'szakdoga2021';
grant all privileges on * . * to 'treka'@'%';
flush privileges;
```

Ezen a terminál felületen is lehetne SQL parancsokkal adatokat hozzáadni az adatbázishoz, de HeidiSQL³ adatbázis kezelő alkalmazást használtam, hiszen ez egy ingyenes és nyílt forráskódú szoftver.

Az egész játékhoz alapvetően négy táblára volt szükség az adatbázisban: `users`, `cards`, `question` és a `ranking_table`.

¹<https://vps4you.hu/>

²<https://www.putty.org/>

³<https://www.heidisql.com/download.php>

4.1.1. Relációs adatmodell

Itt láthatóak az adatbázisban található táblák és azok tulajdonságai. A táblák között nincsen kapcsolat, azonban szemléltetésképpen létrehoztam az adatbázisról a relációs sémamodellt.

users <table border="1"> <thead> <tr> <th>PK</th> <th><u>id</u></th> </tr> </thead> <tbody> <tr> <td></td> <td>username</td> </tr> <tr> <td></td> <td>password</td> </tr> <tr> <td></td> <td>Name</td> </tr> <tr> <td></td> <td>faculty</td> </tr> <tr> <td></td> <td>RegDate</td> </tr> </tbody> </table>	PK	<u>id</u>		username		password		Name		faculty		RegDate	cards <table border="1"> <thead> <tr> <th>PK</th> <th><u>id</u></th> </tr> </thead> <tbody> <tr> <td></td> <td>deckID</td> </tr> <tr> <td></td> <td>colour</td> </tr> <tr> <td></td> <td>questionText</td> </tr> <tr> <td></td> <td>questionMath</td> </tr> <tr> <td></td> <td>resultText</td> </tr> <tr> <td></td> <td>resultMath</td> </tr> <tr> <td></td> <td>cardname</td> </tr> </tbody> </table>	PK	<u>id</u>		deckID		colour		questionText		questionMath		resultText		resultMath		cardname	question <table border="1"> <thead> <tr> <th>PK</th> <th><u>id</u></th> </tr> </thead> <tbody> <tr> <td></td> <td>question</td> </tr> <tr> <td></td> <td>qmath</td> </tr> <tr> <td></td> <td>sol1</td> </tr> <tr> <td></td> <td>sol2</td> </tr> <tr> <td></td> <td>sol3</td> </tr> <tr> <td></td> <td>sol4</td> </tr> <tr> <td></td> <td>answer</td> </tr> <tr> <td></td> <td>category</td> </tr> </tbody> </table>	PK	<u>id</u>		question		qmath		sol1		sol2		sol3		sol4		answer		category	ranking_table <table border="1"> <thead> <tr> <th>PK</th> <th><u>id</u></th> </tr> </thead> <tbody> <tr> <td></td> <td>username</td> </tr> <tr> <td></td> <td>point</td> </tr> </tbody> </table>	PK	<u>id</u>		username		point
PK	<u>id</u>																																																						
	username																																																						
	password																																																						
	Name																																																						
	faculty																																																						
	RegDate																																																						
PK	<u>id</u>																																																						
	deckID																																																						
	colour																																																						
	questionText																																																						
	questionMath																																																						
	resultText																																																						
	resultMath																																																						
	cardname																																																						
PK	<u>id</u>																																																						
	question																																																						
	qmath																																																						
	sol1																																																						
	sol2																																																						
	sol3																																																						
	sol4																																																						
	answer																																																						
	category																																																						
PK	<u>id</u>																																																						
	username																																																						
	point																																																						
(a) users tábla.	(b) cards tábla.	(c) question tábla.	(d) ranking_table tábla.																																																				

4.1. ábra. Relációs sémamodell.

4.1.2. Adatbázis-kapcsolat létrehozása

Java programozási nyelvben a JDBC (*Java Database Connectivity*) API segítségével lehetséges az adatbázis-kapcsolatok kezelése. A JDBC segítségével lehet létrehozni a kapcsolatot az adatforrással (például adatbázissal), SQL utasításokat készíthetünk, azaz lekérdezéseket és módosítási műveleteket hajthatunk végre, valamint az adatforrástól kapott adatokat fel tudjuk dolgozni.

A következő metódussal jön létre a kapcsolat az adatbázis és a program között.

```

1 import java.sql.Connection;
2 import java.sql.DriverManager;
3 import java.sql.SQLException;
4
5 public void GameCon() {
6     try {
7         con = DriverManager.getConnection("jdbc:mysql
8             ://87.229.115.62:3306/cardgame?serverTimezone=UTC", "9
9             treka", "szakdoga2021");
10        dbm.MS("Sikeres csatlakozas a szerverre");
11    } catch (SQLException e) {
12        dbm.MS("Sikertelen csatlakozas a szerverre");
13    }
14 }
```

Ennek a metódusnak a megírásához, a felette látható import-ok voltak szükségesek.

A `java.sql.Connection` egy interfész osztály, amely az adatbázissal kiépített kapcsolatot képviseli.

A `java.sql.DriverManager` egy meghajtókezelő osztály, és ez az osztály határozza meg, hogy mely objektumok kapcsolhatják össze a Java alkalmazást a JDBC meghajtóval. Ez a meghatározás egy URL-en keresztül történik, amelyben a protokollt és az

alprotokollt kell megadni. A `DriverManager` osztály statikus `getConnection` metódusának kell paraméterként átadni ezt az URL-t. Ennél a kódnál minden a 7. sorban történik meg.

A `java.sql.SQLException` pedig nem más, mint egy kivételeosztály, és ahogy a nevéről is adódik, az adatbázis-műveletek elvégzése során bekövetkező hibákat reprezentálja. Kivételek megdobódása során használható lenne csak maga az `Exception` osztály, mint az összes `Exception` ōse, viszont technikailag a legszűkebb kivételre törekedtem.

A 8. és 10. sorban található `dbm` osztály `MS("")`; függvénye, egy általam definiált metódus, amely egy egyszerű párbeszédablakot hoz létre, és a függvénybe beleírt szöveget jeleníti meg az ablak.

4.2. A két játékrész közös osztályai – signinup csomag

A Legyen Ön is milliomos kvíz és a kártyapakli készítő játékrész közös kódrésze nem más, mint a Bejelentkezés és Regisztráció felülete. Ez a `signinup` csomagban található, a csomagban pedig három osztály van: a `SignupDBM`, a `Signinup` és a `Signup`.

A három osztályból a `Signinup` és a `Signup` `JFrame` Form, amely a `javax.swing` csomag, `JFrame` osztályának, azaz a `javax.swing.JFrame`-nek a leszármazottja. A `SignupDBM` pedig Java osztály. A lényegesebb kódrész a `SignupDBM` osztályban van, ugyanis a másik két osztály a grafikus felület létrehozására szolgál.

4.2.1. Signinup és Signup osztály

Ahogyan már említettem, a `Signinup` és a `Signup` osztályok `JFrame` Form-ok. Azaz ezekben az osztályokban nincs a program szempontjából kulcsfontosságú kód és algoritmus, csak a kinézet kialakítása történt itt meg. Mindahhoz, hogy a GUI-t megalósítsam a `javax.swing` csomagra volt szükségem, hiszen ebben a csomagban található minden Java Swing osztály és komponens, amelyekre szükségem volt.

A NetBeans IDE-ben a `JFrame` Form-oknál a "Source" és a "Design" fülek között lehet váltani: a "Source" fülön a kódot látjuk, a "Design" fülön pedig a grafikus felhasználói felület kinézetét, amelyet létrehoztunk. Itt lehet a különböző komponenseket hozzáadni a kinézethez. Az általam legtöbbet használt komponensek a `JPanel`-ek, a `JLabel`-ök, azaz a címkek, a `JTextField`-ek, azaz a szövegmezők, valamint a `JButton`-ök, vagyis a gombok.

Mind a `Signinup`, és mind a `Signup` GUI-jához paneleket, címeket, szövegmezőket, gombokat, valamint jelszó szövegmezőt (`JPasswordField`) használtam.

A `JPasswordField`, mint jelszó szövegmező szerepe, hogy elfedje a tényleges jelszóban szereplő karaktert (• karakterrel), azt azonban jelzi, ha gépelés történik.

A kinézetet az 5.1 A játék menete alfejezetben tárgyalom ábrákkal együtt.

Ennél a két osztálynál a kódban csupán `actionPerformed` metódusok vannak, amelyek a gombokon történő kattintás eseményt kezelik. Ebben a két osztályban a nyomógombok `actionPerformed` eljárásában a következő metódusok hívódnak meg: *Regisztráció* esetén az általam definiált `UserInsert()` eljárás hívódik meg, amelyben egy egyszerű SQL `Insert` történik; *Belépés* esetén pedig a – szintén általam definiált – `LogIn()`, amelyben pedig egy SQL-lekérdezés történik.

Ezen metódusok mellett pedig a `javax.swing.JFrame` osztály `setVisible` metódusát használtam még, a panelek közti átlépéshez.

4.2.2. SignupDBM osztály

A `signinup` csomagban a `SignupDBM` osztály az egyetlen Java osztály, tehát itt találhatóak a program szempontjából releváns kódok. A következőkben két fontosnak vélt metódust fogok bemutatni hosszabban, a kódjaikkal együtt, azonban ezeken kívül is vannak még eljárások az osztályban, amelyeket itt mutatnék be néhány mondatban.

A `UsernameValidator()` metódus beszédes névvel rendelkezik, melyből a funkciója is kiderül: a felhasználónevek ellenőrzésére szolgál. Az adatbázisból egy SQL `Select` segítségével lekérdezi a felhasználóneveket, majd egy `try-catch` blokkba zárva egy `ArrayList`-be helyezi azokat. Ezt az `ArrayList`-et később más eljárások is felhasználják. A `try-catch` blokkra azért van szükség, hiszen SQL-lekérdezés történik, ezért `SQLException` történhet, amelyet a catch ágban le kell kezelní.

A `LoginValidator()` szintén egy SQL `Select` parancssal lekérdezi az adatbázisból a felhasználónevet, valamint a jelszót, amelyeket ezután osztály szintű `private` változókba helyez el, azok `setter` metódusainak segítségével. Erre azért van szükség, mert ezt az eljárást a később tárgyal (4.2.3 alfejezet) `LogIn()` metódus felhasználja.

A `UserInsert()` eljárás szintén beszédes nevéről kiderül, hogy ez a metódus valósítja meg a felhasználók felvitelét az adatbázisba. Egy SQL `Insert` parancssal történik ez meg. Mivel ez a metódus a Regisztráció során hívódik meg, az `Insert`-be a Felhasználónév, Jelszó, Teljes név, Kar kerül, amit a Játékos adott meg, valamint még egy *Regisztrációs dátum* is (év-hónap-nap formátumban), a `LocalDateTime()` segítségével. Az eljárásban meghívásra kerül a `UsernameValidator()`, amely átadja az `ArrayList`-ben tárolt felhasználóneveket. Ennek elemei `String` összehasonlításra kerülnek a Játékos által megadott felhasználónévvel, és ha egyeznek, a Játékos üzenetet kap egy párbeszédablakban, hogy a felhasználónév már foglalt, adjon meg újat.

4.2.3. SignupDBM osztály – LogIn()

A `LogIn()` metódus a bejelentkezés során hívódik meg.

Először a szerverre csatlakozás megy végbe, a már fentebb említett módon (4.1.2 alfejezet), a `Con2()` metódussal, amelyre szükség van, mivel ezután a `LoginValidator()` meghívása történik. Ez egy kvázi ellenőrző függvény, amely a bejelentkezéskor beírt adatokat ellenőrzi, az adatbázisban tároltakkal összevetve, mindezt egy SQL `Select` parancssal lekérve.

A `getText()` eljárás a `javax.swing.JTextField` osztály része, amely segítségével a szövegmezők tartalmát kapjuk meg – ahogyan ez az angol fordításból is egyértelmű: "*get the tex*" = megkapni/elérni a szöveget. Ezen metódus segítségével tudja kezelni a program a Felhasználó által a szövegmezőbe begépelt szöveget.

Az 5. sorban látható még egy eljárás hívás (`Crypting()`), amely a jelszó titkosítására szolgál, erről a következő alfejezetben írok részletesen (4.2.4 alfejezet).

A szükséges metódusok meghívása, valamint a lokális változók deklarálása után egy egyszerű `String` összehasonlítás történik. Először a begépelt jelszó összehasonlítása megy végbe, a `LoginValidator()` által ellenőrzött, és helyesen az adatbázisban tárolt felhasználónévvel (8. sor). Ezen belül pedig a jelszó összehasonlítás történik (9. sor). Megfelelő és nem megfelelő adatok esetén is párbeszédablak ugrik fel a Felhasználó elé, hogy tudja, hogy sikeres volt-e a bejelentkezés, vagy sem (11., 15., 19. sor).

Az eljárás visszatérési értéke logikai érték, ugyanis deklarálva lett egy lokális logikai változó a metódus elején, amely értéke sikeres bejelentkezés esetén `true` lesz, sikertelen esetén pedig `false`.

```

1  public boolean LogIn(JTextField t1, JTextField t2) {
2      Con2();
3      LoginValidator(t1);
4      String username = t1.getText();
5      String pw = Crypting(t2.getText());
6      boolean correct = false;
7
8      if (username.equals(usern)) {
9
10         if (pw.equals(passw)) {
11             GA.setPlayername(username);
12             dbm.MS("Sikeres volt a bejelentkezes! \n Udvozollek a
13                 rendszerben: " + username + " !");
14             correct = true;
15         } else {
16             dbm.MS("Nem megfelelo a jelszo!");
17             correct = false;
18         }
19     } else {
20         dbm.MS("Nem megfelelo a felhasznalonev, \n vagy nincs
21             meg regisztralva!");
22         correct = false;
23     }
24     return correct;
}

```

4.2.4. SignupDBM osztály – Crypting()

A Crypting() metódus fő feladata a Játékos által megadott jelszó titkosítása, valamint a bejelentkezés során beírt jelszó átfordítása és átadása az összehasonlító eljárás számára.

Bemeneti paramétere egy `String`, amely az a karaktersorozat, amely az eljárás során titkosítva lesz – vagyis a Játékos által megadott jelszó. A metódus `SHA (Secure Hash Algorithm)` titkosítási rendszert alkalmaz.

Az `SHA` titkosítás a Hash függvények közé tartozik, melynek lényege, hogy bármilyen hosszságú adatot adott hosszságra képez le. Az ezáltal kapott adat neve hash (hasító érték). Az `SHA 160` bites hash értéket állít elő, amely 40 hexadecimális értékű karaktert eredményez, és 512 bites blokkokban dolgozza fel az üzeneteket.

A hash algoritmusban minden definiálva van, hogy mit kell kezdeni azokkal az esetekkel, amikor töredék csomaggal kellene dolgozni. Legtöbbször valamilyen rögzített értékkel (pl. `0x00` feltöltő byte) kiegészítés történik mindenkor, amíg egész csomag nem képződik. Alapvető elvárás, hogy a hash algoritmus csak a bemenő adat bájtjaitól és azok sorrendjétől függjön, tehát determinisztikus/előre meghatározott legyen.

Egy `byte` típusú tömbbe szétdaraboljuk a jelszót bájtakra, a `.getBytes()` eljárással (6. sor).

A `MessageDigest` típussal deklarált változó állítja be a kódolási algoritmus típusát (9. sor). A létrehozás után ki kell üríteni a változót (11. sor), majd pedig a `md` lokális változót fel kell tölteni a jelszó már bájtakra bontott értékével.

A **for**-cikluson belül történik a jelszó átalakítása, ahol az `encodedPassword` tömbön halad végig az iteráció. Ezen belül a kód akkor fut bele az `if` elágazásba, ha a tömbben tárolt érték `0xff` és `0x10` közé esik (17. sor). Ekkor a `StringBuilder` típusú változóhoz hozzáad egy "`0`" karaktert (18. sor). Ellenkező esetben az algoritmusnak megfelelő értéket tesz a `pw` változóba.

Amikor a metódus futása véget ért, visszatérési értékként adja a kódolt `pw` változót.

```
1 import java.security.MessageDigest;
2
3 public String Crypting(String passw) {
4     String pw = "";
5     String algorithm = "SHA";
6     byte[] plainText = passw.getBytes();
7
8     try {
9         MessageDigest md = MessageDigest.getInstance(algorithm);
10
11     md.reset();
12     md.update(plainText);
13     byte[] encodedPassword = md.digest();
14
15     StringBuilder sb = new StringBuilder();
16     for (int i = 0; i < encodedPassword.length; i++) {
17         if ((encodedPassword[i] & 0xff) < 0x10) {
18             sb.append("0");
19         }
20         pw = String.valueOf(sb.append(Long.toString(
21             encodedPassword[i] & 0xff, 16)));
22     } catch (Exception e) {
23         e.printStackTrace();
24     }
25     return pw;
26 }
```

4.3. Legyen Ön is milliomos! osztályai – game csomag

A Legyen Ön is milliomos kvízjáték osztályai a game csomagban helyezkednek el. Ebben a csomagban öt osztály található: a GameAlgorithms, a PreGame, a Game, a BarChart és a Ranking. Az öt osztályból egyedül a GameAlgorithms osztály az, ami Java osztály, a másik négy pedig JFrame Form.

4.3.1. PreGame és Game osztály

Ahogy feljebb írtam, a PreGame és a Game osztályok JFrame Form-ok, amelyekkel a kinézet kialakítását lehet megvalósítani. Természetesen itt is a javax.swing csomagra volt szükségem, mert ebben található meg minden Java Swing osztály és komponens, amelyeket felhasználtam.

A PreGame osztályban szinte csak JLabel-ök találhatóak, mindezek persze egy JPanel-en, valamint ezek mellett egy JButton is, amely a játék indítására szolgál, és egy JMenuBar, azaz egy menüsor.

A Game osztályban JPanel-ek, JLabel-ök, JButton-ök, valamint egy JList is található, és JMenuBar is.

A pontos kinézetek majd a későbbiekben láthatóak (5.1 alfejezet).

4.3.2. Ranking és BarChart osztály

Erre a két, szintén JFrame Form osztályra két funkció megvalósításához volt szükségem.

A Ranking nem más, mint a Toplista megvalósítása. Az ehhez tartozó eljárásról a következő alfejezetben írok (4.3.3 alfejezet). Az 5.1 A játék menete alfejezetben az elhelyezkedése is látható (5.5 ábra), valamint az 5.3.1 Tesztelés ismerősökkel alfejezetben a JFrame Form kinézete is megtalálható (5.29 ábra).

A BarChart a Közönség segítségéhez tartozó diagram megjelenítéséhez volt szükséges. Ez szintén egy JFrame Form, amely csupán egy JPanel-ből, két JButton gombból, és egy JLabel-ből áll. Ebben a JLabel-ben jelenik meg a ChartMaker()-ben generálódott oszlopdiagram képe. Az 4.3.9 GameAlgorithms osztály – playerhelp2randomizer() alfejezetben írok hosszabban a segítségről és annak kódjáról, valamint 5.1.2 Játékszáolyok alfejezetben látható a JFrame Form kinézete (5.10 ábra).

4.3.3. GameAlgorithms osztály

A következőkben a GameAlgorithms osztály fontosabb, érdekesebb metódusait fogom bemutatni. Természetesen ezeken kívül még több eljárás található az osztályban, azonban úgy gondoltam, hogy csak az érdekfeszítőbbeket és bonyolultabbakat mutatom be részletesen, a kódokkal együtt, külön alfejezetben. Azokról a metódusokról említenék itt néhány mondatot, amelyeket úgymond "kihagyok" a hosszasabb részletezésből.

A getColumn() metódusban történik a három kérdésnéhézségi kategória szerinti ID kiválogatás az adatbázisból, SQL Select lekérdezésekkel. Ehhez három ArrayList volt szükséges, amelyekben a kiválogatott ID-k tárolódnak. Például a randcat1 nevű ArrayList-ben tárolódnak az 1-es nehézségűnek kategorizált kérdések ID-i.

A TestMaker() eljárásban valósul meg a játék felületére a kérdés, valamint a gombokra a válaszok kiíratása. Mindehhez a legfontosabb metódus a setText() a

`javax.swing.JTextField` osztályból, amely segítségével szabályozhatjuk, beállíthatjuk a változó értékét (*"set the text"* = beállítani a szöveget).

A `checkpointcheck()` állítja be, hogy a Játékos melyik szakaszkérdést hagyta már el (5. vagy 10.), hiszen a szakaszkérdések után a szakaszkérdés összege garantált nyeremény a Játékos számára, helytelen válasz esetén is.

A `pausegame()` metódusban valósul meg a Megáll vagy Továbbmegy döntés kezelése. Ha a Játékos továbbmegy, akkor egy logikai változó `true`-ra vált, és a játék folytatódik a következő kérdéssel. Ha viszont a Játékos megáll, akkor az aktuális összeget megnyerte, ez pedig lementésre kerül az adatbázisba a `saveplayermoney()` segítségével.

A `printRandomNumbers()` metódus egy `int` tömbtípusú eljárás, amely két értéket kér be: egy iterációs változót, illetve egy maximális értéket. Ennek segítségével randomizál a kért intervallumok között, mégpedig olyan módon, hogy abban az esetben, ha már a kért intervallumban szereplő szám megtalálható a tömbben, akkor egy új számot randomizál az adott értékhatarok között. Ezután egy `integer` típusú tömbbel tér vissza. Ez a metódus a felező segítségnél játszik szerepet, hiszen ennek köszönhetően kis intervallum esetén (itt $I = [0, 2]$, azaz csupán 3 elem) is képes lesz a kód az ismétlés nélküli randomizálásra.

A `ChartMaker()` készíti el magát a diagram kinézetét a Közönség segítsége statisztiákhoz. Ehhez a `org.jfree.chart.JFreeChart` függvény könyvtárra volt szükség, hiszen ebben található meg minden olyan osztály, amely bármilyen diagram készítéséhez szükséges. Ehhez a diagram elkészítéséhez `barChart`-ra volt szükségem, azaz oszlopdiagramra. Az oszlopdiagramot egy `.jpeg` file-ba generálja a metódus, amelyben lokális változókkal deklaráltam a kép szélességét és magasságát. Későbbiekben ezt az eljárást fogja meghívni a `playerhelp2randomizer`, azaz a Közönség segítsége metódus.

A `saveplayermoney()` mondhatni egy egyszerű SQL `Insert`. A metódus először is `if()` segítségével azt az esetet vizsgálja meg, hogy a Játékos helytelen választ adott. Azaz ekkor megnézi, hogy melyik `checkpoint`-ok igaz értékűek (5. vagy 10.), ha valamelyik igaz volt, akkor az ehhez tartozó összeg a Játékos nyereménye, azaz ez kerül mentésre az adatbázisba, egy SQL `Insert` parancs segítségével. Az `else` ágban pedig a Játékos ugyebár nem veszített, azaz Megállt, ezért ekkor az aktuális összeg kerül fel az adatbázisba, szintén az SQL parancs segítségével.

A `ranking()` szintén egy SQL-lekérdezés. Egy `Select` parancs segítségével az adatbázisból lekérésre kerülnek az első tíz legmagasabb összeget megnyert Játékosok felhasználónevei, valamint a megnyert összegük.

A `endgame()` metódusra a játék befejezéséhez volt szükség. Ha a Játékos eléri a főnyereményhez tartozó 15. kérdést, valamint azt helyesen is válaszolja meg, akkor az aktuális nyereménye 50 000 000 Forint. Az eljárás ezt figyeli, és ennek bekövetkeztekor egy párbeszédablak ugrik fel, mentésre kerül az összeg, valamint megjelenik a "Játék vége" gomb.

A `playerhelp1()` nem más, mint a Felező segítség. Itt a négy gomb közül, azaz a négy válaszlehetőség közül kettő rossz válasznak el kell tűnnie. A metódus először kiválasztja, hogy melyik a helyes válasz, majd ezután a másik három közül számrandomizálással – az általam megírt `printRandomNumbers()` eljárás segítségével – kettőt kiválaszt, amelyek ezután a már korábban említett `setVisible()` eljárás `false` értékre állításával eltűnnek – majd miután a Játékos kiválasztotta az általa jónak vélt választ, újra láthatók lesznek, `true` értékkel.

A `playerhelp3()` metódus a Telefonos segítség. Ebben igazából semmilyen bonyo-

lultság nincs: a kód megvizsgálja, hogy melyik gomb a helyes válasz, azaz az ABCD lehetőségek közül melyik a helyes válasz, majd egy egyszerű párbeszédablakban megjelenik a telefonbeszélgetés szövege, amelyben kiderül a helyes válasz betűje.

A GameAlgorithms osztály legelején találhatóak a szükséges import-ok, az adattípusok megfelelően a private változók, és ezek getter és setter metódusai.

4.3.4. GameAlgorithms osztály – Randomizer()

A Randomizer() metódus futása kezdetén meghívja a getColumn() eljárást – amelyről a rövid bemutatás az előző alfejezetben található (4.3.3 alfejezet) – mely után három integer típusú tömb deklarálása szükséges, azonos mérettel. Ezen tömbök szerepe a kérdések sorszámnak randomizálása szempontjából releváns. Azaz ezekbe a tömbökbe fognak kerülni az adott kategóriába tartozó kérdések sorszámai, minden három kategóriába pontosan 5, hiszen összesen 15 kérdés van, és három nehézségi szint.

Az első for ciklusban történik az intervallumok közti véletlen szám generálás. A kapott értékeknek a megfelelő tömbbe kell kerülniük. Amint ez megtörtént, a metódus három darab for ciklust használ annak érdekében, hogy a három kategóriába sorolt kérdések kategóriájuknak megfelelő sorrendben helyezkedjenek el az ArrayList-ben.

Az eljárásnak nem indokolt a visszatérési érték, hiszen a három tömb adatát egy ArrayList-be vezeti át, amelyet az egész játék során használni fognak különböző metódusok.

```

1  private ArrayList<Integer> randcat1 = new ArrayList<Integer>();
2  private ArrayList<Integer> randcat2 = new ArrayList<Integer>();
3  private ArrayList<Integer> randcat3 = new ArrayList<Integer>();
4  private ArrayList<Integer> questionnum = new ArrayList<Integer>
    >();
5
6  private void Randomizer() {
7      getColumn();
8      Random rand = new Random();
9      int[] questioncategory1 = new int[5];
10     int[] questioncategory2 = new int[5];
11     int[] questioncategory3 = new int[5];
12
13     for (int i = 0; i < 5; i++) {
14         questioncategory1[i] = rand.nextInt(0 + randcat1.size()) - 0;
15         questioncategory2[i] = rand.nextInt(0 + randcat2.size()) - 0;
16         questioncategory3[i] = rand.nextInt(0 + randcat3.size()) - 0;
17     }
18     for (int i = 0; i < 5; i++) {
19         questionnum.add(randcat1.get(questioncategory1[i]));
20     }
21     for (int i = 0; i < 5; i++) {
22         questionnum.add(randcat2.get(questioncategory2[i]));
23     }
24     for (int i = 0; i < 5; i++) {
25         questionnum.add(randcat3.get(questioncategory3[i]));
26     }
27 }
```

4.3.5. GameAlgorithms osztály – GetQuestion()

A `GetQuestion()` metódus felelős azért, hogy a `Randomizer()` által előállított kérdés sorszámokat az adatbázisból kikeresse, majd pedig eltárolja azokat egy `ArrayList`-ben.

Először létrejön az adatbázis-kapcsolat a `GameCon()` (4.1.2 Adatbázis-kapcsolat létrehozása) segítségével, valamint meghívódik a `Randomizer()` metódus, amelyről az előzőekben írtam (4.3.4 alfejezet). Beállítódik a szükséges `ArrayList`, a `questionnum`. A metódus SQL `Select`-et alkalmaz a kérdések adatainak lekérdezésére. Ezért szükséges volt létrehozni egy `sql` nevű változót, amelyben magát a lekérdezést tároltam. Az SQL-lekérdezések kötelező módon `try-catch` blokkba zárandóak az esetleges kivételek miatt.

Ahhoz, hogy az SQL utasítások eljussanak az adatbázisba, az eljárásnak szüksége van a `Statement` utasításobjektum egy példányára, amelyet a `createStatement()` metódus hoz létre (17. sor). A `ResultSet` egy olyan interfész, amelyben a végrehajtott lekérdezések eredményei vannak. Az `rs` ennek a `ResultSet`-nek egy példánya, hiszen ezáltal tud végrehajtódni az SQL-utasítás. Ez a 18. sorban történik meg az `executeQuery()` metódussal, amely a `Statement` interfész része.

Annak érdekében, hogy minden adatot képes legyen a metódus kinyerni `while` ciklusba szerveztem a lekérdezést (19-29. sor), így a tábla minden értékén végig halad és az `allquestion` nevű `ArrayList`-be helyezi az eredményeket, majd pedig a `counter` nevű változó értéka eggyel növelődik.

A `counter` változó szerepe fontos a metódus élete során, hiszen mint a kód is mutatja, csak akkor növelődik az értéke, ha az adott kérdés minden szükséges értékét sikerült kinyerni az adatbázisból. Amint egy kérdésen végigért a lekérdezés, az iteráció folytatódik a következő kérdéssel, egészen a 15. kérdésig.

Az SQL-lekérdezések – és természetesen az adatbázis-műveletek elvégzése – során bármilyen komplikáció felléphet. Ezt a `catch` ág menedzseli, benne az `SQLException` kivételosztály egy példányával a kezelésre. Tehát abban az esetben, ha ilyen kivétel dobódna, a Játékos elé fogja tární egy párbeszédablak segítségével a keletkezett hibát.

```

1 import java.sql.ResultSet;
2 import java.sql.Statement;
3 import java.sql.SQLException;
4
5 private Statement s = null;
6 private ResultSet rs = null;
7
8 public void GetQuestion() {
9     GameCon();
10    Randomizer();
11    String sql;
12    int counter = 0;
13
14    for (int i = 0; i < 15; i++) {
15        try {
16            sql = "Select * from question where id =" + questionnum
17                .get(i) + '"';
18            s = con.createStatement();
19            rs = s.executeQuery(sql);
20        } catch (SQLException e) {
21            System.out.println("Hiba történt!");
22        }
23    }
24}
```

```

19     while (rs.next()) {
20         allquestion.add(rs.getString("question"));
21         allquestion.add(rs.getString("qmath"));
22         allquestion.add(rs.getString("sol1"));
23         allquestion.add(rs.getString("sol2"));
24         allquestion.add(rs.getString("sol3"));
25         allquestion.add(rs.getString("sol4"));
26
27         allquestion.add(rs.getString("answer"));
28         counter++;
29     }
30     rs.close();
31 } catch (SQLException e) {
32     dbm.MS(e.getMessage());
33 }
34 }
35 }
```

4.3.6. GameAlgorithms osztály – LateXConverter()

A `LateXConverter()` metódus kétszer szerepel a `GameAlgorithms` osztályban. Ez azért lehetséges, mert a két eljárásnak más a paraméterszignatúrája. Mindez azért megoldható, mert a Java lehetővé teszi a metódus túlterhelést (*Method Overloading*).

Az első esetben a két bemeneti paraméter egy `JButton` típusú gomb, valamint egy `String`, amely a `LATEX`-re átalakítandó kifejezést tartalmazza. Először is a `TeXFormula` osztály egy példányára van szükség, valamint egy `TeXIcon` típusú változóra, illetve egy `BufferedImage` típusú változóra (a konvertált kifejezés képe). A `TeXIcon` valósítja meg az őt létrehozó `TeXFormula`-t, ugyanis a `TeXIcon` nem példányosítható közvetlen módon, csak a `createTeXIcon()` eljárás segítségével valósítható meg (7. sor).

Ezután a `TeXIcon` változónak átadja az eljárás a gombot (10. sor), valamint a `LATEX` kifejezés képét a `getGraphics()` segítségével. Amint ez végrehajtódott, a gomb értékét a `repaint()` metódussal újrarendezzük, ezáltal megjelenik a `LATEX` kifejezés azon.

```

1 import org.scilab.forge.jlatexmath.TeXConstants;
2 import org.scilab.forge.jlatexmath.TeXFormula;
3 import org.scilab.forge.jlatexmath.TexIcon;
4 import java.awt.image.BufferedImage;
5
6 private void LateXConverter(JButton button, String LateXText) {
7     TexFormula fomule = new TexFormula(LateXText);
8     TexIcon ti = fomule.createTeXIcon(TeXConstants.STYLE_DISPLAY,
9             25);
10    BufferedImage b = new BufferedImage(button.getWidth() + 100,
11            button.getHeight() + 100, BufferedImage.TYPE_4BYTE_ABGR);
12    ti.paintIcon(new JButton(), b.getGraphics(), 0, 0);
13
14    button.setIcon(ti);
15    button.revalidate();
16    button.repaint();
17 }
```

Alább látható a másik eset, amelyben a metódus nagy része megegyezik az előző eljárással. Lényegi különbség viszont, hogy nem JButton gombra kerül a LATEX kifejezés, hanem egy JLabel fogja reprezentálni azt, ezáltal az eljárás bemeneti paramétereinek a típusa különbözik az előzőtől, így megvalósítható a metódus-túlterhelés.

```

1 import org.scilab.forge.jlatexmath.TexConstants;
2 import org.scilab.forge.jlatexmath.TexFormula;
3 import org.scilab.forge.jlatexmath.TexIcon;
4 import java.awt.image.BufferedImage;
5
6 private void LateXConverter(JLabel label, String Question) {
7     TexFormula fomule = new TexFormula(Question);
8     TexIcon ti = fomule.createTexIcon(TexConstants.STYLE_DISPLAY,
9             18);
10    BufferedImage b = new BufferedImage(ti.getIconWidth(), ti.
11        getIconHeight(), BufferedImage.TYPE_4BYTE_ABGR);
12    ti.paintIcon(new JLabel(), b.getGraphics(), 0, 0);
13
14    label.setIcon(ti);
15    label.revalidate();
16    label.repaint();
17}

```

4.3.7. GameAlgorithms osztály – saveq()

A saveq() metódus ellenőrzi, hogy a Játékos által kattintott válaszlehetőség megfelel-e a helyes válasznak. Itt úgy döntöttem, hogy nem illesztem be a metódus teljes kódját, hiszen ez négyszer ugyanaz az if-else és else-if elágazás, annyi különbséggel, hogy különböző gombneveket tartalmaz.

Két eset lép fel egy válaszlehetőség kiválasztásakor. Abban az esetben, *ha a helyes válasz megegyezik a Játékos által kattintott gomb szövegével*, akkor a kód az if ágban folytatja a futást (18-22. sor és 33-37. sor). Ekkor párbeszédablak ugrik fel, mely értesíti a Játékosat, hogy megfelelő választ adott, valamint emellett az aktuális pénzösszeg is átállítódik, és a checkpointcheck() eljárással megvizsgálásra kerül, hogy volt-e már szakaszkérdes.

Ellenkező esetben, *ha a válasz helytelen* a lose logikai változó igazra vált (ez a lépés a saveplayermoney() metódusnak a lefutásában fog szerepet játszani). Ezen kívül a Játékos itt is kap egy üzenetet párbeszédablak formájában, amelyben az is szerepel, hogy mekkora összeget bukott el (24-29. sor és 39-44. sor).

Abban az esetben, hogyha valamilyen probléma történne a kódban, akkor egy biztonsági if elágazás tájékoztat arról, hogy a gombon nincs megfelelő érték, ezáltal nem tud tovább folytatódni a futás (52. sor).

```

1 public void saveq.JButton btn1, JButton btn2, JButton btn3,
2     JButton btn4, ActionEvent e, JTextField moneybar, JLabel lA,
3     JLabel lB, JLabel lC, JLabel lD) {
4
5     btn1.setVisible(true);
6     btn2.setVisible(true);
7     btn3.setVisible(true);

```

```
6  btn4.setVisible(true);
7  lA.setVisible(true);
8  lB.setVisible(true);
9  lC.setVisible(true);
10 lD.setVisible(true);
11 String btn = btn1.getText();
12 String button2 = btn2.getText();
13 String button3 = btn3.getText();
14 String button4 = btn4.getText();
15
16 if (e.getActionCommand() != null) {
17     if (e.getActionCommand().contentEquals(btn)) {
18         if (btn1.getText().equals(result)) {
19             dbm.MS("Gratulalok! Helyes valasz!");
20             money = winprice[i];
21             moneybar.setText(String.valueOf(money));
22             checkpointcheck();
23
24     } else {
25         lose = true;
26         dbm.MS("A jatek sajnos vegetert.\nAz elveszitett
27             osszeg: " + money + " Forint.");
28         saveplayermoney();
29
30         System.exit(0);
31     }
32
33     } else if (e.getActionCommand().contentEquals(button2)) {
34         if (btn2.getText().equals(result)) {
35             dbm.MS("Gratulalok! Helyes valasz!");
36             money = winprice[i];
37             moneybar.setText(String.valueOf(money));
38             checkpointcheck();
39
40     } else {
41         lose = true;
42         dbm.MS("A jatek sajnos vegetert.\nAz elveszitett
43             osszeg: " + money + " Forint.");
44         saveplayermoney();
45
46     }
47
48     ...
49
50     if (e.getActionCommand().isEmpty()) {
51         dbm.MS("Nincs megfelelo ertekek!");
52     }
53 }
54 }
```

4.3.8. GameAlgorithms osztály – generate()

A `generate()` metódus valósítja meg a randomszám generálást minimum és maximum értékek között. Ahhoz, hogy pontos legyen az intervallum közötti randomizálási eljárás, ennek utána kellett néznem. A lenti metódus elkészítésében az interneten talált példák segítették munkámat.⁴

A `Math.random()` metódus egy pszeudorandom lebegőpontos (`double`) számmal tér vissza, amely 0.0 és 1.0 közötti érték, pontosabban [0.0, 1.0].

Az intervallum jelen esetben a $((\max - \min) + 1)$. Ha ez összeszorzásra kerül a `Math.random()` metódussal – amelynek az intervalluma [0.0, 1.0] –, akkor a szorzat után az alsó határ 0 marad, azonban a felső határ $(\max - \min, \max - \min + 1)$ lesz.

Kikényszerített típuskonverzió ("casting it with an `int`") után, az intervallum $[0, \max - \min]$ lesz, és hogyha mindehhez `min`-t adunk hozzá, akkor az intervallum $[\min, \max]$ lesz, ami számunkra szükséges.

```
1 public static int generate(int min, int max) {
2     return min + (int) (Math.random() * ((max - min) + 1));
3 }
```

4.3.9. GameAlgorithms osztály – playerhelp2randomizer()

Ez a metódus valósítja meg a közönség segítségét. Mivel ez a segítség egy diagramot jelenít meg, szükséges volt a diagramokat megvalósító, illetve ezt képpé generáló külső könyvtárra (1. sor).

Az eljárás típusa `CategoryDataset`, amely az `org.jfree.data.category` függvénykönyvtár, `CategoryDataset` interfész utódja. Ennek segítségével adattömbök hozhatók létre, amelyek tárolják a különböző válaszok, különböző százalékos arányát. Kézőbbiekben ezekből a `dataset`-ekből dinamikus módon generálódik a szükséges diagram.

Az oszlopok értékeinek meghatározását a `generate()` metódus végzi (4.3.8 alfejezet). Mivel százalékos arányról van szó, így a maximum érték a 100 lehet (9. sor). Abban az esetben, ha a helyes válasz oszlopát kell meghatározni, 60-80 % között történik a véletlenszám generálása (17. és 36. sor), ellenkező esetben 15-50 % között (23. és 42. sor).

Amint a randomizálás megtörtént, és az értékek bekerültek a `percent` tömbbe, elkezdődik a diagram megrajzolásához szükséges adatok betöltése (26-29. és 45-48. sor). Ahogy látható, a `percent` tömb az első `if` blokkon belül 0. elemként kapja meg a `max` változót, majd ez a `percent[0]` az A válaszlehetőséggel lesz egyenlő. Azaz ebben az esetben az A a helyes válasz, és ennek lesz a legmagasabb az oszlopa, az oszlopdiagramon (Pontosan az az érték, amely 60 és 80 között generálódott).

Ezért szükséges négyeszer elvégeznie a metódusnak szinte ugyanazokat az `if` elágazásokat, azonban minden esetben más indexet kap meg a `percent[]` tömb, és másik betűjellel lesz egyenlő, ezáltal lefedve minden a négy lehetőséget, a jó válaszokra.

A metódustörzs végrehajtása után az eljárás az előállított `dataset`-tel adja vissza a vezérlést a programnak. Az ebben a metódusban visszaadott `dataset`-et a fentiekben említett `ChartMaker()` (4.3.3 alfejezet) metódus használja fel.

⁴<https://www.techiedelight.com/generate-random-integers-specified-range-java/>

```
1 import org.jfree.chart.*;
2 private DefaultCategoryDataset dataset = new
3     DefaultCategoryDataset();
4 private int playerhelp2 = 1;
5
6 public CategoryDataset playerhelp2randomizer(JButton btn1,
7     JButton btn2, JButton btn3, JButton btn4, JButton btn5)
8 throws IOException {
9     dbm.MS("Kozonseg segitsege");
10    int max = 100;
11    Random rand = new Random();
12    int[] percent = new int[4];
13
14    if (playerhelp2 > 0) {
15        playerhelp2--;
16
17        if (btn1.getText().equals(result)) {
18            max = generate(60, 80);
19            percent[0] = max;
20            for (int i = 1; i < percent.length; i++) {
21                if (i == 0) {
22                    i++;
23                } else {
24                    percent[i] = generate(15, 50);
25                }
26            }
27            dataset.addValue(percent[0], "A", "A");
28            dataset.addValue(percent[1], "B", "B");
29            dataset.addValue(percent[2], "C", "C");
30            dataset.addValue(percent[3], "D", "D");
31            if (ChartMaker() == true) {
32                bc.setVisible(true);
33            }
34        }
35        if (btn2.getText().equals(result)) {
36            max = generate(60, 80);
37            percent[1] = max;
38            for (int i = 0; i < percent.length; i++) {
39                if (i == 1) {
40                    i++;
41                } else {
42                    percent[i] = generate(15, 50);
43                }
44            }
45            dataset.addValue(percent[0], "A", "A");
46            dataset.addValue(percent[1], "B", "B");
47            dataset.addValue(percent[2], "C", "C");
48            dataset.addValue(percent[3], "D", "D");
49        }
```

```
50
51     if (ChartMaker() == true) {
52         bc.setVisible(true);
53     }
54 }
55
56     ...
57
58     btn5.setBackground(new java.awt.Color(153,153,153));
59 } else {
60     dbm.MS("Csak egyszer hasznalhato ez a segitseg!");
61 }
62 return dataset;
63 }
```

4.4. Kártyapakli készítő osztályai – deckCreate csomag

A kártyapakli készítő játékrész osztályai a `deckCreate` csomagban találhatóak. Öt osztály található itt: DBM, az egyetlen Java osztály; `FirstPg`, `DeckCreate`, `TextEditor` és `Selector`, a négy JFrame Form. Ezáltal ebben a csomagban is a DBM osztályban található a fontosabb tartalmi kód, a másik négyben pedig a GUI felépítéséhez szükséges kódok.

4.4.1. FirstPg osztály

Ahogyan már említettem, ez az osztály egy JFrame Form, tehát itt csak a kinézet lett megvalósítva különböző JPanel-ekkel, JLabel-ökkel és JButton gombokkal.

Habár ez az osztály a `deckCreate` csomagban található, ahol a kártyapakli készítő játékrész osztályai vannak, ez valójában egy közös része a két játékrésznek, ahol a Játékosnak választania kell a két rész közül, hogy melyikkel szeretne játszani (az 5.1 alfejezetben látható).

4.4.2. TextEditor és DeckCreate osztály

Ez a két osztály szintén JFrame Form, amellyel a GUI létrehozása történt. Azonban még a `DeckCreate` osztályban szinte csak `actionPerformed` eljárások találhatóak a különböző komponensekhez, addig a `TextEditor` osztályban található lényegi kód részlet ezeken az eljárásokon kívül.

Ugyanis a `TextEditor` felülete kerül a Játékos elé, amikor *Nyomtatni*, *Másolni*, vagy *Frissíteni* szeretne egy kártyapaklit. Azaz szükséges a kiválasztott kártya összes komponensének betöltése (háttérzín, felírt szövegek). Először is ez történik meg, *setter* metódusok segítségével.

Aztán annak megfelelően, hogy éppen melyik funkciót hívta meg a Játékos, különböző gomb lehetőségek kerülnek elé, amelyek a megfelelő metódust hívják meg kattintás hatására – `PrintDeck()`, vagy `Insert()`, vagy `Update()`.

4.4.3. Selector osztály

A `Selector` JFrame Form kerül a Játékos elé, mielőtt a fentebb említett három funkciót létre tudja hajtani. A játék során úgy lehetséges a *Nyomtatás*, *Másolás*, vagy *Frissítés*, hogyha előtte a Játékos kiválasztja, hogy melyik kártyapakli, melyik kártyáját szeretné *Nyomtatni*, *Másolni*, vagy *Frissíteni*. A felület kinézete a későbbiekben látható az 5.2 Kártyapakli készítése alfejezetben.

A JFrame Form-on egyszerű JPanel-ek, JButton gombok találhatóak, amely gombok segítségével, és a JTextField-be írt szöveg segítségével a Játékos kiválaszthatja először a kártyapaklit – ID szerint –, majd az abban lévő kártyalapokat – szintén ID szerint. Természetesen nem kell fejből tudnia a Játékosnak, hogy milyen kártyapaklik léteznek már, és azokban milyen ID-val léteznek kártyalapok. A kártyapakli választása előtt egy JTextArea-ban látja a Játékos, hogy milyen ID-val léteznek már kártyapakklik, hogy melyekből tud választani, majd miután kiválasztotta a pakli ID-ját, azután a benne lévő kártyalapok felsorolását láthatja – a kártyák ID-jával, és nevével.

4.4.4. DBM osztály

Ahogyan azt a `GameAlgorithms` osztálynál is tettem, itt a DBM osztálynál is csak az érdekesebb, és bonyolultabb metódusokat mutatom be a kódjaikkal együtt. A többi eljárásról csak rövidebben, kód nélkül teszek említést.

Az `Update()` metódus a "*Pakli frissítése*" funkciót valósítja meg. Teljesen ugyanazt a műveletet hajtja végre, mint a 4.4.5 DBM osztály – `Insert()`, annyi különbséggel, hogy SQL `Insert` helyett, a névből is adódóan, SQL `Update` történik az adott kártyára.

A `DeckLoader()` eljárás a kártyák betöltéséért felelős. Egy SQL-lekérdezés segítségével, az adatbázisból azok azonosítói kerülnek kiválasztásra. Az SQL-lekérdezés eredménye egy `ArrayList`, amelybe a lekért adatok kerülnek. Annak érdekében kerülnek dinamikus tömbbe az adatok, hogy bármekkor kártyapakli számot képes legyen eltárolni a program, anélkül, hogy azt szükséges lenne beégetni, egy fix tömbmérettel.

A `PrintDeck()` metódus hajtja végre a "*Pakli nyomtatása*" funkciót. Az eljárás először SQL `Select` parancs segítségével lekéri az adatbázisból az összes adatot a megadott kártyáról. Szükséges létrehozni egy `PdfWriter` típusú változót, illetve ezzel magát a PDF dokumentumot, amelybe a nyomtatás fog történni. Az adatbázisból kikért összes adatból, a `captureComponent()` metódus segítségével egy screenshot készül, azaz egy `.png` fájl minden kártyalap minden oldaláról, majd ezek a képek kerülnek be a PDF dokumentumba, ezáltal elkészült a nyomtatás.

Mivel a fentebb említett mindenhol eljárásban SQL-lekérdezés történik, ezért természetesen mindenhol metódusban `try-catch` blokkba van zárva, és `SQLException` kivételkezelés történik, a `catch` ágban.

A `DeckChooser()` metódus beállítja egy statikus változóba a `Selector` felületen kiválasztott pakli sorszámát.

A `CardChooser()` eljárás szintén beállítja egy statikus változóba a kiválasztott kártyalap ID-ját – mint a `DeckChooser()` a választott kártyapakli ID-ját.

A `CopyCard()` metódus valósítja meg a "*Pakli másolása*" funkciót. Az eljárás a Játékos által a `Selector` felületen kiválasztott kártyapaklin belüli kártyalap adatait egy SQL `Select` parancs segítségével lekérdezi az adatbázisból, majd a `TextEditor` felületen megjeleníti a kártyalapot. Ekkor a Játékos számára a "Másolás" JButton gomb láthatóvá válik. A másolás úgy történik meg, hogy a Játékosnak meg kell adnia egy új kártyapakli ID-t a felületen, majd a "Másolás" gomb segítségével megtörténik a másolás folyamata, amely a 4.4.5 DBM osztály – `Insert()` metódus meghívásával történik.

A `ContainSlash()` metódus a L^AT_EX parancsokhoz szükséges perjelek vizsgálatára illetve korrekciójára szolgál.

A `MS()` eljárás a szinte legtöbbet használt metódus a kód minden részén. Ez egy általam definiált párbeszédablak. A `JOptionPane` osztály biztosítja a standard, vagy különbözőféle párbeszédablakok létrehozását. Ezt a kód bármely részén lehetőség van meghívni – természetesen, ha a DBM osztály példányosítva lett ott –, egyszerűen az argumentumába kell írni az általunk megjeleníteni kívánt szöveget, mint `String`: `MS("Az üzenet szövege");`

```

1 public void MS(String msg) {
2     JOptionPane.showMessageDialog(null, msg, "Quiz game",
3         JOptionPane.INFORMATION_MESSAGE);
}
```

4.4.5. DBM osztály – Insert()

Az `Insert()` eljárás egy SQL `Insert`-et tartalmaz, amely az adott kártya minden egyes komponensét, minden tartalmát kinyeri a `getter` metódusok segítségével, és ezeket felviszi az adatbázisba.

Az SQL utasítás eljutása az adatbázisba a már ismert módon történik a `Statement` utasításobjektum egy példányával, valamint a `ResultSet` segítségével.

Az esetleges kivételeket természetesen itt is `SQLException` kivételkezelővel kell kezelni, hiszen SQL-lekérdezés történik.

```

1 public void Insert(int did, JTextField t1, JTextField t2,
2 JTextField t3, JTextField t4, JTextField t5, JTextField t6,
3 JTextField t7) {
4
5     String qM = ContainSlash(t2.getText());
6     String rM = ContainSlash(t4.getText());
7     String sql = "Insert into cards(deckID, colour, questionText,
8         questionMath, resultText, resultMath, cardname, img, imgresult,
9         qFSize, qFType, rFSize, rFType) values ('" + did + "', '" +
10        colour + "', '" + t1.getText() + "', '" + qM + "', '" + t3.
11        getText() + "', '" + rM + "', '" + t5.getText() + "', '" + t6.
12        getText() + "', '" + t7.getText() + "' + getSize() + ", '" +
13        getFtype() + "', '" + getSize1() + "', '" + getFtype1() +
14        "')";
15
16     Con();
17     try {
18         s = con.createStatement();
19         s.execute(sql);
20         MS("Sikeresen regisztrált az adatbázisba!");
21     } catch (SQLException e) {
22         MS("Sikertelenül regisztrált az adatbázisba!");
23     }
24 }
```

4.4.6. DBM osztály – captureComponent()

Ez a metódus `String` típusú, ezáltal ilyen típust vár visszatérési értéknek, amely késsébbiekben az elkészült kép fájlneve lesz. Bemeneti paramétere egy komponens a GUI felületről.

Első lépésként a `rect` nevű változónak átadjuk a komponens adatait a `getBounds()` eljárással. `try-catch` blokkba zárjuk az esetleges kivételek miatt, jelen esetben a kép készítés miatt `IOException` megdobódására lehet számítani.

A fájlnevet egy hosszabb kombinációval adtam meg (5-6. sor), annak érdekében, hogy ismétlődés egyáltalán ne fordulhasson elő, vagy csak kis mértékben. A `component` stílusát – azaz minden kinézeti elemét – a kód ráilleszti egy `BufferedImage` típusú változóra (8. sor), amelyet egy IO művelet során kiexportál `.png` formátumba (10-11. sor).

Ezután, ahogy már előbb is említettem, a visszatérési érték a lementett kép fájlneve lesz.

```

1 public String captureComponent(Component component) throws
2   Throwable {
3   java.awt.Rectangle rect = component.getBounds();
4   String fileName = "";
5   try {
6     String format = "png";
7     fileName = getDeckid() + getCardid() + component.getX() +
8       LocalDateTime.now().getSecond() + "." + format;
9     BufferedImage captureImage = new BufferedImage(rect.width,
10        rect.height, BufferedImage.TYPE_INT_ARGB);
11    component.paint(captureImage.getGraphics());
12
13    ImageIO.write(captureImage, format, new File(fileName));
14    Image(fileName);
15  } catch (IOException ex) {
16    System.err.println(ex);
17  }
18  return fileName;
19 }
```

4.4.7. DBM osztály – Image()

Ezen metódus a kártyákról készült kép PDF fájlhoz való hozzáadását végzi el.

Szükséges egy PdfWriter típusú változó létrehozása, amely egy DocWriter osztály PDF készítéséhez (8. sor), illetve magát a PDF dokumentumot is létre kell hozni (10-11. sor). Ezután a kép beillesztéséhez szükséges változókat kell definiálni: ImageData típusú, illetve Image típusú változókat (13-14. sor).

Az image változóban a scaleToFit() metódus segítségével megadható a kép magassága és szélessége, az általunk kívánt méretűre (15. sor). Ezután a képek bekerülnek a dokumentumba, és a PDF dokumentum lezárásra kerül (17-18. sor).

```

1 import com.itextpdf.kernel.pdf.PdfDocument;
2 import com.itextpdf.kernel.pdf.PdfWriter;
3
4 public void Image(String imFile) throws FileNotFoundException,
5   IOException {
6
7   String dest = "pdf/" + getDeckid() + "szamupakli" + ".pdf";
8
9   PdfWriter writer = new PdfWriter(dest);
10  PdfDocument pdf = new PdfDocument(writer);
11  Document document = new Document(pdf);
12
13  ImageData data = ImageDataFactory.create(imFile);
14  Image image = new Image(data);
15  image.scaleToFit(100, 200);
16
17  document.add(image);
18  document.close();
19 }
```

4.4.8. DBM osztály – CardLoader()

A CardLoader() eljárás két bemeneti paraméterrel rendelkezik, egy ArrayList-tel, amelybe a kártya szükséges paramétere lesznek tárolva, illetve egy JTextArea, ahol ezek az adatok megjelenítésre kerülnek.

SQL-lekérdezéssel a megszokott módon ResultSet-en keresztül végrehajtódik a Select parancs, melynek értékei a paraméterlistában szereplő ArrayList-nek adódnak át. Mivel SQL-lekérdezésről van szó, így szükséges lekezelni az esetleges megdobódott SQL-kivételeket, ezért try-catch blokkba kell zárni a kód ezen részét.

Ezután egy foreach ciklus végigjárja az ArrayList-et, és az abban szereplő adatokat beleilleszti a textArea-ba, ezáltal kilistázva a kártyák ID-ját illetve nevét. (5.18 ábra)

```

1 public void CardLoader(ArrayList a, JTextArea textArea) {
2     int i = 0;
3     String selection = "Select id,cardname from cards where
4         deckID=" + getDeckid() + '"';
5     String k1 = "";
6     try {
7         Con();
8         s = con.createStatement();
9         rs = s.executeQuery(selection);
10        while (rs.next()) {
11            k1 = String.valueOf(rs.getInt("id"));
12            a.add(k1);
13            k1 = null;
14            k1 = rs.getString("cardname");
15            a.add(k1);
16            k1 = null;
17        }
18        rs.close();
19    } catch (SQLException e) {
20        MS(e.getMessage());
21    }
22
23    for (Object object : a) {
24        if (i < a.size()) {
25            textArea.append("Card id");
26            textArea.append("      " + String.valueOf(a.get(i)));
27            i++;
28
29            textArea.append("      " + a.get(i));
30            textArea.append("\n");
31            i++;
32        }
33        if (i == a.size() - 1) {
34            break;
35        }
36    }
}

```

4.5. Project Files - pom.xml

A POM, nem más mint (*Project Object Model*, azaz projektobjektum-modell). Ez a Maven projekt egy XML reprezentációja, a pom.xml fájlban tárolva. A POM tartalmaz minden szükséges információt a projektről, mint például a build folyamat során használt plugin-ek konfigurációját, ezáltal a Maven képes további leírások nélkül elvégezni a build folyamatot.

Néhány elemet megemlítenék az XML fájlból. A fájl gyökérelem a <project>.

A következőkben láthatók az alap elemek:

```
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">

    <modelVersion> 4.0.0 </modelVersion>
    <groupId> com.mycompany </groupId>
    <artifactId> Szakdoga </artifactId>
    <version> 1.0-SNAPSHOT </version>
    <packaging> jar </packaging>

    <properties>
        <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
        <maven.compiler.source> 11 </maven.compiler.source>
        <maven.compiler.target> 11 </maven.compiler.target>
    </properties>

    <dependencies> ... </dependencies>

</project>
```

A <dependencies> elemei a függőségek, azaz a külső függvénykönyvtárak amelyektől a program függ. Az adott függőséget a Maven letölti automatikusan. A dolgozatom elején említett (2.1.1 alfejezet) fontosabb külső könyvtárak a következőképpen találhatóak meg a pom.xml fájlból:

```
<dependencies>

    <dependency>
        <groupId> com.itextpdf </groupId>
        <artifactId> itextpdf </artifactId>
        <version> 5.5.9 </version>
    </dependency>

    <dependency>
        <groupId> org.scilab.forge </groupId>
        <artifactId> jlatexmath </artifactId>
        <version> 1.0.7 </version>
    </dependency>
```

```
<dependency>
    <groupId> mysql </groupId>
    <artifactId> mysql-connector-java </artifactId>
    <version> 8.0.19 </version>
</dependency>
...
</dependencies>
```

A pontos Maven dependency-ket online⁵ kerestem ki.

4.6. Futtatható állományba generálás

A programból a futtatható állomány egy .jar állomány.

A .jar fájl létrehozásához szükségem volt egy Maven plugin-ra, a pom.xml fájlból, a build elemen belül.

```
<build>
    <plugins>
        ...
        <plugin>
            <groupId>com.akathist.maven.plugins.launch4j</groupId>
            <artifactId>launch4j-maven-plugin</artifactId>
            <version>2.1.0</version>

            <executions>
                <execution>
                    <id>l4j-gui</id>
                    <phase>package</phase>
                    <goals>
                        <goal>launch4j</goal>
                    </goals>

                    <configuration>
                        <headerType>gui</headerType>
                        <jar>target/${project.artifactId}-${project.version}.jar</jar>
                        <dontWrapJar>false</dontWrapJar>
                        <classPath>
                            <mainClass>signinup.Signinup</mainClass>
                        </classPath>
                        ...
                    </configuration>
                </execution>
            </executions>
        </plugin>
    </plugins>
</build>
```

⁵<https://mvnrepository.com/>

A Launch4j "csomagoló" plugin-re volt szükségem a futtatható állományba csomagoláshoz. Az ehhez szükséges XML kódot a Launch4j⁶ weboldalán találtam meg, valamint a használatról is itt tudtam olvasni.

⁶<http://launch4j.sourceforge.net/docs.html>

5. fejezet

Tesztelés

5.1. A játék menete

A első lépés ahhoz, hogy egy Játékos el tudjon kezdeni játszani, az a **Bejelentkezés** vagy **Regisztráció**.

The screenshot shows a window titled 'Kvíz világ'. The main title is 'A játék használatához Jelentkezz be, vagy Regisztrálj!'. Below it are two input fields: 'Felhasználónév' and 'Jelszó'. Underneath these is a blue button labeled 'Bejelentkezés'. Below that is the word 'vagy' (or). At the bottom is a blue button labeled 'Regisztrálok !'.

(a) Bejelentkezés

The screenshot shows a window titled 'Regisztráció'. The main title is 'Regisztráció'. Below it are four input fields arranged in a 2x2 grid: 'Felhasználónév' (top-left), 'Teljes név' (top-right), 'Jelszó' (bottom-left), and 'Kar' (bottom-right). At the bottom is a blue button labeled 'Regisztráció'.

(b) Regisztráció

5.1. ábra. Bejelentkezés és Regisztráció felülete.

Erre azért van szükség, hogy az elért eredmények rögzítésre kerüljenek az adatbázisban. A regisztrációhoz szükséges egy *Felhasználónév*, valamint egy ehhez tartozó *Jelszó*, és mindenek mellett a *Teljes név* megadása, valamint a *Kar* megadása (5.1 ábra).

A program ezen a ponton különböző hibaüzenetekkel jelezheti, ha a Felhasználó helytelenül adott meg valamit *Bejelentkezés* során. Figyelmeztetést kap a Felhasználó, hogyha helytelenül adta meg a *Felhasználónevet*, vagy a *Jelszót*. Ekkor nem engedi be őt a rendszer, hanem újra kell próbálkoznia (5.2 ábra).

5.1. A játék menete

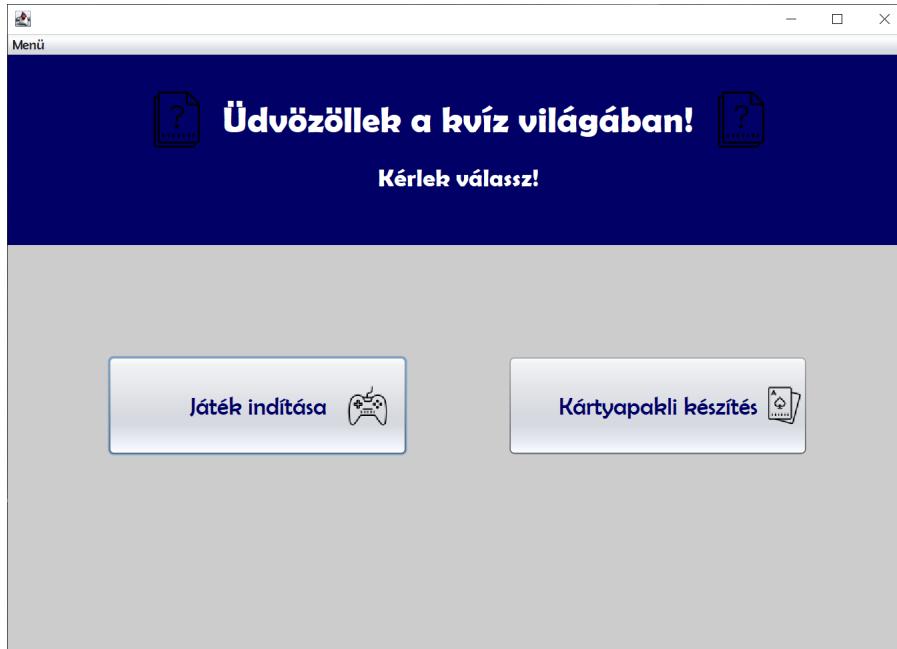


(a) Figyelmeztetés nem megfelelő felhasználónév esetén.
(b) Figyelmeztetés nem megfelelő jelszó esetén.
(c) Sikeres bejelentkezés.

5.2. ábra. Bejelentkezés és Regisztráció felülete.

Azonban sikeres bejelentkezés esetén is kap üzenetet a Felhasználó. A sikeres regisztráció után, már kezdődhet is a játék.

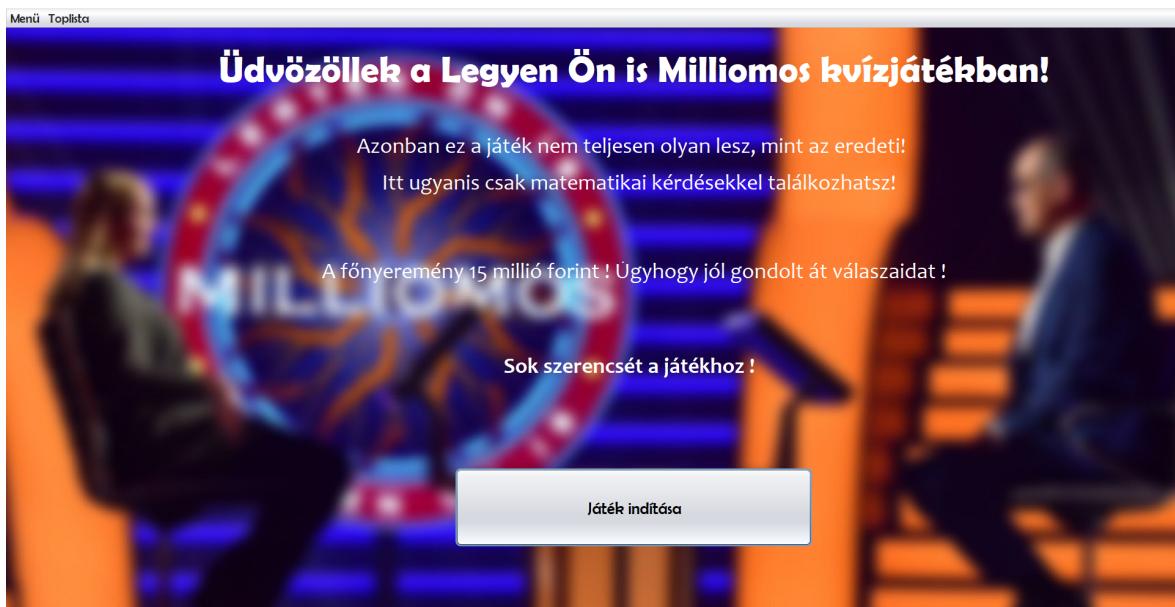
Mivel a Kvízjáték, és a Kártyapakli készítő közös felülete a Bejelentkezés és Regisztráció, ezért a Játékosnak a sikeres bejelentkezés utána választania kell, hogy a kvízjátékkal játszik, vagy kártyapaklit készít (5.3 ábra).



5.3. ábra. Bejelentkezés utáni választás a játékrészek között.

A "Játék indítása" gombra kattintva a következő (5.4 ábra) üdvözlő felület fogadja a Felhasználót, valamint a menüsorban a következő elemekből tud választani (5.5 ábra).

5.1. A játék menete

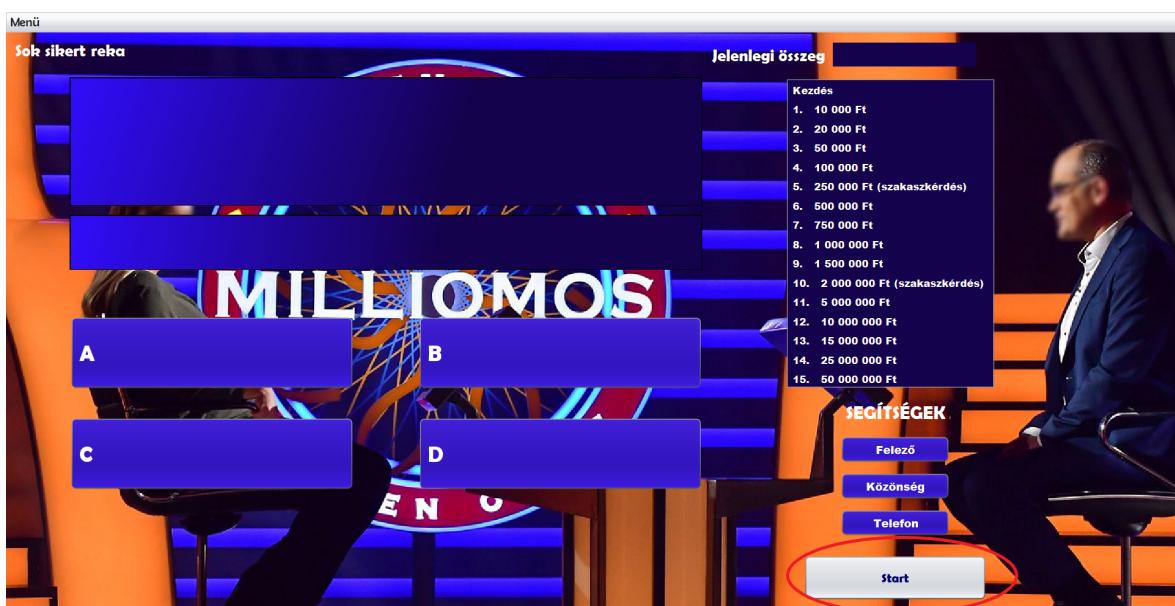


5.4. ábra. A Játék indító felülete.



5.5. ábra. Menüsor elemei.

Ezután a Játékos a "Start" gombra kattintva tudja indítani a játékot (5.6 ábra).



5.6. ábra. A Játék felülete.

A Játékosnak 15 egyre nehezedő feleletválasztós kérdésre kell válaszolnia, és a négy válaszlehetőség közül kell kiválasztania, az általa jónak vélt választ. Ha döntött a négy lehetőség közül, megjelöli az általa helyesnek gondolt választ, majd a játék kijelzi, hogy az helyes volt-e, vagy sem.

A Játékosnak van lehetősége *Megállni*, vagy *Folytatni* a játékot a helyesen megválaszolt kérdések után, valamint két *Segítsége* is van, amelyeket csak egyszer használhat fel az egész játék során.

Az egyes kérdések megválaszolásánál nincsen időkorlát, hiszen nem a gyorsaság a játék lényege, hanem úgymond a feszültségkeltés, hogy az egyre nehezedő kérdések közül meddig tud eljutni a Játékos.

5.1.1. Nyeremények

A következő pénzösszegeket lehet megnyerni a játék során:

1. 10 000 Ft
2. 20 000 Ft
3. 50 000 Ft
4. 100 000 Ft
5. **250 000 Ft első szakaszkérdés**
6. 500 000 Ft
7. 750 000 Ft
8. 1 000 000 Ft
9. 1 500 000 Ft
10. **2 000 000 Ft második szakaszkérdés**
11. 5 000 000 Ft
12. 10 000 000 Ft
13. 15 000 000 Ft
14. 25 000 000 Ft
15. **50 000 000 Ft fődíj-kérdés**

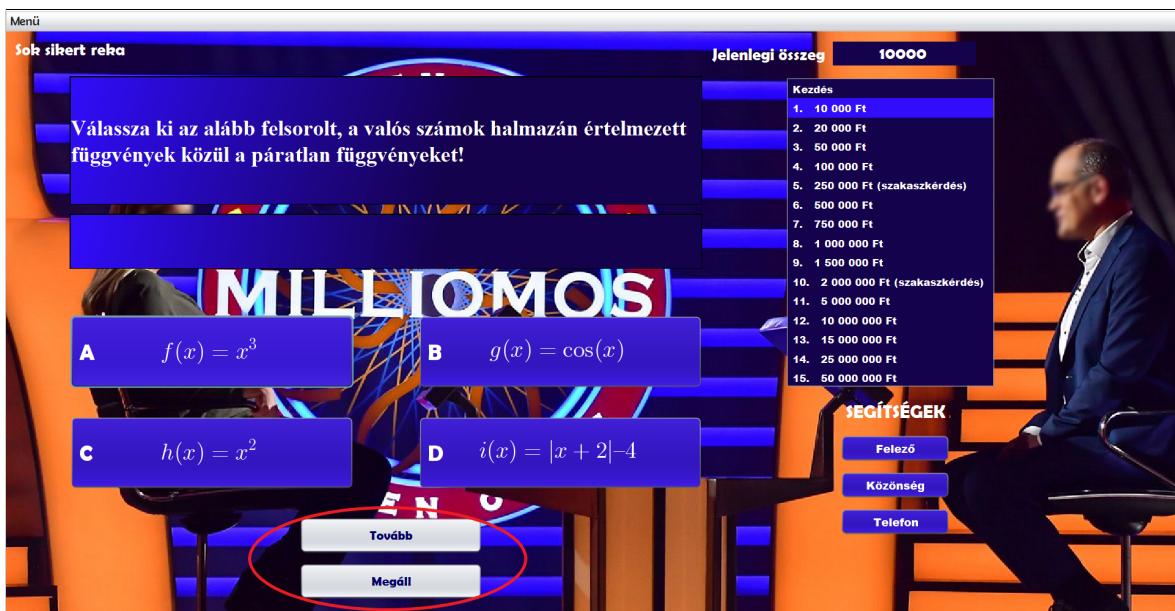
5.1.2. Játékszabályok

A Játékosnak tehát időkorlát nélkül, összesen 15 kérdésre kell helyes választ adnia, ha meg szeretné nyerni a fődíjat. Azonban vannak úgynevezett "**szakaszkérdések**", amely kérdésekhez tartozó pénzösszeg garantált. Ilyen szakaszkérdés az 5. és a 10. kérdés.

Ez példával élve azt jelenti, hogy ha a Játékos tegyük fel a 7. kérdésre rossz választ ad, akkor számára véget ér a játék, nem kapja meg a 750 000 Forintot, ami a 7. kérdéshez tartozik. Azonban nem távozik üres kézzel, hiszen az 5. kérdésen, azaz az első szakaszkérdésen már túljutott, és így a 250 000 Forintot megnyerte.

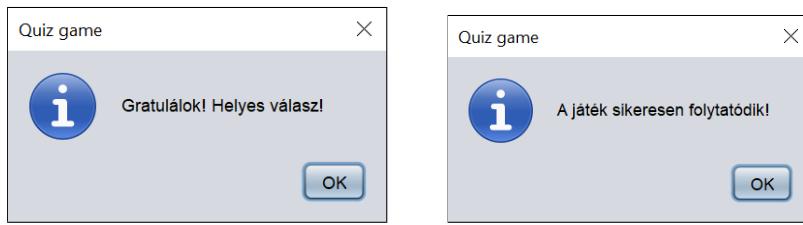
A Játékosnak minden helyesen megválaszolt kérdés után lehetősége van, vagy tovább folytatni a játékot – a "**Tovább**" gombra nyomva – vagy pedig megállni – a "**Megáll**" gombra kattintva (5.7 ábra).

5.1. A játék menete



5.7. ábra. "Tovább", vagy "Megáll" választása helyes válasz után.

Ha továbbhalad a játékban, akkor automatikusan jön a soron következő kérdés (5.8 ábra).



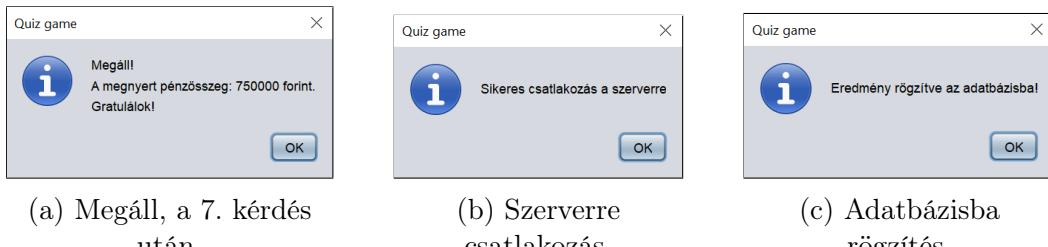
(a) Helyes válasz.

(b) Folytatás.

5.8. ábra. Párbeszédablakok helyes válasz és folytatás után.

Ha azonban meg szeretne állni, az azt jelenti, hogy a jól megválaszolt kérdéshez tartozó összeget megszerzi, és véget ér számára a játék.

Az előző példával élve, ha a Játékos tegyük fel a hetedik kérdésre ebben az esetben jól válaszol, és nem szeretne tovább menni, a "Megáll" gombra kattint, akkor hiába ment túl az ötös szakaszkérdésen, a hetedik kérdésre is helyes választ adott, csak éppen nem szeretné tovább folytatni, ezért a nyereménye 750 000 Ft, a játék így ér véget számára, ezzel az összeggel.



(a) Megáll, a 7. kérdés után.

(b) Szerverre csatlakozás.

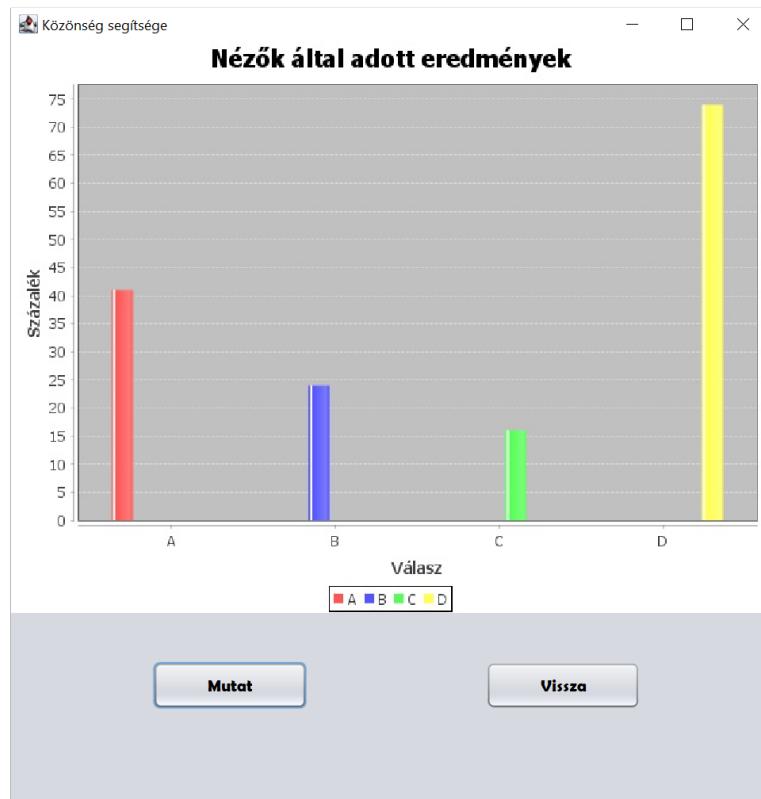
(c) Adatbázisba rögzítés.

5.9. ábra. Párbeszédablakok helyes válasz és folytatás után.

A Játékosnak lehetősége van **három segítség** közül választani a játék során, ha éppen elbizonytalanodna abban, hogy melyik is a helyes válasz. Ez a három segítség a "Felező", a "Közönség" és "Telefon".

A "Felező" segítségével a program a négy válasz közül eltávolít véletlenszerűen kettő rosszat, ezáltal két válaszlehetőség marad csak: egy rossz és a helyes válasz. A Játékos már csak eközül a két válaszlehetőség közül tud választani, ezáltal nagyobb az esélye, hogy eltállja a jó választ.

A "Közönség" segítségével a Játékos egy statisztikát fog maga előtt látni, egy oszlopdiagramot. Ez a diagram megmutatja, hogy a közönség a négy válaszlehetőség közül melyikre milyen százalékban szavazott – az általuk jónak vélt válaszra (5.10 ábra).



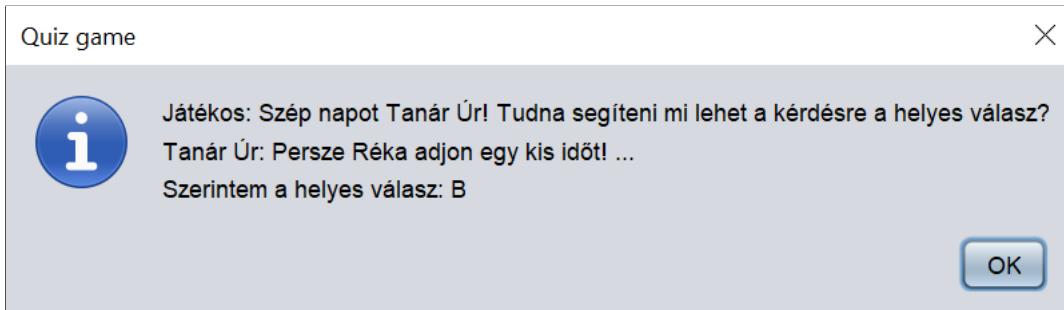
5.10. ábra. Közönség segítsége.

Az eredeti televíziós játékban ténylegesen a jelenlévő közönségnak kell szavazni, ugyanis amikor az adott Játékos kéri ezt a segítséget, akkor minden ott ülő személynek a közönségből egy gomb megnyomásával kell jelezni, hogy szerintük melyik a helyes válasz, majd néhány másodperc után a Játékos előtt megjelenik százalékos összesítésben a szavazatok eredménye. A szavazatok összesítése után pedig meghozza a döntést a Játékos.

Itt az általam elkészített játékban azonban nyilvánvalóan nincsen tényleges közönség, aikik szavazni tudnának. Az oszlopdiagram ábrázolását random szám generálás segítségével valósítottam meg.

A "Telefon" segítség nem más, mint az eredeti játékból származó segítség, ahol a Játékos telefonon felhívhatja egy ismerőt, aki segítséget nyújt neki, hogy szerinte mi a helyes válasz, mindezt időkorláttal ellátva. A Játékosnak ugyanis 30 másodperce van, hogy elmondja a kérdést, a négy válasszal, és ezt meg is válaszolja az általa felhívott személy.

Persze a játékban nincsen lehetőség ténylegesen hívást indítani egy ismerős felé. Ebben a segítségen egy kitalált szöveget írtam meg, mintha tényleg felhívott volna a Játékos valakit, aki erre válaszol is, és elmondja, hogy szerinte melyik a helyes válasz a négy lehetőség közül (5.11 ábra).



5.11. ábra. Telefonos segítség.

A segítségeket természetesen csak egyszer lehet felhasználni a játék során. Ha még egyszer megpróbálja a Játékos kérni a segítséget, akkor erre a játék figyelmezteti, hogy már felhasználta az adott segítséget, valamint ezt színnel is jelöli (5.12 ábra).



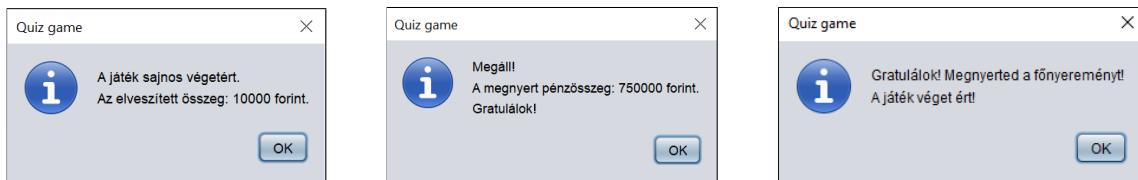
5.12. ábra. Elhasznált segítségek jelzése.

5.1.3. A játék vége

Ahogyan azt már említettem is, a játék többféleképpen is véget érhet. Tehát a játék véget ér, ha a Játékos:

- *rosszul válaszol a kérdésre.* Azaz a Játékos nem tudja a helyes választ, helytelent jelöl meg. Ekkor a nyereménye az utolsó elért szakaszkerdéshez tartozó garantált nyeremény, vagy ha nem ért még el szakaszkerdéshez, akkor 0 Forint.
- *megáll.* Ha a Játékos úgy érzi már túl nehéz lenne a következő kérdés számára, nem tudná a választ rá, akkor a "Megáll" gombra kattint. Így az összeg, amit hazavihet, azaz eddig megszerzett nyeremény, vagyis annak a kérdésnek a nyeremény összege, amelyik után úgy döntött, hogy megáll.
- *a 15. kérdésre is jól válaszol.* Azaz a Játékos minden kérdésre jól tudta a választ, és megnyerte a fődíjat.

5.1. A játék menete



(a) Párbeszédablak rossz válasz esetén.

(b) Párbeszédablak megállás esetén.

(c) Párbeszédablak főnyeremény esetén.

5.13. ábra. A lehetséges játék vége párbeszédablakok.

Természetesen megállás, és a főnyeremény megnyerése esetében az összegek rögzítésre kerülnek az adatbázisba.

A játékszabályok ismertetését a [14] forrás alapján készítettem el.

5.2. Kártyapakli készítése

Ahogyan azt már említettem az 5.1 A játék menete alfejezetben, a Kvízjáték és a Kártyapakli készítés játékrész közös felülete a Bejelentkezés, és Regisztráció (5.1 ábra), valamint a Bejelentkezés után következő oldal, ahol a Játékosnak ki kell választania, hogy melyik résszel szeretne játszani (5.3 ábra).



5.14. ábra. Szín kiválasztása.

A Játékosnak először is a kártyalap színét kell kiválasztania. Ezzel csak egy darab adott kártya színe lesz ilyen, a pakliban lehetnek különböző színű lapok. A Játékosnak lehetősége van az alap színek közül választani, a felettük lévő JCheckBox-al, vagy pedig megadhatja az általa igényelt színt hexadecimális kóddal, "#" hashtag-gel a kód előtt.

Ha a Játékos ezt választja, a kód beírása után a "Mutat" gombbal tudja megnézni a kiválasztott színt. Ha megfelel számára, akkor a "Tovább" gombra kattintva a játék továbbmegy a szövegszerkesztő felületre.

Azonban ha a Játékos úgy dönt, hogy mégis inkább az alapszínekből választana, akkor csak ki kell választani, és így tovább menni.

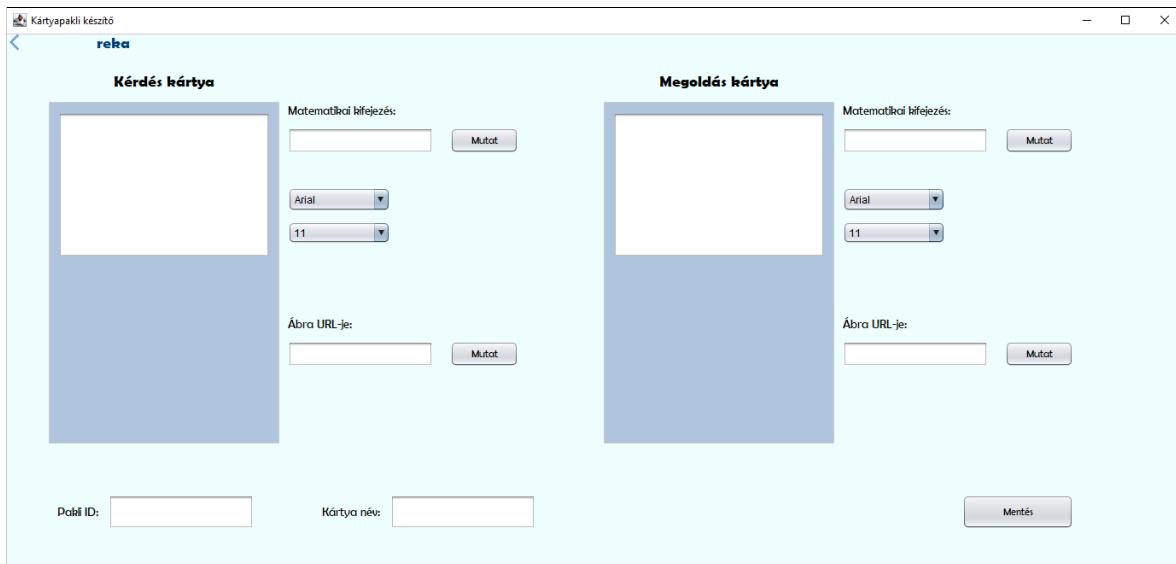
Mindezek ellenére, ha már tovább ment a Játékos, de meggondolja magát, másik színt szeretne, akkor a Szövegszerkesztő oldalon van lehetősége visszalépni egyet, a bal felső sarokban található *Vissza gombbal*

A lent található 5.15 ábrán látható a felület, amely a szín kiválasztása után jelenik meg, a Szövegszerkesztő. A baloldalon szereplő kártya, a kártyalap kérdés része, azaz általában az ilyen kvízkártyáknál a lap eleje. A jobboldalon lévő kártya pedig a *megoldás* része a lapnak, azaz a kártya hátulja.

A kártyalapon lévő fehér négyzetbe kattintva a Játékosnak lehetősége van írni, ugyanis az egy JTextField. Ide írhatja be például a kérdés szövegét. A szövegnek a betűtípusát, és a méretét is át tudja állítani a Játékos, a lap mellett jobbra lévő két legördülő JComboBox elemeiből van lehetősége választani.

A "Matematikai kifejezés" felirat alatti JTextField-be a L^AT_EX paranccsal megírt matematikai kifejezést lehet megadni. A parancsnak nem szükséges a nyelvből ismert §§ jelek között lennie, a program anélkül is képes átkonvertálni a kifejezést.

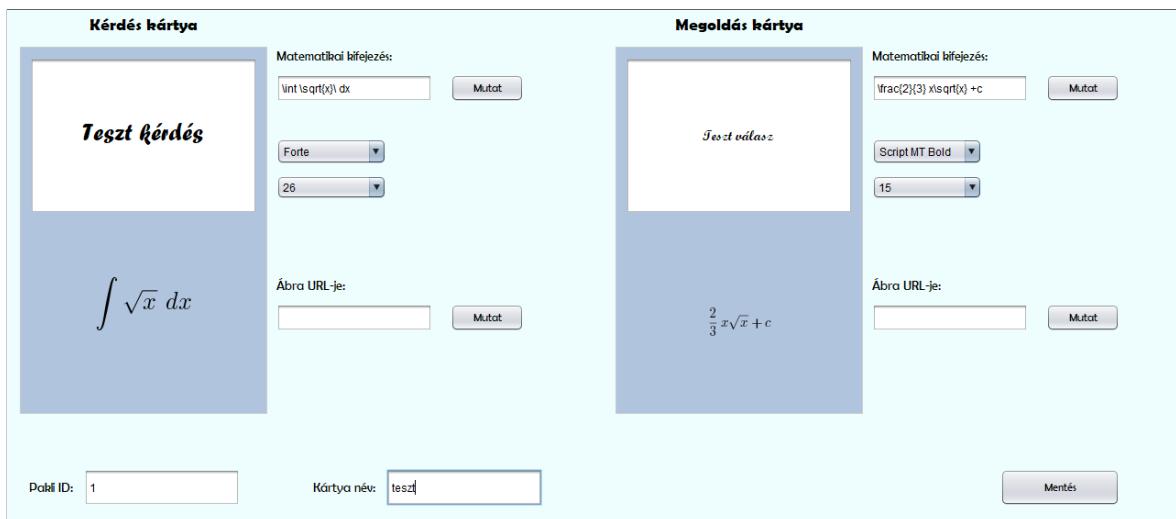
5.2. Kártyapakli készítése



5.15. ábra. Szövegszerkesztő felülete.

A következő ábrán (5.16 ábra) készítettem egy tesztkártyát. Először is legszembeígnőbb a betűméretek különbsége. Ahogyan már említettem meg lehet változtatni a betűméretet az adott lapon, amely ezáltal az egész lapra érvényes. Tehát ahogy látható, a *Kérdés kártyán* szereplő szövegek elég nagy, 26-os betűméretre vannak állítva, és mivel az egész lapra kihat a betűméret, így a L^AT_EX matematikai kifejezés is ezzel a betűmérettel jelenik meg. Ez a másik oldalon a *Megoldás kártyán* is szembeötlő, hogy ott pedig mennyivel kisebb a betűméret.

Ahogy látható, a betűtípusok is változtatva lettek, ezek azonban természetesen csak a begépelt szövegekre érvényesek, hiszen a L^AT_EX matematikai parancsoknak meg van a saját kinézetük, amelynek a betűtípusát nem lehet állítani.



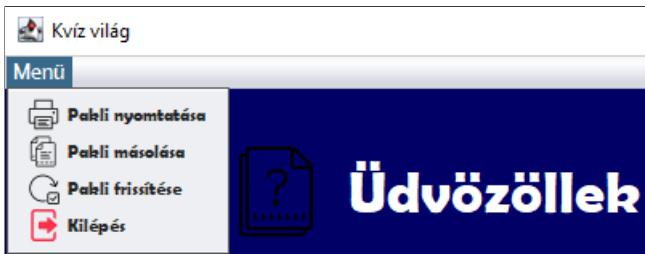
5.16. ábra. Tesztkártya készítése.

Észrevehető még, hogy természetesen megadtam a "Pakli ID"-t, valamint a "Kártya nevét" a bal alsó sarokban, hiszen enélkül nem lehetne menteni a kártyalapot. A "Mentés" gombra kattintva az elkészült kártya bekerül az adatbázisba.

5.2.1. Funkciók – Nyomtatás, Másolás, Frissítés

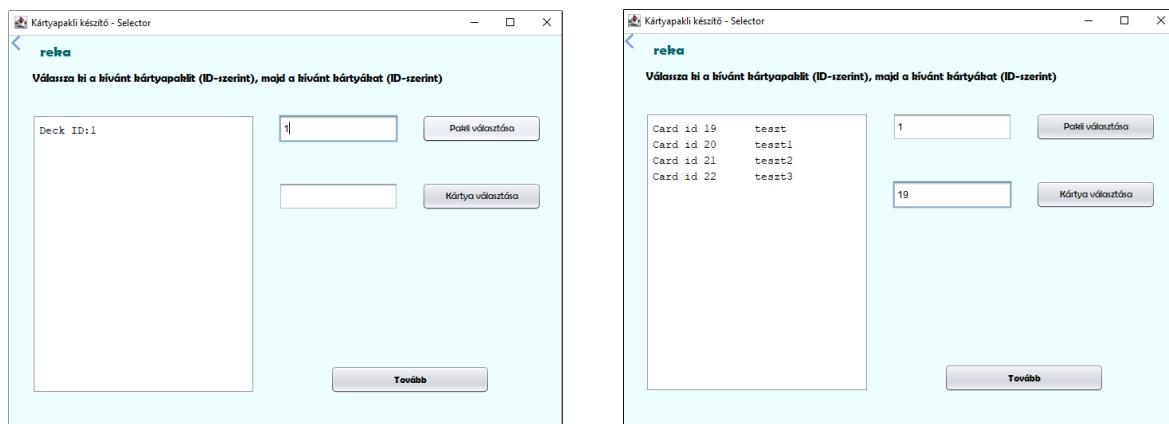
A következőkben a kártyapaklikon és kártyalapokon elvégezhető műveleteket fogom bemutatni.

Ezek a funkciók a játék választás felületen érhetők el, a legördülő menüsorban.



5.17. ábra. Legördülő menüsor: Kártyapakli készítő műveletei.

Először a **Pakli nyomtatásáról** beszélünk. Ha ezt az opciót választja a Játékos, akkor a **Selector** felülete kerül elő. A nevéből is adódóan itt kell kiválasztani, hogy mely paklit szeretné nyomtatni a Játékos.



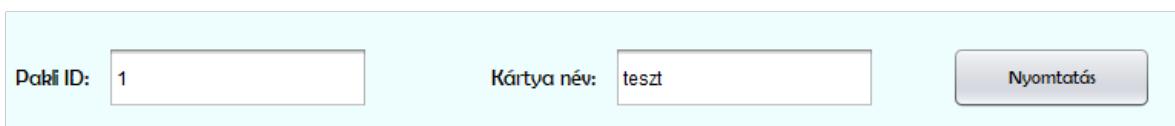
(a) Kártyapakli választása ID alapján.

(b) Kártyalap választása ID alapján
(a név segítségül látszódik).

5.18. ábra. A Selector felülete.

Ahogyan az ábrán is látszódik (5.18 ábra) a Játékosnak először a kártyapaklit kell kiválasztania. Azokból az adatokból választhat, amelyek az adatbázisban vannak, tehát csak olyan elem van kilistázva, ami létezik. A kártyapakli kiválasztása után, a kezdő kártyalapot is ki kell választani, szintén ID alapján.

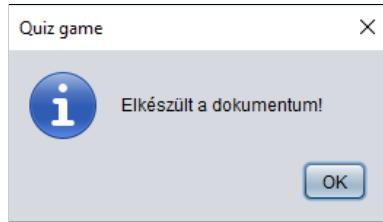
A "Tovább" gombra kattintva újra a **Selector** felülete kerül a Játékos előtt, azonban itt már nem a "Mentés" gombot látja, hanem a "Nyomtatás"-t.



5.19. ábra. Nyomtatás gomb a kártyák alatt.

Erre rákattintva néhány másodperc múlva el is készül a PDF dokumentum, amelyről párbeszédablak formájában üzenetet kap a Játékos (5.20 ábra).

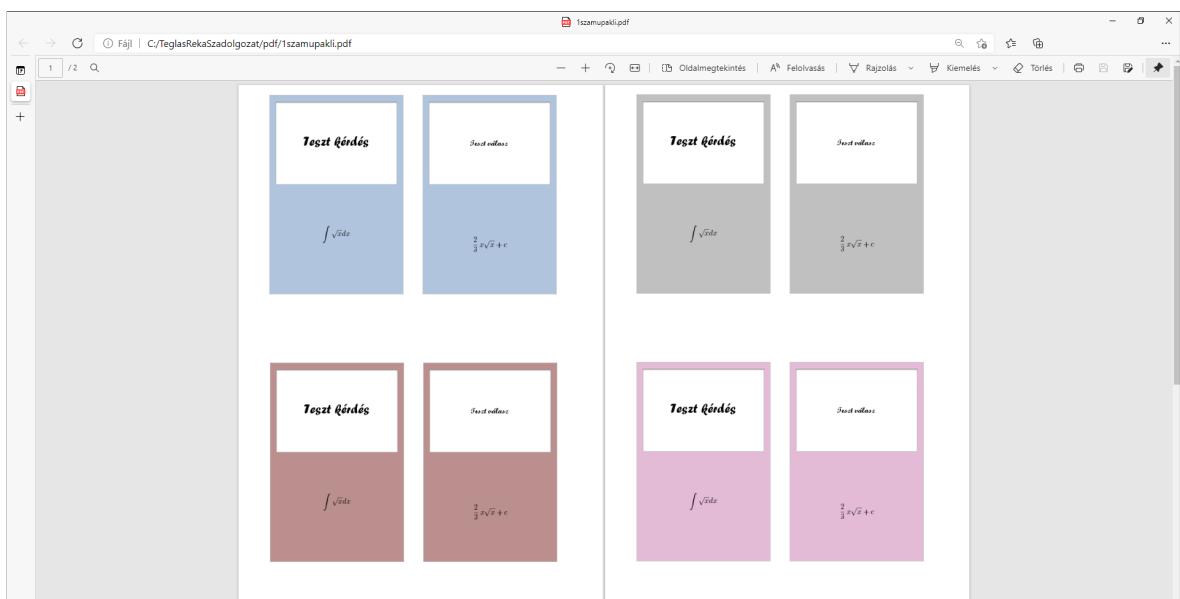
5.2. Kártyapakli készítése



5.20. ábra. Párbeszédablak dokumentum elkészültekor.

Itt látható az elkészült PDF dokumentum tartalma. A kártyák mérete általam úgy lett beállítva a kódban, hogy egymás mellett elférjen egy lapnak a két oldala, valamint egymás alatt is elférjen két lap, így 2 kártya fér minden két oldalával egy A4-es lapra.

Jelen példában azért készült 4 kártyalap, mert az 5.18 ábrán látható példa kártyapaklit nyomtattam, amelyben 4 kártyalap volt.



5.21. ábra. Az elkészült PDF tartalma.

A **Pakli másolása** során a Játékos elé ugyanazok a felületek kerülnek, mint a Pakli nyomtatása során – amelyről fentebb írtam. A Játékos itt igazából csak egy adott kártyát tud átmásolni egy másik pakliba, nincs az egész pakli másolására lehetőség. A Játékosnak ugyanúgy ki kell választania a kártyapaklit a **Selector** felületen, majd az adott kártyapakliban lévő kártyát, amit másolni szeretne.



5.22. ábra. Másolás gomb a kártyák alatt.

Ekkor a kiválasztott kártya megjelenik a Szövegszerkesztő felületen, ahol ezúttal "Másolás" gombot lát a Játékos. A másolás, vagy akár duplikálás úgy lehetséges, hogy a "Pakli ID" szövegmezőbe másik ID-t ad meg a Játékos, vagy ha ugyanazt az ID-t adja meg, amiben eredetileg volt az adott kártya, akkor abban a pakliban a "Másolás" gomb lenyomása után duplán fog szerepelni az a kártyalap. Ekkor jön jól a *Pakli frissítése*.

A **Pakli frissítése** funkció nem más, mint egy adott kártyalap szerkesztése, a megadott kártyapakliban. A Játékos előre ezúttal is a **Selector** felülete kerül, hogyha ezt a funkciót választja, hiszen ugyanitt kell kiválasztania a kártyapaklit, és a benne lévő adott kártyát, amit szerkeszteni szeretne – természetesen most is minden ID szerint. A Játékos által kiválasztott kártyalap visszakerül a Szövegszerkesztő oldalára, és a kiválasztott kártyalap kinézete lesz előtte.

Miután a szükséges módosításokat elvégezte – például kérdés átírása, vagy betűtípus átállítása – jelen esetben a "Szerkesztés" gombot fogja látni a kártyák alatt, ezzel tudja elmenteni az eredményt. Mivel ebben a funkcióban csak a kártyalapot lehet szerkeszteni, ezért ha a Játékos módosítja a "Pakli ID"-t, azzal lényegében egy másolást hajtana végre. Azonban mivel nem a "*Pakli másolása*" funkcióban van, így ez nem fog megtörténni, csak a kártyalapon végrehajtott módosítások fognak mentésre kerülni az adatbázisba. A kártyalap nem fog átmásolódni létező ID esetén, nem létező ID esetén pedig nem fog új pakli keletkezni a kártyalappal.

The screenshot shows a user interface for editing a card. On the left, there is a label 'Pakli ID:' followed by a text input field containing the number '2'. In the center, there is another label 'Kártya név:' followed by a text input field containing the text 'teszt3'. On the right side of the interface, there is a grey button labeled 'Szerkesztés' (Edit).

5.23. ábra. Szerkesztés gomb a kártyák alatt.

5.3. Tesztfuttatás

A játék tesztelése során nagyon sok hibás részletet tudtam feltárni, remélhetőleg minden. Több héten keresztül tartott ez a tesztelés, és nem csak én, de rajtam kívül több ember, többféle számítógépen és operációs rendszeren tesztelte a játékot.

Tapasztalataim alapján az a követelmény (2.3.2), pontosabban az a nem funkcionális követelmény teljesült, amely a hordozhatóságot érintette: az alkalmazás használható szinte minden személyi számítógépen, operációs rendszertől függetlenül.

Hatókonysági szempontból is megfelelőnek mondanám az alkalmazást, ugyanis az 50-200 MB közötti tárigény intervallum is megvalósult. A program futása során többször is figyeltem a tárigény adatait a Feladatkezelőben. A következő adatokat, és képeket onnan nyertem ki.

A játék úgymond "nyugalmi állapotban", amikor nem történik benne semmilyen interakció, kattintás, szöveg írása, vagy éppen adatbázis lekérdezés betöltése, akkor a következő memória mennyiséget kívánja:

Név	Állapot	Processzor	Memória	Lemez	Hálózat	GPU
OpenJDK Platform binary		0%	52,8 MB	0 MB/s	0 Mb/s	0%
Kvíz világ						

5.24. ábra. Használt memória "nyugalmi állapotban".

Bejelentkezés során, tehát amikor a programnak elsősorban el kell érnie az adatbázist, és ellenőrizni a belépéshez szükséges felhasználónévet és jelszót, a következő adatokat figyeltem meg:

Név	Állapot	Processzor	Memória	Lemez	Hálózat	GPU
OpenJDK Platform binary		5,7%	93,1 MB	0,1 MB/s	0 Mb/s	0%
Kvíz világ						

5.25. ábra. Használt memória bejelentkezéskor.

Aztán a kvízjáték indításakor már feljebb ugrott a memóriaigény:

Név	Állapot	Processzor	Memória	Lemez	Hálózat	GPU
OpenJDK Platform binary		14,3%	169,6 MB	0,1 MB/s	0,1 Mb/s	0%
Legyen Ön is milliomos!						

5.26. ábra. Használt memória a "Start" gombra nyomva.

Gombok kattintásakor, azaz a kérdésekre válaszoláskor visszább esett ez érték valamennyivel:

Név	Állapot	Processzor	Memória	Lemez	Hálózat	GPU
OpenJDK Platform binary (2)		0,2%	146,5 MB	0 MB/s	0 Mb/s	0%
Legyen Ön is milliomos!						

5.27. ábra. Használt memória válaszadás közben.

A segítségek használatánál tapasztaltam még kiugró értéket:

Név	Állapot	Processzor	Memória	Lemez	Hálózat	GPU
▼ OpenJDK Platform binary (2)		2,9%	182,3 MB	0 MB/s	0 Mb/s	0%
Legyen Ön is milliomos!						

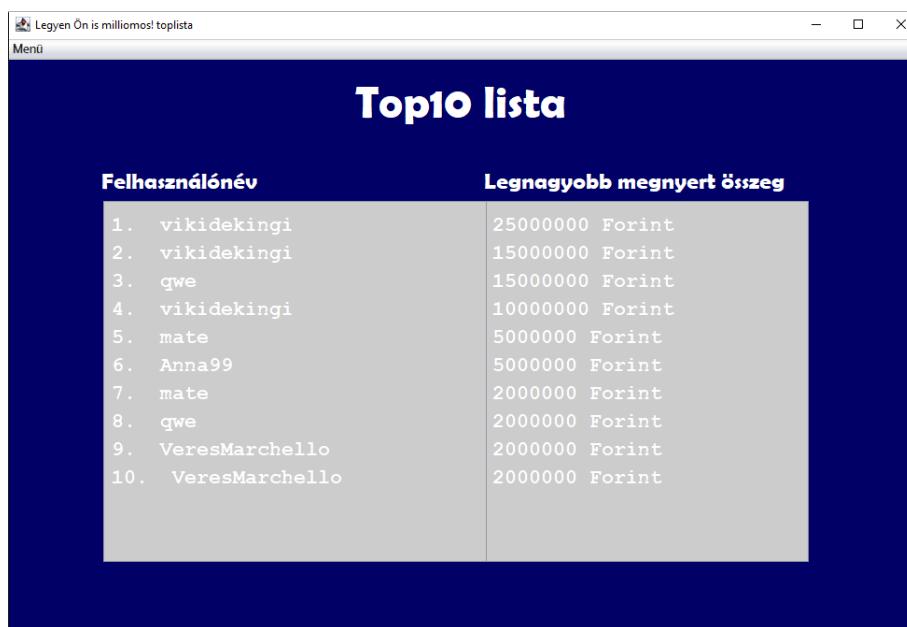
5.28. ábra. Használt memória *Felező segítség* használata közben.

Azonban a megadott intervallumot nem lépte túl egyetlen alkalommal sem, azaz mindenkor volt a memóriaigénye a programnak. A játék futása során csak a segítségek használatakor voltak kiugró értékek. Egyébként a program használata közben átlagosan 140-160 MB közötti volt a használt memória.

5.3.1. Tesztelés ismerősökkel

Az általam végzett tesztelések mellett és után néhány egyetemi hallgató ismerősömet kértem meg, hogy teszteljék a játékot.

Voltak észrevételeik magával a programmal kapcsolatban, amelyeket javítottam is a legjobb tudásom szerint. Most viszont az eredményeiket mutatnám meg, hogy mennyire is teljesíthető a játék.



The screenshot shows the Windows Task Manager with one task listed:

Név	Állapot	Processzor	Memória	Lemez	Hálózat	GPU
OpenJDK Platform binary (2)		2,9%	182,3 MB	0 MB/s	0 Mb/s	0%

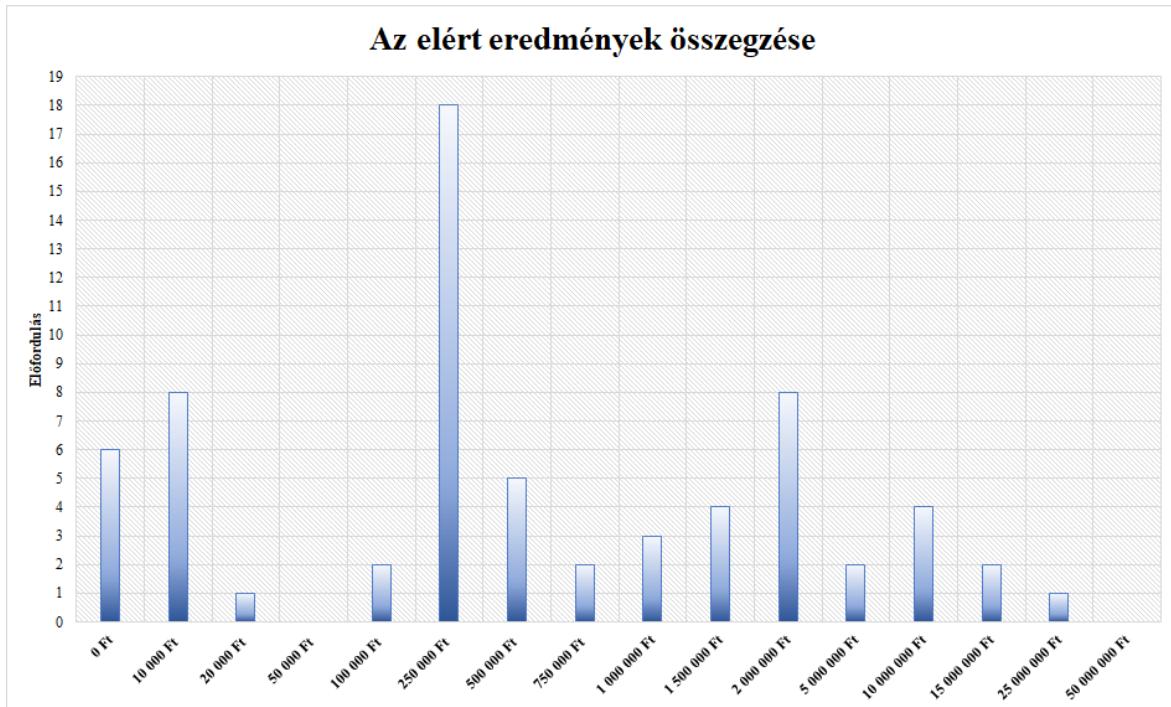
Below the Task Manager, a window titled "Legyen Ön is milliomos! toplistá" displays a "Top10 lista".

Felhasználónév	Legnagyobb megnyert összeg
1. vikidekingi	25000000 Forint
2. vikidekingi	15000000 Forint
3. qwe	15000000 Forint
4. vikidekingi	10000000 Forint
5. mate	5000000 Forint
6. Anna99	5000000 Forint
7. mate	2000000 Forint
8. qwe	2000000 Forint
9. VeresMarchello	2000000 Forint
10. VeresMarchello	2000000 Forint

5.29. ábra. Toplista felülete.

A fentebbi ábrán látható Top10 lista, amely a Ranking felülete (4.3.2 alfejezet), és amely működéséről a 4.3.3 GameAlgorithms osztály alfejezetben írok.

A következő ábrán (5.30) pedig a Tesztelők által elérte eredmények összegzése található. Ahogy látható, a legtöbb elérte eredmény a 250 000 Forint volt, amely megfelel az első szakaszkérdésnek. Ebből azt a következtetést tudom levonni, hogy a többség elérte, és tovább ment az első szakaszkérdésen, viszont utána már kevésbé tudta a válaszokat, azonban inkább nem állt meg, mert tudta, hogy a szakaszkérdés összege fix.



5.30. ábra. Elért eredmények összegzése.

A második leggyakoribb érték a 10 000 Forint, valamint a 2 000 000 Forint volt. Ezek közül a 2 000 000 Forint a második szakaszkerdés, azaz elég nagy számban elértek idáig is Tesztelők.

A legmagasabb összeg, amit valaki elérte, az a 25 000 000 Forint volt, sajnos azonban a főnyereményig egyetlen játékos sem jutott el.

6. fejezet

Összefoglalás

A szakdolgozatom a nyári szakmai gyakorlaton kapott feladat továbbfejlesztésének tekintetében. Egy olyan alkalmazás létrehozása volt a cél, amelynek egyik részeként egy kártyapakli készítő programot kellett írnem, ez a feladat bővült ki később a kvízjáték ötletével és megvalósításával.

A szakdolgozatomban részletesen dokumentáltam a fejlesztési folyamatot, a játék koncepciójának kitalálásától kezdve, a program megtervezésén át, egészen a tesztelésig. A hosszú hónapok alatt, amelyeket a szakdolgozatom elkészítésével töltöttem, rengeteg új és érdekes dolgot tanultam.

A programkód megírása közben mélyebben megismerkedtem a Java nyelvvel, valamint új ismereteket szereztem a Java Maven projekt menedzszerrel eszközeiről. Szintén új ismereteket szereztem a NetBeans integrált fejlesztői környezetről, hiszen ebben fejlesztettem a programomat. Nem csak a kódolás, de a tesztelés és hibajavítás közben is nagyon sokat tanultam. A munkámat alapvetően sikeresnek gondolom, hiszen minden olyan funkciót meg tudtam valósítani, amelyet elvárásoknak megfogalmaztam magamnak. Az általam elkészített alkalmazás átlátható, könnyen kezelhető, és minden mellett fejlesztő jellegű, a kérdések matematikai témája miatt. A kvízkérdések segítségével például szórakoztatás módon lehet megismerkedni az analízis fogalmaival.

Természetesen az elkészült alkalmazás esetén is igaz az, hogy tovább fejleszthető, extra funkciókkal bővíthető. A munkám során több ilyen lehetőség is eszembe jutott, de a rendelkezésre álló idő rövidisége miatt nem tudtam megvalósítani minden. Elsősorban a kvízjátékot lehetne gazdagítani azzal az opcionális, hogy a Játékos kiválaszthatná, milyen témakörből szeretné a kérdéseket kapni. Így nem csak matematikai kérdéseket lehetne megválaszolni, hanem általánosabb témák közül is lenne lehetőség választani a játék indítása előtt, mint például sport, informatika, fizika, biológia, kémia, történelem, irodalom és így tovább. A kiválasztott témakörön belül is létrehozhatnánk kategóriákat, melyekkel a Játékos specifikus tudását lehetne tesztelni, például, ha a jelenleg is létező matematika témakört különböző területekre bontanánk, így külön csoportot képeznénk például az optimalizálással vagy valószínűségszámítással kapcsolatos kérdések.

A kártyapakli készítő részt is lehetne további funkciókkal bővíteni, több lehetőséget adva a felhasználó kezébe azáltal, hogy megtervezhesse a számára legtetszetősebb és legpraktikusabb elrendezést nyújtó kártyalapot.

Összességében úgy gondolom, hogy sikerült teljesítenem a szakdolgozati feladat célkitűzését, és egy olyan alkalmazást készítettem, mely tanulási és szórakozási célra egyaránt jól használható, a kártyapakli készítő programrész pedig további játékok fejlesztésére is lehetőséget nyújt.

Irodalomjegyzék

- [1] Dr. Baksáné Dr. Varga Erika egyetemi docens jegyzete. *Objektum orientált programozás*. Miskolci Egyetem, 2019.
- [2] Szűcs Miklós mesteroktató jegyzete. *Objektum orientált programozás gyakorlat*. Miskolci Egyetem, 2019.
- [3] Pénzes László. Elméleti alapozás – a Java-nyelv rövid története és komponensei. http://www.informatika-programozas.hu/informatika_java_programozas_elmelet_java_tortenet.html, 2021.
- [4] Dr. Olajos Péter jegyzete. *Szövegszerkesztés – E^AT_EX*. Miskolci Egyetem, 2019.
- [5] Ferenc Wettl, Gyula Mayer, and Péter Szabó. *E^AT_EX kézikönyv*. Panem, Budapest, 2004.
- [6] Zsemlye Tamás. Java fejlesztői környezetek. <http://www.biga.hu/java/javaide.htm>, 2021.
- [7] The Apache Software Foundation. Netbeans tutorials. <https://netbeans.apache.org//kb/>, 2021.
- [8] Dr. Mileff Péter PhD egyetemi docens jegyzete. *Szoftvertechnológia*. Miskolci Egyetem, 2019.
- [9] Dr. Mileff Péter, Ficsor Lajos, Krizsán Zoltán. *Szoftverfejlesztés*. Miskolci Egyetem, Általános Informatikai Tanszék, 2011.
- [10] Giachetta Roberto. Szoftvertechnológia. https://people.inf.elte.hu/groberto/elte_szt/eloadas_anyagok/elte_szt_ea02_dia.pdf, 2017.
- [11] The Apache Software Foundation. Maven in 5 minutes. <https://maven.apache.org/guides/getting-started/maven-in-five-minutes.html>, 2021.
- [12] Menyhárt László Gábor. Webfejlesztés. http://xml.inf.elte.hu/archive/2010_11_2/webfejl4/JavaTut_110503.pdf, 2011.
- [13] Ognjen Bubalo. Programozói könyvtárelemző szoftver készítése java alkalmazásokhoz (bsc szakdolgozat). https://people.inf.elte.hu/legendi/resources/theses/OgnjenBubalo_DepChecker.pdf, 2011.
- [14] RTL Klub Magyarország. A Legyen Ön is milliomos játékszabálya. https://rtl.hu/rtlklub/milliomos/cikk/2011/11/17/a_legyen_on_is_milliomos_jatekszabalya, 2021.

Adathordozó használati útmutató

A szakdolgozatomhoz egy darab DVD lemez tartozik, mint melléklet, amelyen a következő fájlok találhatóak:

- A szakdolgozat PDF formátumban: SzakdolgozatTeglasReka.pdf
- Hasznalati.pdf használati útmutató a lemezhez.
- A **TeX_source** jegyzékben található a szakdolgozat TeX formátumban, és a fordításhoz szükséges forrás állományok is.
- A **Projekt** jegyzékben a program minden forráskódja, és az ahhoz szükséges fájlok.
- A **TeglasRekaSzakdolgozat** jegyzékben a futtatható program: KvizJatek.jar, valamint egy README.txt fájl és egy start.bat fájl.