**Témalabor**

**Unity város és gyalogosok generálása**

**Készítették:**

**György Márk Attila (ZCVPZT),**

**Li Jiaxiang (SISU6U)**

**Az alábbiakban:**

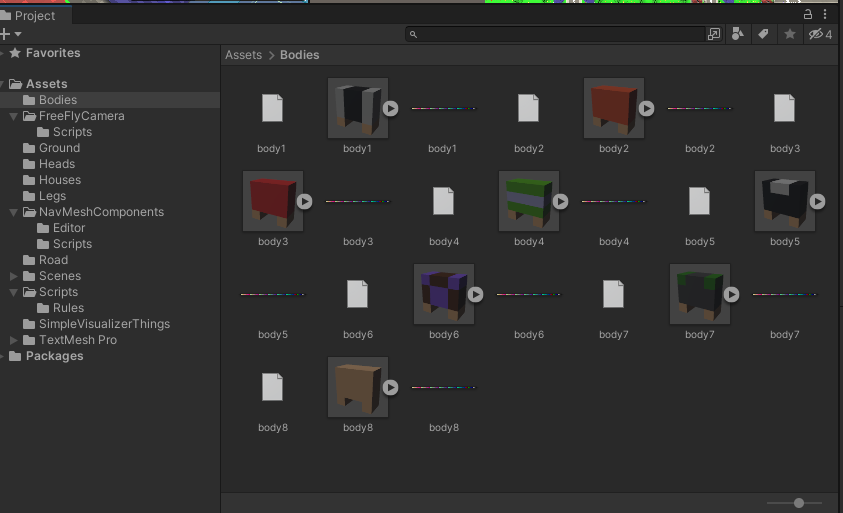
**Gyalogos generálás és mozgatás részfeladat:**

**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, gyalogosok mozogjanak benne. 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. Ezen túl a gyalogosok mennyisége és sebessége is szabályozható szintén a UI-al.

1. **Különböző kinézetű emberek**
   1. **Prefabek**



A cél különböző kinézetű emberek generálása volt. Ennek megvalósítására a Magicavoxel nevezetű programot használtam. Ebben könnyen létrehozhatók és importálhatóak objektumok melyeket Unityben egyszerű használni. Készítettem 8 féle fej, törzs és lábat amely összesen 8x8x8 = 512 fajta ember generálását jelenti. Ezeket Unity-ba importálva prefabként használhatóak.

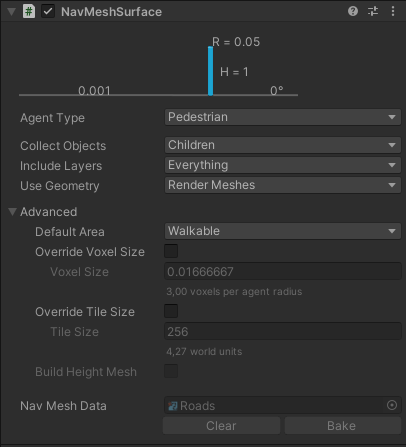
* 1. **Emberek összerakása**

A testrészek tárolására egy külön osztályt hoztam létre a BodyParts nevű osztályt, amely egy GameObjectekből álló tömböt tartalmaz. Egy függvénye van ezen túl, amely ebből a listából egy véletlenszerű elemmel tér vissza. A PedestrianSpawner osztály felelős a gyalogosok generálásáért. Tartalmaz egy BodyParts-okat tartalmazó tömböt. Ebben a tömbben három elem van, az egyik a fejeket tartalmazó tömb, a másik a törzseket a harmadik pedig a lábakat tartalmazza. Egy gyalogos létrehozásakor mindhárom tömbből kiválaszt egy véletlenszerű elemet, majd ezeket létrehozza és egy közös szülő objektumhoz csatolja, így lesznek egy objektumként kezelhetőek. A generált gyalogosokat egy lista tárolja, hogy a város újra generálásakor törölni lehessen őket és újra létrehozni.

1. **NavMeshComponents**

Unity-be van beépített navigációs rendszer mely a UnityEngine.AI-ban található meg. A gyalogosok mozgásának hálózatához ezt választottam kiindulási alapnak. A beépített Navigation Systemnél eggyel bővebb és fejlettebb eszközöket tartalmazó NavMeshComponents package-ből használtam főként elemeket. Ez a projektet az AI Navigation package részeként fejlesztik és GitHub-on elérhető.

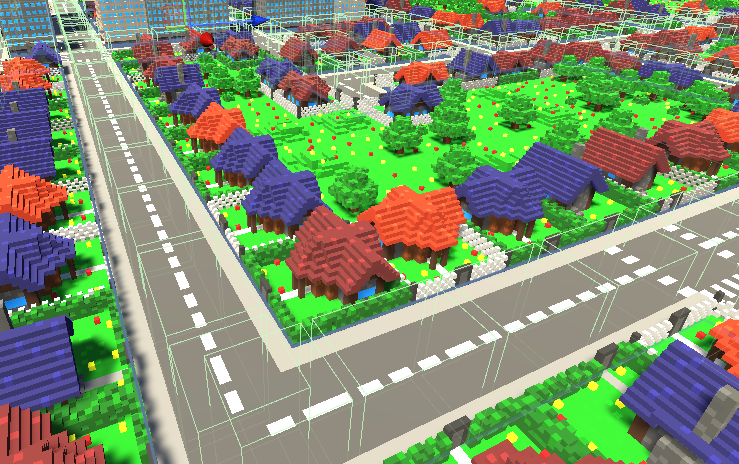
**3.1 NavMeshSurface**

****

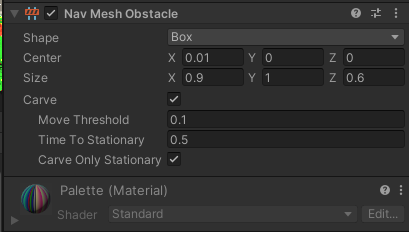
A NavMesh lényege különböző típusú felületek létrehozása. Unity-ben beépítetten léteznek a Walkable, Not Walkable és Jump felületek. Az alapvető ötlet az utak számára való Walkable felület létrehozása volt, amelyen majd a gyalogosok tudnak közlekedni. Ilyen felületeket Bake-elni tudnuk meglévő objektumpokra. Az itt felmerülő probléma, miszerint a program futásidőben generálja a várost így nem léteznek előre az objektumok amire le lehetne generálni a felületeket. Erre adott megoldást a NavMeshComponents NavMeshSurface osztálya amellyel futásidőben tudunk NavMesh felületeket létrehozni és kezelni. A NavMeshBaker osztály szolgál az utakra való NavMesh generálására. A program indításakor az algoritmus alapján legenrálódik a város. Az út objektumokat egy Roads nevű szülő objektum gyerekeiként hozza létre. A Roads objektum a feladata, hogy tartalmazza az összes utat, így egységesen kezelhetőek lesznek. Ezen túl egy NavMeshSurface komponenst és a NavMeshBaker scirptet tartalmaza. A NavmeshSurface komponens miatt önmagára azaz az összes útra egyszerre lesz generálható a járható felület. A NavMeshBaker egy listát tartalmaz a felületekről amelyekre generálni kell a NavMesh-t. Jelen esetben ez egy elemet tartalmaz, a Roads objektum saját magát helyezi el benne. A Navmesh végigmegy a listán és mindegyik felületre meghívja a BuildNavMesh függvényt. Ezáltal létrejött a felület ami csak az utakból áll és járható.

**3.2 NavMeshObstacles**

Az utaknak nem a teljes felülete lehet járható, hanem csak a járda vagy zebra része. Ennek megoldására több fajta technika is létezik, az egyik NavMeshModifierVolume, amellyel felületek típusán tudunk változtatni. Megoldás lett volna ilyen objektumok ráhelyezése az utakra és Not Walkable area-ra állítani őket.



Az általam használt technika a NavMeshObstacle. Ezek lehelyezése tulajdonképpen akadály, amelyeken a gyalogosok nem fognak tudni átmenni és az útvonaltervezéskor ezeket kikerülik. Ilyen objektumokkal lettek lefedve a különböző út típusok. Meg lehet adni nekik formát ( Téglatest, kapszula ) , méretet és az őt tartalmazó objektumhoz képest lévő relatív elhelyezkedését. Ezeknek a paramétereknek egyedi beállításával az összes út típusra létre lett hozva a megfelelő lefedés, ami csak a járdát és zebrát nem fedi. A kanyarodó útaknál kettő ilyen objektum is használva lett, hiszen egy téglatest nem tudja lefedni a kanyart, amihez komplexebb forma szükséges.   
Ezeket az objektumokat az utak létrehozása után rakja rá az attachObstacles függvény, amely az alapján ismeri fel az utakat, hogy hány vagy éppen milyen irányba vannak szomszédjai az adott útszakasznak. A kanyarnál mind a négy esetet külön kellett kezelni, hiszen ilyenkor az Obstacle objektumokat forgatni kellett, nem úgy mint a négyes kereszteződésnél ahol csak középre kellett helyezni egy téglatestet.



A NavMeshObstacle-nek ezen túl van még egy tulajdonsága a carve. Ennek a tulajdonságnak a használatával el lehet érni, hogy míg az obstacle ott van, addig kivájja a vele érintkező objektumot. Ennek használata azért fontos, mivel az útvonalkeresés során így a gyalogosok nem fognak nekimenni az Obstacle-nek és csúszni a felületükön, hanem kikerülni fogják azt. Ez minden Obstacle objektumon true-ra van állítva.

**3.3 NavMeshAgents**

Az elkészült NavMesh felületen NavMeshAgentek tudnak mozogni. Létre lehet hozni több fajta Agentet és nekik be lehet állítani, hogy milyen típusú felületen tudjanak mozogni. Jelen esetben a Walkable felületen mozognak a Pedestrian típusú agentek. Mindegyik gyalogoshoz, azaz a testrészek szülő objektumához a létrehozásokor egy ilyen Agent komponens is hozzáadódik. Az Agentnek van sebessége amellyel tud mozogni. Ez a tulajdonság randomizálva van a felhasználó által. Ezen túl az enable-tulajdonsága engedélyezi a mozgását.

1. **A mozgás megvalósítása**

**4.1 PedestiranMovement**

Ez az osztály felelős a gyalogosok (Agentek) mozgásáért. Minden gyalogos (Agent) tartalmazza ezt a scriptet. Az Agentek kezdeti pozícióját random kapja, a NavMesh.CalculateTriangulation függvény segítségével. Ez a függvény a jelenlegi NavMesh-t osztja fel polygonokra. Ezekből random kiválasztva egy csúcsot majd a SamplePosition függvényt használva, amely az ahhoz legközelebbi pontot adja a NavMesh-en, lesz kiválasztva a kezdő pozíció ahova az Agent kerül. (A Warp függvénny alkalmazható erre, hogy az Agentet a megfelelő helyre tegye). A PedestrianMovement-ben ezen túl a Start és Update függvények vannak definiálva. A Start-ban a szülő objektumtól kéri le az Agent komponenst, valamint kiszámolja a triangulation-t. Az Updateben ami frame-enként hívódik nézi, meg hogy az Agent-nek van-e aktív útvonala, ha nincs akkor hasonlóan a kezdő pozíció generáláshoz, generál egy másik pontot amit beállítva célként elindul az Agent. Ezáltal folyamatosan mozogni fognak, amint elérik a céljukat egy új cél felé fognak elindulni. Ha valamiért nem sikerülne elérni a célt, az autoRepath engedélyezésével újra kalkulálja az útvonalat.

**4.2 A\* algoritmus**

A Unity beépített útvonalkeresője az A\* algoritmust használja. A kezdő és végpontot közötti polygon felosztásokat nézi végig. A NavMesh tartalmazza ezeknek a polygonoknak az információit, egyes polygonok szomszédjait. Ez a felosztás hasznos, hiszen a polygonok belsejében biztosan nincsen akadály. Az A\* algoritmus súlyozott gráfokat használ. Célja egy kezdőpont és végpont közötti legkisebb költségű út megtalálása. Fa gráfot tárol az indulási ponttól kezdve és minden lépésben egyel tovább halad. A következő node kiválasztása az eddigi és a végponthoz eljutásig szükséges költség összege alapján adódik.



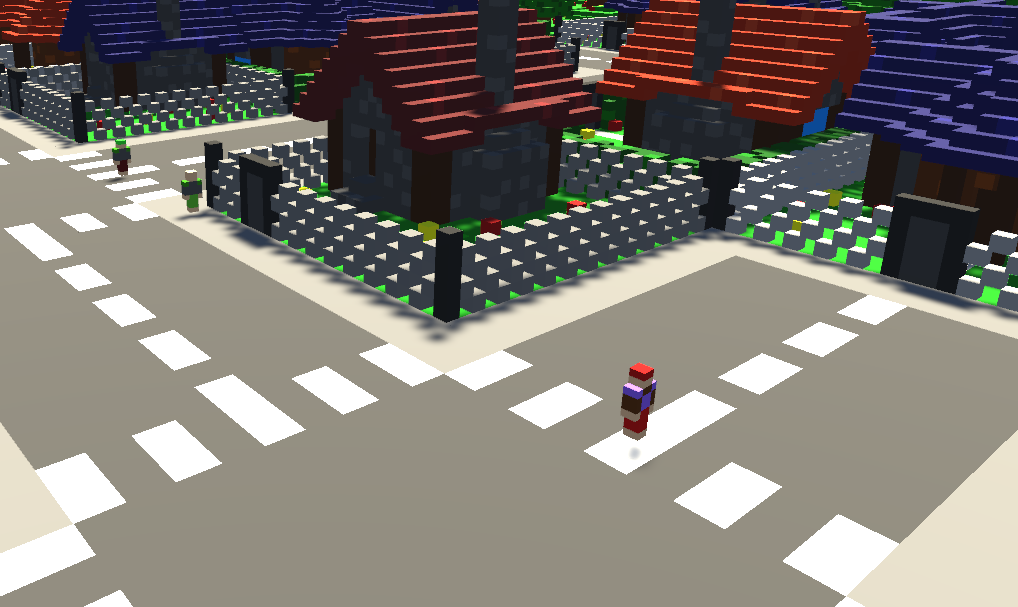
Jelen képletben az „n” a következő node, a g(n) az eddig megtett út költsége a kezdő ponttól, a h(n) pedig a becsült érték n-től a cél node-ig. Minden lépésben a legkisebb f(x) kikerül a sorból, a szomszédjainak értékei ezáltal frissülnek majd ez addig folytatódik, míg a kivett pont egy cél node nem lesz. Így megkapva a legkisebb költségű utat. A node-ok megtartják az előző node-ot így visszavezethetőek lesznek, tehát egy útvonalat fog találni. Unityben a NavMesh polygonjain végigfuttatva az algoritmust megtalálja a kedvező útvonalat.

1. **UI**

****

A Number of Pedestrians alapértéke 50, ez határozza meg, hogy hány gyalogos fog generálódni. Ezt a mércét 0-75-ig lehet állítani. A Max Speed pedig a gyalogosok maximális sebességét határozza meg. Ezt 0,1 és 1 között lehet állítani. A gyalogosok sebessége azonban nem mindegyiküknek ugyanaz, a porgramm véletlenszerűen generálja az értékét 0.1 és a beállított érték között.

**6. Eredmények**

****

A fenti képen pár gyalogos látható, ahogyan a járdán és zebrán közlekednek. Látható ezen túl a különböző kinézetük is.



Egy generált városnak a NavMesh felületei láthatóak (Walkable - kék), ezen túl az utakon elhelyezett Obstacle-ből kialakult hálózat. Sikeresen le lett fedve az utaknál a nem járható rész és jól látszik kékkel az összes járda és zebra, amik járhatóak.

**7.Konklúzió**

Unity-ben a beépített útvonaltervező A\* algoritmust használva hatékony az útvonalkeresés. A NavMeshComponents package is jól kezelhető azonban a CalculateTriangulation költséges művelet, így sok gyalogosnál több időt vesz igénybe az útvonal kiszámítása és pontok keresése.

**8.Fejlesztési lehetőségek**

* Animáció elkészítése a gyalogosoknak, azaz komplexebb objektumokból való összeállítása. (külön karok, lábak).
* Kiegészítők bevezetése, cipők, sapkák stb. A városban lámpák létrehozása, amelyek befolyásolnák a közlekedést.
* A felhasználónak lehetőséget nyújtani egy gyalogos irányítására.
* Egy – egy gyalogos eltűntetése és futás közben másik generálása.
* Más típusú utak bevezetése

**9.Források**

* [https://en.wikipedia.org/wiki/A\*\_search\_algorithm#Description](https://en.wikipedia.org/wiki/A*_search_algorithm#Description)
* <https://docs.unity3d.com/Manual/nav-NavigationSystem.html>
* <https://learn.unity.com/tutorial/runtime-navmesh-generation>
* <https://docs.unity3d.com/560/Documentation/Manual/class-NavMeshSurface.html>
* <https://docs.unity3d.com/ScriptReference/AI.NavMeshAgent.html>