NYILATKOZAT A SZAKDOLGOZAT EREDETISÉGÉRŐL

Alulírott Kovács Bence a **BMSZC Verebély László Technikum 54 213 05 OKJ Szoftverfejlesztői képzésében** részt vevő hallgatója büntetőjogi felelősségem tudatában nyilatkozom és aláírásommal igazolom, hogy

a Cesleh

című szakdolgozat saját, önálló munkám, és abban betartottam az iskola által előírt, a szakdolgozat készítésére vonatkozó szabályokat.

Tudomásul veszem, hogy a szakdolgozatban plágiumnak számít:

- szó szerinti idézet közlése idézőjel és hivatkozás nélkül,
- tartalmi idézet hivatkozás megjelölése nélkül,
- más publikált gondolatainak saját gondolatként való feltüntetése.

E nyilatkozat aláírásával tudomásul veszem továbbá, hogy plágium esetén szakdolgozatom visszautasításra kerül.

Budapest, 2021. április 15.

Kovács Bence

Hallgató aláírása

Budapesti Műszaki Szakképzési Centrum
Verebély László Technikum
54 213 05 Szoftverfejlesztő szakképesítés

Ceslen

Készítette:
Kovács Bence
2020-2021

Tartalom

Köszönetnyilvánítás

Ezúton szeretnék köszönetet mondati Modeller barátomnak Dobai Leventének, aki biztosította számomra a játékban szereplő összes modell-t. Nélküle nem jöhetett volna létre ez a munka és ezért hálás vagyok.

Külön köszönettel tartozom továbbá Kulcsár Dénesnek amiért lektorálta az itt összefoglaltakat. A diszlexiám miatt ez a szöveg nem lehetne hibátlan nélküle.

Továbbá köszönettel tartozom osztályfőnökömnek Juhász Zoltánnak, hogy legjobb tudása szerint egyengette a szakadozgatom irányát.

Nem utolsó sorban köszönet mindenkinek, aki támogatott a munkában bármilyen módon, családtagok, barátok köszönöm mindannyiótoknak, hogy végig mellettem álltatok.

Köszönöm mindenkinek!

Bevezető

Szakdolgozaton nagyon sokat gondolkodtam, hogy mit kellene választanom és végül úgy döntöttem, hogy valami olyat szeretnék csinálni amilyet eddig még nem. Mivel ezelőtt volt már tapasztalatom Játék fejlesztésben csak sokkal elavultabb fejlesztői környezetben, így gondoltam, mint modernebb térként szolgálhat a Unity. Mivel hatékonyabb előbb végcélt kitalálni és csak utána környezetet választani, így egy kicsit megnehezítettem a dolgom, de végül arra a döntésre jutottam, hogy a Catan nevű táblajátékot szeretném számítógépes térbe implementálni. Az ötletem már az elejétől fogva nem 100%-os másolás volt, és nem is szerettem volna a dobó kockás változatnál maradni így ezt valami olyan módszerrel kell helyettesítenem, amivel dinamikát kap a játék és fent tudja tartani a játékos érdeklődését.

A Catan egy hexagonokból álló pályán lehet játszani, amit mindig a játékosok állítanak fel játék előtt, ilyenkor törekednek arra, hogy ne legyen sok nyersanyag, ugyan olyan típusú mező egymás mellett. A játék során ezekkel a mezőkkel való kapcsolat határozza meg a kinyert anyagokat. A játékot tovább lehet bonyolítani további kiegészítőivel mely további lehetőségeket ad hozzá. A digitalizált változatomba a játék tengeri kiegészítőjét adtam hozzá, ami miatt egy központi sziget helyett több kisebb és nagyobb lehet.

Az elkészülő játékban is, mint az eredetiben utakat/hajókat lehet majd helyezni a hexagonok éleire és a sarkaikra pedig épületeket. Az építmények határozzák meg hogy mekkora mennyiséget termelnek ki, az utak pedig ezeket az erőforrásokat kötik össze. Minden játékos kétszer helyezhet le a pályára kezdéskor egy falut és egy utat. Mivel a játék még fejlesztés alatt áll, így a nyersanyagok megszerzése a jelenlegi verzióban csak illusztrálásképp vannak bent, hogy az összekötött városok és falvak megfelelően működnek. Minden mezőhöz csatlakoztatott falu 1 db egységet termel ki, a város pedig 2 darabot. Ha a termelő bábuk össze vannak kötve a termelésük összeadóik az érintett mezőkhöz. A játék célja, hogy minél hamarabb a legnagyobb hálózattal rendelkezzünk, mivel a termelők a sebességükkel egyenlő pontszámot érnek. A táblajátékban eredetileg 12 pontot kell elérni a nyeréshez, viszont a játékban én ezt állíthatóra szeretném programozni játék előtt.

Az eredeti játékból egyelőre még csak ennyi került implementálásra, de a továbbiakban szeretném majd még a Fejlesztő kártyákat és a Dominancia lapokat is behelyezni, további saját ötleteket is belehelyezni.

Fejlesztői dokumentáció

A játék elkészítése során alkalmazott programok:

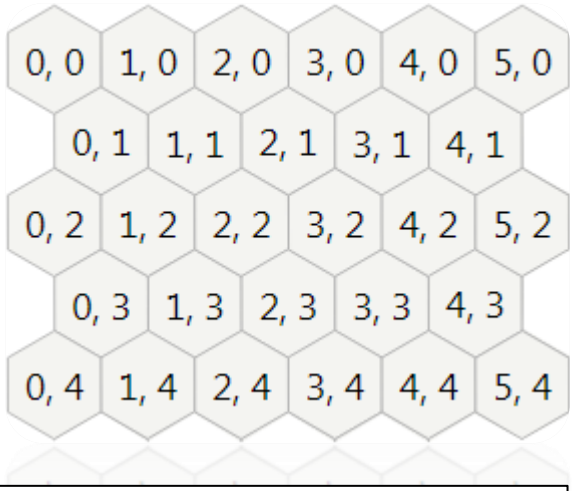
Unity, Visual Studio 2017, GitHub Desktop, Blender, Gimp 2.10, Software Ideas Modeler

Programozási nyelv: C#

Pálya generálás

MapGenerator.cs

[#0 - CreateGrid] Pálya generálása



Befolyásoló változok:

```
int xMax; //Magasság
int yMax; //Szélesség
```

Mivel a pálya hexagonokból áll így alapból a pályán darabjainak az eltárolását is végig kellett gondolnom. Végül a képen látható módot választottam, melynek a generálást a következő kód végezte, és dinamikusan igazodik a megadott szélességhez és magassághoz:

```
for (int x = 0; x < xMax; x++)
{
    //Egy allista mely a sorokat tartalmazza
    List<Cor> sublist = new List<Cor>();
    for (int y = 0; y < yMax; y++)
    {
        Cor temp = null;
        /* attól függően, hogy páros vagy páratlan sor
         * eltolja a megfelelő mennyiséggel */
        if (x % 2 == 0)
            temp =
            new Cor(
                    Instantiate(
                        HexagonObject,
                        new Vector3(x * 8.83f, 0, y * 10.2f),
                        HexagonObject.transform.rotation
                    ),
                    new Vector2Int(x, y)
                );
        else
            temp =
            new Cor(
                    Instantiate(
                        HexagonObject,
                        new Vector3(x * 8.83f, 0, y * 10.2f + 5.1f),
                        HexagonObject.transform.rotation
                    ),
                    new Vector2Int(x, y)
                );

        temp.GameObject.transform.parent = transform.GetChild(0);
        sublist.Add(temp);
    }
    GridElementList.Add(sublist);
}
```

[#1 - GenerateIlands] Szigetek kigenerálása

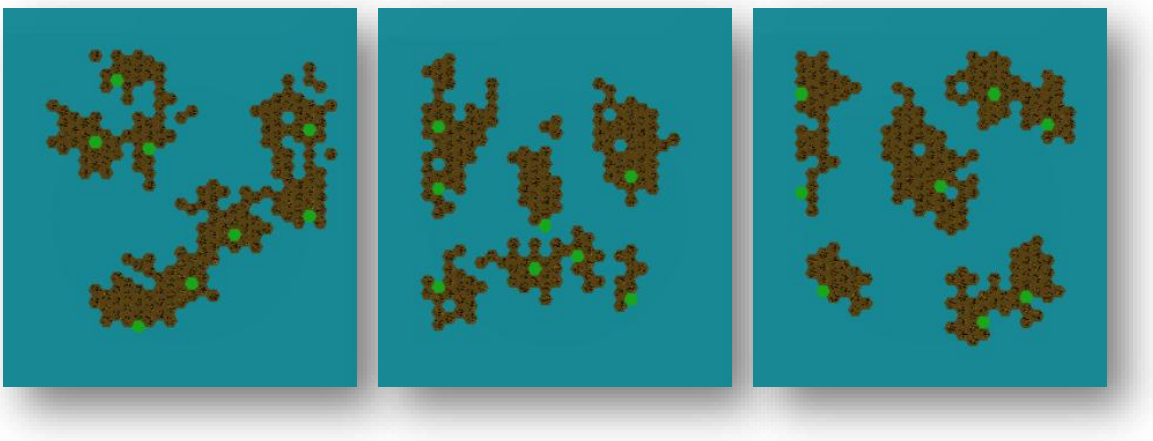
Befolyásoló változok:

```
/* A pálya szélétől számítva befelé egy százalékos
 * érték amin belül lehet LandCore */
float MarginPresent;
/* Azt szabályozza, hogy milyen valószínűséggel alakulhat ki
 * Land a MarginPresent értékeken kivül eső kordinátákon */
float MarginSensivity;
/* Legalább egy tenger elemnek kell lennie a lehendő LandCore
 * és a meglévő összes Land elem között, de ez nem garantálja,
 * hogy külön álló szigetek lesznek */
bool DoIlandOnlyOnSea;
/* Azt határozza meg hogy hány LandCore legyen */
int IlandCount;
/* Sziget átlag méretét határozza meg */
float IlandAvgSize;
/* Minden egyes újonnan generált elem után ennyivel szorozza meg
 * a IlandAvgSize és ezzel csökentve */
float GrowMultiple;

/* Debug Változok melyek segíték a vizualizálást */
bool SlowGenerateIlandHexagonLayer;
bool SlowGenerateIlandHexagon;
bool SlowGenerateIland;
```

A tábla elkészült, a következő kódrész arra szolgál, hogy szigeteket generáljon különböző faktorok alapján. A lehetséges szigetek számától egészen a szigetek átlag méretén át a pálya szélétől való táválságát is meg lehet határozni.

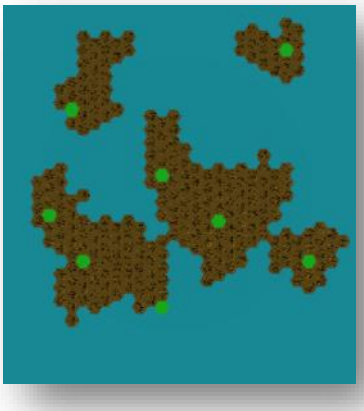
Először a `MarginPresent` és a `MarginSensivity` kiválaszt egy véletlenszerű pontot melyet LandCore-nak nevezünk. A LandCore csak akkor lesz tovább engedve, ha megfelel a `DoIlandOnlyOnSea` és nem lépi túl a 30 próbálkozást a megfelelő pont kigenerálásához. Ez akkor történhet, ha nincs már szabad tengeri rész, ilyenkor automatikusan abbahagyja a generálást.

Amennyiben lett egy valid LandCore-unk megkezdődhet a sziget terjedelmének generálása. Először kiválasztja a vizsgált elem körüli tengerként megjelölt „pálya elemeket” (innentől HexField). Elsőnek mindig a LandCore, a vizsgált HexField és azután pedig mindig a sziget tengert érintő HexField-jei. A kiválasztott HexField-eket véletlenszerű sorrendben átalakítjuk egy százalékos esély alapján melyeket a LandCore-tól való távolság és a pálya széléhez való távolság határoz meg.



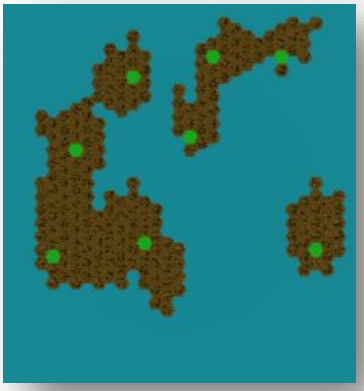
```
var trys = 0;
for (int i = 0; i < IlandCount; i++)
{
    //Egy Random kiválasztott Hexagon a pályán a Margin szabálya szerint
    var IstandCore = GetHexBy2dV(GetRandomHexOnGrid(xMax, yMax, MarginPresent), GridElementList);
    //Egy biztonsági ellenőrzés, ami amiatt kell, ha végtelen ciklusba esne a DoIlandOnlyOnSea
miatt
    if (trys > 30)
    {
        Debug.LogError("I CAN'T GENERATE THE REQUERD LAND COUNT!");
        break;
    }
    if (DoIlandOnlyOnSea &&
        GetHexAround(
         IstandCore.Cordinate,
         GridElementList
        ).Where(x => x.HexField.HexType != HexField.hexType.Sea).Count() != 0)
    {
        trys++;
        i--;
        continue;
    }
    trys = 0;
    MakePreLand(IstandCore, HexField.hexType.Feild); //A kiválasztott Hexagon átalakítom Land-é
    var runbreak = 0; //Számláló, hogy hányat lépett el a Core-tól
    /* Helyi változóba helyezem mivel később módosítani fogom,
     * viszont az eredetire is szükség lesz még */
    var ilandAvgSize = IlandAvgSize;
    //Jelenleg vizsgát Hexagon réteg a Core kőről
    var ilandLayer = GetHexAround(IstandCore.Cordinate, GridElementList);
    /* Addig változtatja a Hexagon-okat ameddig nem marad átrakható terület a szabályok szerint
     * vagy el nem éri a maximum lépést a Core-től */
    while (ilandLayer.Count > 0 && runbreak != IlandAvgSize)
    {
        var iLandSurround = new List<Cor>(); //A következő Hexagon réteget tárolja el
        ilandLayer.Shuffle();//Összekeveri a listát, hogy növelje a randomizálást
        //A Generált rész körüli Hexagon rész tesztelése (legújabb layer)
        for (int ILLC = 0; ILLC < ilandLayer.Count - 1; ILLC++)
        {
            /* Csak a NoNLand területekből ilandAvgSize valószínűsége + a környező Landek növelik a
             * valószínűséget újabb Land kialakulásnak */
            if (ilandLayer[ILLC].Update != 1 &&
                    ilandAvgSize +
                    (GetHexAround(ilandLayer[ILLC].Cordinate, GridElementList)
                     .Where(x => x.Update == 1).Count() / 10) > UnityEngine.Random.Range(0f, 1f) &&
                    MarginMultiple(
                     ilandLayer[ILLC].Cordinate.x, ilandLayer[ILLC].Cordinate.y,
                     xMax, yMax,
                     MarginPresent, MarginSensivity) < UnityEngine.Random.Range(0f, 1f)
                )
            {
                //A kiválasztott Hexagon átalakítom Land-é;
                MakePreLand(ilandLayer[ILLC], HexField.hexType.Wood);
                //Az új Land kőrül a Hexagonokat hozzáadjuk a következő réteghez
              iLandSurround.AddRange(GetHexAround(ilandLayer[ILLC].Cordinate, GridElementList));
                if (SlowGenerateIlandHexagon)
                    yield return null;
            }
        }
        ilandLayer.Clear(); //A jelenleg vizsgált réteget kiürítjük
        //Hozzáadjuk a jelenleg vizsgált listához a következő réteget
        ilandLayer.AddRange(
                iLandSurround.Where(
                        x => x.Update != 1 &&
                        ilandAvgSize > UnityEngine.Random.Range(0f, 1f)).ToList());
        //csökkentjük a Land kialakulásának valószínűséget a GrowMultiple változóval
        ilandAvgSize *= GrowMultiple;
        runbreak++; //Növeljük a Core-tól megtett lépések számát
        if (SlowGenerateIlandHexagonLayer)
            yield return null;
    }
    if (SlowGenerateIland)
        yield return null;
}
```

[#2 - RemoveLakes] Tavak eltüntetése



Befolyásoló változok:

```
/* A lyukakat és öblöket távolítja el, amitől
 * egybefüggőbb lesz a térkép */
bool RemoveLakes;
/* Hány környező Land-nek kell lennie ahhoz,
 * hogy a vizsgált Lake-et Land-re
 * váltakoztassa */
int LakeSensivity;
/* Hányszor nézze át a Land környezetét Lake
 * után keresve */
int LakesRecheck;
```



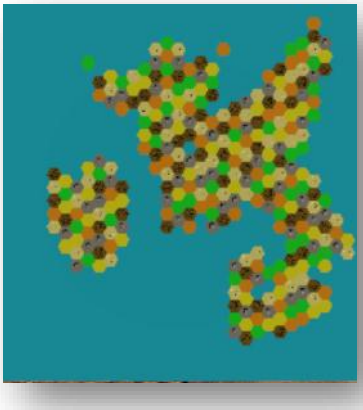
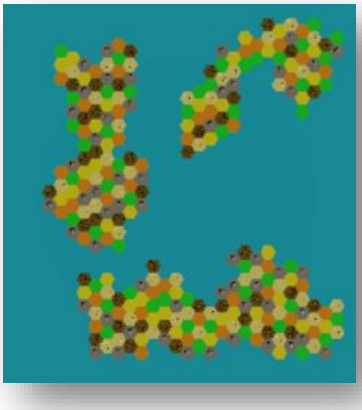
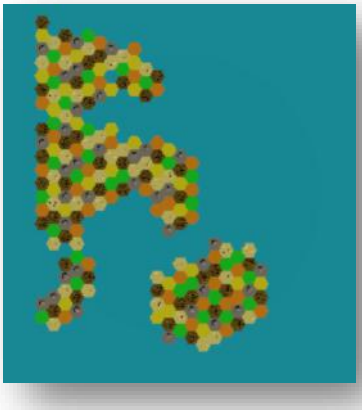
A szigetek létrejöttek, viszont eléggé lyukasok és nagyon tajgás lett. A következő kódrész kikeresi azokat a tenger HexField-eket melyek közvetlen Land mellett vannak és eltárolja őket. A létrejött listát összekeveri, majd bejárja és ha a vizsgálat pillanatában `LakeSensivity`-nek megfelelő mennyiségben vagy több Land van körülötte akkor átalakítja azt is Land-é. Mivel az adott idő pillanatban számít, hogy mi van környezetében ezért fontos, hogy előtte összekeverjük a listát, máskülönben lépcsőzetesen építené a pályát és elvesztené a randomitását. Miután bejárta a listát újra bejárja a pályát keresve tengerparti HexField-eket és megismétli a folyamatot `LakesRecheck` mennyiségszer. Ez az egész folyamat csak abban az esetben fut le ha `RemoveLakes` aktív.

```
if (RemoveLakes)
{
    var IslandBeach = new List<Cor>();
    for (int i = 0; i < LakesRecheck; i++)
    {
        //Bejárja az egész pályát és olyan elemeket keres, ami közvetlen Land mellett lévő Sea-t
        for (int x = 0; x < xMax; x++)
            for (int y = 0; y < yMax; y++)
                if (GridElementList[x][y].Update == 1)
                    foreach (var item in GetHexAround(GridElementList[x][y].Cordinate,
                                                      GridElementList).Where(item => item.Update == 0).ToList()
                        if (!IslandBeach.Contains(item)) //Elkerüli a duplikációt
                            IslandBeach.Add(item);
        /* Összekeveri a találatokat, hogy amikor bejárja a
         * listát ne szabályszerű legyen a kirajzolás ezzel
         * is növelve a pálya randomítását */
        IslandBeach.Shuffle();
        foreach (var item in IslandBeach)
            /* Az éppen vizsgált Sea körül a Land szám <= mit LakeSensivity
             * akkor az adott Sea-ből Land lesz */
            if (GetHexAround(item.Cordinate, GridElementList)
                .Where(x => x == null || x.Update == 0).Count() <= LakeSensivity)
            {
                MakePreLand(item, HexField.hexType.Wood);
                if (SlowGenerateLakeRemove)
                    yield return null;
            }
        if (SlowGenerateLakeRemoveLayer)
            yield return null;
    }
}
```

[#3 - FieldType] Kitermelhetőség megadása

Befolyásoló változok: nincs

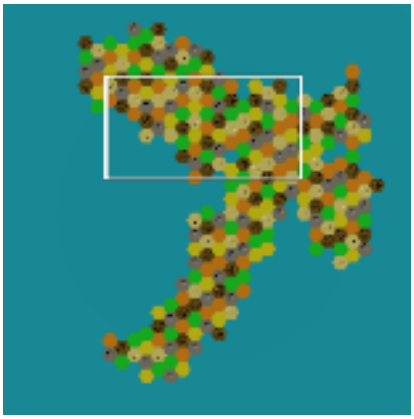
Most, hogy meg van hogyan fog kinézni a tenger és a szárazföld elosztása, meg kell adnunk, hogy milyen anyagokat lehessen kinyerni az adott HexField-ekből. A `GenerateType(List<Cor> HexAround)` elméletileg nem szabadna csak olyan típust visszaadni amilyen még nincsen a környezetében, ezzel elkerülve a monopol helyeket. Viszont valamilyen oknál fogva ezt a részét nem tudtam működésre bírni, és mivel magát a függvényt 3x újra írtam, különböző módszereket használva úgy gondolom, hogy az a függvény lehet a hibás, ami a környező elemeket szedi össze. Ennek a javításával az-az egy gond, hogy már sokszor felhasználva a generálásban ez előtt és így lehet, hogy teljesen megváltoztatná annak menetét (továbbá nem találtam benne a hibát). Mindenesetre eddig még nem volt példa arra, hogy 3-mas gócpontot generáljon csak 2-est.

```
//Azt egészet vissza alakítja vízzé, hogy ne befolyásolja a type-ok kiosztását
for (int x = 0; x < xMax; x++)
    for (int y = 0; y < yMax; y++)
        GridElementList[x][y].HexField.UpdateHexType(HexField.hexType.Sea);
yield return null; //KELL MERT KÜLÖNBEN BUGGOS
for (int x = 0; x < xMax; x++)
{
    for (int y = 0; y < yMax; y++)
    {
        var item = GridElementList[x][y];
        if (item.Update == 1)
        {
            var t = GenerateType(GetHexAround(item.Cordinate, GridElementList));
            item.HexField.UpdateHexType(t);
            item.Update = 2;
            //yield return null;
        }
    }
}

private static HexField.hexType GenerateType(List<Cor> HexAround)
{
    HexField.hexType re;
    do
    {
        re = (HexField.hexType)(RND.Next((Enum.GetValues(typeof(HexField.hexType)).Length-1))+1);
    } while (HexAround.Where(x => x.HexField.HexType == re).Count() > 0);
    return re;
}
```

[#4 - MiniMap] Térkép

Befolyásoló változok: nincs



A kész kigenerált pályáról készül egy orthographic kép felülről, ami a térképként fog szolgálni majd áthelyezi a kamera metszet helyét a pályáról a pálya fölé, ahol egy kamerához rögzített sprite-ról készít felvételt, és azt vetíti majd a térkép fölé, ezzel jelezve, hogy a pálya mely részét nézzük épp.

```
MiniMapCamera.enabled = true;
yield return null;
MiniMapCamera.enabled = false;

MiniMapCamera.nearClipPlane = 201;
MiniMapCamera.farClipPlane = 205;
MiniMapCamera.targetTexture = MiniMapOverLay;
MiniMapCamera.enabled = true;
```

[#5 - CameraBorder] Kamera mozgási tere

Befolyásoló változok: nincs

Annak érdekében, hogy a felhasználó ne tudja kivinni a pályáról a kamerát, betöltés előtt megnézi a legszélső gameobject-eket és azoknak a széléhez képest fogja engedi, hogy meddig mozgathatja a játékos a nézetét.

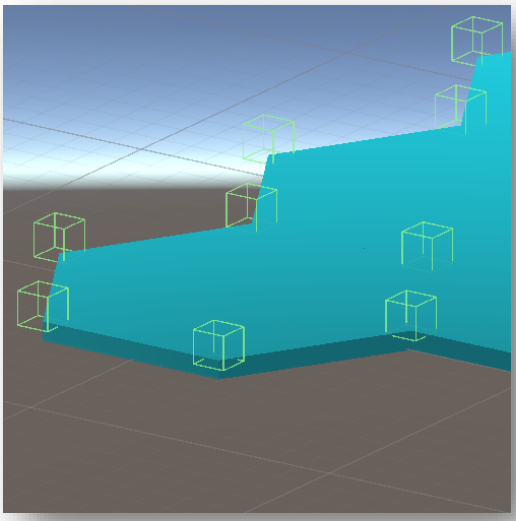
```
for (int x = 0; x < xMax; x++)
    for (int y = 0; y < yMax; y++)
    {
        var SMC = MainCamera.GetComponent<SideMoveCamera>();
        var ElementPos = GridElementList[x][y].GameObject.transform.position;
        if (SMC.MaxDown > ElementPos.z)
            SMC.MaxDown = (int)ElementPos.z;
        if (SMC.MaxUp < ElementPos.z)
            SMC.MaxUp = (int)ElementPos.z;
        if (SMC.MaxLeft > ElementPos.x)
            SMC.MaxLeft = (int)ElementPos.x;
        if (SMC.MaxRight < ElementPos.x)
            SMC.MaxRight = (int)ElementPos.x;
    }
MainCamera.GetComponent<SideMoveCamera>().Offset();
transform.GetComponent<MapVisibility>().enabled = true;
MainCamera.GetComponent<SideMoveCamera>().enabled = true;
PlayersList.SetActive(true);
```

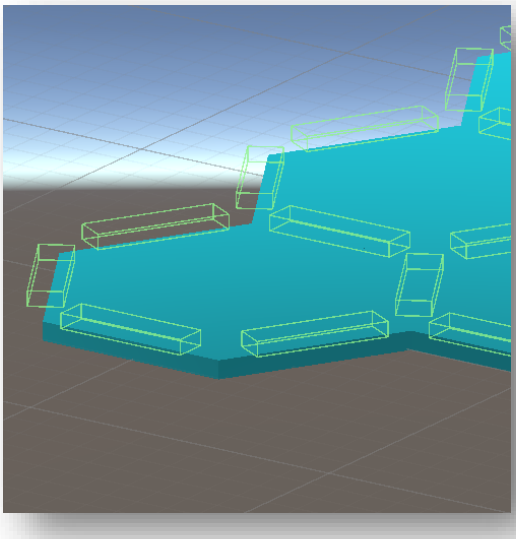
[#6 - Triggers] Bábu érzékelők

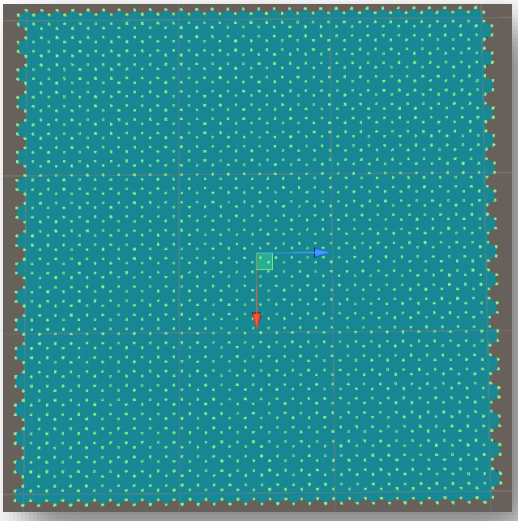
Befolyásoló változok: nincs

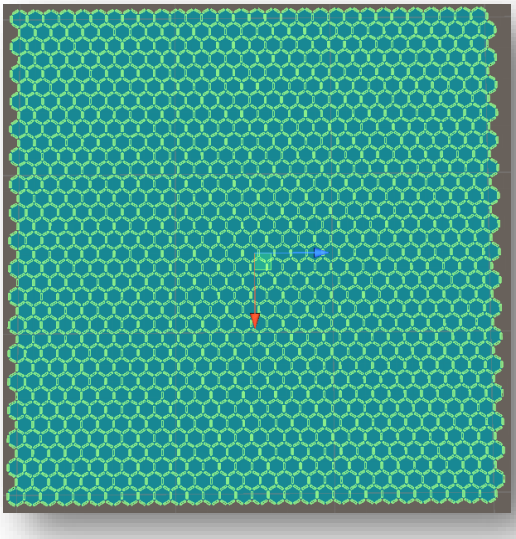
Ez a felhasználó számára nem látható változtatás melynek a lényege, hogy az összes olyan helyre, ahova le kell tudnia helyezni egy bábut oda helyez egy aurát, amit később tesztelni fog a játék. A sarkokon lévő érzékelőket (innentől Trigger) úgy helyezi le, hogy bejárja az összes HexField-et és a közepétől számítva egy vektorral eltolódik egészen a sarkáig, majd elforog 60 fokkal és megismétli annyiszor, hogy az összes sarokra lehelyezzen egyet. A későbbiekben ezzel csak az a gond, hogy 1 sarokra akár 3 db Triggert is lehelyezhet. Ezt a problémát azzal oldottam meg, hogy lehelyezés után várok egy képkockát és ekkor meghívódik bennük az ütközés (innentől Collision) és azonosító alapján eldöntik, hogy ki az újabb, és átmásolja az adatokat magába, majd a régebbi törli magát. Ha ez a folyamat végbemegy, utána az élekre is lehelyezi a Triggereket, úgy hogy a meglévő sarkok között félúton kijelöl egy pontot majd létrehozza az új Triggert és a nézetét az egyik sarok Trigger felé fordítja így egy vonalban lesz az alatta lévő éllel.

A Triggerek csak akkor aktívak amikor a felhasználó le akar helyezni egy bábut és csak azok a Triggerek amik a játékos látóterében vannak.





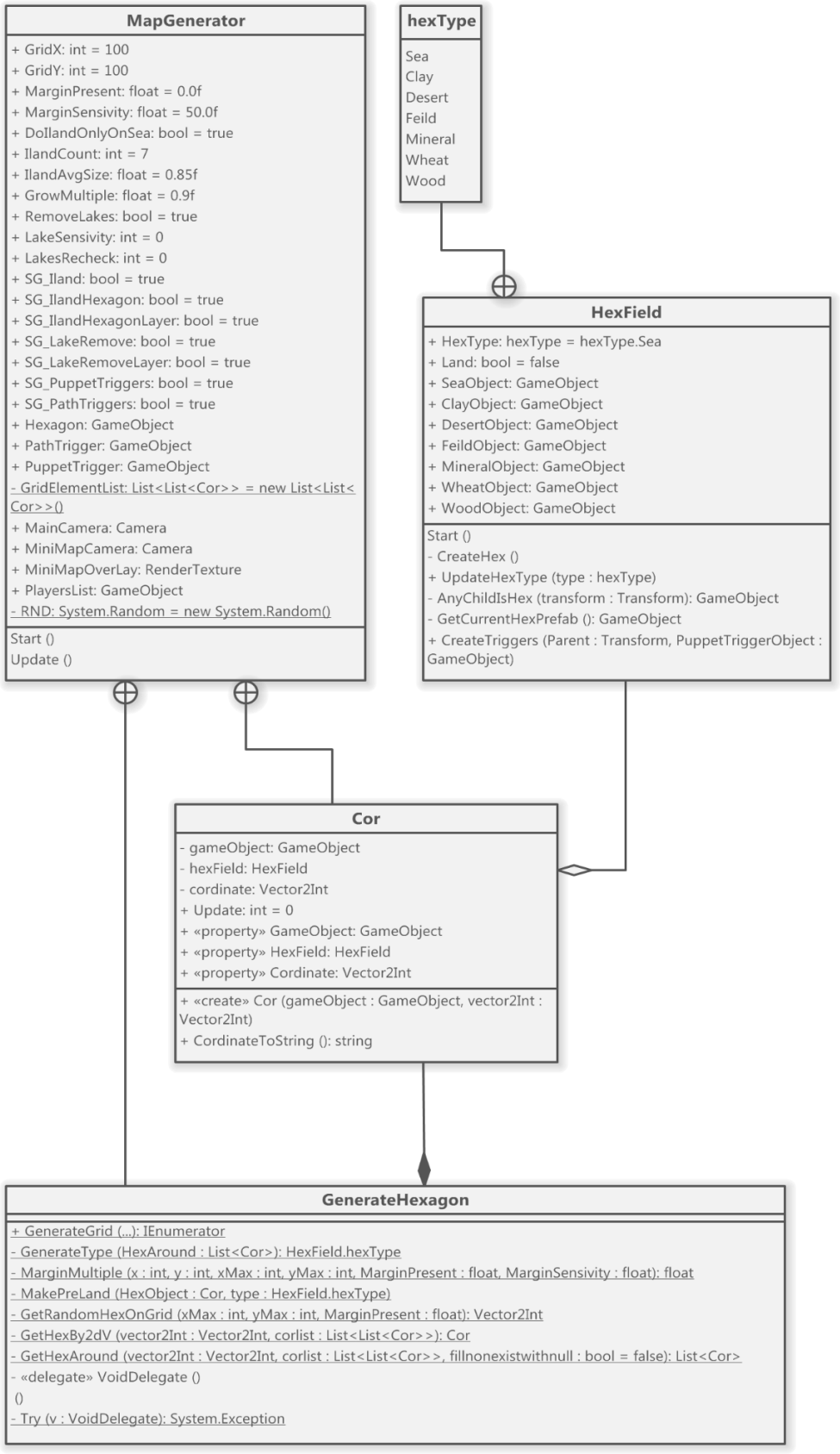




```
//Kigenerálja az össze bábú lehetséges lehelyezési pontját
for (int x = 0; x < xMax; x++)
    for (int y = 0; y < yMax; y++)
    {
        GridElementList[x][y]
         .HexField.CreateTriggers(transform.GetChild(1), PuppetTriggerObject);
        if (SlowGeneratePuppetTriggers)
            yield return null;
    }
/* Fontos mert ha nincs itt nem hívodik meg még a collision és összebuggolhat
 * a SlowGeneratePathTriggers és amiatt is, hogy még nem lettek a felesleges
 * PuppetTriggerek eltüntetve */
yield return null; var Triggers = GameObject.FindGameObjectsWithTag("PuppetTrigger");
//bejárja a meglévő bábú lehelyezési pontokat
foreach (GameObject tr1 in Triggers)
{
    foreach (GameObject tr2 in Triggers)
    {
        //azonos bábú helyeket át ugorja, hogy önmagával ne legyen párba
        if (tr1 == tr2 ||
         (tr1.transform.position - tr2.transform.position).sqrMagnitude > 35.6f)
            continue;

        //A két vizsgált bábú hely közé lehelyez egy út helyet
        GameObject pathtrigger =
            Instantiate(PathTriggerObject,
                        new Vector3(
                        (tr1.transform.position.x + tr2.transform.position.x) / 2, 1,
                        (tr1.transform.position.z + tr2.transform.position.z) / 2)
                        , new Quaternion());
        //elforgatja, hogy jó irányba nézzen
        pathtrigger.transform.LookAt(tr1.transform.position);
        //egy kicsit lejjebb helyezi, hogy ne lebegjen
        pathtrigger.transform.position += new Vector3(0.0f, -0.1f, 0.0f);
        PathTrigger pt = pathtrigger.GetComponent<PathTrigger>();
        //Hozzáadja a 2 bábú helyet, ami által létre lett hozva
        pt.AddPuppet(new GameObject[] { tr1,tr2 });
        //hozzáadja azokat a mezőket, amiből nyersanyagot kaphat
        pt.AddField(tr1.GetComponent<PuppetTrigger>().ConnectedField);
        pt.AddField(tr2.GetComponent<PuppetTrigger>().ConnectedField);
        //Áthelyezi a megfelelő gameobject alá
        pathtrigger.transform.parent = transform.GetChild(2);
        if (SlowGeneratePathTriggers)
            yield return null;
    }
}
yield return null;
var Paths = GameObject.FindGameObjectsWithTag("PathTrigger");
//Bejárjuk az össze kigenerált út helyet
foreach (var item in Paths)
{
    //kikérjük az úthelyhez tartozó bábú helyét
    var pt = item.GetComponent<PathTrigger>().ConnectedPuppetTrigger;
    for (int i = 0; i < pt.Length; i++) //majd bejárjuk
        //és hozzá adjuk ezekhez a bábú helyekhez a jelenlegi út helyeket
        pt[i].GetComponent<PuppetTrigger>().AddPath(new GameObject[] { item });
}
```

UML model

Az eddigiek modellje:



Optimalizálás

MapVisibility.cs

(Mivel az optimalizálás jóval később, a kész program elkészülésekor szokott megtörténni, ezért ez rendhagyó volt, viszont egyben elengedhetetlen is, mivel ennek egy gyengébb számítógépen is el kell majd futnia az iskolában.)

Mivel alaphelyzetben a pálya kigenerálása után az egész pálya aktívan be van töltve, ezért jelentős erőforrásoktól esünk el. Ezt azzal küszöbölöm ki, hogy csak azt a részt tartom betöltve, amit a felhasználó lát, vagy éppen szüksége van rá.

A pálya elkészülése után aktiválódik ez a kód melynek első dolga, hogy két csoportba osztja a meglévő gameobject-eket. Az egyik a `renderGameObject`, mely a vizuálisan megjelenő elemeket tartalmazza, a másik pedig a `physicsGameObject` ami a fizikai interakcióval kapcsolatos elemeket tartalmazza (trigger-eket).

A `renderGameObject`-be tartozó elemek jelen pillanatban kizárólag HexField-ek. Ezek csak akkor aktívak, ha a felhasználó által látott területen helyezkednek el. Ezt abból állapítja meg a program, hogy milyen távol van az adott elem a kamerától az x és z tengelyen.

A `physicsGameObject` ugyanígy működik, azzal az egy előfeltétellel szemben, hogy a játékosnak „fognia” kell egy bábut melyet le akar helyezni.

További optimalizálásképpen hozzá lett adva, hogy a ki-bekapcsolt állapota az elemeknek ne frissüljön minden egyes alkalommal, csak akkor, ha változás történik. Ez annyit takar, hogy nincs bejárva az összes elem minden alkalommal amikor ellenőrzi. Nem próbálja meg folyamatosan inaktívról inaktívra módosítaná vagy aktívról aktívra, csak akkor, ha biztos abban, hogy valamelyik elem értékének meg kell változnia.

```
private bool InView(Transform t) {
    if (t.position.z > (MCT.position.z + -10) &&
        t.position.z < (MCT.position.z + 70) &&
        t.position.x < (MCT.position.x + 75) &&
        t.position.x > (MCT.position.x - 75))
        return true;
    return false;
}
void FixedUpdate() {
    if (SMC.Changed) /* Ha a kamera helye változott */
        foreach (var item in renderGameObject) {
            if (InView(item.transform))
                item.SetActive(true);
            else
                item.SetActive(false);
        }
    /* Az előző vizsgálathoz képest, ha változott az érték akkor tárolja el az újat */
    if (lPM_HoldingPuppet != PM.HoldingPuppet) {
        lPM_HoldingPuppet = PM.HoldingPuppet;
        if (PM.HoldingPuppet)
            foreach (var item in physicsGameObject)
                if (InView(item.transform))
                    item.SetActive(true);
                else
                    item.SetActive(false);
        else
            foreach (var item in physicsGameObject)
                item.SetActive(false);
    }
}
```

Bábu lehelyezés

PathTrigger.cs, PuppetTrigger.cs

A két kód eléggé hasonló ezért egy részen belül magyarázom el.

	PathTrigger	PuppetTrigger
Tárolt bábú melyet a felhasználó helyez bele `Puppet/Path`	Egy Path az-az egy út bábú	Egy Puppet az-az egy város/falu bábú
Hozzá kapcsolódó HexField-ek `connectedField`	4 darabot tároló el	3 darabot tárol el
Egymásban tárolt példányok `connectedPath/-Puppet`	2 darab példány	3 darab példány

Ha ezek a példányok collision-be lépnek egy önmagával megegyező típussal akkor `InstanceID` alapján az marad fent, akinek nagyobb az azonosítója. Mielőtt törölné a fiatalabb példány a másikat, átmásolja belőle az összes olyan adatot, amit eddig még nem tartalmaz, annak függvényében, ha van még benne hely. Mivel a trigger-eket a HexField-ek létrehozáskor ellátják a saját példányukkal így példa ként három darab PuppetTrigger keletkezik egy helyen. Mindegyik tartalmazza az adott sarkot érintő HexField-ek egyikét, majd a collision lefutása alapján lényegében csak egyesülnek és eltárolják a környezetüket. A PathTrigger-errel is ugyan ez a helyzet, azzal az ellentéttel, hogy az ő létrehozási helyük két PuppetTrigger-ből lesz, ezáltal a létrehozás pillanatában azokból másolódik át beléjük, hogy miből áll a környezetük.

Amennyiben a collision egy számukra ideális bábuval fut le belemásolják magukat a bábu legutoljára érintett trigger részébe.

```
private void OnCollisionEnter(Collision collision)
{
    /* Generáláskor több trigger is kerül egy helyre majd van utána
     * egy frame-es várakoztatás, akkor itt a collison meghívodik
     * és az egymásra helyezett triggerekből csak a legújabb trigger
     * marad fent, amit InstanceId alapján határozok meg */
    if (collision.gameObject.tag == "PuppetTrigger") //ha egy azonos object-el "ütközik"
    {
        /* olvassa át a másik triggerből a saját field-jeit
         * (azokat, amikkel nem rendelkezik még) */
        AddField(collision.gameObject.GetComponent<PuppetTrigger>().ConnectedField);
        //Ha a jelenlegi triggernek nagyobb az ID-ja (azaz újabb) akkor törli a régebbit
        if (collision.gameObject.GetInstanceID() < transform.gameObject.GetInstanceID())
            Destroy(collision.gameObject, 0.0f);
    }
    /* Ha várossal vagy falu bábúval "ütközik" akkor hozzá adja magát
     * a bábúhoz (használva a PlaceModel.cs-ben) */
    if (collision.gameObject.name.Contains("City") ||
collision.gameObject.name.Contains("Town"))
        collision.gameObject.GetComponent<Puppet>().TouchedTrigger = gameObject;
}
```

Felhasználó bemenet

InputHandler.cs

A játékos által elvégzett interakció melyhez a játékon belül funkcionalitás kötődik, az itt tárolódik el. Ilyen például a görgő klikk, a görgetés vagy a képernyő széléhez húzott kurzor, mellyel a kamerát tudja befolyásolni a felhasználó.

A görgetés rögtön a hozzá tartozó interakció értékét tárolja el, ez pedig a kamera közelítése. Azonban az itt tárolt érték nem távolság, hanem annak a mennyisége, hogy mennyivel kisebb legyen a kamera látó szöge, ezzel a közelítés illúzióját keltve.

```
if (Input.GetButtonDown("LClick"))
{
    LeftClick.pressing = true;
    LeftClick.pressed++;
}
if (Input.GetButtonUp("LClick"))
    LeftClick.pressing = false;

if (Input.GetButtonDown("MClick"))
{
    MiddleClick.pressing = true;
    MiddleClick.pressed++;
}
if (Input.GetButtonUp("MClick"))
    MiddleClick.pressing = false;

if (Input.GetAxis("Mouse ScrollWheel") < 0f) // backwards
{
    Zoom *= 1.1f;
    if (Zoom > 40f)
        Zoom = 40f;
}
else if (Input.GetAxis("Mouse ScrollWheel") > 0f) // forward
{
    Zoom /= 1.1f;
    if (Zoom < 1f)
        Zoom = 1f;
}

Sides = Vector2.zero;
if (Input.mousePosition.y > Screen.height - 5 && !MiddleClick.pressing)
    Sides.y = 3f;
else if (Input.mousePosition.y < 5 && !MiddleClick.pressing)
    Sides.y = -3f;

if (Input.mousePosition.x > Screen.width - 5 && !MiddleClick.pressing)
    Sides.x = 3f;
else if (Input.mousePosition.x < 5 && !MiddleClick.pressing)
    Sides.x = -3f;
```

Bábu

Puppet.cs

Egy egyszerű kódrész mely a bábuhoz tartozó adatokat tárolja el, mint például, hogy melyik trigger-hez ért hozzá utoljára, avagy milyen modellje van és hogy ahhoz milyen sebességű nyersanyag kitermelés tartozik hozzá.

```
public class Puppet : MonoBehaviour
{
    public GameObject TouchedTrigger;
    private PreViewModel.pwModel model;
    private int collectRate;
    public PreViewModel.pwModel Model {
        get => model;
        set {
            model = value;
            switch (model)
            {
                case PreViewModel.pwModel.City: collectRate = 2; break;
                case PreViewModel.pwModel.Town: collectRate = 1; break;
                default: collectRate = 0; break;
            }
        }
    }
    public int CollectRate { get => collectRate; }
}
```

Camera mozgása

SideMoveCamera.cs

A játékos a kamerát kétféleképpen tudja mozgatni. Az egyik, hogy a görgő klikket letartja és közben az egér mozgásával megegyező irányba tolja el a kamerát. A másik lehetősége pedig az, hogy a képernyő szélső 5 pixeléhez érinti és így eltolja az adott oldalhoz tartozó irányba.

A görgő klikkes módszert legelőször a Raycast alapján szerettem volna megoldani, de végül a képernyő pixel alapon lett megvalósítva. Ez úgy történik, hogy amikor a felhasználó görgő klikkel akkor elmentjük ennek a helyét, mint kezdőpont a képernyő pixel-ei alapján. A kezdőponttól megtett távolságot egy kétdimenziós vektorrá alakítom majd a vektort megszorzom 0.1-gyel így megkapva a 10%. Ezt utána hozzáadom a kamera azon pozíciójához mely a görgő klikkeléskor volt.

A képernyő széléhez érintett kurzort egyszerűen a képernyő méretéből és a kurzor pozíciójából számolom ki. A képernyő méretéből kivonok 5-t és megnézem, hogy az egér pozíciója nagyobb-e, avagy a másik irányba pedig azt nézem, hogy a kurzor pozíciója kisebb-e mint 5. Az adott iránytól függően a kamera pozíciójához hozzáadok vagy elveszek 3-mat.

```
public void Offset()
{
    maxDown += offsetDown;
    maxUp += offsetUp;
    maxLeft += offsetLeft;
    MaxRight += offsetRight;
}

// Start is called before the first frame update
void Start()
{
    mCamera = Camera.main;
    IH = inputHandler.GetComponent<InputHandler>();
    PM = ModelRay.GetComponent<PlaceModel>();
}

// Update is called once per frame
Vector3 v3Old = Vector3.one;
void FixedUpdate()
{
    Vector3 v3 = -Vector3.one;
    if (!m)
    {
        m = true;
        mStart = Input.mousePosition;
        cStart = mCamera.transform.position;
    }

    if (IH.MiddleClick.pressing)
    {
        Vector2 mNow = new Vector2(
             mStart.x - Input.mousePosition.x,
             mStart.y - Input.mousePosition.y
        );
        mNow *= 0.1f;
        v3 = new Vector3(cStart.x + mNow.x, cStart.y, cStart.z + mNow.y);
    }
    else
    {
        Vector3 v3cam = mCamera.transform.position;
        v3 = new Vector3(v3cam.x + IH.Sides.x, v3cam.y, v3cam.z + IH.Sides.y);
        m = false;
    }


    if (!PM.HoldingPuppet && v3 != -Vector3.one)
    {
        if (v3.z > maxUp) v3.z = maxUp;//242
        if (v3.z < maxDown) v3.z = maxDown;//0
        if (v3.x < maxLeft) v3.x = maxLeft;//40
        if (v3.x > maxRight) v3.x = maxRight;//260
        mCamera.transform.position = v3;
        if (v3Old != v3)
        {
            v3Old = v3;
            changed = true;
        }
        else
            changed = false;
    }
    mCamera.fieldOfView = 20f + IH.Zoom;
}
```

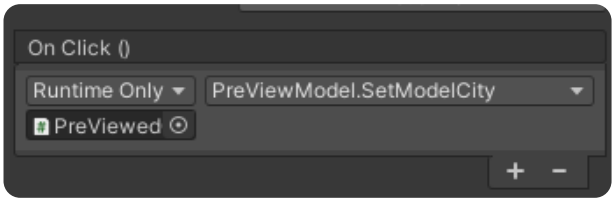
Bábuk működése

PlaceModel.cs, PreViewModel.cs, PressingUIButton.cs, CollectingBehaviour.cs, GlobalBehaviour.cs

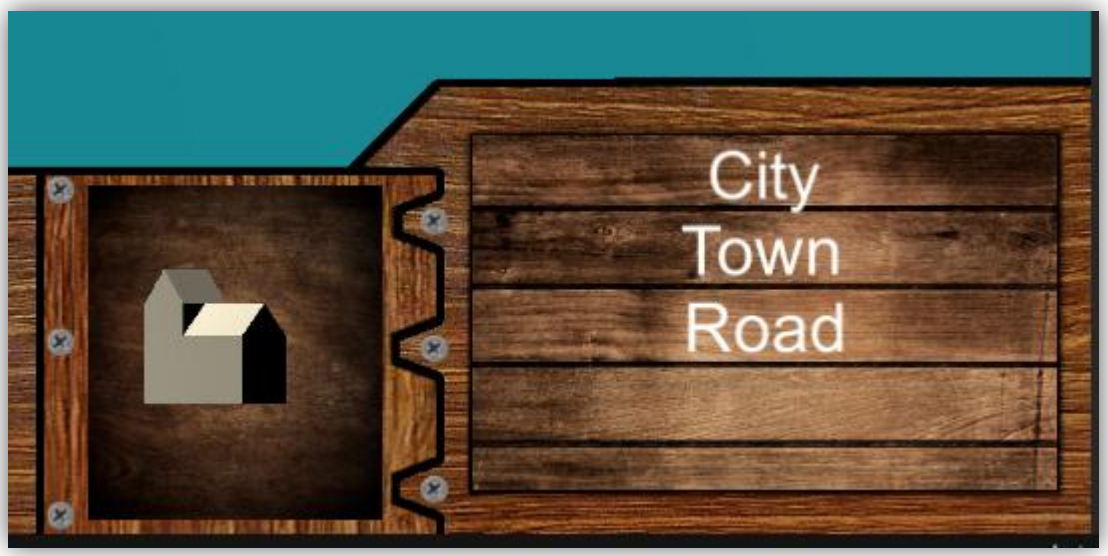
Talán eddig a legbonyolultabb kódrész a bábuk lehelyezése, mivel az egymást érintő bábuk kapcsolatot alkotnak és változtatnak a játék menetén.

Lehelyezés

(Jelenleg nincs szabály arra, hogy hova lehet bábut lehelyezni, tengerre nem lehet semmit, sarok távolság épületeknél, nincs önálló út stb.)



Először is ki kell választania a felhasználónak, hogy milyen bábut szeretne lehelyezni, ezt a jobb alsó sarokban lévő UI ObjectSelector-ban teheti meg. Ha itt rákattint egy elemre akkor a beleépített OnClick() funkció által meghívódik a megfelelő kódrész a PreViewModel.cs-ből és a bábu előnézete megjelenik. Ez az előnézet egy külön kameráról render-elt kép. Ha az előnézeti képen letartja a felhasználó a bal egérgombot és kihúzza a pályára akkor a bábu színével mutatja az elhelyezhetőséget. Ha a bábu piros akkor, ha elengedi a bal klikket, a bábu vissza kerül az előnézetbe, viszont, ha zöld állapotában engedi el, akkor ott marad.




```
public bool Pressing = false;
public bool Hover = false;

public void OnPointerDown(PointerEventData eventData) { Pressing = true; }
public void OnPointerUp(PointerEventData eventData) { Pressing = false; }
public void OnPointerEnter(PointerEventData eventData) { Hover = true; }
public void OnPointerExit(PointerEventData eventData) { Hover = false; }
```

A bábu pályára lehelyezésekor az egérből raycast-elődott pont alapján mozog. Ha a létrejött pont és egy tigger 5 unit-ra van egymástól és a mozgatott object collison-ől a tigger-rel akkor automatikusan illeszkedik a trigger pozíciójára és rotációjához. Amennyiben a raycast-elt pont távolabb van mint 5 unit az adott trigger-től, akkor az object továbbra is a létrejött pontot követi.

```
void LateUpdate()
{
    if (!holdingPuppet && BTN.Pressing && IH.LeftClick.pressing)
        if (PreViewerHold.transform.childCount > 0) {
            Puppet = PreViewerHold.transform.GetChild(0).gameObject;
            holdingPuppet = true;
        }

    if (holdingPuppet && !IH.LeftClick.pressing) {
        holdingPuppet = false;
        if (!gotLocation) {
            Puppet.transform.parent = PreViewerHold.transform;
            Puppet.transform.position = PreViewerHold.transform.position;
            Puppet.transform.localScale = new Vector3(1f, 1f, 1f);
            Puppet.transform.GetChild(0).
            GetComponent<MeshRenderer>().material.color = new Color(1f, 1f, 1f, 1f);
        }
        else {
            var pt = Puppet.GetComponent<Puppet>();
            /* fura hiba javítására szolgál ez a trycatch, ha egy bábut kihúzunk
             * a preview-ból, de csak a hud-ra majd vissza akkor hitbát dob. */
            try
            {
                if (pt.TouchedTrigger.gameObject.tag == "PuppetTrigger")
                    pt.TouchedTrigger.GetComponent<PuppetTrigger>().Puppet = Puppet;
                else
                    pt.TouchedTrigger.GetComponent<PathTrigger>().Path = Puppet;
                Puppet.GetComponent<CollectingBehaviour>().enabled = true; //össze kapcsolódás
            }
            catch(System.Exception) {}
        }
        Puppet = null;
    }
    if (holdingPuppet) {
        if (BTN.Hover) {
            Puppet.transform.parent = PreViewerHold.transform;
            Puppet.transform.position = PreViewerHold.transform.position;
            Puppet.GetComponent<Puppet>().TouchedTrigger = null;
            Puppet.transform.localScale = new Vector3(1f, 1f, 1f);
            Puppet.transform.GetChild(0).
            GetComponent<MeshRenderer>().material.color = new Color(1f, 1f, 1f, 1f);
        }
        else {
            Puppet.transform.parent = transform;
            PreViewerHold.GetComponent<PreViewModel>().SetModelNone();
            Puppet.transform.rotation = new Quaternion();
            Puppet.transform.Rotate(0f,90f,0f);
            Puppet.transform.localScale = new Vector3(0.8f, 0.8f, 0.8f);

            Ray ray = Camera.main.ScreenPointToRay(Input.mousePosition);
            RaycastHit hit;
            if (Physics.Raycast(ray, out hit, 150.0f, clickMask)) {
                Vector3 hitpoint = hit.point;
                GameObject TouchedTrigger = Puppet.GetComponent<Puppet>().TouchedTrigger;
                if (TouchedTrigger != null) {
                    var TriggerPos = TouchedTrigger.transform.position;
                    TriggerPos.y = hitpoint.y;
                    if (Vector3.Distance(TriggerPos, hitpoint) < 5.0f) {
                        Puppet.transform.position = TouchedTrigger.transform.position;
                        Puppet.transform.GetChild(0).
                        GetComponent<MeshRenderer>().material.color = new Color(0f, 1f, 0f, 1f);
                        Puppet.transform.rotation = TouchedTrigger.transform.rotation;
                        gotLocation = true;
                    }
                    else
                        TouchedTrigger = null;
                }

                if (TouchedTrigger == null) {
                    Puppet.transform.position = hitpoint;
                    Puppet.transform.GetChild(0).
                    GetComponent<MeshRenderer>().material.color = new Color(1f, 0f, 0f, 0.2f);
                    gotLocation = false;
                }

            }
        }
    }
}
```

Össze kapcsolódás

A lehelyezés végénél meghívódik a CollectingBehaviour.cs ami egy előkészülettel kezdi.

```
Puppet.GetComponent<CollectingBehaviour>().enabled = true;
```

Ebben a részben feltérképezi a közvetlen környezetét a trigger-ben tároltak alapján és a talált bábukat hozzáadja a csoportjához.

```
GameObject trigger = transform.GetComponent<Puppet>().TouchedTrigger;
if (trigger.tag == "PuppetTrigger")
{
    AddPuppet(gameObject);
    //A jelenlegi trigger-hez csatlakoztatott út trigger-eknek az út object-je.
    AddPath(trigger.GetComponent<PuppetTrigger>().
    ConnectedPathTrigger.OfType<GameObject>().ToList().
    Select(x => x.GetComponent<PathTrigger>().Path).ToList());
}
else
{
    AddPath(gameObject);
    var cPuppet = trigger.GetComponent<PathTrigger>().ConnectedPuppetTrigger.ToList();
    cPuppet.ForEach(tPuppet =>
    AddPath(tPuppet.GetComponent<PuppetTrigger>().
    ConnectedPathTrigger.OfType<GameObject>().ToList().
    Select(tPath => tPath.GetComponent<PathTrigger>().Path).ToList()));
    AddPuppet(cPuppet.Select(tPuppet =>
    tPuppet.GetComponent<PuppetTrigger>().Puppet).ToList());
}

bool re = false;
do
{
    re = false;
    connectedPath.ToList().ForEach(path => {
        if (path != gameObject &&
            AddPath(path.GetComponent<CollectingBehaviour>().
            Master.GetComponent<CollectingBehaviour>().ConnectedPath))
            re = true;
    });
    connectedPath.ToList().ForEach(path => {
        if (path != gameObject &&
        AddPuppet(path.GetComponent<CollectingBehaviour>().
        Master.GetComponent<CollectingBehaviour>().ConnectedPuppet))
        re = true;
    });

    connectedPuppet.ToList().ForEach(puppet => {
        if (puppet != gameObject &&
        AddPath(puppet.GetComponent<CollectingBehaviour>().
        Master.GetComponent<CollectingBehaviour>().ConnectedPath))
        re = true;
    });
    connectedPuppet.ToList().ForEach(puppet => {
        if (puppet != gameObject &&
        AddPuppet(puppet.GetComponent<CollectingBehaviour>().
        Master.GetComponent<CollectingBehaviour>().ConnectedPuppet))
        re = true;
    });
} while (re);
```

Az összekapcsolódott részekben mindig a legfiatalabb elem a „Master”. A Master lényege, hogy eltárolja a csoportban lévő elemeket és hogy milyen HexField-ekhez van csatlakozva a csoport. Amikor egy csoport Master-je megváltozik a régi Masterből átmásolódik minden az újba, és a régi tartalma törlődik memória felszabadítás szempontjából.

Mivel jelenleg a játék még fejlesztés alatt áll, ezért a bábuk egy véletlenszerű színnel jelölik, hogy milyen csoportba tartoznak.

```
var col = new Color(
(float)RND.NextDouble(), (float)RND.NextDouble(), (float)RND.NextDouble(), 1f);
```

Amikor egy új Master lesz egy csoportban akkor mindig bemásolja a saját példányát az össze csoport elembe, hogy amikor egy új elem kerül a csoportba az alapján megtudja találni, hogy honnan kell átmásolni a csoport többi tagját.

```
foreach (GameObject item in connectedPath)
{
    AddField(item.GetComponent<Puppet>().
    TouchedTrigger.GetComponent<PathTrigger>().ConnectedField);
    item.transform.GetChild(0).GetComponent<MeshRenderer>().material.color = col;
    var cb = item.GetComponent<CollectingBehaviour>();
    cb.Master = gameObject;
}
foreach (var item in connectedPuppet)
{
    var puppet = item.GetComponent<Puppet>();
    AddField(puppet.TouchedTrigger.GetComponent<PuppetTrigger>().ConnectedField);
    collectRate += puppet.CollectRate;
    item.transform.GetChild(0).GetComponent<MeshRenderer>().material.color = col;
    var cb = item.GetComponent<CollectingBehaviour>();
    cb.Master = gameObject;
}
```

A létrejövetel után összeszámolja, hogy mennyi a csoport „gyűjtési mennyisége”, mint korábban említettem a városé 2 és a falué pedig 1. Továbbá összeszámolja, hogy milyen és mennyi HexField-ekhez van csatlakozva a csoport és ez alapján hozzá adja magát GlobalBehaviour.cs-hez.

```
collectRateType.Clear();
foreach (HexField.hexType hex in
        (HexField.hexType[])HexField.hexType.GetValues(typeof(HexField.hexType)))
    collectRateType.Add(hex, 0);
foreach (var item in connectedField)
    collectRateType[item.GetComponent<HexField>().HexType] += 1;

var Global = GameObject.FindGameObjectWithTag("Global").GetComponent<GlobalBehaviour>();
foreach (HexField.hexType hex in
        (HexField.hexType[])HexField.hexType.GetValues(typeof(HexField.hexType)))
{
    GlobalBehaviour.TickHandler handler = ((sender, e) =>
    Owner.GetComponent<User>().AddCollected(hex, collectRateType[hex] * collectRate));
    Global.eSeconds += handler;
    Events.Add(handler);
}
```

A nyersanyagok gyűjtése a GlobalBehaviour.cs-ben event-eken keresztül történik melyek jelenleg minden másodpercben meghívódnak viszont ez később módosulni fog.

```
public event TickHandler eSeconds;
public event TickHandler eMinutes;
public event TickHandler eHours;

public EventArgs e = null;
public delegate void TickHandler(GlobalBehaviour m, EventArgs e);
private int called = 0;
private int seconds = 0;
private int minutes = 0;
private int hours = 0;

void FixedUpdate()
{
    called++;
    if (called == 50)
    {
        eSeconds(this, e);
        called = 0;
        seconds++;
        if (seconds == 60)
        {
            eMinutes(this, e);
            seconds = 0;
            minutes++;
            if (minutes == 60)
            {
                eHours(this, e);
                minutes = 0;
                hours++;
            }
        }
    }
}
```

Ezek az event-ek az összes kódrész számára elérhetőek és szabadon adhatnak hozzá elemeket.

Nyersanyag tárolás és kijelzés

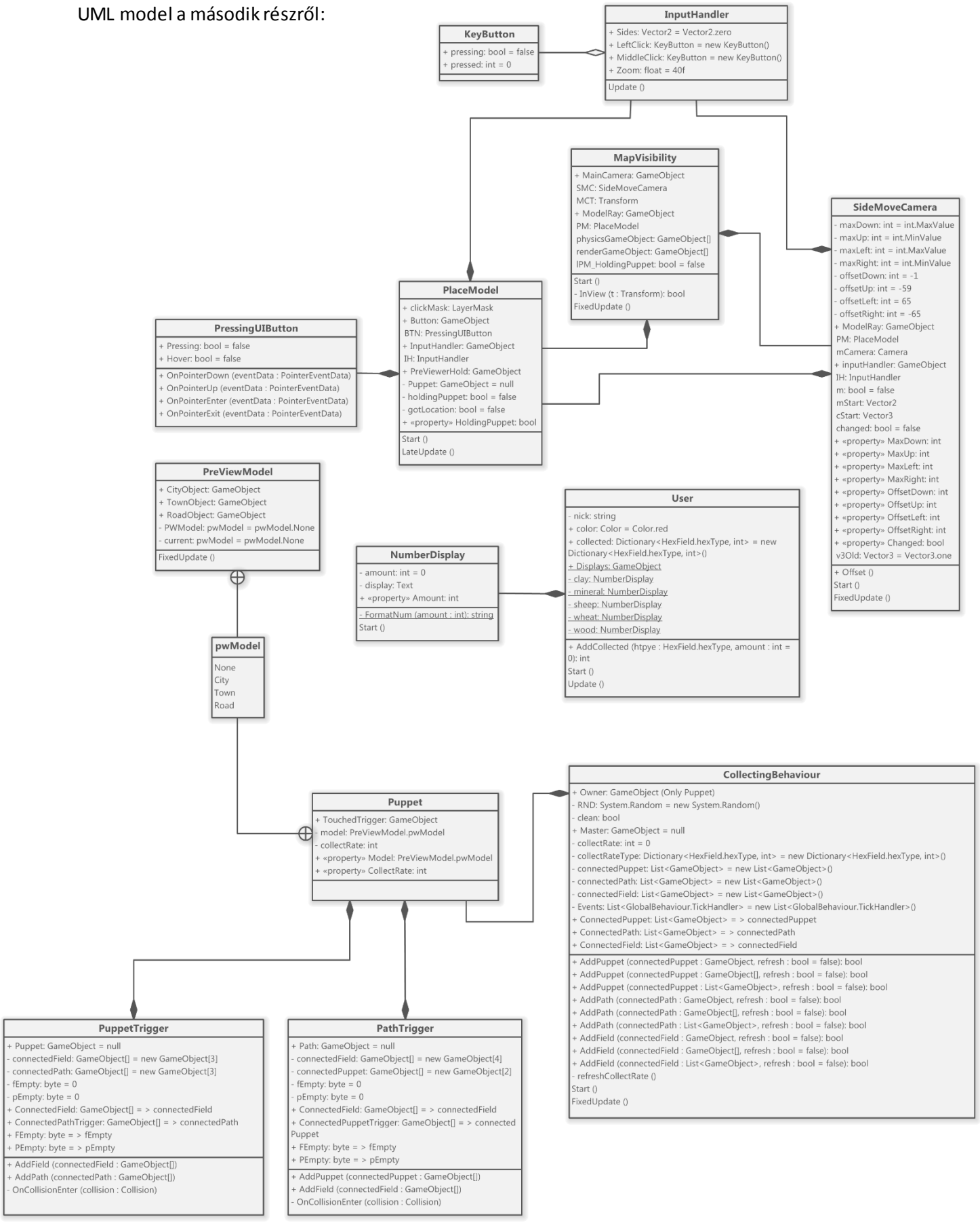
User.cs, NumberDisplay.cs

Amikor egy bábu lehelyeződik, akkor a változói között szerepelni fog egy tulajdonos gameobject. Ez a példány a helyi játékosunk adatai. Az adat struktúrát a User.cs tartalmazza. Amikor nyersanyag termelődik akkor ezen a referencián keresztül kapja meg a felhasználó a nyersanyagokat melyet utána a NumberDisplay.cs megjelenít a képernyő bal felső sarkában.


```
private string nick;
public Color color = Color.red;
public Dictionary<HexField.hexType, int> collected = new Dictionary<HexField.hexType,
int>();
public static GameObject Displays;
private static NumberDisplay clay;
private static NumberDisplay mineral;
private static NumberDisplay sheep;
private static NumberDisplay wheat;
private static NumberDisplay wood;

public int AddCollected(HexField.hexType htpye, int amount = 0)
{
    if (collected.ContainsKey(htpye))
        collected[htpye] += amount;
    else
        collected.Add(htpye, amount);
    switch (htpye)
    {
        case HexField.hexType.Clay: clay.Amount = collected[htpye];
            break;
        case HexField.hexType.Feild:
            sheep.Amount = collected[htpye];
            break;
        case HexField.hexType.Mineral:
            mineral.Amount = collected[htpye];
            break;
        case HexField.hexType.Wheat:
            wheat.Amount = collected[htpye];
            break;
        case HexField.hexType.Wood:
            wood.Amount = collected[htpye];
            break;
        default:
            break;
    }
    return collected[htpye];
}

void Start()
{
    nick = gameObject.name;

    Displays = GameObject.FindGameObjectWithTag("CDisplay");
    clay = Displays.transform.GetChild(0).GetComponent<NumberDisplay>();
    mineral = Displays.transform.GetChild(1).GetComponent<NumberDisplay>();
    sheep = Displays.transform.GetChild(2).GetComponent<NumberDisplay>();
    wheat = Displays.transform.GetChild(3).GetComponent<NumberDisplay>();
    wood = Displays.transform.GetChild(4).GetComponent<NumberDisplay>();
}
```

A hely felhasználó már tartalmazza egyes változókat melyek később fontosok lesznek a multiplayer-nél

UML model a második részről:



Felhasználói dokumentáció

(Jelen pillanatában a produktum még nem minősíthető játéknak véleményem szerint, viszont a továbbiakban játékként fogok rá hivatkozni)

Rövid leírás

A játék kész formájában egy valós idejű stratégiai játék lesz, melyet interneten keresztül játszatunk majd akárhonnan, ahol elérhető a program egy példánya, internet kapcsolat és erre alkalmas eszköz.

A játék lényege, hogy a területek elfoglalásával minél gyorsabban fejlődjünk és építsünk minél nagyobb hálózatot. Minél több épülete van egy játékosnak annál több pontja van és ezáltal tudja megnyerni a játékot.

Rendszerkövetelmények

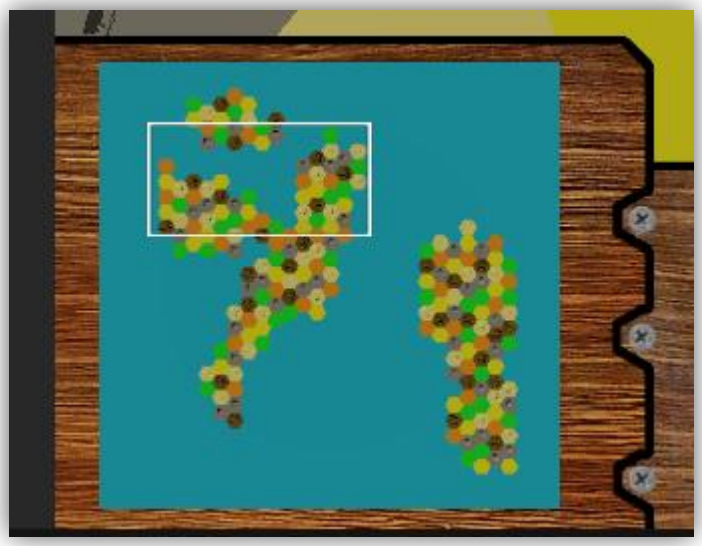
- Operációs rendszer: Windows 7 SP1 vagy újabb
- Processzor: 1 magos 1 GHz vagy jobb
- Memória (RAM): szabad 1 GB vagy több
- Tárhely: 100MB vagy több
- Szoftver: .NET keretrendszer 4.8.0 vagy újabb
- Grafikai API: DX10, DX11, vagy DX12
- Grafikai memória (VRAM): 64 MB vagy több

Játékmenet

Térkép és pálya

A játék indítása után egy véletlenszerű pályát hoz létre. Ez a jobb alsó sarokban láthatja térkép formájában. A térképen megjelenő fehér négyzet mutatja, hogy a terület mely részét láthatjuk éppen a képen.

FONTOS tudni, hogy a térkép nem mutatja ki a lehelyezett bábukat!



Bábu lehelyezése

A létrejött pályán akárhová lehelyezhetünk egy bábut mely nyersanyagot fog gyűjteni.

A bábu lehelyezéséhez ki kell egyet választani a jobb alsó menüből.

- City = Város
- Town = Falu
- Road = Út

Ha kiválasztotta a tetszőleges bábut a jobb alsó menüből, kattintson a bal egérgombbal az ön által kiválasztott szövegre. Ezt követően elölnézet jelenik meg a báburól, a menütől jobbra eső részen.

Amennyiben a kiválasztott bábu nem tetszik, tetszés szerint változtathatja a menü segítségével.

A bábut úgy lehet kihelyezi a pályára, hogy az elölnézeti képkeretbe helyezzük a kurzort majd a bal egérgomb folyamatos lenyomásával egy tetszőleges pályarészre húzzuk. Amennyiben várost vagy falut választottunk ki, a bábu lehelyezhető három darab sarok találkozásához. Ha utat szeretnénk lehelyezni azt a hatszögek élére kell húzni.

A lehelyezni kívánt bábu csak akkor marad ott, ha zölden világít, mielőtt felengedjük a bal egérgombot. Ha felengedéskor a mozgatott bábu piros volt, akkor a bábu vissza kerül az elölnézeti keretbe.

Nyersanyagok

A pályán jelenleg 7 fajta mezőt különböztetünk meg. Egyes mezők már rendelkeznek egyedi kinézettel, viszont egyeseket még csak szín alapján lehet azonosítani. A megszerzett nyersanyag a bal felső sarokban látható.

Nyersanyagokat a lehelyezett várossokkal vagy falvakkal lehet szerezni. Minden város vagy falu nyersanyagot nyer ki azokból a mezőkből melyek az adott sarkot alkotják. Ha ezekhez hozzákötünk utakat akkor az úthálózatot érintő összes mezőből is nyersanyagot termelünk. Az egy hálózatba helyezett bábuk termelése összeadódik.

Felhasznált külső források

A játékban használt nyersanyag ikonok alapja: https://game-icons.net

Felmerülő kérdéseimre a válaszok Unity-vel kapcsolatban: https://forum.unity.com/

A pálya felépítésének alapja: https://www.redblobgames.com/grids/hexagons/

YouTube csatornák, ahonnan tutorial-okat néztem a munkám alatt:

https://www.youtube.com/channel/UCbx1TZgxfIauUZyPuBzEwZg

https://www.youtube.com/channel/UCFK6NCbuClVzA6Yj1G_ZqCg