# **Algoritmus elmagyarázása**

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.

## **GenerateArray()**

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 RandomWalk algoritmus 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.

## **RenderMap()**

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.

## **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.