**UNIVERZITA J. SELYEHO – SELYE JÁNOS EGYETEM**

**FAKULTA EKONÓMIE A INFORMATIKY**

**GAZDASÁGTUDOMÁNYI ÉS INFORMATIKAI KAR**

**SIMULÁCIA EPIDÉMIE NA GRAFOCH  
EPIDEMIC SIMULATION ON GRAPHS  
JÁRVÁNYSZIMULÁCIÓ GRÁFOKON**

**Diplomovská práca – Diplomamunka**

**Kristian Salma   
2021**

**UNIVERZITA J. SELYEHO – SELYE JÁNOS EGYETEM**

**FAKULTA EKONÓMIE A INFORMATIKY**

**GAZDASÁGTUDOMÁNYI ÉS INFORMATIKAI KAR**

**SIMULÁCIA EPIDÉMIE NA GRAFOCH  
 JÁRVÁNYSZIMULÁCIÓ GRÁFOKON**

**Diplomovská práca – Diplomamunka**

Študijný program: Aplikovaná informatika  
Tanulmányi program: Alkalmazott informatika  
Názov študijného odboru: Informatika  
Tanulmányi szak megnevezése: Informatika  
Védúci záverečnej práce: Dr. habil. Attila Elemér Kiss, CSc.  
Témavezető: Dr. habil. Attila Elemér Kiss, CSc.  
Školiace pracovisko: Katedra matematiky a informatiky  
Tanszék elnevezése: Matematika és informatika tanszék

**Kristian Salma Komárno**

**2023**



Univerzita J. Selyeho

Fakulta ekonómie a informatiky

**ZADANIE ZÁVEREČNEJ PRÁCE**

|  |  |  |
| --- | --- | --- |
| **Meno a priezvisko študenta:** | | Kristián Salma |
| **Študijný program:** | | Aplikovaná informatika (Jednoodborové štúdium, bakalársky I. st., denná forma) |
| **Študijný odbor:** | | 18. - informatika |
| **Typ záverečnej práce:** | | Bakalárska práca |
| **Jazyk záverečnej práce:** | | maďarský |
| **Sekundárny jazyk:** | | slovenský |
| **Téma:** | Automatikus Arduinoval és weboldallal vezérelt ajtó | |
| **Anotácia:** | A szerző feladata egy Arduinoval vezérelt automata tyúkólajtó létrehozása, amelynek beállításait egy jelszóval védett weboldalon lehessen megadni. Az ajtó nyitására és zárására különböző módok közül választhasson a felhasználó: előre megadott nyitási és zárási időpont, fényérzékelővel vezérelt nyitás és zárás, vagy nyitás és zárás napkelte és napnyugta alapján (ehhez meg lehessen adni a tyúkól GPS koordinátáit, ami alapján interneten elérhető ingyenes API alapján határozza meg a napkelte és a napnyugta időpontját, pl. https://sunrisesunset.org/api segítségével). | |
| **Vedúci:** | Dr. habil. Attila Elemér Kiss, CSc. | |
| **Katedra:** | KINF - Katedra informatiky | |

**Vedúci katedry:** Ing. Ondrej Takáč, PhD.

**Dátum zadania:** 05.10.2020

**Dátum schválenia:** 21.10.2020 Ing. Ondrej Takáč, PhD.

vedúci katedry

**Abstrakt**

Cieľom mojej bakalárskej práce je vytvorenie systému automatizovaného otvárania dvier, ktorý bude ľahko prevádzkovateľný a maximálne flexibilný. Tento systém umožní používateľom ovládanie jedných, prípadne viacerých dverí naraz v tom istom čase. Prepojenie webovej stránky s databázou a mikropočítačom Arduino nám umožňuje nastavovať a ovládať mechanizmus otvárania a zatvárania dverí. Mechanizmus je riadený mikropočítačom Arduino, ktorý komunikuje s webovým serverom prostredníctvom Wi-Fi modulu. Systém poskytuje tri možnosti nastavenia automatizovaného ovládania dvier. Prvé nastavenie ovláda dvere podľa času východu a západu slnka. Systém metódou API call zistí z internetu podľa GPS súradníc presný čas východu a západu slnka v danej lokalite a v uvedenom čase otvorí, resp. zatvorí dvere. Druhé nastavenie umožňuje otváranie, resp. zatváranie dvier v presnom čase, ktorý je možné zadať v používateľskom rozhraní. Posledné, tretie nastavenie spočíva vo využití lokálne inštalovaného svetelného senzora, ktorý nepotrebuje pripojenie k internetu. Otvorenie, resp. zatvorenie dverí je riadené na základe dosiahnutia hraničných hodnôt svetelného toku zaznamenávaného svetelným senzorom – fotobunkou. V používateľskom rozhraní, prostredníctvom webovej stránky sa musí používateľ, po predchádzajúcej registrácii, prihlásiť. Po prihlásení sa používateľovi zobrazí administratívný panel, ktorý umožňuje ovládať a kontrolovať všetky nastavenia. Nakoľko všetky vykonané operácie sa zaznamenávajú v databáze, je možné spätne kontrolovať vykonané zmeny, podla čoho sa dajú analyzovať problémy. Vo svojej práci chcem poukázať na široké možnosti využitia uložených dát, pomocou ktorých je možné zdokonaliť prevádzku, eliminovať výskyt chýb a získať viac informácií o prevádzke systému.

Kľúčové slová : Arduino, PHP, databáza

**Absztrakt**

A szakdolgozat célja egy automatizált ajtónyitó rendszer kivitelezése, mely működtetése mindamallett, hogy könnyen kivitelezhető, maximálisan flexibilis is. A projekt segítségével az ügyfelek szabályozni tudják egy, vagy több ajtó működését, ugyanabban időben.   
A weboldal segítségével kommunikálunk az adatbázissal és az Arduino mikro számítógéppel. Így lehetőségünkben áll szabályozni és mozgatni az ajtónak a mechanizmusát a weboldalon keresztük, különféle beállítások segítségével.

A projekt egy „Arduino” mikro számítógép segítségével, egy RTC, fényérzékelő és Wi-Fi modul segítségével kommunikál a fizikai (ajtó)nyitó rendszerrel. A szoftver három beállítás alapján tudja szabályozni az ajtó nyitását. Az első egy „API” lekérdezéssel működik, ami az internetről GPS adatok alapján lekért napkelte és napnyugta idejéhez időzíti az ajtók működését. A második lehetséges beállítás egy konkrét idő megadásával történhet, amit a weboldalunkon tudunk beállítani. Végezetül a fényérzékelővel ellátott ajtónyitás, amelynek nincs szüksége internetre - vagy szerver oldali kommunikációra. A weboldalon lehetőségünk van bejelentkezni, regisztrálni, rálátást kapunk az Arduino adminisztrációs panelra, ahonnan a vezérlők irányítása történik és ellenőrizhetjük, hogy megfelelően működnek-e a vezérlők. Az elvégzett műveleteket egy adatbázisban tároljuk, így probléma eseten bármikor visszanézhető mikor történt probléma. Dolgozatomban kitérek még a lehetőségek tárházára, amivel tökéletesíteni tudnánk a vezérlők működését, amivel több információt és kevesebb hibalehetőséget kaphatunk.

Kulcsszavak: Arduino, PHP, adatbázis,

**Abstract**

My thesis is going to cover an automatized, software-based door-opener and closure system, which is aiming to be both user friendly and flexible at the same time. The projects’ base is an Arduino microcomputer that is communicating through a Wi-Fi module with a website where it's possible to set up the door closure system. This “base” system is also extendable with an RTC and a light sensor. The software has three possible setups for the opening and closing . The first one is working with an API call, which automatically adjusts the opening and closing to the sunrise and sunset based on our GPS coordinates. The second setup is when we manually set the time of opening and closing through our dedicated website. The third one uses a light sensor which controls the opening and closing times and requires no internet at all.

On the website – after the registration and log in process - we access the Arduino admin panel, where we can adjust and monitor the controller’s behaviour and check if there are any issues with the system. All commands are stored in a database, in case of a failure everything can be trace back. I also mention in my thesis how could we improve this system to be widely used and make it more efficient and reliable.

Keywords: Arduino, PHP, database,

Tartalomjegyzék

[1 Kiírás - Járványszimuláció gráfokon 10](#_Toc125882664)

[2 Elmélet 11](#_Toc125882665)

[2.1 Projekt kezdése 11](#_Toc125882666)

[2.2 Vírus elmélet 11](#_Toc125882667)

[2.3 Környezet 11](#_Toc125882668)

[2.3.1 Programozási nyelv c# 12](#_Toc125882669)

[2.3.2 .Net 6 framework 12](#_Toc125882670)

[2.4 Felhasznált eszközök 12](#_Toc125882671)

[2.4.1 Visual Studio 2022 12](#_Toc125882672)

[2.4.2 SQL adatbázis 13](#_Toc125882673)

[2.4.3 SQL server management studio 13](#_Toc125882674)

[2.4.4 Git 13](#_Toc125882675)

[2.5 Gráfok 13](#_Toc125882676)

[2.5.1 }Irányítatlan gráf 15](#_Toc125882677)

[2.5.2 Irányított gráf 15](#_Toc125882678)

[2.5.3 Mélységi bejárás (DFS -Depth-first search) 15](#_Toc125882679)

[2.5.4 Szélességi bejárás(BFS - Breadth-First Search) 17](#_Toc125882680)

[3 Járványszimuláció 19](#_Toc125882681)

[3.1 Virus terjedésének lehetőségei és rövid megnevezései 19](#_Toc125882682)

[3.1.1 Világjárvány 19](#_Toc125882683)

[4 Szlovákia adatok 21](#_Toc125882684)

[5 Gyakorlat 22](#_Toc125882685)

[6 Felhasznált irodalom 23](#_Toc125882686)

Ábrajegyzék

**No table of figures entries found.**

# Kiírás - Járványszimuláció gráfokon

Lehessen generálni, vagy beolvasni nagy irányítatlan gráfot. A gráf csúcsaiba véletlenszerűen elhelyezünk egészséges és fertőzött egyéneket. Az iteráció áll egy fertőző lépésből és egy elmozduló lépésből. Fertőző lépés: Ha egy csúcsban legalább egy fertőzött egyén van, akkor valamilyen fertőzési modell alapján az egy helyen lévő egyének megfertőződnek. Ha nem volt még megfertőzve, akkor p valószínűséggel, ha eddig k-szor volt megfertőzve, akkor p/k valószínűséggel. A fertőzöttek T iteráció után h valószínűséggel meghalnak, (kiesnek a rendszerből), 1-h valószínűséggel életben maradnak és gyógyultak. Elmozduló lépés: Minden egyén a gráfban szomszédos csúcsok valamelyikére lép 1/n valószínűséggel, ha n szomszédos csúcs van. A szimulációt különböző típusú gráfokra kell, különböző paraméterekre többször lefuttatni nagy iterációszámra. Táblázatokat, grafikonokat kell készíteni, amely a fertőzés arányait, tulajdonságait mutatja. A megvalósítás tetszőleges nyelven készülhet, az eredmény weben kell megjeleníteni.

Môže generovať alebo nahrat veľké neorientované grafy. Do vrcholov grafu náhodne umiestňujeme zdravé a infikované osoby. Iterácia pozostáva z kroku nakazenia a kroku premiestnenia. Infekčný krok: ak sa vo vrchole nachádza aspoň jeden infikovaný jedinec, potom podľa určitého infekčného modelu budú infikovaní aj jedinci na rovnakom mieste. Ak predtým nebol infikovaný, je infikovaný s pravdepodobnosťou p, ak bol predtým infikovaný k-krát, je infikovaný s pravdepodobnosťou p/k. Po T iteráciách infikovaní jedinci zomrú (odpadnú) s pravdepodobnosťou h, prežijú a vyliečia sa s pravdepodobnosťou 1-h. Krok presunu: každý jedinec sa presunie do jedného zo susedných vrcholov grafu s pravdepodobnosťou 1/n, ak existuje n susedných vrcholov. Simulácia sa musí vykonať niekoľkokrát pre rôzne typy grafov s rôznymi parametrami pre veľký počet iterácií. Mali by sa vytvoriť tabuľky a grafy zobrazujúce mieru a vlastnosti infekcie. Implementácia môže byť v ľubovoľnom jazyku, výsledok by sa mal zobraziť na webe.

# Bevezetés

Napjainkban elég sok problémával kell az emberiségnek szembenéznie. Ezeket a problémákat csoportosíthatjuk emberek által létrejött problémáknak, amik lehetnek a pénzügyi válságok, háborúk. A másik nagyobb csoport pedig a természet által létrejött problémák, amik lehetnek árvizek, vulkán kitörések, tűzvész, tornádó, hegyomlás, földrengés. A mi diplomamunkánknak is a támája az egyik ilyen természeti problémának/katasztrófának a kutatása és szimulálása lesz. Az emberiség már a kezdetek óta küzd a vírusok elleni harcában. Vissza tekintve a múltba több katasztrófát is fel tudunk sorolni, amik vírusos fertőződéshez köthetőek. A mi diploma munkánkban is különféle vírusok szimulációját fogjuk tudni lefuttatni, különféle gyorsasági sebességgel és mindenféle fertőződési modellek alapján kialakított szimulált helyzetekben. A szimuláció során számításba fogunk venni nagyon sok környezeti befolyást is amik lehetnek az immunrendszer gyengesége, az emberek találkozása, gyógyulási modellje. A szimulációban kialakítottunk különféle környezeteket, ahol az emberek kapcsolatba kerülhetnek egymással. Ezek a környezetek lehetnek: otthonok, munkahelyek, üzletek, korházak, iskolák. Ezeken a helyeken megszabjuk, hogy melyik ember melyik lakásban lakik kik lesznek a családtagjai vagy esetleg egyedülálló is lehet, ha a rendszer úgy generálja le nekünk egy random funkció segítségével. A másik fontos tényezője az embernek az életkora lesz az életkor szerint lesz eldöntve, hogy az adott személy iskolába, munkába fog-e járni. Az életkorhoz még egy nagyon fontos paraméter is fog járulni az pedig az immunrendszernek az erősége. Ugyebár tudjuk, hogy ahogy az emberek öregednek kissé mindig veszítenek az immunrendszerük erősségéből és már jobban kiszolgáltatottabbak lesznek erősebb vírusokkal szemben. A szimuláció során beállíthatjuk, valamennyi számú egyénnek azt, hogy a szimuláció során véglegesen immunis lesz a vírussal szemben. Az egyének a szimuláció megkezdésével 2 féle lépést fognak elvégezni az egyik lépés a fertőződés lesz amikor az adott személyek egy helyen tartózkodnak egy már megfertőződött személlyel és valamilyen szimulációs modell alapján azt ő tovább tudja kapni vagy sem. A második lépés pedig az elmozdulás lesz, amikor az egészséges és fertőzött személyek egyaránt elmozdulnak egy másik helyre, ami a fentebb említett helyek lehetnek. A helyeket, ahol az emberek tartózkodnak kitudjuk magunk is alakitani, vagy betudjuk tölteni, vagy véletlenszerűen a programra bízhatjuk a legenerálását. Az egyik nem említett fő téma, ami a szimulációnkhoz kapcsolódik az pedig a gráfok lesznek. A helyek gráfokhoz hasonlóan fognak kapcsolódni egymáshoz. Ezeket innentől kezdve csúcsoknak fogjuk nevezni. A csúcsaink között az emberek különféle közlekedési járműveket használhatnak, melyeknek külön meglesz szabva vagy akár személyre lehet szabni a fertőzés súlyosságát. A szimulációban az emberek természetesen meg is gyógyulnak, de sajnos meg is halhatnak, amely személy meghalt az végleg kiesik a rendszerből. A szimuláció során az adatokat felhasználjuk és adatbázisba mentjük, ahonnan később pedig weboldalon fogjuk tudni megjeleníteni a grafikonokat a fertőzési szimulációnak a segítségével.

# Felhasznált programok

## Visual Studio 2022

A Visual Studio a Microsoft által kifejlesztett integrált fejlesztőkörnyezet (IDE), amelyet először 1997-ben adtak ki. Azóta a környezet több jelentős frissítésen és átdolgozáson esett át. A 2019-ben megjelent legújabb verzió számos új funkciót és fejlesztést tartalmaz a korábbi verziókhoz képest.

A fejlesztők körében a legnépszerűbb eszközök közé tartozik a szoftveralkalmazások készítéséhez, teszteléséhez és telepítéséhez. A termék mindent tartalmaz, amire egy fejlesztőnek szüksége van, például kódszerkesztőt, hibakereső programot, fordítóprogramot és számos más eszközt és szolgáltatást a szoftverfejlesztési folyamat egyszerűsítéséhez.

A termék emellett számos programozási nyelvet támogat, köztük a C#, a C++ és a Python nyelveket, így a fejlesztők kiválaszthatják a projektjükhöz legmegfelelőbb nyelvet, miközben megkapják a munkájukhoz szükséges eszközöket. A termék sablonok és kódrészletek széles skáláját támogatja, amelyek felgyorsíthatják az új projektek létrehozását, és felgyorsíthatják a fejlesztési folyamatot, hogy megkönnyítsenek bizonyos tevékenységeket a fejlesztők számára.[7]

Nugget package

Bookmarks

Extensions

Msbuild

Debug

Release

Solution explorer

### Nugget csomagok

Minden modern fejlesztési platform alapvető eszköze egy olyan mechanizmus, amellyel a fejlesztők hasznos kódot hozhatnak létre, oszthatnak meg és fogyaszthatnak. Gyakran az ilyen kódot "csomagokba" csomagolják, amelyek lefordított kódot (DLL-ként) tartalmaznak az ezeket a csomagokat használó projektekben szükséges egyéb tartalmakkal együtt.

A .NET (beleértve a .NET Core-t is) esetében a Microsoft által támogatott kódmegosztási mechanizmus a NuGet, amely meghatározza, hogy a .NET csomagok hogyan hozhatók létre, hogyan tárolhatók és hogyan fogyaszthatók, és biztosítja az eszközöket az egyes szerepkörökhöz.

Egyszerűen fogalmazva, a NuGet csomag egy .nupkg kiterjesztésű ZIP-fájl, amely lefordított kódot (DLL-eket), a kódhoz kapcsolódó egyéb fájlokat és egy leíró manifesztet tartalmaz, amely olyan információkat tartalmaz, mint például a csomag verziószáma. A kódot terjeszteni kívánó fejlesztők csomagokat hoznak létre, és közzéteszik azokat egy nyilvános vagy privát tárhelyen. A csomagfelhasználók ezeket a csomagokat a megfelelő tárhelyekről szerzik be, hozzáadják a projektjeikhez, majd a projektkódjukban meghívják a csomag funkcióit. A NuGet maga gondoskodik a közbeeső részletekről.

Mivel a NuGet a nyilvános nuget.org tárhely mellett támogatja a privát tárhelyeket is, a NuGet csomagok segítségével olyan kódot is kiadhat, amely kizárólag egy szervezet vagy munkacsoport számára elérhető. A NuGet csomagokat kényelmes módszerként is használhatja saját kódjainak megosztására, kizárólag saját projektjei számára. Röviden, a NuGet-csomag a kód egy megosztható egysége, de nem követeli meg vagy nem feltételezi a megosztás egy bizonyos típusát.

Felhasznált Nugget csomagok a projektben

#### Avalonia

#### Avalonia Diagnostic

#### Avalonia ReactiveuI

### Bővtmények

#### Avalonia Toolkit

#### Visual Studio spell checker

#### Resharper

#### SQL server data tools

### Használt funkciók

#### Könyvjelzők

#### Solution explorer

#### Branch kezelő

#### Tabs

## Visual Studio Code 2019

## SQL szerver

## Git

## Avalonia keretrendszer

### Keretrenszer

Általánosságban a keretrendszer egy olyan valós vagy fogalmi struktúra, amely támaszként vagy útmutatóként szolgál valaminek az építéséhez, amely a struktúrát valami hasznos dologgá bővíti. A számítógépes rendszerekben a keretrendszer gyakran egy réteges struktúra, amely jelzi, hogy milyen programokat lehet vagy kell építeni, és azok hogyan kapcsolódnak egymáshoz. Néhány számítógépes rendszer keretrendszere tényleges programokat is tartalmaz, programozási interfészeket határoz meg, vagy programozási eszközöket kínál a keretrendszerek használatához. Egy keretrendszer vonatkozhat egy rendszeren belüli funkciók halmazára és azok egymáshoz való viszonyára; egy operációs rendszer rétegeire; egy alkalmazás-alrendszer rétegeire; arra, hogy a kommunikációt hogyan kell szabványosítani egy hálózat bizonyos szintjén; és így tovább. Egy keretrendszer általában átfogóbb, mint egy protokoll, és előíróbb, mint egy struktúra.[11]

### Avalonia

Az Avalonia egy cross-platform UI keretrendszer a dotnethez, amely rugalmas stílusrendszert biztosít és számos operációs rendszert támogat, mint például Windows, Linux, macOS. Az Avalonia kiforrott és gyártásra kész. Béta kiadásban van iOS, Android támogatás és korai fázisban a WASM-en keresztül böngésző támogatás is. Nyílt forráskódú felhasználói felület keretrendszer, amely lehetővé teszi a fejlesztők számára, hogy pixel-tökéletes, platformok közötti alkalmazásokat hozzanak létre C# és XAML használatával. Egyetlen kódbázisból készíthet alkalmazásokat Windows, macOS, Linux, iOS, Android és a web számára.[10]

# Felhasznált nyelvek

## C#

Objektum oriental …

LINQ

Random

Thread

Canvas

Binding

WPF

Model

ViewModel

MVVM

Axaml

### Verziók bemutatása

### Property

### Interface

### Attributumok

### Eventek

## .Net

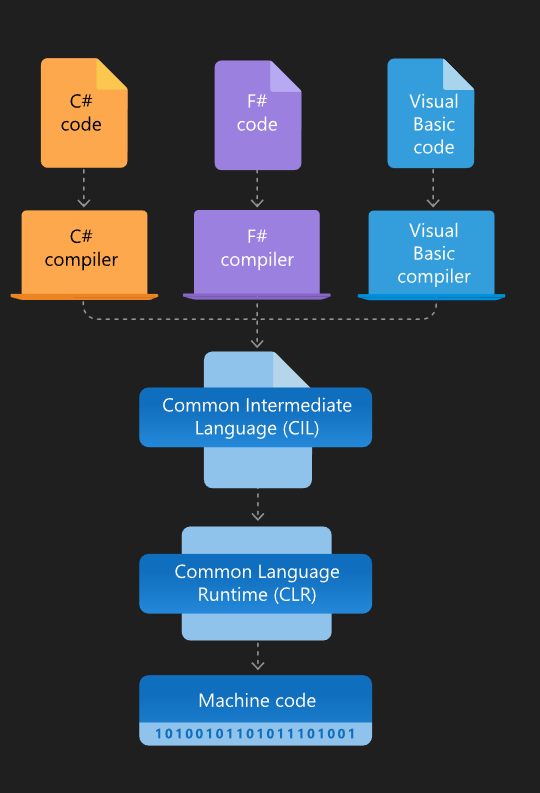
A .NET Framework két fő összetevője a Common Language Runtime és a .NET Framework Class Library.

A Common Language Runtime (CLR) az alkalmazások futtatásáért felelős futtatómotor. Olyan szolgáltatásokat nyújt, mint a szálkezelés, a szemétgyűjtés, a típusbiztonság, a kivételkezelés és még sok más.

Az osztálykönyvtár API-kat és típusokat biztosít a közös funkciókhoz. Típusokat biztosít a karakterláncokhoz, adatokhoz, számokhoz stb. Az osztálykönyvtár API-kat tartalmaz fájlok olvasásához és írásához, adatbázisokhoz való csatlakozáshoz, rajzoláshoz és így tovább.

A .NET alkalmazások a C#, F# vagy Visual Basic programozási nyelven íródnak. A kódot egy nyelvfüggetlen közös közvetítő nyelvre (Common Intermediate Language, CIL) fordítják. A lefordított kódot összeállításokban - .dll vagy .exe kiterjesztésű fájlokban - tárolják.

Az alkalmazás futtatásakor a CLR átveszi az assembly-t, és egy just-in-time (JIT) fordítóprogrammal olyan gépi kóddá alakítja át, amely a számítógép adott architektúráján futtatható, amelyen az alkalmazást futtatják. [10]



## HTML

## Razor

# Bevezetés a Virus szimulacioba

# Gráfok

# Kivitelezés és bemutatás

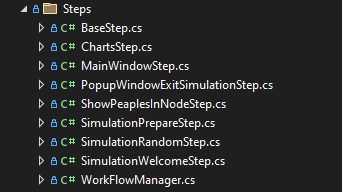
A szimuláció két fő projectből áll. Az első projekt amit a felhasználónak futtatni kell az a VirusSimulator\_UI alkalmazás.

## MVVM használata

Az Avalonia framework egy WPF hez hasonló project környezet ahol általában megszokott hogy a V- Window osztályt használjuk az adatok megjelenítésére. Első lépésként az újonnan kialakított Windowokhoz hozzárendeltük az ősüket a Window osztályt.Ezt követően létrehozzuk a VM - ViewModel osztályunkat amiből majd a többi ViewModel osztályunkat fogjuk létrehozni. A Model az adott esetünkben BaseStep nevezető ősosztálybó ered mely tartalmaz egy event handler, egy User Control publikus metódust amin keresztül az adott osztályhoz tartozó View-t (kinézetet) kérhetjük le.

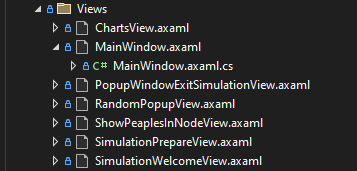
### M-Model,Step

Az MVVM struktúrának az egyik legfontosabb elem az a Model avagy a mi esetünkben a Stepnek elnevezett osztályok. Ezekben az osztályokban történik meg, hogy létrehozzuk a hozzájuk rendelt ViewModelt és a View osztályokat. Továbá létrehoztunk egy Statukis osztályt melynek szigorúan csak egy instanciája létezik. Ezt az osztályt WorkFlowManager-nek neveztük el. A workflow manager ben létrehoztük egy statukis Dictionary szótárt amit a Step modelek megőrzésére fogunk hazsnálni. Ezzel a Workflowmanager osztállyal egyszerűen fogunk tudni lépegetni a User Control ablakokon keresztül. A fukcionalitás mögötte az, hogy a szótárban elmentett Step osztályokhoz hozzárendeltünk egy System.String kulcsot. A WorkflowManager-ben implementáltunk egy SaveStep metódust aminek a feldata amikor új instanciák vannak létrehozva az adott Step modelünkből azt elmentjük a szótárunkba. Az elmentés során elősször ellenőrizzük, hogy már az adott insatncia szerepel-e már a szótárban amennyiben igen akkor csak az adott kulcshoz a instanciát felülirjuk. Az ellenőrzés az a GetType() reflekcióval történik aminek a segitségével az instanciónak .Name megadásával megkapjuk a nevét. A könyvtár tartalmának a kinyeréséhez pedig létrehoztunk egy publikus GetStep metodust aminek visszatérési peremáterként a BaseStep osztályt adtuk meg. Mivel tudjuk hogy az osztályaink a BaseStep osztályből lettek öröktve azért a poliformizmus szabályainak köszönhetőén az osztályok átt tudjuk polifomizálni a saját Steppünkre.



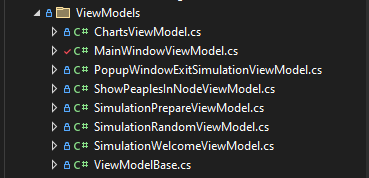
### V- View

Minden egyes View fájból áll, az első fájl az egy Avalonia XAML kiterjesztésű fájl. Ebben a fájlban megadjuk az alapvető képernyő beállitásokat mint páldául a képernyő szélessége és hosszúsága, a képrenyő indulásának helye amia z esetünkben az ablakunk közepe és esetünkben a CanResize=”False” ra állítása ami a képrenyőnk nagyításánka és kicsinyítésének a megtiltását teszi lehetővé. Az említett fő paraméterek után pedig már minden egyes Stackpanel vagy esetleg Grid rács segítségével jelenítjük meg az adatokat. A Stackpanel számunkra előnyösebb mivel kevesebb kódolást kíván a használatához. A stack panel mondhatni a HTML nyelveben a <div> nek felel meg, amiben az eleminket horizontálisan vagy vertikálisan tudjuk rendezni. A grid rács pár sorral több sor kódot igényel mivel előre definiálnunk kell a sorainkat és oszlopainkat és azoknak nagyságait.



### VM- ViewModel

Eza z osztály felelős azért, hogy az osztályban hazsnált változókat property formájában deklaráljuk. Azokhoz a propertykhez amelyekről tudjuk, hogy változni fognak a futás során egy [Reactive] attributumot rendelunk. Erről az attributumről az Avalonia framework gondoskodik, ami annyit tesz számunkra, hogy figyeli a változónak az értékét és ha észlel változást akkor meghivja a INotification interfacen keresztül az INotifyPropertyChange metódust amit keresztül Invokolva van egy event hívás aminek az eredményeként megváltozik a változónk értéke a UI-on. A kész ViewModelt bele integraáljuk a View osztályunkba a DataContex-en keresztül. Ennek köszönhetően majd tudjuk használni Bindingen keresztül a változókhoz az érték hozzárendelését.



## Főoldal

Az alkalmazás megnyitásával megnyílik számunkra a főoldal ahol fentről bal felső sarokban találunk egy vissza gombot, ez a gomb a szimuláció során bármikor alkalmazható a főoldalra visszajutáshoz. Az ablak közepén egy számlátót találonk melynek az a funkciója hogy a szimuláció elindtása után számlálja nekünk az eltelt napokat. Közvetlen mellette pedig 3 egyforma nagyságú gombot találunk melyek a szimuláció elindítására, megállítására és befejezésre szolgálnak, ezek a gombok csakis a szimuláció során alkalmazhatóak. Amig a szimuláció nincs elinditva addig ezeknek a gomboknak a használata le van tiltva. Az applikáció ablaka nem méretezhetőre szabtuk, így garantáljuk azt, hogy a grafikus interfészen nem fognak bekövetkezni probélák a nagysággal kapcsolatban. Ezek alatt található egy üdvözlő szöveg, ami alatt pedig három gomb található a szimulációnak a kezdéséhez vezetnek. Az első gomb egy teljesen új szimulációt hoz nekünk létre, a második gomb már előre elmentett szimulációnak a beolvasását engedélyezi a harmadik pedig egy teljesen véletlenszerú gráfot hoz nekünk létre.A főoldal felépítéséről még annyit kell tudnunk, hogy az oldal külső struktúrája fix ezért a MainWindow felel. A Mainwindowban pedig találunk egy User controll ablakot amit a szimuláció során váltogatunk a nekünk szükséges ablakra.

## Új szimuláció

# Elmélet

## 

## Projekt kezdése

SOC - Separation Of Concerns

## Vírus elmélet

A mi járványszimulációnkban minden, fertőzött személy képes lesz fertőzni akivel találkozik. Minden egyes kigenerált személynek generálunk barátokat ismerősöket, akik napról napra találkoznak egymással. A szimulacioban több lehtőséget is integráltunk mint például a léguti csatornák eltakarása maszkok segitségével. A szimulácio legelső lépése, hogy létrehozzuk az ember osztályt ahol az ember karakterisztikáját fogjuk ledefiniálni mint például :   
Immunitás 1%   
Fertőzőképesség amennyiben a személy nem fertőzőtt ennek a deaful értéke 0

Amennyiben fertőzött akkor 0 és 100 % között lesz a fertőzőképessége, amivel aztán képes lesz megfertőzni a barátait.

Gauszien -Fontos tényező lesz számunkra a Gauszien eloszlás. Guaszien elosztás abban lesz a s egitségünkre, hogy képesek leszünk pontosabb random értékeket megadni, nagyobb v alószinűséggel fog nekünk a median áttlag mellől generálni egy számot.

Mask

Lockdown

Barátok

Beállitható lesz egy ugynevezett socail socialty

A program megírása az ember osztállyal kezdődött, ahol deklaráltuk a fent említett változókat. Az osztály létrehozása után pedig létrehoztunk egy Inicializációs metódust amivel az egész szimulációnkat elindítjuk, a megfelelő kezdeti értékekkel. Kezdeti beállitásoknak 1% teljes immunitást és 500 000 ember generálunk és 0 fertőzöttel kezdünk. Majd a felhasználónak lehetősége lesz kiválasztani, hogy melyik országokban szeretné elhelyezni a vírust. Ezt követően a több szálon fog futni a programunk.

## Környezet

Általános értelemben a programozási környezet olyan hardvert és szoftvert egyesít, amely lehetővé teszi a fejlesztő számára, hogy alkalmazásokat készítsen. A fejlesztők általában integrált fejlesztőkörnyezetekben vagy IDE-kben dolgoznak. Ezek összekapcsolják a felhasználókat a kódjuk helyes megírásához és teszteléséhez szükséges összes funkcióval.[6]

### Programozási nyelv c#

A C# egy objektumorientált, komponensorientált programozási nyelv. A C# olyan nyelvi konstrukciókat biztosít, amelyek közvetlenül támogatják ezeket a fogalmakat, így a C# természetes nyelvvé válik a szoftverkomponensek létrehozásához és használatához. A C# a kezdetek óta új funkciókkal bővült, hogy támogassa az új munkaterhelést és a kialakulóban lévő szoftvertervezési gyakorlatokat. Alapjában véve a C# egy objektumorientált nyelv. Típusokat és azok viselkedését definiáljuk.[3]

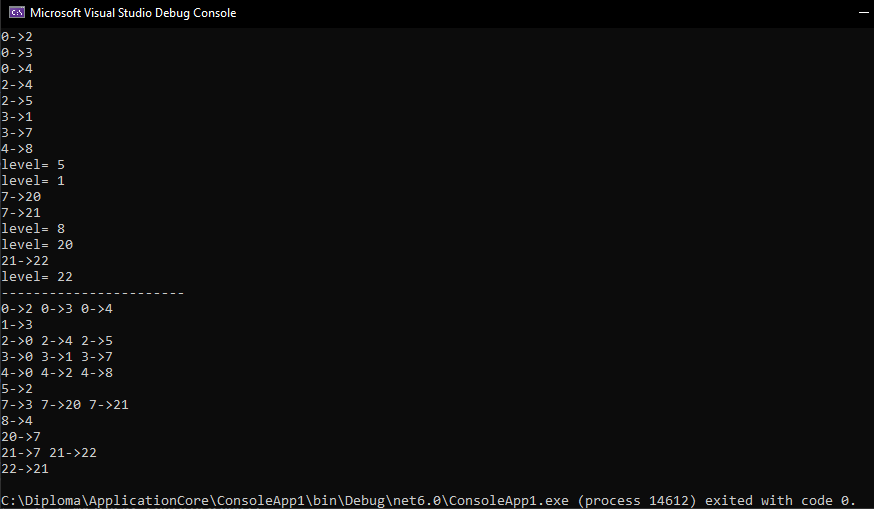
### .Net 6 framework

.NET egy ingyenes, platformokon átívelő, nyílt forráskódú fejlesztői platform, amely számos különböző típusú alkalmazás létrehozására alkalmas.A .NET segítségével többféle nyelvet, szerkesztőt és könyvtárat használhat webes, mobil, asztali, asztali, játék és IoT alkalmazások készítéséhez.

## Felhasznált eszközök

### Visual Studio 2022

Az integrált fejlesztőkörnyezet (IDE) egy olyan funkciógazdag program, amely a szoftverfejlesztés számos aspektusát támogatja. A Visual Studio IDE egy kreatív indítópad, amelyet kódszerkesztésre, hibakeresésre és kódépítésre, majd egy alkalmazás közzétételére használhat. A Visual Studio a legtöbb IDE által biztosított szabványos szerkesztőn és hibakeresőn túl tartalmaz fordítókat, kódkiegészítő eszközöket, grafikus tervezőket és még sok más, a szoftverfejlesztési folyamatot javító funkciót.



### MSBUILD

### SQL adatbázis

Az SQL Server a Microsoft által kifejlesztett és forgalmazott relációs adatbázis-kezelő rendszer (RDBMS).

Más RDBMS szoftverekhez hasonlóan az SQL Server is az SQL-re épül, amely a relációs adatbázisokkal való interakcióra szolgáló szabványos programozási nyelv. Az SQL Server a Transact-SQL-hez, vagyis a T-SQL-hez, az SQL Microsoft általi implementációjához kapcsolódik, amely egy sor saját programozási konstrukcióval egészül ki.

Az SQL Server több mint 20 éve kizárólag Windows környezetben működik. A Microsoft 2016-ban Linuxon is elérhetővé tette. Az SQL Server 2017 2016 októberében vált általánosan elérhetővé, amely Windowson és Linuxon is futott.[2]

### SQL server management studio

Az SQL Server Management Studio (SSMS) egy integrált környezet bármilyen SQL infrastruktúra kezelésére. Az SSMS segítségével az SQL Server, az Azure SQL Database , az Azure SQL Managed Instance, az SQL Server on Azure VM és az Azure Synapse Analytics összes összetevőjét elérheti, konfigurálhatja, kezelheti, adminisztrálhatja és fejlesztheti. Az SSMS egyetlen átfogó segédprogramot biztosít, amely a grafikus eszközök széles csoportját számos gazdag szkriptszerkesztővel ötvözi, hogy a fejlesztők és az adatbázis-adminisztrátorok számára minden képzettségi szinten hozzáférést biztosítson az SQL Serverhez.[1]

### Git

A Git egy verziókezelő rendszer, amelyet a számítógépes fájlok változásainak nyomon követésére használnak. Általában a szoftverfejlesztés során a forráskódok kezelésére használják.

A Git a forráskódban bekövetkezett változások nyomon követésére szolgál.  
Az elosztott verziókezelő eszközt a forráskód kezelésére használják  
Lehetővé teszi több fejlesztő közös munkáját  
Támogatja a nem lineáris fejlesztést a több ezer párhuzamos ága révén

A Git jellemzői Nyomon követi az előzményeket, Ingyenes és nyílt forráskódú, Támogatja a nem ,lineáris fejlesztést, Biztonsági mentéseket készít, Támogatja az együttműködést, Könnyebb az elágazás

### UI

WPF segitségével

## Gráfok

Irányítatlan gráf meghatározása

Az irányítatlan gráf egy gráf, azaz olyan objektumok (csúcsok vagy csomópontok) halmaza, amelyek egymással kapcsolatban állnak, és ahol minden él kétirányú. Az irányítatlan gráfot néha irányítatlan hálózatnak is nevezik. Ezzel szemben az olyan gráfot, ahol az élek egy irányba mutatnak, irányított gráfnak nevezzük.

Irányítatlan gráf rajzolásakor az élek jellemzően a csomópárok közötti vonalak formájában jelennek meg, ahogyan azt a következő ábra szemlélteti.

Gráfok generálása  
DFS();

BFS();

public void BFS2(int node = 0, int infectedPercent = 2)

{

//For be able to generate random graph, probably I should start with 0

// I going to assign 0 -> new friend/s then increment 1 -> new friends

// If I follow this logic then it may be possible to generate correctly.

if (peoplesList[node].FriendsList.Count > 0)

{

if (!visitedNodes.Contains(node))

{

foreach (var nod in peoplesList[node].FriendsList)

{

if (visitedNodes.Contains(nod))

{

if (peoplesList[node].FriendsList.Count <= 1)

{

//here is our leafe

Console.WriteLine("level= " + node);

//here node has no more childrens

//these objects-peaple will infects as first randomly

}

}

else

{

//here we can check the parent of the node of node is infected then

//we can pass the virus to his parent

//End of checking

if (!myStack2.Contains(nod))

{

myStack2.Enqueue(nod);

}

leafNodes.Add(nod);

Console.WriteLine(node + "->" + nod);

}

}

visitedNodes.Add(node);

}

}

else

{

Console.WriteLine(node + "-> None");

}

if (myStack2.Count > 0)

{

BFS2(myStack2.Dequeue());

}

### Irányítatlan gráf

### Irányított gráf

### Mélységi bejárás (DFS -Depth-first search)

Mi az a Depth First Search (DFS)

A DFS algoritmusban egy forráscsomóponttal kezdünk, és a lehető legmélyebbre megyünk. Más szavakkal: egy ágon belül meglátogatjuk az összes gyermekét, mielőtt továbblépnénk egy másik ágra.

Ha jól ismerjük a Depth First Search-et, akkor nagyon könnyű lenne megérteni a rendszertervezési koncepciókat és feltörni az interjúkérdéseket.

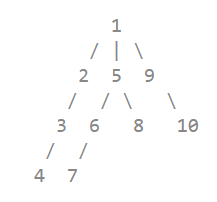
DFS

A forráscsúcson kezdődik

Kiválasztja az egyik nem látogatott szomszédot, és meglátogatja azt.

Ez a folyamat ismétlődik rekurzívan, és a rekurzió ismétlődik egy ágba, amíg nem tud tovább menni.

Az alábbiakban a DFS C# nyelven történő megvalósításához szükséges mintakódrészletet mutatjuk be. Létrehoztam egy fát az előző bejegyzésemben tárgyalt kóddal.  
A DFS természetéből adódóan minden egyes ág mélységébe kell mennünk, mielőtt egy másik ágra lépnénk. Az algoritmus az 1. csomópontból kiindulva a 2., 3. és 4. csomóponton halad át. Végül az 5. csomópont következő ágára lép.



Mivel az 5. ág szintén két további ágra oszlik, az algoritmus a 6. és 7. csomóponthoz megy, és végül a 8. csomóponthoz érkezik.

Végül a 9. és 10. csomópontokból álló utolsó ágon halad át.

Továbbá, ahogy a kódban látható, ezt rekurzívan végzem a DFSHelper módszerben. A kódban egy tömböt tartottam a meglátogatott csomópontok jegyzeteléséhez.

public class GraphD

{

LinkedList<int>[] linkedListArray;

public GraphD(int v)

{

linkedListArray = new LinkedList<int>[v];

}

///

/// The method takes two nodes for which to add edge.

///

///

///

///

public void AddEdge(int u, int v, bool blnBiDir = true)

{

if (linkedListArray[u] == null)

{

linkedListArray[u] = new LinkedList<int>();

linkedListArray[u].AddFirst(v);

}

else

{

var last = linkedListArray[u].Last;

linkedListArray[u].AddAfter(last, v);

}

if (blnBiDir)

{

if (linkedListArray[v] == null)

{

linkedListArray[v] = new LinkedList<int>();

linkedListArray[v].AddFirst(u);

}

else

{

var last = linkedListArray[v].Last;

linkedListArray[v].AddAfter(last, u);

}

}

}

internal void DFSHelper(int src, bool[] visited)

{

visited[src] = true;

Console.Write(src + "->");

if (linkedListArray[src] != null)

{

foreach (var item in linkedListArray[src])

{

if (!visited[item] == true)

{

DFSHelper(item, visited);

}

}

}

}

internal void DFS()

{

Console.WriteLine("DFS");

bool[] visited = new bool[linkedListArray.Length + 1];

DFSHelper(1, visited);

}

}

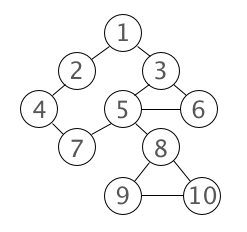


### Szélességi bejárás(BFS - Breadth-First Search)

A szélesség-első keresés annyiban egyedülálló a mélység-első kereséshez képest, hogy a szélesség-első keresés segítségével meg lehet találni a legrövidebb utat 2 csúcs között. Ez súlyozatlan gráfot feltételez. A legrövidebb út ebben az esetben a két csúcs közötti legkisebb számú éllel rendelkező út.

Ezekben az esetekben hasznos lehet, ha a kezdőcsúcstól a gráf összes csúcsához vezető legrövidebb utat kiszámítjuk, és egy olyan függvényt biztosítunk, amely lehetővé teszi, hogy az ügyfélalkalmazás lekérdezze a legrövidebb utat bármely más csúcshoz. Létrehoztam egy ShortestPathFunction funkciót C# nyelven, amely pontosan ezt teszi. Kiszámítja a kezdőcsúcstól az összes csúcshoz vezető legrövidebb utat, majd visszaad egy függvényt, amely lehetővé teszi, hogy az eredeti kezdőcsúcstól bármelyik csúcshoz vezető legrövidebb utat lekérdezzük.

A Breadth-first keresést arra használjuk, hogy a kezdőcsúcstól kiindulva bejárjuk a gráfot, és az egyes csomópontokhoz ( az előző csomópontokhoz ) vezető utat egy previous nevű C# szótárba tároljuk. Egy csomóponthoz vezető legrövidebb út megtalálásához a kód megkeresi a célcsomópont előző csomópontját, és addig folytatja az összes előző csomópont keresését, amíg el nem jut a kiindulási csomóponthoz. Mivel ez lesz az útvonal visszafelé, a kód egyszerűen megfordítja a listát, és visszaadja azt.



Console.WriteLine("\r\n BFS");

/// Breadth-First Search

var vertices = new[] { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };

var edges = new[]{Tuple.Create(1,2), Tuple.Create(1,3),

Tuple.Create(2,4), Tuple.Create(3,5), Tuple.Create(3,6),

Tuple.Create(4,7), Tuple.Create(5,7), Tuple.Create(5,8),

Tuple.Create(5,6), Tuple.Create(8,9), Tuple.Create(9,10)};

var graphB = new Graph<int>(vertices, edges);

var algorithms = new Algorithms();

Console.WriteLine(string.Join(", ", algorithms.BFS(graphB, 1)));

// 1, 2, 3, 4, 5, 6, 7, 8, 9, 10



# Járványszimuláció

## Virus terjedésének lehetőségei és rövid megnevezései

| **Részlegek** | **Description** | **Denotation** |
| --- | --- | --- |
| Az első megerősített eset | Az első COVID-19-vel fertőzött beteg | *P* |
| Fertőzésnek kitett személy | Az egyén szoros kapcsolatban állt fertőzött személlyel. Az egyén a lappangási időszakban van, miután megfertőződött a betegség kórokozójával, és még nincsenek látható klinikai tünetei. | *E* |
| Fertőző | Az egyén befejezte a lappangási időszakot, megfertőzhet más embereket, és klinikai tünetek jelentkeznek nála. | *I* |
| Fertőző, fel nem fedezett | Az egyén továbbra is megfertőzhet más embereket, klinikai tünetei lehetnek, de a hatóságok nem észlelik és nem regisztrálják. Ezen időszak után az ebben a rekeszben lévő egyén a betegség súlyosságától és az egészségügyi állapottól függően átkerül a Gyógyult (Run) vagy a Halott (D) rekeszbe. | *Iun* |
| Visszakapott, de fel nem fedezett | Az egyén korábban nem bizonyult fertőzőnek, az is lehet, hogy a tünetmentes eset, felépült, és természetes immunitást alakított ki a vírussal szemben. | *Run* |
| Kórházban vagy otthon karanténban van, és fel fog épülni. | Az egyén kórházban van (elkülönítve vagy karanténban otthon), és még mindig megfertőzhet másokat. Az egyént a hatóságok észlelték és jelentették. Enyhe esetként ennek az állapotnak a végén az egyén felépül. | *HR* |
| Recovered but previously detected as infectious | Az egyén korábban fertőzőnek bizonyult, meggyógyult, és természetes immunitást alakított ki a vírussal szemben. | *Rde* |
| Kórházba kerül, de meg fog halni | Az illető kórházba kerül, és még mindig megfertőzhet más embereket. Súlyos esetként ennek az állapotnak a végén az egyén meghal. | *HD* |
| Halott | Az egyén nem élt túl új koronavírusos megbetegedést. | *D* |

### Világjárvány

A világjárvány egy új betegség világméretű elterjedése. A vírusos légzőszervi megbetegedések, például az új influenzavírus vagy a COVID-19 koronavírus által okozott megbetegedések válhatnak a legnagyobb valószínűséggel világjárvánnyá.

A világjárvány nem azonos a járvánnyal. Járvány esetén egy egészségügyi állapot sokkal több megbetegedés fordul elő, mint amennyi egy közösségben vagy régióban normális esetben kialakulna, de a betegség nem terjed tovább.

A múltban számos influenzajárvány volt. A pandémiás influenzák gyakran állati influenzavírusokból erednek, és nem azonosak a szezonális influenzával. A pandémiás influenzavírus ellen - ha egyáltalán van - csak kevés embernek van immunitása - még akkor is, ha szezonális influenzán vagy szezonális influenza elleni védőoltáson esett át.[7]

# Bugs/ issues

myRectanglesPoints.Count; // here we need to fix the bug counting for Id

# Felhasznált irodalom

[1] <https://docs.microsoft.com/en-us/sql/ssms/sql-server-management-studio-ssms?view=sql-server-ver15>

[2] <https://www.sqlservertutorial.net/getting-started/what-is-sql-server>

[3] <https://docs.microsoft.com/en-us/dotnet/csharp/tour-of-csharp/>

[4] <https://www.techtarget.com/searchdatacenter/definition/Linux-operating-system>

[5] <https://www.computerhope.com/jargon/w/windows.htm>

[6] <https://themanifest.com/software-development/blog/programming-engineer>

[7] <https://www.incredibuild.com/integrations/visual-studio>

[8] <https://learn.microsoft.com/en-us/nuget/what-is-nuget>

[9] <https://dotnet.microsoft.com/en-us/learn/dotnet/what-is-dotnet-framework>

[10] <https://github.com/AvaloniaUI/Avalonia>

[11] <https://www.techtarget.com/whatis/definition/framework>