Eötvös Loránd Tudományegyetem

Informatikai Kar

Komputeralgebra Tanszék

Hibatűrő keresőalgoritmusok és kódolás

Burcsi Péter
Egyetemi docens, PhD

Hortobágyi Mónika
Programtervező Informatikus BSc

Budapest, 2018

Tartalomjegyzék

1. fejezet

Bevezetés

Az egyetemi tanulmányaim során a kódolás témakör volt az, ami elsőre igazán megfogott. Így nem volt nehéz kérdés, hogy milyen témájú szakdolgozatot szeretnék írni.

A dolgozat célja néhány hibakorlátozó kódolással kapcsolatos algoritmus implementálása, illetve ezek alkalmazása az ún. hibatűrő keresési feladatok megoldásában. A szoftver megvalósítja a kapcsolódó ún. Rényi-Ulam-játékot is.

A Rényi-Ulam-játék a barkochba játékra épül. A barkochba játéknak többfajta változata létezik. A szoftver csak az egész számokkal való játékra tér ki. A Rényi-Ulam-játék esetében is 2 játékos van. Előre megbeszélik, hogy mi az az intervallum, amiből az egyik játékos gondolhat egy számra. A másik játékos pedig olyan kérdéseket tesz fel neki, amire *Igen/Nem* válaszokat adhat.
A játék sajátossága, hogy a játékmester - aki gondolt a számra - hazudhat[1] is. Így előre már nem csak az intervallumot, hanem a hazugságok maximum számát is meg kell határozniuk. Mivel előre nem tudjuk, hogy hány kérdést kell feltenni a gondolt szám kitalálásához, így azt sem lehet garantálni, hogy a játékmester az összes hazugságát elhasználja.

[1] A hazugság azt jelenti, hogy ha *Igen*-t kellett volna válaszolni, akkor *Nem*-et mond helyette vagy a fordított esetben, *Nem* helyett mond *Igen*-t.

1.1. Rényi-Ulam-probléma

Stanisław Ulam az alábbi problémát elemezte:

> Valaki gondol egy egész számra 1 és 1.000.000 között. Egy másik ember maximum 20 kérdést tehet fel neki, amelyre a másik fél csak Igaz/Hamis válaszokat adhat. Természetesen a szám az első kérdésnél is megtippelhető: A szám az első fél millióban van? Ezek után megint lecsökkentheti a tartományt a következő kérdésben a felére és így tovább. Végül a szám kisebb lesz mint $\log_2(1000000)$.
>
> Tegyük fel, hogy aki a számra gondolt az egyszer-kétszer hazudhatott. Utána hány kérdést kéne a másiknak feltennie, hogy megkapja a helyes választ? A kérdezőnek biztosan szüksége lesz több mint n kérdésre ahhoz, hogy találgasson 2^n szám közül, mivel a kérdező nem tudja mikor hazudik a válaszadó.
>
> Ez a probléma általában nem megoldható. [1]

Rényi Alfréd nagyon hasonló problémát fogalmazott meg:

> „ ... A játéknak a következő változatát gondoltam ki, amelyet „Hazudós barkochba játék"-nak neveztem el. Tegyük fel, hogy meg van szabva, hány kérdéssel kell a gondolt dolgot kitalálni és a felelőnek jogában áll bizonyos meghatározott számú esetben hazudni; a kérdező persze, nem tudja, hogy melyik válasz igaz és melyik nem; a felelő, persze, nem köteles annyiszor hazudni, ahányszor ez meg van engedve: hazudhat kevesebbszer is. Például, ha csak két dolog valamelyikére lehet gondolni, és legfeljebb egy hazugság van megengedve, akkor három kérdésre van szükség. ..." [2]

> „ ... Ha négy dologra lehet gondolni és egy hazugság van megengedve, akkor, mint könnyen belátható, öt kérdésre van szükség. Ha két vagy több hazugság van megengedve, akkor már elég bonyolult a minimálisan szükséges kérdések számának meghatározása. Azt hiszem, a hazudós barkochba, ha több hazugság van megengedve, akkor játéknak túl bonyolult ..." [2]

A két felvázolt problémát együttesen nevezzük Rényi-Ulam-problémának.

2. fejezet

Megoldási terv

Az 1.1 szakaszban felvázolt probléma alapján pedig így néz ki egy általános játék: [3]

Fixálunk egy listát -az intervallumot reprezentálja-, ennek kezdete mindig 1, a vége a felhasználó által megadott szám a „The end of the interval" kérdésre. Emellett van egy $e \geq 1$ szám, ami a hazugságok száma. Az egyszerűség kedvéért a felhasználó által megadott maximum hazugságok számát jelöljük e_n-nel. A játékmester által gondolt számot pedig az X jelöli.

Ha megengednénk az $e = 0$ esetet, akkor egy klasszikus bináris keresést kéne alkalmazni. Ebben az esetben a játékos lemondhat a kérdése által feltett halmaz egyik részéről, attól függően, hogy a válaszadó *Igen*-t vagy *Nem*-et válaszol.
Az $e_n \geq 1$ esetben a játékos csak akkor zárhat ki egy számot, ha arra a játékmester e_n-nél több alkalommal hazudott. Minden más szám még lehetséges.

Tehát a játékosnak fel kellene jegyeznie, hogy melyik számra hányszor hazudhatott a válaszadó. Ehhez használjunk egy úgynevezett **hazugságvektor**-t, ami a következőképpen nézzen ki:

$$[\{e = 0 \text{ számok}\}, \{e = 1 \text{ számok}\}, ..., \{e = e_n \text{ számok}\}]$$

Az „$\{e = 0$ számok$\}$" elnevezés arra utal, hogy az ott szereplő számokra még egyszer sem hazudott a válaszadó. Folytatva a logikát az „$\{e = e_n$ számok$\}$" pedig azt jelenti, hogy az ottani számokra **pontosan** e_n**-szer** hazudott már. Akkor mondhatjuk, hogy biztosan tudjuk melyik számra gondolt a játékmester, ha nem lesz olyan szám, amire még nem hazudott és **pontosan egy** olyan lesz, amire már legalább egyszer hazudott, de maximum e_n-szer. Ekkor a válaszoló pontosan erre a számra gondolt, ha tartotta magát a játékszabályokhoz.

A játék minden esetben a játékos kérdésével indul, amit egy válasz követ a játékmestertől majd ismét egy kérdés a játékostól és így tovább, egészen addig amíg a játékos kérdése már egy kijelentés nem lesz, miszerint tudja melyik számra gondolt a játékmester.

Most pedig nézzük meg, hogy mi alapján keletkezik, változik a hazugságvektor.

1.lépés Ekkor a játékos még nem tett fel kérdést.

A hazugságvektor $e = 0$ részében szerepelnek 1-től az intervallum végéig a számok, a többi részlista üres.

Például: Ha 10 az intervallum vége és $e_n = 2$, akkor így néz ki a hazugságvektor a kezdeti állapotban:

$$[\{1, 2, 3, 4, 5, 6, 7, 8, 9, 10\}, \{\}, \{\}]$$

2.lépés A játékos felteszi a kérdést.

Ahhoz, hogy fel tudja tenni a kérdést a hazugságvektor részlistáit el kell feleznie külön-külön és ezeknek az unióját vennie. Páratlan hosszú részlista esetén a darabszám felső egész része legyen véve.

Az előző példát folytatva a felbontás így nézne ki:

$$[\underbrace{(\{1, 2, 3, 4, 5\}, \{6, 7, 8, 9, 10\})}_{\text{e = 0 számok}}, \underbrace{(\{\}, \{\})}_{\text{e = 1 számok}} , \underbrace{(\{\}, \{\})}_{\text{e = 2 számok}}]$$

Innen már leolvashatóak a kérdés feltételéhez az intervallumok, az első részlista esetén ez az [1..5] intervallum, míg a másik két részlistában nincsenek elemek. Tehát így a kérdés:

$$X \stackrel{?}{\in} [1..5]$$

3.lépés A játékmester válaszol.

A válaszadónak el kell döntenie, hogy *Igen*-t vagy *Nem*-et válaszol. Először jobb ha azt állapítja meg, hogy mi lenne az igaz -hazugságmentes- válasz a kérdésre. Majd ezután eldönti, hogy él-e a hazugság lehetőségével vagy sem. A végleges válaszát pedig megosztja a játékossal is.

4.lépés Egy kör lezárulása.

A játékos már ismeri a játékmester válaszát, így meg tudja állapítani, hogy hogyan néz ki a hazugságvektor.

Az eddigi példa alapján:

- Ha a válasz *Igen* volt, akkor: $[\{1, 2, 3, 4, 5\}, \{6, 7, 8, 9, 10\}, \{\}]$
- Ha a válasz *Nem* volt, akkor: $[\{6, 7, 8, 9, 10\}, \{1, 2, 3, 4, 5\}, \{\}]$

Ezzel lezárul egy kör, az új kör a *2. lépéstől* indul újra.

3. fejezet

Felhasználói dokumentáció

A program a Rényi-Ulam-probléma játékba átültetett megvalósítása. A játékban két különböző mód érhető el. Az egyikben a felhasználó a játékos, azaz ő próbálja kitalálni, hogy a gép melyik számra gondolhatott. A másik módban pedig fordítva vannak a szerepek, a felhasználó a játékmester, Ő gondol egy számra és a gép próbálja meg ezt kitalálni különböző kérdésekkel.

3.1. A játékprogram indítása

A felhasználónak elég a mappában Windows 10 64 bit-es operációs rendszer [4] esetében a *Barkochba.exe* fájlra kattintani és a program elindul, Ubuntu 16.04 64 bit-es operációs rendszer [5] esetén a *Barkochba.out* fájlra kell kattintani. Fontos megjegyezni hogy az adott mappában kell lennie *realLogo.png* fájlnak is.
A kép hiányában a program el tud indulni, de a szoftverben használt képet és a játék logóját nem tudja megjeleníteni. A program ezen felül egy lokális adatbázist használ, amit a korábbi fájlokkal megegyező mappában helyez el az első elindítás alkalmával.

A játékhoz egy logó is készült, aminek szintén a program többi fájljával egy könyvtárban kell lenni.

3.1. ábra. A játék logója

A program kinézete operációs rendszer specifikus, azaz máshogy néz ki egy Linux rendszeren, mint egy Windows rendszeren. A 3.2a képen egy Ubuntu 16.04-es kinézet látható, míg a 3.2b képen egy Windows 10-es kinézet.

(a) Ubuntu 16.04

(b) Windows 10

Az ablak minden esetben átméretezhető, saját igény szerint állítható. A program nevével megegyező sorban, az ablak keretén a gombok funkciói: az ablak tálcára rakása, a teljes képernyős mód aktiválása/inaktiválása, a program végleges bezárása.

Visszalépni az előző ablakra minden esetben a Back gomb megnyomásával lehetséges, ha ez szerepel az ablakon. Továbblépéshez pedig az OK gomb megnyomása szükséges.

Figyelem! Egy megkezdett játék esetén a program végleges bezárásakor a felhasználó által megadott adatok az Eredménytáblán szerepelni fognak, ilyenkor a „Not finished" szöveg lesz látható majd.

3.2. Főmenü

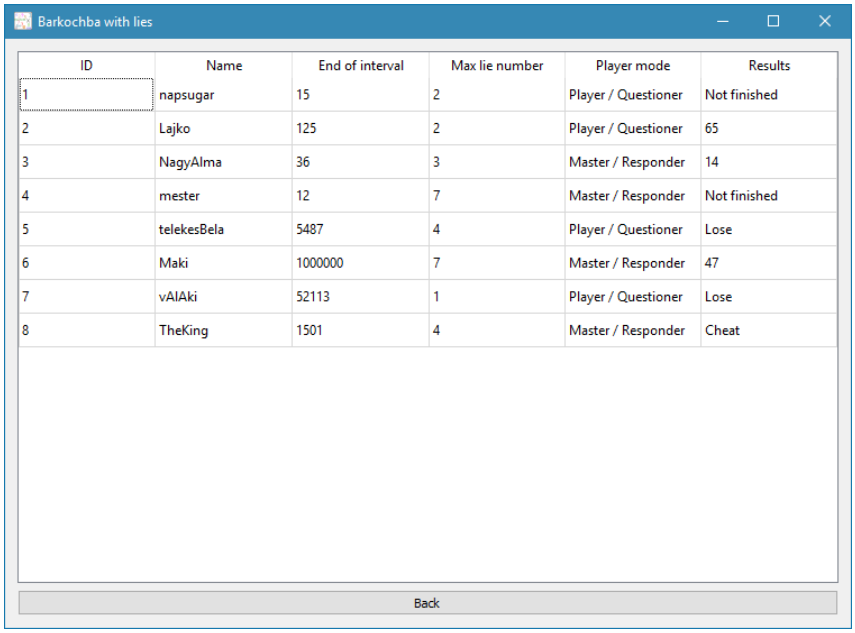
A program elindításával megjelenik az első ablak (lásd 3.2a és 3.2b ábra). Ezen olvasható a készítő neve, alatta a játék logója, valamint alul két gomb New Game és Score felirattal.

A New Game gomb megnyomása esetén egy új ablak látható, amin a felhasználónak néhány kérdésre kell válaszolni. Ezt részletezni a 3.4 alfejezetben fogjuk.

A Score gomb választása esetén pedig az Eredménytábla jelenik meg. Erről bővebben a 3.3 alfejezetben lehet olvasni.

3.3. Eredménytábla

Ezen az ablakon az összes olyan eredmény szerepel, amit az adott gépen, az adott felhasználói fiók alatt indítottak el. A 3.3 kép alapján látható, hogy az adatbázis 6 értéket tárol egy felhasználóról.



3.3. ábra. Eredménytábla

Fontos megjegyezni, hogy egy név szerepelhet többször is az Eredménytáblán, ugyanis a felhasználók azonosítása nem a Name mező alapján történik.

Az Eredménytábla 1. oszlopa a program számára szolgál, ezáltal azonosítja az adott felhasználót. Minden megkezdett kör után új ID jön létre. A 2. – 5. oszlopban láthatóak azok az adatok, amiket a felhasználó az Új játék fülön megadott. A 6. oszlopban látható a játék végeredménye. Egy játéknak több kimenetele lehet:

- A felhasználó nyert vagy fair módon játszott.
  Ekkor az oszlopban a kitaláláshoz szükséges kérdésszám jelenik meg.
  Player mód esetén ez akkor következik be, ha ki tudja találni, hogy melyik számra gondolt a gép.
  Master mód esetén pedig akkor, ha a gép eltalálta, hogy melyik számra gondolt a játékos.

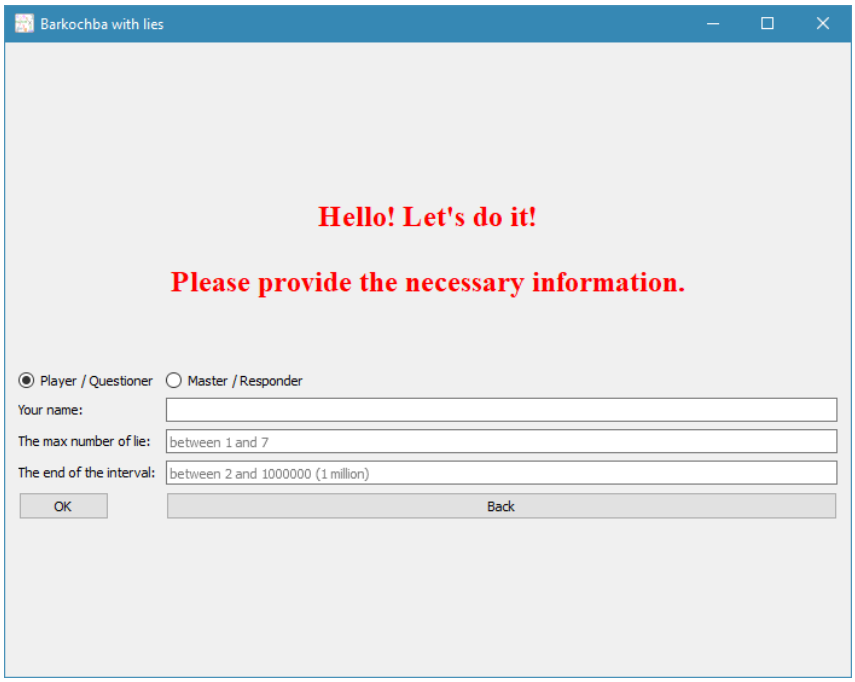
- A felhasználó veszít.
  Ilyenkor a „Lose” felirat látható az oszlopban.
  Player mód esetén akkor veszít, ha nem találja el, hogy melyik számra gondolt a gép.

- A felhasználó csalt.
  Ekkor az Eredménytáblán a „Cheat” szöveg jelenik meg.
  Master mód esetén akkor, ha nem a szabályoknak megfelelően játszott.

- Nem fejezte be a játékot.
  Be nem fejezett játék esetén a „Not finished” szöveg olvasható. Amennyiben az adatokat már megadta és rákattintott az OK gombra akkor az egy megkezdett körnek számít és így a beírt adatok megjelennek az Eredménytáblán.

3.4. Új játék

Az ablak tetején olvasható egy „köszöntő szöveg”, amivel egyben a felhasználót a játékra is buzdítja. Valamint megkéri, hogy adja meg a játék paramétereit, azaz töltse ki a mezőket. A felhasználónak ezen az ablakon 4 kérdésre kell válaszolni.(lásd 3.4 ábra)



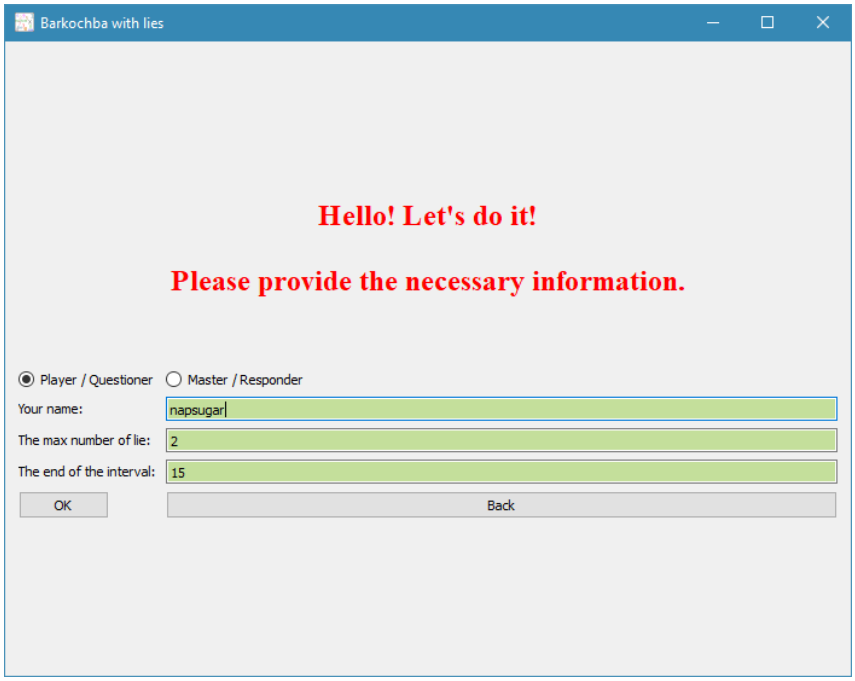
3.4. ábra. Új játék

A programban alapértelmezetten a beviteli mezők háttérszíne fehér, amennyiben a felhasználó helyes formátumú értéket ír be, akkor az átvált zöld színre.

Először a játékmódot kell kiválasztani. Itt két lehetőség közül választhat: Player/Responder vagy Master/Questioner. Egyszerre csak egy játékmód választható ki. A program alapértelmezetten a Player módot jelöli meg.
A Player (játékos) mód akkor javasolt, ha a felhasználó szeretné kitalálni a gép által gondolt számot. Azt, hogy ehhez milyen kérdéseket tehet fel és hogyan tud ebben a módban játszani az a 3.5 részben van bemutatva. Master (játékmester) mód esetén a felhasználónak kell egy számra gondolnia, amit a gép próbál majd kitalálni. Erről a 3.6 részben írunk bővebben.

A következő mezőben a nevet kell megadni, amivel majd az Eredménytáblán szeretne a felhasználó szerepelni. A név az angol ABC kis- és nagybetűit tartalmazhatja. Speciális karakterek, számok nem megengedettek.

Ezek után pedig már csak az maradt kérdéses, hogy maximum hány hazugságos játékot szeretne játszani és mekkora intervallumon. A maximum hazugság $[1, 7]$ közötti egész szám lehet. Az intervallum mindenképpen 1-gyel kezdődik, így itt a lehetséges legkisebb szám a 2 míg a felső korlát az 1.000.000. A 3.5 ábrán a helyes kitöltésre látható egy példa. Jól látszik, hogy minden mező háttérszíne zöld és a játékmódnál is az egyik előtt telt a kör.

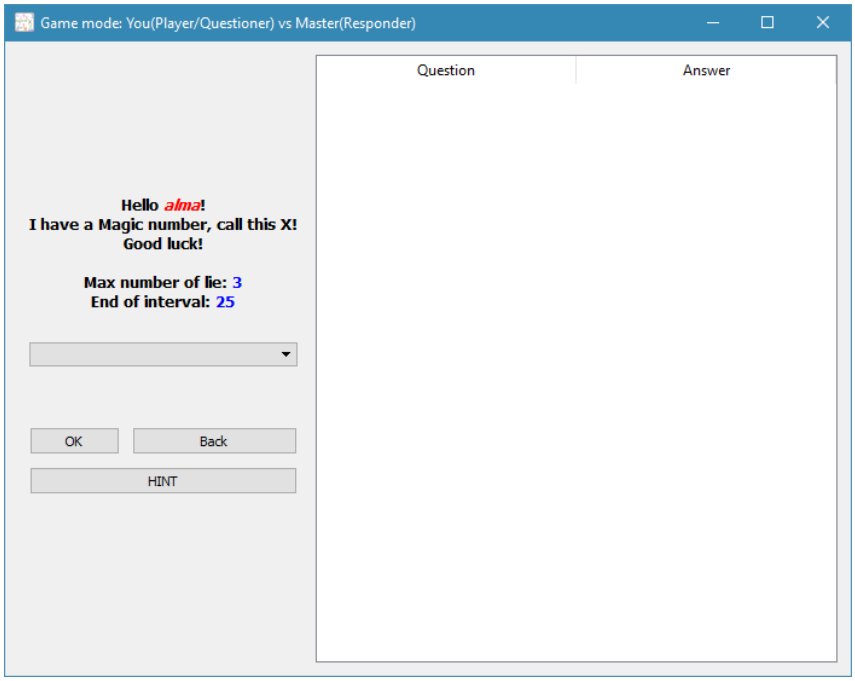


3.5. ábra. Példa a helyes kitöltésre

Az OK gomb megnyomásával véglegesíteni lehet az eddig beírtakat. Amennyiben nem megfelelő adatot írt be, vagy éppen üresen hagyott egy mezőt, akkor a program nem engedi tovább a felhasználót, amíg megfelelő értékek nem lesznek a mezőkben. A program működése közben felmerülő hibaüzeneteket a 3.7 alfejezetben tárgyaljuk.
Back nyomása esetén pedig ahogyan már mondtuk az előző ablakra lehet visszalépni, jelen helyzetben ez a Főmenüt jelenti.

3.5. Játékos mód

Ha a felhasználó a Player játékmódot választotta, akkor az alábbi kép fog előtte megjelenni.



3.6. ábra. „Player” mód

A program bal oldalán a kérdező felület látható, míg jobb oldalt egy táblázat foglal helyet, amiben a feltett kérdések és a gép válaszai láthatóak. Bal oldalt leolvashatók a játékos adatai is. A gép ugyanis köszönti és elmondja, hogy Ő már kitalálta a „mágikus szám”-ot, azaz gondolt egy számra. Ezzel együtt a játékosnak megmutatja, hogy az előző ablakon, milyen paramétereket adott meg a játékhoz. Tehát a hazugságok maximum számát és az intervallum felső határát. A jobb olvashatóság és átláthatóság érdekében a szövegben a fontos adatokat más színnel jeleníti meg.

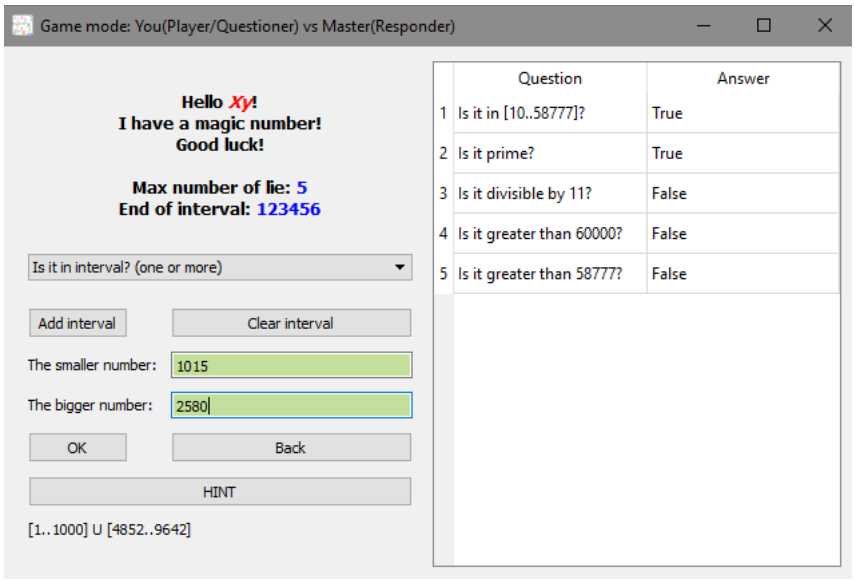
3.5.1. Kérdező felület

Az adatok alatt látható egy lenyitható fül, itt a játékos által feltehető kérdések olvashatók. Az alábbiak közül választhat:

- Is it in interval? (one or more) (lásd 3.7 ábra)
  Ebben az esetben feltehet egy vagy több intervallumra vonatkozó kérdést. Egy intervallum esetén csak a megjelenő két mezőt kell helyesen kitölteni, a felső mezőbe a kisebb számot, míg az alsóba a nagyobbat kell beírni.
  Több intervallum esetén miután megadta a játékos az első intervallum számait az Add interval feliratú gombra kell kattintania. Ekkor egyszerre két dolog fog megváltozni a képernyőn. Az eddigi kitöltött két mező -ahol a két számot adta meg- üres lesz, ezzel egyetemben a megadott számok a HINT feliratú gomb alatt megjelen-

nek. Amennyiben már több intervallumot megadott akkor ∪ jellel összekapcsolva jelennek meg az intervallumok.

Amennyiben a játékos felcserélve adta meg az intervallum határait nincsen semmi gond. A háttérben működő program ezt kicseréli és a megfelelő formátumban kérdezi meg. Ez látható a Válaszadó felületen a Question oszlopban és több intervallum esetén a lent megjelenő szövegben is. Így emiatt nem kell kitörölni a már megadott intervallumokat.
Intervallumok egyenkénti törlésére nincsen lehetőség. A Clear interval gombra kattintva van lehetőség az összes eddigi intervallumot kitörölni. Sikeres törlés esetén észrevehető, hogy az eddigi intervallumokról a szöveg üres lesz, így ott nem jelenik meg semmi sem.



3.7. ábra. Intervallum(ok) megadása, törlése

- Is it greater than ...?
  Ebben az esetben már csak egy értéket kell megadni. Azt a számot, amihez viszonyítva szeretné tudni a játékos, hogy a gondolt szám **szigorúan nagyobb-e**, tehát az egyenlőség nincs megengedve.

- Is it smaller than ...?
  Az előző lehetőséghez hasonlóképp működik ez is, csak itt a **szigorúan kisebb** esetet vizsgálja meg.

- Is it divisible by ...?
  Szintén egy szám adható meg. Itt az oszthatóságra kérdez rá.

- Is it prime?
  Itt nem kell számot megadni. Egyszerűen felteszi azt a kérdést, hogy a gondolt szám prím-e[1].

- Is the number of digits smaller than ...?
  Ismét egy számot kell megadni. A számjegyek összegéhez viszonyítva nézi, hogy a kérdezett szám **szigorúan kisebb-e**, mint a gondolt szám számjegyeinek az összege.

- Is the number of digits greater than ...?
  Itt is ugyanaz a helyzet, mint az előző esetben csak a **szigorúan nagyobb-e** kérdés lesz feltéve.

- I ask! Is the magic number ...?
  Ebben az esetben a játékos pedig felteheti a nagy kérdést, azaz valóban arra a számra gondolt-e a gép, mint amire szerinte. Ennek két kimenetele lehet. Az egyik miszerint a gép azt válaszolja, hogy „You win!", azaz valóban kitalálta a számot. A másik esetben pedig, amikor nem találja ki a „You lose! The number is..." szöveg jelenik meg, ahol a ... helyére a gép által gondolt szám fog kerülni. A kérdés feltétele után mindenképp véget ér ez a kör játék és az eredménnyel megtekinthető lesz az Eredmény táblán.

Halmazra vonatkozó kérdést úgy lehet feltenni, hogy kiválasztja a „Is it in interval? (one or more)" lehetőséget. Itt a kisebb és a nagyobb számhoz is a kérdezni kívánt halmaz egy elemét írja be, majd hozzáadja mint intervallum. Ezután újra megadja a következő halmazbeli számot a kisebb és nagyobb helyre. Ez azért lehetséges így, mert az intervallum esetén az egyenlőség megengedett.
A kiírás ugyan intervallumként jelenik meg, pl.: [1..1] de abba belegondolhatunk, hogy az 1-től 1-ig tartó intervallum valójában egy egy elemű halmazt jelent.

3.5.2. Segítségkérés

A játékosnak van még lehetősége segítséget kérni, a HINT gomb megnyomásával. A segítségkérés az alábbi esetekben nem vehető igénybe:

- ☐ Ha a hazugságok maximum száma (e_n) 1. Tehát 1 hazugság esetén mindegy mekkora intervallumot ad meg a játékos nem lesz segítsége.

- ☐ Ha az intervallum vége kisebb vagy egyenlő, mint 10. Ebben az esetben mindegy hány hazugsággal játszik a játékos nem lesz segítsége.

[1] A legkisebb prímszám a 2.

□ Ha már elhasználta az összes segítségét.

Ha a fent felsorolt esetek állnak fent, akkor a HINT feliratú gomb nem kattintható.

Legjobban úgy lehet megfogalmazni, hogy a játékosnak akkor van segítsége, ha a hazugságok maximum száma legalább 2 és az intervallum vége legalább 11.

Ha nem az előbb felsorolt esetek valamelyik áll fent, akkor a játékosnak kezdetben $e_n - 1$ db segítsége van.[2] Ha a gomb felé viszi az egeret, akkor el tudja olvasni, hogy hány segítsége van még. Minden segítség eggyel csökkenti az aktuális értéket. A segítségek között nincsen sorrend felállítva, de mindegyik csak egyszer jelenhet meg. A gép a segítségek során csak igazat mond. Az alábbi szövegek jelenhetnek meg a gombra kattintva:

- *I lied for ... times.*
  Azt mondja meg, hogy eddig összesen hányszor hazudott. Ez az érték mindenképpen a $[0, e_n]$ intervallumban van.

- *I was lied first at ... . answer.*
  Ekkor azt lehet megtudni, hogy melyik válasz esetén hazudott először.

- *I was lied last at ... . answer.*
  Az előzőhöz hasonlóan itt azt lehet megtudni, hogy mikor hazudott utoljára.

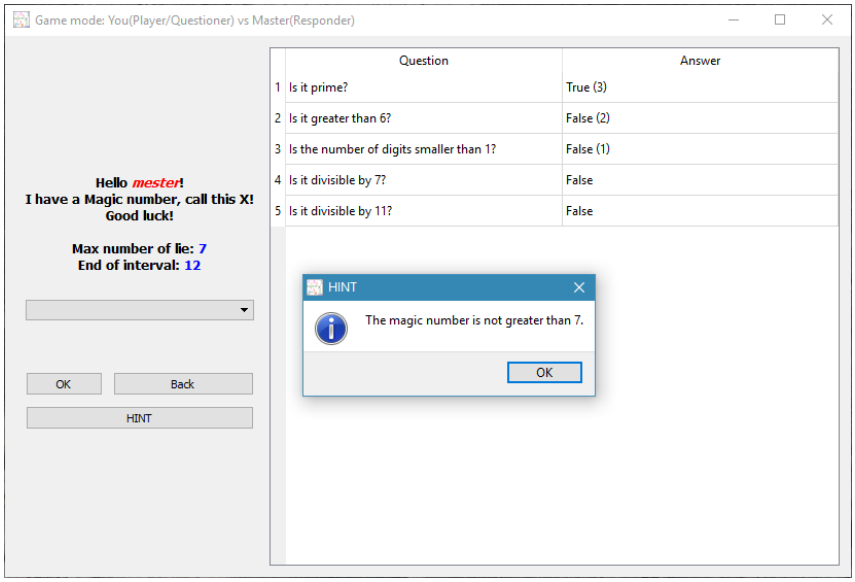
- *The magic number is not greater than ... .*
  A gondolt számnál nem nagyobb számot ad meg. (lásd 3.8 ábra)

- *The magic number is not smaller than ... .*
  Abban segít, hogy a gondolt szám nem kisebb mint az általa most megadott szám.

- *I'm sorry, now I would not like to help.*
  Ez egy segítség, ami valójában nem segít. Ekkor a gép úgy gondolja, hogy nincs kedve segíteni.

Minden esetben az OK gombra kattintva lehet megkérdezni a gépet. Minden ilyen eset egy kérdésnek számít, így adódik össze az a szám, hogy hány kérdésből sikerült kitalálnia. A rákérdezés is kérdésnek számít! Ekkor már nem lehet hazudni.
Amennyiben egy játék befejeződött, akkor a program visszalép az Új játék fülre (3.4 alfejezet). Itt minden mező üresen lesz látható, mint az eredeti állapotban, csak a Player mód lesz bejelölve.
A kérdezés csak a mezők helyes kitöltése esetén lehetséges. Amennyiben a játékos helyes formátumban adja meg az értéket a mező háttérszíne zöld-re vált át, ellenkező esetben fehér színű. A megjelenő hibaüzenetekről a 3.7 alfejezetben lehet olvasni.

[2] e_n-nel jelöltük a lehetséges hazugságok maximumát.



3.8. ábra. Egy segítség

3.5.3. Válaszadó felület

A program jobb oldalán látható táblázat szolgál arra, hogy megmutassa milyen kérdéseket tett fel eddig a játékos és azokra milyen válaszokat adott a gép. A táblázatban a sorok meg vannak számozva, ezzel is mutatva, hogy eddig hány kérdést tett fel a felhasználó.

A válaszok úgy jelennek meg, hogy a játékos által kiválasztott kérdés és az abban szereplő „ ... ” rész helyére kerül a megkérdezett szám. Intervallum esetén az „interval” szót helyettesíti a megadott intervallum, vagy intervallumok uniója.

Amennyiben a felhasználó úgy érzi, hogy egy adott kérdésnél a gép hazudott, akkor módosítani tudja a táblázat Answer oszlopának az adott értékét. Ez esetben a játék végéig az az érték fog ott szerepelni és csak a felhasználó által módosítható vissza a gép eredeti válaszára.

Amennyiben a játékos segítséget vesz igénybe, jó ötlet lehet a segítség szövege alapján módosítani a táblázatban a gép válaszait. Erre látható a két ábrán példa. Az első képen (3.9a ábra) az eredeti válaszok láthatók. Majd a játékos felhasznált egy segítséget, amiben megtudta, hogy a gép első alkalommal a 2. válaszánál hazudott. Így az ottani értéket módosította és feljegyezte, hogy ott biztosan elhasznált egy hazugságot. A módosítás után a válaszadó felület a 3.9b ábrán látható módon néz ki.

	Question	Answer
1	Is it prime?	False
2	Is it smaller than 10?	False
3	Is it greater than 11?	False
4	Is the number of digits smaller than 2?	False
5	Is it divisible by 5?	True
6	Is it divisible by 10?	True
7	Is it in [9..11]?	False

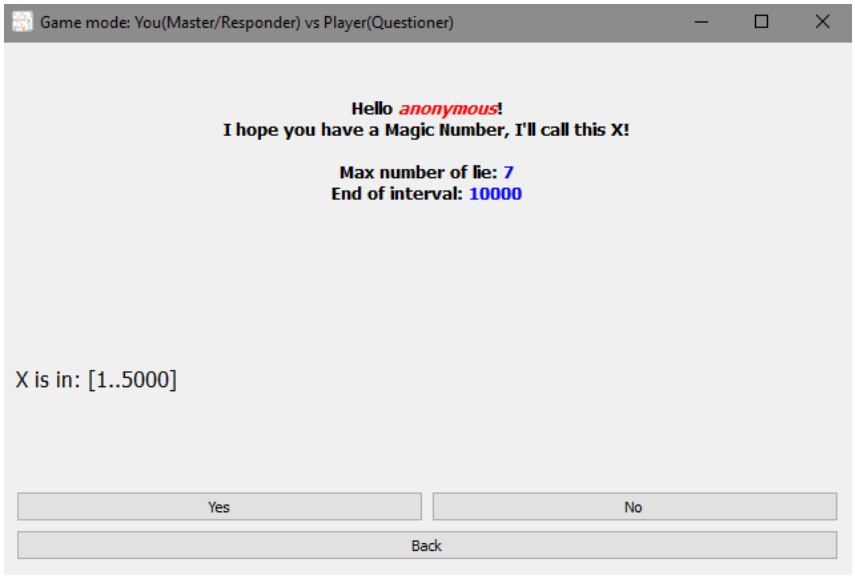
(a) Az eredeti válaszok

	Question	Answer
1	Is it prime?	False
2	Is it smaller than 10?	True (1)
3	Is it greater than 11?	False
4	Is the number of digits smaller than 2?	False
5	Is it divisible by 5?	True
6	Is it divisible by 10?	True
7	Is it in [9..11]?	False

(b) Módosítás után a válaszok

3.6. Master mód

Ekkor a felhasználónak kell gondolnia egy számra, ugyanis a gép a valódi játékos és a felhasználó a játékmester. Ebben a módban a program az alábbiképpen néz ki (lásd 3.10 ábra). Itt is olvasható kiemelten, színesen a köszöntés és a fontosabb információk az előző ablak adatairól. A játékmesternek ilyenkor csak arra kell figyelni, hogy számolja a hazugságainak a számát.

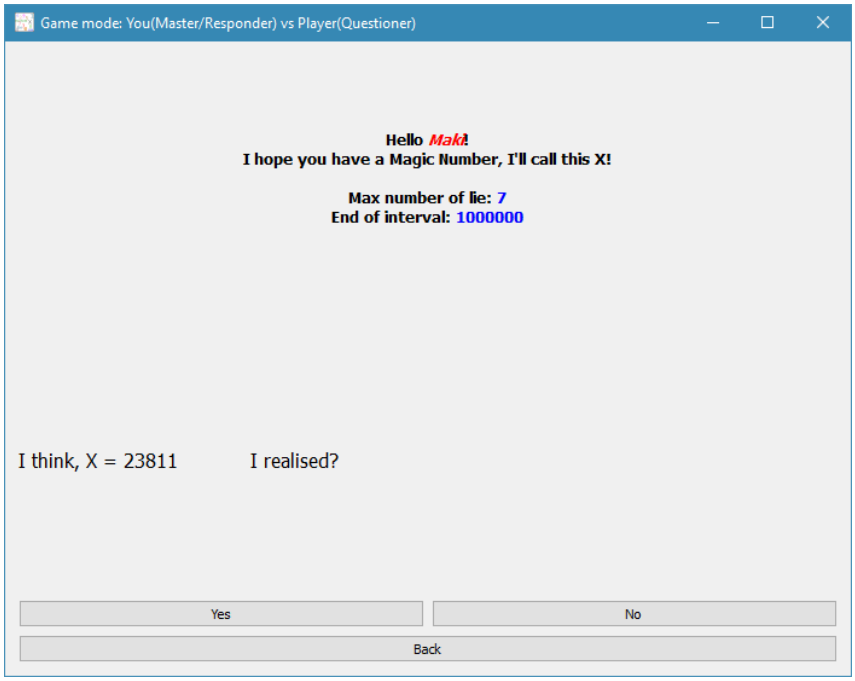


3.10. ábra. „Master” mód

A programban ebben az esetben nem sok funkció érhető el, ugyanis a játékmesternek, azaz most a felhasználónak csak *Igen/Nem* választ kell adni a gép által feltett kérdésre.

Ekkor a játékosnak nincsen szüksége segítségre, így sosem fogja arra kérni a felhasználót, hogy adjon neki támpontot. A gépnek ebben az esetben csak arra van szüksége, hogy a játékmester a maximálisnál többször ne hazudjon és az előre fixált intervallumból gondoljon egy számra. Ekkor valahány kérdés után mindenképpen rá fog tudni kérdezni

egy pontos számra. Ezt a következőképpen fogja megtenni (lásd 3.11 ábra).

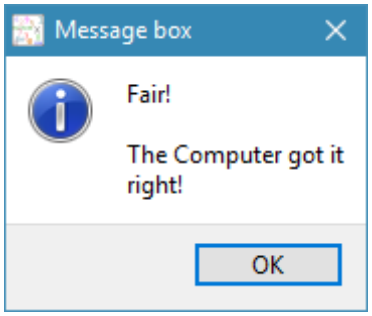


3.11. ábra. A gép rákérdez a gondolt számra

A játékmesternek a rákérdezéskor is *Igen*-t vagy *Nem*-et kell nyomni. Ekkor azonban már **nem hazudhat**.

Amennyiben a gép kitalálta a számot a „Fair!" felirat jelenik meg. Ekkor a játékmester a szabályoknak megfelelően játszott, nem hazudott többször a lehetségesnél. (lásd 3.12 ábra)

A gép abban az esetben nem tudja csak kitalálni a számot, ha a játékmester az intervallumon kívül gondolt egy számra, vagy ha e_n-nél több alkalommal hazudott. Ebben az esetben a „Cheat!" felirat jelenik meg.



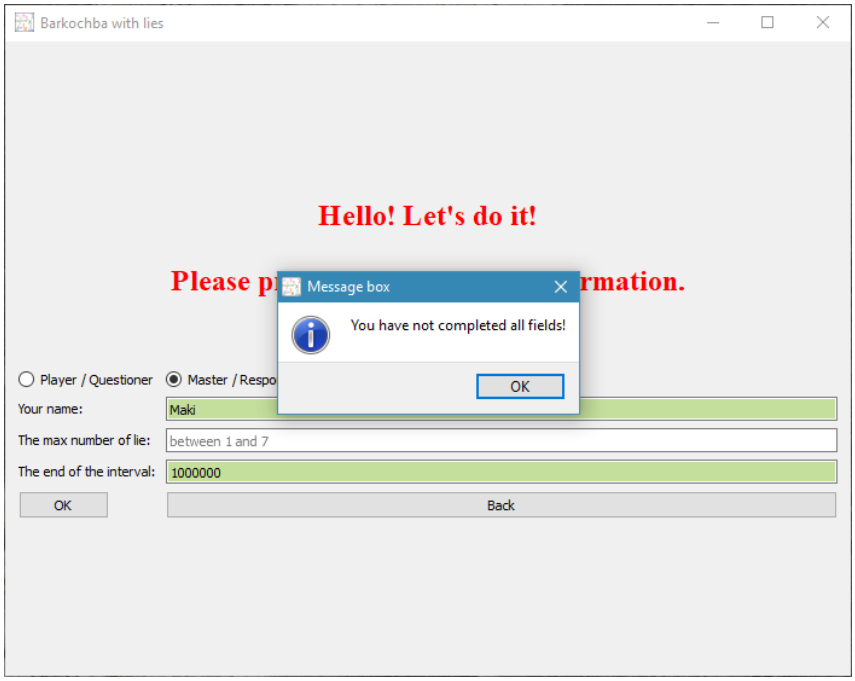
3.12. ábra. Győzelem esetén az üzenet

3.7. Hibaüzenetek

Ebben a részben azok a hibaüzenetek lesznek bemutatva, amiket a felhasználó a program használata során tapasztalhat. Itt csak a játék elindításától -a Főmenü megjelenésétől- számított hibákat mutatjuk be. Az itt felsoroltakon kívül más hibaüzenet nem jelenik meg.

Elsőként Új játék indítása esetén találkozhat a felhasználó hibaüzenettel. Több minden okozhatja ilyenkor a problémát:

- *You have not completed all fields!*
  Amennyiben nem töltötte ki az összes mezőt az alábbi képet fogja látni. Ebben az esetben még az is észrevehető, hogy nem mindegyik beviteli mező háttérszíne zöld. (lásd 3.13 ábra)



3.13. ábra. Hibaüzenet, ha nem töltötte ki az összes mezőt

- *The end of interval must be greater than 1!*
  Ez a szöveg akkor jelenhet meg, ha az intervallum végének 2-nél kisebb számot adott meg.

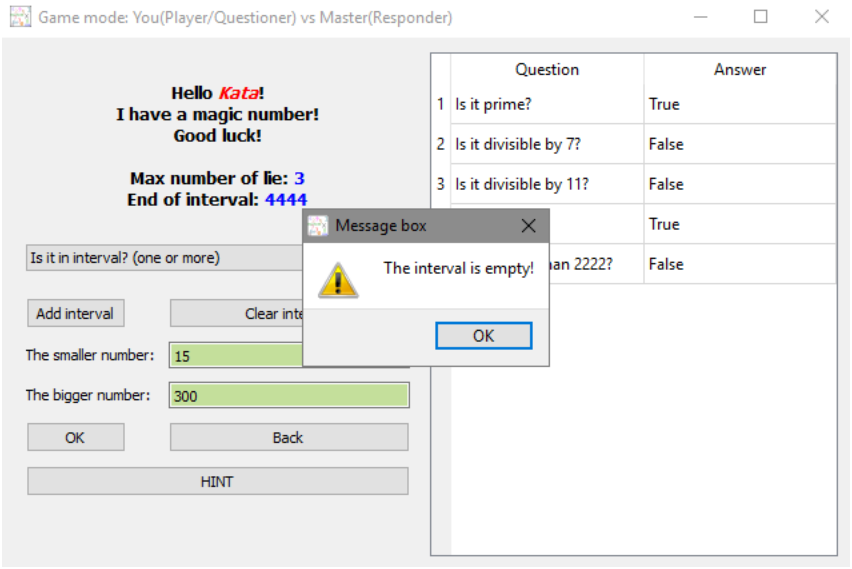
- *You did not choose Game mode!*
  Ha nem választotta ki a játékmódot. Amennyiben úgy látja a felhasználó, hogy ki van az egyik választva, akkor jelölje be egy pillanatra a másikat majd újra a választott játékmódot.

- *The max number of lie must be between 1 and 7!*
  A hazugságok maximum számának, azaz e_n-nek mindenképpen 1 és 7 közötti egész számnak kell lennie.

Player mód esetén az alábbi hibaüzeneteket kaphatja:

- *You have not completed all fields!*
  Ilyen hibaüzenet esetén a játékos nem töltötte ki az összes mezőt, ami a kérdés feltételéhez szükséges.

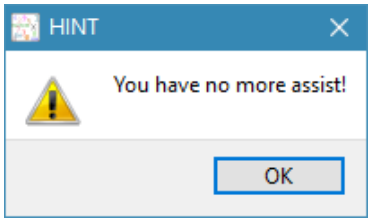
- *The interval is empty!*
  A játékos ebben az esetben még nem adta meg az intervallumot, amit törölni szeretne. Intervallumot úgy tud hozzáadni, hogy az „Add interval” gombra kattint. Ekkor lentebb megjelenik szövegszerűen az előbb hozzáadott intervallum. Amennyiben ilyen szöveget nem lát, akkor még nem adott hozzá intervallumot. Csak a kisebb, nagyobb mezők kitöltése még nem jelenti automatikusan az intervallum hozzáadását.



3.14. ábra. A program és a hibaüzenet,
ha üres listát szeretne törölni

A 3.14 ábrán az az eset látható, amikor üres még a lista, de a játékos a „Clear interval” gombra kattintott. Látszódik, hogy a HINT gomb alatt nincsenek intervallumok kiíratva nem úgy, mint a 3.7 ábra esetében.

- *You have no more assist!*
  Ha a kérdező már elhasználta az összes segítségkérését akkor jelenik meg ez a hibaüzenet. A segítségkérésről bővebben a 3.5.2 alfejezetben lehet olvasni.



3.15. ábra. Hibaüzenet, ha már nincs több segítsége

4. fejezet

Fejlesztői dokumentáció

4.1. A feladat specifikációja

A program az 1.1 alfejezetben tárgyalt Rényi-Ulam-probléma játékban való bemutatása. A játékban két fajta mód érhető el, a Player azaz játékos mód és a Master vagyis játékmester mód.

Játékos mód esetén a gép gondol egy számra. Ekkor a gép oldaláról csak az a kérdéses, hogy mikor hazudjon. Az adott kérdéskor dönti el, hogy most akar-e élni a hazugság lehetőségével vagy nem. Nagy intervallum és sok hazugság esetén a felhasználónak nincs olyan könnyű dolga, hogy kitalálja a gép által gondolt számot. Emiatt használhat segítségeket is a játék során és többfajta formában is felteheti a kérdéseit.

Játékmester mód esetében a hazugságvektor alapján tudja a gép kitalálni a felhasználó által gondolt számot. A vektor függ az intervallum nagyságától és a hazugságok maximum számától is. A legnagyobb lehetséges intervallum 1.000.000 számot tartalmaz. Nagyobb intervallum esetén is működne a játék azonban a mostani megjelenítés szempontjából nem optimális. A hazugságok száma az $[1, 7]$ intervallumban mozoghat. Itt is lehet bővíteni az intervallumot. Akár a 0 hazugság is megengedhető lenne, azonban a játék most a hazugságok kiszűrésére fókuszál.
Ekkor jön elő az 1.1 alfejezetben tárgyalt probléma gépi megvalósítása. A megoldáshoz egy fa szerkezet lett használva. Mely minden lépésben összesen 3 csúcsot tart nyilván: a szülőt és annak a bal, illetve a jobb oldali gyerekét. A csúcsok maguk a hazugságvektorok. Mivel csak 3 csúcs van nyilvántartva így a program számolása, a csúcsok előállítása még 1.000.000 szám esetén is észrevehetetlen.

A program Python 2 nyelvben lett írva, a grafikus felület PyQt4 keretrendszer segítségével lett előállítva, így a fejlesztéshez több *library* is szükséges, melyeket a 4.2 alfejezetben tárgyalunk. A 4.4 alfejezetben mutatjuk be hogyan lehet a felhasználónak előállítani a futtatható *Barkochba.out* és *Barkochba.exe* fájlokat.

4.2. Futtatási környezet

A program Linux és Windows operációs rendszerű gépeken lett megírva. Ezeken mindenképpen lehet fejleszteni a szükséges könyvtárak mellett. A szoftver fejlesztéséhez nem használtam IDE-t. A futtatása parancssoron keresztül történik.

4.2.1. Linux

Ubuntu 16.04-es verzió alatt működő megoldás következik.

Első sorban nyitni kell egy *Terminál*-t. Ott ki kell adni a *sudo apt-get install python2.7* parancsot. A telepítés csak rendszergazdai jogok mellett engedélyezett.
Ezek után a *sudo apt-get install python-qt4* parancsot kell kiadni, ami fel fogja telepíteni a PyQt4 keretrendszert.

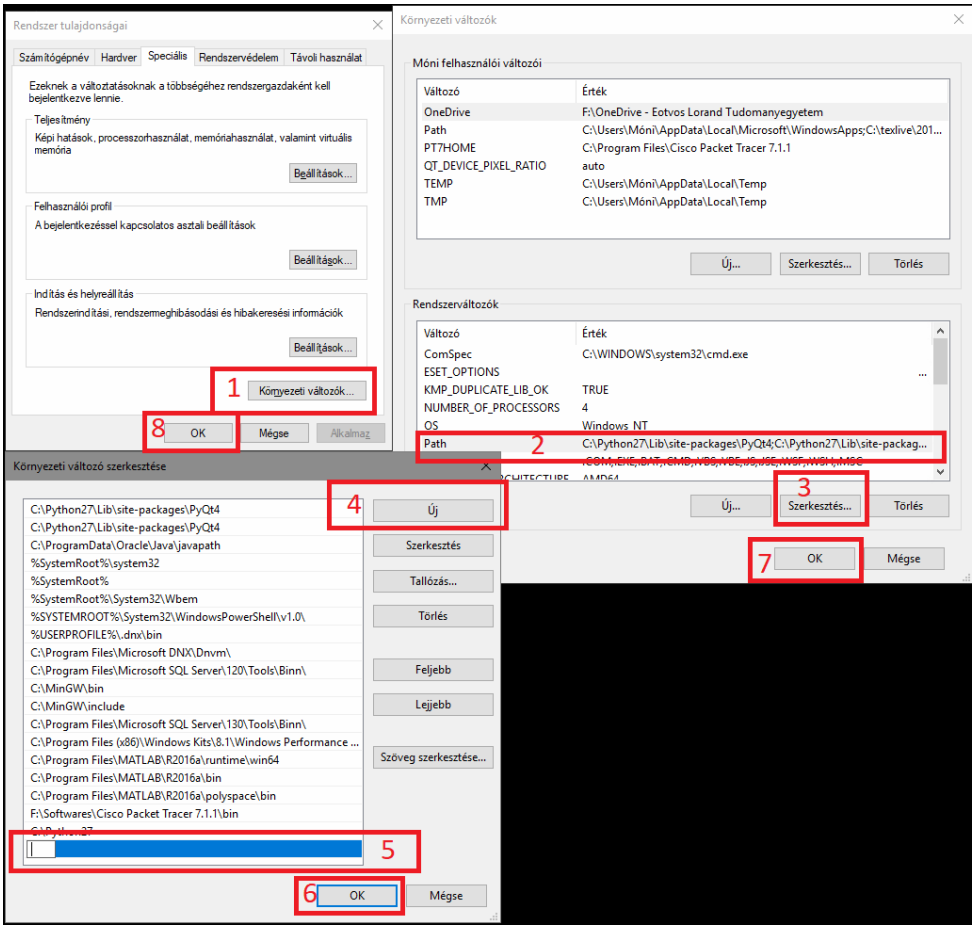
A sikeres telepítést az alábbi módon lehet kipróbálni. A fejlesztő nyit egy *Terminált*. Ott kiadja elsőként a *python* parancsot majd az új felületen az *import PyQt4* parancsot. Amennyiben ez nem száll el hibával a telepítés sikeres volt.

4.2.2. Windows

Windows 10 64-bites verzió alatt működő megoldás lesz bemutatva.

Elsőként a Python 2 programot kell feltelepíteni. Ehhez szükséges letölteni a Python 2.7.14 64 bit-es csomagot a https://www.python.org/downloads oldalról. Az oldalról egy futtatható *.exe* fájl került letöltésre. Fontos, hogy semmiképpen se Python 3-as csomag legyen letöltve!
Letöltés után a fájlra kattintva elindul a telepítés. A program telepítésének a helye az alapértelmezett meghajtó gyökér könyvtára legyen. Például: `C:\`

A telepítés végeztével ekkor a meghajtón lesz egy `Python27` nevezetű mappa. Következő lépésként ezt a mappát hozzá kell adni a környezeti változókhoz. Ezt úgy lehet megtenni, hogy **Keresés a Windowsban -> A rendszer környezeti változóinak módosítása -> Környezeti változók... -> Rendszerváltozók -> Path -> Szerkesztés -> Új -> megadni a Python27 mappa elérési útvonalát -> OK -> OK -> OK**. A helyes elérési út megadása, amennyiben a `C` meghajtó gyökérkönyvtárába lett mentve: `C:\Python27` . A 4.1 ábrán láthatók az előbb leírt lépések.



4.1. ábra. Környezeti változók beállítása

A https://sourceforge.net/projects/pyqt/files/PyQt4/PyQt-4.11.4/ oldalról a `PyQt4-4.11.4-gpl-Py2.7-Qt4.8.7-x64.exe` nevű fájlt kell letölteni majd futtatni. A telepítés helyének az előzőekben létrehozott `Python27` mappát kell megadni. Sikeres település után használható lesz a PyQt4 keresztrendszer.

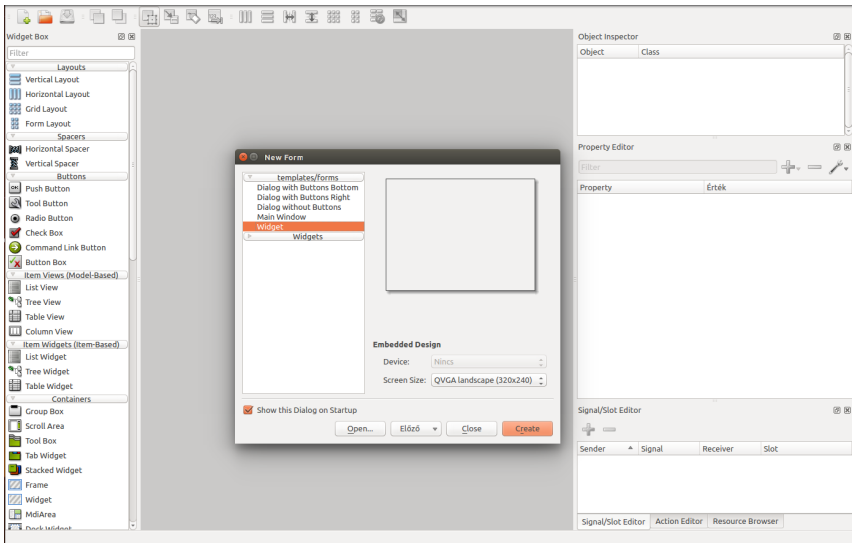
A sikeres telepítést az alábbi módon lehet kipróbáni. A fejlesztő nyit egy *Parancssor*-t vagy egy *Windows PowerShell* ablakot. Ott kiadja elsőként a *python* parancsot majd az új felületen az *import PyQt4* parancsot. Amennyiben ez nem száll el hibával a telepítés sikeres volt.

4.3. PyQt4

Maga a Qt egy cross platform c++ framework (qt-project.org). [6] Magyar nyelven az alábbi anyagot találtam, ami nagy vonalakban mutatja be néhány dián keresztül a *PyQt*-t. [7]
A keretrendszer többek között Linux, Windows, iOS rendszereken is használható.

A PyQt két fő modulja van: *QtCore* és *QtGui*. Ezek segítségével minden megvalósítható, ami egy GUI-s alkalmazáshoz elengedhetetlen.

Aki nem szeretné a kinézetet teljes egészében magától megírni, de könnyen össze tudná „kattintgatni", annak sem kell csüggednie. A PyQt Designer segítségével ez megtehető. Ez a program mind Linux, mind Windows operációs rendszerű gépre feltelepíthető. A szoftver által parancssoron keresztül könnyedén ki lehet generáltatni az ablak kinézetét tartalmazó *.py* fájlt.
Az alábbi kép fogja fogadni a felhasználót az indításkor.

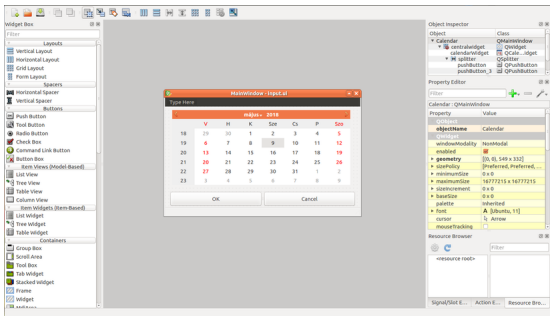


4.2. ábra. PyQt Designer

A program közepén látható a *New Form* felület, ahol kezdetben az adható meg, hogy mekkora méretű és milyen típusú ablakot szeretne a felhasználó.
A program bal oldalán láthatóak többek között az elérhető *Layout*-ok, *Spacer*-ek, *Button*-ok. Ezek csak az ablak kiválasztása után lesznek elérhetőek. Így lehetőség nyílik egy fejben már kigondolt kinézet gyors előállítására.
Jobb oldalt látható az *Object inspector* és a *Properity Editor* felület. Itt lehetőség van beállítani az osztály-, a változók nevét. Akár kezdőértéket is meg lehet adni egy beviteli mező esetén. Lehetőség van arra is, hogy megmondjuk egy adott gomb megnyomásakor mi történjen.

Miután minden be lett állítva a fájlt el kell menteni. Majd egy *Terminál*-t kell nyitni az előbbi mappába és kiadni a *pyuic4 input.ui -o output.py* parancsot. Itt most az *input.ui* a bemeneti PyQt Designer által lementett fájl, az *output.py* pedig annak a fájlnak a neve, amit kimenetei fájlként szeretne a felhasználó.

Az alábbi képeken ez a 3 állapot látszik.
A 4.3a képen a PyQt Designer-en belüli kinézet látszódik. Egy egyszerű naptár van megjelenítve az ablakban. A 4.4 képen a kinézetből előállított forráskód látható. Míg a 4.3b képen már a futtatott Python fájl általi kinézet. Észrevehető, hogy szinte semmilyen különbség sem fedezhető fel a két kinézet között.



(a) PyQt Designer Naptár kinézet tervezése

(b) Felhasználó által látott Naptár program

```
from PyQt4 import QtCore, QtGui
try:
    _fromUtf8 = QtCore.QString.fromUtf8
except AttributeError:
    def _fromUtf8(s):
        return s
try:
    _encoding = QtGui.QApplication.UnicodeUTF8
    def _translate(context, text, disambig):
        return QtGui.QApplication.translate(context, text, disambig, _encoding)
except AttributeError:
    def _translate(context, text, disambig):
        return QtGui.QApplication.translate(context, text, disambig)
class Ui_Calendar(object):
    def setupUi(self, Calendar):
        Calendar.setObjectName(_fromUtf8("Calendar"))
        Calendar.resize(535, 269)
        self.centralwidget = QtGui.QWidget(Calendar)
        self.centralwidget.setObjectName(_fromUtf8("centralwidget"))
        self.calendarWidget = QtGui.QCalendarWidget(self.centralwidget)
        self.calendarWidget.setGeometry(QtCore.QRect(20, 10, 501, 211))
        self.calendarWidget.setObjectName(_fromUtf8("calendarWidget"))
        self.splitter = QtGui.QSplitter(self.centralwidget)
        self.splitter.setGeometry(QtCore.QRect(14, 240, 521, 41))
        self.splitter.setOrientation(QtCore.Qt.Horizontal)
        self.splitter.setObjectName(_fromUtf8("splitter"))
        self.splitter.raise_()
        self.calendarWidget.raise_()
        Calendar.setCentralWidget(self.centralwidget)
        self.menubar = QtGui.QMenuBar(Calendar)
        self.menubar.setGeometry(QtCore.QRect(0, 0, 535, 25))
        self.menubar.setObjectName(_fromUtf8("menubar"))
        Calendar.setMenuBar(self.menubar)
        self.statusbar = QtGui.QStatusBar(Calendar)
        self.statusbar.setObjectName(_fromUtf8("statusbar"))
        Calendar.setStatusBar(self.statusbar)
        self.retranslateUi(Calendar)
        QtCore.QMetaObject.connectSlotsByName(Calendar)
    def retranslateUi(self, Calendar):
        Calendar.setWindowTitle( translate("Calendar", "MainWindow", None))
```

4.4. ábra. Naptár kinézet előállított forráskódja

4.4. Hogyan jön létre a futtatható állomány?

A *Python* nyelvű programoknál is előjön az a probléma, hogy hogyan adjuk oda a szoftvert egy felhasználónak? Egy átlagos, hétköznapi ember nem fog feltelepíteni magának egy fordító programot csak azért, hogy játszhasson és a környezeti változókat sem fogja beállítani, nagy valószínűséggel azt sem tudja, hogy létezik ilyen fogalom.

Erre a problémára a *PyInstaller* nevezetű program lehet segítség. [8]
Mind Windows, Linux és Mac OS X [9] rendszerekre is elérhető. A telepítést nem részletezzük, erről bővebben a https://pythonhosted.org/PyInstaller/installation.html oldalon lehet olvasni.
A szoftver nem támogat minden könyvtárat, a támogatott könyvtárak listája a https://github.com/pyinstaller/pyinstaller/wiki/Supported-Packages oldalon érhető el.

Használni a következőképpen lehet:
A telepítést követően győződjünk meg arról, hogy a *Python* programunk működik. Nyissunk egy parancssort az adott fájlok mappájában.
Itt adjuk ki a következő parancsot: *pyinstaller –onefile –windowed main.py*
A *–onefile* paraméterrel van lehetőség egy futtatható fájlt létrehozni. Míg a *–windowed* paraméterrel azt lehet megadni, hogy az alkalmazás ablakos, azaz nem szükséges hozzá konzolos felület. A *main.py* pedig az a fő fájl, amiből a kívánt program indul.
A *PyInstaller* a fenti parancs hatására 2 mappát hoz létre. Az egyik a *build* nevű mappa, amibe a log fájlok kerülnek, ez a mappa a parancs lefutása után kitörölhető. A másik mappa a *dist*, ebbe kerül a futtatható fájl. Amennyiben nem használtuk a *–onefile* opciót, akkor itt több fájl lesz és csak ezek együttes meglétével lesz futtatható az ottani *.out* vagy *.exe* fájl. Ellenkező esetben az itt található *.out* vagy *.exe* fájl már önmagában fut. Egyik esetben sem kell ezek után hozzá semmilyen *Python* fordító, bátran továbbítható a felhasználónak.

Az eddig leírtak csak a grafikus kinézetet biztosították. A szoftverben megjelenő képet, és a program logóját nem.
A *dist* és a *build* mappával egy időben létrejött egy *.spec* fájl is.[10]
Ez nem került bele egyik létrejött mappába sem, hanem a példaként felhozott *main.py* fájllal megegyező mappában van. Ezt a *.spec* fájlt egy szövegszerkesztő programban nyissuk meg.

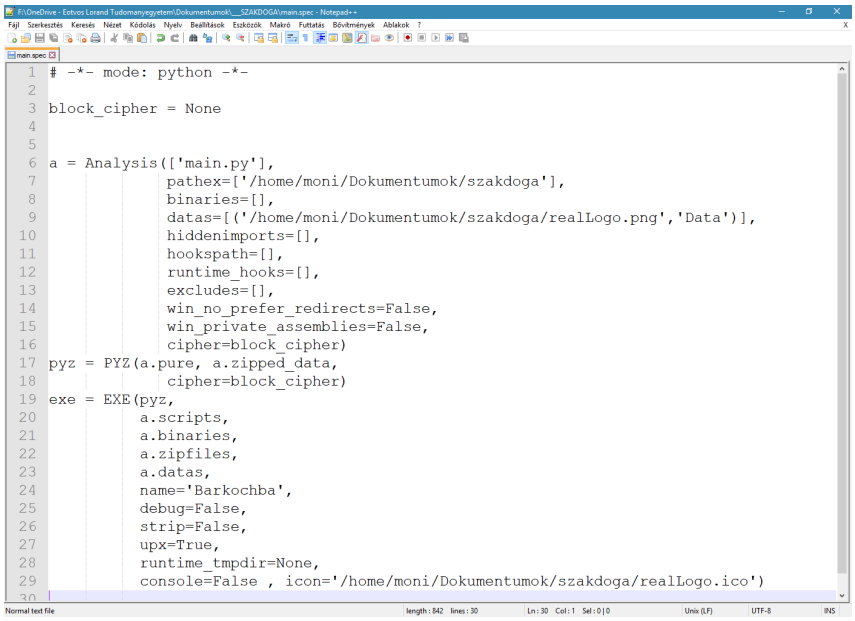
A kép megjelenítéséhez:
Az *a = Analysis* kezdetű parancsot keressük meg benne és ennek a *datas =* részét hozzuk az alábbi formára:

datas = [('abszolút elérési útvonal a kép nevével együtt', 'Data')],

Azaz az első paraméter legyen a kép abszolút elérési útvonala. Míg a második paraméter minden esetben a *'Data'* legyen.

Az ikon megjelenítéséhez:
Az *exe = EXE* kezdetű parancsot keressük meg benne és az *icon =* sort egészítsük ki az ikon fájl abszolút elérési útvonalával. A kép kiterjesztése itt legyen *.ico*. [15]

Példa egy helyes *.spec* fájlra:



4.5. ábra. *.spec* fájl

Miután mindkét sort módosítottuk 3 dolgunk maradt már csak a következő sorrendben:

1. Töröljük le a *build* és a *dist*.
2. Adjuk ki a *pyinstaller main.spec* parancsot.
3. A *dist* mappába másoljuk át az összes képet, amit a szoftverben szeretnénk látni (az *.ico* fájlt is).

Ezek után, ha elindítjuk a *dist* mappában lévő futtatható állományt az ikon is látszódni fog és a benne lévő képek is. Amennyiben a szoftver tartalmaz képeket, akkor a képeket és ikon fájlokat is továbbítani kell a felhasználó számára!

A 4.6 ábrán látható a PyInstaller grafikus változata. Azok számára nagy segítség lehet, akik nem szeretnének parancsokat gépelni és könnyebben eligazodnak egy grafikus felületen. A képre kattintva megnézhetik ezt hogyan tudják megtenni: [11]

4.6. ábra. PyInstaller grafikus felülete

4.5. Adatbázis

A szoftver a *SQLite 3*-féle adatbázist használ.[12]
A *SQLite 3* adatbázist a *Python 2* nyelv támogatja. A nevében lévő *Lite* szó utal arra, hogy kevés helyet foglal, akár beágyazható adatbázis-kezelő. Többnyire egy fájlban tárolja az adatokat. A programnak egyetlen egy táblára volt szüksége, ezért esett erre a választás.

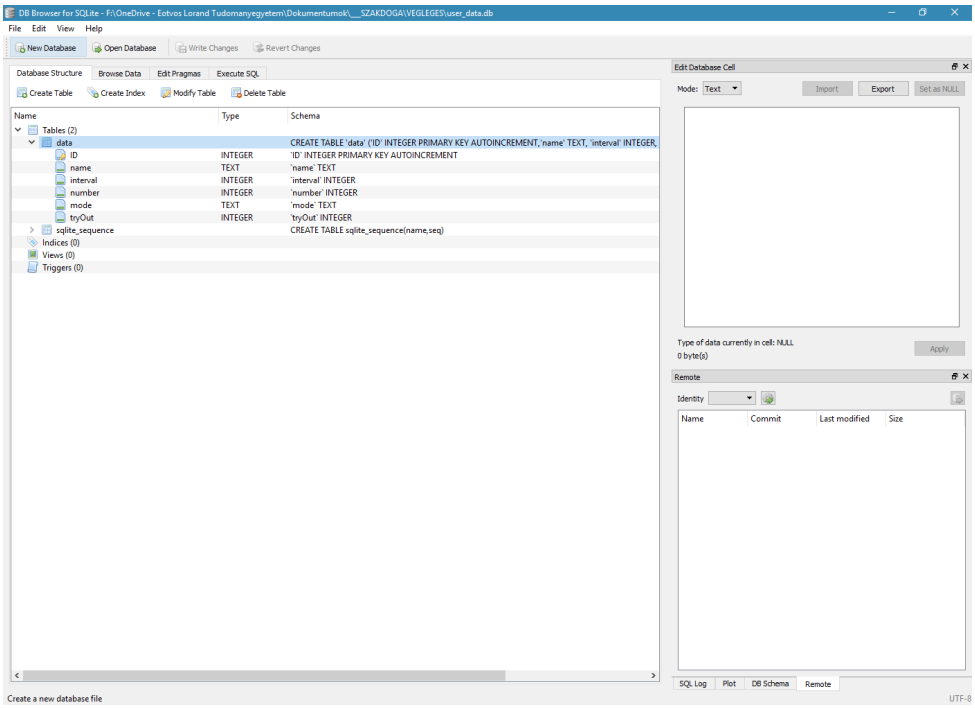
Az adatbázis fölé egy megjelenítő réteg lett készítve az előző fejezetben bemutatott PyQt4 keretrendszer segítségével. Így az adatbázissal két fájl (*score.py* és *inputDialog.py*) képes kommunikálni. Az előbbi fájl felel a megjelenítésért míg az utóbbi által kerülnek az adatbázisba az adatok.

Arra törekedtünk, hogy a játék bárki számára elérhető legyen, ezért az adatbázis nem szerveren fut. Így nincs arra esély, hogy kapcsolódási problémák miatt megszakadjon a játék. Az adatbázis az első adat beírásakor jön létre lokálisan a *main.py* fájllal megegyező mappában. Amennyiben már létezik, akkor minden új játék adata abba íródik bele.
Mivel nem szerveren fut, így nem láttuk annak értelmét, hogy külön belépő (login) felületen keresztül kelljen belépni a játékba. Így több felhasználó is tud ugyanazzal a névvel játszani. Az adatbázis egy *ID* mező alapján különbözteti meg a játékosokat. [13]

Az adatbázisnak csak az alapfunkciói kerültek leimplementálásra. Így többek között van olyan függvénye, amivel a kapcsolat jön létre közte és a játék között, van beszúró

függvénye, amit meghívva és jól felparaméterezve kerülnek tárolásra az adatok.
A megszokott függvényeken túl van egy magyarul ún. *kiíratő* függvénye, ami segítségével jeleníti meg az adatokat az Eredménytáblán.

Az alábbi ábrán látható az adatbázis felépítése.

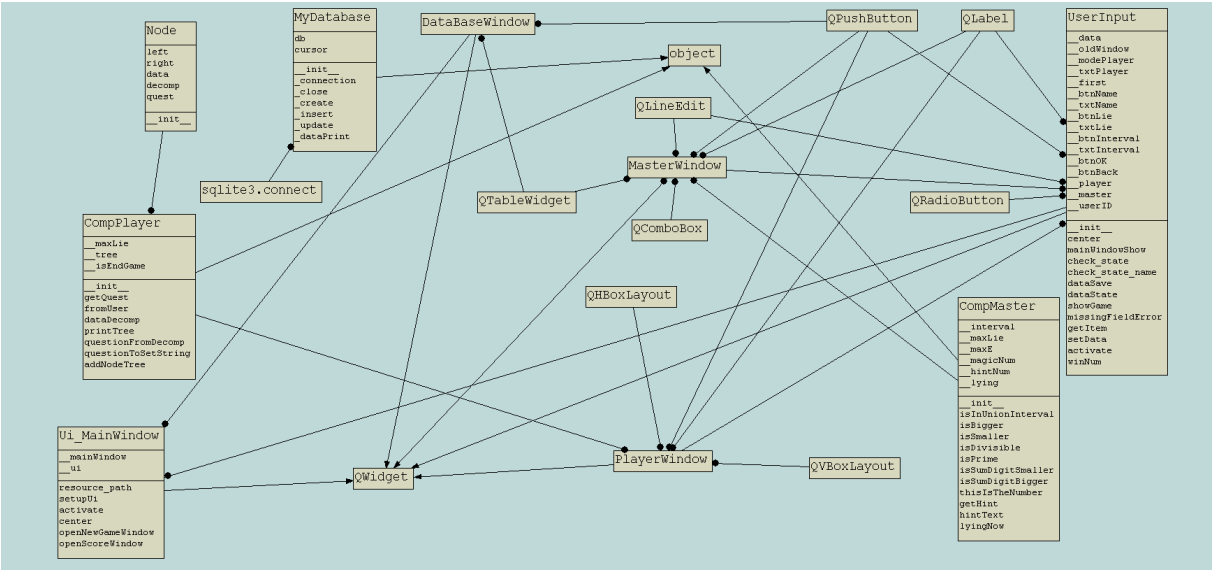


4.7. ábra. Adatbázis

4.6. Osztályok

A 4.8 ábrán látható a szoftver osztályainak UML diagramja.[14]

A program 9 db *Python* fájlból áll. A fő *main.py* fájlt leszámítva minden fájlban egy külön osztály van definiálva. Ez alól kivétel az a fájl, amelyben a fa szerkezet került implementálásra. Ekkor két osztály van a fájlon belül, ez a fájl a *computerIsPlayer.py*. Az adatbázisról már egy korábbi fejezetben beszéltünk, így azt nem tárgyaljuk.



4.8. ábra. Osztályok UML diagramja

Az osztályok és rövidebb leírásuk:

- A megjelenítésért felelős osztályok:

Osztály név	**Ősosztály**	**Paraméterek**	**Leírás**
Ui_MainWindow	object	-	A Főmenü kinézetéért felel. A felhasználó döntése alapján egy új ablakot fog előtérbe helyezni.
UserInput	QWidget	*oldWindow*: Az előző ablakot jelenteni, hogy Back gombnyomás esetén az előző ablakot helyezze előtérbe.	A felhasználónak itt kell megadnia a játék paramétereit, adatait. Az osztály fő alapja egy űrlap kinézet. Az űrlapból kiindulva menti el az adatokat és továbbítja azt az Adatbázisnak. A szükséges adatokat pedig továbbadja a felhasználó által kiválasztott játékmód kinézetének.

MasterWindow	QWidget	*oldWindow*: Ugyan amiatt, mint az előbb. *inputUserData*: A felhasználó által megadott, számára szükséges adatokat kapja meg lista formájában.	Ekkor a gép a játékmester. Az osztálynak két fő része van. Az kérdező felületet megvalósító űrlap kinézet és a válaszokat mutató táblázat. Ez az osztály a kinézetért felel, a felhasználó által megadott kérdés, érték párokat továbbítja egy másik osztálynak (CompMaster).
PlayerWindow	QWidget	*oldWindow*: Amiatt, mint az eddigi esetekben. *inputUserData*: Mint a korábbi esetben.	A gép a játékos. Az ablakon csak 3 gomb látható. Kinézetben nem annyira összetett, mint a korábbi osztályok. A játékmester válaszait továbbítja.
DataBaseWindow	QWidget	*oldWindow*: Mint a korábbi esetekben, itt is az előző ablakot jelenti.	Az adatbázissal kommunikál, de csak lekérdezi az adatokat, módosítást nem végez.

- A játék irányításáért felelős osztályok:

Osztály név	**Ősosztály**	**Paraméterek**	**Leírás**
CompMaster	object	*inter*: Az intervallum végének az értéke. *lieNum*: Az e_n érték.	A felhasználó kérdésére ad *Igen* / *Nem* válaszokat. Az adott kérdésnél dönti el, hogy él-e a hazugság lehetőségével vagy sem.
Node	-	*value*: A csúcs adata. *dec=[]*: A csúcs felbontása. *que=[]*: A csúcshoz tartozó kérdés.	A kezdeti paramétereket leszámítva van még *left* és *right* adattagja. Ez az 5 érték így együtt tesz ki egy csúcsot.

CompPlayer	object	*inter*: Mint a fentebbi esetben. *lieNum*: Szintén, mint a korábbi esetben.	A játékmester *Igen / Nem* válaszai alapján építi ki a fát. Mindig csak 3 csúccsal dolgozik: gyökér, jobb gyerek, bal gyerek. A csúcshoz tartozó adat minden esetben a hazugságvektor. A kérdést is ebből állítja elő.

4.6.1. Hazugságvektor használata program szinten

Nagyon sokat beszéltünk már a hazugságvektor használatáról, hogy miért jó. A 2. fejezetben már leírtuk általánosan a működését. Most pedig az fog következni, hogy hogyan is lehet ezt leprogramozni.

Az előző alfejezetben bemutatott *Node* és *CompPlayer* osztályok együtt valósítják meg ennek a működését.
A *Node* osztály csak a fa struktúráját, szerkezetei felépítését adja meg. Mindehhez 5 adattagot használ:

- left
  Kezdetben ennek a csúcsnak nincsen értéke. Ő a bal gyerek. A szülőcsúcsból fog előállni.

- right
  Kezdetben ennek sincs értéke. A jobb gyerek szerepét tölti be és szintén a szülőcsúcs kerül előállításra.

- data
  A szülő csúcs, ami egyben a gyökér is. Az értéke egy listákból álló lista, ami pontosan a hazugságvektor.

- decomp
  A gyökér felbontása. Felbontás alatt a *2. fejezet 2. lépésében* bemutatott módszert értjük.

- quest
  A csúcshoz tartozó kérdés. Ez egy string, azt az intervallumot takarja, amire a játékmesternél fog a játékos rákérdezni.

Kezdetben a *CompPlayer* osztályban létrejön egy *self.__tree* adattag, ami egy *Node* és a *data* adattagja egy lista, ami 1-től az intervallum végéig tartalmazza a számokat. Például:

$$\text{self.\_\_tree.data} = [[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15]]$$

Miután ez megvan a *dataDecomp* függvény segítségével előáll a csúcs felbontása. A felbontás úgy zajlik, hogy a csúcs minden részlistáján végigmegyünk és ezeket elfelezzük. Ha az adott lista páratlan számú elemet tartalmaz, akkor a felső egész részét vesszük. Létrehozunk két üres listát, az egyikbe a listák első felét fűzzük hozzá a másikba a listák második felét. Ezeket pedig egy nagy listába tesszük. Így a végeredmény egy olyan lista, aminek két nagy részlistája van, ez adja a *self.__tree.decomp* tag értékét. Az előző példa esetében:

$$\text{self.\_\_tree.decomp} = [[1, 2, 3, 4, 5, 6, 7, 8], [9, 10, 11, 12, 13, 14, 15]]$$

Jól látható, hogy 15 hosszú listánál az első 8 elem kerül egy listába, mivel a felső egész részt kellett venni. A maradék elem pedig a második részlistába kerül.

A gyökér felbontásából a kérdés nagyon könnyen előállítható, ugyanis az első részlista minden páros számú részlistájának az elemeit teszi fel.
Ez könnyebben megérthető egy példán keresztül.
Ha

$$\text{self.\_\_tree.data} = [[1, 2, 3], [4, 5, 6, 7, 8], [9, 10]]$$

és így

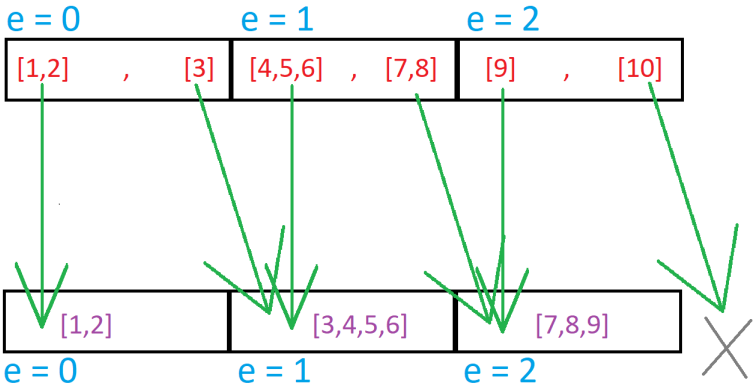
$$\text{self.\_\_tree.decomp} = [[[1, 2], [3]], [[4, 5, 6], [7, 8]], [[9], [10]]]$$

akkor

$$\text{self.\_\_tree.ques} = [1..2] \cup [4..6] \cup [9]$$

Mivel a gyökér felbontását már előállítottuk, így ebből könnyen előállítható a két gyerek is. Erre az *addNodeTree* függvény szolgál. Mint ahogy a neve is mutatja a két gyereket adja hozzá a már meglévő gyökérhez és így alkot ez a 3 csúcs egy fát.

A két gyerek előállítására inkább mutatunk egy-egy ábrát, mivel leírva sokkal bonyolultabbnak hangzik, mint amilyen gyakorlatban.
Ha az előző példát vesszük, akkor a bal gyerek így néznek ki. A bal gyerek az **igaz** ág, tehát amikor a felhasználó a kérdésre a Yes gombot nyomja meg.



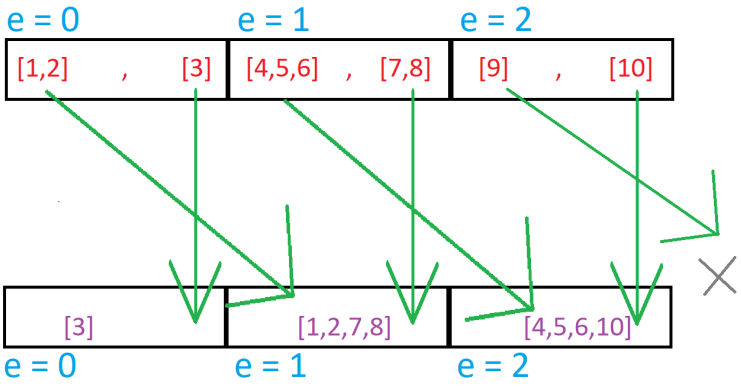
4.9. ábra. Bal gyerek előállítása

A kérdés ugye az volt, hogy a gondolt szám benne van-e a $[1..2] \cup [4..6] \cup [9]$ intervallumok valamelyikében. Ha erre a Yes-t nyomja a játékmester, akkor két lehetőség van:

- Olyan számra gondolt, ami benne van ebben a halmazban. Ekkor nem hazudott, és így a kérdésben szereplő számokra nem érkezett hazugság. Így maradnak ez eddigi helyükön.
- Olyan számra gondolt, ami **nincs benne a halmazban**, de mégis azt mondta hogy benne van, mivel a Yes-t nyomta meg, tehát hazudott. Így a kérdésben nem szereplő számok közül gondolhatott valamelyikre. Az ilyen számoknak a hazugság értéke megnő 1-gyel. Azaz az eggyel nagyobb hazugságértékű részlistába kerülnek át.

Így alakult ki a gyökér left adattagja.

A jobb gyerek előállítása pedig így néz ki az előző példára:



4.10. ábra. Jobb gyerek előállítása

A jobb gyerek a **hamis** ág, tehát amikor a játékmester a No gombot nyomja meg. Ekkor ugye szintén két lehetőség áll fent, mint az előző esetben.

- Olyan számra gondolt, ami tényleg nincs benne a halmazban. Ekkor igazat mondott. Emiatt kérdésben nem szereplő számok maradnak az eddigi helyükön.
- Olyan számra gondolt, ami **benne van a halmazban**. Ekkor hazudott. Ezért a kérdésben szereplő számoknak a hazugság értékük eggyel megnő, és ezáltal a következő részlistába kerülnek át.

A gyökér right adattagja pedig így alakult ki.

A fentebb lerajzolt képek után a 4.11 ábrán látható az ezt megvalósító függvény programkód szinten.

```
def addNodeTree(self, no):
    if sum([len(t) for t in self.__tree.data]) <= 2:
        self.__isEndGame = True
        no.decomp = self.dataDecomp(no)

    leftData = []
    for i in range(1,self.__maxLie+1):
        if i == 1:
            start = 1
        else:
            start = len(no.decomp[0]) if len(no.decomp[0]) < (i-1)*2+1 else (i-1)*2+1
        size = len(no.decomp[0]) if len(no.decomp[0]) < i*2+1 else i*2+1
        temp = []
        for j in range(start,size):
            temp.extend(no.decomp[0][j])
        leftData.append(temp)
    no.left = Node([])
    no.left.data.append(no.decomp[0][0])
    no.left.data.extend(leftData)
    no.left.decomp = self.dataDecomp(no.left)

    rightData = []
    for i in range(1,self.__maxLie+1):
        if i == 1:
            start = 1
        else:
           start = len(no.decomp[1]) if len(no.decomp[1]) < (i-1)*2+1 else (i-1)*2+1
        size = len(no.decomp[1]) if len(no.decomp[1]) < i*2+1 else i*2+1
        temp = []
        for j in range(start,size):
            temp.extend(no.decomp[1][j])
        rightData.append(temp)

    no.right = Node([])
    no.right.data.append(no.decomp[1][0])
    no.right.data.extend(rightData)
    no.right.decomp = self.dataDecomp(no.right)
```

4.11. ábra. A két gyereket a fába tevő függvény

Most hogy mindhárom csúcs értékei megvannak fel is lehet tenni a kérdést a játékmesternek.
A válasza alapján pedig két helyzet állhat fent:

- Yes-t nyom.
  Ekkor azt mondjuk, hogy az eddigi gyökért írjuk felül a gyökér bal oldali gyerekével.
- No-t nyom.
  Ekkor az eddigi gyökért írjuk felül a gyökér jobb oldali gyerekével.

Programkód szinten ezt a 4.12 ábrán látható függvény végzi.

Mindezek után hívjuk meg az *addNodeTree* függvényt, hogy az új gyökérnek is elő legyenek állítva a gyerekei. A kódban jól látható, hogy a válasz alapján mondjuk meg melyik gyerek legyen a gyökér csúcs.

```
def fromUser(self,ans):
    if ans == 'y':
        self.__tree = self.__tree.left
    elif ans == 'n':
        self.__tree = self.__tree.right
    self.addNodeTree(self.__tree)
    return self.getQuest()
```

4.12. ábra. Gyökér felülírása a játékmester válasza alapján

Az osztály néhány függvényének programkód szintű leírása következik az ábrákon:

```
def dataDecomp(self, no):
    yesCase = []
    noCase = []
    isFirst = False
    theFirstValuePlace = -1
    if self.__isEndGame:
        for i in range(0,len(self.__tree.data)):
            if self.__tree.data[i] != [] and not isFirst:
                theFirstValuePlace = i
                isFirst = True
    for j in range(0,len(no.data)):
        half = len(no.data[j])/2
        if len(no.data[j]) % 2 != 0:
            half = half + 1

        if self.__isEndGame:
            if theFirstValuePlace == j:
                yesCase.append(no.data[j][0:0+half])
                yesCase.append(no.data[j][half:])
                noCase.append(no.data[j][half:])
                noCase.append(no.data[j][0:0+half])
            else:
                yesCase.append(no.data[j][half:])
                yesCase.append(no.data[j][0:0+half])
                noCase.append(no.data[j][0:0+half])
                noCase.append(no.data[j][half:])
        else:
            yesCase.append(no.data[j][0:0+half])
            yesCase.append(no.data[j][half:])
            noCase.append(no.data[j][half:])
            noCase.append(no.data[j][0:0+half])
    return [yesCase,noCase]
```

4.13. ábra. A felbontást végző függvény

A fenti képen (4.13 ábra) az a függvény látható, amely segítségével készül el az adott csúcs felbontása. A kódon látszódik, hogy a fentebb leírtak alapján működik. Először lét-

rehoz két teljesen üres listát. Majd végigmegy egy *for* ciklus segítségével a paraméterként kapott csúcs *data* adattagjának listáin. Azokat a megfelelő módon elfelezi. Majd bepakolja őket két külön álló listába. A *yesCase* listába előbb az adott részlista első fele kerül bele, majd csak azután a második fele. Míg a *noCase* listába pont fordítva.

Jól látható, hogy megjelent egy *self.__isEndGame* adattag. Ez egy *bool* (logikai) változó, mely akkor kap *True* értéket, amikor a végjáték kezdődik. A végjáték akkor kezdődik, ha a gyökér *data* tagjában 2 vagy annál kevesebb szám lesz.

A 4.14 ábrán két függvény látszódik. A fenti függvény (*questionFromDecomp*) a paraméterként kapott csúcs *decomp* tagjából kiszedi azon lehetséges számokat, amikre a program rá szeretne kérdezni. Ezt úgy teszi meg, hogy veszi az első listát, azaz valójában a fentebb kialakított *yesCase* listát, és ennek minden páros számú részlistájából kimenti a számokat. Ezáltal megkapja egy listába az összes olyan számot, amit kérdésként akar majd feltenni.
Ezt a listát továbbadja a *questionToSetString* függvénynek, amely a listát intervallumokká alakítja, majd az intervallumokat ∪ jellel összeköti. Az *outQues* visszatérő adat lesz értékül adva a *self.__tree.quest* változónak.

```
def questionFromDecomp(self, no):
    ques = []
    for i in range(0,len(no.decomp[0]),2):
        ques.extend(no.decomp[0][i])
    if self.__isEndGame and len(ques) == 2:
        ques = [ques[0]]
    elif self.__isEndGame and sum([len(t) for t in no.data]) == 1:
        return "I think, X = " + str(ques[0]) + "\tI realised?"
    return self.questionToSetString(no,ques)
def questionToSetString(self,no,ques):
    stringQues = "X is in: "
    if len(ques) == 1:
        outQues = stringQues + str(ques)
    else:
        ques.sort()
        j = 0
        for i in xrange(0,len(ques)-1):
            l = True
            volt = False
            i = j
            while i <len(ques)-1 and l:
                if ques[i]+1 == ques[i+1] and i < len(ques)-1:
                    _begin  = i
                    j = i+1
                    while j<len(ques)-1 and ques[j]+1 == ques[j+1]:
                        j += 1
                    _end = j
                    stringQues += "[" + str(ques[_begin]) + ".." + str(ques[_end]) + "]"
                    l = False
                    volt = False
                elif not volt:
                    j = i+1
                    l = False
                    volt = True
                if (ques[i]+1 != ques[i+1] and ques[i-1]+1 != ques[i]) and i < len(ques)-1:
                    _begin = i
                    j = i+1
                    stringQues += "[" + str(ques[_begin]) + "]"
                    l = False
                if j == len(ques)-1 and volt:
                    _begin = j
                    stringQues += "[" + str(ques[_begin]) + "]"
                    j = i+1
                    l = False
        outQues = string.replace(stringQues,'][',"] U [")
    return outQues
```

4.14. ábra. A kérdést előállító függvények

4.7. Tesztelés

Két fajta tesztelést hajtottunk végre. Az egyikben azt néztük, hogy a felületen való gombnyomásokra megfelelően reagál-e a szoftver. A másikban pedig a játék szempontjából fontos függvényeket teszteltük.

Elsőként a felületen való tesztelést mutatjuk be.

- A Főmenü ablak tesztelése

Tesztelési eset	**Várt reagálás**
New Game gomb megnyomása	Megjelenik a következő ablak, ahol a játék „paramétereit" kell megadnia a felhasználónak.
Score gomb megnyomása	Az Eredménytábla felület jelenik meg.

- Az Eredménytábla (Score ablak) tesztelése

Tesztelési eset	**Várt reagálás**
Megjelenés	Az ablak megnyitásával egy időben a táblázatba betöltésre kerülnek az adatbázisból az adatok.
Back gomb megnyomása	Az előző ablak tér vissza. Jelen esetben ez a Főmenü ablakot jelenti.

- Az Új játék ablak tesztelése

Tesztelési eset	**Várt reagálás**
Megjelenés	Megjelenik a köszöntő szöveg és az űrlap.
Player és Master mód kiválasztása	Ez esetben a kiválasztott mód előtt jelenik meg a beszínezett kör. Alap helyzetben automatikusan a Player mód kerül kiválasztásra.
Űrlap mezőbe gépelés	Amennyiben helyes formában adja meg a felhasználó az adott paramétert, az adott szövegdoboz zöld színre vált.
OK gomb megnyomása és nem töltött ki minden mezőt	Ekkor a „You have not completed all fields!" hibaüzenet jelenik meg.
OK gomb megnyomása és nem megfelelő adat valamelyik mezőben	Ekkor olyan hibaüzenet jelenik meg, ami megmondja, hogy melyik érték a hibás. A hibaüzenetben az is olvasható, hogy milyen adat kerülhet az adott mezőbe.
OK gomb megnyomása és a mezők formátumai helyesek	A kiválasztott játékmód ablakára vált át.
Back gomb megnyomása	Az előző ablak jelenik meg újra. Jelen esetben ez a Főmenü ablakot jelenti.

- A Player mód ablakának tesztelése

Tesztelési eset	**Várt reagálás**
Megjelenés	Az ablak megnyitásával egy időben megjelenik a kérdező-, és a válaszadó felület is. Az ablakon helyesen jelennek meg a felhasználó által előző ablakon megadott adatok.
Lenyíló ablakra való kattintás	Megjelenik a feltehető kérdések listája.
Lenyíló ablakon belüli kiválasztás	Az adott kérdéshez tartozó mezők megjelennek.
Add interval gomb megnyomása	Amennyiben helyesen töltötte ki a két mező értékét az értékek megjelennek az ablak bal alsó sarkában. Hibás formátum esetén hibaüzenet jelenik meg.
Clear interval gombra kattintás	Amennyiben adott már hozzá intervallumot, azaz a bal alsó sarokban láthatók intervallumok, akkor azokat kitörölni. Amennyiben üres a lista „The interval is empty!" hibaüzenet jelenik meg.
HINT gomb megnyomása	Amennyiben van még segítsége, akkor egy felugró ablakban megjelenik a segítség. Ha már nincs több segítsége, akkor a „You have no more assist!" üzenet jelenik meg. Amennyiben a játék elején sincs segítsége, akkor a gomb nem kattintható.
HINT gomb fölé viszi az egeret	Megjelenik egy olyan szöveg, amely megmondja még hány segítsége van a játékosnak.
OK gomb megnyomása és nem töltött ki minden mezőt	Ekkor a „You have not completed all fields!" hibaüzenet jelenik meg.
OK gomb megnyomása és nem megfelelő adat valamelyik mezőben	Ekkor szintén hibaüzenet jelenik meg.
OK gomb megnyomása és a mezők formátumai helyesek	A kérdés benne a megadott értékkel/értékekkel megjelenik a válaszadó felületen a játékmester válaszával együtt.
Konkrétan rákérdez a gondolt számra és OK gombot nyom	Akkor megjelenik egy felugró ablak, amin a játék kimenetele látható (nyert/veszített) és a *Főmenü* ablak jelenik meg.
Back gomb megnyomása	Az előző ablak tér vissza. Jelen esetben ez a Új játék ablakot jelenti.

- Master mód ablakának tesztelése

Tesztelési eset	**Várt reagálás**
Megjelenés	Az ablak megnyitásával egy időben megjelenik az első kérdés is. Az ablakon helyesen jelennek meg a felhasználó által előző felületen megadott adatok.
Yes gomb megnyomás	Új kérdés jelenik meg. Lehetséges, hogy a gép egymás után többször is ugyanarra kérdez rá.
No gomb megnyomás	Új kérdés jelenik meg. Lehetséges, hogy a gép egymás után többször is ugyanarra kérdez rá.
A játékos rákérdez a gondolt számra és Yes vagy No gombnyomás történik	Ekkor a játéknak mindenképp vége. Megjelenik egy felugró ablak, amiben a játék kimenetele látszik. Valamint a Főmenü felület jelenik meg újra.
Back gomb megnyomása	Az előző ablak tér vissza. Jelen esetben ez a Új játék ablakot jelenti.

A fontosabb, összetettebb függvények tesztelése:

■ CompMaster osztály tesztelése

□ $\_\_init\_\_(inter, lieNum)$

Például: $\_\_master = CompMaster(11,2)$

Az osztályon belül beállítja az intervallum végét az első paraméternek, a hazugságok maximum számát pedig a 2. paraméter értékének.

Tegyük fel, hogy a gép a **7**-es számra gondolt.

□ $isInUnionInterval(low, high, quesNum)$

Például: $\_\_master.isInUnionInterval([1,5],[3,7],1)$ A függvény páronként végig megy az első két lista elemein. Azaz a fent példát véve:
Először megnézi, hogy a gondolt szám benne van-e az $[1, 3]$ halmazban, nincs benne így megy tovább. Megnézi a következő párt, azaz az $[5, 7]$ intervallumot. Benne van így meghívja a „hazudjak-e” (*lyingNow*) függvényt *True* paraméterrel.
A 3. paraméter azt mondja meg, hogy hanyadik kérdését tette fel a játékos. Ezt is továbbítja a „hazudjak-e” függvénynek.

- $isBigger(num, quesNum)$

  Például: *__master.isBigger(8,2)*

  Megnézi, hogy a gondolt szám szigorúan nagyobb-e, mint a paraméterként kapott *num* érték. Jelen esetben nem így *False* értéket ad első paraméterként a *lyingNow* függvénynek. Ezen felül a 2. paraméterként kapott kérdésszámot is továbbítja.

- Az *isBigger* függvény mintájára működik az *isSmaller*, az *isDivisible*, az *isSumDigitSmaller* és az *isSumDigitBigger* függvény is.

- $isPrime(quesNum)$

  Például: *__master.isPrime(3)*

  Csak a kérdésszámot kapja meg paraméternek. Eldönti, hogy a gondolt szám prím szám-e. A 7 szerinte prím szám, ezért *True* értéket továbbít a „hazudjak-e" függvénynek. Valamint természetesen a kérdésszámot is továbbadja.

- $thisIsTheNumber(num)$

  Például: *__master.thisIsTheNumber(10)*

  Már kérdésszámot nem kap, mivel itt nem hazudhat már, így nincs jelentősége. Ellenőrzi, hogy a *num* érték egyenlő-e az általa gondolt számmal. Tehát a $10 == 7$ erre *False* értéket fog visszakapni, így a „*You lose! The number is 7.*" szöveget fogja visszaadni.

- $getHint(quesNum)$

  Például: *__master.getHint(10)*

  Kigenerál egy véletlen számot 1 és e_n között, ami itt most azt jelenti, hogy 1 és 2 között. Az értéket továbbadja a *hintText* függvénynek, ami így visszaad egy segítséget. Legyen a kigenerált random szám az 1, ekkor azt a szöveget kapja a játékos segítségül, hogy „*I lied for 0 times.*". Azaz egyszer sem hazudott még.

- $lyingNow(answer, quesNum)$

  Például: *self.lyingNow(True,3)*

  Ezt a függvényt csak az osztályon belüli függvények hívjak, annak érdekében, hogy eldöntse akar-e hazudni miután tudja már, hogy mi lenne a hazugság nélküli válasz. Megnézi, hogy hányszor hazudott már eddig, mondjuk azt most, hogy egyszer sem. Ekkor kigenerál egy véletlen számot 0 és 1 között. Legyen ez most a 0.11767149162258184 szám, ez kisebb mint 0.75. Így a valódi, hazugságmentes választ kapja meg a játékos.

■ CompPlayer osztály tesztelése

Mivel az osztály működését már részletesen tárgyaltuk a 4.6.1 alfejezetben ezért csak egy átfogó példát mutatunk be.

- ☐ $\_\_init\_\_(self, inter, lieNum)$
  Például: *self.__player = CompPlayer(5,1)*
  Beállítja a hazugságok maximum számát 1-re. A gyökérnek (*self.__tree*) pedig az $[1, 2, 3, 4, 5]$ listát adja kezdeti értéknek.
- ☐ $addNodeTree(no)$
  Például: *self.addNodeTree(self.__tree)*
  Előállítja a paraméterként kapott *no* csúcs jobb és bal oldali gyerekét. Jelen esetben ezek:
  self.__tree.left.data = [[1, 2, 3], [4, 5]]
  self.__tree.right.data = [[4, 5], [1, 2, 3]]
- ☐ $questionFromDecomp(no)$ és $questionToSetString$ függvények egymásba ágyazva
  Például: *self.questionFromDecomp(self.__tree)*
  Előállítják azt a kérdést, hogy „*X is in: [1..3]*”. Ezt az értéket kapja meg a *self.__tree.quest* tag is.

■ MyDatabase osztály tesztelése

- ☐ $\_connection$
  Csatlakozik a mappában található *user_data.db* adatbázishoz.
- ☐ $\_close$
  Bezárja az adatbázist.
- ☐ $_create$
  Létrehozza a táblát a megfelelő paraméterekkel.
- ☐ $\_insert(inData)$
  Például: *db._insert(['Moni', '154', '3', 'Master / Responder', -100])*
  A paraméterként kapott adatokat beleírja az adatbázisba. Az *ID* mező értékét automatikusan beállítja, így azt nem kell paraméterként odaadni.
- ☐ $\_update(inDataID, inDataValue)$
  Például: *db._update(1,25)*
  Az első paraméterként kapott *ID* értéket megkeresi az adatbázisban és az ehhez tartozó *tryOut* értéket felülírja a 2. paraméterként kapott értékre. Azaz az előző

példa esetében, ha a „Moni” felhasználó az 1-es *ID* értéket kapta, akkor nála ezentúl a *Results* oszlopban a 25 érték fog látszódni.

5. fejezet

Összegzés

A szoftver írása közben sok érdekes probléma felvetődött. Többek között az is, hogy milyen adatszerkezettel lenne érdemes dolgozni a *CompPlayer* osztály esetében. Hosszas keresgélés és mérlegelés után döntöttem a fa szerkezet használata mellett. Úgy gondolom ez nem volt egy rossz választás. A program gyorsan működik, és továbbfejlesztés szempontjából a fa szerkezet egy jó alapnak tűnik.

Ennél a játékmódnál nem arra törekszik a szoftver, hogy a lehető legkevesebb kérdéssel találja ki a mágikus számot. Nagyon érdekes és összetett maga az a gondolkodás mód, ami a hazugságvektor használatához kell. Kisebb intervallumok esetén ezt papíron is végig lehet játszani. Megdöbbentő, hogy mennyi mindenre kell figyelni 1-1 csúcs esetében. Emiatt programozási szempontból is kihívás volt ennek megvalósítása.

A másik nagy osztály a *CompMaster*, ami esetében azok voltak a legnagyobb kérdések számomra, hogy mik azok a kérdések, amivel élvezetessé lehet tenni ezt a játékot? Mi az, amit én is kérdezni szeretnék egy játékmestertől? Úgy gondolom, hogy a mostani kérdések segítségével már lehet játszani és ki is lehet találni a gép által gondolt számot. Természetesen sok továbbfejlesztési lehetőség van benne.

Továbbfejlesztés szempontjából sok potenciál van a szoftverben.

A kinézet esetében el lehetne gondolkodni színesebb kinézetben.
A játékos mód esetében a felhasználónak lehetne azzal segíteni, hogy intervallumonként megmutatja, mik a mostani lehetséges számok, a játékmester válaszai alapján. Ehhez hozzá lehetne akár azt is venni, hogy ha módosítja a gép válaszát a válaszadó felületen, akkor milyen számokra gondolhatott.
A játékmester mód esetében pedig meg lehetne nézni, hogy milyen más módon lenne értelme kiíratni a kérdezett intervallumot. Akár egy számegyenes segítségével. Meg lehetne mutatni azt is a felhasználónak, hogy a gép tudja melyik számra hányszor hazudott már. Ehhez elég lenne hazugságonként kirajzolni 1-1 halmazt, amibe azok a számok vannak, amikre pontosan annyi hazugság érkezett eddig.

Az adatbázis jelen pillanatban lokálisan fut, ezt ki lehetne emelni egy szerverre és egy bejelentkező felülettel összekapcsolni.

Összességében a szoftver mostani formájában alkalmas játékra. A továbbfejlesztési tervekkel együtt pedig csak még izgalmasabb játék állna rendelkezésünkre.

Irodalomjegyzék

[1] Stanisław Ulam. *Adventures of a Mathematician.* 1976.

[2] Rényi Alfréd. *Napló az információelméletről.* 1976.

[3] Ferdinando Cicalese. *Fault-Tolerant Search Algorithms*, 2.fejezet. 2012.

[4] Windows 10 64 bit-es operációs rendszer:
https://www.microsoft.com/hu-hu/software-download/windows10

[5] Ubuntu 16.04 64 bit-es operációs rendszer: http://releases.ubuntu.com/16.04/

[6] PyQt Tutorial oldala: https://www.tutorialspoint.com/pyqt/index.htm

[7] László Vendel *GUI írás pyqt framework használatával.*
Elérhető: https://deep.reak.bme.hu/attachments/download/735

[8] PyInstaller hivatalos oldala: https://pythonhosted.org/PyInstaller/index.html

[9] Mac OS X hivatalos oldala: https://www.apple.com/hu/macos/high-sierra/

[10] PyInstaller speciális fájlának használata:
https://pythonhosted.org/PyInstaller/spec-files.html

[11] PyInstaller grafikus használatáról videó: https://www.youtube.com/watch?v=-WF8tCRFtlM

[12] SQLite hivatalos oldala: https://www.sqlite.org/index.html

[13] Adatbázis esetén az elsődleges kulcs leírása:
http://www.sze.hu/ szorenyi/sz03/htm/doc/elskulcs.htm

[14] Az UML nyelv hivatalos oldala: https://www.uml-diagrams.org/

[15] Az ICO fájlformátum leírása: https://en.wikipedia.org/wiki/ICO_(file_format)