**Programozás III.**

A kapott DataGenerator.dll –ben van egy DataGenerator namespace, azon belül egy Generator osztály, aminek van egy public XDocument GenerateData(string neptun, int num, int width, int height) metódusa. A metódus a megadott darabszámú véletlen kitörést hoz létre, az ebből jövő betegeket majd a képernyőn megjelenő karakterek fognak szimulálni; a cél a kitörések kordában tartása.

1. Hozzon létre egy solution-t, amiben legyen egy konzol alkalmazás, amihez kézzel adja hozzá a kapott DLL-t referenciaként (a szokványos „Add Project Reference” ablak alján lévő „Browse” gombbal). Ezt követően próbálja ki, hogy a metódus használható –e a konzol alkalmazásból.
2. Hozzon létre egy tesztelő class library-t, amihez adja hozzá a szükséges nUnit nuget referenciákat, valamint a kapott DLL-t is referenciaként. A GenerateData() metódus-hoz írjon teszteseteket:
	1. Ha a neptunkód paraméter null, ArgumentNullException-t kell dobnia.
	2. Ha a neptunkód nem 6 karakteres, ArgumentException-t kell dobnia.
	3. Ha bármelyik szám paraméter nem pozitív, akkor ArgumentException-t kell dobnia.
	4. A paraméterként megadott NUM paraméterrel azonos mennyiségű OUTBREAK node van az eredményben.
	5. Mindegyik X érték 0..width-1; mindegyik Y érték 0..height-1; mindegyik strength érték 0..10 közötti (három külön tesztmetódus, amik egy közös, LINQ segédmetódust használnak).
3. A konzol alkalmazásban generáljon le 5 outbreak-et. Mind az öt outbreak-hez indítson el 1-1 Taskot. A taskok feladata: a location-ben megadott X,Y koordináta a kitörés központja, és 5 másodpercenként folyamatosan a strength-ben megadott darabszámú „fertőzöttet” helyez el, az X,Y koordinátától maximum 3 egység távolságban. Minden fertőzöttet egy csillag karakter jelképezzen a konzol képernyőn.
4. Mindeközben a felhasználó a WASD gombokkal mozgathatja a szöveges kurzort, és szóköz megnyomására „gyógyít meg” egy beteget. Ekkor a csillag eltűnik, és közben mentsük el a gyógyításokat egy listába: minden gyógyításhoz eltárolandó a gyógyítás azonosító sorszáma, a kitörés azonosító sorszáma, az X és Y koordináta, és az időpont (ami most egy egész szám: a szimuláció kezdete óta eltelt másodpercek száma: DateTime.Now vagy Stopwatch segítségével).
5. Escape hatására a szimuláció befejeződik (a Taskok szépen leállnak). Ekkor az összes gyógyítást mentsük el egy adatbázisba, valamint egy saját XML állományba is.
6. Kilépés előtt jelenítsük meg az alábbiakat:
	1. Kitörés azonosítónként a gyógyítások darabszámát, a legelső és a legutolsó gyógyítás időpontját.
	2. Minden Y koordinátához szeretném látni az adott sorban lévő gyógyítások kitöréseinek darabszámát.
	3. Az eredeti GenerateData() eredményből kiszedett kitörés-listához joinoljuk a „6/A” kérdésre adott választ, és szeretném látni a kitörések koordinátáit, a kitöréshez tartozó gyógyítások darabszámával együtt.

**A szabályok azonosak a laborban írt ZH-val: Nem megengedett segédeszköz használata, vagy a használat megkísérlése is csalásnak számít!**

**Programming III.**

In the received DataGenerator.dll you have a DataGenerator namespace, with a Generator class, that has a public XDocument GenerateData(string neptun, int num, int width, int height) method. The method will generate the specified number of random outbreaks, the patients will be marked as characters in the console; our aim is to cure the patients as they arive.

1. Create a solution, with a console app inside. Manually add the DLL as a reference (using the „Browse” button in the usual „Add Project Reference” window). Then, try if the method is usable from the console app.
2. Create a tester class library, add the necessary nUnit nuget references and the received DLL as well. Write test cases for the A GenerateData() method:
	1. If the neptuncode parameter is null, it should throw an ArgumentNullException.
	2. If the neptuncode is not 6 characters, it should throw an ArgumentException.
	3. If any of the number parameters are non positive, it must throw an ArgumentException.
	4. The result must have as many OUTBREAK nodes that is the value of the NUM parameter.
	5. All X values must be between 0..width-1; all Y values between 0..height-1; all strength values between 0..10 (should be three test methods, each using a common LINQ helper method).
3. In the console app, you should generate 5 outbreaks. Launch 1-1 Task for each outbreaks. The role of the tasks: continuously once in every 5 seconds, “infected” people should be randomly placed around the X,Y coordinate specified in the location XML data node. The maximum distance between the location and the infected people should be 3; and the “strength” data node shows how many infected people is placed down in one step. Every infected is shown as a star character in the console screen.
4. In the meantime, the user can use the WASD keys to move around the console window, and using the space character an infected person is “healed”. Then the star disappears; but also a list is saved about all healings: there is an ID number of the healing, the ID number of the outbreak, the X and Y coordinates, and the timestamp of the healing (which is an integer: the number of seconds since the start of the simulation; use DateTime.Now or Stopwatch).
5. After pressing Escape, the simulation stops (the Tasks are nicely stopped). Then all healings are saved into a database and to a custom self-made XML file as well.
6. Before exiting, show the followings:
	1. For every outbreak ID, show the number of healings, and the first and the last timestamp
	2. For every Y coordinates I want to see the number of outbreaks that were healed in that row.
	3. Using the original GenerateData() output, join the outbreak-list to the results of the “6/A” exercise; I want to see the coordinates of the outbreaks, and include the number of healings for the given outbreak

**Rules are the same as with midterms written in a computer lab: using non-authorised help, or even trying to use non-authorised help is considered cheating!**