

Virtuális színterek procedurális generálása és optimalizálási problémáinak vizsgálata

Bordás Milán

2024. március 18.

Tartalomjegyzék

1. Bevezető	3
2. Színterek típusai	4
3. Algoritmusok	5
3.1. Kétdimenziós színterek algoritmusai	5
3.1.1. Wave function collapse	5
3.1.2. Implementáció	8
4. Útmutató	10
4.1. GC	10
4.2. WFC / WFCSelectorScene	11
4.3. SubwaySurfer	12

Ábrák jegyzéke

2.1. Felülnézetes színtér Heroes III	4
2.2. Oldalnézetes színtér Terraria	4
3.1. Példa a szomszédos cellákra	7
3.2. Példa elakadási helyzetre	7
3.3. Egyszerű input kép	8
3.4. Üres ScriptableObject	8
3.5. Feltöltött ScriptableObject	8
3.6. Horizontális generálás	9
3.7. Vertikális generálás	9
3.8. Entrópiás generálás	9

1. fejezet

Bevezető

A színterek kulcsfontosságú szerepet töltenek be a digitális világban. Egy film, játék, animáció nem lenne teljes egy megfelelően kialakított helyszín, pálya, háttér nélkül. Emiatt szükséges olyan módszereket kialakítani, melyek effektív eszközként szolgálnak ezen elemek létrehozására. A szakdolgozatom eszközként a procedurális generálást taglalja.

A színterek procedurális generálása sok mindent magába foglal. Ha egy könyv leírása alapján gyurmából elkészítünk egy csatateret, lényegében generáltunk procedurálisan egy színteret. Az elkészítendő objektum nem is igazán lényeges, csak az, hogy miként készítettük el. Jelen esetben nem valódi tárgyakat készítünk el, hanem digitális adatokat generáltatunk a számítógéppel, amit megfelelő környezetben színtérként tudunk értelmezni.

Több generálási módszert tárgyal a szakdolgozatom, leginkább videójátékokban használatos pályák, hátterek előállítására. A célterülettől függően két külön esetet vizsgálunk : 2 dimenziós és 3 dimenziós színtereket előállító algoritmusokat.

2. fejezet

Színterek típusai

A színterek egyik legalapvetőbb tulajdonsága a dimenzióinak száma.



2.1. ábra. Felülnézetes színtér
Heroes III



2.2. ábra. Oldalnézetes színtér
Terraria

Példa 2D színterekre

3. fejezet

Algoritmusok

Az algoritmusok egy fontos tulajdonsága, hogy milyen helyzetben használhatóak, milyen problémákra adnak megoldást. A már említett két eset szerencsére nem különül el egymástól annyira, hogy teljesen új megoldási módszereket kelljen kialakítani.

3.1. Kétdimenziós színterek algoritmusai

Ezen algoritmusok egyszerűbbnek tekinthetők a színterek kontextusában.

3.1.1. Wave function collapse

A "WaveFunctionCollapse" nevű algoritmust először githubra publikálta egy felhasználó ¹. Az algoritmus nevét a szerző a "hullámfüggvény összeomlásáról" nevezte el, ami egy kvantummechanikai kifejezés. Magának az algoritmusnak nincs kapcsolata ezzel a jelenséggel, de az ihlet onnan jött.

Az algoritmus eredeti ötlete, hogy egy inputként megadott bitmap pixeljeit felhasználva generálunk egy, az eredetihez lokálisan hasonló általában nagyobb bitmapet. A módszer azonban nem korlátozott csak bitmapekre: egy adott képet is fel lehet fogni egy bitmapként, amit tetszőlegesen darabolhatunk fel, így lényegében bármilyen képre elvégezhetjük a lépéseket. Ez kifejezetten hasznos több területen is. Egy ilyen felhasználási módszer amit a forrás is említ, hogy az algoritmussal csempézni is lehet a teret.

Az algoritmus működése

1. Beolvassuk az input kép összes pixeljét, majd kialakítunk egy paraméter által megadott $N \times M$ -es rácsot, feldarabolva ezzel a képet téglalapokra, hívjuk őket celláknak.
2. A cellákhoz hozzárendeljük a pozíciójukat, majd meghatározunk egy szomszédsági kapcsolatot a többi cellával. A szomszédokat az alábbiként definiáljuk:
 - Egy cella szomszédja egy másik cella, ha az az eredeti input képen a cellától Nyugatra, Keletre, Északra vagy Délre van.

¹<https://github.com/mxgmn/WaveFunctionCollapse>

- A cellákat egy tóruszon képzeljük el. Ez azt jelenti, hogy a képen a balszélső celláknak a szomszédja a jobbszélső vele egy sorban lévő cella is. Persze ez oda-vissza, és fenti-lenti cellákra is igaz. Erre példa a 3.1 ábra.
- Paraméterként megadott igaz/hamis érték szerint egy cella önmagával szomszédos lehet még akkor is, ha az inputon ez nem így van.

Az output cellákat úgy helyezzük el, hogy egy cella csak olyan helyre kerülhet, ahol a körülötte lévő cellák vagy üresek, vagy mindegyike a szomszédja.

3. Kiválasztjuk, hogy milyen cellát szeretnénk letenni a rácsra. Ez a kiválasztás egy súlyozott táblával történik, ahol minden cella súlya, az input képen való előfordulásának száma.
4. A cellákat az alábbi eljárások egyikével letesszük a cellákat egy KXL -es outputra:
 - Horizontálisan / vertikálisan, azaz elindulunk egy soron / oszlopon és folytonosan tesszük le a szabályok szerint a cellákat.
 - Véletlenszerűen, azaz minden cellát egy olyan véletlenszerűen választott helyre teszünk, ahol még szabad a hely
 - Entrópiát számolunk, és mindig a legalacsonyabb entrópiájú helyre tesszük le a cellát. Ezt az entrópiát az üres cellákra számoljuk ki amit a

$$\sum_{i=1}^n P_i * \log 1/P_i$$

képlettel kapunk, ahol P_i az adott helyre letehető i -edik cellának a letehetési valószínűsége. A letehetési valószínűség: az adott cella súlya, osztva az összes cella számával.

Ennek az eljárásnak nagy a költsége, így ennek optimalizálásával később majd foglalkozunk.

5. Addig ismételjük a 3. ponttól az algoritmust, míg le nem fedtuk az outputot, vagy el nem akadtunk.

Az elakadás egy probléma ami akkor jelenik meg, ha olyan üres rácshoz érünk, melynek szomszédjai kizárják egymást. Ekkor az algoritmust újratekzdjük, vagy leállunk. Egy ilyen elakadási helyzetre példa a 3.2 ábra.

1	2	3
4	5	6
7	8	9

3.1. ábra. Példa a szomszédos cellákra
Az 1-es cella szomszádja a 2-es, 3-as, 4-es, 7-es cella.

1	1	1	3
1	2	1	1
2	0	2	1
1	2	1	1

Kiinduló tábla

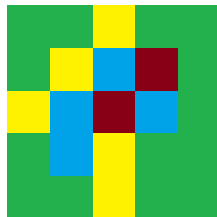
0	X	3	

Elakadás az outputon

3.2. ábra. Példa elakadási helyzetre
A hármas cella mellé csak egyes cella, a nullás cella mellé csak kettes cella kerülhet, így a kettő között nem lehet semmilyen cella.

3.1.2. Implementáció

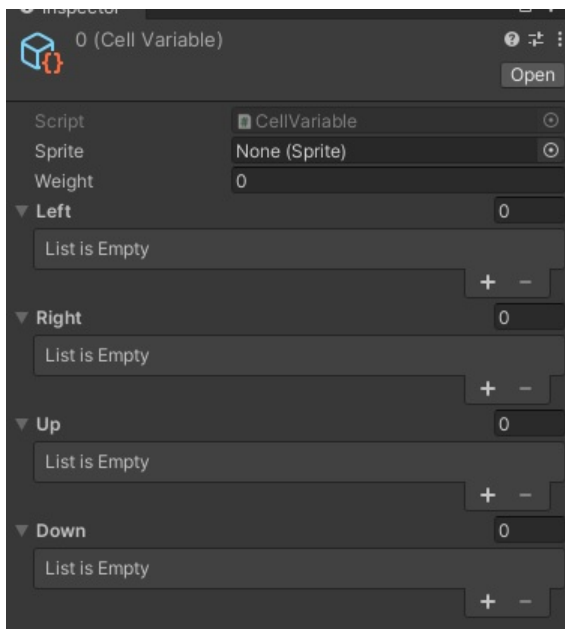
Az implementációhoz először választanunk kell egy képet. Válasszunk egy egyszerű, pár különálló blokkból álló képet például a képen láthatót. Ha nem jól elválasztható blokkokból áll a kép, akkor az algoritmusunkat nem tudjuk használni. Látható, hogy



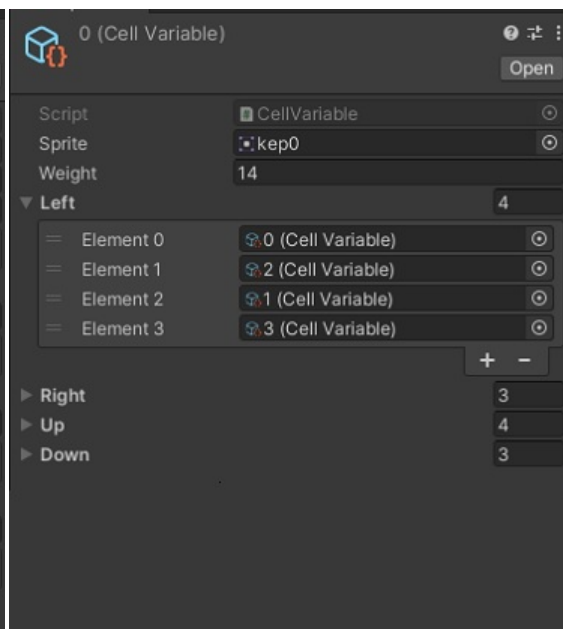
3.3. ábra. Egyszerű input kép

a kép 4 elkülöníthető blokkból áll, zöld, sárga, kék, piros blokkokból. Gondoljunk ezekre a blokkokra, úgy mintha virágok lennének egy mezőn. A zöld blokkok jelölik a füvet, a színesek pedig a virágokat. Az elvárásunk az lehet, hogy egy nagyobb mezőt alkossunk, és a virágok aránya nagyjából megegyező legyen az input képben látható arányokkal.

A következő lépésben feldaraboljuk ezt a képet, és kialakítjuk a cellákat. A feldarabolt cellákat szeretnénk megtekinteni és tetszőlegesen paraméterezni. Ehhez használjuk a Unity egyik felxibilis eszközét, a ScriptableObject-et. Ebben az osztályban tároljuk le a cellának a képét, a súlyát és a szomszédjait. Ebből az objektumból annyit kell automatikusan létrehozni a beolvasónknak, ahány cellára daraboltuk fel az inputot. Miután megtörtént a létrehozás, feltöltjük az objektumokat.



3.4. ábra. Üres ScriptableObject

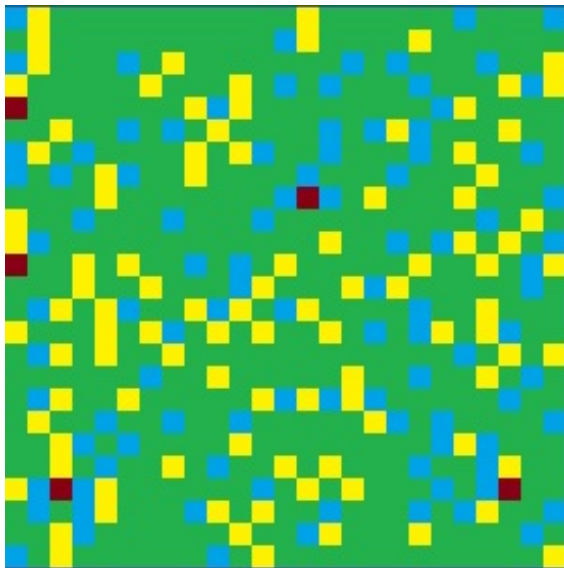


3.5. ábra. Feltöltött ScriptableObject

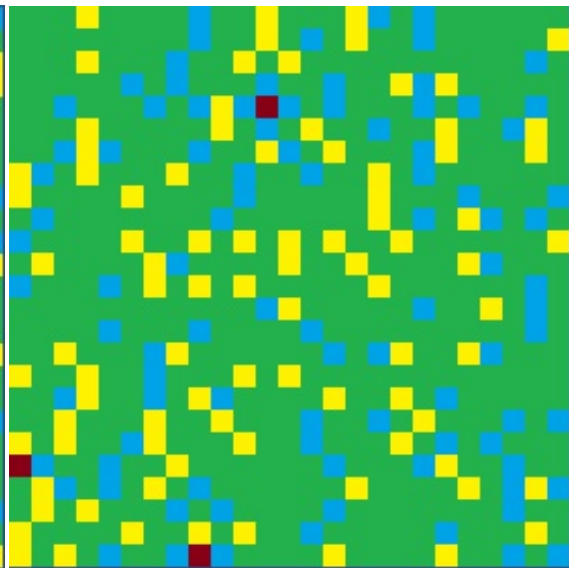
Cella-paraméter leíró objektumok

A feltöltött objektumokat pedig megadjuk az algoritmusnak, ezekből lesznek a cellák.

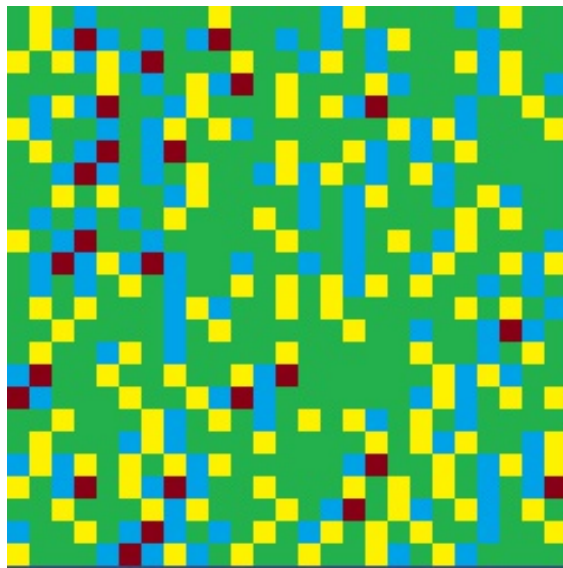
Az algoritmus futása előtt kiválasztjuk a preferált módszert a 4-es lépésben definiáltak közül, illetve hogy mekkora outputot szeretnénk generálni. A példához egy 25×25 nagyságú outputot választottam. A véletlenszerű generálás nem adott eredményt, mert csak elakadást produkált.



3.6. ábra. Horizontális generálás



3.7. ábra. Vertikális generálás



3.8. ábra. Entrópiás generálás

Output generálás különböző módszerekkel

4. fejezet

Útmutató

A program elején egy jól ismert mini-játék fogad az úgynevezett "Dino game". Az implementációm egy jóval egyszerűbb verziója az eredeti játéknak, egy játékos menüként fogható fel.

- Ugrani a "SPACE" billentyűvel lehet
- Piros háromszögek jönnek elő jobbról, ezeknek ha neki ütköziünk akkor a háromszög tetején látható scene-re irányítódunk át.

Fontos: minden scene rendelkezik egy felugró menüvel, ezt az "ESCAPE" billentyűvel érhetjük el. Innen a "Main menu" gombot lenyomva áttérhetünk egy hagyományosabb kezdő menüre.

Az alábbi scene-ek érhetőek el a menüből:

GC

WFC / WFCSelectorScene

SubwaySurfer

4.1. GC

Irányítás:

- Nagyítani-kicsinyíteni a görgővel lehet.
- A kamerát a WASD és a nyilak segítségével mozgathatjuk.

Használat:

- A "Run algorithm" gombbal generálhatunk új világot. Ez paraméterezés nélkül is működik, előre beállított értékekkel.
- World dimensions:
 - Width: a világ négyzetrácsának vízszintes nagysága
 - Height: a világ négyzetrácsának függőleges nagysága
- Max number of stars: megadható hogy maximum hány csillag generálódjon a térben. A végleges száma a csillagoknak ettől eltérő lehet (kevesebb). Maximum

*Width * Height*

darab csillag generálható.

- Max number of black holes: megadható hogy maximum hány fekete lyuk generálódjon a térben. A végleges száma a feketelyukaknak ettől eltérő lehet (kevesebb). Maximum

$$Width * Height - Csillagok$$

feketelyuk generálható.

- Resource density: megadható milyen sűrűen legyenek elhelyezve "relikviák". 0-1 közötti érték, az üresen maradt helyek száma határozza meg a generálást.
- Spacing: a világ négyzetrácsainak nagyságát lehet állítani vele. A generálás egy-egynyi nagyságú négyzetrácsokon történik, aztán kerül sor a transzformációra.
- Position noise: a négyzetrácsok közepétől mekkora sugarú körben vándorolhatnak el az égitestek. A változásra még a "Spacing" paraméter hatása előtt kerül sor.

4.2. WFC / WFCSelectorScene

Használat:

- A program beolvassa a /Szakdolgozat2D_Data/StreamingAssetsPath/input mappában lévő png kiterjesztésű fájlokat, ezen képeket használhatjuk fel a generálásra. Valamely képre kattintva, majd a felugró ablakon a "load" gombot lenyomva haladhatunk tovább.
- Cell can neighbor itself: ezen beállítással módosítható, hogy a képből generált cellák szomszédjai lehetnek-e saját maguknak mindenféleképpen, vagy csak akkor, ha az a referencia képen is látható.
- Continous block picking: ezen beállítással módosítható, hogy a beolvasás során a blokkok takarhatják-e egymást vagy sem. Igaz érték esetén minden NxM-es blokkot beolvas a program, hamis esetén pedig úgy, mintha dominók lennének.
- Block dimensions:
 - Width: a blokkok szélessége.
 - Height: a blokkok magassága.
- Image size: mekkora a referencia képünk nagysága pixelekben mérve.
- Potential block count: ha minden blokk különböző lenne, akkor mennyi blokk képződne az eljárás során.
- Generate: a gomb lenyomásával legenerálódnak a blokkok. A gomb lenyomása után két gomb válik láthatóvá:

- Check blocks: ha kevesebb mint 40 blokk generálódott, akkor megtekinthetőek a gomb lenyomásával. A feulgró ablakban láthatóak a blokkok, illetve alattuk az, hogy mekkora súlya van egy blokknak. Ez a súly változtatható, illetve negatív érték esetén a blokk bellítható "wildcard" blokknak (ilyen esetben a blokk súlya nem változik).
- Next: a következő paraméterezésre léphetünk át.

Next gomb lenyomása után:

- Animate the algorithm: csak cella módban működik. A generálás blokkról-blokkra láthatóvá válik. A generálás során kikapcsolható.
- Cell mode: cella módban, vagy egyszerűsített módban generáljon az algoritmus. A cella mód egyenként fog kép objektumokat generálni, az egyszerűsített mód pedig egy képet generál. Nagy méretű textúra generálás esetén ajánlott az egyszerűsített mód.
- Use local weights: a generálás során lokális vagy globális súlyokat használjon a program.
- Block dimensions: a szélességnek és magasságnak oszthatónak kell lennie a blokk szélességével / magasságával.
 - Width: a generált kép szélessége.
 - Height: a generált kép magassága.
- Image size: mekkora lesz a tényleges képméret, képpontokban megadva.
- Model: kiválasztható melyik bejárési model szerint generáljon a program.
 - HORIZONTAL / VERTICAL: vízszintes / függőleges bejárési irány.
 - RANDOM: véletlenszerű bejárési irány. Magasabb az esélye hogy a generálás elakad, ha sok a megszorító szomszédsági reláció.
 - SHANNON_MODEL: Shannon-entrópiát használó bejárás. Lassabb mint a többi.
- Run algorithm: a képgenerálás lefuttatása. A futtatás után a paraméterek újra változtathatóak. A blokkok változtatása csak a menübe kilépve érhető el újra.

4.3. SubwaySurfer

Ez a scene csak a "runner" scene-ből érhető el.

- Mozogni a nyilakkal, illetve a WASD billentyűkkel lehet. Ütközés esetén az "ESCAPE" lenyomása után a "runner" menüpontból érhető el a scene.