

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 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 város legenerálásához 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, és a legenerált szöveget hogyan jelenítem meg a Unity használatával. A felhasználó igénye/kedve szerint tud egyes dolgokat módosítani az algoritmuson, ezzel megváltoztatva a legenerált város kinézetét, méretét.

2. Más város generáló algoritmusok

Város generáló algoritmusokból sok verziót készítettek már, melyekhez különböző megvalósításokat használtak. Ezek mind különböző kinézetű városokat eredményeztek. Ebben a fejezetben ezek közül mutatok be hármat.

2.1. Pszeudo végtelen város

Stefan Greuter, Jeremy Parker, Nigel Stewart és Geoff Leach által megalkotott algoritmus volt a pszeudo végtelen város generálás. Ez az algoritmus úgy működött, hogy a játékos ahogy halad végig a térképen és folyamatosan generálódnak az épületek procedurálisan. Az épület kinézete a térképen elhelyezkedő pozíciójából van legenerálva, így pszeudo végtelen ideig tud a felhasználó menni a térképen, anélkül, hogy két ugyanolyan épület menjen el. Ahhoz, hogy ez kivitelezhető legyen, elengedhetetlen, hogy az épületek betöltése és törlődése optimalizálva legyen, különben ez az algoritmus nagyon leterhelné az algoritmust futtató számítógépet.

2.2. Perlin-zajjal generált város

Egy másik megvalósítás Niclas Olsson és Elias Frank által lett megalkotva. Ez az algoritmus a Perlin-zajt használja alapjául, amit Ken Perlin talált ki, eredetileg modellek textúrázására. Ők úgy közelítették meg a város legenerálását, hogy először a területet kerületekre bontották; minden kerületnek különböző a kinézete. Ezután legenerálják a fő és mellékutakat, majd az utak mellett legenerálják a különböző épületeket, az épületet tartalmazó kerületre jellemző épület kinézettel.

2.3. Egy másik L-rendszerrel generált város

Pascal Müllernek van egy L-rendszeren alapuló megvalósítása a város generálásra. A felhasználó ennek az algoritmusnak csak a földrajzi adatokat kell megadja; hogy hol vannak hegyek például. Ebben a megvalósításban is ugyanarra van használva eredetileg az L-rendszer, mint az általam írt programban: először az L-rendszerrel legenerálja az utat, De ez az L-rendszer sokkal több, és komplexebb szabályokat tartalmaz. Ezen kívül az épületek kinézete is L-rendszerrel van legenerálva, ami alapvetően három különböző típusú lehet. Ebből adódóan a legenerált város sokkal valóságosabb, mint az általam megvalósított.

3. Jelentések

Valaminek a <i>Pozíciója</i>	Valaminek a Vector3 típusú koordinátája
Valamihez képest <i>Fel</i>	Valaminek a <i>pozíciója</i> + Vector3(0,0,1)
Valamihez képest <i>Le</i>	Valaminek a <i>pozíciója</i> + Vector3(0,0,-1)
Valamihez képest <i>Jobbra</i>	Valaminek a <i>pozíciója</i> + Vector3(1,0,0)
Valamihez képest <i>Balra</i>	Valaminek a <i>pozíciója</i> + Vector3(-1,0,0)
<i>Út</i>	1*1 egységnyi objektum
<i>Úttest</i>	2 pontot összekötő utakból felépülő egyenes
Valaminek a <i>szomszédja</i>	Valaminek a <i>pozíciójához</i> képest <i>fel</i> , <i>le</i> , <i>jobbra</i> , vagy <i>balra</i> helyezkedik el

4. A teljes algoritmus megvalósítása

A játék elindulása után, vagy a generate gomb megnyomásának hatására indul el a teljes algoritmus, ami több elkülönülő lépésből épül fel: Az L-rendszer legenerálja a stringet, ami alapján az utak megjelennek a játéktérben, eredetileg az összes csak az egyenes út kinézettel, ezután azokon az utakon, amik kereszteződések, kanyarok, vagy csak út végződés, azokon átmegy az algoritmus, és kicseréli őket a megfelelő típusúra és jó irányba forgatja őket. Ezután az összes út mellé, ami mellett van szabad hely, be fog kerülni egy épület, úgy, hogy minden egyes épületre megnézi az algoritmus, hogy az az épület mekkora szabad területtel van körülvéve, erre a DFS algoritmust használja. Majd amelyik épületeknél az jön ki, hogy kisebb területen helyezkedik el, mint 8 egység, akkor egy magas épületet helyez le oda, egyébként kertes házat helyez el oda. Így kialakulnak a városon belül kertvárosos részek és belvárosos részek is. Majd ezek után az algoritmus végig megy az összes legenerált épületen, és mindegyik körül 11*11 egység területen földet helyez el, ezzel megszüntetve az esetlegesen keletkező lyukakat a városban, és körbeveszi a város.

4.1. 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 ABC-je, amiben szerepelnek a konstansok és a változók, vannak szabályok, amik egy változót kicserélnek a szóból egy előre meghatározott (hosszabb) ABC beli karakter sorozatra, és még az algoritmusnak meg kell adni egy kezdeti állapotot is. Majd a szabályok alapján a generált szóra lefuttatható az algoritmus, annyiszor, amennyi meg lett adva, minden lefutásra hosszabbá téve az előző szót. Az általam megvalósított algoritmus is ilyen.

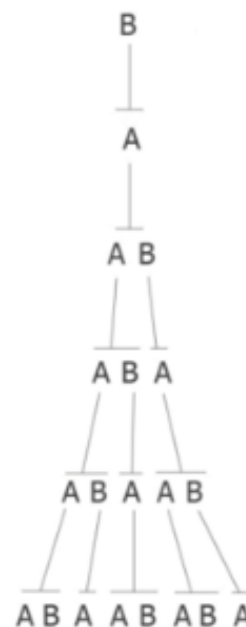
4.1.1. Az L-rendszer működése

Az algoritmus eredeti verziója, az „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
...	...



4.1.2. Az L-rendszer módosítása

- Változók: F, G
- Konstansok: +, -, [,]
- Kezdet: [F]-[G]-[F]-[G]
- Szabályok:
 - $(F \rightarrow F[+G])$ vagy $(F \rightarrow G[+F])$
 - $(G \rightarrow F[-G])$ vagy $(G \rightarrow G[-F])$ ¹

Amiben a karakterek jelentése:

- F: Menjen a teknős előre egy megadott távolságot.
- G: Menjen a teknős előre egy megadott távolságot.
- +: Forduljon jobbra 90°-ot a teknős.
- -: Forduljon balra 90°-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. Azért egy olyan megvalósítást választottam, mai hasonlít a sárkány-görbéhez, mivel abban is négyzetek szerepelnek, és egy város is nagyjából négyzetekből és téglalapokból (ami több négyzet igazából) épül fel. Ezért, ha a sárkány-görbében lévő négyzetek oldalait meghosszabbítjuk, meg párat elhagyunk, akkor a keletkező görbe az szinte ugyan az, mint az általam készített algoritmus által generált görbe.

4.2. Az utak legenerálása

Az algoritmus végigmegy a legenerált stringen, F és G karaktert találva megváltoztatja a jelenlegi pozíciót 2-vel, abba az irányba, amit jelenleg az irány változó jelöl, majd lerak az egyenes út típusból abba az irányba 2-őt, ami jelenleg be van állítva (a fordulások és a mentés betöltések alapján), úgy, hogy elforgatja a jó irányba, amit az alapján tud megtenni, mivel tudja, hogy ez az úttest melyik irányba megy (az úttest generálásánál meg kell adni a kiinduló pontot, az irányt, meg a hosszt). A + és – karakterek annyit csinálnak, hogy megváltoztatják az osztályban szereplő direkción változót. [karaktert olvasva elmenti a stack tömb tetejére a jelenlegi pozíciót és irányt.] karaktert olvasva a jelenlegi pozíciót és irányt beállítja a stack tetején lévő értékekre, majd ezeket az értékeket törli onnan. Miután létrejött az összes út, akkor végigmegy egy algoritmus az

¹ Úgy kell értelmezni ezeket a szabályokat, hogy az egy sorban szereplők közül véletlenszerűen választ a program, hogy melyik legyen megvalósítva, az aktuális változóra.

összes úton, és beállítja, hogy hármass kereszteződés, négyes kereszteződés, kanyar, vagy út vége típusú legyen, amit az alapján tesz, hogy hány szomszédja van, és azok az adott úttól milyen irányba vannak:

- Ha 1 szomszédja van, akkor az egy úttestnek a vége, tehát út vége típusú lesz, úgy forgatva, hogy abba az irányba nézzen, amerre a szomszédja van.
- Ha 2 szomszédja van, akkor lehet kanyar vagy egyenes típusú út:
 - Ha a szomszédjai ellenkező irányban vannak, akkor nem változik meg, tehát marad egyenes út típusú.
 - Ellenkező esetben pedig kanyar típusú kell legyen, és úgy kell forgatni, hogy abba a két irányba nézzen az út, amerre van az a két szomszéd.
- Ha 3 szomszédja van, akkor hármass kereszteződés típusú, és úgy kell forgatni, hogy abba a három irányba nézzen, amelyik irányokban vannak szomszédjai.
- Ha 4 szomszédja van, akkor csak ki kell cserélni négyes kereszteződés típusú útra, amit nem szükséges elforgatni, mivel ez minden irányban ugyanúgy néz ki.

4.3. Az épületek legenerálása

Egy algoritmus végigmegy az összes lerakott úton, és egy Dictionary-ba beleteszi azokat a pozíciókat, amik nem tartalmaznak utat, és legalább egy szomszédja út, ezeket kulcsként tárolja el, és értéként hozzárendeli, hogy az egyik szomszédos úthoz képest ez milyen irányban van. Ezután ezen a Dictionary-n végigmegy és mindegyik kulcsra megnézi, hogy mekkora területen van: DFS algoritmussal meghatározza, hogy az utak hány egységnyi területű olyan területet vesznek körbe, ami tartalmazza azt a pozíciót, aminél jelenleg tart. Majd miután ezt a számot megtudta, ha az a terület 8-nál kisebb, akkor egy magas épületet fog oda rakni, egyébként egy kertes házat:

- A magas épület: Jelenleg 2 különböző magas épület típus van, amik között a különbség az jelenleg csak a szín, de a jövőben ez kibővíthető. Egy magas épület 3 különböző prefab-ból áll:
 - Egy darab base: ez kerül legalulra, ezen van egy ajtó, ami miatt el kell fordítani úgy ezt a prefab-et lerakáskor, hogy út felé nézzen.
 - N darab középső rész: a felhasználó meg tudja adni, hogy ebből minimum és maximum hány darab kerülhet a base-re, de az alapértelmezett, az 1 és 100, és a minimum és a maximumot a felhasználó csak ez a két érték közöttire veheti föl. A középső részt nem forgatja el az algoritmus, mert úgy terveztem, hogy ugyanúgy nézzen ki mind a 4 irányba.
 - Egy darab tető: ez kerül rá a legfelső középső részre, és ez sincs elforgatva.
- Kertes ház: jelenleg összesen van 12 különböző, ezek közül véletlenszerűen választ egyet az algoritmus, és azt rakja oda, a jó irányba forgatva.

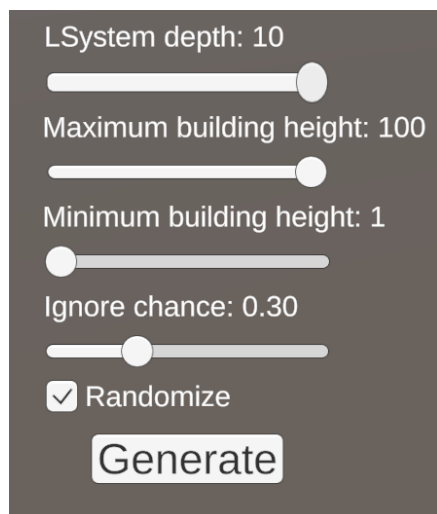
A leírt algoritmus jellegéből adódóan általában úgy fog legenerálódni a város, hogy szinte mindenhol kertesház van, és egyes területeken csak magas épületek helyezkednek el, ezzel olyan hatásúvá téve azt a területet, mintha az lenne a belváros, a többi terület pedig a város kertvárosias része.

4.4. A föld legenerálása

Az utakhoz és az épületekhez képest az földterületek legenerálódása sokkal egyszerűbb. Ennek az algoritmus a abból áll, hogy végigmegy az összes lerakott épületen és az aktuális épületet véve középpontul, 11*11-es területen azokra a helyekre, ahova még nem lett helyezve sem út, sem épület, oda lehelyez egy föld típusú mezőt, ami véletlenszerűen van kiválasztva a megadottak közül (jelenleg 4 fajta lehet), és véletlenszerű irányba elforgatja azt.

5. Program használata

A program indulásakor legenerálódik egy város, amiben a felhasználó körbe tud „repkedni”. A WASD gombokkal tud mozogni, az egérrel tud körbe nézni, amikor éppen mozog valamelyik irányba a felhasználó, akkor a „shift” billentyű lenyomva tartásával tud gyorsabban menni. Az egérrel a jobb kattintás hatására nem tud körbe nézni többet, megjelenik a kurzor a képernyőn, amivel tudja módosítani, az algoritmus egyes értékeit, majd újbóli jobb kattintásra megint fogja tudni mozgatni a kamerát.

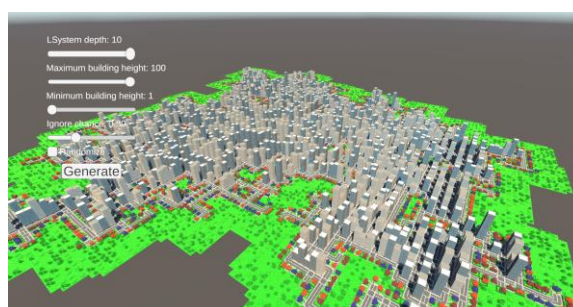
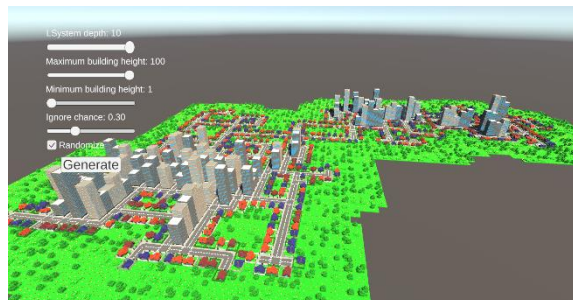


LSystem depth: 10
Maximum building height: 100
Minimum building height: 1
Ignore chance: 0.30
☒ Randomize
Generate

- L-System Depth: Az L-rendszerben algoritmus ennyiszor fut le.
- Maximum building height: Azok az épületek, amiknek a magasságuk változó, maximum ennyi középső részt tartalmaznak.
- Minimum building height: Azok az épületek, amiknek a magasságuk változó, minimum ennyi középső részt tartalmaznak.
- Ignore chance: Ha a randomize ki van pipálva, akkor ennyi eséllyel fog az algoritmus egy levágni ágakat (behelyezni utat mielőtt az algoritmus elérné a kívánt mélységet).
- Randomize: Ha ez be van kapcsolva, akkor az Ignore chance-ben szereplő értékkel az algoritmus levág ágakat.
- Generate: Ha a felhasználó erre a gombra kattint, akkor a jelenlegi város törlődik, és egy új generálódik le.

6. Eredmények

Ebben a fejezetben bemutatok pár legenerált várost, különböző felhasználó által megadott paraméterek alapján:



A bal oldali két legenerált városnál az L-rendszert az algoritmus 7 mélységig járta be, míg a jobb oldaliakat 10 mélységig. A felső két legenerált városban a randomize be volt kapcsolva, míg az alsók esetében nem. Jól látszódik, hogy amikor a randomize ki volt kapcsolva, akkor sokkal nagyobb kiterjedésű város lett legenerálva. Továbbá amikor nagyobb mélységig lett bejárva az algoritmus, akkor is nagyobb kiterjedésű lett a város. A bal felsőnél a város úgy néz ki, mint ami még csak most kezd modernizálódni, pár darab irodaépülettel a központjában. A jobb felső esetben olyan látszatot kelt a legenerált város, mintha két város egybenőt volna, és a két város, ahol találkozik ott kertvárosi rész alakult volna ki. A bal alsó város szerintem a leg kevésbé élethű, mivel nagyrészt kertváros van, és teljesen véletlenszerűen vannak irodaépületek, amik nem igazán illenek így össze. A jobb alsó pedig a legnagyobb, és ebből kifolyólag a legtovább tartott ennek a legenerálása, de ez hasonlít a legjobban szerintem egy valós modern városra.

7. Konklúzió

Az L-rendszer algoritmus használható élethű városok generálására, de csak jól beállított szabályokkal és jó paraméterekkel fog úgy kinézni a város, mint ahogy egy város a valóságban is kinéz, de még akkor sem feltétlenül minden esetben.

8. Továbbfejlesztési lehetőségek

Az algoritmus által legenerált városra rá lehet helyezni különböző dolgokat, amik megváltoztatják, tovább fejlesztik, valóságosabbá teszik a várost:

8.1. Több fajta épület

Először is lehetne belerakni több fajta magas épületet, meg kertes házat. Lehetne ezen a kettőn kívül még épület típus. Lehetne olyan épületeket belerakni, amik nem 1*1-es méretűek olyanokat, amikből van megadva minimum vagy maximum darabszám, amennyi kell legyen a városban.

8.2. Nem játékos által irányított karakterek a városban

A városban lehetnének karakterek, akik mozognak a városban, valamilyen randomizált algoritmus szerint, esetleg legyen mindegyiknek egy úticélja, hogy a térkép egyik pontjáról a másikra akar elérni. Továbbá, ha majd lesz egy játékos is, akkor az is megoldható lehet, hogy a játékos interakcióba lépjen ezekkel a karakterekkel.

8.3. Autók + közlekedési lámpák

Ha már vannak nem a játékos által irányított karakterek, akkor lehetne, hogy ezek autóval is közlekednek a térképen, a közlekedési szabályokat betartva. Esetleg megoldható lenne egy okos városos megvalósítás is, amiben egy algoritmus arra figyel, hogy a városban keresztezések miatti dugókat próbálja meg lecsökkenteni, azzal, hogy változtatja a piros és zöld jelzéseket a közlekedési lámpákon.

8.4. Domborzat

Jelenleg a város csak egy síkban generálódik le, de megvalósítható, hogy esetleg a Perlin-zajt használva legyen domborzat is beépítve a város generálásba.

9. Források

- https://en.wikipedia.org/wiki/L-system#Stochastic_grammars
- https://hu.wikipedia.org/wiki/Lindenmayer_Arisztid
- <http://www.diva-portal.org/smash/get/diva2:1119094/FULLTEXT02.pdf>
- https://cgl.ethz.ch/Downloads/Publications/Papers/2001/p_Par01.pdf