

Unity Project: Debuggaus ja Optimointi

21.9.2020

Päivän aihe

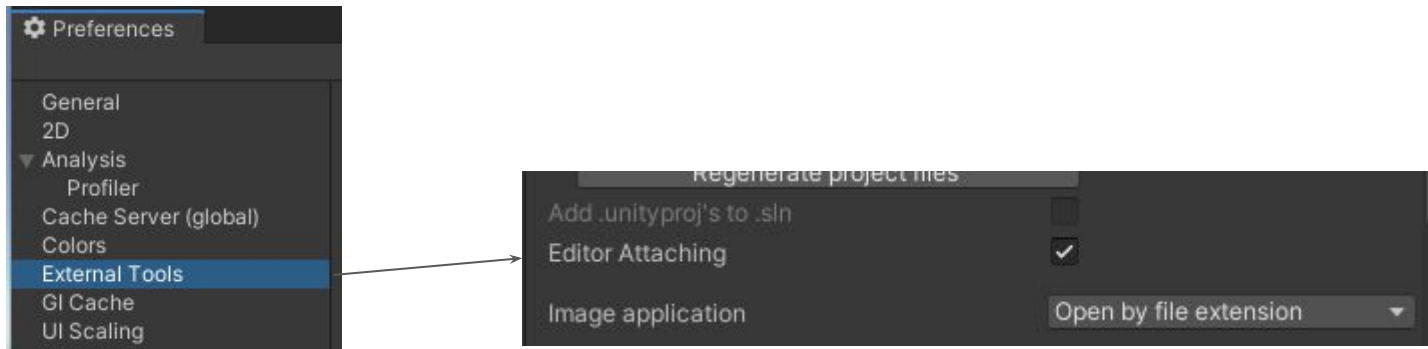
- C# Debuggausta Unityssä
- Pelin optimointia
 - Visuaalinen Optimointi
 - LOD, Batching ja Occlusion Culling
 - Scriptien Optimointi
 - Fysiikan Optimointi
 - Valotuksen Optimointi

C# Debuggaus Unityssä

Debuggauksen avulla voidaan tutkia ja korjata ongelmia C# scripteissä. Sen avulla voidaan käydä läpi scriptin ominaisuuksia & muuttujia tietyissä kohtaa peliä.

Debuggaus on helppo tapa varmistaa, että scripti ja muuttujat toimivat oikein tietyissä kohtia peliä, jolloin debuggauksen break point kohtaa kutsutaan.

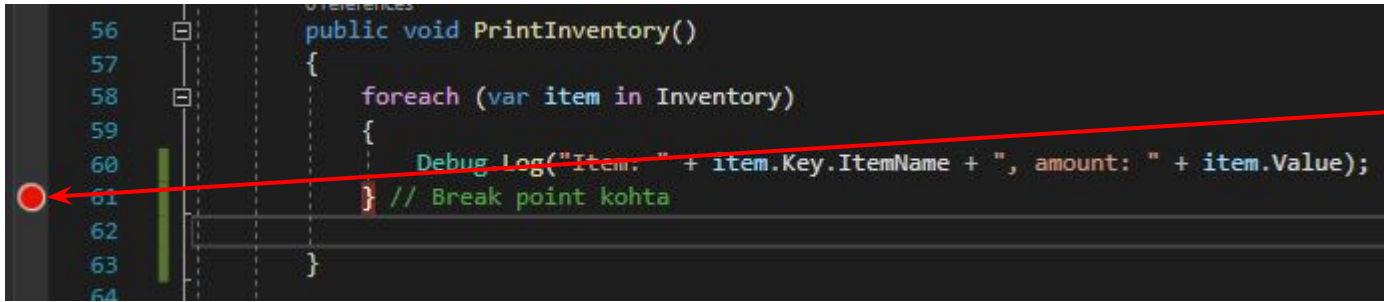
Jotta Debuggaus onnistuu Visual Studion kautta, täytyy “Editor Attaching” olla päällä “Preferences/External Tools” välilehdellä:



C# Debuggaus Unityssä

Kun halutaan debuggaa tiettyä kohtaa scriptissä, täytyy riville ensin lisätä break point. Break pointteja on kohta, johon pysähdytään scriptissä, mikäli break point lisätty kohta kutsutaan. Break pointteja voi olla scriptissä useampi kuin yksi.

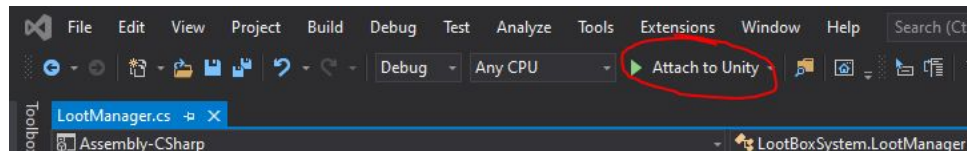
Alla olevassa kuvassa Break Point (punainen pallo) on lisätty foreach loopin jälkeen, eli kun peli on käynnissä, peli pysähtyy kun Inventory lista on tulostettu Debug ikkunaan.



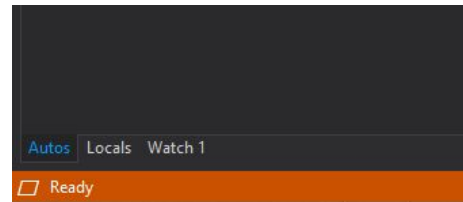
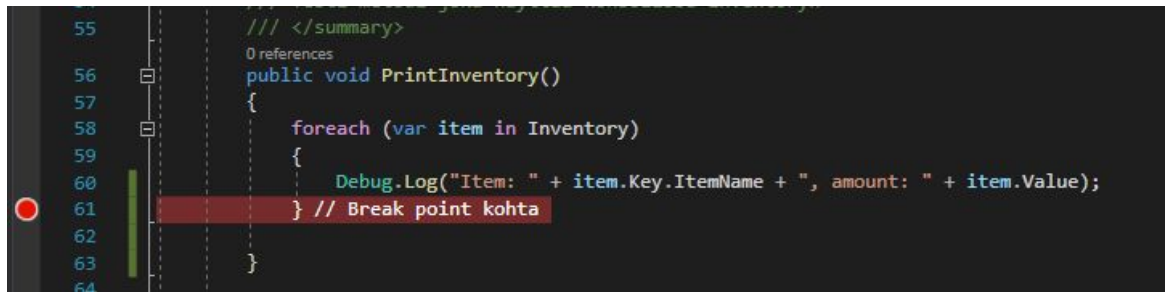
Break point lisätään painamalla harmaata osiota rivinumeron vierestä

C# Debuggaus Unityssä

Ennen Debuggauksen aloittamista, täytyy Visual Studiossa painaa yläpalkissa olevaa “Attach to Unity” nappia.



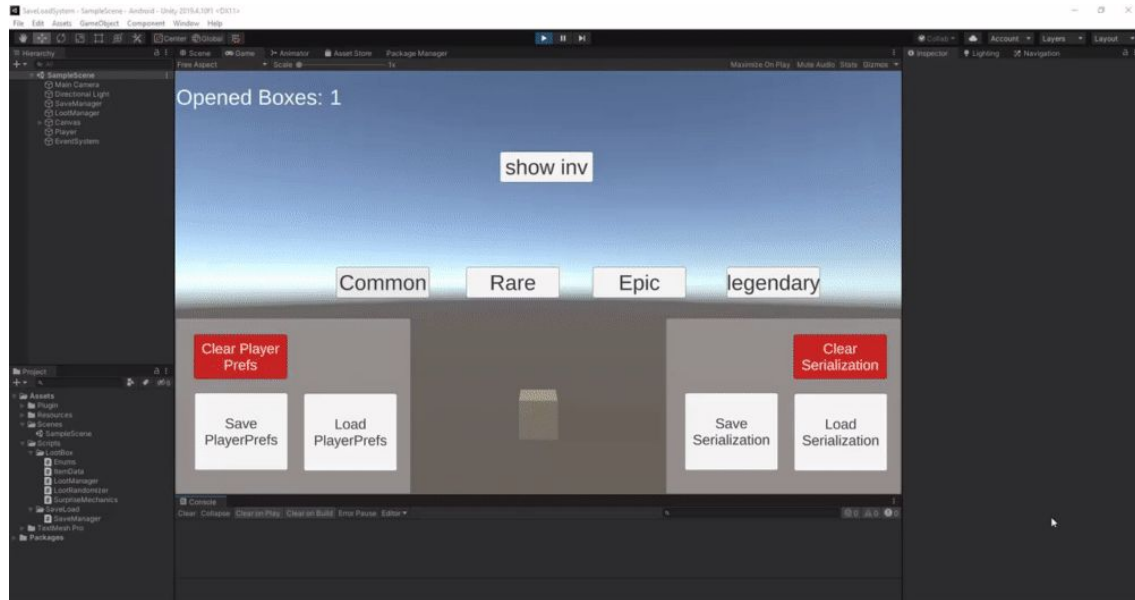
Tämän jälkeen Visual Studio alapalkissa lukee “Ready” ja debugattu rivi muuttuu punaiseksi:



C# Debuggaus Unityssä

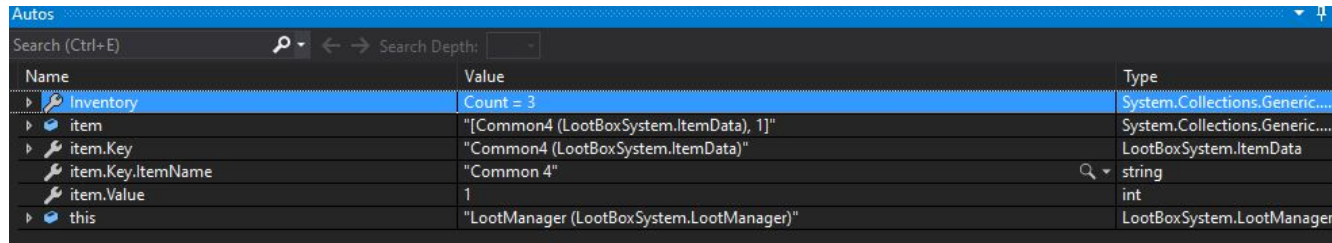
Kun on Play tilassa, ja painetaan “Show Inv” nappia pelissä, peli tulostaa inventoryn Debug ikkunaan, jonka jälkeen peli pysähtyy, visual studio aukeaa ja hyppää break pointin kohdalle joka asetettiin:

Nyt voidaan lähteä tutkimaan erilaisia muuttujia jota on käyty läpi scriptissä ja metodissa.



C# Debuggaus Unityssä

Visual Studiossa on nyt näkyvillä uusi välilehti, joka näyttää scriptissä käsittelyssä olleet muuttujat ja datat:

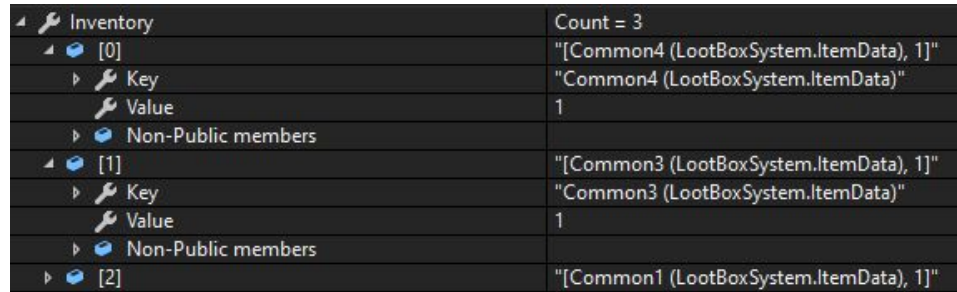


The screenshot shows the 'Autos' window in Visual Studio. It has a search bar at the top with the text 'Search (Ctrl+E)' and a search icon. Below the search bar is a table with three columns: 'Name', 'Value', and 'Type'. The table lists the following variables:

Name	Value	Type
Inventory	Count = 3	System.Collections.Generic....
item	"[Common4 (LootBoxSystem.ItemData), 1]"	System.Collections.Generic....
item.Key	"Common4 (LootBoxSystem.ItemData)"	LootBoxSystem.ItemData
item.Key.ItemName	"Common 4"	string
item.Value	1	int
this	"LootManager (LootBoxSystem.LootManager)"	LootBoxSystem.LootManager

Tässä esimerkissä tutkitaan Inventory listan dataa, joka sisältää 3 "ItemData" nimistä objektia, joilla on omat arvonsa.

Tässä voidaan tutkia onko kaikki itemit asetettu oikein inventoryyn, ja onko ItemData objekteilla oikeat arvot.



The screenshot shows the 'Watch' window in Visual Studio. It displays the 'Inventory' variable and its elements. The table shows the following structure:

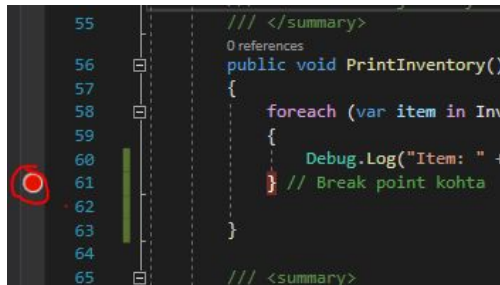
Inventory	Count = 3
[0]	"[Common4 (LootBoxSystem.ItemData), 1]"
Key	"Common4 (LootBoxSystem.ItemData)"
Value	1
Non-Public members	
[1]	"[Common3 (LootBoxSystem.ItemData), 1]"
Key	"Common3 (LootBoxSystem.ItemData)"
Value	1
Non-Public members	
[2]	"[Common1 (LootBoxSystem.ItemData), 1]"

C# Debuggaus Unityssä

Debug - tilan käyttö lopetaan painamalla “Stop” näppäintä Visual Studio yläpalkissa:



Pelin sammuttaminen Editorissa ei lopeta debug tilan käyttöä. Debug tilan käytön jälkeen kannattaa poistaa asetetut Break Pointit, mikäli niille ei ole enää tarvetta!



Break point poistetaan painamalla punaista palloa.

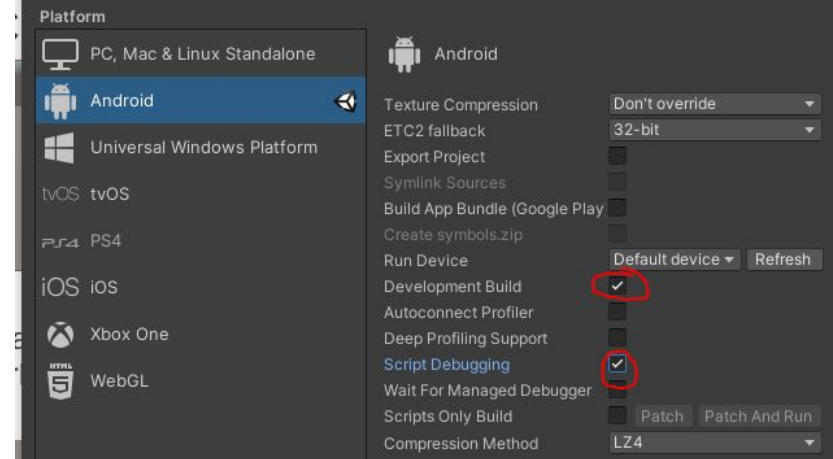
Tässä se on rivillä 61

Buildin Debuggaus

Buildattua peliä voidaan myös debugga, jonka avulla voidaan tutkia samalla tavalla kuin Editorissa.

Build Asetuksissa täytyy valita “Development Build” ja “Script Debugging”, jotta tämä buildin debuggaus toimii.

Kun halutaan debugga buildattua peliä, täytyy Visual Studio “Attach to Unity” napista valita kone jota käytetään

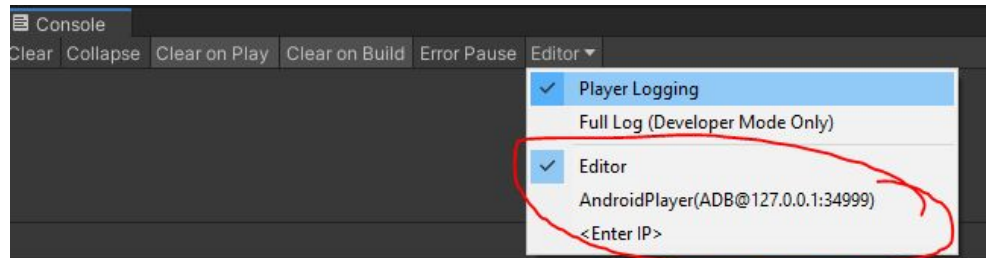


Buildin Testaus ja Console

Buildatun pelin console ikkunaa voidaan myös tutkia. Tämän avulla voidaan tutkia error viestejä, jotka mahdollisesti esiintyvät vain buildatussa pelissä.

Tämän saa toimimaan Development Build - merkityn pelin kanssa. Buildatun pelin console ikkunaa voidaan tutkia Editorin console välilehdellä, mutta pelin sisälle ilmestyy myös errorit mikäli niitä tulee.

Console ikkunan saa yhdistettyä buildattuun peliin valitsemalla “Editor” listalta käynnissä oleva buildattu peli:



Optimointi

Pelin voi optimoida monella eri tavalla, mutta pääsääntöisesti jokainen peli on erilainen ja optimoidaan eri tavalla. Jos on kehittämässä massiivista RPG fantasia peliä, kuten the Witcher 3, tulee sen pelin optimoinnissa keskittyä eniten visuaaliseen optimointiin. Mutta tietyissä peleissä optimointi voi olla myös scripteissä, tekoälyssä ja online ominaisuudessa.

- Visuaalinen / Graafinen Optimointi
- Scriptien Optimointi
- Fysiikan Optimointi

Visuaalinen optimointi

Visuaalista optimointia voidaan toteuttaa parilla eri tavalla.

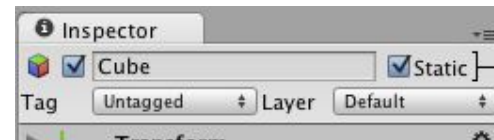
- Helpoin näistä on pitää peli mahdollisimman pienenä, eli jos kentässä on todella paljon ylimääräisiä asioita, joita pelaaja ei edes huomaa, kannattaa ne poistaa suoraan scenestä.
- **Draw Call batching.** Tämän ideana on “yhdistää” GameObjektit samaan “verkkoon”, eli meshiin.
 - Dynamic Batching on tarkoitettu pienille mesh:ille, joka yhdistää samanlaiset pisteet (vertices) yhteen ja piirtää ne samaan aikaan.
 - Static Batching yhdistää staattiset, ei liikkuvat, objektit yhdeksi isoksi mesh:iksi, jonka jälkeen niiden objektien renderointi on huomattavasti nopeampaa.

Visuaalinen optimointi

- **Objektien optimointi, jotka ovat näkyvillä.** Level of Detail (LOD) lisäys objekteihin on myös nopea tapa optimoida malleja. Tämä tarkoittaa sitä, että objektit jotka ovat lähellä pelaajaa (ja näkyvillä), näyttävät todellinen mallin high res tekstuureilla, mutta ne sama objektit, jotka ovat kaukana, muuttuvat automaattisesti low-poly versioiksi joiden tekstuurit ovat resoluutioltaan pienempiä.
- **Occlusion Culling.** Pelissä halutaan renderöidä ja piirtää asiat mitä oikeasti nähdään pelaajan perspektiivistä. Kun pelaaja katsoo isoa taloa, me ei haluta renderöidä objekteja sen talon takana, sillä se olisi täysin turhaa. Unityssä on automaattisesti käytössä Frustum Culling. Toisena vaihtoehtona on Occlusion Culling, joka vaatii enemmän aikaa, mutta optimoi peliä parhaiten!

Visuaalinen optimointi: Batching

- **Dynamic Batching:** Tämän tarkoituksena on yhdistää pienet mesh-verkot ja piirtää ne samalla draw call:illa. Unity voi automaattisesti hoitaa tämän liikkuville objekteille, mikäli niillä on samat materiaalit ja täyttää muut vaatimukset:
 - GameObjectit eivät saa olla peilattuja toisiinsa nähden (esim. GameObject A Scale on +1 ja GameObject B -1)
 - Mesh-verkkossa on enintään 300 kärkipistettä (vertice)
- **Static Batching:** Mahdollisuus vähentää minkä tahansa kokoisen geometrian draw call:ia, edellyttäen, että objekti ei liiku, käänny, muuta kokoa ja joka jakaa saman materiaalin. Se on tehokkaampi kuin Dynamic Batchin mutta käyttää enemmän muistia. Jotta Static Batchingiä voidaan käyttää, tulee GameObjecti muuttua staattiseksi



Visuaalinen Optimointi: Batching

Dynaaminen ja Staattinen batching toimii samankaltaisten materiaalien kanssa.

Tämän takia on hyvä yhdistää materiaalin tekstuurit samaan tekstuuri tiedostoon. PBR materiaaleissa on yleisimmin neljä high resolution tekstuuria yhdistettynä (tämä voi olla: base/albedo, metallic, normal, height, emission tai occlusion mapit)

Tämän takia voi olla hyvä yhdistää tekstuurit yhteen tai useampaan Texture Atlakseen, joka on 2D:ssä usein käytetty nimikkeellä “Sprite Sheet”.

Tätä voidaan siis myös hyödyntää 3D peleissä, jonka tarkoituksena on optimoida pelin tekstuureita 3D malleissa.



Visuaalinen Optimointi: Level of Detail (LOD)

Level of Detail:in idea on pienentää kaukana sijaitsevien objektien resoluutiota.

LOD:n avulla pystytään automaattisesti muuttamaan kauempia malleja eri “tasolle”, jotta pelin ei tarvitse renderöidä high detailed objektia, mitä pelaaja ei todennäköisesti edes huomaa. Mutta kun pelaaja haluaisi mennä sen objektin luokse, halutaan se silloin muuttaa tarkempaan tasoon.

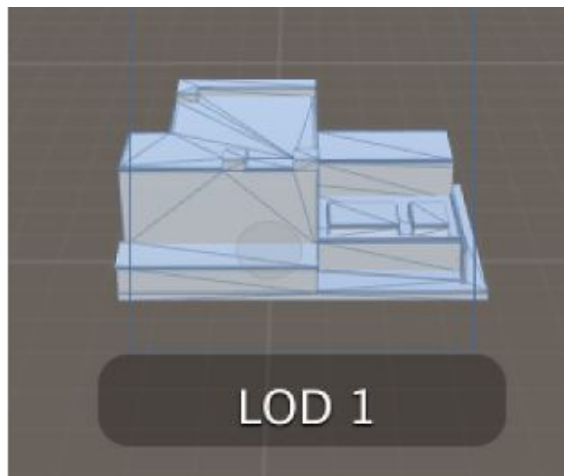
Unityssä on sisäänrakennettu LOD Systeemi, mutta se voi vaatia manuaalista työtä. Sen avulla pystytään lisäämään objektille eri LOD versioita, jotka muuttuvat etäisyyden perusteella

Jokainen LOD taso objektista on täysin oma mesh ja omilla tekstuureilla ja materiaaleilla, tämän takia LOD systeemi voi olla työläs.

Visuaalinen Optimointi: Level of Detail (LOD)

Jokainen LOD taso on edellistä tasoa epätarkempi, jonka tarkoituksena on optimoida scene pyörimään paremmin.

Alla olevassa kuvassa näkyy kaksi LOD tasoa, 0 joka on alkuperäinen versio mallista ja LOD 1 joka on siitä seuraava taso, joka voidaan määritellä haluamalla tavalla.



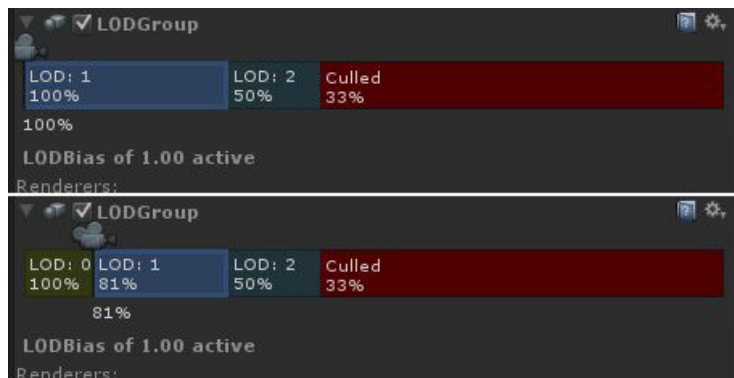
LOD 1 tulee voimaan tietyn etäisyyden perusteella (jota voi itse muuttaa asetuksista).

Esimerkiksi jos pelaaja seisoi LOD 0 talon vieressä, halutaan nähdä talon kaikki yksityiskohdat, mutta kun liikutaan kauemmas, automaattisesti vaihdetaan LOD 1 tasolle

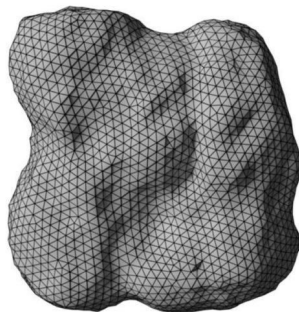
Visuaalinen Optimointi: Level of Detail (LOD)

Vasemmalla oleva kuva näyttää miltä “LODGroup” komponentti näyttää game objektissa.

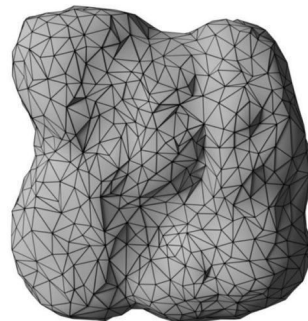
Oikea kuva näyttää eri LOD tasojen polygon vähennyksen



LOD 0



LOD 1



LOD 2

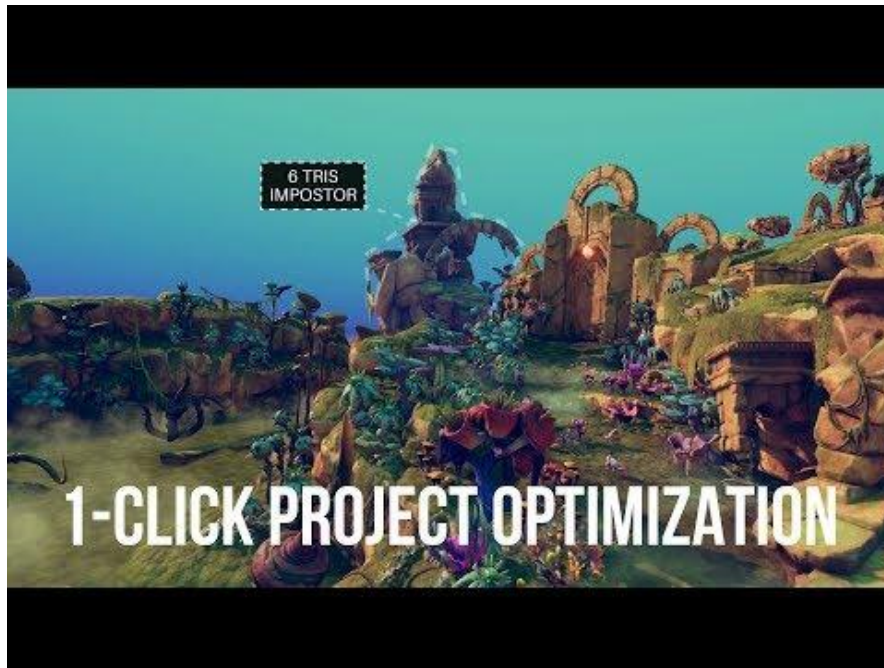


Visuaalinen Optimointi: Amplify Impostors

Amplify Impostors on todella hyvä assetti joka parantaa level of detailing lisäystä.

Se korvaa perinteisen LOD systeemin, ja käyttää sen sijaan billboardia hyödyksi. Billboard on LOD menetelmä, joka heijastaa objektin 2D tasolle.

[Amplify Impostors Unity Asset Store linkki.](#)

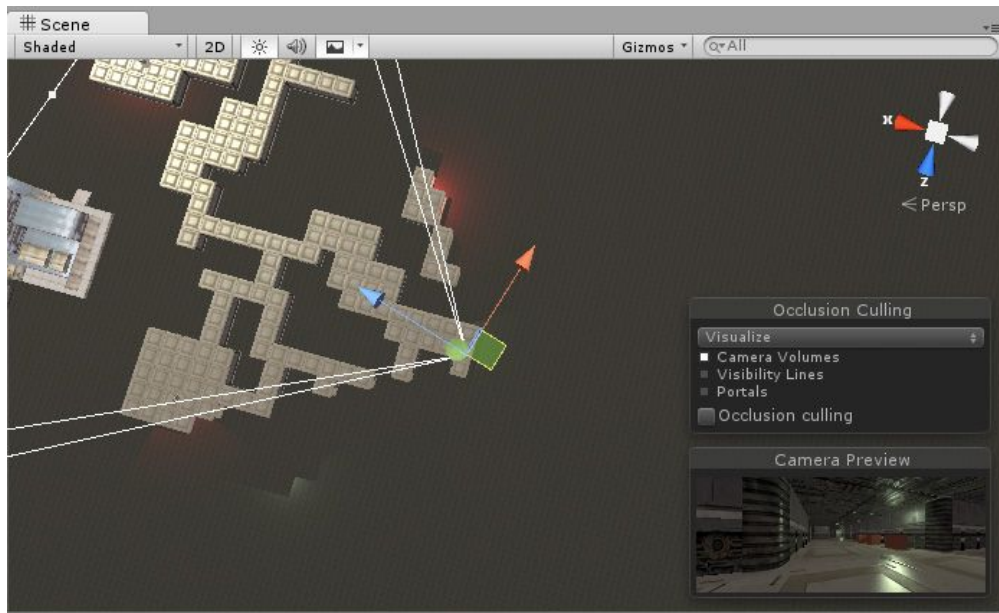


Visuaalinen Optimointi: Frustum Culling

Unityssä on automaattisesti kameroissa käytössä Frustum Culling, jonka avulla renderöidään kaikki mitä kameran edessä on.

Frustum Culling on siitä huono, että peli renderöi automaattisesti kaikki objektit jotka ovat kameran edessä (kameran asetusten mukaisesti).

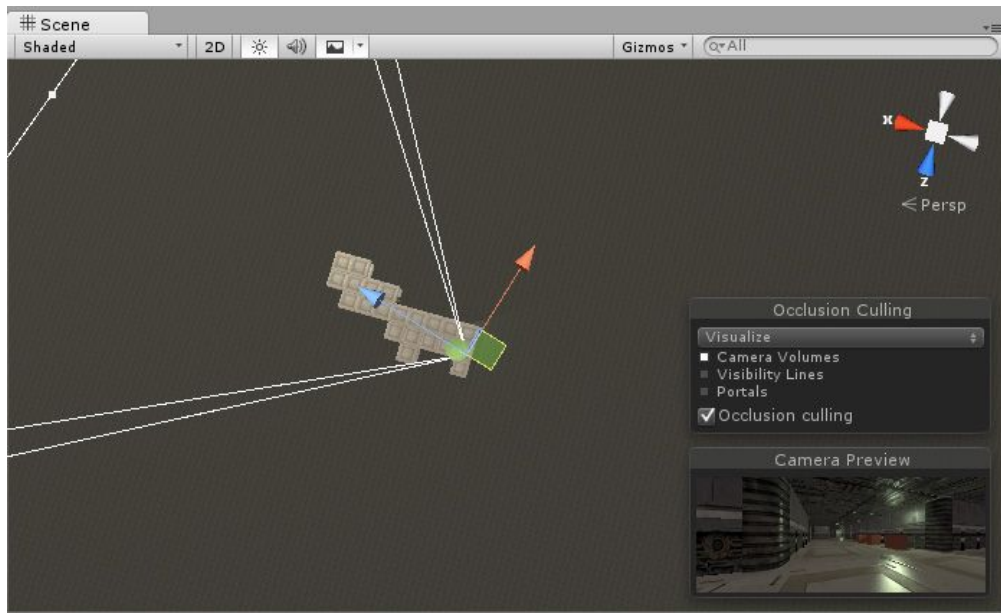
Peli siis renderöi objektit, jotka ovat myös täysin piilossa muiden objektien takana, joka lisää pelin draw calleja.



Visuaalinen Optimointi: Occlusion Culling

Occlusion Culling on toinen Object Culling menetelmä, jonka avulla kamera renderöi VAIN ne objektit, mitkä ovat näkyvillä. Se ei renderöi objekteja jotka ovat täysin piilossa muiden objektien takana.

Tämän menetelmän avulla voidaan säästää satojen objektien renderöinti, jotka ovat piilotettuna esimerkiksi seinien takana kokonaan. Kun kamera liikkuu toiseen huoneeseen, renderöi se tällöin kaikki näkyvät objektit.



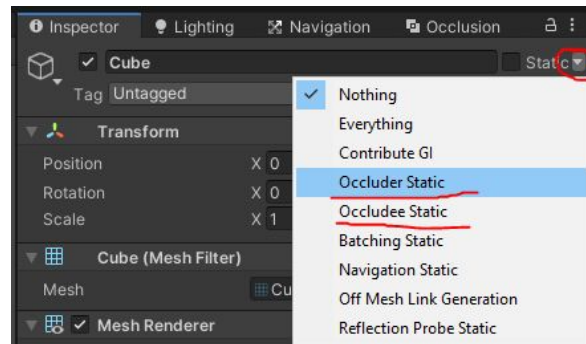
Visuaalinen Optimointi: Occlusion Culling

Occlusion Culling suositellaan käytettävän kentissä, joissa on pieniä tiloja erotettuna toisistaan, esimerkiksi talo jossa on omat huoneet.

Occlusion Cullingin käyttöönotto voi olla aikaa vievää, kun pitää jokainen objekti merkitä erikseen “Occluder” tai “Occludee”.

Occluder merkintä lisään objekteihin, jotka ovat tarpeeksi suuria piilottamaan muita objekteja. Occludee merkintä lisään kaikkeen jotka voivat mennä piiloon, esimerkiksi pienet objektit kuten tynnyrit.

*Huomioitavaa: Isot objektit voivat olla Occluder JA Occludee,
Mutta pienet kannattavat aina pitää pelkkänä Occludee:nä*

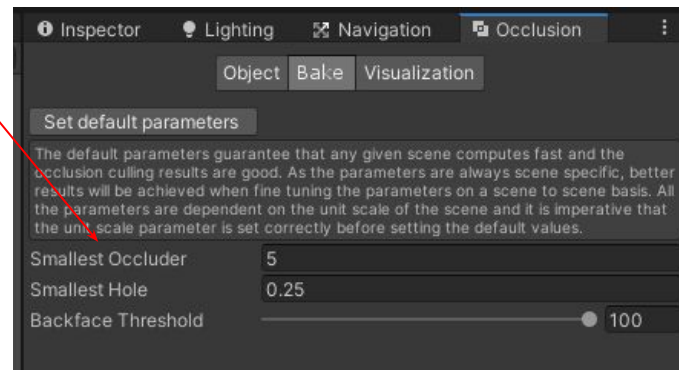


Visuaalinen Optimointi: Occlusion Culling

Occlusion Cullingin voi manuaalisesti lisätä erikseen jokaiseen objektiin, mutta nopein tapa on occlusion culling baking, joka on saatavilla Occlusion Culling välilehdeltä.

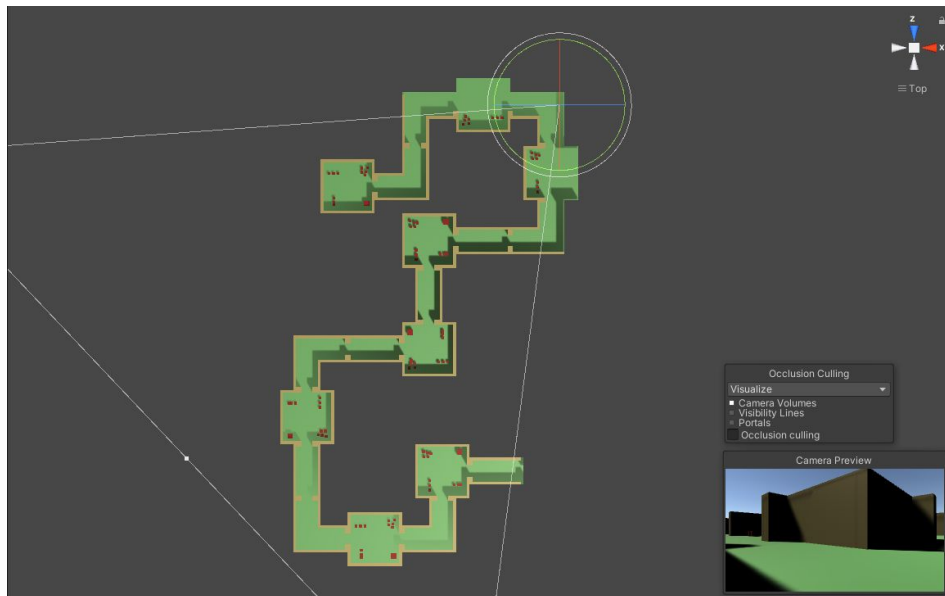
Occlusion Culling baking arvoja voidaan muuttaa halutessa, mutta on suositeltavaa pitää alkuperäiset arvot. Nämä asetukset muuttavat miten baking algoritmi merkkaa Occluder ja Occludee objektit.

Tässä, pienin Occluder koko on arvoltaan 5, joka tarkoittaa, että jokainen objekti joka on leveydeltään tai korkeudeltaan isompi kuin 5.

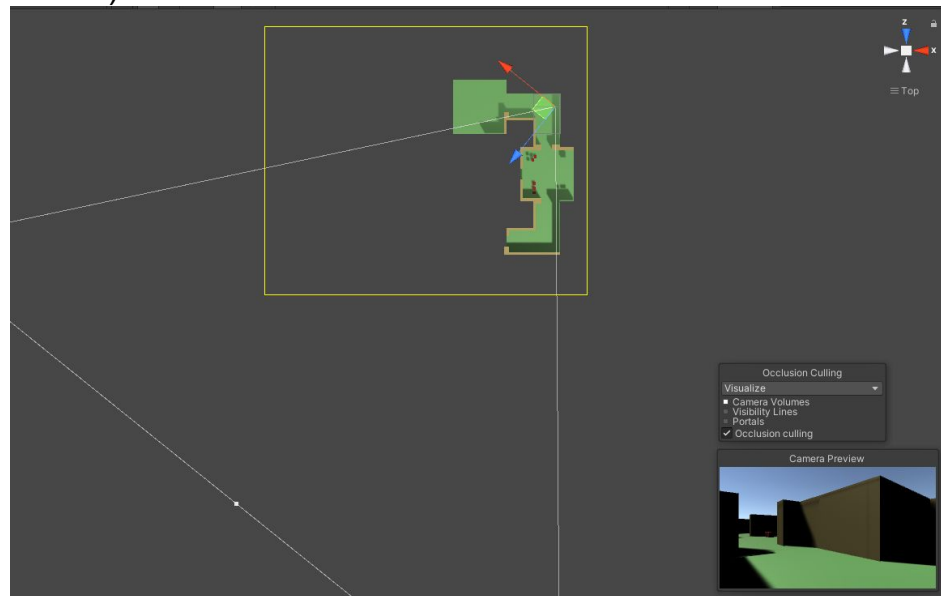


Visuaalinen Optimointi: Occlusion Culling

Frustum Culling (Automaattinen Object Culling Unityssä)



Occlusion Culling (Huoneet ja Corridorit Static jonka jälkeen Bake)



Occlusion Culling huomattavasti vähentää turhien GameObjectien renderöintiä, kun ne ei ole näkyvillä

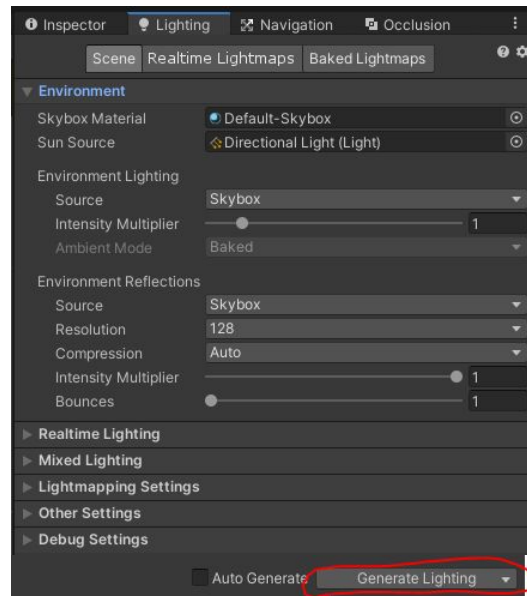
Visuaalinen Optimointi: Lighting

Valojen optimointi on myös yksi helppo tapa parantaa pelin optimointia.

Valojen beikkauksella valotus tallennetaan objekteihin scenessä. Tämä toimii parhaiten staattisten valojen kanssa, sillä ne valot eivät liiku scenessä. Valon pystyy beikkaamaan VAIN staattisiin objekteihin.

Valojen beikkaus voi olla myös todella haastava asia toteuttaa, sillä asetuksia beikkaukselle on paljon ja jokaisella on oma tärkeä arvo jota noudattaa.

Valot beikataan “Lighting” välilehdestä



U5 Realtime



Unity Realtime lighting
(ei beikattu)



Valot beikattu
scenen objektien
pintoihin

U5 Baked

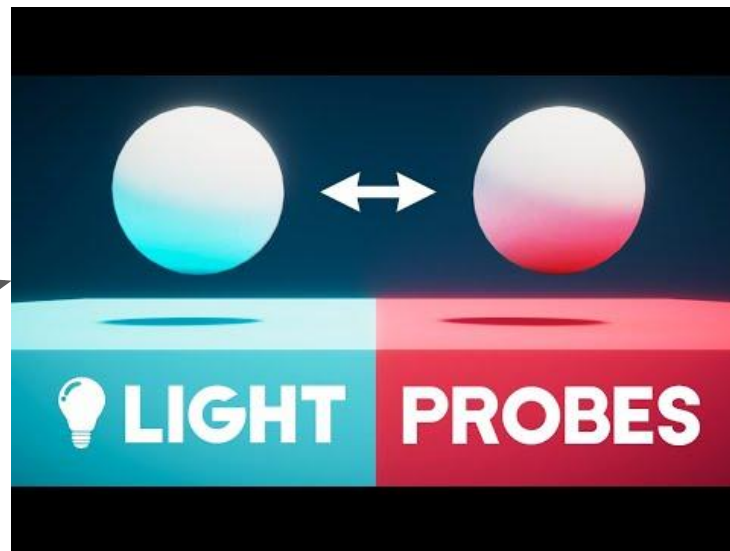
Visuaalinen Optimointi: Light Probes

Light probes on toinen tapa beikata valot sceneen.

Suurin ero lightmappiin on se, että light probe tallentaa tiedon valon kulkemisesta tyhjän tilan läpi scenessä, kun taas lightmap tallentaa valon osumakohdat pintoihin.

Light Proben avulla voidaan luoda tasainen vaihdos eri valotusten välillä:

Tarkka video aiheesta



Scriptien Optimointi

Scriptejä pystytään optimoimaan myös, joka tarkoittaa käytännössä sitä, että optimoimalla scriptejä vähennetään roskan luontia (Garbage Collection).

Scriptien optimointi voi viedä aikaa, mutta ne on hyviä käytäntöjä ottaa suoraan käyttöön ennen kuin luo tiettyjä scriptejä.

Suurin osa roskasta tulee kun luodaan ja tuhotaan objekteja pelin aikana, tämä voi olla esimerkiksi ase, joka luo (stantiate) luodin ja kun luoti osuu johonki pintaan, se tuhotaan (destroy).

Tämän voi muuttaa kätevästi Object pooliksi, jonka tarkoituksena on vähentää uusien objektien luontia ja tuhoamista. Sen sijaan, että luodit luodaan ja tuhotaan pelin aikana, niin luodaan tietty määrä luoteja pelin alussa, joita sitten otetaan ja poistetaan käytöstä (Enable ja Disable)

Scriptien Optimointi: Object Pool

Vieressä on esimerkki, miten toteutetaan Object Poolin aseiden ampumista varten.

Pelin alussa luodaan 20 luotia, jotka suoraan pistetään pois käytöstä (obj.SetActive(false)), jokainen luoti lisätään “bullets” listaan.

Kun halutaan ampua, haetaan “bullets” listasta uusi luoti, joka ei ole aktiivinen, siirretään se aseeseen piipun kohdalle ja asetetaan ase aktiiviseksi. Kun luoti osuu, asetetaan luoti jälleen SetActive(false) tilaan.

```
5 public class ObjectPool : MonoBehaviour {
6
7     public GameObject bullet;
8     public int pooledAmount = 20;
9     private List<GameObject> bullets;
10
11     // Use this for initialization
12     void Start () {
13         bullets = new List<GameObject> ();
14         for (int i = 0; i < pooledAmount; i++) {
15             GameObject obj = Instantiate (bullet) as GameObject;
16             obj.SetActive (false);
17             bullets.Add (obj);
18         }
19         InvokeRepeating ("Fire", .05f, .05f);
20     }
21
22     void Fire ()
23     {
24         for (int i = 0; i < bullets.Count; i++) {
25             if (!bullets [i].activeInHierarchy) {
26                 bullets [i].transform.position = transform.position;
27                 bullets [i].transform.rotation = transform.rotation;
28                 bullets [i].SetActive (true);
29                 break;
30             }
31         }
32     }
33 }
```

Scriptien Optimointi: Muita käytänteitä

- Vältä Update - metodin käyttöä, mikäli mahdollista. Update käydään läpi pelin jokaisella framella, joten jos peli pyörii 60fps -> update käydään läpi 60 kertaa sekunnissa.
- Tallenna (cache) komponentit omiksi muuttujiksi. Jos käytät scriptissä Rigidbody, Animator tai muita komponentteja useammin kuin kerran (esim. Update metodeissa käytät GetComponent), kannattaa ne tallentaa mieluummin omaksi muuttujaksi joko scriptin alussa tai metodin alussa.
 - Esim: Rigidbody kannattaa muuttaa *“private Rigidbody rb”* muuttujaksi ja hakea Awake - tai Start metodissa *“rb = GetComponent<Rigidbody>();”*
- Vältä *“GameObject.Find”* tai *“GameObject.FindWithTag”* käyttöä, sillä *“Find”* käy läpi **jokaisen** GameObjectin scenessä. Pyri referoimaan komponentit jota käytät inspectorin kautta mieluiten *“[SerializeField] private component”* tai *“public component”* muodossa.
Tip: “Camera.main” on sama kuin “FindWithTag(“MainCamera”)”, joten vältä turhaa toistoa sen käytölle

Fysiikan Optimointi

Fysiikan optimointi perustuu artistien ja ohjelmoijan yhteistyöhön.

- Ohjelmoijan käytäntöihin kuuluu scriptien optimointia Rigidbodyn ja Raycastin kanssa.
 - Rigidbody objektit voidaan asettaa “nukkumis” tilaan silloin kuin niitä ei esimerkiksi ole näkyvillä tai ne ovat paikallaan, tämän avulla vältetään turhat fysiikan laskelmat.
 - Raycastien käyttö ei ole suorituskyyvylle niin raskasta, mutta kannattaa kuitenkin välttää niiden turhat käytöt.
- Artistit voivat parantaa pelin fyysikan suorituskyykyä luomalla low-poly collidereita high-poly malleille. Jos high-poly objekti käyttää “mesh collider” komponenttia, voi se olla raskasta fyysikka laskelmien kanssa, sillä colliderin polygon määrä on suuri. Tämän pystyy helposti kiertämään luomalla low-poly versio mallista ja käyttäen low-poly mallin mesh collideria high-poly mallin päällä.
 - *Tip: Tämä toimii hyvin jos on LOD tasoille tehty omat mallit*

Low-poly mallin “Mesh Collider” lisätään High-poly mallin päälle

