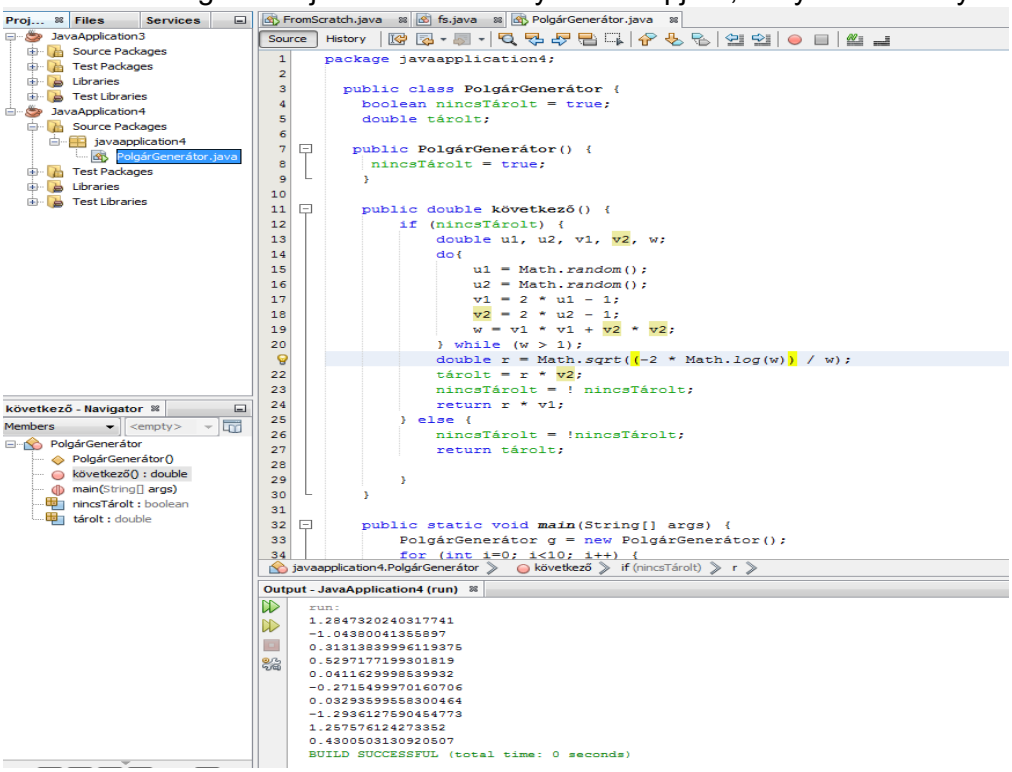
Készítette: Oszkocsil Krisztián

1.hét

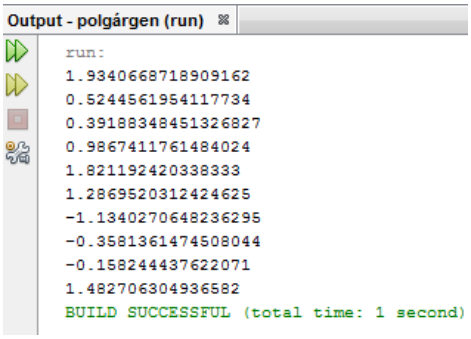
1.1. OO szemlélet:

Az objektumorientált vagy objektumelvű programozás egy programozási paradigma, ami az objektumok fogalmán alapul. Az objektumok egységbe foglalják az adatokat és a hozzájuk tartozó műveleteket. Az adatokat ismerik mezők, attribútumok, tulajdonságok néven, a műveleteket metódusokként szokták emlegetni. Az objektum által tartalmazott adatokon általában az objektum metódusai végeznek műveletet. A program egymással kommunikáló objektumok összességéből áll. A legtöbb objektumorientált nyelv osztály alapú, azaz az objektumok osztályok példányai, és típusuk ez az osztály.
A program létrehoz a polgárgenerátor osztály egy példányosított tagját, amit eltárol egy „g” változóban. A polgárgenerátor osztály tartalmaz egy boolean típusú (nincsTárolt) és egy double típusú (tárolt) változót, ezen kívül pedig egy következő metódust, amit a g.következő parancs tud meghívni. A következő metódus meghívása után a nincsTárolt egy igaz vagy egy hamis értéket ad vissza, ha pozitív visszajelzést ad, akkor ez azt jelenti, hogy az érték még nem volt eltárolva, ha viszont elvolt, akkor eltárolja és vissza adjuk a tárolt értket.
A feladat rávilágít az objektum orientált nyelvek alapjára, melyet az osztály alkot.



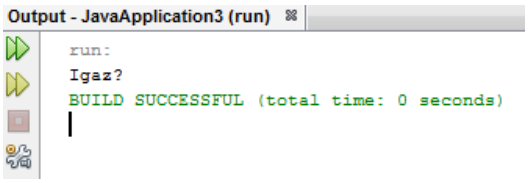
Futtatva:

```
public static void main(String[] args) {
    PolgárGenerátor g = new PolgárGenerátor();
    for (int i=0; i<10; i++) {
        System.out.println(g.következő());
    }
}
```



1.4. Yoda

A feladat egy olyan program írása volt, amely hogyha nem követi a Yoda conditions feltételeit
akkor egy NullPointerException-nal kilép. A NullPointerException egy RuntimeException vagyis a hiba csak futtatás után fog jelentkezni. Ezt unchecked kivételnek nevezzük ellenben a checked kivételekkel melyekre már futtatás előtt felhívja figyelmünket a fordító. NullPointerException-t leggyakrabban akkor kapunk, hogyha megpróbálunk egy olyan metódust vagy változót futtatni, vagy esetünkben elérni aminek referenciája a null értékre “mutat”, vagy olyan helyen próbálunk meg null értéket használni ahol a program egy “igazi” értéket várna.
Erre példa a YodaConditions programban megfigyelhető. Itt is az equals() metódus egy igazi értéket várna, így mikor egy null értéket tartalmazó változót adunk meg a program NullPointerExceptionnel kilép futás közben.

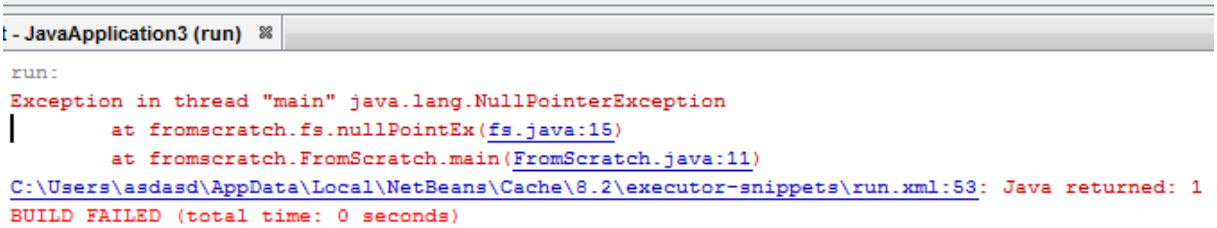



```
package fromscratch;


public class FromScratch {


    public static void main(String[] args) {
        fs fs = new fs();
        fs.notNullPointEx();
      //  fs.nullPointEx();

    }

}
```

```
package fromscratch;
/**
 *
 * @author Speedelek
 */
public class fs {
    fs(){

    }

    private final String myString = null;

    public void nullPointEx(){
        if (myString.equals("barmi")) {
            System.out.print("Nem szabad kiirnia semmit");
        }
    }

    public void notNullPointEx(){
        if ("barmi".equals(myString)) {
            System.out.println("Hamissal tér vissza");
        }else{
            System.out.println("Igaz?");
        }
    }
```

fs.nullPointEx() futtatása után:

```
package fromscratch;

public class FromScratch {

    public static void main(String[] args) {
        fs fs = new fs();
       // fs.notNullPointEx();
        fs.nullPointEx();
    }
}
```



1.5. Kódolás from scratch

A kódban egy long int típusú változó double típusba való kényszerítésére is felfigyelhetünk többszöri alkalommal a pontosabb értékek kiírása végett. Ez ugye több biten ábrázolt érték lesz, így értéktévesztés helyett pontosabb lesz az érték.

PiBBP.java

```
/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package piertek;


public class PiBBP {

    String d16PiHexaJegyek;

    public PiBBP(int d) {

        double d16Pi = 0.0d;

        double d16S1t = d16Sj(d, 1);
        double d16S4t = d16Sj(d, 4);
        double d16S5t = d16Sj(d, 5);
        double d16S6t = d16Sj(d, 6);

        d16Pi = 4.0d*d16S1t - 2.0d*d16S4t - d16S5t - d16S6t;

        d16Pi = d16Pi - StrictMath.floor(d16Pi);

        StringBuffer sb = new StringBuffer();

        Character hexaJegyek[] = {'A', 'B', 'C', 'D', 'E', 'F'};

        while(d16Pi != 0.0d) {

            int jegy = (int)StrictMath.floor(16.0d*d16Pi);
```

Output - JavaApplication5 (run)

```
run:
6C65E5308BUILD SUCCESSFUL (total time: 2 seconds)
```

```
ng n16modk(int n, int k) {

    = 1;
while(t <= n)
    t *= 2;

long r = 1;

while(true) {

    if(n >= t) {
        r = (16*r) % k;
        n = n - t;
    }

    t = t/2;

    if(t < 1)
        break;

    r = (r*r) % k;

}

return r;
}
```

2.hét

2.1. Liskov helyettesítés sértése

Javaban különböző osztályokkal könnyedén megsérthetjük a Liskov elvet.
Maga a Liskov-elv így hagzik:
-Az előfeltétel nem lehet erősebb a leszármazott osztályokban
-Az utófeltétel nem lehet gyengébb a leszármazott osztályokban
A Liskov elv megsértését könnyen tudjuk szemléltetni, tipikus példa erre az előadásokon és gyakorlatokon is említett madár-pingvin példa. A Liskov elv megsértésénél a pingvin objektum is kap olyan tulajdonságot a madár osztály miatt, mely szerint az repülni
tud.
Az alábbi programokban szintén a Liskov elv megsértésére kerül sor, miszerint, levetítve a programra, van egy faj(ember), illetve osztályok és annak a leszármazottai(a dolgozó ember, és a gyerek, illetve java nyelvben a játszó gyerek és a felnőtt). A leszármazottak sokban hasonlítanak, de nem feltétlenül képesek ugyan azokra, mászóval mások lehetnek a tulajdonságaik (programom esetében a gyerek nem dolgozik, a felnőtt nem játszik).

C++ nyelven:

Start here | liskovrafigyel.cpp

```cpp
class Ember {};

class Program {
public:
	void fgv ( Ember &ember ) {};
};

class DolgozoEmber : public Ember {
public:
	virtual void dolgozik() {};
};

class Felnőtt : public DolgozoEmber
{};

class Gyerek : public Ember
{};

int main ( int argc, char **argv )
{
	Program program;
	Ember ember;
	program.fgv ( ember );

	Felnőtt felnőtt;
	program.fgv ( felnőtt );

	Gyerek gyerek;
	program.fgv ( gyerek );

};
```

Java nyelven:

Liskov.java | Source | History

```java
package liskov;
public class Liskov {
    public static class Program{
        public static void fgv(Ember ember) {
            ember.játszik();
    }};

    public static class Ember {
        void játszik() {};
    };

    public static class Gyerek extends Ember
    {};

    public static class Felnőtt extends Ember
    {};

    public static void main(String[] args) {
     //Program program;
     Ember ember = new Ember();
     Program.fgv(ember);

     Gyerek gyerek = new Gyerek();
     Program.fgv(gyerek);

     Felnőtt felnőtt = new Felnőtt();
     Program.fgv(felnőtt);
    }
};
```

liskov.Liskov > main

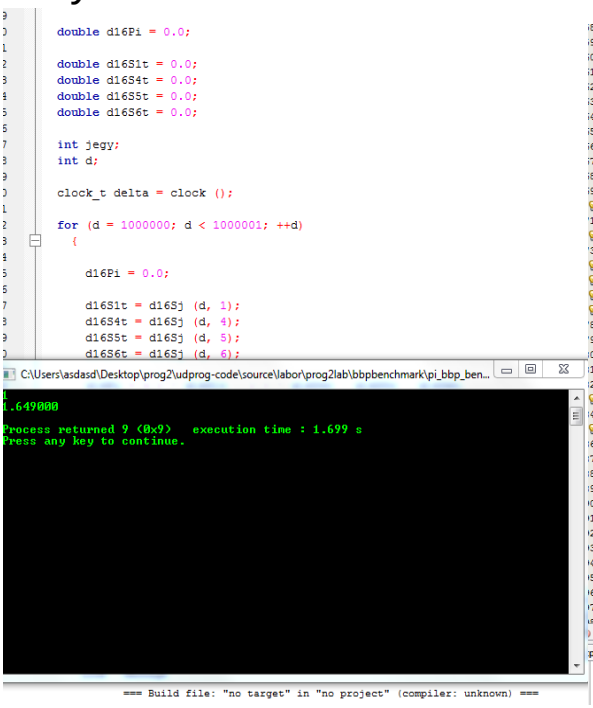
Output - liskov (run)

```
run:
BUILD SUCCESSFUL (total time: 0 seconds)
```

2.3.Anti OO

A C nyelven írt kód gyorsabban kiszámolta a jegyek értékét, mint a Java. A kódokban a már említett Long int változót double típusra kellett cserélni a már korábban megírt programban az értékek pontosságának megtartása miatt.
Az alábbi összevetésekben , latszik tisztan ,hogy a nyelvek amelyek közelebb vannak a memóriához , jobban teljesítenek , viszont kicsivel hosszabb a kód / nehezebb a szintaktika. De ugyanakkor , hogy kicsit mélyebbről lássuk a dolgokat , meg lehetne fogalmazni úgy , hogy a "low level language" programozó , látja maga előtt a memóriát , hiszen pontosan tudja ,hogy akarja elérni azt a viselkedést , amit lekódol , amíg a magasabb szintű programozási nyelvben a programozó csak olyan úton érheti el a megoldást , amilyen módon engedélyezi a nyelv . (alacsonyabb szintű nyelveknél több szabadsága van a programozónak).

C nyelven:

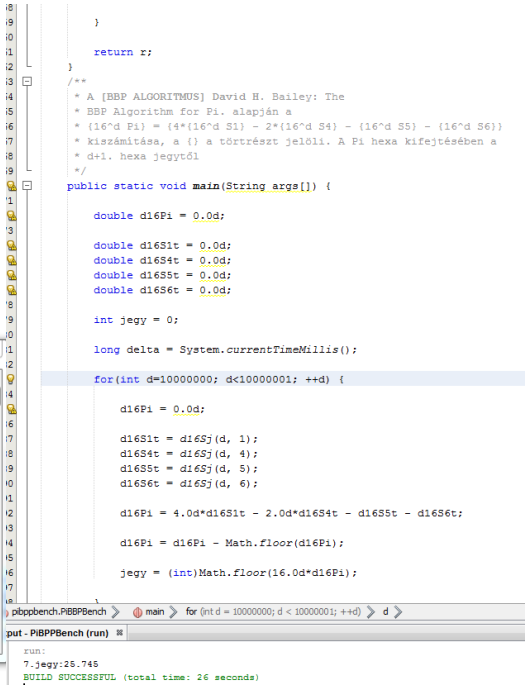


```
    double d16Pi = 0.0;

    double d16S1t = 0.0;
    double d16S4t = 0.0;
    double d16S5t = 0.0;
    double d16S6t = 0.0;

    int jegy;
    int d;

    clock_t delta = clock ();

    for (d = 1000000; d < 1000001; ++d)
      {

        d16Pi = 0.0;

        d16S1t = d16Sj (d, 1);
        d16S4t = d16Sj (d, 4);
        d16S5t = d16Sj (d, 5);
        d16S6t = d16Sj (d, 6);
```

```
C:\Users\asdasd\Desktop\prog2\udprog-code\source\labor\prog2lab\bbpbenchmark\pi_bbp_ben...
1.649000

Process returned 9 (0x9)   execution time : 1.699 s
Press any key to continue.
```

```
=== Build file: "no target" in "no project" (compiler: unknown) ===
```

Java nyelven:



```
        }

        return r;
    }
    /**
     * A [BBP ALGORITMUS] David H. Bailey: The
     * BBP Algorithm for Pi. alapján a
     * {16^d Pi} = {4*{16^d S1} - 2*{16^d S4} - {16^d S5} - {16^d S6}}
     * kiszámitása, a {} a törtrészt jelöli. A Pi hexa kifejtésében a
     * d+1. hexa jegytől
     */
    public static void main(String args[]) {

        double d16Pi = 0.0d;

        double d16S1t = 0.0d;
        double d16S4t = 0.0d;
        double d16S5t = 0.0d;
        double d16S6t = 0.0d;

        int jegy = 0;

        long delta = System.currentTimeMillis();

        for(int d=10000000; d<10000001; ++d) {

            d16Pi = 0.0d;

            d16S1t = d16Sj(d, 1);
            d16S4t = d16Sj(d, 4);
            d16S5t = d16Sj(d, 5);
            d16S6t = d16Sj(d, 6);

            d16Pi = 4.0d*d16S1t - 2.0d*d16S4t - d16S5t - d16S6t;

            d16Pi = d16Pi - Math.floor(d16Pi);

            jegy = (int)Math.floor(16.0d*d16Pi);
```

```
pibppbench.PiBBPBench  main  for (int d = 10000000; d < 10000001; ++d)  d
put - PiBBPBench (run)
run:
7.jegy:25.745
BUILD SUCCESSFUL (total time: 26 seconds)
```

5.Ciklomatikus komplexitás

A ciklomatikus komplexitás a kódunknak a bonyolultságát számolja ki. A különböző ciklusoknak, elágazásoknak adja meg a számosságát.

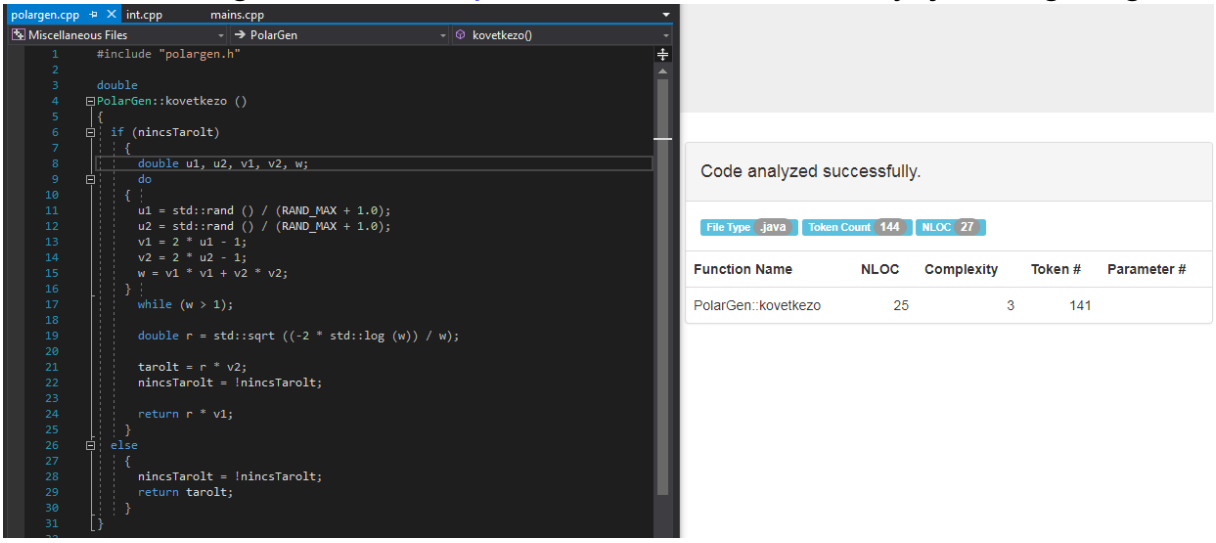
A ciklomatikus komplexitás értéke:

$$M = E - N + 2P$$

ahol

- E: A gráf éleinek száma
- N: A gráfban lévő csúcsok száma
- P: Az összefüggő komponensek száma

Egy korábban már megírt kódot, a Polártranszformáció kódját alkalmaztam a feladat megoldásához C++ nyelven.
A feladat elvégzéséhez a http://www.lizard.ws/ oldal nyújtott segítséget.

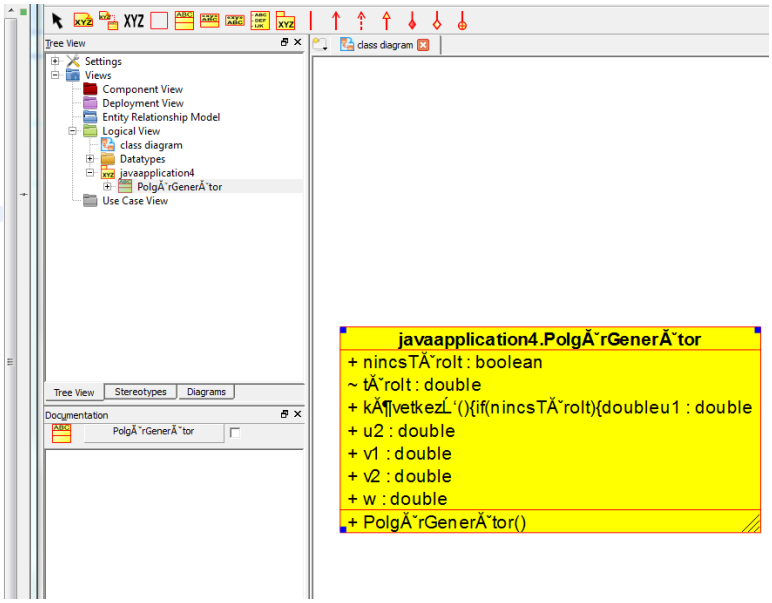


3.hét

3.1.Reverse engineering UML osztálydiagram

A feladatban egy UML diagramot készítettem a Polárgenerátor feladathoz. Ebben a feladatban visszafelé dolgozunk, tehát előbb dolgozunk, vagyis megírjuk a programunkat, aztán készítjük el a dokumentációt (ilyenkor mások számára egyszerűbbé tesszük a programunk megértését).
A program használata letisztult, bár a karakterisztikája kevésnek bizonyult az ékezetesbetűk felismerésének hiánya miatt.
Maga a kód kissé másként épül fel mint az importált alap, de a főbb osztályok, függvények felismerhetők, könnyen megtalálható helyen vannak.

```
package javaapplication4;

public class PolgárGenerátor {
  boolean nincsTárolt = true;
  double tárolt;

  public PolgárGenerátor() {
    nincsTárolt = true;
  }

  public double következő() {
      if (nincsTárolt) {
          double u1, u2, v1, v2, w;
          do{
              u1 = Math.random();
              u2 = Math.random();
              v1 = 2 * u1 - 1;
              v2 = 2 * u2 - 1;
              w = v1 * v1 + v2 * v2;
          } while (w > 1);
          double r = Math.sqrt((-2 * Math.log(w)) / w);
          tárolt = r * v2;
          nincsTárolt = ! nincsTárolt;
          return r * v1;
      } else {
          nincsTárolt = !nincsTárolt;
          return tárolt;

      }
  }

  public static void main(String[] args) {
      PolgárGenerátor g = new PolgárGenerátor();
      for (int i=0; i<10; i++) {
          System.out.println(g.következő());
      }
  }
}
```



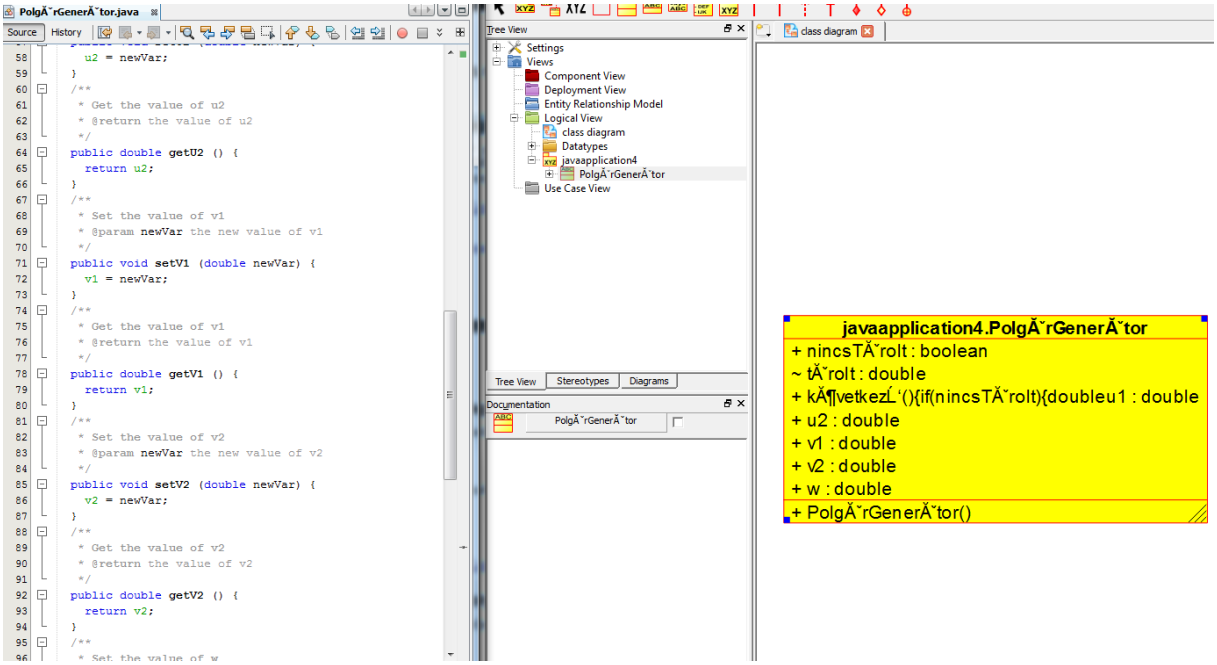
3.2.Forward engineering UML osztálydiagram

generált java kód: C:\Program Files\umbrello\javaapplication4

A feladat az, hogy egy meglévő osztálydiagrammból forward engineeringel kódot kapjunk. Ehhez a feladathoz az előző feladatban kapott UML diagrammot fogjuk felhasználni és szintúgy a VIsual Paradigm szoftverével oldjuk meg.

Ebben az esetben szintén a Tools/Code/ -on belülrol választunk, csak most nem az instant reverset hanem az instant generatort.

Az exportált kód formája az eredeti kódhoz képest sokban eltér, az üres sorok, felesleges kommenttel teli sorok száma rengeteg, ez a kód méretét nagyban növeli, széthúzza azt. Maga a kód felimerhető, de sok a változás.



3.4.BPMN

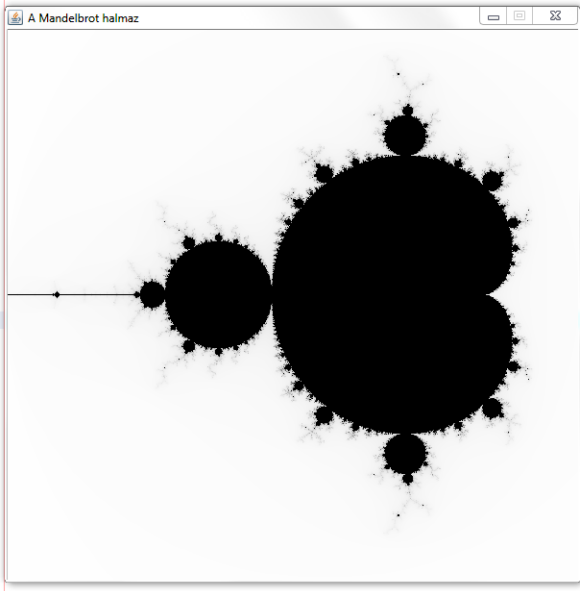
A kódoláshoz képest egy könnyebb technika, a prezentációkészítést és az információ továbbadásában sokat segíthet. Egy szemléltetésszerű tervezésben is átlátható, letisztult képet adhat, bár a bonyolultabb kódokat kevésbé képes az ember ennek segítségével ábrázolni.

4.hét:

4.1 Encoding

Ebben a feladatban a MandelbrotHalmazNagyito programot kellett lefuttatni. A **matematikában** a Mandelbrot-halmaz azon c **komplex számokból** áll (a „komplex számsík” azon pontjainak mértani helye, halmaza), melyekre az alábbi (komplex szám értékű) **rekurzív sorozat**. Nem tart végtelenbe, azaz abszolút értékben (hosszára nézve) korlátos. A Mandelbrot-halmazt a komplex számsíkon ábrázolva, egy nevezetes (és hasonnevű) fraktálalakzat adódik.

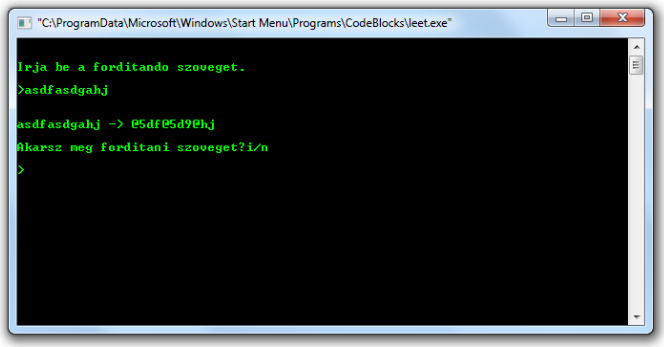
```
/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package l334d1c4;

/**
 *
 * @author asdasd
 */
public class MandelbrotHalmaz extends java.awt.Frame implements Runnable {
    /** A komplex sík vizsgált tartománya [a,b]x[c,d]. */
    protected double a, b, c, d;
    /** A komplex sík vizsgált tartományára feszített
     * háló szélessége és magassága. */
    protected int szélesség, magasság;
    /** A komplex sík vizsgált tartományára feszített hálónak megfelelő kép.*/
    protected java.awt.image.BufferedImage kép;
    /** Max. hány lépésig vizsgáljuk a z_{n+1} = z_n * z_n + c iterációt?
     * (tk. most a nagyítási pontosság) */
    protected int iterációsHatár = 255;
    /** Jelzi, hogy éppen megy-e a számítás? */
    protected boolean számításFut = false;
    /** Jelzi az ablakban, hogy éppen melyik sort számoljuk. */
    protected int sor = 0;
    /** A pillanatfelvételek számozásához. */
    protected static int pillanatfelvételSzámláló = 0;
    /**
     * Létrehoz egy a Mandelbrot halmazt a komplex sík
     * [a,b]x[c,d] tartománya felett kiszámoló
     * <code>MandelbrotHalmaz</code> objektumot.
     *
     * @param      a              a [a,b]x[c,d] tartomány a koordinátája.
     * @param      b              a [a,b]x[c,d] tartomány b koordinátája.
     * @param      c              a [a,b]x[c,d] tartomány c koordinátája.
     * @param      d              a [a,b]x[c,d] tartomány d koordinátája.
     * @param      szélesség      a halmazt tartalmazó tömb szélessége.
     * @param      iterációsHatár a számítás pontossága.
     */
    public MandelbrotHalmaz(double a, double b, double c, double d
```



4.3 L33T

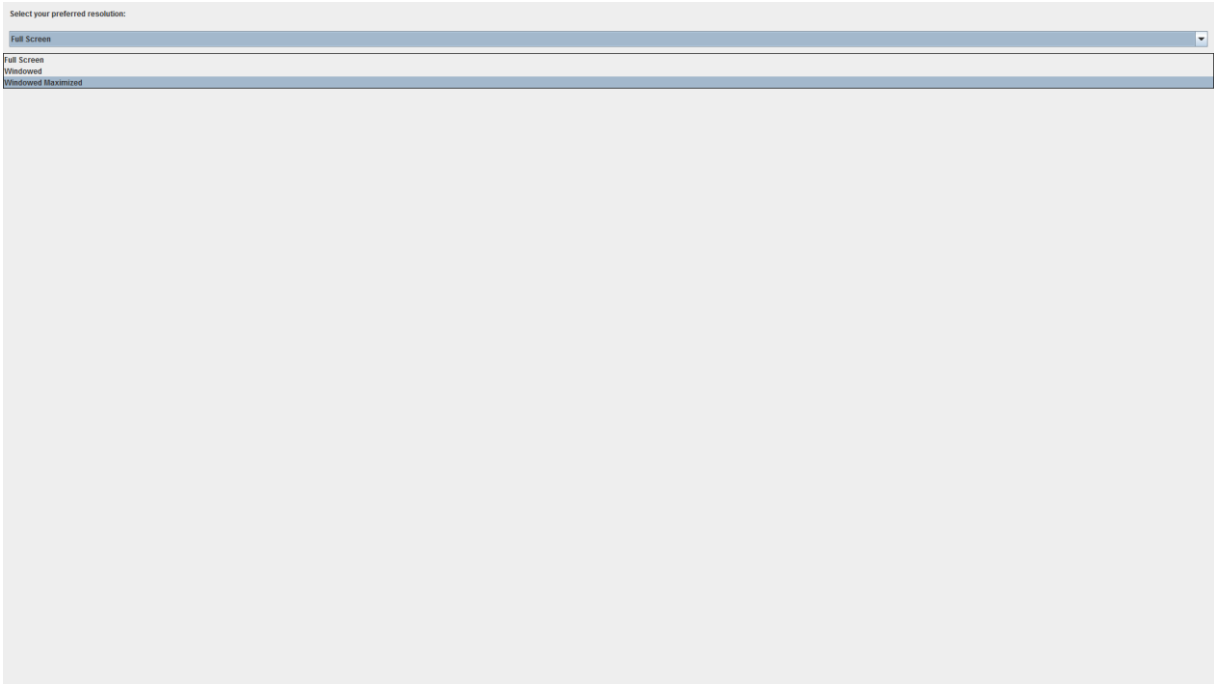
A leet egy olyan írásmód, amelyik alkalmazkodva a modern írásbeliséghez, merőben szimbolikus. A szimbolikusság egyik jellemzője, hogy az értelmezés alapja a közmegállapodás, ám egy szimbólumrendszer csak azoknak a személyeknek lesz értelmezhető, amelyek be vannak vonva ebbe a közmegállapodásba, be vannak avatva a szabályokba. Az írásbeliséget az iskolában tanuljuk meg, azaz a tanító avat be minket az írásbeliség szabályrendszerébe.

```
string leet[52]={"4","8","(","D","3","F","9","H","1","J","|<","L","M","N","0","P","Q","R","5","7","U","\/","W","X","Y","2","@","b","(","d","3","f","9","h","i"
int main()
{

    //Enters the main game loop for the program
    enterGameLoop();

    return 0;
}
void enterGameLoop()
{
     do
    {
    word = askWord("\n\nIrja be a forditando szoveget.\n\n>");


    word = convertWord(word);

    cout << word << "\n\nAkarsz meg forditani szoveget?i/n\n\n>";

    getline(cin,again);
    }
    while(again == "i");
}
string askWord(string prompt)
{

       string word;

       cout << prompt;

       getline(cin,word);

       cout << "\n\n" << word << " -> ";

       return word;
}
string convertWord(string word)
{
      for(size_t i = 0; i < word.size(); i++)
{
   for(int j = 0; j < 52; j++)
```



4.4 Fullscreen:

A fullscreen programozási feladatban meg kellett írni egy teljes képernyőre kiterjesztett programot, melybe a különböző paraméterek, mint például a vonalvastagság, méret, név megadása után létrehoztam egy lenyíló ablakot, ezután beletettem az ablak módot, ami egy 600x400 pixel méretű ablakba teszi a programot, illetve a kiterjesztett ablak módot, ami a teljes képernyőre kiteszi a programot, csak az ablak keretét meghagyja.
A program megnyitás után egyből teljes képernyőre nyílik meg, ezután lehet állítani amíg ki nem lépünk belőle.



Maga a kód:

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.border.*;

public class Resolution extends JPanel
                        implements ActionListener {
    static JFrame frame = new JFrame("Resolution");
    String currentPattern;

    public Resolution() {
        setLayout(new BoxLayout(this, BoxLayout.PAGE_AXIS));
        String[] patternExamples = {
                "Full Screen",
                "Windowed",
                "Windowed Maximized",
                };

        currentPattern = patternExamples[0];

        JLabel patternLabel = new JLabel("Select your preferred resolution: ");

        JComboBox patternList = new JComboBox(patternExamples);
        patternList.addActionListener(this);

        JPanel patternPanel = new JPanel();
        patternPanel.setLayout(new BoxLayout(patternPanel,
                               BoxLayout.PAGE_AXIS));
        patternLabel.setBorder(new EmptyBorder(10, 10, 10, 10));
        patternList.setBorder(new EmptyBorder(10, 10, 10, 10));
        patternPanel.add(patternLabel);
        patternList.setAlignmentX(Component.LEFT_ALIGNMENT);
        patternPanel.add(patternList);
        add(patternPanel);

        add(Box.createRigidArea(new Dimension(0, 1000)));


    }

    public void actionPerformed(ActionEvent e) {
        JComboBox cb = (JComboBox)e.getSource();
        String newSelection = (String)cb.getSelectedItem();
        switch (newSelection){
            case "Windowed": windowed(); break;
```

```
    public void actionPerformed(ActionEvent e) {
        JComboBox cb = (JComboBox)e.getSource();
        String newSelection = (String)cb.getSelectedItem();
        switch (newSelection){
            case "Windowed": windowed(); break;

            case "Full Screen": fullScreen(); break;

            case "Windowed Maximized": winmaxed(); break;
        }
    }

    private static void createGUI() {
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        JComponent newContentPane = new Resolution();
        frame.setContentPane(newContentPane);
    }

    private static void fullScreen() {
        frame.dispose();
        frame.setExtendedState(JFrame.MAXIMIZED_BOTH);
        frame.setUndecorated(true);
        frame.setVisible(true);
    }

    private static void windowed() {
        frame.dispose();
        frame.setUndecorated(false);
        frame.setVisible(true);
        frame.getContentPane().setPreferredSize(new Dimension(600, 400));
        frame.pack();
    }

    private static void winmaxed() {
        frame.dispose();
        frame.setUndecorated(false);
        frame.setVisible(true);
        frame.setExtendedState(JFrame.MAXIMIZED_BOTH);
    }

    public static void main(String[] args) {
        javax.swing.SwingUtilities.invokeLater(new Runnable() {
            public void run() {
                createGUI();
                windowed();
```

5.hét:

5.3 RSA hibásan implementálva:

Az RSA-eljárás **nyílt kulcsú** (vagyis „aszimmetrikus”) titkosító **algoritmus**, melyet 1976-ban **Ron Rivest**, **Adi Shamir** és **Len Adleman** fejlesztett ki (és az elnevezést nevük kezdőbetűiből kapta). Ez napjaink egyik leggyakrabban használt titkosítási eljárása. Az eljárás elméleti alapjait a **moduláris számelmélet** és a **prímszámelmélet** egyes tételei jelentik.

```
package rsapelda;

public class RSAPelda {


    public static void main(String[] args) {

        KulcsPar jszereplo = new KulcsPar();
        String tisztaSzoveg = "Hello, Vilag!";
        byte[] buffer = tisztaSzoveg.getBytes();
        java.math.BigInteger[] titkos = new java.math.BigInteger[buffer.length];

        for (int i = 0; i < titkos.length; ++i) {
            titkos[i] = new java.math.BigInteger(new byte[] {buffer[i]});
            titkos[i] = titkos[i].modPow(jszereplo.e, jszereplo.m);
        }

        for (int i = 0; i < titkos.length; ++i) {
            titkos[i] = titkos[i].modPow(jszereplo.d, jszereplo.m);
            buffer[i] = titkos[i].byteValue();
        }

        System.out.println(new String(buffer));
    }
}
```

```
package rsapelda;
class KulcsPar {

    java.math.BigInteger d, e, m;

    public KulcsPar() {

        int meretBitekben = 700 * (int) (java.lang.Math.log((double) 10) / java.lang.Math.log((double) 2));
        java.math.BigInteger p = new java.math.BigInteger(meretBitekben, 100, new java.util.Random());
        java.math.BigInteger q = new java.math.BigInteger(meretBitekben, 100, new java.util.Random());
        m = p.multiply(q);
        java.math.BigInteger z = p.subtract(java.math.BigInteger.ONE).multiply(q.subtract(java.math.BigInteger.ONE));
        do {
            do {
                d= new java.math.BigInteger(meretBitekben, new java.util.Random());
            } while (d.equals(java.math.BigInteger.ONE));
        } while (z.gcd(d).equals(java.math.BigInteger.ONE));

        e = d.modInverse(z);
    }
}
```

5.2 Másoló-mozgató:

Másoló, illetve mozgató konstruktorok külön példákba futtatva: Az esszében bővebben le vannak írva a dolgok.

5.5.Esszé:

Másoló-Mozgató konstruktorok

Az int.h és a az int.cpp tartalmazza az osztály függvényeinek definícióját illetve deklarációját.
Implementálva van a kon- illetve destruktor, a másoló- illetve mozgató értékadás és kontruktor is.
Az implementálás célja, hogy ne az alapértelmezett dolgokat használjuk, hiszen az bitről bitre másol.
Ezesetben a megvalósítás majdnem hogy kötelező, mert az osztályunk tartalmaz dinamikus adattagot (ertek) is.
A main.cpp példákat mutat, hogy bizonyos helyzetekben melyik kerül felhasználásra.
Vegyük sorba a példákat!

1) Int a(32) és Int b(99)

Csak példányosítunk, egyszerűen, használva az inicializáló konstruktort.

2) Int c = a

Itt létrehozzunk egy 'c' objektumot rögtön az 'a' objektum értékével.
Azaz lemásoljuk; másoló konstruktor.

3) Int d(b.getErtek()-1)

Itt 'd' objektumot hozunk létre, amelynek értéke b.getErtek()-1, azaz itt egy egyszerű inicializáló kunstruktor hívódik meg.

4) d = a és d = b

Jelen esetben a 'd' és 'a' objektumok már léteznek.
A 'a' objektum értékét kapja meg a 'd' objektum.
Másoló értékadás, mivel csak átmásoljuk az értéket, nem kényszerítjük ki a mozgatást.

5) Int e

Egyszerű példányosítás, de nem használunk inicializáló konstuktor, mivel nem adunk rögtön értéket neki. Sima kontruktor.

6) e = move(b)

Az 'e' objektumunk már létezik. A 'b' is már létezik.
Itt már használjuk a 'move' függvényt, amivel kikényszerítjük a mozgatást.
Értékadás történik, mivel az 'e' objektumot már létezik.
Fontos megjegyezni, hogy ekkor a 'b' objetum 'megszűnik'.

7) Int f = move(a)

Hasonló a fentihez, bár itt mozgató konstruktor hívódik meg.
Kitérő:

```
vector<Int> v;
v.reserve(10);
```

Csinálunk egy vektort, ami Int-ekből áll.
Beállítjuk, hogy 10-nek csináljon helyet egyből, egyébként ha nem, akkor a benne lévőket mindig átmásolja, másoló konstruktorok segítségével.
Folytassuk!

8) v.push_back(d)

A v vektorba 'másoljuk be' a 'd' (már létrehozott) objektumot.
Másoló konstruktor.

9) v.push_back(move(d))

Hasonló a fentihez, bár itt kikényszerítjük a mozgatást.
A 'd' objektum ezután 'megszűnik'.

10) v.push_back(Int(777))

Itt három dolog történik.
Példányosítunk egy objetumot inicializáló konstruktorral: inicializáló konstruktor.
Átmásoljuk azt a v vektorba: mozgató konstruktor.
Rögtön töröljük is az új (névtelen) objektumot, mivel csak ideiglenesen hoztuk létre: destruktor.
A megfelelő destruktorok meghívására nem térnék ki, akkor hívódik meg, amikor a program végéhez érünk, létrehozási sorrendben, vagy akkor, amikor már nincs rá szükség (lásd 10. pont).
A program futtatásokor hasonlóan látszik minden, ami fentebb említve lett.

6. hét:

6.3 STL map:

Az STL map nevezetű program egy meghatározott minimum és maximum érték között (0-1000) vesz egy random számot, hozzárendel egy embert. Eleinte kiírja a map elemeit, vagyis az emberek ”azonosítóját” majd később feltölti a map minden elemét egy az előbbiekben tárgyalt random értékkel, a legvégén pedig ezeket sorba rendezve írja ki.

```
int randomNumber(int min, int max){
    return min+ rand()%(max-min);
}

void mapPrinter(std::map<std::string, int>& in){
    for(auto & it: in){
        std::cout << it.first << " " << it.second << std::endl;
    }
}

void mapFiller(std::map<std::string, int> &in){
    for(int i = 0; i<10; i++){
        std::string temp = "ember" + std::to_string(i);
        in[temp] = randomNumber(0, 1000);
    }
}

std::vector<std::pair<std::string, int>> mapSorter(std::map<std::string, int>& in){
    std::vector<std::pair<std::string, int>> or


    for(auto &it :in){
        std::pair<std::string, int> temp{it.fir
        //std::cout << it.first << "\t" << it.s
            ordered.push_back(temp);
    }

    std::sort (
                std::begin ( ordered ), std::en
        [ = ] ( auto && p1, auto && p2 ) {
                return p1.second > p2.second;
        }
    );



    //std::sort(std::begin(ordered), std::end(o

    return ordered;
};
```

```
"C:\Users\asdasd\Desktop\egyetem\prog2\kÚsz feladatok\5.hÚt\copy-move\mapPelda.exe"
ember0 54
ember1 477
ember2 881
ember3 223
ember4 589
ember5 353
ember6 580
ember7 899
ember8 505
ember9 123
Rendezett map elemei:
ember7 899
ember2 881
ember4 589
ember6 580
ember8 505
ember1 477
ember5 353
ember3 223
ember9 123
ember0 54

Process returned 0 (0x0)   execution time : 0.036 s
Press any key to continue.
```

6.4 Alternatív tabella

A `comprateTo` metódus összehasonlítja az objektumot az átvett objektummal, és visszatérési értéke negatív egész, nulla, vagy pozitív egész, ha az átvevő objektum kisebb, egyenlő vagy nagyobb, mint az átvett objektum. Ha a metódus nem tudja összehasonlítani a két objektumot, akkor `ClassCastException`-t dob.

```
  {1, 3, 1, 0, 2, 0, 0, 0, 0, 1, 1, 1, 1, 1, 0, 0},
  {0, 0, 1, 0, 1, 0, 3, 1, 1, 0, 0, 1, 2, 1, 3, 0}
};

int[][] pontotSzerez = new int[kereszt.length][kereszt.length];

for (int i = 0; i < pontotSzerez.length; ++i) {
  for (int j = 0; j < pontotSzerez[i].length; ++j) {
    pontotSzerez[i][j] = 0;
  }
}

for (int i = 0; i < pontotSzerez.length; ++i) {
  for (int j = 0; j < pontotSzerez[i].length; ++j) {

      switch (kereszt[i][j]) {
          case 1:
              // zφld

              ++pontotSzerez[i][j];
              break;
          case 2:
              // sαrga

              ++pontotSzerez[i][j];
              ++pontotSzerez[i][i];
```

```
Matrix
run)
2857142857142, 0.08333333333333333, 0.0, 0.1111111111111111, 0.0, 0.08333333333333333, 0.1111111111111111, 0.0, 0.0, 0.083333333333333
5714285714285, 0.08333333333333333, 0.08333333333333333, 0.1111111111111111, 0.14285714285714285, 0.08333333333333333, 0.2222222222222
5714285714285, 0.0, 0.16666666666666666, 0.0, 0.14285714285714285, 0.08333333333333333, 0.1111111111111111, 0.1, 0.1, 0.16666666666666
2857142857142, 0.0, 0.08333333333333333, 0.0, 0.14285714285714285, 0.08333333333333333, 0.0, 0.1, 0.1, 0.08333333333333333, 0.11111111
```

6.5 Esszé:

stl map

A feladat célja egy saját allokátor elkészítése volt. Az STL konténerek valamint a string osztály erőforrás tárolóként működnek.

Ezek lefoglalnak és felszabadítanak memóriát, annak céljából, hogy az elemeiket tárolni tudják. Ennek érdekében ok allokátorokat használnak. Az alapvető célja egy allokátornak, hogy memóriát biztosítson egy adott típus számára, valamint egy olyan helyet jelent ahova a már nem szükséges memóriát vissza lehet téríteni/"adni".

A kódom:

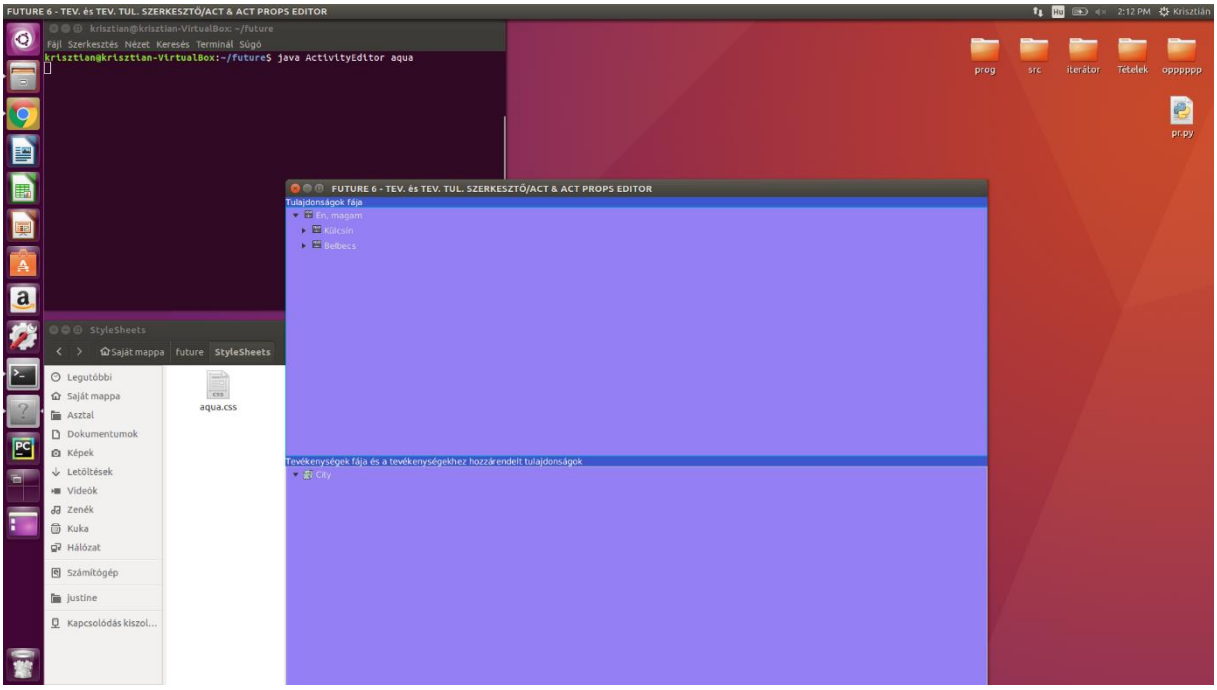
A melgelelő include-ok után:

```
int randomNumber(int min, int max);
void mapPrinter(std::map<std::string, int>& in);     //kiirja a map elemeit
void mapFiller(std::map<std::string, int> &in);     //feltolti a mapot nevekkel és random pontokkal
std::vector<std::pair<std::string, int>> mapSorter(std::map<std::string, int> &in); //a map elemeit a pont szerint rendezve visszaadka egy vektorba
//bool comparePoints(auto &&a, auto &&b);

int main(){
    srand((unsigned int)time(NULL));   // előjeles egész típusú random szám generálása történik
    std::map<std::string, int> ikszde;
    mapFiller(ikszde);
    std::cout << "Rendezetlen map elemei:" << std::endl;
    mapPrinter(ikszde);
    std::vector<std::pair<std::string, int>> orderedIkszde = mapSorter(ikszde);

    std::cout << "Rendezett map elemei:" << std::endl;
    for(auto& i :orderedIkszde){
        std::cout << i.first << " " << i.second << std::endl;
    }
    return 0;
}

int randomNumber(int min, int max){
    return min+ rand()%(max-min);               //minimum és maimum érték közötti random értéket ad vissza
}
void mapPrinter(std::map<std::string, int>& in){
    for(auto & it: in){
        std::cout << it.first << " " << it.second << std::endl;
    }
}
void mapFiller(std::map<std::string, int> &in){
    for(int i = 0; i<10; i++){
        std::string temp = "ember" + std::to_string(i);
        in[temp] = randomNumber(0, 1000);
    }
}
std::vector<std::pair<std::string, int>> mapSorter(std::map<std::string, int>& in){
    std::vector<std::pair<std::string, int>> ordered;

    for(auto &it :in){
        std::pair<std::string, int> temp{it.first, it.second};
        //std::cout << it.first << "\t" << it.second << std::endl;
            ordered.push_back(temp);
    }
    std::sort (
                std::begin ( ordered ), std::end ( ordered ),
        [ = ] ( auto && p1, auto && p2 ) {
                return p1.second > p2.second;
        }
    );

    //std::sort(std::begin(ordered), std::end(ordered), comparePoints)
    return ordered;
};
```

7. hét:

7.1 ActivityEditor:

Feladatomban a Tevékenység szerkesztő programot kellett módosítanom a Future programhoz.
Jelenleg hozzáadtam css formátumú egy textúra packot, amivel a színét változtatom.



7.2 OOCWC Boost ASIO hálózatkezelése

A scanf használata:

forrás: https://www.geeksforgeeks.org/g-fact-31/

A lexerben a scanf segítségével fogunk tudni adatokat beolvasni. A carlexer.ll-ben megtalálható, hogy milyen formájú
kifejezések várhatóak. Ezek beolvasására használjuk a scanf függvényt. A függvény megvizsgálja a bejövő adatokat,
és ha megfelelő a formátumuk, akkor feldolgozza őket, változóknak értékül adja.

```
 GANGSTERS      "<gangsters"
 STAT   "<stat"
 DISP   "<disp>"
 %%
 {DISP}                                 {
                                          m_cmd = 0;
                                        }
 {POS}{WS}{INT}{WS}{INT}{WS}{INT}       {
                                          std::sscanf(yytext, "<pos %d %u %u", &m_id, &from, &to);
                                          m_cmd = 10001;
                                        }
 {CAR}{WS}{INT}                         {
                                          std::sscanf(yytext, "<car %d", &m_id);
                                          m_cmd = 1001;
                                        }
 {STAT}{WS}{INT}                              {
                                          std::sscanf(yytext, "<stat %d", &m_id);
                                          m_cmd = 1003;
                                        }
 {GANGSTERS}{WS}{INT}                   {
                                          std::sscanf(yytext, "<gangsters %d", &m_id);
                                          m_cmd = 1002;
                                        }
 {ROUTE}{WS}{INT}{WS}{INT}({WS}{INT})*  {
                                 int size{0};
                                 int ss{0};
                                 int sn{0};

                                 std::sscanf(yytext, "<route %d %d%n", &size, &m_id, &sn);
                                 ss += sn;
                                 for(int i{0}; i<size; ++i)
                                 {
                                   unsigned int u{0u};
                                   std::sscanf(yytext+ss, "%u%n", &u, &sn);
                                   route.push_back(u);
                                   ss += sn;
                                 }
                                 m_cmd = 101;
                               }
 {INIT}{WS}{WORD}{WS}("c"|"g")  {
                                 std::sscanf(yytext, "<init %s %c>", name, &role);
```

7.4 Qt slot-signal:

A slot-signal mechanizmus a Qt központi eleme/funkciója.
A Qt-ban az objektumok képesek magukból signal-okat kiadni, ha történt vele valami fontos – például megváltozott az állapota.
A Qt-ban az objektumoknak lehetnek slot-jaik is, melyekkel képesek hallgatózni: figyelni arra, ha egy másik objektum signalt küld.

Amikor a signal-t a slot-hoz kötjük, tulajdonképp elmondjuk a slot-nak, hogy melyik objektum melyik signaljára kell figyelnie.
A signalt küldő objektum nem tudja, hogy figyelnek-e rá.

Minden osztály, amely a 'QObject'-ből öröklődik, vagy maga a QObject, képes előre megadott/saját signalokat és slotokat tartalmazni.
Lehetséges az, hogy egy signalt több slothoz, illetve egy slotot több signalhoz kapcsoljunk.
Sőt, signalt signalhoz is kapcsolhatunk. (Ekkor mindkettő meghívódik megfelelő sorrendben.)

A signalokat emit segítségével "hívjuk" meg.

Az egész hasonló elven működik, mint Javaban az Observer interfész és a hozzá tartozó Observable osztály.

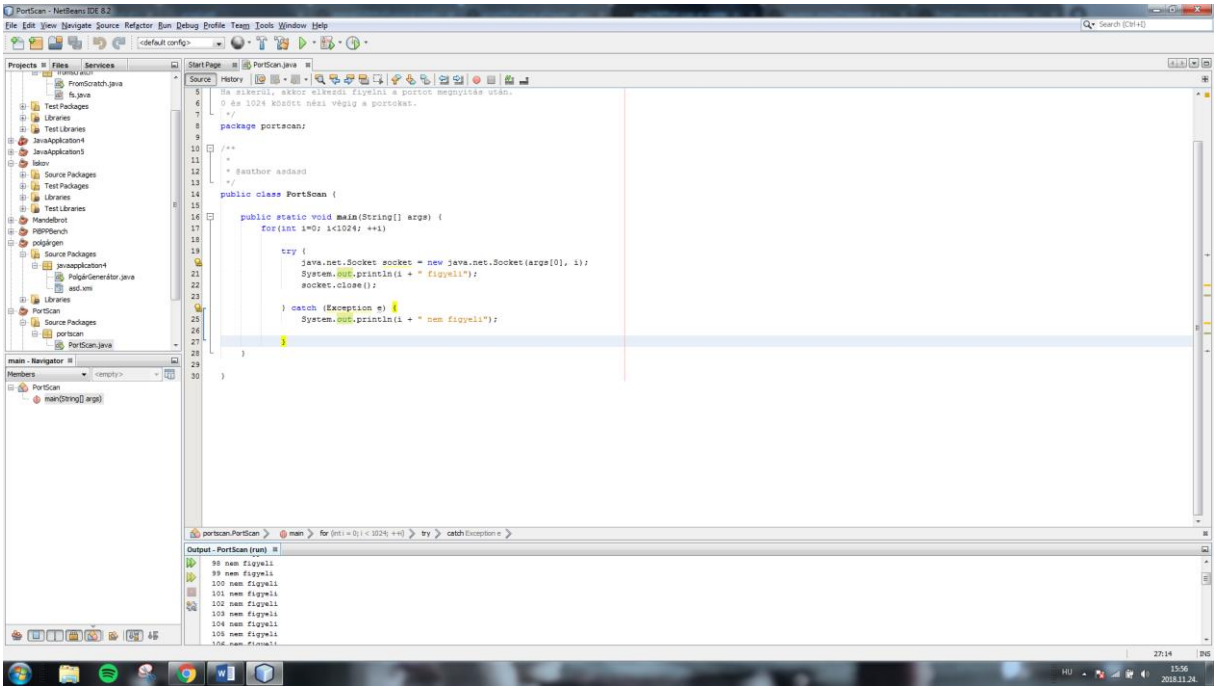
Hogyan használjuk?
A projectben megadott signalokat a "BrainBThread.h" tartalmazza, míg a slotokat a "BrainBWin.h".

A programunk a BrainBWin.cpp forrás osztályának egy példánya, ez felelős többek között a tartalom kirajzolásáért, az egér(állapotok), valamint a kapott signalok feldolgozásáért.
A BrainBWin kontruktorában példányosítunk egy BrainBThread objektumot is, ez felel az négyzetekért.
Továbbá itt kapcsoljuk össze a signalokat a megfelelő slotokkal. (QObject::connect())
A BrainBThread signaljait a BrainBWin slotjaival.

8. hét

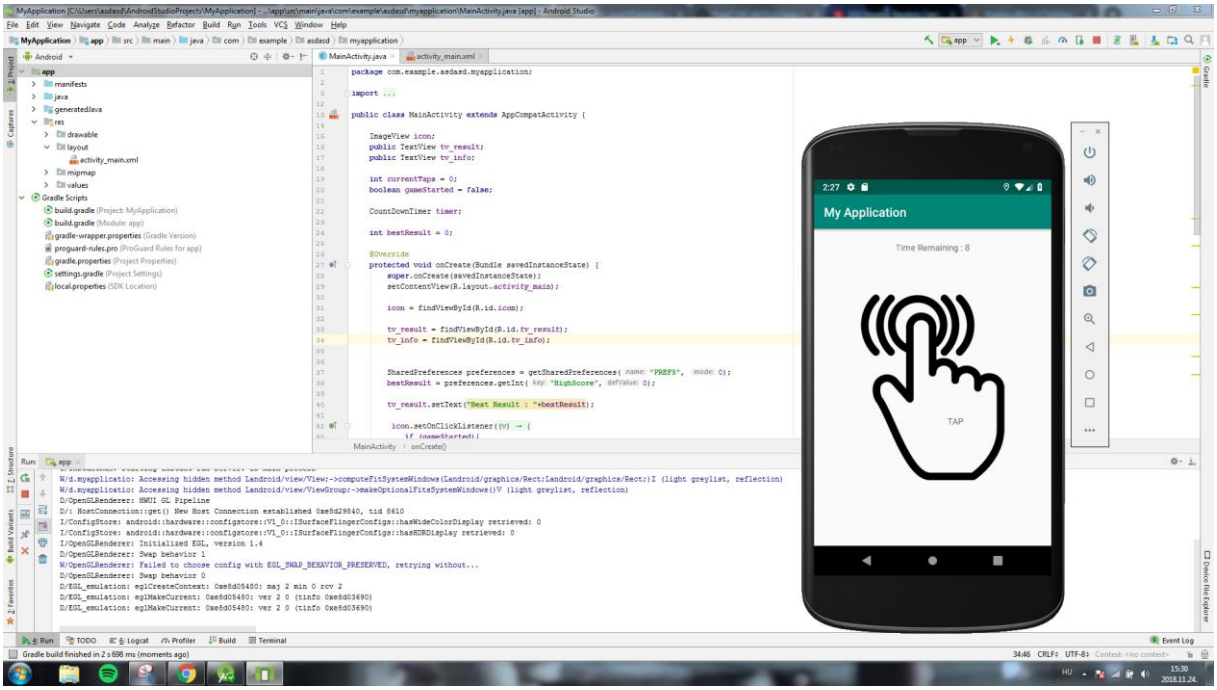
8.1 Port Scan

Try-catch: kivételkezelés.
Try blokk: TCP kapcsolat létesítése java.net Socket függvénnyel, hostname-el és porttal.
Catch blokk: Ha nem sikerül kapcsolatot létesíteni, kivételt ad és "elkap".
Ha sikerül, akkor elkezdi fiyelni a portot megnyitás után.
0 és 1024 között nézi végig a portokat.



8.3 Androidos játék

forrás : https://www.youtube.com/watch?v=Jn1e7Vkd2tk
A forrásban megjelölt videó alapján csináltam ezt az androidra fejlesztett játékot. Egyszerű dolgokra képes, semmi bonyolult.



8.5 Esszé:

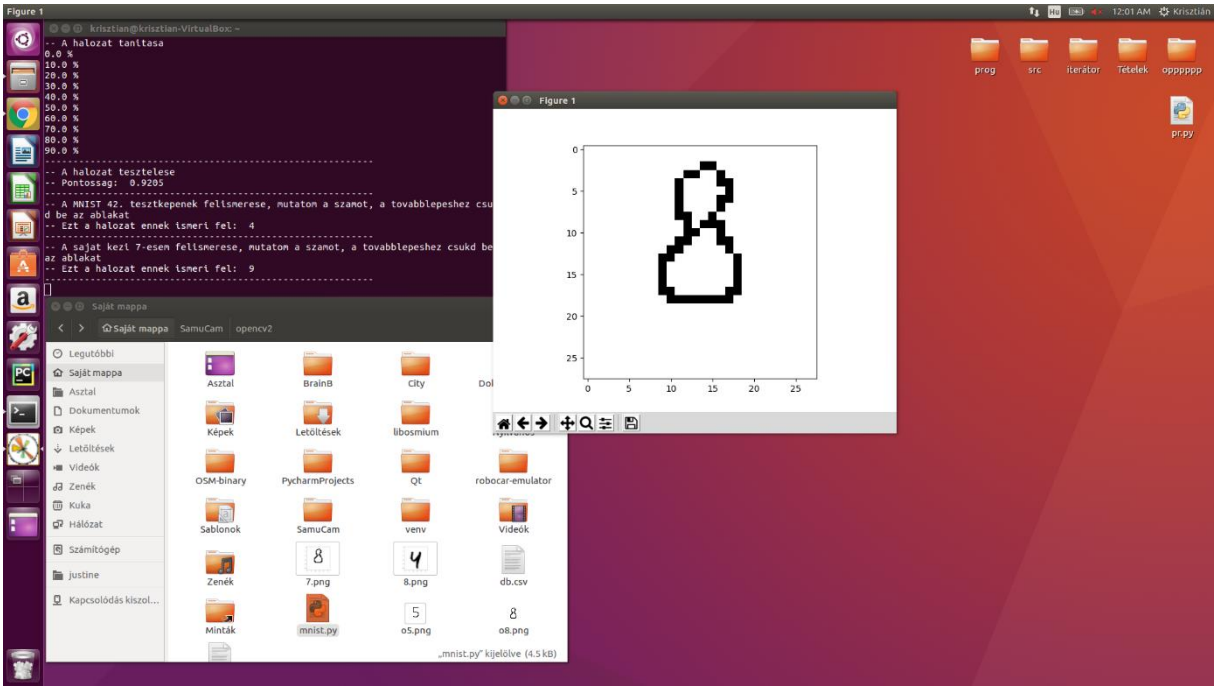
Androidos játék

Az androidos játék Java nyelvben, illetve xml típusú textúrával lett elkészítve. a Játéknak a lényege, hogy az elinduláskor 0 legjobb éréket tároló változóba később a legjobb érték íródik felül.
A képernyő közepén lévő ikon jelzi, hogy hova kell kattintani, habár majdnem az egész kijelzőn érzékeli a tapintást a program.
Minden tapintás egyet ad hozzá a változónkhoz, amiben a jelenlegi értéket tároljuk. ha a Jelenlegi értéket eltároló változóban tárolt szám nagyobb lesz a legjobb eredményt tartalmazó integer típusú, egész számokat eltároló változónkba tárolt értéknél, akkor az lesz az új érték.
Az új rekord megalkotásához, a régi rekord érintés megdöntésére 10 másodpercünk van, ha ez letelik a Program kiírja a legjobb eredményt, vagyis a rekord mennyiséget, illetve a jelenlegiérintések számát. Ezek után, ha új érintés következik be, akkor a Program automatikusan kezdi a számlálást 10 másodperctől és nullázva a jelenlegi értéket kezdi elölről a számolást.
Az android stúdió nevezetű program sokat segített a program elkészítésében, a függvények kiegészítése, már meglévő változók kiegészítése, ajánlása, típusok kiegészítése, javaslása, hasonlók.
A függvényekben lévő kötelezően megadható változók típusait jelezte, könnyed leírást adott ezekről, a függvények használata és lényege teljesen kivehető a program segítségének alkalmazásával.
Az xml résznél van egy olyan mód, amivel az xml fájl változtatása nélkül tudunk irányítani, egér használatának segítségével tudunk gombokat, képeket, objektumokat, szöveget, hasonló apró dolgokat a későbbi megjelenítendő kijelző főbb részeire. Sokban segített maga a program, lehet, hogy későbbiekben használni is fogom kedvtelésből kisebb alkalmazások készítésére is.
A forrásban megjelölt videós elég sok mindenben tud segítséget nyújtani, segítőkész, elég sűrűn rak fel videókat hasonló játékok tervezésével kapcsolatban.

9. hét:

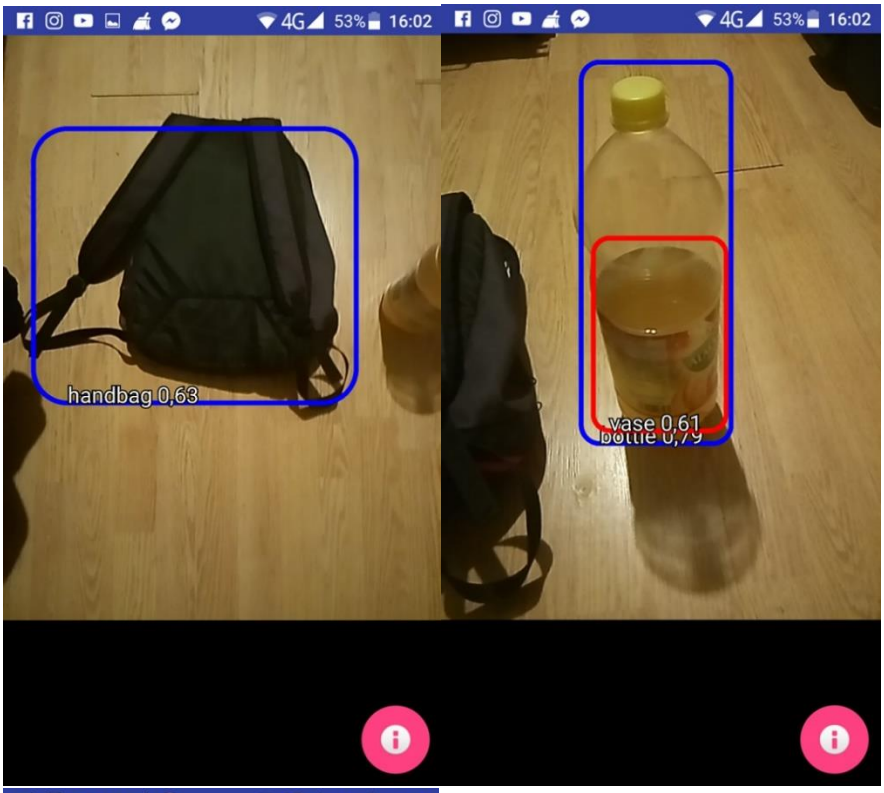
9.1 MNIST

A MNIST program a benne megnevezett képet (ezesetben”  ”) felismeri, és megpróbálja azonosítani a képen ábrázolt számot. Sokszor hibázott, de a program maga lefordul, lefut.



9.4 Tensorflow objektum detektálása:

A Tensorflow által kiadott objektum detektáló programot letöltöttem a telefonomra, kipróbáltam, hogy is működik.
A tárgyak egy részét felismerte, amint a képernyőre kerültek.



9.5 Esszé:

Tensoflow objektum detektálás

A Tensorflow objektum Detektáló program a telefon kameráját használva keres tárgyakat, amik az adatbázisába tettek.
A felismerés valós idejűleg történik, amint a kamera észleli a tárgyakat, egy kis késéssel természetesen.
A google által kifejlesztett alkalmazás a technológiájának és az adatbázisának köszönhetően képes rengeteg tárgyat felsimerni.
A program próbál mindig tökéletes kijelölést, tökéletes azonosítást adni a legrövidebb idő alatt.
Az adatbázisában rengeteg eszköz, tárgy, élőlény található, melyek közé tartozik például az autó, lámpa, ajtó, flakon, váza, ágy, ventillátor, étkezőasztal, lépcső, torta, sütemény, televízió, szépek, távirányító, telefon, egér, billentyűzet, zászló, monitor, és még sok más, ezen felül képes felismerni az embereket, person, személy elnevezéssel illetve őket.
Az alkalmazás rendkívül hasznos a fejlődő világunk számára, ahol a mesterséges intelligenciát próbáljuk létrehozni, ez az alkalmazás nem csak hogy megtudja könnyíteni a tanítás folyamatát, melyeknél a tárgyakhoz neveket kell kötni, hanem egy gyors adatátvitellel fel is tudja gyorsítani ezt a folyamatot.
Habár egyre több és több dolgot ismer fel, a rendszer még korántsem naprakész, rengeteg tárgyat kell még vele megtanítani.
A rossz fényviszonyok, háromdimenziós valóságunkban lévő szintén három dimenziós tárgyakat nehéz minden szögből betanítani, ezért sokszor vétheti el a gép az előtte álló tárgy mivoltát.
Az adatbázis bővítése még nem elég ahhoz, hogy tökéletesen működjön a program, a többféle színű, mintájú, kinézetű dolgot nehéz felfognia még eleinte, és csak meglévő, azonosított tárgyak alapján kalkulál a rendszer, ezek alapján nevezi meg a tárgyat, amit elé rakunk.
Sokszor ez a tárgy nem felismerhető, félreismerhető tárgy a számára, mint ahogy az általam készített képeken is látható az ágyat tortának, előtt fánknak, a hátizsákot először helyesen felismerte, majd kézitáskának titulálta, a flakon félig helyes, másrészről pedig helytelen felismerése is hiba, amikor vázának és flakonnak jelölte be különböző távolságban, különböző méretekkel.
Az alkalmazás próbálja a távolságot is szemügyre venni, figyeli, hogy mettől meddig ér, minden irányba próbálja mérni az adott tárgyat, hogy a legpontosabb mérések által ismerje fel a dolgot precízen. Ez nagyban megnehezíti a dolgát, amikor élőben, mozgatott telefonnal, akár pici rezgések által félremér a kezünk rezgése miatt, ráadásul a nagyobb változtatások még jobban megnehezítik a valós idejű felismerést.
Mindent összevetve maga az előrelépés, a gondolat tetszik, sok fejlesztésre és bővítésre szorul, de kifejezetten tetszik, hogy már most megvalósították a kezdetét ennek a folyamatnak.