Haladó Programozás Beadandó
Szenzorszimuláció
„Áradás Észlelő Applikáció”

A projekt célja:

Szenzorhálózat szimulációjának megvalósítása C# konzolalkalmazás formájában. A választott téma egy **vízszintmérő rendszer**, amely egy nem létező folyó vízszintjét monitorozza a hét minden napján 5 darab mérőpontos.

A program 5 db különálló szenzor-csomópontot szimulál, melyekhez véletlenszerűen generálnak mérési adatokat 0 és 500 cm között. A rendszer feladata a mért adatok összegyűjtése, kiértékelése (átlagolás, kritikus értékek figyelése), valamint riasztások küldése áradás, vagy annak esélye esetén. A mérési eredményeket a program program mindig eltárolja, JSON-ban és LiteDB-ben is!

A program működése:

A program elindítása után automatikusan szimulál egy teljes hetet (hétfőtől vasárnapig). Minden napra generál 5 darab szenzoradatot. A program beépített logikája alapján a héten véletlenszerűen kiválaszt egy napot, amikor "katasztrófa eseményt" (áradást) szimulál, ekkor a mért értékek garantáltan magasabbak a kritikus szintnél.

Bemenet

A bemenetet (vízszint cm-ben) a rendszer véletlenszám-generátorral állítja elő a 0 - 500 cm tartományban szimulálva a szenzorok méréseit. Nincsen kézzel bevitt adat.

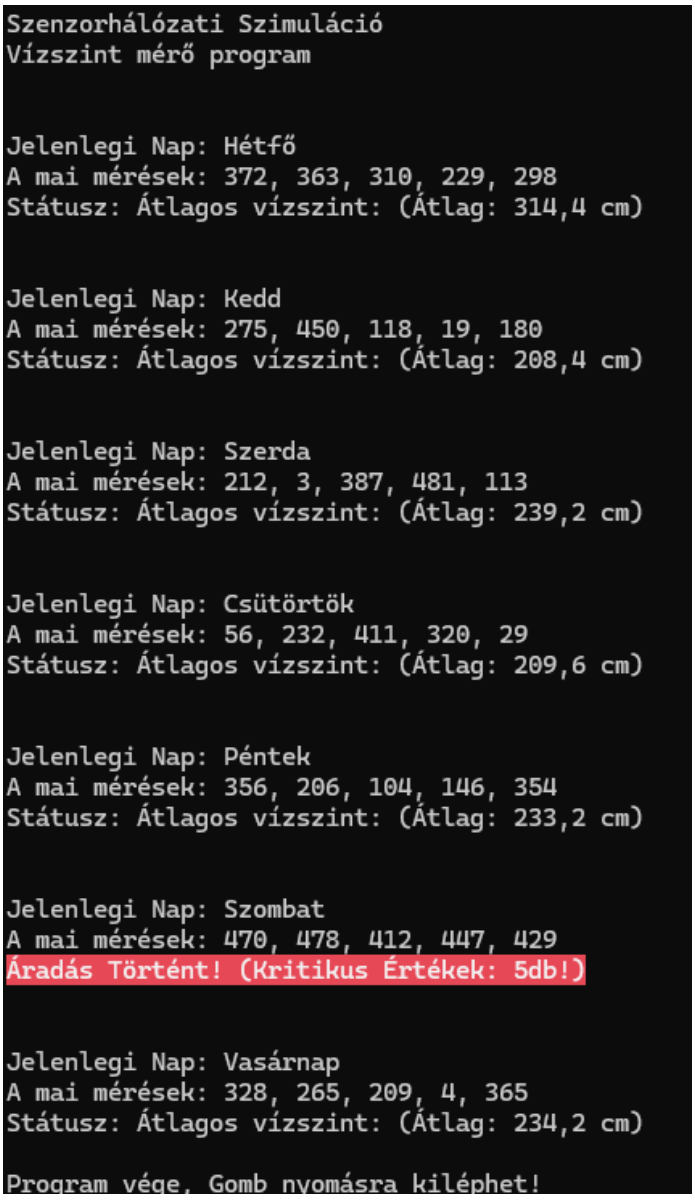
Kimenet

A kimenetet Konzol képernyőben valósítottuk meg, melyben kiíratjuk Napi státuszüzenetek (Nap neve, mért értékek felsorolása). Továbbá tartalmaz státusz jelentéskét:

Normál*:* "Átlagos vízszint" + átlagérték.

Figyelmeztetés*:* Sárga háttérrel "Magas Vízszint Veszélye". Pirossal ha az áradás protokol lép életbe

Riasztás: Piros háttérrel "Áradás Történt!" (Eseménykezelés által kiváltva).



Fájl kimenetek

A kimeneteket több fájl típusba is exportáljuk ilyenek a meresek.json fájl, vagy a adatbazis.txt fájl illetve az adatok,db adatbázis fájl.

meresek.json-ban a heti adatok strukturált mentését tároljuk.(JSON formátumban)

adatbazis.txt.-ban szöveges, pontosvesszővel tagolt fájlt hoztunk létre az adatokkal.

adatok.db: LiteDB adatbázis fájlba a strukturált tároláshoz.

```
    static void mentes(List<NapiMeres> adatok)
    {
        //JSON fájlba mentés
        string jsonString = System.Text.Json.JsonSerializer.Serialize(adatok, new JsonSerializerOptions { WriteIndented = true });
        File.WriteAllText("meresek.json", jsonString);

        // elmentjük a méréseket szöveges fájlban
        using (StreamWriter sw = new StreamWriter("adatbazis.txt"))
        {
            sw.WriteLine("Nap;SzenzorID;Vizszint;Statusz");
            foreach (var napAdat in adatok)
            {
                foreach (var meres in napAdat.Meresek)
                {
                    sw.WriteLine($"{napAdat.Nap};{meres.SzenzorID};{meres.VizszintCm};{napAdat.Statusz}");
                }
            }
        }

        //elmentjük adatbázisban
        var db = new LiteDatabase("Filename=adatok.db; Connection=shared;");
        var meresek = db.GetCollection<NapiMeres>();
        foreach (var meres in adatok)
        {
            meresek.Insert(meres);
        }
    }
}
```

Program főbb részei

DLL: Főbb tulajdonságai

A DLL fő funkciója a mérési adatok szimulálása, amelyet a SzenzorMeresGeneralas osztály Mérés generálás metódusa végez:

- **Bemenet:** Egy bool *GarantaltAradas* paramétert vár. Ha ez true (igaz), akkor kényszerített áradást szimulál, ha false, akkor normál működést.
- **Kimenet:** Egy 5 elemű List<MeresAdat> listával tér vissza, amely a generált mérési eredményeket tartalmazza.
- **Generálási szabályok:**
    - *Normál esetben:* 0 és 500 cm közötti véletlenszerű értéket generál.
    - *Garantált áradás esetén:* Az első 3 szenzor értékét 401 és 500 cm közé állítja (vészhelyzeti érték), biztosítva ezzel a kritikus szint elérését.

- **Listák (List<T>):** Ez a leggyakrabban használt adatszerkezet a kódban, mivel a mérések száma és a heti adatok mennyisége dinamikusan változhat (bár itt a mérések száma fix 5, a lista rugalmasságot biztosít).
    - List<MeresAdat> aktualisMeresek: Az adott napon generált 5 szenzor mérési eredményét tárolja.

    - List<NapiMeres> hetiAdatok: A teljes hét adatait (nap neve, mérések listája, státusz) gyűjti össze egyetlen listába a fájlba mentéshez.

- **Tömb (Array):**
    - string[] napok: A hét napjainak neveit tárolja. Mivel ez egy fix, előre ismert adathalmaz, a tömb a legegyszerűbb választás.
- **Saját osztály (NapiMeres):**
    - Ez egy összetett adatszerkezet (adatmodell), amely összefogja egy naphoz tartozó összes információt: a nap nevét (string), a mérések listáját (List<MeresAdat>) és a napi státuszt (string).

2. Technológiák és Könyvtárak

A főprogram számos modern C# technológiát és külső könyvtárat integrál:

- **LINQ (Language Integrated Query):** A specifikációnak megfelelően a program LINQ lekérdezéseket használ az adatok tömör és olvasható feldolgozására:
    - .Count(x => x.VizszintCm > 400): A kritikus (400 cm feletti) értékek megszámlálása.
    - .Average(x => x.VizszintCm): A napi átlagos vízszint kiszámítása.
    - .Select(x => x.VizszintCm): Csak a vízszint értékek kiválasztása a megjelenítéshez (projekció).
- **LiteDB (NoSQL Adatbázis):** A LiteDB könyvtárat használja a program egyszerű, fájl alapú (embedded) adatbázis kezelésre. Ez lehetővé teszi az objektumok (NapiMeres) közvetlen mentését relációs táblák létrehozása nélkül.
- **JSON Szerializáció (System.Text.Json):** Az adatok hordozható formátumban való mentéséhez a JsonSerializer-t használja, amely a hetiAdatok listát alakítja át formázott (indentált) szöveges JSON kimenetté.
- **Eseménykezelés (Event Handling):** A program a C# eseménykezelési mechanizmusát (delegate, event) használja a kritikus áradások jelzésére. Az OnAradas esemény akkor váltódik ki, ha 3 vagy több szenzor mér kritikus értéket, ezzel elválasztva a logikát a megjelenítéstől (FigyelmeztetesKiiro).
- **Fájlkezelés (System.IO):** A StreamWriter osztályt használja a hagyományos, pontosvesszővel tagolt szöveges fájl (adatbazis.txt) írására.