**Témalabor**

**Unity város generálás**

**Készítették:**

**György Márk Attila (ZCVPZT),**

**Li Jiaxiang (SISU6U)**

1. **Bevezetés**

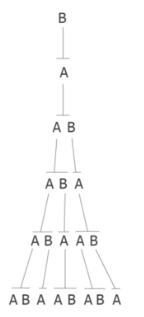
Az általunk választott program egy város generáló algoritmus volt, a Unity játékmotor használatával. Ezt előre megadott szabályok szerint, de mégis randomizálva teszi. A cél az volt, hogy a legenerált város valamennyire élethű legyen. A programmal legenerált városok úgy kell kinézzenek, hogy kibővíthető legyen a program egy teljes értékű játékká. A felhasználó körbe tud repülni a városban, és a UI-al tudja módosítani a legenerált város kinézetét, amit, ha szeretne, újra tud generálni, ha nem tetszik neki a város alaprajza, épületek elrendezkedése, vagy csak szeretne egy másik várost legenerálni esetleg más paraméterekkel.

A fő használt algoritmus az az L-rendszer, vagy a Lindenmayer-System (továbbiakban L-rendszer), amely egy kiinduló szövegből megadott szabályok alapján legenerál egy másik szöveget. Majd miután ez az algoritmus végzett, a legenerált szóból Unity-ben legenerálódik a város. A következőkben fogom leírni, hogy milyen szabályok alapján működik a megvalósított L-rendszer. A felhasználó igénye/kedve szerint tud egyes dolgokat módosítani az algoritmuson, ezzel megváltoztatva a legenerált város kinézetét, méretét.

1. **Az L-rendszer megvalósítása**

Az L-rendszer az egy formális nyelv, amit Lindenmayer Arisztid magyar biológus talált ki eredetileg a növények sejtjeinek modellezésére. Ennek az algoritmusnak számtalan megvalósítása létezik, de mindegyiknek ugyanaz az alapötlete: van egy alap szó, aminek a karaktereit kicseréli valamilyen szabály szerint más karakter(ekre). Az általam megvalósított algoritmus is ilyen.

1. **Az L-rendszer működése**

Az algoritmus eredeti verziója alga:

* Változók: A, B
* Kezdet: A
* Szabályok: (A->AB), (B->A)

Ezekkel a beállításokkal az algoritmus n lefutás után így néz ki:

|  |  |
| --- | --- |
| n=0 | B |
| n=1 | A |
| n=2 | AB |
| n=3 | ABA |
| n=4 | ABAAB |
| n=5 | ABAABABA |
| n=6 | ABAABABAABAAB |
| … | … |

Ezt az algoritmust lehet módosítani, sok féle képen, de amit én választottam az a teknősös módszer:

* Változók: F, G
* Konstansok: +, -, [, ]
* Kezdet: [F]-[G]-[F]-[G]
* Szabályok:
  + (F->F[+G]) vagy (F->G[+F])
  + (G->F[-G]) vagy (G->G[-F])[[1]](#footnote-1)

Amiben a karakterek jelentése:

* F: Menjen a teknős előre egy megadott távolságot.
* G: Menjen a teknős előre egy megadott távolságot.
* +: Forduljon jobbra 90⁰-ot a teknős.
* -: Forduljon balra 90⁰-ot a teknős.
* [: A teknős jelenlegi irányát és pozícióját mentse el a stack tetejére
* ]: A stack tetején levő pozícióra menjen vissza a teknős, és forduljon az ott elmentett irányba, és törölje a stack tetejéről azt.

Ez a megvalósítás nagyon hasonlít a sárkány-görbéhez, de mivel a kezdeti állapot meg a szabályok is mások, mint az abban leírtak, így az algoritmus alapján generált görbe eltérően néz ki. Továbbá a megvalósított algoritmusomban van randomizálás is, ami vagy „levág” a fából ágakat, meg adott változó is véletlenszerű, hogy melyik szabály lesz rá alkalmazva.

1. Úgy kell értelmezni ezeket a szabályokat, hogy az egy sorban szereplők közül véletlenszerűen választ a program, hogy melyik legyen megvalósítva, az aktuális változóra. [↑](#footnote-ref-1)