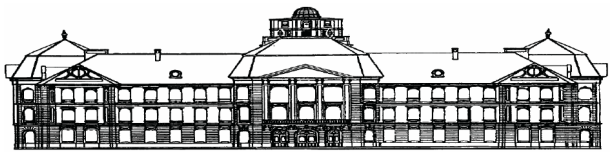

Eszterházy Károly Főiskola
Matematikai és Informatikai Intézet

Mobil eszköz alkalmazási lehetőségei fizikai kísérletekben

Készítette:
Asztalos Gergő
Programtervező informatikus

Témavezető:
Biró Csaba
Adjunktus

Eger, 2016

Tartalomjegyzék

Bevezetés

1. fejezet

Tervezés

Amikor egy projektről beszélünk, számomra az első, és az egyik legfontosabb lépés az, hogy megfelelően megtervezzük a programunkat. Ezen folyamat során fontos megbeszélnünk, hogy milyen lesz a program felépítése, struktúrája, designja. Fontos ezeket még a tervezési fázisban megbeszélni, hisz egy programnál bármit szeretnénk utólag módosítani, sokkal nehezebb lesz a feladatunk, mint az első lépésekben. Célszerű a tervezési fázisban megbeszélteket feljegyezni valamilyen formában. Ilyenkor sokan a rajzoláshoz, íráshoz folyamodnak és ezzel időt és energiát spórolnak maguknak.

Én a tervezési szakaszt hasonlóan kezdtem el. Elsőként felépítettem a számomra megfelelő struktúrát, mind ezt persze papíron, ceruzát használva. Tudtam, hogy nem csak egy alkalmazásom lesz, hiszen főbb céljaim között szerepelt a számítógép és okos telefon közötti Real-Time adatátviteli[1] kapcsolat kialakítása. Hasznos dolognak bizonyult még, a telefon szenzorainak kihasználása és azok alkalmazása a fizikában. Átgondoltam, hogy külön a telefonon és külön a számítógépen lévő programoknak milyen lenne a kinézete, milyen oldalak, ablakok követnék egymást. Elsőként az okos telefonra való fejlesztésnek kezdtem neki, azon belül is az Activity-k és Layout-ok kialakításába, de ez még csak a könnyebb része az egész programnak. Ezután el kellett gondolkoznom azon is, hogy miként fog kommunikálni az a két eszköz? Milyen szenzorral dolgozzunk? Hogyan vigyük át az adatot úgy, hogy megközelítőleg valós idejű legyen?

Természetesen az ilyen kérdésekre a válasz legtöbbször akkor derül ki, amikor már elkezdjük magát a programozást, megválaszolásukra pedig ismét csak papírt és tollat kellett ragadnom. A megfelelő adatokat más, segéd programokkal tudtam csak megjeleníteni, hisz az átlag felhasználók számára ezek a szenzor adatok lényegtelenek.Viszont ezekkel dolgozva, már tudtam készíteni diagramot, amellyel szemléltethettem, milyen értékekről is van szó és azokat hogyan tudnám alkalmazni az én projektemben.

[1] Valós idejű adatfeldolgozás

1.1. Accelerometer

A gyorsulásmérő egy műszer, amely nevéből adódóan gyorsulás mérésére szolgál. A gyorsulást viszont elég nehéz mérni, ezért leginkább a gyorsuláskor fellépő erőt mérjük. Számtalan helyen használják és használhatják: okos telefonokban, digitális fényképezőgépekben, táblagépekben, repülésnél és még sok más helyen. Ezek a szenzorok alkotják a mikromechanikai szenzorok egyik nagy csoportját. A mérési elvek közül a legelterjedtebb a Newton 2. törvénye alapján működő elv, amelynek jellemzője a szeizmikus tömeg:

$$F = m \times a$$

ahol a szenzor az m tömegre ható F erőt méri.

1.1.1. Gyorsulásmérő okostelefonokban

Okostelefonokban a gyorsulásmérő (Accelerometer) arra szolgál, hogy a készülék érzékelhesse a különböző mozgásokat, amikhez így feladatokat tudunk párosítani. A leggyakoribb és legelterjedtebb, az átlagos felhasználók által leginkább használt felhasználási módja az, amikor a telefont elforgatva a képernyő is automatikusan áttájolódik, fekvő és álló tájolási mód között váltakozva. Ez viszont (okostelefonokat tekintetbe véve) a legáltalánosabb felhasználási módszer, ezen kívül még számos esetben használhatjuk a gyorsulásmérőnket, így például gesztusok kezelésében[2], de akár játékok irányításában is.

1.1.2. Gyorsulásmérő alkalmazása a fizikában

Ahogy a példák is mutatják, számos dologra használhatjuk szenzorunkat, miért ne használnánk tehát fizikai kísérletekben segédeszközként? Amikor ilyen kísérletekről beszélünk, természetesen nem olyan kísérletekre gondolunk, ahol a telefon víz vagy tűz állóságát teszteljük, esetleg a szabadesést vizsgáljuk, de például egy inga használatára kiváltképp alkalmas lehet. Kihasználva a telefonra ható gravitációs erőt, illetve a gyorsulást, már is monitorozni tudjuk a telefonunkat, és használhatjuk segédeszközként.

[2] A telefont megrázva valamilyen irányítást eszközlünk

2. fejezet

Szerver alkalmazás

Miután nagyjából megterveztem, hogy milyen is lesz a programom, kialakítottam egy drótváz modellt, és fejben már tudom, hogyan fog működni az applikációm, itt volt az ideje, hogy elkészítsem a szerveroldali alkalmazásomat is.

Első, és talán az egyik legfontosabb feladatom az volt, hogy eldöntsem, milyen nyelven fogom elkészíteni. Több nyelv is bekerült a számomra aktuális nyelvek listájába. Felmerült az is, hogy Python lesz a választott nyelv, hiszen a hasonló matematikai vagy fizikai megoldások felettébb optimalizáltak és a futási ideje sem másodlagos.

A választásom mégis csak a C# nyelvre esett, ennek pedig nem egy oka van. Talán azért is választottam ezt a nyelvet, mert tudtam, hogy az internet adta segítségeket leszámítva is el tudom készíteni az alkalmazásom, hiszen már a főiskolán is ez volt az első nyelv amivel megismerkedtem és ebből adták le a legnagyobb tudásukat az oktatóim is. Külön tantárgy foglalkozott a grafikai rajzolással, megjelenítéssel, s mivel a témámhoz nagyban szükség volt ezekre, így ez csak még egy okot adott arra, hogy ennél a nyelvnél maradjak. Továbbá a fejlesztői környezet nagybani ismerete is megkönnyítette a munkám, hisz nem kellett másikkal megismerkednem. Ez a fejlesztői környezet pedig a Visual Studio volt.

2.1. Visual Studio

A Visual Studio a Microsoft olyan fejlesztői környezete, mely több nyelvet is tartalmaz mint plédául a Visual Basic, C++, C# és ezek mellett még a z XML-t[1] is támogatja, ezek a nyelvek pedig mind IDE[2] fejlesztői környezetet használnak. Különböző Project Template-k közül választhatunk, mint például a Console Application, Windows Form Application, Windows Presentation Foundation. A Visual Studio egy meglehetősen jó fejlesztői környezet, melynek egyszerű és áttekinthető a fájl struktúrája, így

[1] Extensible Markup Language - Kiterjeszthető jelölő nyelv

[2] Integrated Development Environment, azaz integrált fejlesztői környezet

könnyebbé és gyorsabbá teszi a munkát a felhasználója számára.A Project Templateket böngészve sokat tűnődtem azon, hogy melyiket is válasszam, de végül a Windows Form Application-nél maradtam.

Ezt azért választottam, mert a tervezési folyamatnál már tudtam, hogy létre kell hoznom egy TCP/IP kapcsolatot a telefon és a számítógép között, emellett azt is, hogy valamilyen formában rajzolni szeretnék a formomra, habár azt, hogy pontosan mit, akkor még nem tudtam. Ezt a két dolgot már csak tapasztalatból is tudtam, hogy nem nehéz megvalósítani, pláne ha olyan környezetben próbálom, amiben már dolgoztam, esetleg az adott témával kapcsolatban is.

2.2. TCP/IP kapcsolat

A Transmission Control Protocol, azaz TCP, az internet egyik legfőbb részét, az úgymond "gerincét" alkotó TCP/IP protokollcsalád egyik fő protokollja. Ezen protokollcsalád két legfontosabb transzport protokollja a TCP és az UDP[3]. A TCP egy kapcsolatorientált, megbízható protokoll. A kommunikáció megkezdése előtt ki kell építenünk a kapcsolatot, majd ezután megkezdhetük az adatátvitelt. Amennyiben hiba történik, mint például elveszik egy csomag, vagy meghibásodik, esetleg már hibásan érkezik, a TCP saját maga újraindítást kér. Az átvitel úgy történik, hogy az adat, amit szeretnénk átküldeni egy úgynevezett byte-folyam, amit a TCP szétszeparál csomagokra és elküldi. A kapcsolat tehát full-duplex[4], továbbá rendelkezik egy olyan szinkronizációs mechanizmussal, ami megakadályozza, hogy az adó elárassza a vevőt. Emellett a TCP figyeli a kapcsolatot és megpróbálja megtippelni a sávszélességét a tórlódásokból, válaszidőből stb., amit később felhasznál az adatsebesség beállításakor.

Az én programomat nézve az okostelefon minősül az adónak és a számítógép a vevőnek. Értelemszerű, hiszen a szenzor adat a telefonról érkezik és azt kell továbbítanunk a vevőnek. Ez úgy történik, hogy a telefon egy IP cím és egy Port szám segítségével kérést küld a szerver oldal felé a kapcsolódásra, amit a már várakozó szerver később fogad. Ezután meg is kezdődik az adatátvitel. Esetünkben, mivel az egész program a RealTime adatátvitelen alapszik, ezért a kommunikáció nyitott, mindaddig, míg azt "erőszakkal" be nem zárjuk.

2.2.1. Adatok feldolgozása

Az élő kapcsolat fenntartása annyit jelent, hogy a beérkező adatot azonnal feldolgozzuk és a célunknak megfelelően fel is használjuk, adatról-adatra. A hangsúly azon van, hogy megpróbáljuk abban a pillanatban feldolgozni szerver oldalon az adatot, amikor

[3] User Datagram Protocol

[4] Megengedi a kétirányú kapcsolatot, szimultán módon

az előállítódik a kliens oldalon. Természetesen száz százalékosan ez nem teljesíthető, hiszen ez a folyamat elég erőforrás igényes.

A kapcsolat beérkezése után meg is kezdődik az adatok feldolgozása. Ez a feldolgozási folyamat természetesen attól függ, mit szeretnénk csinálni az adatunkkal.

```
int port = 8888;
label5.Invoke(new Action(() => label5.Text =
   GetLocalIPAddress()));
TcpListener myListener = new
   TcpListener(IPAddress.Parse(GetLocalIPAddress()), port);
myListener.Start();
label1.Invoke(new Action(() => label1.Text = "Varakozas
   kliensekre a " + port + " porton."));
TcpClient connectedClient = myListener.AcceptTcpClient();
label1.Invoke(new Action(() => label1.Text = "Kliens
   csatlakozott"));
```

A csatlakozást követően szükségünk van egy StreamReader-re, ahhoz, hogy a kapcsolódott eszköz által küldött információt meg is tudjuk jeleníteni, vizsgálni. Az adatok, azaz a 3 tengely értéke 1 adott sorként érkezik át a kapcsolat folyamán. Ha ezekkel az adatokkal dolgozni is szeretnénk (márpedig szeretnénk), szükségünk lesz arra, hogy azt az 1 sort, szétbontsuk 3, különálló értékre. Így megkapjuk az x, y és z tengelyek értékét.

$$-0{,}10534488 \qquad 9{,}787497 \qquad -1{,}091756 \qquad Time : 0$$

A szétbontásuk pedig a következőképp történik:

```
string s = r.ReadLine();
string[] str_array = s.Split(' ');
x = double.Parse(str_array[0],
   CultureInfo.InvariantCulture);
y = double.Parse(str_array[1],
   CultureInfo.InvariantCulture);
z = double.Parse(str_array[2],
   CultureInfo.InvariantCulture);
```

Ebben az esetben a(z) s változónk tárolja az átküldött sort, a(z) str_array pedig annak a szétbontott változatát. Így meg is kaptuk a szövegként átküldött sorunkat, változókra szétbontva és megkezdhetjük a feldolgozásukat, beillesztésüket a feladatunkba.

2.3. Gyorsulási diagram

Most már megvannak az adataink, fel is tudjuk hát dolgozni őket. Első és legfontosabb dolgunk az, hogy létrehozzunk egy diagramot, ami segítségével meg tudjuk jeleníteni (tengelyenként) a telefonunk gyorsulását. Ez már önmagában véve is egy olyan szemléltetés, melyet használhatnánk a fizika tanórákon. Szerencsére van erre egy beépített Tool[5], melynek értékeket átadva már el is készíti a megfelelő diagramot. Ezen Tool neve: Chart. Dolgunk nehezedik annyiban, hogy ennek a diagramnak előre definiálnunk kell bizonyos pontokat a megjelenítéshez, viszont bármikor tudunk hozzáadni új pontokat. Ezzel csak annyi a probléma, hogy az új pontokat beszúrja, a többi pont mellé, így viszont bizonyos idő után feltorlódnak az adatok és a vonaldiagramunk elveszti varázsát.

Erre beépített megoldásunk sajnos nincs, magunknak kell megírnunk a függvényünket úgy, hogy a bent lévő adatok egy része megmaradjon, viszont a régi adatokat kitöröljük. Erre egy olyan megoldást eszközöltem, hogy amennyiben a diagramunk pontjainak száma elér egy előre megadott mennyiséget, a pontokat elmentjük egy listába, a listát megfelezzük, az első részét töröljük a diagram összes elemével együtt, majd a lista második része lesz a diagramunkra ráillesztve. Ezzel azt érjük el, hogy a diagram törlődik, az elemek fele elvész, a másik fele pedig újból beíródik, így a diagramunknál nem lesz torlódás, mégis folyamatosnak látszik a megjelenítés.

[5] Visual Studioban a felhelyezhető gombokat, mezőket egy ToolBoxban találjuk

3. fejezet

Kliens alkalmazás

3.1. Android Studio

4. fejezet

Használati útmutató

4.1. Ezvalami

Irodalomjegyzék

[1] Szerző: Cím, Kiadó, Hely, évszám.