PROJEKTIRANJE 3D MODELA U BLENDERU I UNITY GAME ENGINEU

Mehmetović, Filip

Veleučilište u Karlovcu
Strojarski odjel
preddiplomski stručni studij *Mehatronike*

Filip Mehmetović

Projektiranje 3D modela u Blender-u i Unity game engine-u

3D modeling in Blender and Unity game engine

Završni rad

Karlovac, 2019. godina

Veleučilište u Karlovcu
Strojarski odjel
preddiplomski stručni studij *Mehatronike*

Filip Mehmetović
0112615018

Projektiranje 3D modela u Blender-u i Unity game engine-u

3D modeling in Blender and Unity game engine

Završni rad

mentor: Mr. sc. Vedran Vyroubal

Karlovac, 2019. godina

Izjavljujem da sam završni rad izradio samostalno.
Prezentirano znanje je okvir osobnog stečenog znanja tijekom studija, te također intuitivno snalaženje i traženje odgovora pomoću navedene literature.

Zahvaljujem se profesoru i mentoru *Mr.sc. Vedranu Vyroubalu* na svim savjetima i pružanju stručne pomoći tijekom izrade rada.

Filip Mehmetović

Sažetak

Završni rad sastoji se od dva dijela koje ću opisati kroz sažeti koncept.

Prvi i glavni dio je izrada 3D modela u programu zvanom *Blender* [1] , opširniji opis slijedi u sljedećem poglavlju. Odlučio sam kreirati svoje modele u *Blender-u* iz razloga jer je vrlo kompatibilan s *Unity game engine-om* [2]. Također lakše je uređivati 3D modele u *Blender-u* [1] te precizno skalirati objekte i dodavati teksture itd.

Drugi dio se sastoji od izrade simulacije u *Unity game engine-u* [2]. Odlučio sam ukomponirati svoje 3D modele unutar *Unity-a* [2] da bih pokazao kako se suhoparni modeli iskoriste za realizaciju te im se doda stvarna fizika prostora. *Unity* [2] također ima *„World editor“* ili projektiranje svijeta u realnom vremenu, ali se sastoji od osnovnih oblika koje nije moguće modelirati.

Unutar *Unity-a* [2] se ostvaruje logika objekata unutar simulacije. Logika se svodi na objektno programiranje u C#-u [3] koje ću detaljnije objasniti kasnije.

Odlučio sam se na ovu temu završnog rada s razlogom da bih usavršio objektno programiranje i 3D modeliranje. S obzirom na to da sam unutar svojeg smjera imao *osnovno programiranje u C jeziku* [4] i 3D modeliranje u *Autocadu* [5] htio sam još više proširiti svoje znanje.

Summary

Graduation work is formed out of two parts that I will describe through this short concept.

The head part of this thesis is 3D modeling in a program called *Blender* [1], more detailed explanation will be in the following chapter. I've decided to create my own models in *Blender* [1] because it's very compatible with *Unity game engine* [2]. Also it's easier to edit, precisely scale objects and add textures in *Blender* [1].

The tail or the second part is making a simulation in *Unity game engine* [2]. I've decided to implement my 3D models in *Unity* [2] so that I could present plain models used in a 3D realisation with real physics of space and time. *Unity* [2] also has *"World editor"* or projecting models in real time, but it's made of basic shapes that you are not able to model.

Inside the *Unity* [2] you are creating the logic for your object to work inside the simulation. The logic is focused on *Object-oriented programming in C#* [3] *(C sharp)* that I will be explaining in details a bit later in this thesis.

I've decided to write a graduation work thesis on this theme because I want to improve in C# *O-O programming* [3] and also 3D modeling. Given that I had college studies in both *basic C programming language* [4] and *3D modeling in Autocad* [5]. I wanted to expand my knowledge

Sadržaj:

Popis slika:

Osnovno o Blender-u

[6]

Besplatan program sa otvorenim tipom koda (eng. *„open source“*) , 3D računalno grafički softverski alat korišten za kreiranje animiranih filmova, vizualnih efekta, crteža, 3D printabilnih modela, interaktivnih 3D aplikacija i video igra.

Blender [1] sadrži pojedinosti kao što su: 3D modeliranje (eng. *„3D modeling“)*, UV mapiranje (eng. *„UV unwrapping“)*, teksturiranje (eng. *„texturing“*), raster grafičko uređivanje (eng. *„raster graphics editing“)*, kosturna animacija (eng. *„rigging and skinning“)*, simulacije fluida i dima (eng. *„fluid and smoke simulations“)* , simulacije čestica (eng. *„particle simulations“)*, simulacija mekanog tijela (eng. *„ soft body simulation“),* digitalno oblikovanje(eng. *„sculpting“*), animiranje (eng. *„animating“),* refenetno pomicanje (eng. *„match moving“)*, rendering (eng. *„rendering“*), pokretna grafika (eng. *„motion graphics“)*, video uređivanje i kompoziranje (eng. *„video editing and compositing“*).

Trenutna verzija ima i integrirani *game engine* (jezgra igre) ali će se u sljedećoj 2.8 verziji to izbaciti.

Neke od važnijih stavki *Blendera* [1] su:

- podrška za razne geometrijske oblike, uključujući poligon mrežu (eng. *„polygon mesh“)*, brzo modeliranje subdivizijskih površina, Bezier krivulje, NURBS površine, implicitne površine (eng. *„metaballs“*), icosfere (eng. *„icospheres“*), viserezolucijsko digitalno oblikovanje (eng. *„multi-res digital sculpting“*), rubni fontovi (eng. *„outline fonts“*), i novi B-mrežni (eng. *„B-mesh“*) sustav.

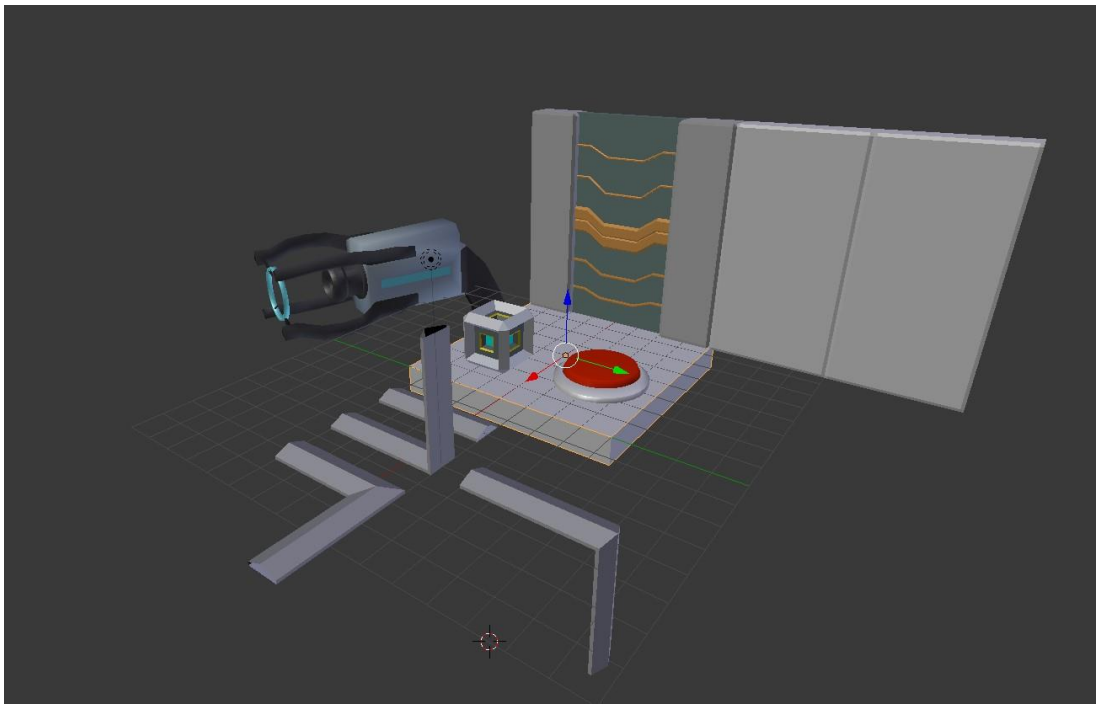
- kontrola u realnom vremenu što se tiče simulacija i *renderinga* (*proces obrade)*

- praćenje kamere i objekta.

- sustav čestica koja podržava čestice bazirane na kosi.

- *Python* skriptiranje za kreiranje vlastitih alata i prototipa, logike igre itd.

- Putno-prateći render engine (eng. *„Pathtracer render engine“*) zvan Cycles koji uzima prednosti GPU-a za *rendering*. (eng. *„GPU-Graphics processign unit“ ;* hrv. grafički procesor).
- Podržava Open Shading programski jezik.



1.1 Slika svih 3D modela izrađenih u Blenderu

Pokretanje *[7]*

Prilikom pokretanja *Blender-a* [1], pojavljuje se početni zaslon na sredini prozora. Sadržava pomoć u obliku poveznica koji vas odvode na *Blender* [1] web stranicu.

Uvod

Nakon pokretanja *Blender-a* [1] i zatvaranja početnog prozora očekuje nas korisničko sučelje koje je isto na svim platformama. (sl. 1.2)

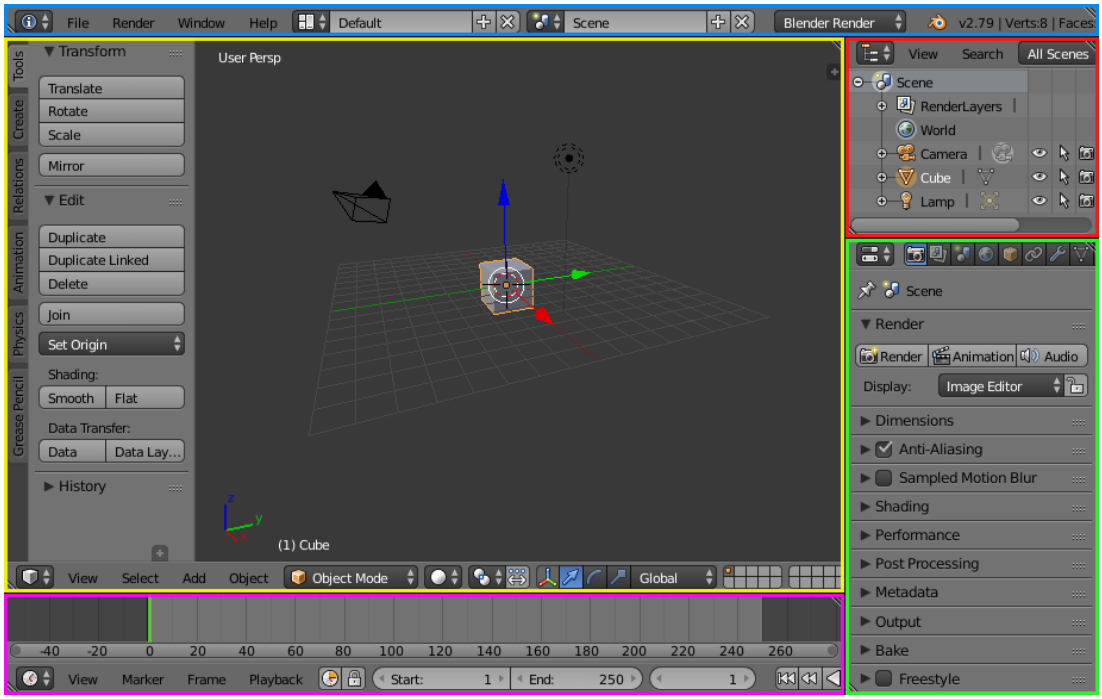
Elementi sučelja:

`Window ▸ Screen ▸ Areas ▸ Editors ▸ Regions ▸ (Tabs) ▸ Panels ▸ Controls`

Standardni zaslon:

Podijeljen na 5 sekcija koje sadržavaju sljedeće uređivače (eng. *„Editors“*):

- Uređivač s informacijama, na vrhu
- Velika 3D scena i pogled
- Vremenska crta na dnu
- Sadržaj scene, desno na vrhu
- Uređivač za postavke, dolje na dnu

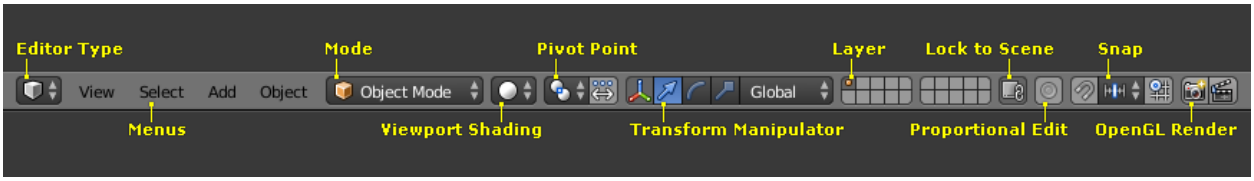


1.2 Standardno korisničko sučelje unutar Blendera

Info (plavo), 3D pogled (eng. *„View“*) (žuto), sadržaj (crveno), Postavke (zeleno), vremenska crta (ljubičasto)

[8]

Objašnjenje zaglavlja (Header):



1.3 Traka zaglavlja (header)

Izbornici (Menus):

- Pogled (View) dozvoljava alatima da navigiraju u 3D prostoru
- Selektiranje (*Select*) sadrži alate za selekciju objekata
- Dodavanje (*Add*) ponuđuje listu različitih tipova objekata koji mogu biti dodani na scenu
- Objekt (*Object*) pojavljuje se u Objektnom načinu (eng. *„Object Modu“*), sadrži alate za uređivanje objekata. U uređivačkom (*Edit*) načinu će se promijeniti u prilagođeni izbornik s uređivačkim alatima

Kontrole (Controls): [9]
Način: 3D Pogled ima više načina za uređivanje različitih tipova podataka:

- Objektni način (eng. *„Object Mode“*)
- Uređivački način (eng. *„Edit Mode“*)
- Oblikovni način (eng. *„Sculpt Mode“*)
- Bojanje po najvišim točkama (eng. *„Vertex Paint“*)
- Bojanje po težini (eng. *„Weight Paint“*)
- Bojanje tekstura (eng. *„Texture Paint“*)
- Uređivanje čestica (eng. *„Particle Edit“*)
- Način poziranja (eng. *„Pose Mode“*)
- Način uređivanja poteza povlačenja (eng. *„Edit Strokes Mode“*)

Sjenčanje u poglednom prozoru (eng. *„Viewport Shading“*):

- mijenja kako će objekti biti prikazani u poglednom prozoru

Točka okreta (eng. *„Pivot point“*):

- mijenja referentnu točku pivotiranja. Razni mrežni (mesh) alati se koriste o-vom opcijom.

Manipulator transformacije (eng. *„Transform Manipulator“*):

- selektori koji omogućuju da se objekt pomiče ili rotira s mišem u odnosnu na os.

Sloj (eng. *„Layer“*):

- slojevi u radu

Zaključavanje na scenu (eng. *„Lock to Scene“*):

- zaključavanje scene, u odnosnu na aktivnu os sve se mijenja.

Hvatanje (eng. „Snap“):

- pomaže pri pomicanju objekta po preciznoj mreži

To su neki od glavnih dijelova programa, te opisi alata s kojima sam se susreo u radu.

Modeliranje u Blender-u

Opisat ću proces rada zahtjevnijeg modela s obzirom na to da sadrži najveći broj korištenih alata.

Proces izrade modela pištolja (eng. *„Portal Gun-a“*)

Započinjem rad sa standardnim prozorom bez ikakvih dodataka i izmjena te modifikacija.

Počeo sam s prvobitnom kockom. Prvo sam izradio tijelo pištolja, običan pravokutnik. Podijelio sam pravokutnik na pola te obrisao drugu stranu te dodao modifikaciju zrcaljenja po Y osi. Sada sam si olakšao polovicu posla jer mogu uređivati samo jednu stranu a na drugoj će sve biti zrcaljeno. Uređivanje modela se najviše sastoji od izlučivanja (lica, kutova, ili stranica) i skaliranja. Nakon izrađenog kostura tijela, produžio sam model te napravio ručku pištolja. Standardno izlučivanje lica po osi te rotacija kutova i poravnavanje završnih točaka. Da bi model bio što prirodniji i uglađeniji, svaka stavka se dijeli na što sitniji broj da bi se izvlačenjem dobio neki zaobljeniji oblik.

Nakon izrade cjelovitog kostura pištolja, dodao sam valjak na početku. Uz obrađene rubove i zaobljenja te šupljinu na prednjoj strani pridodaje izgled cijevi na pištolju. Tu sam koristio opciju sužavanja podijeljenih lica iznutra, po određenoj osi.

Treći objekt na pištolju su stabilizatori Portala. Ovdje je trik bio poigrati se sa stranicama i kutovima jednog stabilizatora da prione uz tijelo pištolja. S obzirom na to da ima krivudavi izgled potrebno je bilo podijeliti objekt na više dijelova te označiti par lica koja su odstupala od tijela pištolja i približiti ih. Špicasti vrhovi se dobe sužavanjem jednog vrha kocke.

Stabilizatori su se trebali fiksirati na lokaciji i u određenoj rotaciji s obzirom na to da sam zrcalio po Y i Z osi. Na stabilizatore se nadovezuje prsten u sredini. Izrađen je od običnog objekta kruga s izbrisanim licem u sredini te izlučenim rubom.

Zadnja stavka je bila udubljenje za „monitor“ na poleđini pištolja, te izrada okidača ili gumba na ručki. Ništa više nego obično skaliranje prema unutra te dijeljenje lica na više dijelova radi lakšeg oblikovanja kasnije.

Radi lakšeg uređivanja pištolja, odnosno dodavanja strukture i boja, potrebno je zrcaljenje i prepoloviti pištolj s označavanjem određenih stranica. Kada raspakiramo to što smo podijelili, dobivamo *2D spriteove (raspakirani oblik 3D tijela u 2D obliku)* odnosno pojedine dijelove za lakše teksturiranje pod određenim kutom svijetla te bacanja sjena itd.

Zatim smo dodali boje na pojedine dijelove pištolja te spremili sve *spriteove* u poseban folder koji ćemo kasnije prebaciti u simulaciju u *Unity-u* [2].

Proces izrade tekstura je odrađen u *Photoshopu* [10]. Napravio sam 3 vrste tekstura koje ovise o kutu gledanja i svijetlu unutar *Unity-a* [2].
Općenito teksture se inače mogu odraditi kroz *Blender* [1], razlog je taj da model onda ima više promjena pod određenim kutom i svijetlom i unutar raznih sjena itd. Ja sam odabrao jednostavnu opciju iz razloga jer sami projekt ima više dijelova te sam htio ubrzati proces izrade.

Svi modeli su spremljeni u *fbx* tipu datoteka. Općenito svi programi za modeliranje koriste ovaj tip datoteka s razlogom da je lakše prebacivati model u kojem god programu znamo raditi.

Osnovno o Unity-u

[11]

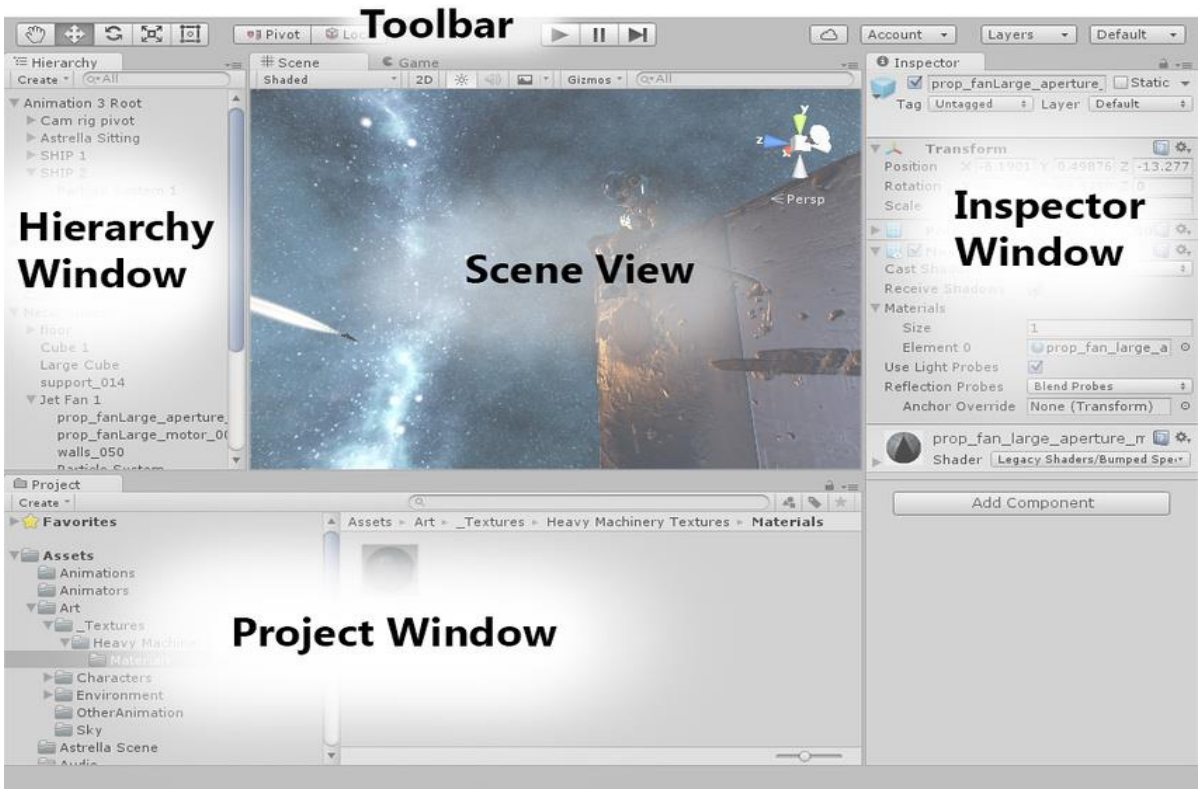
Više platformski (eng. *„cross platform“)* alat (engine) za izradu u trenutnom vremenu (eng. *„real time“*). Najavljen i pušten na tržište u lipnju 2005 na Apple Inc. Internacionalnoj konferenciji za developere. Do danas podržava 27 platformi za pokretanje igara. Alat može biti korišten za kreiranje 3D i 2D igara kao i raznih simulacija. *Unity* [2] daje svojim korisnicima mogućnost da kreiraju igre i razna interaktivna iskustva.

Alat nudi primarno API skriptiranje (eng. *„application programming interface“*) u C# [3], za *Unity* uređivač [2] i razne dodatke (eng. *„pluginove“*) te same igre, i povuci i pusti (eng. *„drag and drop“*) funkcije. Usprkos tome da je C# [3] glavni programski jezik u *Unity-u* [2], nekada je to bio *Boo* [12]. Koji je bio odstranjen iz *Unity-a 5* [2], te je verzija *JavaScripta* [13] nazvana *UnityScript* ukomponirana 2017 u korist C# [3].

Unity [2] ima podršku za sljedeće grafičke API-e:
Direct3D na Windows i Xbox one sustavu; OpenGl na Linux, maxOs i Windows sustavu OpenGL ES na Android i iOS sustavu; WebGL na Internet; te ostale potrebne API verzije za konzole.

Što se tiče 2D igara, *Unity* [2] podržava dodavanje *sprite-ova* i napredni 2D world renderer. Za 3D igre i simulacije *Unity* [2] dozvoljava posebne specifikacije kompresije tekstura (eng. *„texture compression“*), mipmapa (eng. *„mipmaps“*), postavke rezolucija za svaku platformu.

Također posjeduje podršku za mapiranje ispupčenja (eng. *„bump mapping“*, mapiranje refleksija (eng. *„reflection mapping“*, mapiranje paralaksa (eng. *„parallax mapping“*), SSAO (eng. „screen space ambient occlusion“). Dinamičke sjene koristeći mape sjena (eng. *„Shadow maps“*), i renderiranje po teksturi (eng. *„render-to-texture“*) te efekt pozadinskog procesa preko cijelog zaslona (eng. *„full screen post processing effect“).*



1.4 Unity sučelje

[14]

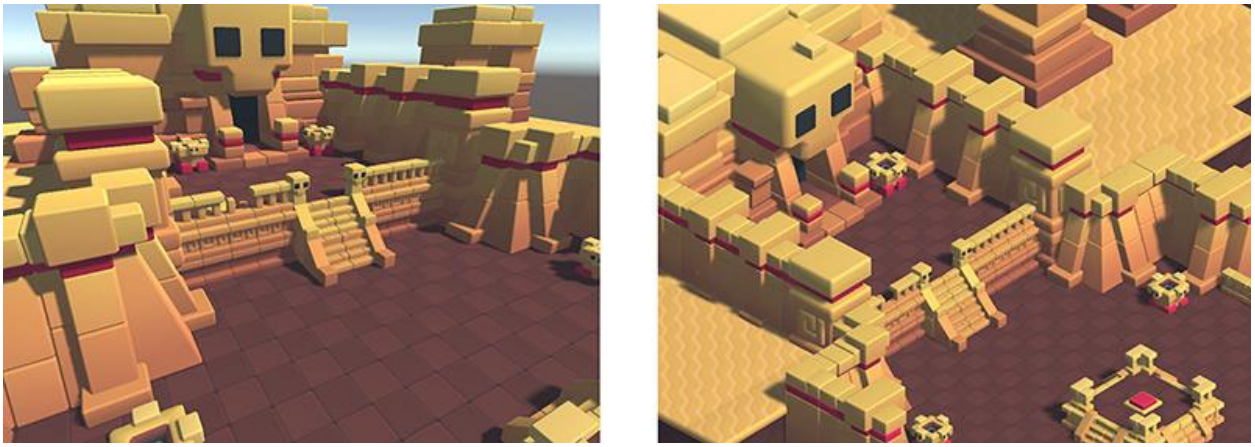
Izgled uređivača može biti drugačiji u svakom projektu, odnosno može biti personaliziran. Standardni izgled daje praktičan pristup često korištenih prozora u uređivaču.

Prozor Projekta (Project Window): [15]
Lijevi dio panela prikazuje strukturu datoteka jednog projekta i to kao hijerarhijsku listu. Kada se označi datoteka s liste, automatski se sav sadržaj datoteke pokazuje na desnoj strani panela. Svaki projekt sadrži individualni sadržaj datoteka koje sami kreiramo tijekom izrade. Najčešći sadržaj pridonose skripte, materijali, *prefabs* (objekti sa zajedničkim modelom, radi uštede vremena), itd. Također iznad panela se nalazi pretraživač liste.

Scenski Pogled (Scene View): [16]
Interaktivni pogled u svijet koji smo sami kreirali. Koristimo Scenski Pogled za selektiranje i pozicioniranje „scenarija“, likova, kamera, svijetla, i svi ostalih objekta.

Navigacija unutar Scenskog Pogleda:

Postoji set navigacijskih kontrola koje nam olakšavaju kretanje kroz svijet. Scenski Kompas (eng. *„Scene Gizmo“*), nalazi se u desnom gornjem kutu prozora Scenskog Pogleda.
Označava orijentaciju trenutne kamere u sceni te olakšava promjenu kuta gledanja i projekcije.
Ruke Scenskog Kompasa označavaju tri glavne osi orijentacije u prostoru,
X , Y i Z.
Također postoji više projekcijskih načina, od kojih je jedan *Perspektivni* te drugi *ortogonalni (isometrički).* Ortogonalni pogled nema perspektivu te se objekt može gledati samo iz jednog kuta. Koristi se radi lakšeg oblikovanja.



1.5 Prikaz Perspektivnog pogleda (Lijevo) i Ortogonalnog (Desno)

Simulacijski Pogled (Game View): [17]

Prikazuje renderirani pogled kamera u simulaciji. Prikazuje finalni izgled simulacije. Aktiviraj (eng. *„Play“*) načinom se kontrolira Simulacijski scenarij (Aktiviraj, Zaus-

tavi). Također promjene koje se naprave u Simulacijskom scenariju su trenutne. Također postoji opcija poboljšanja realnog renderinga Simulacijske scene. Gdje se može promijeniti sve od rezolucije do kvalitete, skaliranja igre itd.

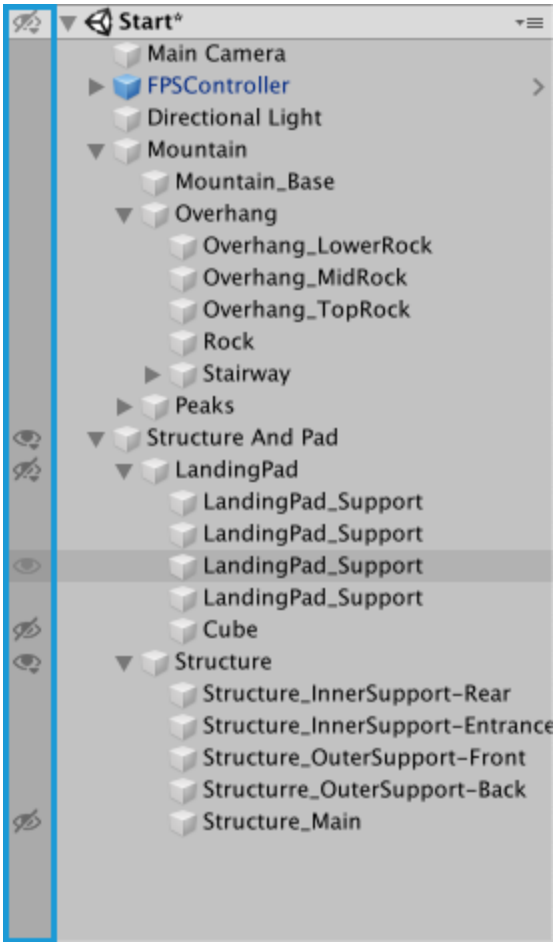
Hijerarhijski Prozor (Hierarchy Window): [18]

Hijerarhijski prozor sadrži listu svakog Simulacijskog Objekta (eng. *„Game Object“*) unutar trenutne Scene. Kako se objekti dodaju i brišu iz scene tako se dodaju i brišu s liste. Standardno objekti su prikazani po redoslijedu kako su kreirani, ali mogu se pomicati po listi kako želimo.

Također moramo obratiti pozornost na odnos Roditelj-Dijete (eng. *„Parent-Child“)* u listi koja se u prijašnjim verzijama *Unity-a* [2] mogla zaobilaziti na razne načine. Ali u novijim verzijama to više nije dopušteno.

Primjer, imamo već definirani objekt kroz 3 scene. Unutar tog objekta se nalazi više pod objekta ili „dodataka“.

Ako želimo izvući dodatak izvan okvira objekta možemo to napraviti jedino tako da raspakiramo *Prefab (već definirani objekt).*

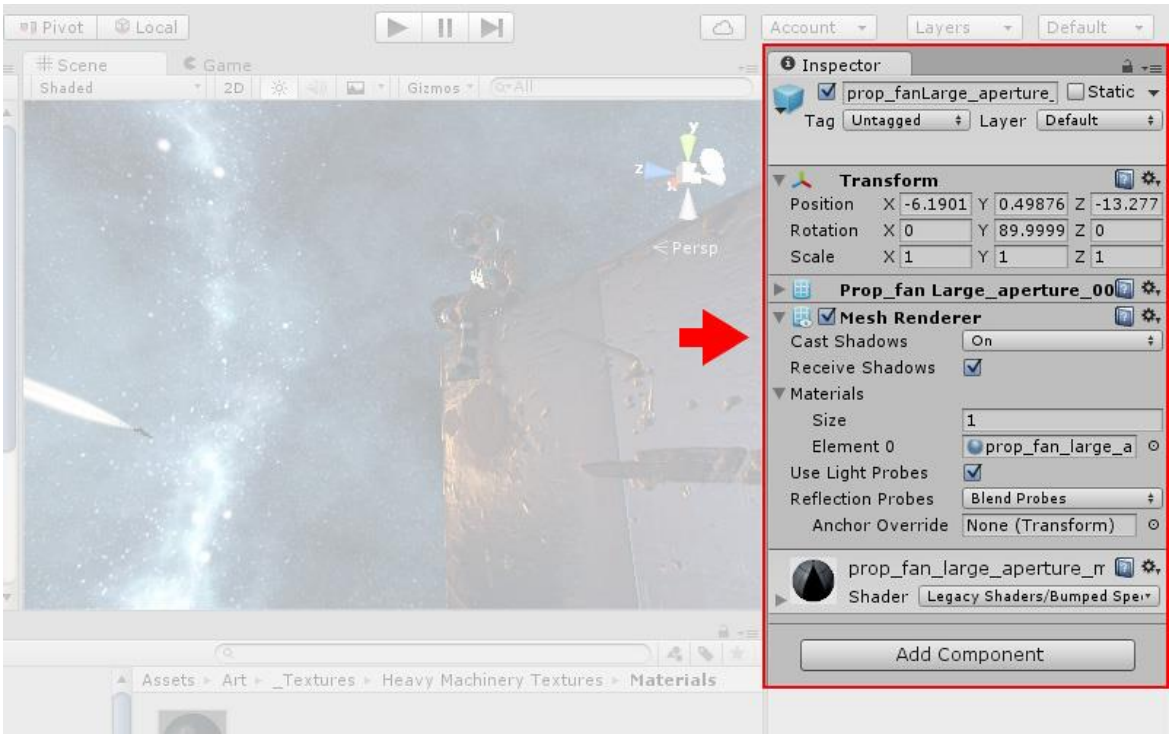


1.6 Preglednost scene

Roditelj-Dijete odnos (Parent-Child)

Koncept koji nastaje prilikom kreiranja i sortiranja objekta. Kada se kreira grupa objekta, na vrhu se nalazi Objekt Roditelj (eng. *„Parent-object“*), a svi grupirani objekti ispod objekta Roditelja nazivaju se Objekti Djeca (eng. *„Child-objects“*).

Pregledni Prozor (Inspector Window): [19]
Projekti unutar *Unity-a* [2] sadrže veći broj Objekta koji posjeduju razne skripte, modele i ostale grafičke elemente kao što je Svijetlo. Pregledni Prozor prikazuje detaljne informacije o trenutno selektiranom Objektu, uključujući dodane komponente i njihova svojstva te dozvoljava modificiranje funkcija Objekta unutar Scene.



1.7 Prikaz Preglednog Prozora na standardnoj poziciji u Unity-u

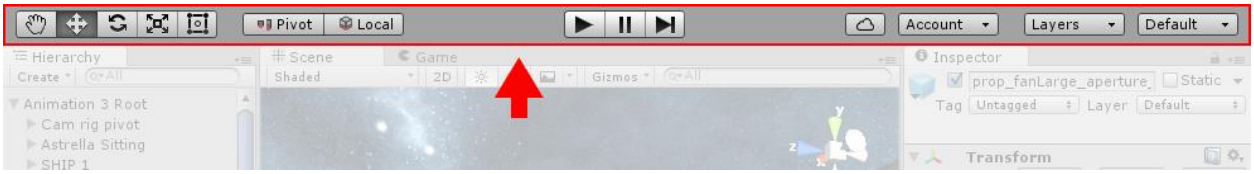
Koristi se za pregledavanje i uređivanje postavka i opcija za skoro sve stavke u *Unity uređivaču* [2]. Uključujući sve fizičke stavke kao što su Objekti, Dodaci, Materijali, ređivačke postavke.

Kada selektiramo Objekt iz liste ili Scene, unutar Preglednog Prozora ćemo dobiti prikaz svih postavki, Materijala tog objekta. Također nam daje prikaz i poziciju glavne kamere u Sceni u odnosu na selektirani objekt (Pozicija, Rotacija, Veličina). Sve postavke je moguće uređivati.

Kada neki Objekt sadrži osobne Skripte, Pregledni Prozor prikazuje javne (eng. *„public") varijable* te skripte koje se mogu uređivati. Služe kao parametri za postavljanje objekta u slučaju da postoje.

Alatna Traka (Toolbar): [20]

Daje pristup najbitnijim alatima u *Unity-u* [2]. S lijeve strane se nalaze osnovni alati za manipulaciju Scene i Objekta. U sredini se nalaze Aktiviraj, Zaustavi, Postepeno kontroliraj gumbi (eng. *„Play, Pause, Step Control“*). Desno stoje postavke za spremanje na *Unity Cloud* Servis te osobni *Unity* račun (eng. *„Unity Account“*) i izbornik vidljivosti *slojeva* (eng. *„Layer visibility menu“*)



1.8 Toolbar

Strukturiranje u Unity-u

Odlučio sam napraviti simulaciju baziranu na igri *"Portal".*

Prvi zadatak je bio naučiti se orijentirati u *Unity-u* [2]. Započeo sam s osnovama programiranja u *C#-u* [3]. Iako se ne razlikuje od svoje osnove (*C jezika* [4]) puno, trebao sam samo napraviti taj skok na objektno orijentirani dio jezika. Nakon prvih par sati upoznavanja s osnovnim varijablama, metodama, petljama, počeo sam koristiti *Vector3* biblioteku (eng. *„library")*, te neke od ostalih *Unity* [2] posebnih *biblioteka*.

Nakon tog uvoda u neko početno objektno programiranje (pomicanje objekta, kamere, i ostalih stvari), modificirao sam već postojeći model igraća u prvom licu. Zatim sam dodavao i kreirao razne objekte kroz *Unity* Uređivač [2], kao i izravna svijetla i dinamička svijetla razne inačice u samoj igri.

Da bi neki od objekta bili interaktivni morao sam stvoriti neku logiku simulacije i realizirati je povezujući međusobno skripte interaktivnih objekta s igračem i njegovim funkcijama te samom logikom svijeta u kojem je simulacija kreirana. Primjer, otvaranje vrata.

Najteži dio bilo je kreiranje *Raycast funkcije* ili hrv. Bacanje zrake. Funkcija koja samom igraću ne pokazuje nikakvu funkcionalnost sve dok ta „zraka" ne stvori *portal*. Naime radi se o funkciji koja povezuje objekt (zid) i model igrača tako da je igrač s funkcijom *Raycast* otvorio portal na „reakcijskom zidu". *Raycast funkcija* radi na principu dometa u odnosu na poziciju i delta vrijeme (trenutno proteklo vrijeme) igrača da bi se portal kreirao te bi igrač na kraju prošao kroz prvi portal i stvorio se na mjestu drugog portala.

S obzirom na to da u simulaciji postoji kutija koja je reakcijski objekt za otvaranje vrata. Morao sam kreirati klasu za hvatanje objekta koja bi se onda nadovezala na sve objekte koju bi igrač mogao uhvatiti i približiti ili spustiti.

Također sam morao urediti *collidere* (imaginarne linije koje ne dopuštaju prolaz jednog tijela kroz drugo), i gravitaciju te još neke sitne inačice *Unity-a* [2]. Opet najveći problem su predstavljali *collideri* zida zbog kreiranja *Portala*, trebali su se ručno podesiti (grafički) ne matematički ili proračunski.

Kreiranje samih *portala* i povezivanje funkcija na model igrača objekta i same simulacije je bilo vrlo zamorno ali sam taj proces riješio tako da sam svaki programibilan dio igre odvajao u zasebne skripte. Međusobno sam povezivao skripte ovisno o funkciji. Najveći problem kod *portala* je bilo kreiranje samih *portala* pod raznim kutovima, što se na kraju riješilo koristeći normale(matematičke) u odnosi na objekt a ne na kameru igrača. Također kreiranje teleportacijskog efekta i efekta zrcaljenja kod portala. Na kraju sam morao još modificirati teleportiranje igrača kroz portale da izlazi iz portala u odnosu na stranu na koju je ušao, znači da gleda ravno a ne prema portalu.

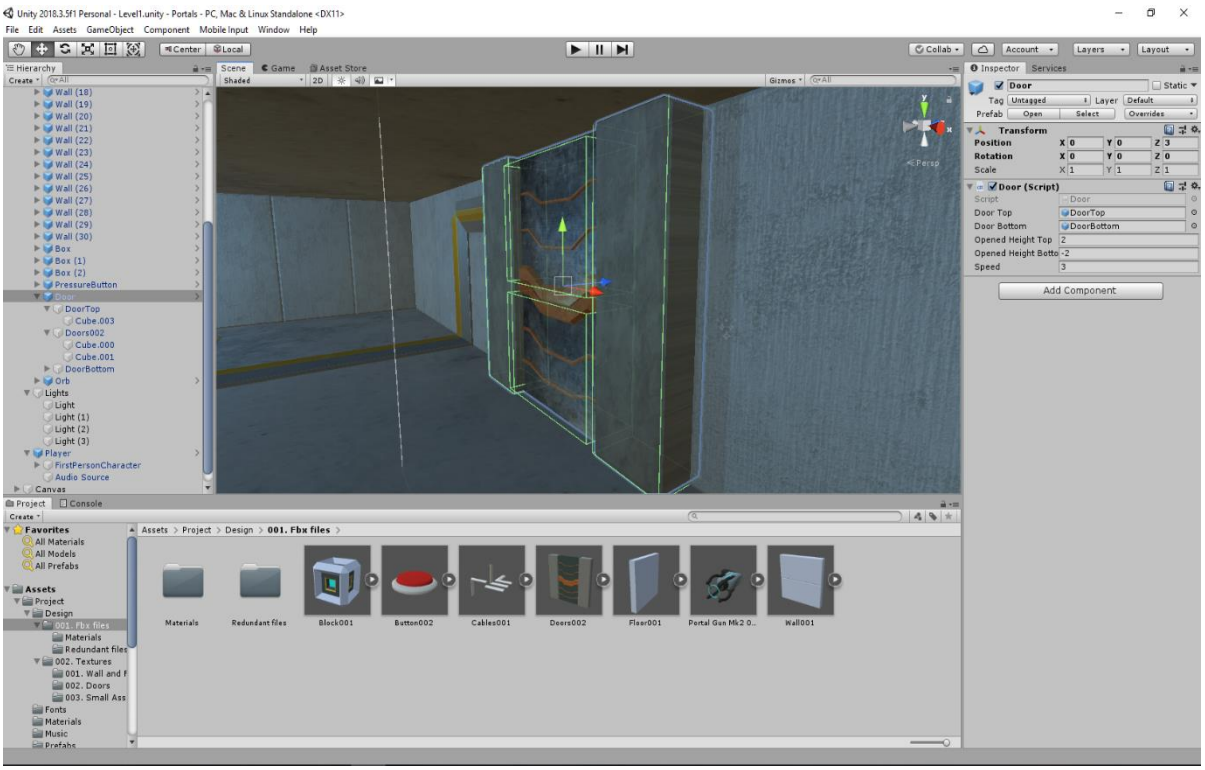
Kreirao sam Dvije razine(eng. *„level“*) i glavni meni.

Prva razina se sastoji od vrata koja treba otvoriti da bi se završila ta razina, potrebno je uzeti jednu kutiju te ju staviti na podni gumb i pričekati da se vrata otvore. Također je moguće stati na gumb i baciti zraku portala u drugu sobu kada se vrata otvore, te kreirati drugi portal u sobi u kojoj se trenutno nalazimo.

Cilj simulacije je što brže doći do kraja (morate proći kroz zadani rotirajući objekt na razini).

Vrijeme se nalazi u gornjem desnom kutu te se ostale bitne poruke ispisuju na ekranu.

Druga razina se sastoji od prepreke koju je potrebno zaobići portalom da bi se došlo do potrebne kutije za otvaranje vrata. Iza tih vrata vas čeka još kutija koje trebate posložiti da bi se popeli na platformu i završili tu razinu.



1.9 Model vrata unutar Unity-a (sa svim teksturama)

Osnovno o C#-u (i unutar Unity-a)

[3]***C# („C Sharp“)***, je opće uporabni programski jezik koji zaokružuje cjelinu na više stavka kao što su:

- ***strong typing*** (hrv. *„jezik strogih pravila“*) (stroža pravila pri kompajliranju, što navodi da se greške i iznimke često dešavaju).
- ***lexically scoped*** (hrv. *„leksičko ciljani jezik“*) (asocijativan jezik koji nam pokazuje postojeće varijable, klase objekte itd.).
- ***imperative*** (hrv. *„imperativan jezik“*) (imperativno programiranje kod kojeg se mijenja stanje programa).
- ***declarative*** (hrv. *„deklaracijski jezik“* (strukturirano programiranje elemenata bez opisivanja toka).
- ***functional*** (hrv. *„funkcijski jezik“*) (matematičko funkcijska orijentacija, suzbija promjenu stanja podataka).
- ***generic*** (hrv. *„generičan jezik“*) (tip algoritma koji se pišu u terminu –stavke koje će se opisati kasnije- te se onda pojavljuju kao parametri koje treba definirati).
- ***object oriented*** (hrv. *„objektno orijentirani jezik“*) (najvažnija stavka ovog jezika, bazirana na konceptu objekta koji sadrži razne parametre koji se mogu modificirati, tj. bazirana na klasama).
- ***component-oriented*** (orijentiran na primjeni kroz razne komponente).

Razvio ga je Microsoft 2000.-ih godina, unutar .NET inicijative. Kasnije je odobren kao standard, ECMA i ISO. Dizajnirao ga je Anders Hejlsber, a trenutni team za razvoj predvodi Mads Torgersen.

Također prati korak uz razvoj *Visual Studio-a* koji je ujedno i IDE(eng. *„Integrated development environment“*) platforma za uređivanje koda.

```
using System;

class Program
{
    public static void Main(string[] args)
    {
        Console.WriteLine("Hello, world!");
    }
}
```

2.1 Sintaksa C# jezika

Skriptiranje u Unity-u

Govori Objektima kako da se ponašaju. Odnos skripta i komponenata povezanih s Objektom te na koji način surađuju i ostvaruju simulaciju ili igru. Skriptiranje u *Unity-u* [2] razlikuje se od običnog programiranja u osnovnom dijelu gdje ne trebamo kreirati *'kod'* da pokrenemo aplikaciju već to *Unity* [2] napravi za nas te se mi samo usredotočimo na razradu simulacije kroz skripte.

Unity [2] radi u jednoj velikoj petlji. Prvo očita sve podatke koji se nalaze u Sceni. Npr. svijetla, objekte, načine ponašanja zatim sve procesuira. Pokreče jednu po jednu sličicu (eng. *„frame"*) a mi odlučujemo sa svojim instrukcijama koje pišemo u obliku *'koda'* u skriptama, kako ćemo izvesti *game scenu.*

Da bi postigli visoki broj sličica (eng. *„frame rate"*) to ne znači da samo simulacija mora izgledati fluidna, već i programske skripte koje će biti pokrenute puno češće odnosno responzivnije.

```
1 using System.Collections;
2 using UnityEngine;
3
4 public class DemoScript: MonoBehaviour  {
5
6     //Variables
7     //Functions
8     //Classes
9
10    //Use this initialization
11
12    void Start () {
13
14    }
15
16
17    //Update is called once per frame
18
19    void Update () {
20
21    }
22
23 }
```

2.2 Struktura C#-a unutar Unity-a

Varijable (Variables)
sadrže vrijednosti i reference na objekt. To su kao kutije koje sadrže nešto što bi se moglo iskoristiti. Varijable počinju s malim slovom.

U *Unity-u* [2] skripte se pišu tako da se prvo poslože alati koje trebamo na vrhu, to je obično deklariranje varijabli. Iz primjera možemo vidjeti da postoje javne i privatne varijable što se tiče vidljivosti, nakon toga se deklarira tip podataka, te ime varijable.

```
5 public class DemoScript: MonoBehaviour {
6
7     public Light myLight;
8     private Light myOtherLight;
9
10 }
```

2.3 Deklaracija varijabli

Postoji više vrsta vidljivosti varijabli, ali dvije najbitnije su javne i privatne.

Razlika je u tome da ako kreiramo skriptu za određeni objekt te deklariramo javnu varijablu (Speed) i privatnu varijablu (Position). Sada unutar *Unity-a* [2] možemo mijenjati Speed stavku ali ne i Position s obzirom na to da je ona samo vidljiva unutar skripte.

Privatne varijable omogućuju *'kodu'* da bude čišći, s obzirom na to da znamo da te vrijednosti možemo mijenjati samo unutar te pojedine klase. Ovakav postupak olakšava *debugging* (rješavanje problema unutar *'koda'*) te preglednost.

Javne varijable zadaju veliki broj problema kada dođe do greške. Moramo proći kroz bazu cijelog *'koda'* kako bi uočili izvor problema s obzirom na to da svaki objekt ima pristup toj varijabli. Javne varijable/funkcije/klase su potrebne da bi objekti mogli komunicirati međusobno.

2.4 Tip podataka varijabli

Još jedna bitna stavka je tip podataka varijable koji definira kakvu vrijednost varijabla sadrži u memoriji. Može biti broj, tekst, ili nešto kompleksnije kao npr. *Transform light ili Demo Script.* To su zapravo reference na komponente. *Unity* [2] mora znati koji je to tip objekta kako bi odučio kako njime može raspolagati.

Treća bitna stavka o varijablama je ime. Ime varijable ne smije počinjati s brojem, i ne smije imati razmak u imenu. Zbog toga postoji stil pisanja u C#-u [3], nazvan *camelCase.* Počinjemo s malim slovima te dodajemo riječi, bez razmaka, s prvim velikim slovom. Npr. „*myLight*“, *unity* [2] će prikazati riječ „*My Light*“, ako je javna varijabla.

Funkcije (Functions)

Skup linija *'koda'* koje uspoređuju i manipuliraju varijablama. Funkcije počinju s velikim slovom. Funkcije se kreiraju zato što ih se lakše može pozvati i koristiti više puta u različitim dijelovima programa.

Postoji više vrsta funkcija koje se automatski pokreću unutar *Unity-a* [2].

```
/*
    Awake()
    Start()
    Update ()
    FixedUpdate()
    LateUpdate

*/
```

2.5 Lista automatski pokrenutih funkcija u Unity-u

Awake():

Funkcija koja je pozvana samo jednom, kada je Objekt aktivan skupa sa tom komponentom. U slučaju da je Objekt *neaktivan,* neće biti pozvana funkcija.
Awake funkcija je pozvana čak i kada je Objekt pozvan ali komponenta nije omogućena. *Awake* se može iskoristiti za inicijalizaciju svih varijabli kojima treba dodati neku vrijednost.

Start():

Bit će pozvana samo u slučaju kada je aktivan Objekt te omogućena komponenta.

Update():

Pozvana jednom po „sličici“ scene (eng. „*once per frame“*). Ovdje se definira *'kod'* logike koji će kontinuirano raditi u simulaciji, npr. animacije, AI (umjetna inteligencija), i ostali dijelovi simulacije ili igre.

FixedUpdate():

Služi kada želimo da bilo koji dio *engine-a* fizike obavi neki posao.

LateUpdate():

Funkcija slična funkciji Update(), ali će LateUpdate() biti pozvan na kraju sličice. *Unity* [2] će pregledati sve Objekte, pronaći sve *Update-e()*, te pozvati sve *Late Update-e*(). Vrlo korisno u slučaju kamera u Sceni. Kao primjer, recimo da želimo pomaknuti igrača u simulaciji. Slučajno se sudara s drugim igračem te završava na drugoj lokaciji. Ako pomaknemo kameru u isto vrijeme kada i igrača, kamera će se „uzdrmati“ i ne bi bila na mjestu gdje treba biti. Stoga nam je potrebna druga petlja.

Kada kreiramo funkciju moramo imati na umu da počinju s povratnim tipom funkcije na početku, popraćeni s imenom funkcije, zatim parametrima. Imena funkcija počinju s velikim slovom a tijelo funkcije dolazi u uglatim zagradama.

```
19
20 void MyFunction () {
21         myLight.enabled = !myLight.enabled;
22 }
23 }
```

2.6 Sintaksa pisanja funkcije

Funkcije mogu napraviti kalkulacije zatim vratiti vrijednost. Možemo dati upit funkciji da napravi nešto, procesuira informaciju, te onda vrati vrijednost u obliku odgovora. Ako koristimo tip podataka *void*, funkcije ne vraćaju vrijednost.

Klase (Classes)

Služe za strukturiranje *'koda'* odnosno zaokruživanje cjelina varijabli i funkcija zajedno da bi se kreirala paleta koja definira postavke objekta.

```
6 public class DemoScript: MonoBehaviour {
7
8     public Light myLight;
9
10    void Awake () {
11        int myVar = AddTwo(9,2);
12        Debug.Log(myVar);
13    }
14
15    void Update () {
16        if (Input.GetKeyDown ("space")) {
17            MyFunction ();
18
19        }
20
21    }
22
23    void MyFunction () {
24
25        myLight.enabled = !myLight.enabled;
26    }
27
28    int AddTwo (int var1, int var2) {
29        int returnValue = var1 + var2;
30        return returnValue;
31
32    }
33
34 }
```

2.7 Primjer klase

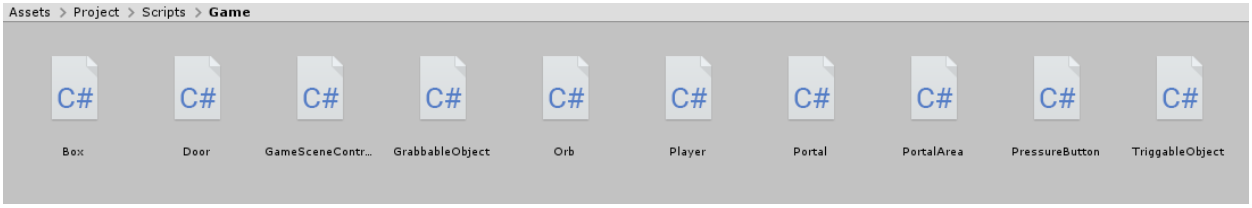
Jako je bitno znati da ime klase more biti isto kao i ime C# [3] skripte kako bi radila. Točnije da bi se mogla dodati na Objekt, mora se razlikovati od klase *„MonoBehaviour“* koja je automatski kreirana kada prvi put kreiramo skriptu. Klase također mogu biti javne ili privatne.

Ako u *Unity-u* [2] kreiramo proizvoljnu klasu moramo dati upit *serialize.it*. To znači da će biti konvertirana u jednostavniji tip podataka koji *Unity* [2] može vidjeti u Preglednom Prozoru.

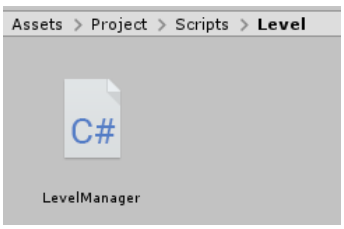
Skriptiranje je primarno uspoređivanje ovih objekata i njihovih trenutnih statusa i vrijednosti. Bazirano je na logici koja odlučuje o rezultatu, tj. predviđa krajnji rezultat.

Programiranje u C#-u (unutar Unity-a)

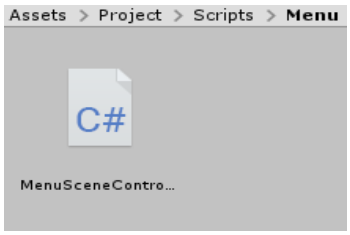
U sklopu *Unity-a* [2] sam kreirao folder *Scripts* u kojem sam podijelio svoju simulaciju na tri glavna dijela za programiranje.



2.10 Game skripte



2.9 Level skripte



2.8 Menu skripte

Game skripte:

Unutar foldera se nalaze sve skripte vezane za igrača i objekte. S obzirom na to da se najveći dio interakcije odvija s igračem, unutar skripte *Player* su najviše povezani svi objekti s funkcijama igrača.

Skripta *Player* referira kao klasa *Player* na klasu *MonoBehaviour* iz koje *Unity* [2] p tječe, odnosno u kojoj se nalaze sve bitne stavke za povezivanje C#-a [3] i *Unity-a* [2].

```
using System;
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class Player : MonoBehaviour {

    [Header("Portal")]
    public GameObject portalPrefab;
    public RenderTexture[] renderTextures;

    [Header("Grabbable objects")]
    public float grabbingDistance = 1.5f;
    public float throwingForce = 150f;
    public float MaxZoom = 4f;
    public float MinZoom = 1.5f;

    public Action onCollectOrb;

    private GrabbableObject grabbableObject;
    private Camera playerCamera;
    private List<Portal> portals;
    private bool shouldUseFirstPortal = false;
```

3.1 Skripta Player, globalne varijable

Uvodni dio se sastoji od globalnih varijabli *Portala* i *dohvatnih objekta.*

Povezan je objekt portala s istim onim kreiranim u *Unity-u* [2], također je kreirana *RenderTexture[ ]* varijabla koju ćemo iskoristiti kasnije kao efekt *Portala* s obzirom na to da sam kreirao efekt zrcaljenja *Portala*. *RenderTexture[ ]* omogućava da se kamera implementira u sami objekt s obzirom na to da nam treba pogled iz Portala prema van da bi dobili pogodni efekt zrcaljenja.

Varijable dohvatnog objekta su također globalne i odnose se na udaljenost dohvaćanja objekta, silu bacanja objekta, maksimalno i minimalno pomicanje objekta tijekom lebdenja i držanja u zraku.

OnCollectOrb je završni objekt za kraj razine koji će nam biti okidač.

Također je kreirana kamera igrača i lista za objekte *Portala*. Razlog korištenja liste je jednostavan, imamo dva portala koja će se izmjenjivati pri pozivu cijelo vrijeme.

```
// Use this for initialization
void Start ()
{
    portals = new List<Portal> ();
    playerCamera = transform.GetComponentInChildren<Camera> ();
}

// Update is called once per frame
void Update ()
{
    // Clicking logic for grabbing objects and shooting portals
    bool interactedWithObject = false;

    if (Input.GetKeyDown ("e"))
    {
        // if holding grabbable object, release it
        if (grabbableObject != null)
        {
            Release ();
            interactedWithObject = true;
            grabbingDistance = 1.5f;

        }

        else
        {
            // Raycast for grabbable objects
            RaycastHit hit;
            if (Physics.Raycast (transform.position, playerCamera.transform.forward, out hit, grabbingDistance))
            {
                // Check if looking at grabbable object
                if (hit.transform.GetComponent<GrabbableObject> () != null)
                {
                    GrabbableObject targetObject = hit.transform.GetComponent<GrabbableObject> ();

                    // Hold the object
                    if (grabbableObject == null)
                    {
                        Hold (targetObject);
                        interactedWithObject = true;
                    }
                }
            }
        }
    }

    //  Logic for spawning portals
    if (Input.GetMouseButtonDown (0) && interactedWithObject == false )
    {
        // Perform the raycast
        RaycastHit hit;
        if(Physics.Raycast(playerCamera.transform.position, playerCamera.transform.forward, out hit))
        {
            if (hit.transform.GetComponent<PortalArea> () != null)
            {
                SpawnPortal (hit.point, hit.normal, hit.transform.GetComponent<PortalArea>());
            }
        }
    }

    // Logic for holding the grabbable object
    if (grabbableObject != null)
    {
        grabbableObject.transform.position = playerCamera.transform.position + playerCamera.transform.forward * grabbingDistance;

        // grabbing Distance move forward and backwards
        if (Input.GetAxis ("Mouse ScrollWheel") > 0)
        {
            grabbingDistance = grabbingDistance + 0.5f;

            if (grabbingDistance > MaxZoom)
            {
                grabbingDistance = MaxZoom;
            }
        }

        if (Input.GetAxis ("Mouse ScrollWheel") < 0)
        {
            grabbingDistance = grabbingDistance - 0.5f;
        }

        else if (grabbingDistance < 1.5f)
        {
            grabbingDistance = MinZoom ;
        }
    }
```

3.2 Skripta Player, uvodne funkcije

Void Start(), inicijalizacija portala u novu listu, i kamere igrača ali s naznakom da se traži objekt ili „djeca“ objekta među komponentama svih dijelova objekta kamere.

Void Update(), pod konstantnom inicijalizacijom spadaju dijelovi funkcija kao što su:

logika za dohvaćanje objekata:

- Sadržava *boolean* varijablu koja nam u kasnijoj razradi govori da li postoji interakcija s objektom. Također sadrži if-else grananje vezano za držanje objekta u slučaju da je pritisnuta tipka „e“ na tipkovnici.
- U slučaju *if* , gdje je objekt dohvaćanja različit od *„null“* tj. ne pokazuje na varijablu u memoriji (držali smo objekt pa ga sad puštamo). Aktivirat će se funkcija *Release (),* koju sam definirao nešto kasnije, *interactedWithObject* postaje istinit (*true*), te je postavljen *grabbingDistance*.

***Raycast* funkcija:**

- U slučaju *else*, pozvana je funkcija *RaycastHit* s varijablom hit, sadrži više grananja od kojih je prvo *if* upit koji govori da će se *glavni else* aktivirati u slučaju ako je *Raycast* pogodio objekt u odnosu na kameru igrača.

logika za držanje objekta u zraku:

- Drugo grananje je *if* upit za provjeru da li igrač gleda prema objektu koji se može dohvatiti. U slučaju da je moguće dolazi na red treće grananje *if* u kojem se aktivira *Hold ()* funkcija.

logika za kreiranje Portala:

- Prvo grananje *if* postavlja se upit slučaja u kojem se traži pritisnuta lijeva tipka na mišu zajedno s upitom da li ne postoji interakcija s objektom.
- U tom slučaju se aktivira *RaycastHit* funkcija s *hit* varijablom i dolazi do drugog granja *if* u kojem *Raycast* opet traži parametre da u odnosu na poziciju kamere igrača, pukne zraku samo u smjeru naprijed i postavi hit kao aktivnu varijablu.
- Zadnje grananje unutar drugog daje upit da li je pogođen validan dio unutarparametara za okolinu portala. U slučaju *true* stvorit će se portal, koji traži 3 parametra. Točku kreiranja zrake, zatim matematičku normalu na tu točku, te zadnje lokaciju da li je dostupna za kreiranje portala u odnosu na *hit* točku. Nije potrebno dodati *else* funkciju u slučaju da se ne može kreirati portal, zato što se portal jednostavno neće kreirati ako nisu zadovoljeni parametri.

logika za približavanje i odmicanje objekta u zraku.

- Sačinjava se od 3 upita i dvije glavne varijable koje su definirane unutar *if* grane logike za držanje objekta. Postavio sam da se objekt odmiče kada pomičem kotačić na mišu prema naprijed, te približava kada pomičem kotačić u nazad. Potrebne su mi bile dvije glavne varijable, maksimalna udaljenost i minimalna udaljenost. Obje su povezane sa *grabbingDistance* ili varijablom daljine dohvata objekta.
- Prvi upit traži ulaz kada je os kotačića u pozitivnom dijelu, te tada je trenutna udaljenost jednaka toj udaljenosti + 0.5. Ako je trenutna udaljeno pokušava prijeći maksimalnu udaljenost, to neće biti moguće. Kontradiktorno tome za os kotačića u negativnom dijelu, trenutna udaljenost je manja za 0.5. Također ako trenutna udaljenost pokušava prijeći minimalnu udaljenost, neće biti moguće. Objekt bi se tada pojavljivao iza periferije kamere igrača.

```
private void Hold (GrabbableObject targetObject)
{
    grabbableObject = targetObject;
    grabbableObject.GetComponent<Collider> ().enabled = true;
    grabbableObject.GetComponent<Rigidbody> ().useGravity = false;
}

private void Release ()
{
    grabbableObject.GetComponent<Collider> ().enabled = true;
    grabbableObject.GetComponent<Rigidbody> ().useGravity = true;
    grabbableObject.GetComponent<Rigidbody> ().AddForce (playerCamera.transform.forward * throwingForce);
    grabbableObject = null;
}

void OnTriggerEnter (Collider otherCollider)
{
    if (otherCollider.GetComponent<Orb> () != null)
    {
        if (onCollectOrb != null)
        {
            onCollectOrb ();
        }
    }

    if (otherCollider.GetComponent<Portal> () != null)
    {
        Portal enterPortal = otherCollider.GetComponent<Portal> ();
        Portal exitPortal = enterPortal == portals [0] ? portals [1] : portals [0];

        transform.position = exitPortal.transform.position + exitPortal.transform.forward;
        transform.GetComponent<UnityStandardAssets.Characters.FirstPerson.FirstPersonController> ().CopyRotation (exitPortal.transform);
    }

}
   private void SpawnPortal (Vector3 spawnPoint, Vector3 normal, PortalArea area)
   {
       Portal currentPortal;

       if (portals.Count < 2)
       {
           GameObject portalObject = Instantiate (portalPrefab);
           currentPortal = portalObject.GetComponent<Portal> ();
           currentPortal.GetComponentInChildren<Camera> ().enabled = false;
           portals.Add (currentPortal);

           if (portals.Count == 2)
           {
               portals [0].GetComponentInChildren<Camera> ().enabled = true;
               portals [1].GetComponentInChildren<Camera> ().enabled = true;

               portals [0].GetComponentInChildren<Camera> ().targetTexture = renderTextures [0];
               portals [1].GetComponentInChildren<Camera> ().targetTexture = renderTextures [1];

               portals [0].GetComponentInChildren<Renderer> ().material.SetTexture ("_MainTex", renderTextures [1]);
               portals [1].GetComponentInChildren<Renderer> ().material.SetTexture ("_MainTex", renderTextures [0]);
           }

       }

       else
       {
           currentPortal = portals [shouldUseFirstPortal ? 0 : 1];

           shouldUseFirstPortal = !shouldUseFirstPortal;
       }

       currentPortal.transform.position = spawnPoint;
       currentPortal.transform.forward = normal;
   }
}
```

3.3 Skripta Player, vanjske funkcije

Sve funckije su definirane izvan *Void Update-a ()* tako da budu dostupne globalno s razlogom jer unutar *Void Update-a()* postoji previše grananja.

Hold ():

- Privatna funkcija bez povratne vrijednosti, kao parametar traži dohvatni objekt. U pravilu samo gasimo komponente *Unity-a* [2], u mojem slučaju *Collider* ostaje upaljen (dokle god držimo objekt u zraku da se i dalje onemogući prolaz objektu kroz zid ili neke druge prepreke), *Rigidbody* točnije da bi objekt lebdio u zraku gasimo *gravity* komponentu (gravitaciju).

Release ():

- Kontradiktorna funkciji *Hold(),* ima iste parametre s drugim komponentama. *Collider* i dalje ostaje upaljen, omogućujemo *gravity,* te pri puštanju objekta dodajemo silu bacanja koja funkcionira samo u pogledu kamere prema naprijed te se množi s varijablom koju smo kreirali kao silu bacanja. Objekt se postavi u *null* referentni dio zbog logike koju sam kreirao iznad.

OnTriggerEnter ():

- Funkcija koja služi kao „most“ između igrača i svih objekata s kojima igrač može imati interakciju. Kao parametar traži *Collider* ili imaginarnu liniju sudaranja objekta sa drugim objektima. Sadržava definirani upit u slučaju da igrač prođe kroz završni rotirajući objekt za kraj razine, te poziva funkciju *onCollectOrb ().* Drugi upit je vezan za portale, u kojem se definira odnos kada igrač ulazi u portal, te kada izlazi iz njega. Najviše se rješava problem pogleda kamere igrača pri ulasku i izlasku iz portala.

SpawnPortal ():

- Također funkcija bez povratne vrijednosti, sadrži 3 parametra za kreiranje *Portala.* Lokaciju kreiranja, vektor normale na pogođenu lokaciju, i pogodnu okolinu za portal. Kreira se početna varijabla trenutnog portala te se traži sljedeće. U slučaju da je broj *Portala* manji od dva (maksimalnih). *Portal* postaje već definirani portal iz *Unity-a* [2], trenutni portal postaje *Portal* koji je kreirani te se traže njegove komponente. Onemogućene su komponente kamere tog *Portala*. Te se dodaje na kraj kreiranog polja.

U slučaju da je broj *Portala* jednak dva, omogućuju se sljedeće stavke.
Komponente svih dijelova kamera oba *Portala.*
Teksture renderiranja tj. efekt zrcaljenja.

- *Else* dio grane postavlja upit da li trenutni *Portal* treba biti početni prvi *Portal.*

- Zadnji dio, izvan grane za kreiranje *Portal*, definiran je pogled vektora normale na trenutnu poziciju *Portala*. Odnosno da se *Portal* kreira u odnosu na objekt ravno a ne pod kutom bacanja „zrake".

Ostale važne skripte

GameSceneController:

- ovdje je definiran HUD (eng. *„Heads-up Display"*) ili lebdeći tekst koji je stalno prikazan tijekom simulacije. Također je definiran vremenski brojač, te da se sve to prebacuje u sljedeću scenu nakon završene razine.

Door:

- definirana logika za otvaranje vrata. Referira se na skriptu *TriggableObject* u kojoj su definirani elementi za sve interakcijske objekte.

PressureButton:

- definirana logika za aktivaciju podnog gumba koji otvara vrata.

Level skripte:

- Odnosi se na upravljanje između trenutno aktivnih scena i prebacivanja na sljedeću scenu. Simulacija se sastoji od 3 scene. *Menu, Level 1 i Level 2* scene.

Menu skripte:

Sadrži funkciju za pokretanje prve razine odnosno Izbornika.

```
public class LevelManager
{

    private static LevelManager instance;

    public static LevelManager Instance
    {
        get
        {
            if (instance == null)
            {
                instance = new LevelManager ();
            }
            return instance;
        }
    }

    private int level;

    private const int MAXIMUM_LEVEL = 2;

    public void LoadFirstLevel()
    {
        level = 1;
        SceneManager.LoadScene ("Level" + level);
    }

    public void LoadNextLevel ()
    {
        level++;

        if (level <= MAXIMUM_LEVEL) {
            SceneManager.LoadScene ("Level" + level);
        }

        else
        {
            SceneManager.LoadScene  ("Menu");
        }
    }
}
```

3.4 Skripta LevelManager

Zaključak

Općenito izrada bilo kakve simulacije sa modelima je vrlo zahtjevan posao s obzirom na to da se takvi zadaci rješavaju u timu ljudi gdje svatko radi jednu vrstu cjelokupnog zadatka. U ovom radu sam pokazao kako i na koji način projektirati 3D modele, na što sve obratiti pažnju prilikom izrade *spriteova* i ostalih stavki modela te na koji način ukomponirati modele u 3D simulaciju. Također sam pokazao kako ukomponirati *Unity* [2] i *Blender* [1].

Što se tiče samog programiranja ovdje je potrebno biti u toku. *Unity* [2] ima tendenciju poboljšavati i mijenjati kompletno cijelu strukturu samog programiranja pa čak i korisničkog sučelja. Upravo tu nastaje problem s kojim sam se ja susreo. U slučaju da dođe bilo kakva nadogradnja za program vi niste obavezni imati tu zadnju verziju programa, ali u slučaju da želite prijeći na novu verziju morate biti svjesni posljedica da će se više toga izmijeniti u vašem projektu.

Stavke koje nisam smatrao potrebnim ali se mogu dodati su:

- razrada *collidera* s kutijama, prilikom nošenja kutija *collideri* nestaju stoga kutije prolaze kroz statične objekte.
- također razrada *collidera* i ostalih stavki fizičkog svijeta za pištolj, trenutna situacija je da prilikom dodavanja takvih stavki skripta igrača se ne može ukomponirati sa pištoljem stoga dolazi do greške.

Smatram da sam s ovim radom pokazao u kojem spektru se 3D modeliranje i programiranje može proširi.

Bibliografija

[1] »Blender,« Blender, Dohvaćeno: 2019. , Dostupnost: https://www.blender.org.

[2] »Unity,« Unity Technologies, Dohvaćeno: 2019. , Dostupnost: https://unity.com.

[3] »C#,« C# programski jezik, Dohvaćeno: 2019. , Dostupnost: https://en.wikipedia.org/wiki/C_Sharp_(programming_language).

[4] »C,« C programski jezik, Dohvaćeno: 2019. , Dostupnost: https://sh.wikipedia.org/wiki/C_(programski_jezik).

[5] »Autocad,« Autodesk, Dohvaćeno: 2019. , Dostupnost: https://www.autodesk.com.

[6] »Blender_(software),« Dohvaćeno: 15.8.2019 , Dostupnost: https://en.wikipedia.org/wiki/Blender_(software).

[7] »Blender_User_Interface,« Dohvaćeno: 2019. , Dostupnost: https://docs.blender.org/manual/en/latest/interface/index.html#.

[8] »Blender_Window_Sytem,« Dohvaćeno: 2019. , Dostupnost: https://docs.blender.org/manual/en/latest/interface/window_system/introduction.html.

[9] »Blender_3D_View,« Dohvaćeno: 2019. , Dostupnost: https://docs.blender.org/manual/en/latest/editors/3dview/introduction.html.

[10] »Photoshop,« Photoshop, Dohvaćeno: 2019. , Dostupnost: https://www.photoshop.com.

[11] »Unity_game_engine,« Dohvaćeno: 22.8.2019 , Dostupnost: https://en.wikipedia.org/wiki/Unity_(game_engine).

[12] »Boo,« Boo, Dohvaćeno: 2019. , Dostupnost: http://boo-lang.org/.

[13] »JS,« Javascript, Dohvaćeno: 2019. , Dostupnost: https://www.javascript.com/.

[14] »Unity_interface,« Dohvaćeno: 2019. , Dostupnost: https://docs.unity3d.com/Manual/LearningtheInterface.html.

[15] »Unity_Project_View,« Dohvaćeno: 2019. , Dostupnost: https://docs.unity3d.com/Manual/ProjectView.html.

[16] »Unity_Scene_View_Navigation,« Dohvaćeno: 2019. , Dostupnost: https://docs.unity3d.com/Manual/SceneViewNavigation.html.

[17] »Unity_Game_View,« Dohvaćeno: 2019. , Dostupnost: https://docs.unity3d.com/Manual/PositioningGameObjects.html.

[18] »Unity_Hierarchy_Window,« Dohvaćeno: 2019. , Dostupnost: https://docs.unity3d.com/Manual/SceneVisibility.html.

[19] »Unity_Inspector_Window,« Dohvaćeno: 2019. , Dostupnost: https://docs.unity3d.com/Manual/UsingTheInspector.html.

[20] »Unity_Toolbar,« Dohvaćeno: 2019. , Dostupnost: https://docs.unity3d.com/Manual/Toolbar.html.

[21] »C_Sharp_Unity,« Dohvaćeno: 2019. , Dostupnost: https://unity3d.com/learning-c-sharp-in-unity-for-beginners.