Haladó Fejlesztési Technikák
(C# / Prog3 – Neptun kód: AAAAAA)

A világjárvány kitörése óta minden epidemiológus a terjedés modellezésével és a különböző korlátozások hatásának vizsgálatával foglalkozik. Az a feladat, hogy egy egyszerű szimuláción keresztül bemutassuk a távolságtartás és a maszkviselés hatását a fertőzés terjedésére.

1. Hozzon létre egy **PROG3_NEPTUNKÓD** nevű solution-t. A solution tartalmazzon egy **DataGenerator** nevű Class Library-t, azon belül egy Generator nevű osztályt, aminek legyen egy `public XDocument GenerateData(string neptun, int num, int width, int height)` szignatúrájú metódusa.

```
<outbreak>
  <location x="109" y="25" />
  <strength>7</strength>
</outbreak>
```

   A metódus a *num* paraméterben megadott darabszámú véletlen fertőzési gócpontot (*outbreak*) hoz létre a fent látható XML formátumban (a paraméterben megadott szélesség/magasság korlátok közötti véletlen X és Y értékekkel, valamint 1..9 közötti véletlen strength értékkel). A több gócpontot egyetlen XDocument példányként adja vissza, később majd a gócpontokból létrejövő személyeket a konzolon megjelenő karakterekkel fogjuk reprezentálni.

2. A személyek reprezentálására hozzon létre egy **InfectionLogic** nevű Class Library-t, amiben definiáljon egy `Person { int ID, int x, int y, bool IsInfected, bool IsMasked }` osztályt.
   a. Legyen az osztályban egy `public void MovePerson(int dx, int dy, int width, int height)` szignatúrájú metódus, ami a (dx, dy) értékeket mozgásvektorként értelmezve odébb mozgatja a személyt (ügyelve arra, hogy a paraméterként megkapott szélességű+magasságú konzolról ne lépjen le).
   b. Legyen az osztályban egy `public bool InteractWith(Person otherPerson)` szignatúrájú metódus.
   c. A metódusok és az osztály itt nem részletezett működése a következő feladatban definiált tesztekből a TDD elveinek megfelelően implementálandó.

3. Hozzon létre egy **InfectionTests** nevű class library-t, amihez adja hozzá a szükséges nUnit nuget referenciákat. A Person osztály metódusaihoz írjon teszteseteket (és közben a Person osztályt fejlessze ennek megfelelően). A tesztesetek az alábbi működési részleteket ellenőrizzék.
   a. Az egymás után létrehozott Person osztálybeli példányok létrehozáskor beállított ID tulajdonsága sorban automatikusan egyesével növekszik, ezáltal a személyek azonosíthatóak (ezt NEM a DatabaseGenerated attribútummal kell most megoldani, hanem a **legegyszerűbb** módon). Oldja meg, hogy a teszthez használhatóak legyenek az alábbi TestCase attribútumok: [TestCase(95)] [TestCase(42)] [TestCase(23)] (használja is ezeket az attribútumokat a forráskódban)!
   b. A MovePerson hívásakor a személy nem léphet ki a paraméterként megadott konzolos méretkorlátokból (egyik irányban sem).
   c. Ha az InteractWith metódus otherPerson paramétere null, akkor az InteractWith metódusnak ArgumentNullException-t kell dobnia.
   d. Ha az InteractWith metódusban bármelyik személy (a paraméterként kapott vagy az aktuális példány) IsMasked tulajdonsága false, és a távolságuk nem nagyobb, mint 2, és bármelyik személy fertőzött (IsInfected), akkor mindkét személy fertőzött lesz, és a metódus visszatérési értéke true.
   e. Minden más esetben a személyek IsInfected tulajdonsága nem változik, és az InteractWith metódus visszatérési értéke false.

4. Hozzon létre egy **InfectionApp** nevű konzol alkalmazást.
    a. Generáljon le 6 gócpontot (a `GenerateData` metódusban a konzol méreteit felhasználva). Az eredményből mindegyik outbreak pozíciójában hozzon létre egy fertőzött személyt, csak a felén legyen maszk (IsMasked). A személyeket rakja bele egy listába.
    b. Ugyanebbe a listába rakjon bele 50 nem fertőzött, véletlenszerű pozícióban lévő személyt is, ezeknek is csak a felén legyen maszk.
    c. Jelenítse meg a legenerált személyeket a konzolon! Megjelenítéskor használjon különböző színt a fertőzött és nem fertőzött személyhez, valamint használjon különböző karaktert a maszkkal rendelkező és nem rendelkező személyhez.

5. Billentyűleütésig (Console.KeyAvailable) menjen egy „végtelen” ciklus, amely tizedmásodpercenként fusson le.
    a. Minden személy véletlenszerűen mozogjon valamerre maximum egy pozícióval (a `MovePerson` metódus dx és dy paraméterei [-1 ; +1] közötti értéket vehetnek fel).
    b. Minden fertőzött személy-példánynak hívja meg az InteractWith() metódusát az összes **nem** fertőzött személlyel.
    c. Az InteractWith() hívások eredményei alapján készítsen a ciklus futása közben egy listát, amelynek elemei azt tárolják, hogy melyik személy fertőzött, melyik személyt fertőzte meg, és melyik pozícióban (a fertőző személy pozíciója kell ide).
    d. Rajzolja ki a személyeket ismét a konzolon!

6. Amíg a fenti ciklus fut, addig folyamatosan írja ki (Console.Write) a listában lévő fertőzött és nem fertőzött személyek darabszámát a konzol ablak bal felső sarkába, fehér színnel. Ehhez hozzon létre és indítson külön szálat Task alapon; a megoldáskor ügyeljen a helyes szálszinkronizációra.

7. Billentyűleütés hatására a szimuláció befejeződik, és a Task-ot is megkérjük a leállásra (CancellationTokenSource segítségével). Kilépés előtt a ciklus futása közben elkészült listát felhasználva jelenítsük meg az alábbiakat:
    a. Személyenként az okozott fertőzések darabszámát, és ezen fertőzések átlagos X és átlagos Y pozícióját; darabszám szerinti csökkenő sorrendben.
    b. Az eredeti GenerateData() eredményből kiszedett gócpont-listához joinoljuk a fertőzésekről készített listát, és szeretnénk látni az eredeti kitörések koordinátáit (X,Y), az ugyanazon pozícióban megtörtént fertőzésekkel (id1_fertőzőPersonId, id2_fertőződöttPersonId) együtt.

8. A 7/A lekérdezés eredményét Entity Framework Core használatával mentse el egy Service-Based Database tetszőleges adattáblájába.

A szabályok azonosak a laborban írt ZH-val – a kódnak fordulnia kell, és a nem megengedett segédeszköz használata, vagy a használat megkísérlése is csalásnak számít.

Feltölteni a teljes solution könyvtárat ZIP formátumban betömörítve kell, a Moodle rendszeren keresztül a megadott határidőig. A Moodle rendszer hibája esetén a feltöltés https://users.nik.uni-obuda.hu/ff/ URL-en keresztül lehetséges.

Kérdéseket privát chat-ben a Teams rendszeren keresztül lehet feltenni, ezen kívül ZH közben kérjük a „Prog3 / ZH / 2020” team –ben leírt közleményeket folyamatosan figyeljék!

Neptun kód: AAAAAA

Advanced Development Techniques
(C# / Prog3 – Neptun code: AAAAAA)

Since the start of the epidemic, all doctors work with modelling the spread of the disease and the effectiveness of the protective measures. The task is to create a simple simulation that shows the effectiveness of keeping distance and wearing masks to limit the spread of the disease.

1. Create a solution with the name **PROG3_NEPTUNCODE**. The solution should contain a Class Library named **DataGenerator**, with a class named Generator inside, that should have a method with the signature `public XDocument GenerateData(string neptun, int num, int width, int height)`.

```
<outbreak>
  <location x="109" y="25" />
  <strength>7</strength>
</outbreak>
```

   The method should use the *num* parameter to determine how many outbreaks should be generated in the XML format seen above (with random X and Y values using the width/height parameters as limits, and random strength values between one and nine). You should return multiple outbreaks in a single XDocument instance. Later the outbreaks will be used to generate people, and the people will be displayed as characters in the console window.

2. To represent the people, create an **InfectionLogic** Class Library, with a `Person { int ID, int x, int y, bool IsInfected, bool IsMasked }` class inside.
   a. The class should have a `public void MovePerson(int dx, int dy, int width, int height)` method, that should interpret the received (dx, dy) values as a movement vector to shift the location of the person (be careful that it should not leave the console, the console sizes are also specified in the parameters).
   b. The class should have a `public bool InteractWith(Person otherPerson)` method.
   c. All operations of the methods and the class that are not written here should be implemented using the test requirements and following the TDD principles.

3. Create an **InfectionTests** class library, add the required nUnit nuget references. Write test methods for the Person class (and in the meantime, develop the Person class as needed). The test cases should check the following operational details:
   a. The Person class instances that are created after each other should have an initialized ID property where the values are sequentially incremented automatically, so the people can be identified (do NOT use the DatabaseGenerated attribute for this, use the **simplest** approach). Make it so that the following TestCase attributes are usable with the test method: [TestCase(95)] [TestCase(42)] [TestCase(23)] (and use these attributes in the source code)!
   b. When calling the MovePerson, it is not allowed to step out of the size limits specified in the parameters (in any direction whatsoever).
   c. If the otherPerson parameter of the InteractWith method is null, then the InteractWith method should throw an ArgumentNullException.
   d. In the InteractWith method, if any person (the one in the parameter or the current instance) have the IsMasked property as false, and their distance is not bigger than two, and either person is infected (IsInfected), then both people should become infected, and the return value of the method should be true.
   e. In all other cases, the IsInfected property should not change at all, and the return value of the InteractWith method should be false.

4. Create an **InfectionApp** console application.
    a. Generate six outbreaks (use the console sizes in the `GenerateData` method). Using the result, in every outbreak position create an infected person. Only half of them should wear masks (IsMasked). Put the instances into a list.
    b. Into the same list, generate 50 non-infected, randomly positioned people. Only half of them should wear masks,
    c. Display the generated people in the console! When displaying people, use different colours for the infected and non-infected people. Use different characters for the masked and non-masked people.

5. Until a keypress (Console.KeyAvailable) there should be an "infinite" loop running ten times in every seconds.
    a. Every person should be moved randomly to a neighbouring position (the dx and dy parameters of the `MovePerson` method should have values from [-1 ; +1]).
    b. You should call the InteractWith() method for all infected people with all **non**-infected people in the parameter.
    c. Using the return values of the InteractWith() method calls, you should prepare a list during the execution of the loop: the items should store which person was infected by which other person, and in which position (use the position of the person that gave the infection to the other).
    d. Draw the people to the console again!

6. Define and launch a new thread using a Task, that will go on while the previous loop is running; and it will continuously write out (Console.Write) the number of infected and non-infected people to the top-left corner of the console window, using a white colour. During the implementation, take extra care of thread synchronization.

7. When a keypress happens, the simulation loop stops, and we should ask the Task to stop as well (using a CancellationTokenSource). Before exiting, using the list prepared during the loop execution you should display these:
    a. The number of infections caused for every person instance, along with the average X and Y coordinates of the infections; ordered by the number of infections, descending.
    b. Using the results of the original GenerateData() output, join the list of outbreaks to the list of infections. We want to see the original outbreak coordinates (X,Y) along with the infections that happened on that location (id1_infectedByThisPersonId, id2_personIdThatWasInfected).

8. Save the results of the 7/A query using Entity Framework Core into a custom datatable of a Service-Based Database.

The rules are the same as for midterms written in a computer lab: the code must compile, and using unauthorized help or even trying to use any unauthorized help is considered cheating.

You have to upload the full solution folder, compressed to a ZIP file, via the Moodle system until the specified deadline. In case of errors with the Moodle system, you have to use the https://users.nik.uni-obuda.hu/ff/ URL for uploads.

You can ask questions in a private chat via the Teams system, and during the midterm please watch the „Prog3 / ZH / 2020" team, as we will publish announcements there!

Neptun code: AAAAAA