**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. **Jelentések**

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

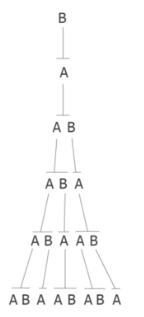
1. **A teljes algoritmus megvalósítása**

A játék elindulása után, vagy a generate gomb megnyomásának hatására indul el a teljes algoritmus, ami több elkülönülő lepésből 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 végződések, 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 mezőt helyez el helyez el, ezzel megszüntetve az esetlegesen keletkező lyukakat a városban, és körbeveszi a város.

* 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.

* + 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 |
| … | … |

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

* + 1. **Az L-rendszer módosítása**
* 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. 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.

* 1. **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 összes úton, és beállítja, hogy hármas 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 hármas 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.
  1. **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 értékként 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 álltalában 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.

1. **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.

* L-System Depth: Az L-rendszerben algoritmus ennyiszer fut le.

Majd ide be kell rakni a végleges UI-ról egy képet

* 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.

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)