Programiranje v okolju Unity 3D

Rok Kos

17. april 2016

Kazalo

Slike

Povzetek

V svoji projektni nalogi bom predstavil rokovanje z programskim okoljem Unity 3D. Kot primer bom vzel svojo aplikacijo, ki sem jo ustvaril z Unity-jem, ki predstavlja vodiča po Gimnaziji Vič.

Ključne besede: Unity3D

1 SET UP

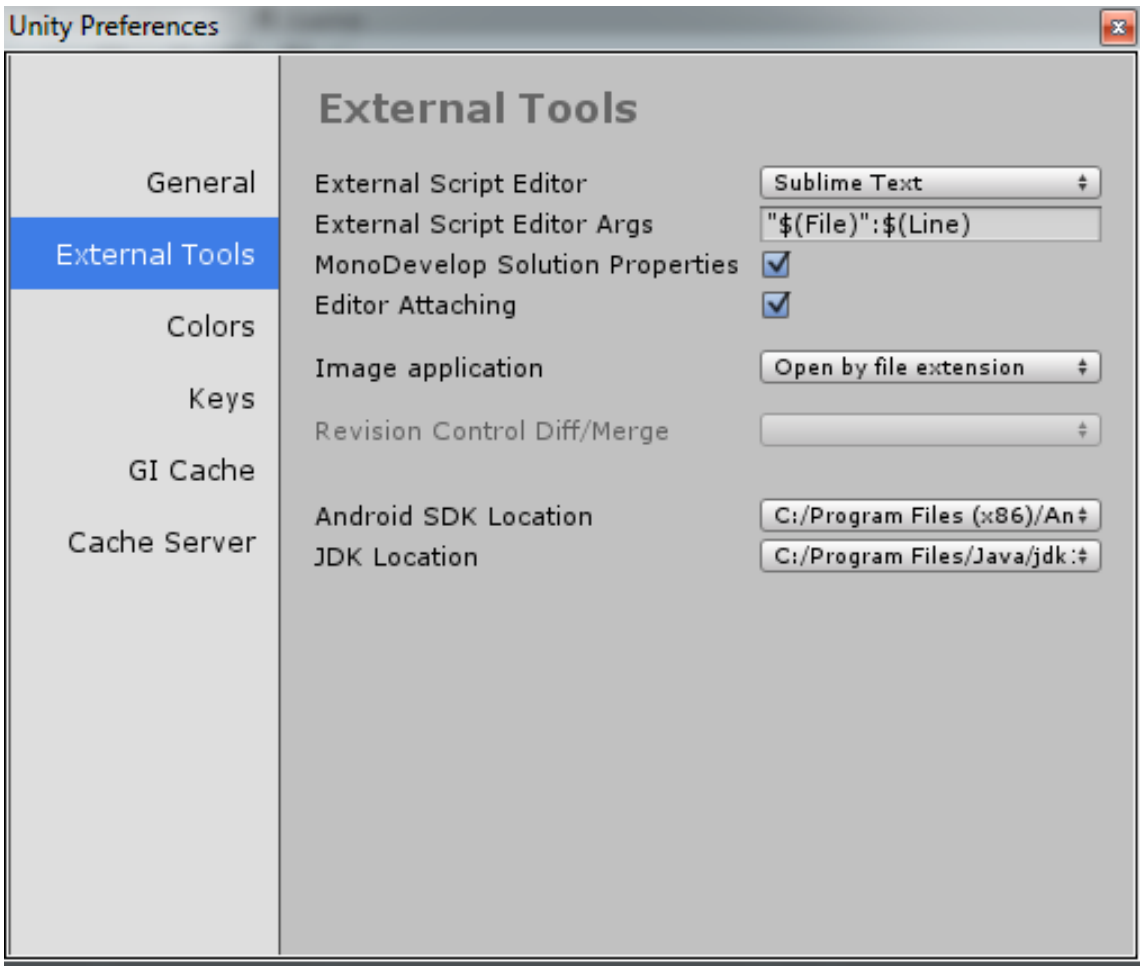
1.1 Editor

Kot vsako razvojno okolje ima tudi Unity 3D svoj privzeti editor. To je MonoDevelop, ki podpira večplatformski razvoj(to pomeni, da lahko isto kodo pišemo za različne operacijske sisteme Linux, Windows, Mac, Android, IOS, PS3 itd.). Podpira naslednje jezika:

- C#
- F#
- VisualBasic
- . . .

Ima tudi integrirano dopolnjevanje kode(code auto-completion) in svoj debuger. Sam sem kar nekaj časa uporabljal MonoDevelop in v njemu razvijal, ampak mi pri je pri njem vedno nekaj manjkalo. Zdelo se mi je, da je njihov code auto-completion vsiljeval svoje stvari in da mi highliter ni vedno obarval kakšnih klasov. Zato sem se odločil, da bom začel uporabljati meni ljubši urejevalnik besedila to je Sublime Text 3. Pri tem urejevalniku sem dobil veliko svobodo pri urejanju, iskanju, popravljanju ter pri samem urejanju urejevalnika. To sublime omogoča s svojim Package Controlom, ki se ga da inštalirati preko te strani: https://packagecontrol.io/installation. Preko Package Controla lahko inštaliramo dodatne plugine in snippete za sublime, ki izboljšajo samo delovanje tega. O tej temi bi se dalo še veliko govoriti zato jo bom pustil za drugič. Seveda moramo Unityju povedati, da naj svoje datoteke odpira z sublimom. To naredimo tako:

1. Inštaliramo Sublime Text 3/2 preko te strani
2. Gremo v **Edit → Preferences → External Tools**(kot lahko vidimo v spodnji sliki) in spremenimo na[urejevalnik
3. Spremenimo **External Script Editor Args** v "$(File)":$(Line) zato, da bo sublime skočil v vrstico, kjer je error
4. Sedaj bi nam moral Unity odpreti sublime, ko dvokliknemo na datoteko, ki je skripta
5. V sublimu bomo še spremenili, katere datoteke hočemo videti v projeknem drevesu. To naredimo tako, da gremo v **Project → Save Project As** in notri vtipkamo tole kodo:



Slika 1: Sublime Text Set Up

```
{
        "folders":
        [
                {
                        "path": "Assets/Scripts",
                        "file_exclude_patterns":
                        [
                                "*.dll",
                                "*.meta"
                        ]
                }
        ]
}
```

in potem shranimo datoteko. To nam scrije vse nepotrebne .meta in .dll datoteke

6. S Package Controlom na koncu še inštaliramo dodatne snippete, ki nam pomagajo pri auto-completion in barvanju kode. Vse potrebne package najdemo na tej strani http://wiki.unity3d.com/index.php/Using_Sublime_Text_as_a_script_editor, kjer so tudi bolj podrobna navodila za celoten setup Sublime Text-a.

Kot zanimivost pa bi vam rad pokazal, kako enostavno se da narediti sni-

ppete v Sublime Text-u, ki se potem sprožijo ob določenem zaporedju tipk in nakoncu tabulatorja.

```
<snippet>
        <content><![CDATA[
//Author: Rok Kos <rocki5555@gmail.com>
//File: $TM_FILENAME
//File path: $TM_FILEPATH
//Date: $1
//Description: $2
]]>
        </content>
        <!-- Optional: Set a tabTrigger to define how to trigger the snippet -->
        <tabTrigger>sig</tabTrigger>
        <!-- Optional: Set a scope to limit where the snippet will trigger -->
        <!-- <scope>source.python</scope> -->
</snippet>
```

1.2 Plugini

V Unity-ju si lahko pomagamo z različni skriptami, ki jih ustvarimo izven Unity-ja. Tem skriptam pravimo plugini. Te nam lahko pomagajo pri samem urejanju projekta, olajševanju in otomatiziranju nekaterih stvari ali pa nam dodeljuje kakšne funkcije za prav posebno platformo. Prvim pravimo Managed plugins, drugim pa Native plugins [Tec]. Managed plugine ponavadi pišemo v C#, saj uporavljajo samo knjižnico .NET. Te plugini se skompajlajo v dinamične knjižnice(dynamical linked library) oz. DLL, ki jih potem vključimo v projekt. Native plugini pa so napisani v C, C++, Objective-C in ostalih jezikih. To pomeni, da damo možnost Unity, da lahko npr. kliče kodo Java ali C++. Seveda je glavna prednost tega, da napišemo svoje knjižnice za določeno platformo npr. za IOS svojo in za Android svojo.

2 SHADERS

2.1 Definicija

Shaderji so skripte, ki nam povejo, kako naj se vsak piksel na zaslonu zrendera. To seveda ni samo od tega kako so napisani shaderji ampak tudi od tega kakšne materiale in teksture uporabljamo. V shaderjih so zapisani algoritmi, ki uporabljajo vektorje za svetlost in barvo in s tem povejo kako naj se obarva piksel. Z verzijo Unity 5 je prišel ven tudi njihov Standart Shader, ki je zelo uporaben in se lahko zelo prilagaja. Ampak v tem projektu bom predstavil, kako napisati lasten shader.

2.2 Lastnosti shaderjev

Tole je prikaz enostavnega shaderja [Jos]:

```
Shader "DT\Basic\SimpleShader"{
        SubSHader{
```

```
3                   Tags = {"RenderType" = Opaque}
4                   CGPROGRAM
5                   #pragma surface surf Lambert
6                   struct Input{
7                           //(1.0, 1.0, 1.0, 1.0) R, G, B, A
8                           float4 color : COLOR;
9                   };
10                  void surf(Input IN, inout SurfaceOutput o){
11                          o.Albedo = 1;
12                  }
13                  ENDCG
14          }
15          Fallback "Diffuse"
16  }
```

3 CLASSES

3.1 Objektno programiranje

Kot nam že samo ime pove temelji objektno programiranje na objektih, ki programiranje bolj približujejo človeku. Na tem principu deluje tudi Unity s svojim jezikom C#. V vsaki igri kot tudi v realnem svetu imamo stvari z podobnimi lastnostmi ali pa celo z enakimi atributi. Tukaj v igro vstopijo objekti. Te nam pomagajo specificirati atribute določenega predmeta. Vzemimo npr. svetilko. Svetilka ima žarnico, baterijo in stikalo za vklop in izklop. Pozna tudi metodo vklop in izklop. Lahko bi vzeli tudi drugi klas npr. zrcalo, ki bi spet imelo drugačne atribute. Ampak glavna ideja za tem je, da imamo lahko sedaj v nasi igri poljubno mnogo instanc teh svetilk in zrcal in zato nismo rabili pisati code za vsakega posebej. Lahko bi rekli, da imamo glavni klas in iz njega samo štancamo ven nove objekte. Pri klasih pa pridemo tudi do dedovanja, dostopnosti(public, protected, private) in polimorfizma.

3.2 Pisanje classov

Primer mojega klasa, ki sem ga uporabil v svoji aplikaciji in predstavlja šolo.

```
1  //Author: Rok Kos
2  //Date: 05.12.2015
3  //Description: Class for classroms
4
5  using UnityEngine;
6  using System.Collections;
7
8  public class Sola : MonoBehaviour {
9
10         public class Ucilnica{
11                 /*
```

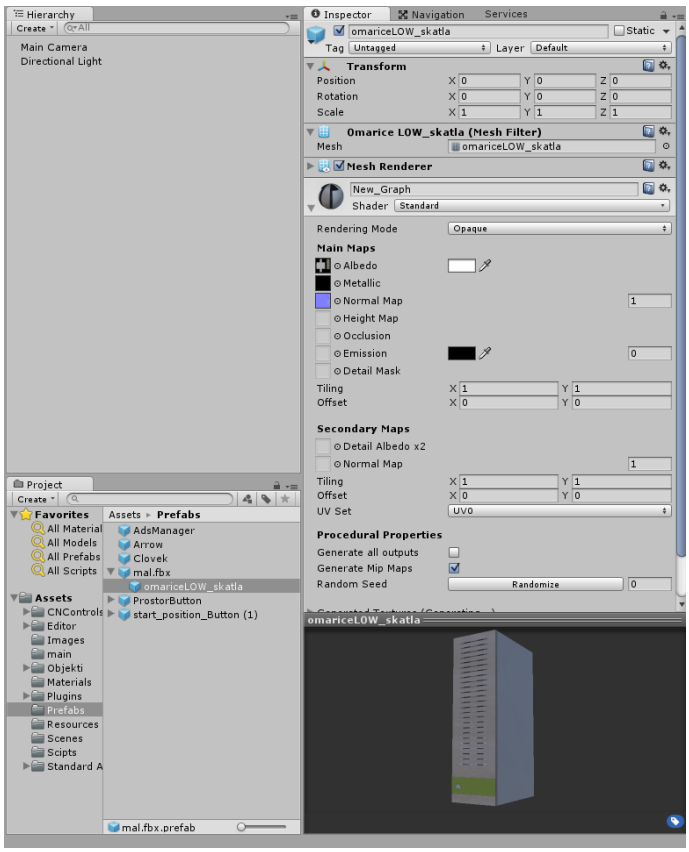
```
12                  * To je class v katerem bomo definirali pozicijo vsake ucilnice,
13                  * ime ucilnice, nfc tag ucilnice.
14                 */
15                 public Vector3 pozicija;
16                 public Quaternion rotacija;
17                 public string ime_Ucilnice;
18                 public int NFC_tag;
19
20                 //Construktor, to so default vresnosti
21                 public Ucilnica(){
22
23                         pozicija =  new Vector3(0f,0f,0f);
24                         rotacija = new Quaternion(0f,0f,0f,0f);
25                         ime_Ucilnice = "UNKNOWN";
26                         NFC_tag = -1;
27                 }
28                 //Construktor z vsem
29                 public Ucilnica(Vector3 vnesena_pozicija,Quaternion vnesena_rotacija,
30                         string vneseno_ime_Ucilnica, int vnesen_NFC_tag){
31                         pozicija = vnesena_pozicija;
32                         rotacija = vnesena_rotacija;
33                         ime_Ucilnice = vneseno_ime_Ucilnica;
34                         NFC_tag = vnesen_NFC_tag;
35                 }
36                 //Construktor brez rotacije
37                 public Ucilnica(Vector3 vnesena_pozicija,
38                         string vneseno_ime_Ucilnica,
39                         int vnesen_NFC_tag){
40                         pozicija = vnesena_pozicija;
41                         rotacija = new Quaternion(0f,0f,0f,0f);
42                         ime_Ucilnice = vneseno_ime_Ucilnica;
43                         NFC_tag = vnesen_NFC_tag;
44                 }
45
46                 public Vector3 vrni_pozicijo(){
47                         return pozicija;
48                 }
49
50                 public Quaternion vrni_rotacijo(){
51                         return rotacija;
52                 }
53
54                 public string vrni_ime(){
55                         return ime_Ucilnice;
56                 }
57
58                 public string vrni_class(){
59                         return ime_Ucilnice + pozicija.ToString ("G4") + "\n";
```

```
60                      }
61                      //Destructor
62                      ~Ucilnica(){
63
64                      }
65              }
66  }
```

4 PREFABS

4.1 Definicija

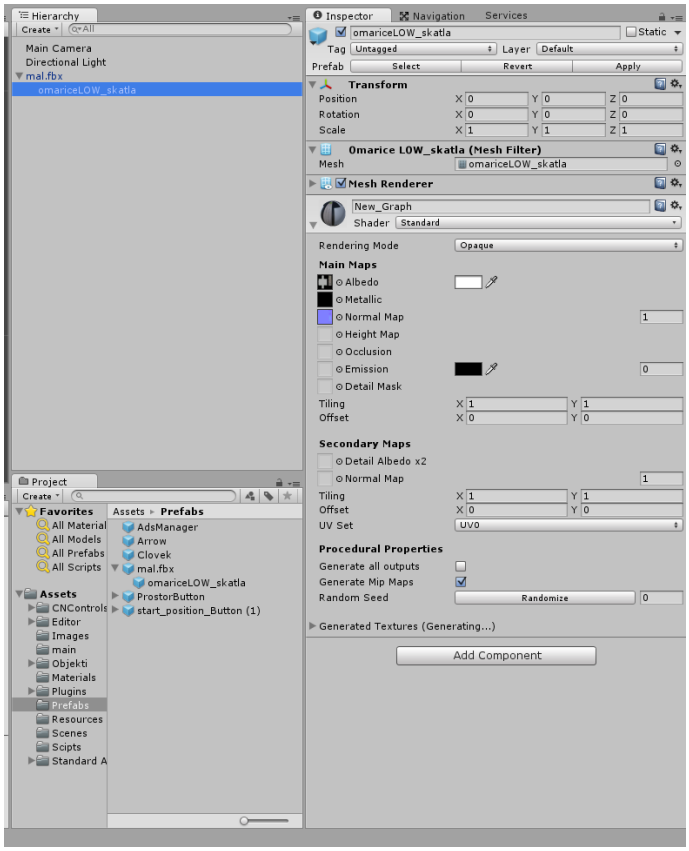
Prefab si v Unity okolju lahko predstavljamo, kot klas GameObject-a, na katerega so pripete scipte, texture, slike, rigidbody-ji itd. To nam da možnost, da shranimo celotno GameObject-e, ki jih lahko kasneje uporabimo v sceni oz. naredimo poljubno instanc. Za boljšo pojasnitev lahko vzamemo primer drevesa. Najprej naredimo eno drevo z vsemi atributi ga shranimo in nato lahko instance v sceni in s tem dobimo efekt gozda. To lahko naredimo tudi med izvajanjem same igre, kar nam da še večje možnosti. Enako lahko naredimo za nasprotnike, zidove, zgradbe itd.



Slika 2: Prefab

4.2 Uporaba

Ena izmed najbolj uporabnih lastnosti prefabov je, da če spremenimo en element in potem te sprembe shranimo se spremenijo vsi objekti. Z enim gumbom lahko naredimo globalno spremembo. To naredimo tako, da gremo na prefab v sceni ali v folderju, kjer se nahaja ter v inspektorju zgoraj pod imenom in vidimo 3 možnosti: Select, Revert, Apply. Naredijo ravno to. Revert od naredi vse globalne spremembe, apply pa naredi gloabno spremembo. Prefabe se da zelo enostavno instancirati med igro.



Slika 3: Prefab spremembe

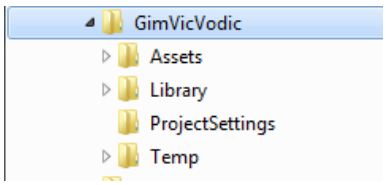
Kot spodaj vidimo. Narejenim objektom lahko tudi spreminjamo njihove lastnosti med igro.

5 IMPORTING

5.1 Lokacija in tipi datotek

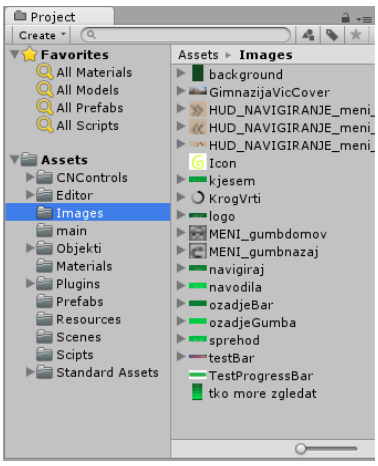
Unity pokriva velik spekter različnih formatov iz različnih programov, kot so npr. Blender, Maya, Photoshop itd. Vsi datote se nahajajo v Asset mapi, ki je potem tudi vidna v samem Unity editorju. V spodnji sliki lahko tudi vidimo tipično strukturo map, ko se naredi nov Unity projekt.

Pri tipičnih formatih datotek kot so npr. JPG, PSD, PNG, MA, BLED, FBX, WAW itd. lahko datoteko, kar kopiramo v asset mapo ali jo celo povlečemo



Slika 4: Struktura map

v prav specifično mapo kjer hočemo in se bo prikazala v Unity urejevalniku. Takole so potem urejene datoteke v samem unity-ju.
Ko mi kopiramo datoteko v asset mapo, unity poskrbi, da glede na konč-



Slika 5: Pregled map v unity-ju

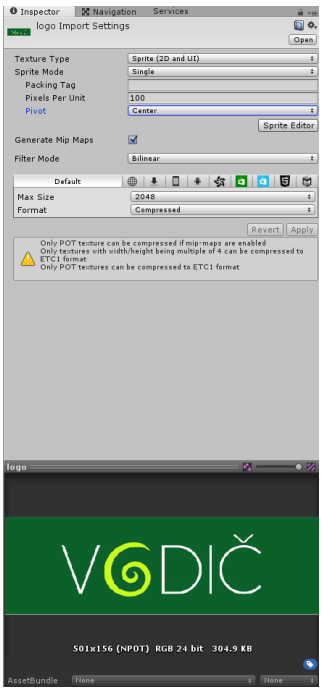
nico oz. tip se pravilno vključi, compresira in nakoncu postane tudi presentativna v urejevalniku. Pri tem lahko tudi sami spreminjamo določene parametre:

- kakšnega tipa je datoteka(sprite, textura, normal map itd.)
- kako gosto naj bojo piksli(če je slika)
- kaliko prostora lahko zasede na disku
- kako naj se kompresira

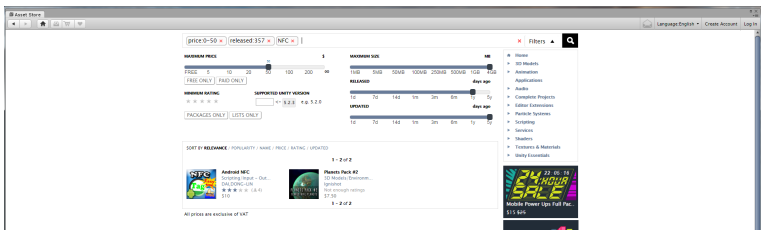
Nakoncu je ponujena možnost, da za različno platformo(Android, IOS, PC, PS3) različno nastavimo import nastavitve. Spodaj lahko na primeru vidimo kakssne opcije so namna voljo.

5.2 Asset store

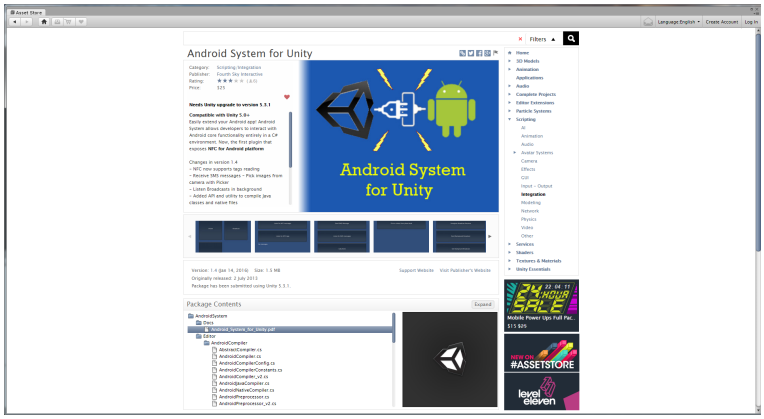
To je trgovina, kjer so brezplačni in plačljivi asseti narejeni s strani Unity Technologies in velike skupnosti razvijalcev. Pod te assete spadajo vse od textur, 3D modelov, slik, projektov itd. Po trgovini se lahko zelo lahko navigira in išče za prav specifične assete.
Vsak asset ima potem tudi opis kaj vsebuje, ceno, kakšne slike in mogoče tudi video implementacije ali primer projekta.



Slika 6: Import nastavitve

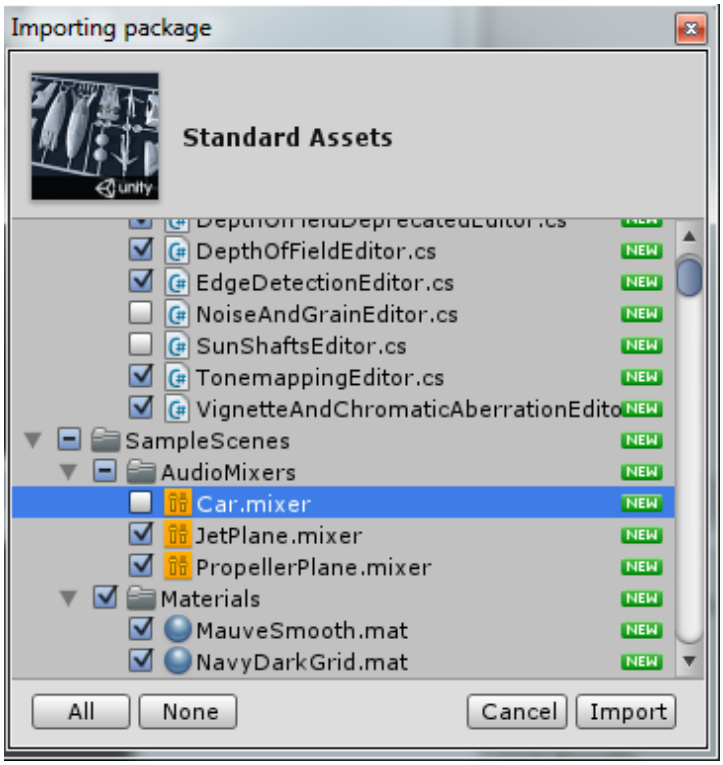


Slika 7: Iskanje assetov



Slika 8: Predstavitev asseta

Glavna prednost je da se zelo hitro vključijo v trenutni projekt in za njih ni potrebno spreminjati nastavitev. Razvijalec lahko tudi izbira katere stvari hoče prenesti v projekt in katere ne.



Slika 9: Izbira kaj vključiti

5.3 Standart Assets

To je velika skupina assetov, ki je spravljena v en paket. Unity je naredil ta paket, da lahko začetniki hitro in brez večjega truda naredijo enostavno aplikacijo. V tem paketu je tudi veliko primerov uporabe teh assetov. Zelo olajšajo kakšne stvari za katere, bi enemu samemu razvijalcu vzelo veliko časa, da bi jih implementiral. V mojem projektu sem tudi uporabila Joystick implementacijo iz standart assetov. To sem naredil tako, da sem šel **Assets → Import Package** in izbral področje katerega sem želel(v moje primeru je to bilo CrossPlatformInput).

In nato se odpre okno, ki ga lahko vidimo tudi na sliki 9.

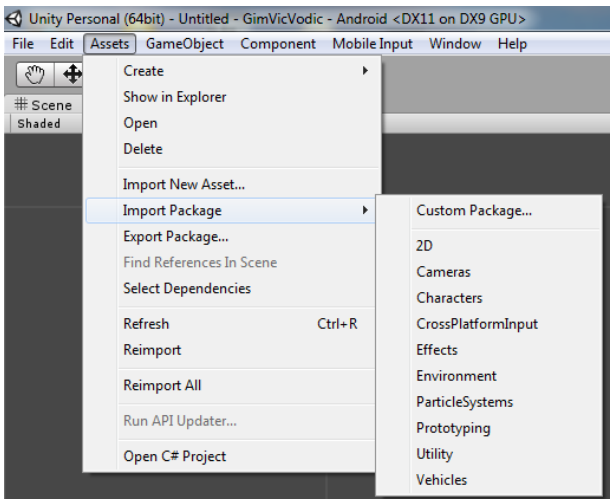
Je pa tudi velika past saj z vsako novo verzijo Unity-ja pridejo novo standart asseti, ki se lahko obnašajo drugače, kot prejšnji zato moramo vedno pogledati za kakšno verzijo, so bili izdani asseti.

6 BUILDING PROJECT

7 VERSION CONTROL

7.1 Integracija v Unity-ju

Unity ponuja z v samem editorju dva različna version control sistema. To sta Perforce(https://www.perforce.com/) in Plastic SCM(https://www.plasticscm.com/). Sedaj se pojavi tudi vprašanje zakaj bi uporabljali



Slika 10: Standart Assets

kateri koli version control sistem. Ena izmed ključnih lastnosti je, da lahko spremljamo potek sprememb in da lahko kadarkoli v procesu se vrnemo na prejšnjo rešitev. Da nam pa tudi funkcije, kot npr. kdo v ekipi je naredil spremembo, kakšne so spremembe med samimi commiti in lažje vidimo kje so nastale napake ali hrošči. V mojem seminarski nalogi nebom uporabljal teh dveh orodji, ampak bom uporabljal čisto osnovni GIT, kot stran oz. server kjer bo držal bo uporabil GitHub(https://github.com/). Najprej bomo seveda nastavili Unity nastavili tako, da bomo lahko primerjali datoteke med sabo. To naredimo tako da **Edit → Project Settings → Editor menu**. Tukaj nastavimo, da se vse datoteke nujno spremenijo v .txt datoteke.

Saj po default nastavitvah ima Unity nekatere objekte zapisane v binarnem sistemu, pri katerih je skoraj nemogoče spremljati spremembe.

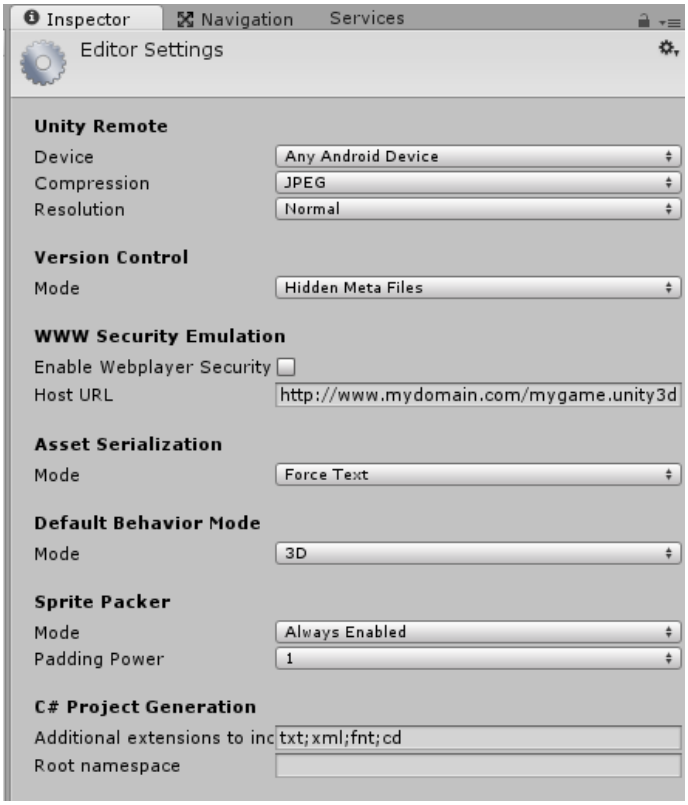
7.2 GIT

Git je brezplačni in odprtokodni version control sistem, ki ima veliko uporabnih funkcij. Navedel bom samo nekatere, ki so najbolj pogoste za uporabo. Vse funkcije imajo pred ukazom besedo git. Funkcije:

- **init** : Naredi prazeno git mapo ali pa na novo reinicializira že obstoječo mapo.
  Comanda bi zgledala nekako takole:

```
$ git init
```

- **clone** : Klonira že obstoječo mapo v trenutno mapo
  Najbolj pogosto se ta komanda uporablja, da kloniramo mapo ali projekt iz kakšne strani, na naš računalnik. Lahko pa tudi lokalno pri sebi kloniramo iz ene mapo v drugo.
  Za kloniranje map iz url naslova git podpira ssh, git, http in https. Url naslov mora biti v eni izmed takih oblik:
  - ssh://[user@]host.xz[:port]/path/to/repo.git/
  - git://host.xz[:port]/path/to/repo.git/



Slika 11: Version control set up

– http[s]://host.xz[:port]/path/to/repo.git/

In ko vse skupaj sestavimo bi bil primer komande takšen:

```
$ git clone
↪ https://RokKos@bitbucket.org/RokKos/gimvicvodic.git
```

- **config** : Nastavi globalne ali lokalne spremeljivke mape.
  Ne bom se poglabljav v vse možnosti uporabe te komande ampak bom nastel samo nekaj osnovnih. V bistvu bosta nas zanimali samo dve in to sta:

```
$ git config --global user.name "Rok Kos"
$ git config --global user.email RokKos@gmail.com
```

- **add** : Doda datoteko ali skupino datotek k delovnemu drevesu.
  Ta komanda doda datoteka, tako da se upoštevajo in beležijo vse spremembe narejene na njej. Na začetku, ko inicializiramo mapo, hočemo da se upoštevajo vse nato pa dodajamo posamezne datoteke.
  Par primerov uporabe te komande:

```
$ git add --all
$ git add Documentation/\*.txt
$ git add Assets/ControllerScript.cs
```

- **commit** : Zapiše spremembe v mapi

To komando pokličemo, po komandi add ali pa ko mergamo skupaj dve veji. Poleg tega imamo možnost, da poleg commita dodamo tudi sporočilo oz. poročilo kaj smo naredili. Primer:

```
$ git commit -m "Meseage of commit"
```

- **push** : Naloži datoteke na oddaljen strežnik
  To komando pokličemo čisto nakoncu, ko smo dodali vse datoteke, jih zapisali(commitali). Mamo pa tudi možnost, da pushamo na različne veje(branche).
  Primer:

```
$ git push origin master
```

- **pull** : Potegne datoteke iz oddaljenega strežnika in jih združi z lokalno mapo.
  Ta komanda požene git fetch komando, ki dobi podatke iz strežnika in potem za tem požene git merge, ki združi lokalno verzijo z verzijo iz oddaljenega strežnika. Primer:

```
$ git pull origin
```

- itd.

[Tec] [Ala14] [Tho15]

Viri in literatura

[Ala14] THORN Alan. *Pro Unity Game Development with C#*. Vol. 1. izdaja. New York: Apress, 2014. ISBN: 9781430267461.

[Tho15] FINNEGAN Thomas. *Learning Unity Android Game Development*. Vol. 1. izdaja. Birmingham: Packt Publishing, 2015. ISBN: 9781784394691.

[Jos] KINNEY Joshua. *Introduction to Scripting Shaders in Unity*. URL: http://www.digitaltutors.com/tutorial/1438-Introduction-to-Scripting-Shaders-in-Unity (visited on 2015).

[Tec] Unity Technologies. *Unity Manual*. URL: http://docs.unity3d.com/Manual/index.html (visited on 2015).