# Pályageneráló algoritmusok

Ebben a fejezetben három darab generálási módszert fogok bemutatni, amelyek a Perlin zaj, a celluláris automata, valamint a Random Walk algoritmusok. Mindhárom algoritmust a Unity keretrendszerben implementáltam és vizsgáltam meg, hogy melyik módszer illeszkedne legjobban a játékomhoz.

## A térképgenerálás megtervezése

Egy olyan dungeon-stílusú térképet szeretnék létrehozni, amelynek a külső szélei mindig falak, amelyek nem engedik leesni a játékost a pályáról, így a játék határai egyértelműen megmaradnak, valamint lehetővé teszi a játékosok számára, hogy szabadon navigáljanak anélkül, hogy áthatolhatatlan akadályokba ütköznének, például olyan falakba, amelyek megakadályozzák őket a tárgyak begyűjtésében. Tehát olyan pályageneráló algoritmus illeszkedne a játékomhoz, amely nem generál elzárt tereket, szobákat.

A térképgenerálás logikája, valamint a menete a következőképpen történik:

Bináris használatával különböztetjük meg, hogy mi a tile és mi nem az. Az 1 bekapcsolva, a 0 pedig kikapcsolva. Az összes térképünket egy 2D-s egész számtömbben tároljuk, amelyet minden egyes funkció végén (kivéve, amikor renderelünk) visszaküldünk a felhasználónak.

Az új térkép generálása előtt a térképen a meglévő tile-ok törlődnek. Ez biztosítja, hogy az új térkép generálása üres vászonnal kezdődjön, megakadályozva az új adatok átfedését vagy összeolvadását a régi tile-okkal. Ezután egy metódus segítségével legenerálunk egy N\*N méretű tömböt, amiben az értékek a 0 vagy 1 értéket vehetik fel. Miután a tömb legenerálása befejeződött, meghívjuk a procedurális mapgeneráló algoritmusunkat, majd egy renderelő függvényt, amely az általunk kiválasztott tiletípust felfesti a térképre. A térképgenerálás folyamatábrája a 3.1.1. ábrán megtekinthető.

A képen szöveg, képernyőkép, diagram, sor látható

Automatikusan generált leírás

1.2.1.1. ábra. A térképgenerálás folyamatábrája

## A térképgenerálás logikáját megvalósító algoritmusok bemutatása

Ebben a fejezetben a térképek generálásáért felelős metódusokat fogom részletesen bemutatni.

### A GenerateArray() metódus

A HardMapGenerator osztály GenerateArray() függvényének célja egy 2D-s int[,] típusú tömb létrehozása és inicializálása, amely a térképrácsot képviseli a procedurális generáláshoz. Ez a tömb szolgál alapként a procedurális mapgeneráló algoritmusok alkalmazásához a dungeon alaprajzának generálásához.

**Paraméterek:**

* int width, int height : Ezek a paraméterek határozzák meg a térkép méreteit, specifikusan a szélességét és a magasságát.
* bool empty: Ez a bool típusú változó határozza meg, hogy hogyan inicializáljuk a tömböt. Ha az értéke igaz, akkor üresként inicializáljuk, ha false, akkor pedig teliként.

Az int[,] map = new int[width,height] sorral egy új 2 dimenziós tömböt inicializálunk. A tömb minden eleme egy egész számot reprezentálhat.

A beágyazott for ciklusok a tömb minden egyes elemén végigmennek, az x a szélességen, az y pedig a magasságon iterál.A map.GetUpperBound(0) és a map.GetUpperBound(1) a tömb dimenzióinak felső határainak megadására szolgál. A GetUpperBound(0) az első dimenzió (szélesség) maximális indexét adja vissza, a GetUpperBound(1) pedig a második dimenzió (magasság) maximális indexét.

A GetUpperBound használata általában a tömb megadott dimenziójának utolsó érvényes indexét adja vissza. Egy szélesség és magasság dimenziójú tömb esetében a GetUpperBound(0) valójában a width - 1, a GetUpperBound(1) pedig a height - 1 értéket adná vissza, mivel a tömbindexek 0-nál kezdődnek.

A belső cikluson belül az üres paraméter feltételes ellenőrzése dönti el, hogy a tömb helyét 0-val (ami üres vagy nyitott helyet jelez) vagy 1-gyel (ami kitöltött vagy blokkolt helyet jelez) töltse-e ki.

**if (empty) { map[x, y] = 0; } else { map[x, y] = 1; }**: Ez gyakorlatilag beállítja a térkép minden egyes cellájának kezdeti állapotát. Egy üres térkép (empty = true) azt jelenti, hogy minden cellát 0-ra inicializálunk, egy nem üres térkép (empty = false) pedig azt, hogy minden cellát 1-re állítunk.

**return map;:** Miután a tömb teljesen feltöltődött, visszakerül a hívó számára. Ez a tömb most a kezdeti rácsállapotként szolgál a további feldolgozáshoz, például a RandomWalk algoritmus alkalmazásához.

A GenerateArray() metódus az 1.2.1.1. ábrán látható.

A képen szöveg, képernyőkép, szoftver, képernyő látható

Automatikusan generált leírás

1.2.1.1. ábra. A GenerateArray() metódus

### A RenderMap() metódus

A HardMapGenerator osztály RenderMap() függvénye felelős a generált térkép vizuális megjelenítéséért egy Unity Tilemap-en egy adott TileBase segítségével. Ez a függvény a numerikus térképadatokat (egy 2D-s egész számtömbben tárolva) a játéktérképen lévő tényleges tile-okká alakítja át.

**Paraméterek:**

* int[,] map: Egy két dimenziós tömb, amely a dungeon alaprajzát reprezentálja, ahol az egyes cellák értéke határozza meg, hogy üres vagy tele van-e az adott cella.
* Tilemap tilemap: Ez a Unity Tilemap, amelyre a tile-ok fognak felrajzolódni.
* TileBase tile: Ez a térkép kitöltött területeinek vizuális ábrázolására szolgáló tile. (Esetemben RuleTile)

A tilemap.ClearAllTiles() paranccsal, mielőtt az új térképet lerenderelnénk, töröljük az összes meglévő tile-t a térképről. Ez kulcsfontosságú annak biztosítása érdekében, hogy a korábbi tile-ok ne maradjanak láthatóak. Ez a metódus is egymásba ágyazott for ciklusokat használ ahhoz, hogy bejárja a map tömb minden elemét. A ciklusok a tömb minden egyes tengelyén végigmennek a map.GetUpperBound(0) segítségével a szélesség (x) és a map.GetUpperBound(1) segítségével a magasság (y) esetében.

A cikluson belül a függvény minden egyes koordinátánál (x, y) ellenőrzi az értéket. Az if(map[x,y] == 1) feltétel ellenőrzi, hogy az aktuális koordinátáknál lévő cella tele van-e ( 1 az érték). Ha tele van, akkor a Tilemap megfelelő pozíciójára egy új tile kerül. A tilemap.SetTile(new Vector3Int(x,y,0),tile) parancs egy tile-t helyez el a Tilemap (x,y) pozíciójában.

A RenderMap() metódus az 1.2.2.1. ábrán látható.

A képen szöveg, képernyőkép, szoftver, Betűtípus látható

Automatikusan generált leírás

1.2.2.1. ábra. A RenderMap() metódus

## A Perlin-zaj (Perlin Noise)

A Perlin-zaj egy algoritmus, amelyet Ken Perlin hozott létre az 1980-as évek elején, és széles körben használják a játékfejlesztésben bármilyen hullámszerű anyag vagy textúra létrehozásához. Például a Perlin-zajt használhatjuk procedurális domborzati alakzatok (Minecraft szerű domborzati térkép hozható létre a Perlin-zaj algoritmus segítségével), tűzeffektek, víz és felhők létrehozásához. Ezek a hatások főleg a második és harmadik dimenzióban tükrözik a Perlin-zajt, de kiterjeszhető a negyedik dimenzióra is. Ezen kívül az algoritmus használható még az 1 dimenziós térben is, mint például egy „side-scroller” terep létrehozásához, vagy kézzel írt vonalak illúziójának megteremtésére.

Sőt mi több, ha az algoritmust a 2. vagy a 3. dimenzióra is kiterjesztjük, valamint az extra dimenziókra úgy tekintünk, mint az időre, akkor meg is tudjuk a kreált alakzatokat animálni. Az alábbiakban néhány képet láthatunk a különböző méretű zajokról és néhány felhasználási módjukról futás közben:

|  |  |  |
| --- | --- | --- |
| Zaj dimenziószáma | A nyers zaj (szürkeárnyalatos) | Felhasználási mód |
| 1 | A képen 1D perlin-zaj látható | A képen kör, vázlat, minta látható  Automatikusan generált leírás  A zaj offsetként való használata kézzel írt vonalak létrehozásához. |
| 2 | A képen szürke, képernyőkép, természet látható  Automatikusan generált leírás | A képen minta, csomagolópapír, térkép, Színesség látható  Automatikusan generált leírás  A zajt szigetek létrehozásához is lehet használni |
| 3 | A képen vázlat, fekete-fehér, minta, táj látható  Automatikusan generált leírás | A képen térkép látható  Automatikusan generált leírás  Egy módosított Perlin-zaj implementációval dombok, völgyek és barlangok hozhatók létre |

Amint láthatjuk, a Perlin-zaj számos természetben előforduló jelenségre alkalmazható.

A Perlin-zaj gradiens zajgenerálási technikát alkalmaz, ami a pontok közötti természetesebb és simább átmenetet eredményez. Ez a megközelítés élethűbbnek tűnő tájképet hoz létre. Az algoritmus egy rácshálós keretrendszerben működik, ahol a rácsháló minden egyes metszéspontjához egy gradiensvektor tartozik. Ezek a vektorok döntő fontosságúak a zaj mintázatának és irányítottságának kialakításában.

A Perlin-zaj egyik fő jellemzője a rácspontok közötti interpoláció alkalmazása, ami hozzájárul a jellegzetes simasághoz. Ez a sima átmenet éles ellentétben áll a teljesen véletlenszerű zajgenerálásra jellemző hirtelen változásokkal. A Perlin-zajt eredetileg 3D-s grafikához fejlesztették ki, de a 2D-s alkalmazásokban is széles körben használják, többek között a videojátékok terepgenerálásában és a procedurális textúrák létrehozásában.

A generált minták összetettségének fokozása érdekében az algoritmus gyakran alkalmaz rétegezési technikát, amely több "oktávnyi" zajt tartalmaz. Minden egyes oktáv külön frekvenciával és amplitúdóval működik, és amikor ezeket a rétegeket kombinálják, bonyolultabb és változatosabb mintákat hoznak létre. Az algoritmus állítható paramétereket kínál, mint például a frekvencia, az amplitúdó és a perzisztencia, ami lehetővé teszi a generált zaj megjelenésének részletes szabályozását, és a terep vagy a textúra testre szabott szimulációját.

A játékokban és a számítógépes grafikában való alkalmazásán túl a Perlin-zaj elterjedt más területeken is, mint például tudományos szimulációk készítése, ahol olyan természeti jelenségeket modellez, mint a felhőképződmények, vagy egy táj jellegzetességei.

### A Perlin-zaj algoritmus implementálása és vizsgálata

A PerlinNoiseDungeon() metódus a Perlin-zaj segítségével módosítja a rácsos térképet, hogy simább átmeneteket hozzon létre a kitöltött és üres terek között.

**Paraméterek:**

* int[,] map : A módosítandó kezdeti térképtömb.
* float modifier: Egy skálázási tényező, amely a Perlin zajfüggvény frekvenciáját állítja be.

A függvény a megadott térképmező minden egyes celláján végigmegy a beágyazott for ciklusok segítségével. Ha az aktuális cella a térkép szélén van (x == 0, y == 0, vagy az x vagy y utolsó indexein), a függvény a cella értékét 1-re állítja, így biztosítva, hogy az egy fal maradjon.

A Mathf.PerlinNoise(x\*modifier, y\*modifier) függvényhívás egy Perlin-zajértéket generál (0 és 1 között) a modifier által beállított koordinátákhoz. A modifier a bemeneti koordináták skálázására hat, ami befolyásolja a zajfrekvenciát. A Mathf.RoundToInt() metódus az eredményül kapott Perlin zaj lebegő értékét a legközelebbi egész számra kerekíti annak eldöntésére, hogy a cella fal vagy üres tér legyen.

A függvény visszaadja a módosított térképtömböt az új terepjellemzőkkel együtt, készen állva a renderelésre.

A PerlinNoiseDungeon() metódus az 1.3.1.1. ábrán megtekinthető.

A képen szöveg, képernyőkép, szoftver, képernyő látható

Automatikusan generált leírás

1.3.1.1. ábra. A perlin zajt megvalósító metódus

A továbbiakban a Perlin-zaj által generált térképek vizsgálatáról lesz szó.

## RandomWalkCave()

A HardMapGenerator osztály RandomWalkCave() metódusa a térképrács módosítására szolgál a Random Walk algoritmuson alapuló procedurális generálási technikával. Ez a függvény egy barlangszerű struktúrát váj ki egy adott kétdimenziós tömbben.

**Inicializáció:**

Random seed inicializálása: A seed.GetHashCode() paranccsal egy random magot inicializálunk. Ez a mag biztosítja, hogy a generált véletlenszámok reprodukálhatóak legyenek, ha ugyan azt a magot használjuk, ami hasznos a hibakeresésnél vagy bizonyos térképek újragenerálásánál

**Kiinduló pozíció kiválasztása:** A funkció véletlenszerűen választ ki egy kezdő pozíciót (floorX, floorY) a térkép határain belül, de nem a széleken, hogy biztosítsa, hogy legyen hely a barlang bővítésére. Innen fog kezdődni a véletlen bolyongás.

**reqFloorAmount:** A reqFloorAmount a térképen lévő összes cellaszám százalékaként (requiredFloorPercent) kerül kiszámításra. Ez a változó határozza meg, hogy hány cellát kell átalakítani teliből (value = 1) üressé (value = 0) a barlanggenerálás befejezéséhez.

**floorCount inicializáció:** A floorCount nulláról indul, és minden alkalommal növekszik, amikor egy cellát teliből üressé alakítunk.

A while ciklus addig megy, amíg a floorCount értéke el nem éri a reqFloorAmount értékét. A while cikluson belül egy switch utasítást alkalmazok az irányok kezelésére. Az út generálásának a logikája a következő:

* A switch utasítás minden egyes esete egy iránynak felel meg.
* Mielőtt az algoritmus bármilyen irányba mozogna, biztosítjuk, hogy a lépés nem halad ki a térkép határain kívülre.
* 0.eset: Növeljük a floorY értékét, ha a felfelé mozgás a határokon belül marad.
* 1.eset: Csökkentjük a floorY értékét, ha a lefelé mozgás a határokon belül marad.
* 2.eset: Növeljük a floorX értékét, ha a jobbra mozgás a határokon belül marad.
* 3.eset: Csökkentjük a floorX értékét, ha a balra mozgás a határokon belül marad.

Minden esetben megnézzük, hogy a jelenlegi pozíció teli-e. Ha az, akkor átkonvertáljuk üresre, és a procedúra addig folytatódik, amíg el nem érjük a kívánt cellák számát.