Izrada računalne igre biljar u programskom alatu Unity

Nenadić, Karlo

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Karlo Nenadić

IZRADA RAČUNALNE IGRE BILJAR U PROGRAMSKOM ALATU UNITY

ZAVRŠNI RAD

Varaždin, 2020.

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Karlo Nenadić

Matični broj: 45863/17–R

Studij: Informacijski sustavi

IZRADA RAČUNALNE IGRE BILJAR U PROGRAMSKOM ALATU UNITY

ZAVRŠNI RAD

Mentor:

Prof. dr. sc. Danijel Radošević

Varaždin, kolovoz 2020.

Karlo Nenadić

Izjava o izvornosti

Izjavljujem da je moj završni rad izvorni rezultat mojeg rada te da se u izradi istoga nisam koristio drugim izvorima osim onima koji su u njemu navedeni. Za izradu rada su korištene etički prikladne i prihvatljive metode i tehnike rada.

Autor potvrdio prihvaćanjem odredbi u sustavu FOI-radovi

Sažetak

U okviru rada izradit će se 3D igra biljara u programskom alatu Unity pomoću C# programskog jezika. U tu svrhu, izradit će se model biljarskog stola te štap i kugle za biljar u 3D Blenderu. Pogled je odozgo te podsjeća na igru s dvije dimenzije. Igra je namijenjena za dva igrača koji naizmjence igraju na istom računalu u istom prozoru. Pomicanjem miša lijevo-desno rotira se štap oko bijele kugle, a na pritisak lijeve tipke miša štap se udaljava od bijele kugle odnosno raste sila kojom će bijela kugla biti udarena. Za izradu rada potrebno je imati predznanja o matematici, fizici, programiranju i dizajniranju. Ideja rada je povezati vizualni izgled s programskom logikom iza njega te opisati kompletan postupak u izradi jedne takve igrice.

Ključne riječi: Unity, 3D blender, C#, biljar, 3D računalna igra

Sadržaj

1. Uvod

Tema ovog završnog rada je izrada 3D računalne igre u programskom alatu Unity namijenjena za dva igrača koji ju igraju na istom računalu s Windows operacijskim sustavom. Dakle, nije moguće igrati protiv „računala“ ili protiv igrača koji ima pokrenutu igru na drugom računalu, već dva igrača naizmjenično igraju na istom računalu. Nadalje, Igrač 1 uvijek počinje igru i njegova boja kugli je određena prvom kuglom koju on ubaci u rupu. Ako igrač 1 ne ubaci niti jednu kuglu, na redu je igrač 2 i vrijedi isto pravilo o određivanju boje. Cilj je prvi ubaciti sve svoje kugle i zatim crnu kuglu s time da se vodi računa da se prilikom udaranja bijele kugle ne dotiču kugle drugog igrača. U igri su dostupne postavke o vidljivosti pomoćne linije, njezinoj boji i duljini te o teksturi stola. Dvoboj je moguće pauzirati i nastaviti bez obzira ugasili igru ili ostali u njoj. Igra treba biti jednostavna i oku ugodna.

Tema završnog rada je značajna radi upoznavanja drukčijeg iskorištavanja programskih jezika te povezivanja dosad naučenog iz programiranja klasičnih konzolnih aplikacija s programiranjem računalne igre. Bitno je shvatiti kako je C# samo jedan od alata za ostvarivanje željenog, ali i isto tako da on sam ne može puno, nego da mu treba grafički dio koji je, buduće da se radi o 3D igri, 3D model objekata. Itekako je značajno razumjeti kako nešto tako jednostavno, a opet složeno funkcionira i kako se to može najbolje izgraditi.

Moja motivacija za odabir ove teme je proširivanje znanja iz programiranja, želja za upoznavanjem C# programskog jezika te stvaranje nečeg svojeg, izrada takozvanog remek djela, barem na neki način i barem u ovom trenutku. Motiviran sam izazovom budući da nikad nisam dizajnirao 3D objekt ni napravio 3D igricu kao ni napravio program pomoću C#-a. Imam iza sebe jednu vrlo jednostavnu 2D igricu napravljenu u Unity-u i nasreću je ispalo da je to sasvim dovoljno predznanje za izradu ovog završnog rada.

2. Metode i tehnike rada

Glavni alat za izradu igre je Unity [55]. Naravno, on nije dovoljan tako da je on korišten za izradu same igre odnosno povezivanje *backend-a* i *frontend-a*, tj. programskog koda pisanom pomoću alata Visual Studio Community 2019 [56] i 3D objekata izrađenih u alatu Blender [57]. Za pisanje dokumentacije odnosno ovog dokumenta je korišten Word [58], za izradu i uređivanje slika alat GIMP [59] te za uređivanje zvukovnih zapisa alat Audacity [60].

Za velik dio istraživačkih aktivnosti, službena dokumentacija Unity-a [1],[2] je bila izvrsna i sasvim dovoljna uz jednostavno pretraživanje klasa, metoda i svojstava, međutim dio istraživačkih aktivnosti je odrađen na web-aplikacijama YouTube, Stack Overflow i Unity Answers budući da dokumentacija Unity-a ne može baš sve pokriti i tako dobro objasniti kao što može netko pomoću videa ili svojim riječima.

Izrada igre odvija se po vodopadnom modelu. Dakle, razrada teme kreće od izrade modela u alatu Blender, odnosno od izrade stola s rupama, štapa i kugli za biljar. Nakon postavljanja svih objekata na početne pozicije, slijedi faza pisanja koda kako bi igra funkcionirala što je i najbitnija, najsloženija i najduža faza. Zatim, kada je kod za funkcionalnost igre napisan, slijedi izrada grafičkog sučelja, postavki i mogućnosti spremanja igre. Na kraju dolazi faza testiranja do savršenstva i izrada dokumentacije.

Blender je alat u kojem se kreiraju 3D objekti, a neke od mogućnosti koje ću ja koristiti u izradi svojih modela su kreiranje objekata, izduživanje stranica, pomicanje i spajanje stranica, rubova i vrhova, rotiranje i skaliranje objekata, primjenjivanje materijala na dijelove objekta, omatanje teksture oko objekta te podjela stranica objekta na manje dijelove [3].

S druge strane alat Unity je okruženje za izradu računalnih igara u kojem ću provesti velik dio vremena kao i u okruženju za pisanje koda Visual Studio-u [4]. Temelj Unity igre je scena, kojih ima od jedne pa naviše, objekata u samoj sceni te njihovih svojstava i ponašanja [4]. U svom ću radu kreirati dvije scene s nekoliko objekata koji čine igru te grafičkim sučeljem koje krasi svaku igru. Koristit ću se prethodno stečenim znanjem s Unity Learn-a [5], novootkrivenim znanjem iz službene dokumentacije, videa te forumskih objava.

3. 3D modeliranje

Kroz ovo poglavlje prolazim kroz proces izrade modela biljarskog stola, štapa i kugli u alatu Blender.

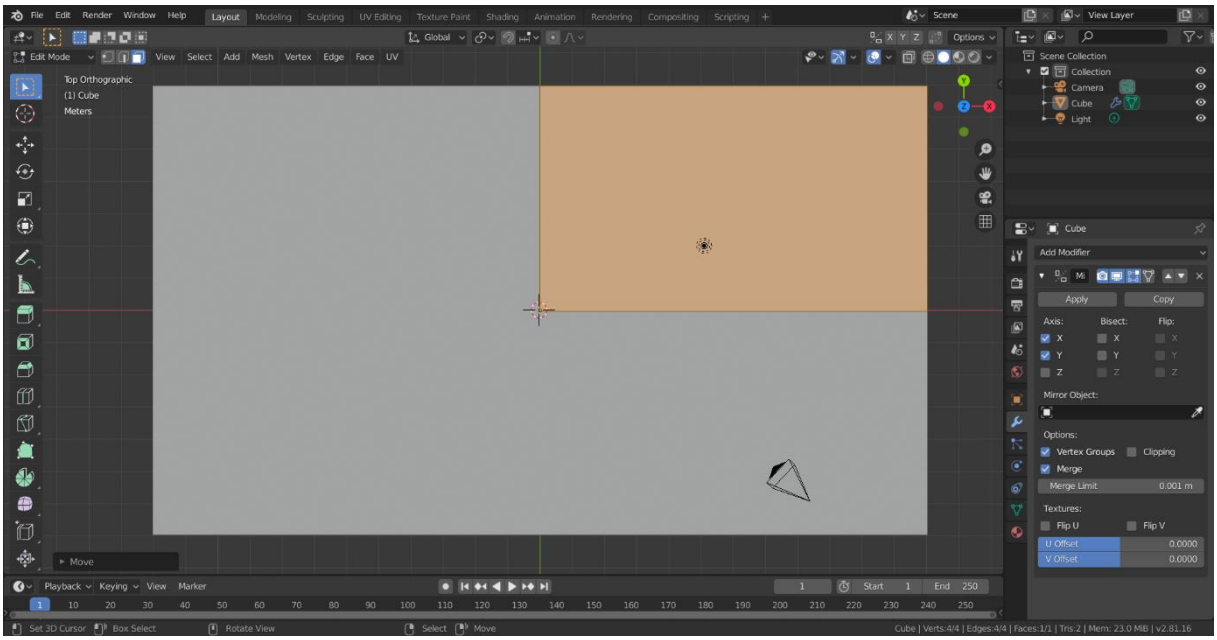
3.1. Modeliranje stola

Prvo ćemo riješiti ono najteže, a to je stol na kojem se igra. Za potrebe moje igre, koja je 3D i u kojoj postoji gravitacija kao i svi ostali zakoni fizike, model stola mora biti što vjerniji originalu kako bi fizika što bolje funkcionirala i sam osjećaj igre bio što realniji. Ideja za model stola je uzeta od jedne već poznate i uspješne biljarske igre „8 Ball Pool“ namijenjenoj za mobitele, a model možemo vidjeti na slici 1.

Slika 1: Model stola za biljar [6]

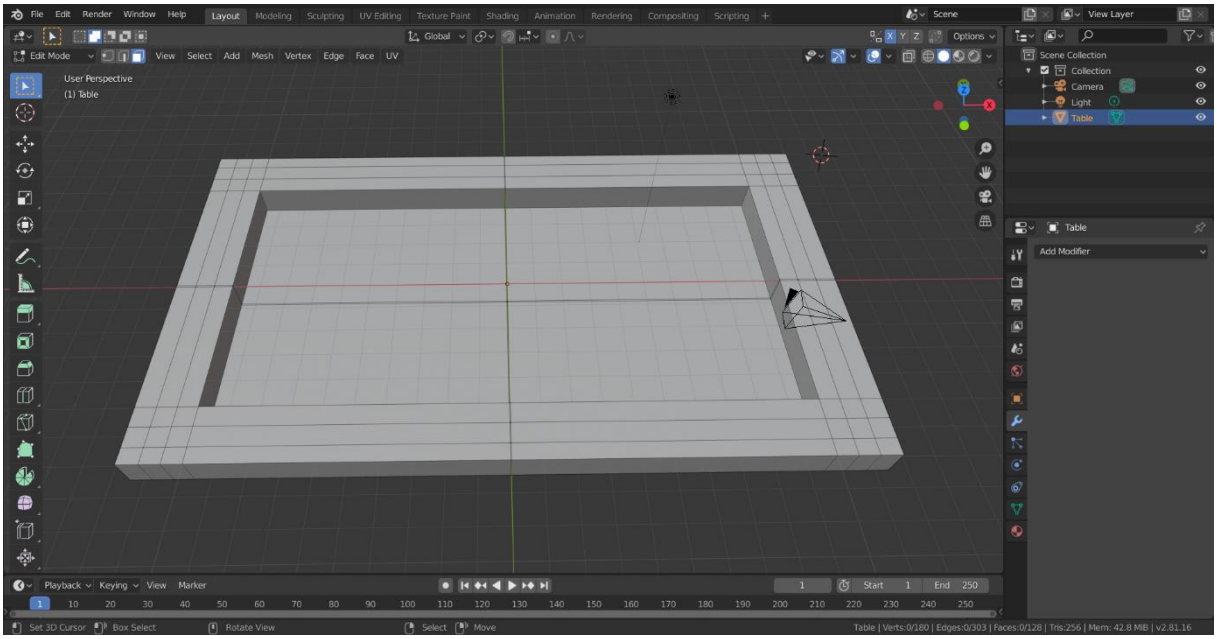
Moj model stola će se, naravno, značajno razlikovati od modela sa slike 1, međutim potrebna mi je osnova za raspored rupa i njihovu udaljenost tako da neke sličnosti mora biti.

Za početak u novoj datoteci u alatu Blender brišemo već postavljenu kocku te dodajemo plohu i postavljamo joj širinu na 12 i dužinu na 7 kvadrata, što možemo još kasnije izmijeniti. Kako bi sve bilo lakše i savršenije, plohi dodamo modifikator zrcalo (*Mirror modificator)* po X i Y osi i tako dobivamo dva puta širu i dva puta dužu plohu. To znači da kada napravimo rupu na jednom od četiri dijela plohe, identična rupa će biti napravljena i na ostala tri dijela plohe [8]. Početni izgled možemo vidjeti na snimci zaslona slike 2.



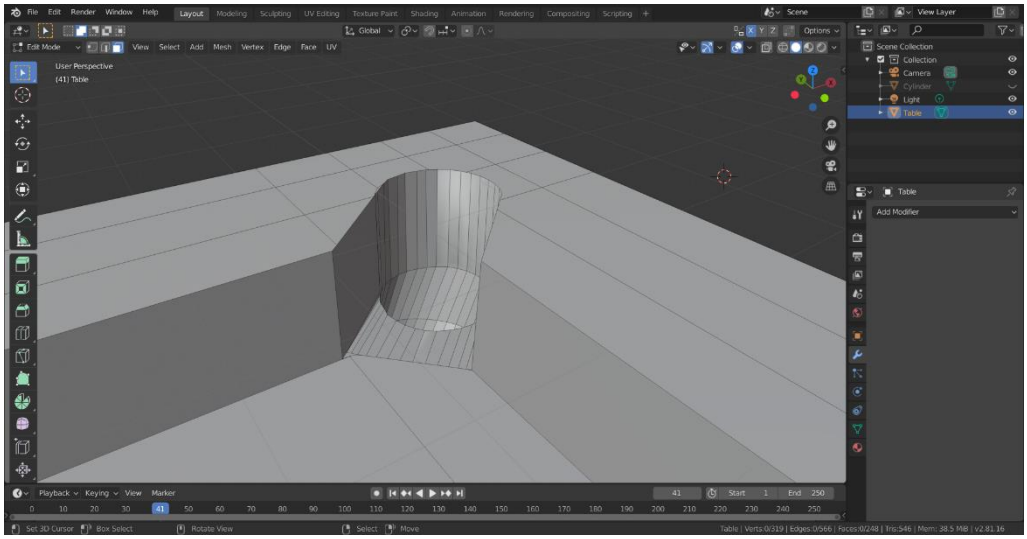
Slika 2: Zrcaljena ploha u alatu Blender [snimka zaslona]

Sada slijedi podjela plohi na rubnike i samu površinu za igranje što radimo pomoću kombinacije tipki Ctrl+R te podijelimo sam rub plohe na tri mala dijela horizontalno i vertikalno. Nove male dijelove označimo te pomoću kombinacije tipki G+Z pomaknemo označene dijelove iznad ostatka stola kako bismo dobili rub. Trebali bismo dobiti nešto kao na slici 3.

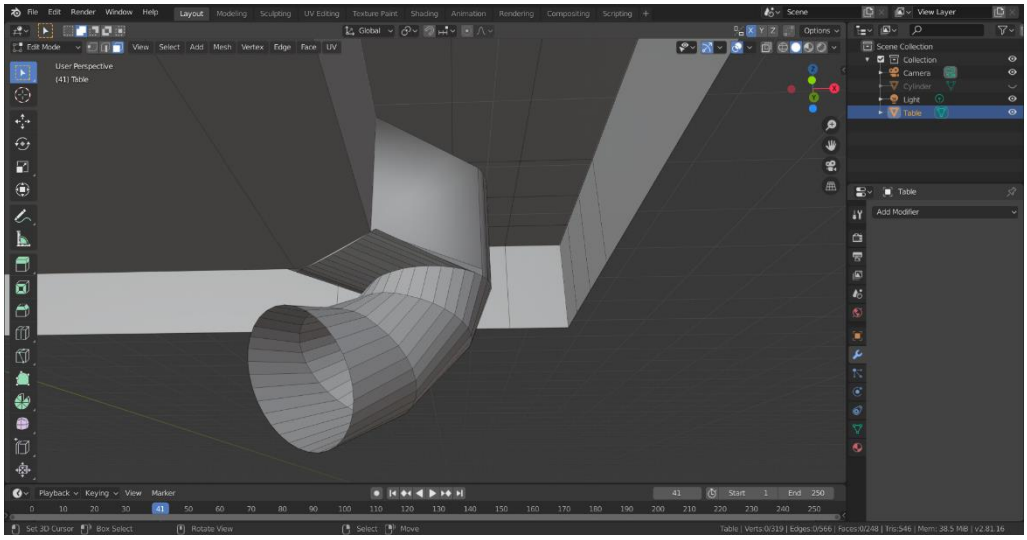


Slika 3: Rub modela stola u alatu Blender [snimka zaslona]

Sljedeće što je bitno napraviti su rupe. Odlučio sam napraviti cijev koja bi odvodila kugle ispod stola umjesto da one samo padnu kroz rupu. Za tu ideju mi je pomogao video [7] gdje se objašnjava kako cilindar položimo na željeno mjesto, odnosno gdje želimo da bude rupa, i zatim primijenimo na stolu modifikator istinitosti (*Boolean modificator)* te postavimo svojstvo na razlika *(Difference)*, a za objekt odaberemo novokreirani cilindar. Cilindru skaliramo promjer na željenu dužinu i pomoću istiskivanja *(Extrude)* mu izdužimo donje lice koje rotiramo po jednoj od osi te taj postupak ponavljamo dok ne dobijemo zaobljenu cijev. Slijedi spajanje rupe i stola, a to radimo na način da odaberemo pola od ukupnog broja lica cilindra koja su najbliže stolu te ih izvučemo dok ne probiju oba zida oko rupe te tako stvore prilaz. Nakon toga usavršimo prilaz rupi pomičući vrhove i rubove u ravnu liniju koja spaja rubove zida oko rupe. Kako to trenutno izgleda možemo vidjeti na slici 4 i 5.

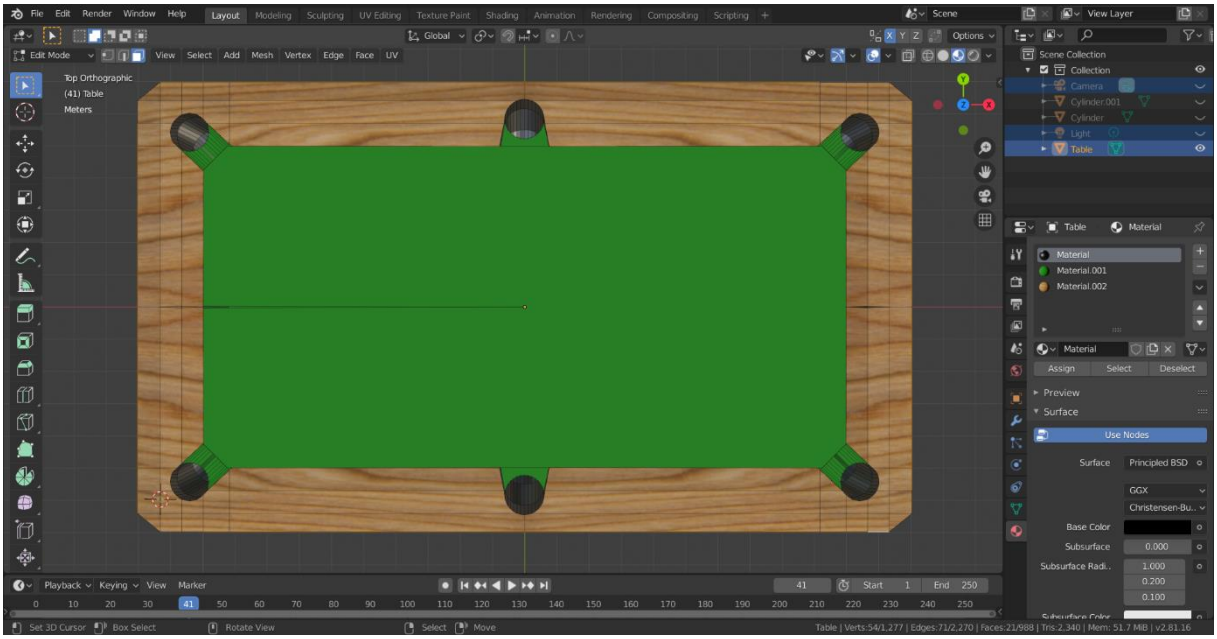


Slika 4: Gornja strana rupe u alatu Blender [snimka zaslona]



Slika 5: Donja strana rupe u alatu Blender [snimka zaslona]

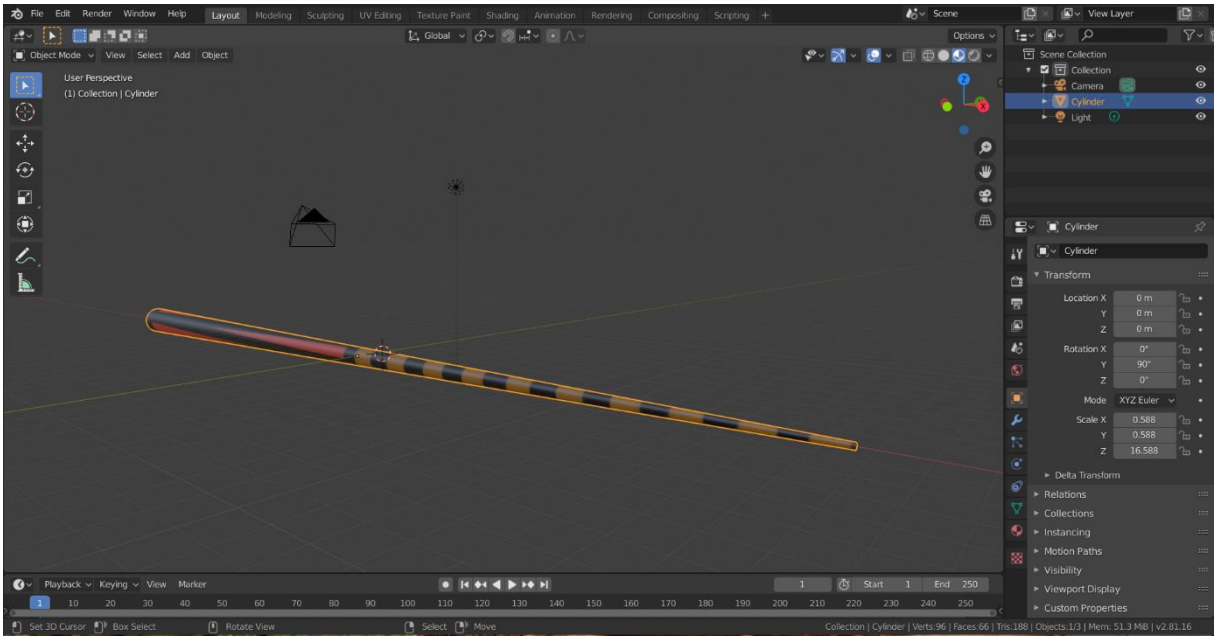
Nakon uspješno kreirane jedne rupe, na sličan način napravimo kopiju cilindra i primijenimo ga u sredinu većeg ruba stola kako bismo dobili središnju rupu budući da se ona razlikuje od rupe u kutu po veličini kuta spoja rupe i površine za igranje. Tada jednostavno označimo cijeli objekt i u izborniku mreže *(Mesh)* odaberemo simetriju te ju namjestimo po X i Y osi kako bi se ostale rupe same napravile [8]. Zatim možemo uređivati rubnike odnosno zaobliti ih kako bi stol ljepše izgledao. Ono što preostaje je primijeniti materijale na određene dijelove stola. Podloga će biti od zelene boje koja manje odbija svjetlost, dok će rupe biti od crne boje koja jače odbija svjetlost odnosno više se sjaji. Rubovi stola će biti od materijala koji je, umjesto boje, baziran na teksturi slike drveta preuzete s interneta [9]. Isprobamo kako izgledaju i tamna i svijetla drvena tekstura te ih spremimo za kasnije [11],[12]. Konačno, model stola je gotov i prikazan na slici 6. Moguće je da prilikom izvoza modela i dodavanja u alat Unity, neke stranice budu prozirne, odnosno vide se samo s jedne strane. To je zbog normala i poznat je problem kod izvoza iz Blender-a u Unity, a potrebno je samo prozirnim stranicama odnosno licima okrenuti normalu [14].



Slika 6: Konačan model stola u alatu Blender [snimka zaslona]

3.2. Modeliranje štapa

Štap je relativno jednostavniji za izradu od stola budući da je to cilindar nešto uži na jednom kraju, a nešto širi na drugom. Tako i u alatu Blender dodajemo cilindar odnosno valjak

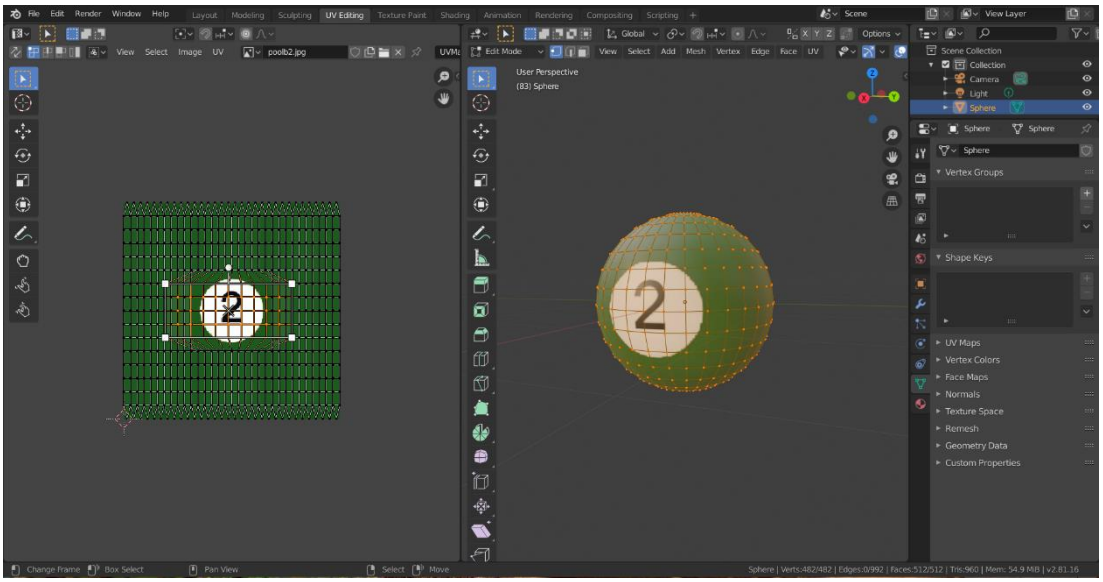


Slika 7: Model štapa u alatu Blender [snimka zaslona]

i jednom od krajnjih lica suzimo promjer, izdužimo štap po Z osi, rotiramo ga te nam samo preostaje još primijeniti materijal odnosno teksturu. U prozoru sjenčanje *(Shading)* označimo cijeli objekt te mu dodamo materijal čiju boju vežemo za boju naše teksture [13]. Moj će štap za teksturu imati sliku preuzetu s interneta [10]. Nakon odabira teksture, kako nam je objašnjeno u spomenutom videu [13], prelazimo na prozor UV uređivanje *(UV Editing)* gdje precizno odabiremo kako će se tekstura omotati oko objekta budući da je to jedna slika koja treba biti u određenim omjerima na određenim dijelovima našeg objekta. Ako nam je sve pošlo za rukom, malo usavršimo područja teksture te na kraju dobivamo model kao što je na slici 7.

3.3. Modeliranje kugli

Kada smo napravili model štapa, možemo napraviti najjednostavniji model a to je model kugli. Kreiramo gotovi objekt UV sfera *(UV Sphere)* te takvoj, za nas savršenu, samo dodamo još teksturu na isti način kao i kod štapa. Sreća postoje dobri ljudi koji su već za nas pripremili svih 15 slika kugli iz biljara koje sam preuzeo s interneta [10], a samo ih je potrebno omotati oko svake kugle zasebno. Za bijelu kuglu ne moramo koristiti nikakvu sliku već joj jednostavno postavimo boju površine u bijelu boju. Kako izgleda uređivanje teksture za kuglu s brojem 2 možemo vidjeti na slici 8. Razlog zbog kojeg smo kod štapa i kugli koristili UV uređivanje, a ne jednostavno dodavanje materijala na pojedine dijelove objekta kako kod stola, je taj što smo imali gotovu sliku koju je trebalo omotati oko štapa odnosno kugle, a kod stola smo imali sliku drveta koju nije trebalo omotati oko cijelog stola nego samo oko njegovog ruba. Da smo imali gotovu sliku cijelog stola, tada bismo također koristili UV uređivanje.



Slika 8: Uređivanje teksture kugle u alatu Blender [snimka zaslona]

4. Izrada igre

Nakon izrade objekata koji će činiti igru, dolazi najbitniji dio a to je pisanje koda kako bi igra mogla funkcionirati. Počet ćemo s osnovnim klasama i dijelovima koda koji čine srž igre, nastaviti s izradom grafičkog sučelja i dodavanjem ostalih klasičnih mogućnosti koje danas ima svaka igra, te završiti s fazom testiranja.

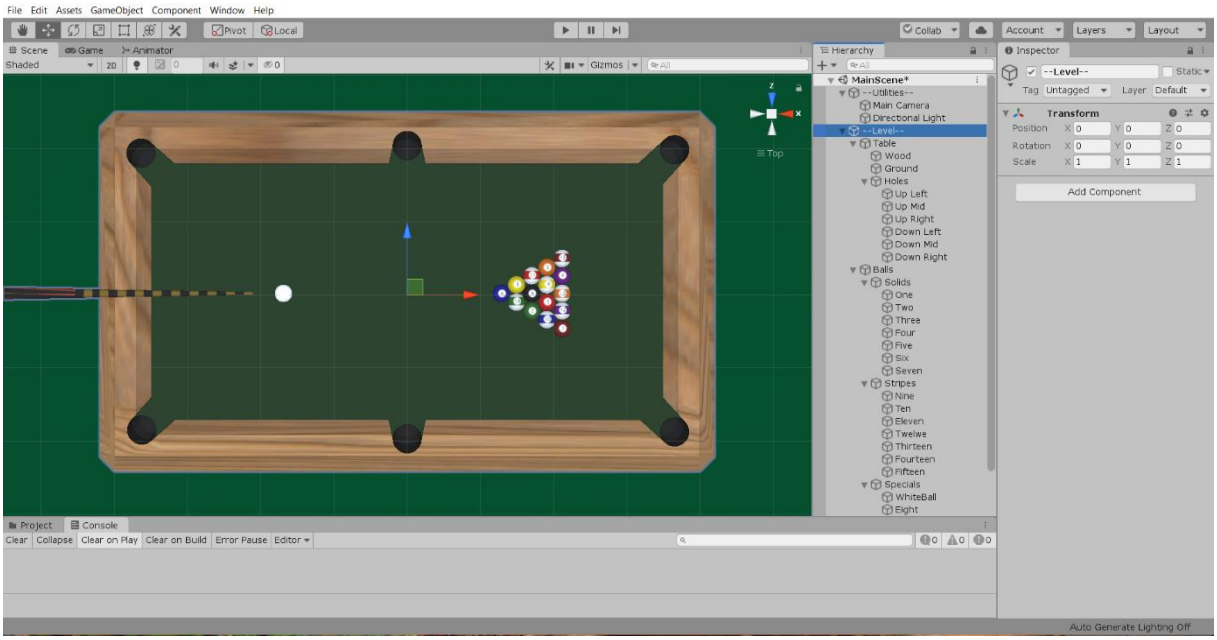
4.1. Izrada logike igre

Za početak, kreiramo novi 3D projekt u alatu Unity. Kako želimo da nam igra bude s pogledom odozgo te da sliči 2D igri, kameru rotiramo po X osi za 90 stupnjeva te postavljamo projekciju na ortografsku umjesto na zadanu perspektivnu [1, /Camera-orthographic.html]. Želimo da nam pozadina bude kao neka soba za biljar odnosno da bude tamno zelena. Za to je potrebno izraditi nebo *(Skybox)* kojemu možemo postaviti boju zemlje, neba, sunca i njegovu veličinu te ostale neke stvari. Nama je najbitnije postaviti zemlju na tamno zelenu. Nakon toga u postavkama svjetlosti *(Lighting settings)* promijenimo izvor neba na naš novokreirani objekt kao što je to objašnjeno u video zapisu [15].

4.1.1. Početno postavljanje

Sada možemo početi s izradom scene. Potrebno je postaviti sve objekte koji su nam potrebni, a to su oni koje smo izradili u alatu Blender. Kako bi sve bilo organiziranije i preglednije, centar stola postavljamo u „centar svijeta“ odnosno na koordinate (0,0,0), dodajemo i kugle na stol koje slažemo u trokut, tj. njihovu početnu postavu te štap stavljamo nešto iznad stola kako se on prilikom pomicanja ne bi sudarao sa stolom ili kuglama. Kao što sam rekao, bitno je držati sve organizirano tako da slobodno možemo kreirati prazne objekte u hijerarhijskom prozoru te ih prikladno nazivati, a stvarne objekte razvrstati kao njihovu djecu. Tako nam stol čini zelena površina za igranje, drveni rubovi i rupe te objekti za detekciju prolaza kugli kroz rupe koji će biti kasnije objašnjeni. Kugle se dijele na pune, prazne i specijalne koje čine crna i bijela. Kako bi sve ostalo na svom mjestu kada se pokrene igra, potrebno je dodati komponentu kruto tijelo *(Rigidbody)* svakom objektu kojemu želimo primijeniti svojstva fizike poput prijenosa sile, odbijanja, usporavanja uslijed trenja ili padanja uslijed djelovanja gravitacije [1, /Rigidbody.html]. U ovom trenutku je nužno da objekti poput stola i kugli imaju navedenu komponentu. Stolu, budući da neće moći primiti silu nego je samo zadati, potrebno je uključiti svojstvo kinematičnosti *(Is Kinematic)* kako bi se izračuni fizike smanjili i time poboljšale performanse igre [1, /Rigidbody-isKinematic.html]. Osim komponente *Rigidbody*, kuglama i stolu je potrebno dodati komponentu koja sprječava prolazak drugih tijela

kroz njih odnosno omogućava detekciju sudara. Navedena komponenta se zove *Collider* i ovisi o obliku u kojem se nalazi [1, /Collider.html]. Tako će kugle imati sferni detektor sudara, a stol mrežni detektor sudara. Štapu nije potrebnu ni kruto tijelo ni detektor sudara budući da on nikada neće udariti kuglu već će to biti samo iluzija. Jedna od bitnih stvari je i kreiranje fizičkih materijala za kugle, podlogu stola i rub stola. Fizički materijal pridodajemo *Collider* komponenti objekta s time da će kugle i rub stola imati jače odskakivanje, a podloga stola nikakvo odskakivanje uz jače dinamičko trenje [2, /class-PhysicMaterial.html]. Kako izgleda početni raspored možemo vidjeti na slici 9.



Slika 9: Početni raspored glavne scene u alatu Unity [snimka zaslona]

4.1.2. Pomicanje štapa

Prva funkcionalnost koju želimo omogućiti je pomicanje štapa oko bijele kugle. Za tu potrebu kreiramo prvu skriptu u C#-u s nazivom Stick.cs. Prije pisanja koda, da bismo znali što napisati, moramo znati što želimo. Ja želim da se štap rotira oko bijele kugle po Y osi. Ako to napravim u samom Unity uređivaču u prozoru preglednik svojstva *(Inspector),* mogu vidjeti da se rotiranjem po Y osi štap rotira oko samog sebe odnosno oko svog središta. Post na forumu nam objašnjava kako trebamo kreirati prazan objekt kojem odabiremo središte postavljajući ga na željenu lokaciju, u našem slučaju na bijelu kuglu, te sam objekt, čije središte mijenjamo, postavljamo kao njegovo dijete [16]. Konkretno, kreiramo prazan objekt StickPivot čija je lokacija ista kao i od bijele kugle te mu sam objekt štapa postavljamo kao dijete. Ako sada testiramo rotiranje štapa, tj. pivota štapa po Y osi vidimo da se štap savršeno okreće oko bijele kugle iako rotiramo pivot štapa a ne sam štap. To znači da će i skripta Stick.cs biti komponenta

pivota štapa. U samoj skripti ćemo prilikom stvaranja svakog novog okvira igre, pomoću ugrađene funkcije *Update()* koja se izvršava baš prije stvaranja novog okvira [1, /MonoBehaviour.Update.html], postaviti središte pivota na iste koordinate na kojima se nalazi bijela kugla, samo malo iznad jer ne želimo vizualno preklapanje štapa i drugih objekata. Za to nam treba objekt bijele kugle kojeg definiramo kao polje tipa *GameObject*, a sam objekt bijele kugle u pregledniku svojstva skripte u alatu Unity postavimo kao vrijednost polja. Nakon postavljanja lokacije pivota, želimo rotirati pivot štapa po Y osi onoliko koliko je igrač pomaknuo miš ulijevo ili udesno odnosno po X osi površine ispod miša, samo malo brže, zato množimo pomak s 1.25 [1, /Transform.Rotate.html, /Input.GetAxis.html]. Kako to izgleda u kodu možemo vidjeti u odlomku ispod.

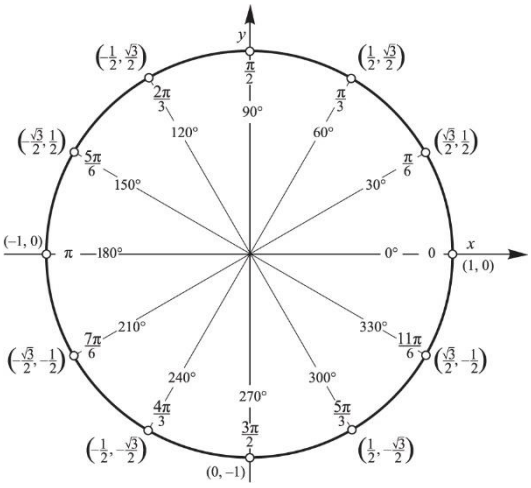
```
public GameObject whiteBall;

void Update(){
    transform.position = whiteBall.transform.position +
                         new Vector3(0f, 6f, 0f);
    transform.Rotate(0f, 1.25f * Input.GetAxis("Mouse X"), 0f);
    }
```

4.1.3. Udaranje bijele kugle

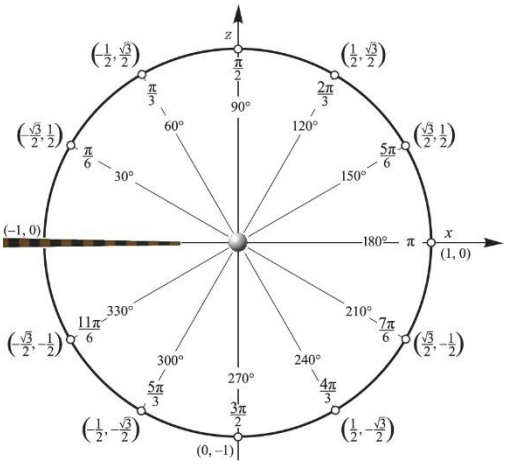
Nakon što možemo birati smjer u kojem će bijela kugla biti ispucana, trebamo implementirati samo ispucavanje bijele kugle u istom smjeru. Prvo što moramo znati je u kojem smjeru ćemo ispucati bijelu kuglu, a to dobivamo iz rotacije pivota štapa. Pogledajmo neke od mogućih slučajeva. Dakle kada mu je rotacija po Y osi jednaka 0, što mu je početna pozicija, bijelu kuglu treba ispucati duž X osi. Ako mu je rotacija 90 stupnjeva po Y osi, bijelu kuglu treba ispucati duž Z osi. Ako je rotacija 45 stupnjeva po Y osi, tada bijelu kuglu treba ispucati između X i Z osi. Možemo primijetiti da bi nam jedna odnosno dvije matematičke funkcije ovdje jako dobro došle. Radi se o sinusu i kosinusu. Našu bijelu kuglu možemo zamisliti kao centar jedinične kružnice, a samu kružnicu da joj je nula na lijevoj strani umjesto na desnoj, dakle zrcaljenu po Y, odnosno u ovom slučaju, po Z osi. Ono što nam vrati kosinus od negativnog kuta rotacije pivota štapa (kut mora biti negiran zbog zrcaljenja po Y osi jer što je na normalnoj kružnici 0 stupnjeva na našoj je 180) je jačina kojom udaramo bijelu kuglu po X osi, odnosno što nam vrati sinus je jačina po Z osi.

Koristi ćemo svojstvo C#-a gdje možemo kao povratni tip imati n-torku što smo saznali od dobrih kolega programera [18]. Tako će naša funkcija vraćati 2-torku tj. jačinu po X i Z osi te će, kao što je prikazano u kodu ispod, prvo stupnjeve morati pretvoriti u radijane jer *Cos()* i *Sin()* metode primaju radijane kao parametre [19],[20].

```
public (float,float) ComputeCoordinates (float ry) {
        ry = -ry * (float)Math.PI / 180f;
        return ((float)Math.Cos(ry), (float)Math.Sin(ry));
        }
```



Slika 10: Brojevna kružnica [17]



Slika 11: Zrcalna brojevna kružnica po Y(Z) osi (vlastito uređivanje [17])

Kada nam je jasno kako funkcionira jačina udaranja kugle po osima, ostaje samo to pretvoriti u kod. Naravno, želimo da što duže igrač drži lijevu tipku miša da to bude jači udarac, ali opet da postoji neka gornja granica budući da je i u stvarnosti ograničena jačina udaranja kugle. Tako ćemo u kodu u *Update()* metodi u skripti WhiteBall.cs, koja je komponenta bijele kugle, provjeriti je li pritisnuta lijeva tipka miša pomoću metode *GetMouseButton()* [1, /Input.GetMouseButtonUp.html]. Ako je, početna će vrijednost varijable jačine udarca *(thrust)* rasti u svakom okviru u kojem se drži lijeva tipka miš. Isto tako, igrač bi trebao znati da mu jačina udarca raste, stoga ćemo pomaknuti štap dalje od bijele kugle odnosno od svog pivota u svakom okviru kada se drži lijeva tipka miša. To radimo na način da lokalnu poziciju štapa uvećamo za promjenu vremena od stvaranja prošlog okvira [1, /Time-deltaTime.html]. Štap će ići u pravilnom smjeru jer prilikom rotacije pivota štapa, mijenja se lokalna pozicija štapa u točnim omjerima kao i rotacija tako da ne moramo dodatno računati u kojem smjeru i kojom jačinom pomicati štap već je dovoljno samo lokalnu poziciju povećati za promjenu vremena. Jednom kada otkrijemo da je otpuštena lijeva tipka miša pomoću metode *GetMouseButtonUp()* [1, /Input.GetMouseButton.html], potrebno je ispucati kuglu u smjeru koji štap pokazuje, jačinom koju je igrač odredio držeći lijevu tipku miša. Prije ispucavanja provjeravamo je li jačina ispucavanja veća od gornje granice, ako je, tada ju postavljamo na gornju granicu. Nakon toga pozivamo metodu *AddForce()* s parametrima jačina udarca pomnožena s jačinom u smjeru osi X i jačina udarca pomnožena s jačinom u smjeru osi Z te oblikom prijenosa sile *VelocityChange* koji pripada enumeraciji *ForceMode* što znači da želimo

odjednom prenijeti svu silu ignorirajući masu objekta [1, /Rigidbody.AddForce.html, /ForceMode.html]. Želimo da udarac bude trenutan i jak, a ne kontinuiran i ovisan o masi, zato koristimo *VelocityChange*, iako bi sasvim ispravno i jednako bilo, u našem slučaju, koristiti i *Impulse* način.

Testiramo li ono što smo napravili, možemo primijetiti da kugla ide u smjeru u kojem želimo jačinom kojom želimo samo što to radi jako sporo. Jedna od mogućnosti kako ubrzati fizičke radnje je promjena postavki vremena u postavkama projekta [21]. Dovoljno je vremensku skalu postaviti na željeni koeficijent, u našem slučaju neka on iznosi 2.25 [21]. Ono što nam je još bitno jest da kada se bilo koja kugla miče da bude onemogućeno udaranje bijele kugle kao i da bude skriven štap na način da isključimo njegovu komponentu za prikaz *(MeshRenderer)* [23]. Za to postavljamo jednu varijablu *isMoving* tipa *bool* kojoj vrijednost postavljamo na istinu kada se lopta udari i tada sakrijemo štap, a provjeru je li lijeva tipka miša stisnuta radimo samo ako je vrijednost varijable laž. Prilikom učitavanja svakog okvira provjerimo brzinu bijele kugle, ako je ona manja od neke relativno najmanje granice (ne mora biti nužno jednaka nuli jer to dosta kasno postiže), vrijednost varijable *isMoving* vraćamo na laž te prikazujemo štap ponovno uključujući njegovu *MeshRenderer* komponentu [22],[23]. Imamo jedan od dva glavna mehanizma napravljen, ostaje detekcija bijele kugla kada upada u rupu i njezino slobodno pomicanje po stolu.

4.1.4. Pomicanje bijele kugle

Prvo što moramo napraviti prije omogućavanja pomicanja bijele kugle jest detektirati je li ona upala u rupu ili možda slučajno ispala sa stola. Na scenu postavljamo prazne objekte koji imaju detektor sudara koji je okidač [1, /Collider-isTrigger.html] kako oni ne bi zaustavili nijednu kuglu u rupi već pustili da ona prođe. Postavimo objekte koji imaju sferni detektor sudara (oni koji se nalaze u rupi) ili pravokutni (onaj koji je ispod i oko stola) kako bismo znali da se kugla više ne nalazi na stolu. Budući da imamo veliki detektor sudara ispod stola koji će aktivirati svaka kugla koja upadne u rupu, detektori sudara u rupama nam trenutno nisu potrebni, međutim možda će kasnije biti potrebno znati u koju je rupu upala zadnja kugla kako bi se crna kugla morala ubaciti u istu rupu, zato ih je najlakše sada kreirati kada već pokrivamo tu cjelinu. Kako je već objašnjeno u izvoru [1, /Collider-isTrigger.html], prilikom detekcije sudara poziva se metoda *OnTriggerExit()*. Iako su rupe ispod stola, za svaki slučaj koristimo ovu funkciju kako se ne bi prebrzo bijela vratila natrag na stol nego tek kada izađe kompletno iz detektora sudara. Svaki detektor sudara ima oznaku „Hole“ kako bismo znali o kakvom se detektoru sudara radi [1, /GameObject-tag.html]. Ako je detektor označen kao rupa, tada brzinu koju bijela kugla ima u tom trenutku postavljamo na nulu pomoću svojstava *velocity, angularVelocity* i *useGravity*, kako nam se ne bi nastavila kretati po stolu jednom kad joj

promijenimo poziciju [24]. Također, postavljamo vrijednost varijable *waitingOnMove* tipa *bool* na istinu kako bismo prilikom učitavanja sljedećeg okvira znali da trebamo omogućiti pomicanje bijele kugle. Pomicanje omogućavamo na način da joj zamrznemo Y poziciju da ne bi upala u rupu tokom micanja, pomaknemo joj poziciju iz rupe natrag na stol točno ispred rupe u koju je upala kako ne bi slučajno pomakla drugu kuglu (najmanja je vjerojatnost da je neka druga kugla točno iznad rupe u koju je bijela upala) te iz istog razloga postavimo svojstvo kinematičnosti krutog tijela drugih kugli na istinu što je i prikazano u kodu ispod [25], [1, /GameObject.FindGameObjectsWithTag.html].

```
void ChangeRigidBody (string tag, bool on) {
        GameObject[] balls;
        balls = GameObject.FindGameObjectsWithTag(tag);
        foreach (GameObject ball in balls)
        {
            ball.GetComponent<Rigidbody>().isKinematic = on;
        }
}
```

Isto tako, želimo da igrač zna da može pomicati bijelu kuglu pa ćemo mu to vizualno dočarati treptanjem bijele kugle odnosno mijenjanjem njezine prozirnosti. To radimo na način da kreiramo generator tj. metodu koja će sama sebe pozivati dok ju ne zaustavimo, a prilagodili smo je našim potrebama od one u pronađenom video zapisu [26]. Ostaje nam još provjeriti varijablu *waitingOnMove* prilikom učitavanja novog okvira. Ako je istinita, u privremenu varijablu spremamo njenu trenutnu poziciju promijenjenu za pomak miša po X odnosno Y osi. U *FixedUpdate()* metodi radimo fizikalne promjene budući da se navedena metoda poziva ne prije svakog okvira nego prilikom fizičkih izračuna, a odvija se neovisno o specifikacijama računala (zato se i zove fiksno ažuriranje) [27]. U pozivu navedene funkcije mijenjamo poziciju bijele kugle kako bismo bili sigurni da neće proći kroz neki drugi objekt odnosno da će se svi fizički izračuni dogoditi [27], [28], [1, /Rigidbody-position.html]. U metodi *Update()* ako detektiramo otpuštanje lijeve tipke miša dok igrač miče bijelu kuglu, postavljamo vrijednost varijable koja označava micanje bijele kugle na laž, zaustavljamo generator treptanja, resetiramo prozirnost bijele kugle te isključujemo kinematičnost krutog tijela ostalih kugli [1, /MonoBehaviour.StopCoroutine.html],[29].

4.1.5. Određivanje boje

Sad kada možemo ispucati bijelu kuglu u željenom smjeru te ju micati nakon što upadne u rupu, trebamo omogućiti da dva igrača naizmjenično igraju, pokušavajući ubaciti sve svoje kugle. Prvo nam treba enumeracija za boju kugle *(BallColor)* gdje postoje stanja puna, prazna, crna i neodređena. Na početku igre boja kugle oba igrača je neodređena, nakon prve ubačene kugle boja postaje puna ili prazna te, jednom kada su sve ubačene, boja kugli postaje crna. Za tu svrhu izrađujemo klasu Igrač *(Player)* koja ima svojstva boja kugle i istinosnu varijablu

isHisTurn kako bismo znali koji igrač je na potezu. Trebat će nam i glavna klasa koja će pratiti tok igre, a nazvat ćemo ju upravitelj igre *(GameManager)* kako to i preporučuju [30], i bit će komponenta praznog objekta koji će biti zadužen za, baš kao što i naziv kaže, upravljanje igrom. Ona će imati dva objekta tipa Igrač koji će se na početku kreirati s time da će igrač 1 biti na potezu, a igrač 2, pretpostavljate, neće. Trebat će nam i jedna javna metoda u upravitelju igre koja će biti zadužena za izmjenu poteza, a ona će jednostavno promijeniti vrijednosti varijable *isHisTurn* kod oba igrača. Potrebno je i kreirati OtherBall.cs skriptu koja će biti komponenta svih kugli osim bijele. U toj skripti možemo koristiti već prije spomenutu metodu *OnTriggerExit()* u kojoj provjeravamo ima li objekt koji napuštamo oznaku „Hole“. Ako ima, znači da je ubačena kugla te samo provjerimo u objektu upravitelja igre koliko je dosad ubačenih kugli, bilo punih bilo praznih, stoga da nakon provjere povećamo broj ubačenih kugli ovisno o boji kugle koja je upravo ubačena. Ako nije nijedna od navedenih kugli ubačena, tada je ova kugla prva te njezinu boju spremamo u privremenu varijablu upravitelja igre koji u *Update()* metodi provjerava jesu li boje odlučene. Ako nisu, a imamo ubačenu prvu kuglu, boja kugli igrača koji je na potezu postaje ista kao i od navedene prve kugle koja je ubačena. Time smo riješili određivanje boje kugli igrača s time da ne smijemo zaboraviti uništiti samu kuglu jednom kada je upala u rupu budući da kugla više ne sudjeluje u igri te više nije potrebno računati njezin položaj ili vršiti druge fizičke izračune [1, /Object.Destroy.html]. Kako bismo spriječili ispadanje kugli sa stola, u već gotovim skriptama WhiteBall.cs i OtherBall.cs u *Update()* metodi provjerimo je li brzina kugle pozitivna u smjeru prema gore. Ako je, tada ju vratimo na nulu kako kugla ne bi odletjela sa stola. Budući da je brzina tipa *Vector3*, možemo joj pročitati vrijednost y varijable i promijeniti ju pridružujući joj novi vektor s nulom na y koordinati [1, /Rigidbody-velocity.html, /Vector3.html].

4.1.6. Izmjena poteza

Što se tiče same izmjene poteza, moramo kreirati sustav za otkrivanje prekršaja odnosno nedopuštenih radnji. Tako ćemo kreirati dvije metode za početak i kraj poteza. Na početku poteza, dakle prilikom otpuštanja lijeve tipke miša spremamo trenutni broj ubačenih punih i praznih kugli, postavljamo istinosne varijable za prekršaj i fatalni prekršaj na laž te boju prve udarene kugle postavljamo na neodređenu. U nastavku WhiteBall.cs skripte koristimo ugrađenu metodu *OnCollisionEnter()* koja se poziva prilikom sudara objekta s drugim objektima koji imaju detektor sudara *(Collider)* [1, /Collision-gameObject.html]. U navedenoj metodi uspoređujemo oznaku objekta kako bismo znali o kojoj boji kugle se radi, pomoću metode *CompareTag()* [1, /GameObject.CompareTag.html]. Spremanje boje kuglu u privremenu varijablu *firstHitBall* radimo samo ako su boje određene i radi se o prvoj udarenoj kugli. I konačno, radimo provjeru prekršaja odnosno treba li izmijeniti potez ili ne. Na kraju poteza, ako su boje i dalje neodređene, znači da igrač nije ubacio niti jednu pa se mijenja

potez. Isto tako se mijenja potez ako je broj ubačenih kugli trenutnog igrača isti kao i prije poteza, što znači da nije ubacio nijednu svoju kuglu. Fatalni je prekršaj ako je igrač prvo udario kuglu drugog igrača, a ne svoju, ako je udario prvo crnu kuglu, a ne smije ubaciti crnu ili ako uopće nije udario niti jednu kuglu. Fatalni prekršaj označava ne samo izmjenu poteza nego i pravo drugog igrača da pomakne bijelu kuglu gdje god on želi. Prekršaj je i ako igrač treba ubaciti crnu kuglu, a to nije uspio. Bitno je napomenuti da sve ove provjere za prekršaj radimo samo ako je podignuta zastavica za provjeru prekršaja, a ona je podignuta dokle god crna nije ubačena na neregularan način. Dakle, ako igrač ubaci crnu prije nego što smije, tada nije potrebno provjeravati prekršaje jer je navedeni igrač svakako izgubio bez obzira učinio prekršaj ili ne. Ukoliko je igrač prvo dirao svoju kuglu, uspio ubaciti barem jednu svoju kuglu i pritom nije ubacio crnu, uspio je ne napraviti prekršaj i slobodno može nastaviti sa sljedećim potezom.

4.1.7. Kraj igre

Igra završava ubacivanjem crne kugle u rupu. Cilj je ubaciti crnu nakon što na stolu više nema kugli igračeve boje. Naravno, ako crna upadne prije nego što je dozvoljeno ju ubaciti, igrač koji ju je ubacio je gubitnik. Dakle prilikom ubacivanja kugle u rupu u OtherBall.cs skripti ćemo provjeriti trenutni broj ubačenih kugli iste boje. Ako on, nakon što ga uvećamo, iznosi sedam znači da je igraču, čija je boja ista kao promatrane kugle, dozvoljeno ubaciti crnu kuglu. Igračevu boju ne mijenjamo još već samo podižemo zastavicu kako bismo na kraju poteza mogli provjeriti je li počinio prekršaj udarivši prvu nedozvoljenu kuglu. Tek mu nakon provjere prekršaja mijenjamo boju. Tako ćemo mi na upad crne kugle u rupu jednostavno provjeriti smije li ju igrač koji je na potezu ubaciti. Ne možemo odmah oglasiti pobjednika budući da, iako je crna u rupi, sve kugle moraju stati kako se ne bi dogodilo da bijela kugla upadne u rupu što bi značilo poraz igrača koji je upravo odigrao potez. Dakle, ako je upala crna kugla u rupu i igrač koji je na potezu ju smije ubaciti, postavljamo ga u varijablu pobjednika te podižemo zastavicu o provjeri prekršaja jer također ubacivanje crne kugle uz napravljen prekršaj znači poraz. Ako igrač koji je ubacio crnu kuglu nema pravo na to, spuštamo zastavicu o provjeri prekršaja te spremamo drugog igrača u varijablu pobjednika. U metodi *MoveFinished(),* koja se poziva na kraju poteza odnosno kada sve kugle stanu, provjeravamo je li podignuta zastavica za kraj igre. Ako je podignuta ona i zastavica za prekršaje, provjeravamo prekršaje te ukoliko postoje mijenjamo pobjednika. Ista stvar je se događa u *Update()* metodi u WhiteBall.cs skripti prilikom provjere je li upala bijela kugla. Ako se dogodio fatalni prekršaj, zamijenimo pobjednika i zaustavljamo igru.

4.1.8. Predviđanje prvog sudara

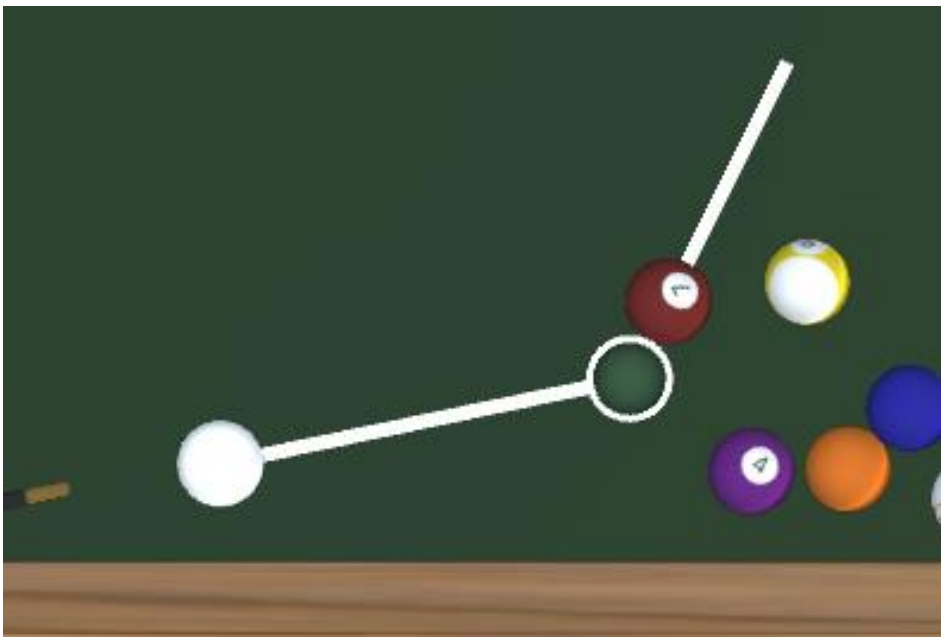
Igranje bez pomoćne linije je dosta teško i izazovno, a želimo da naša igra bude pristupačna te prilagodljiva. Zbog toga ćemo pokušati predvidjeti prvi sudar bijele kugle kako bi igrač prije udaranja bijele kugle znao gdje će ona završiti odnosno u što će prvo udariti. Za tu potrebu bijeloj kugli unutar preglednika svojstava dodajemo komponentu crtača linije *(LineRenderer)* pomoću koje ćemo moći prikazati liniju kojom će se bijela kugla kretati do prvog sudara [31]. S druge strane, u *Start()* metodi u skripti WhiteBall.cs kreiramo primitivni oblik – sferu koja će vizualizirati mjesto prvog sudara bijele kugle i drugog objekta [1, /GameObject.CreatePrimitive.html]. Postavimo je na neutralnu poziciju, s veličinom kao što je bijela kugla te joj uklonimo detektor sudara kako sama pomoćna kugla ne bi pomakla druge kugle [32]. Želimo da pomoćna kugla bude zelene boje kao stol te da ima obris bijele boje kao linija koja spaja nju i bijelu kuglu. Za to koristimo osjenčivač *(Shader)* koji smo preuzeli s interneta od dobrog kolege programera [33]. Postavimo materijal pomoćne kugle, dodamo komponentu za crtanje linije *(LineRenderer)* te joj postavimo kraj i početak u središtu same kugle za početak [1, /Renderer-material.html, /GameObject.AddComponent.html]. Sve dok igrač ne udari bijelu kuglu odnosno sve dok je vidljiv štap, želimo da bude vidljiva i pomoćna kugla kao i linija do nje. Dakle, u *Update()* metodi pozivamo metodu *DrawGuide()* prosljeđujući joj smjer u kojem će bijela kugla ići, znači x i z parametre. Uključujemo komponentu za crtanje crte bijele kugle kao što uključujemo i pomoćnu kuglu [1, /GameObject.SetActive.html],[34]. Video objavljen na platformi YouTube nam je jako pomogao objasnivši kako da „ispucamo“ kuglu u određenom smjeru do prvog objekta što je točno ono što nama treba u ovom slučaju [35]. Kreiramo zraku *(Ray)* s početkom u središtu bijele kugle, u smjeru u kojem štap pokazuje (x i z parametri) [35]. Ako nam metoda *Physics.SphereCast()* vrati istinu znači da se dogodio sudar i ono što mi želimo je postaviti pomoćnu kuglu na mjesto sudara pomaknutom za polumjer pomnožen s normalom kako bi ona dirala sam objekt, a ne prolazila kroz njega [35], [36].

```
RaycastHit raycastHit;
Ray ray = new Ray(start, direction);
if (Physics.SphereCast(ray, radius, out raycastHit)) {
      end = raycastHit.point + radius * raycastHit.normal;
      }
```

Također kraj linije, koja spaja bijelu i pomoćnu kuglu, postavljamo na istu točku. Potrebno je kreirati još jednu pomoćnu liniju koja će, ako se radi o kugli a ne o rubu stola, predvidjeti gdje će se udarena kugla odbiti kako bi igrač lakše naciljao rupu. Početak druge pomoćne linije se nalazi u središtu pomoćne kugle, odnosno u istoj točki kao i kraj prve pomoćne linije. Druga pomoćna linija treba ići od središta pomoćne kugle u smjeru točke udarca koja se nalazi u varijabli *raycastHit.point*. Međutim linija ne treba tu stati već treba proći tu točku udarca i bit

dugačka onoliko koliko je jak udarac, odnosno kolika je razlika u kutu smanjena za određeni koeficijent [1, /Vector3.Angle.html]. Udarac je jači što je razlika kuta normale i kuta razlike kraja i početka prve pomoćne linije veća.

```
Vector3 endLine = raycastHit.point - raycastHit.normal *
Vector3.Angle(raycastHit.normal, end - start) / 150f;
```



Slika 12: Pomoćna kugle i dvije pomoćne linije [snimka zaslona]

4.1.9. Slučajni razmještaj kugli

Napravili smo početni razmještaj kugli još prije pisanja ijedne linije koda, međutim ne želimo da kugla bude na istom mjestu svaki put kada se pokrene nova igra kako se ne bi dogodio isti rasplet. Stoga ćemo napraviti klasu o poziciji kugle *(BallPosition)* koja će imati svojstvo tipa *Vector3* i označavat će samu točku pozicije te jednu zastavicu je li pozicija zauzeta ili nije. U *Awake()* metodi u skripti *GameManager.cs* inicijaliziramo svih 14 mogućih pozicija za smještaj kugli [1, /MonoBehaviour.Awake.html]. Crna kugla se uvijek nalazi u sredini trokuta i njena pozicija se neće mijenjati. U nastavku uzimamo nasumičan broj koji će označavati koliko će se trokut pomaknuti po X osi kako se također ne bi dogodio isti rasplet. Za svaku kuglu, osim crne, generiramo nasumičan broj od 1 do 14 te na prvu slobodnu poziciju smještamo kuglu. Kada su sve kugle smještene, ovaj put uključujemo i crnu, pomičemo poziciju kugle za već generirani nasumični pomak.

4.2. Izrada klasičnih mogućnosti

U ovom dijelu dokumentacije prikazujem izradu klasičnih mogućnosti koje bi danas trebala imati svaka računalna igra. To su elementi poput zvuka, glavnog izbornika, postavki igre, pauziranja igre te njezino spremanje i nastavljanje.

4.2.1. Zvučni efekti

U igri biljara nije potrebna neka glazba u pozadini, ali je potrebno auditivno dočarati što se događa u samoj igri. Tako sam odlučio preuzeti zvukovni zapis videa na YouTube-u u kojem su odlični biljarski zvučni efekti [37]. Skinuti zvukovni zapis sam izrezao na više zapisa te ih uredio u alatu Audacity tako da svaki zapis počinje u prvoj mogućoj milisekundi reproduciranja zvuka kako bi se u trenutku sudara odmah čuo i zvuk sudara bez odgode. Tako u skriptama OtherBall.cs i WhiteBall.cs dodajemo dva javna svojstva tipa *AudioClip* (jedan za sudar s drugom kuglom i jedan za ubacivanje u rupu) te jedno privatno svojstvo tipa *AudioSource* [38]. Tada u preglednik svojstava skripti povučemo zvučni zapis na mjesto svojstva tipa *AudioClip-a* koja smo kreirali [38]. Zvuk sudaranja kugli želimo reproducirati u samom trenutku sudara, dakle koristimo metodu *OnCollisionEnter()* s time da bi bilo odlično kada bi jačina zvuka bila u korelaciji s jačinom udarca [38]. Tu nam pomaže parametar tipa *Collision* same metode koji ima svojstvo relativne brzine i njezine jačine i to je upravo ono što nama treba [38]. Dakle, ako je objekt u koji se kugla zabila druga kugla, reproduciramo zvuk budući da nemamo zvuk sudaranja kugle i rubnika. Ne moramo provjeravati je li kugla crna ili bijela budući da će one proizvesti zvuk u trenutku sudara sa svim kuglama.

```
if (collision.gameObject.CompareTag("Solid") ||
collision.gameObject.CompareTag("Stripe")) {
      audioSource.PlayOneShot(collisionClip,
      collision.relativeVelocity.magnitude / 80f);
      }
```

Što se tiče rupe, njezin detektor sudara je okidač tako da koristimo metodu *OnTriggerEnter()* također jednom reproducirajući zvuk upadanja u rupu, međutim ovaj put nam glasnoća nije bitna jer želimo da sva ubacivanja u rupu zvuče isto [38].

Potrebno je još samo kod udaranja bijele kugle, dakle u dijelu skripte WhiteBall.cs gdje detektiramo otpuštanje lijeve tipke miša, dodati istu naredbu kako bismo reproducirali zvuk udaranja bijele kugle. Nije potrebno da neki udarac bijele kugle bude glasniji od drugog, ali kada bismo to htjeli implementirati, dodali bismo samo jačinu udarca *(thrust)* kao drugi parametar metode *PlayOneShot()* [38].

4.2.2. Glavni izbornik

Kada bismo imali igru, kao što je naša u ovom trenutku, samim ulaskom u nju odmah bismo se našli na glavnoj sceni na početku nove igre. Ono što je profesionalnije i danas nekakav standard je izrada glavnog izbornika. U našem glavnom izborniku u pozadini će biti stol sa štapom i par kugli na njemu, kamera će polako ići lijevo desno te ćemo imati izbor od nastavljanja igre (ako postoji nedovršena partija), započinjanja nove igre, mijenjanja postavki te izlaza iz same igrice. Prvo postavljamo stol, kugle i štap slično kao i na glavnoj sceni, međutim kameru ne postavljamo iznad stola u ortografskom načinu projekcije već tako da se stol vidi pod određenim kutom s perspektivnom postavkom projekcije [2, /class-Camera.html]. Rotiranje kamere će obavljati skripta RotateCamera.cs koju ćemo dodati kao komponentu glavnoj kameri. U navedenoj skripti imamo svojstvo brzina i smjer gdje je brzina realan pozitivan broj, a smjer -1 ili 1. Tako u *Update()* metodi rotiramo kameru oko Y osi, a budući da je malo nakrenuta koristimo *RotateAround()* metodu umjesto *Rotate()* [39]. Jednom kada dođe do određenog kuta lijevo ili desno, mijenjamo smjer. Kreirat ćemo gumbove za spomenute opcije, za predložak gumba ćemo koristiti preuzetu sliku [43], a tekst ćemo oblikovati komponentom *TextMeshPro* [42].

Kako bi nam sve nadalje funkcioniralo poput postavki, pauziranja i nastavljanja igre, moramo prilagoditi malo upravitelja igre. I u glavnom izborniku i u glavnoj sceni odnosno samoj igri nam je potreban upravitelj igre za kojeg želimo da postoji samo jedna instanca kroz cijelu igru. Kako su dobri kolege programeri objasnili na internetu [30], u *Awake()* metodi u skripti GameManager.cs moramo pozvati metodu *DontDestroyOnLoad()* kako nam se ne bi prilikom učitavanja nove scene obrisala trenutna instanca upravitelja igre [30]. Nadalje, ako u glavnom izborniku prvi put kreiramo upravitelja igre te započnemo igru i zatim se vratimo u glavni izbornik, upravitelj igre će se još jednom stvoriti te ćemo imati dvije instance [30]. To nikako ne želimo, stoga uklanjamo upravitelja igre iz glavne scene i glavnog izbornika te kreiramo pomoćnu scenu koja će se prva učitati i na nju se nikad nećemo vratiti, a njena je uloga samo kreirati upravitelja igre i učitati glavni izbornik [30],[1, /SceneManagement.

SceneManager.LoadScene.html]. Dodamo delegat na mijenjanje scene te u vlastitoj metodi provjeravamo o kojoj se sceni radi i skladno tome izvodimo kod koji je nužan za funkcioniranje iste scene [1, /SceneManagement.SceneManager-sceneLoaded.html]. Budući da se upravitelj igre nalazi u sceni u kojoj je on jedini element, ne možemo povući grafičke elemente poput gumbova ili grupa u upravitelj svojstava već to moramo raditi u samom kod u *OnSceneLoaded()* metodi [44]. Dakle, u navedenoj metodi provjeravamo o kojoj se sceni radi

Slika 13: Glavni izbornik igre [snimka zaslona]

i ovisno o njoj, tražimo grafičke elemente u toj sceni prema imenu u polju svih objekata [1, /Resources.FindObjectsOfTypeAll.html] te pronađenom objektu dodajemo delegat na klik gdje izvodimo naredbe o promjeni scene, grafičkoj promjeni ili izlazu iz igre [45].

```
newGameButton.onClick.AddListener(delegate { File.Delete(path);
      LoadScene("MainScene"); newGameButton.enabled = false; });
exitButton.onClick.AddListener(ExitGame);
```

4.2.3. Pauziranje i nastavljanje igre

Nešto kompliciranija radnja za implementaciju je pauziranje odnosno spremanje i nastavljanje igre. Ne želimo da ukoliko jedan od igrača želi promijeniti nešto za vrijeme igre, izgubi napredak, nego želimo omogućiti mogućnost pauziranja. U tu svrhu kreiramo izbornik za pauziranje. Pauziranje igre je vrlo jednostavno, u *Update()* metodi u skripti GameManager.cs provjeravamo je li stisnuta tipka za izlaz *(Esc)*, ako je, spremamo trenutnu vremensku skalu *(Time.timeScale)* i postavljamo ju na nulu [40] te aktiviramo grafičko sučelje s dva gumba – jedno za nastavak igre, jedno za izlazak u glavni izbornik. Ukoliko igrač opet

pritisne tipku za izlaz ili klikne na gumb Nastavi, nastavljamo igru odnosno isključujemo grafičko sučelje i kursor te vraćamo vremensku skalu na spremljenu vrijednost.

Na pritisak gumba za povratak u glavni izbornik pozivamo metodu za spremanje igre. U videu na platformi YouTube, savršeno je objašnjeno kako kreirati sustav za spremanje i učitavanje igre koji ćemo mi slijediti u izradi našeg sustava [41]. Prvo moramo kreirati klasu koja se može spremiti u binarnom obliku, dakle sve varijable koje želimo spremiti (poput podataka o kuglama i igračima) moramo pretvoriti u jedan od četiri jednostavna tipa podataka (istinosni, cjelobrojni, realni, znakovni) [41]. To radimo jer podatke ne želimo spremiti u čitljivom obliku već u binarnom, a binarno se mogu zapisati samo četiri navedena tipa podataka [41]. Željene varijable pretvaramo u osnovne tipove i spremamo u jednostavan objekt koji je instanca klase s atributom *Seriazable* kako bi se mogao spremiti u binarnu datoteku [41]. Potrebno nam je znati koje su kugle u igri, njihove pozicije i rotacije, koliko je kugli koje boje ubačeno, koji igrač je na potezu, jesu li boje odlučene te ako da, koja je boja kojeg igrača. Nakon kreiranog binarnog pretvarača *(BinaryFormatter)* i otvorenog datotečnog kanala *(FileStream)*, pretvaramo podatke u binarni oblik i zapisujemo u datoteku s bilo kojim nastavkom [41].

Slika 14: Pauzirani izbornik u igri [snimka zaslona]

Igraču ne želimo omogućiti prijelaz u glavni izbornik sve dok traje potez odnosno sve dok se ijedna kugla miče. Stoga ako je igra pauzirana u trenutku kada je štap vidljiv, znamo da se nijedna kugla ne miče te možemo uključiti gumb Izbornik. Ako štap nije vidljiv, isključujemo gumb Izbornik te prikazujemo popratni tekst uključujući mu vidljivost (slika 14) [49]. Prijelaz i glavni izbornik ne želimo da bude grub i trenutan već da se lagano zacrni ekran

i animirano prikaže glavni izbornik. To lako radimo prateći video [26] na način da kreiramo objekt crne boje koji će prekriti cijeli zaslon te dodamo animacije za mijenjanje prozirnosti objekta u oba smjera. Tada u glavnom izborniku klikom na gumb Nova igra/Nastavi ili u pauziranom izborniku klikom na gumb Izbornik, prije učitavanja scene, pokrećemo animaciju i tek kada ona završi, koristeći generator, vraćamo vremensku skalu na spremljenu vrijednost te mijenjamo scenu [26]. Kako to izgleda, prikazano je u kodu ispod.

```
public void LoadScene(string sceneName) {
      Time.timeScale = timeScale;
      StartCoroutine(SceneLoad(sceneName));
      }

public IEnumerator SceneLoad(string sceneName) {
      sceneTransition.SetTrigger("end");
      yield return new WaitForSeconds(1.5f);
      SceneManager.LoadScene(sceneName);
      }
```

Ako dolazimo iz glavnog izbornika klikom na gumb Nastavi, potrebno je postaviti scenu kao što je bila kada je igra spremljena. Ako postoji datoteka s istim imenom kao što smo ju spremili, u kojoj su svi podatci o igri, pretvaramo binarni zapis sadržaja datoteke u objekt iste klase, kao što je bio kada smo ga spremali u binarnu datoteku [41]. Potrebno je sve pripremiti za nastavak igre, dakle svojstva pročitanog objekta pretvaramo u nama poznate varijable upravitelja igre [41]. Kugle koje su bile ubačene će se svejedno pojaviti na sceni na svojoj zadanoj poziciji, zato je potrebno provjeriti je li kugla u igri odnosno je li ubačena ili ne. Ako je ubačena, uništavamo kuglu metodom *Destroy()*, a ako nije ubačena, postavljamo joj poziciju i rotaciju prema pročitanim svojstvima [1, /Object.Destroy.html]. Prilikom klika na gumb Nova igra u glavnom izborniku, brišemo datoteku o staroj igri ako ona postoji. Tek kada se u pauziranom izborniku pritisne gumb Izbornik, kreiramo binarnu datoteku i spremamo stanje igre u nju. Tako u glavnom izborniku provjeravamo postoji li navedena datoteka. Ako postoji prikazat ćemo gumb Nastavi, a ako ne postoji znači da se nema koja igra nastavit te isti gumb sakrivamo.

4.2.4. Postavke igre

Kreiranje postavki igre nije pretjerano težak posao jednom kada nam je upravitelj igre ispravno postavljen. Prvo kreiramo grafički izgled postavljanjem gumbova, klizača i padajućih izbornika na platno glavnog izbornika. Ono što želimo da bude moguće mijenjati je vidljivost pomoćne linije, njezinu boju i dužinu, glasnoću zvukova u igri te teksturu stola. Ako igrač uključi pomoćnu liniju, može izabrati njezinu dužinu pomoću klizača te njezinu boju pomoću padajućeg izbornika, s time da bira između crne, bijele i žute. Ako ju isključi potrebno je isključiti mogućnost mijenjanja boje i dužine linije [49]. Glasnoća se zvukova zadaje pomoću klizača i tada se mijenja svojstvo glasnoće statične klase *AudioListener*, a tekstura stola se bira iz

padajućeg izbornika i moguće je odabrati smeđu, bijelu i tamnu [50]. Postavke ćemo prilagoditi temi igre koristeći kugle za označavanje trenutno postavljenih vrijednosti. Gotovu sliku kugli sam preuzeo s interneta [48], te ju pomoću alata GIMP izrezao i razdvojio u sliku svake kugle zasebno. Na slici 15 je prikazano kako izgledaju postavke.

Slika 15: Izgled postavki u igri [snimka zaslona]

Želimo da klikom na gumb Postavke u glavnom izborniku, glavni gumbovi (Nastavi, Nova igra, Postavke, Izlaz) nestanu, a pojave se grafički elementi postavki. Tako ćemo kreirati dva prazna objekta koji će služiti kao grupiranje grafičkih elemenata, s time da će imati komponentu grupa platna *(Canvas Group)*, gdje će glavni gumbovi imati vidljivost postavljenu na jedan, a elementi postavki na nula [46]. Dakle, u skripti GameManager.cs postavljamo slušatelj klika na gumb Postavke da na sam klik pozove metodu *FadeOut()* [45]. *FadeOut()* metodi ćemo proslijediti grupu platna, u ovom slučaju glavne gumbove, kako bismo znali koju grupu prikazati, a koju sakriti. Tu ćemo koristiti metodu naših kolega programera gdje se prozirnost vidljive grupe platna smanjuje svakim prolazom kroz generator sve dok ne postane jednaka nuli [47]. Tada ćemo isključiti navedenu grupu kako, iako je nevidljiva, ne bi se moglo kliknuti na njezine elemente. S druge strane uključujemo drugu grupu platna te joj na isti način povećavamo vidljivost sve dok nije kompletno vidljiva [47]. Isto to se događa kada igrač u postavkama klikne na gumb Povratak, samo što će ovaj put grupe platna biti u zamijenjenim ulogama. Kako izgleda smanjivanje vidljivosti unutar generatora, možemo vidjeti u kodu ispod.

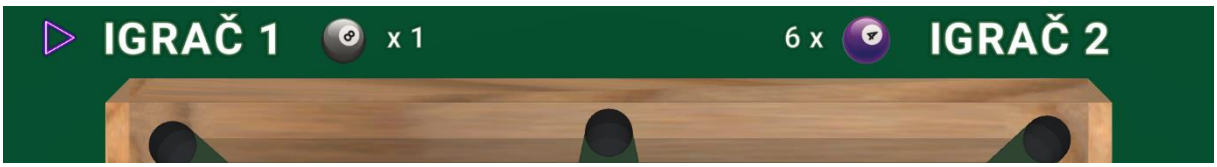
```
float time = 0f;
while (buttonGroup1.alpha > 0) {
      time += Time.deltaTime;
      buttonGroup1.alpha = Mathf.Clamp01(1.0f - time);
      yield return null;
      }
buttonGroup1.gameObject.SetActive(false);
```

Kao što je već rečeno, prilikom učitavanja scene glavnog izbornika, grafičkim objektima dodajemo slušatelje promjene te koju metodu trebaju pozvati kada se dogodi promjena vrijednosti odnosno koje naredbe izvršiti. Kreirat ćemo i pet javnih varijabli u kojima će biti zapisane postavke igre. Njih također možemo spremiti u binarnu datoteku kako se ne bi prilikom svakog pokretanja igre poništila, međutim u ovom trenutku to nije potrebno. Bitno je na promjenu neke postavke promijeniti i vezanu javnu varijablu kako bi se dalje u kodu znalo što koristiti. Konkretno, ako igrač isključi pomoćnu liniju nećemo uopće predviđati mjesto prvog sudara. Isto tako prilikom učitavanja glavne scene, postavit ćemo teksturu stola na onu vrijednost koja je u navedenoj javnoj varijabli, kao što ćemo i dužinu druge pomoćne linije pomnožiti s jačinom sudara.

4.2.5. Grafičko sučelje glavne scene

Igrici trenutno nedostaje još samo grafički dio na glavnoj sceni kako bi igrači znali tko je na potezu te koliko mu je kugli još preostalo. Tako ćemo pomoću već spomenute komponente *TextMeshPro* kreirati dva natpisa za svakog igrača s jedne strane. Preuzeo sam sliku trokuta [51] koja će označavati koji je igrač na redu. Lakše je kreirati dvije instance i po potrebi isključivati odnosno uključivati instancu ovisno o tome je li igrač na potezu ili ne. Na početku nove igre, trokut je uključen kod lijevog (prvog) igrača, a kod stare igre kod onog koji je na potezu. U već kreiranoj metodi *ChangeTurns()* gdje mijenjamo svojstvo igrača je li na potezu ili nije, možemo dodati da se sukladno tome i uključi ili isključi trokut pored njegovog imena koristeći svojstvo *activeSelf* tako da postavimo aktivnost trokuta na negaciju trenutne aktivnosti [1, /GameObject-activeSelf.html]. Tako ćemo kreirati i mjesto za kuglu kako bi igrač znao koja je njegova boja kao i koliko još kugli mu ostaje za ubaciti. Na početku je navedeno skriveno i nepoznato sve dok ne upadne prva kugla. Na upad prve kugle, osim što određujemo boje, uključit ćemo i navedeno grafičko sučelje. Sliku pune kugle premještamo kod igrača 1 ili 2 te dodajemo tekst oblika „x N“ gdje je N broj kugli koji mu je preostao za ubaciti [42]. Isto radimo i za sliku prazne kugle koje su preuzete s već spomenutog izvora [48]. Kod svakog ubacivanja kugle u rupu, pozvat ćemo metodu za osvježavanje grafičkog sučelja koja će izračunati novi broj preostalih kugli te ga prikazati. Ukoliko je zadnja kugla jedne boje ubačena, mijenjamo sliku pune ili prazne kugle u sliku crne kugle te postavljamo da je N jednak jedan budući da treba ubaciti samo jednu crnu kuglu koliko je to i moguće [52]. Na ubacivanje crne kugle postavit ćemo N na nulu što će ujedno označavati i kraj igre. Slika 16 prikazuje grafičko

sučelje u situaciji kada je igrač 1 na potezu i mora ubaciti crnu kuglu, a igrač 2 ima još šest punih kugli za ubaciti.



Slika 16: Grafičko sučelje glavne scene [snimka zaslona]

4.2.6. Animacija na kraju igre

Nakon ubacivanja crne kuglu u rupu, želimo prikazati i lijepu animaciju koja će označiti da je igra gotova, prikazati ime pobjednika te učitati scenu glavnog izbornika. Tako ćemo inspirirani animacijom predložene Unity igre za obrazovanje [53] u posebnu grupu platna dodati sustav čestica, *TextMeshPro* i sliku koja će biti polu prozirna kako bi zatamnila igru glumeći pozadinu. Budući da sam preuzeo projekt [53] te kreirao sustav čestica, kao što je i u njemu samom na kraju nogometne utakmice, bilo je vrlo lako podesiti postavke. Promijenio sam trajanje čestica, njihovu brzinu, 3D veličinu, početnu rotaciju, početnu boju, broj emitiranih čestica po vremenskoj jedinici, ispaljivanja, oblik, ograničenje brzine i snage tokom životnog vijeka, promjenu boje, rotacije i veličine tokom životnog vijeka te dodao strujanje prateći izvor [53], ali mijenjajući neke stvari kako bi ih više prilagodio mojem slučaju uz preveliku pomoć lekcije [54]. Kako na kraju izgleda animacija za kraj igre, vidimo na slici 17.

Aktiviranje same animacije je dosta jednostavno, budući da su sve stvari već spomenute i poznate. Nakon ubacivanja crne kugle i zaustavljanja svih kugli, znamo tko je pobjednik te pozivamo metodu *FinishGame()* u kojoj postavljamo vrijednost tekst objekta *TextMeshPro* na naziv igrača koji je pobijedio, aktiviramo grupu platna za kraj igre, polako joj postavljamo vidljivost na jedan već kreiranim generatorima, spuštamo zastavicu koja označava

da je igra u tijeku kako se ne bi mogao micati štap, ispucati bijela kugla ili aktivirati izbornik za pauzu te učitavamo scenu glavnog izbornika jednom kada je animacija gotova.

Slika 17: Animacija za kraj igre [snimka zaslona]

4.3. Testiranje

Vjerojatno psihički najteža faza dolazi prije samog kraja, a to je kada je igrica napravljena te je potrebno proći kroz sve moguće scenarije i uvjeriti se da sve radi kako treba. Idealno bi bilo kada bi sve radilo iz prve, ali to apsolutno nikad nije slučaj. Nikako ne treba misliti da je sve gotovo kada je sve napravljeno, baš suprotno, u mojem slučaju tada sam bio na otprilike 70% napretka, a ne 100%. Odlučio sam prvo napraviti cijelu igricu i tek onda ići na testiranje budući da sam sâm u timu. Možda bi bilo bolje da sam testirao igricu dok sam pisao kod, međutim baš sam se htio koncentrirati na isključivo jednu stvar te ju raditi dok ona nije napravljena dovoljno da ostalo radi. Tako sam prvo provjeravao radi li logika igre odnosno izmjena poteza i detektiranje prekršaja što je imalo dosta nepravilnosti. Također, kod spremanja i nastavljanja igre su se znale dogoditi stvarno čarobne, fizički neobjašnjive stvari, koje sam, srećom, uspio popraviti. Tako je nakon svakog ispravka bilo potrebno testirati svaki mogući slučaj kako bih se uvjerio da sam popravio neispravnu stavku, a opet da nisam onesposobio nešto drugo što je dotad radilo. To je značilo odigrati barem pedesetak partija, od kojih bi pola bilo sa spremanjem i nastavljanjem, a pola odigrano bez pauziranja. Jednom kada više nisam mogao otkriti nepravilnosti, završio sam fazu testiranja i počeo izrađivati ovu dokumentaciju.

5. Zaključak

U ovom završnom radu imali smo priliku vidjeti kako teče proces izrade jedne 3D računalne igre od same nule do testiranja. Ideja je bila samo s predznanjem o programiranju u C++ programskom jeziku napraviti računalnu igricu. Do izrade ove igrice, znao sam napraviti jedino konzolnu aplikaciju i to je stvarno prvi korak za izradu nečeg većeg i složenijeg. Uz naporan rad, čitanjem službene dokumentacije, proučavanjem službenih videa Unity-a, neslužbenih videa ostalih kreatora igrica i dobrim pretraživanjem interneta, igru sam uspješno kreirao. Naravno, u trenutku izrade igre tek sam se upoznavao s C# programskim jezikom i samom idejom izrade računalnih igara što znači da je sve moglo biti napravljeno bolje i jednostavnije, međutim ovo remek djelo funkcionira, stoga ne treba ga popravljati. Mogu reći da je zaista bitno shvatiti kako programerski razmišljati odnosno kako reći računalu da napravi točno ono što želimo. Itekako su bitni sadržaji koji se istražuju jer ništa od ovog ne bih znao bez tako dobrih videa, objava ili dokumentacije čijim sam autorima vječno zahvalan. Dakle, vrlo je važno znati i kako pretraživati.

Najviše se koristio alat Unity kao platforma za izradu same igre, koji je sam po sebi ekstremno jak i napredan alat, međutim nije dovoljan. Modele sam tako izradio u alatu Blender, programski kod pisao u alatu Visual Studio, grafičke datoteke uređivao u alatu GIMP, a audio zapise u alatu Audacity. S alatom Unity sam bio već upoznat budući da sam prije ovog rada kreirao jednu vrlo jednostavnu 2D igru, dok mi je alat Blender bio potpuno nov i nepoznat.

Pretpostavka je bila, dakle, da je dovoljno znati C++ i da se može kreirati računalna igra. To nikako nije dovoljno, mogli smo vidjeti da se koristila i matematika kod smjera kugle kao i kod predviđanja prvog sudara te, ono itekako bitno, fizika. Zbog fizike su se kugle odbijale, zaustavljale i upadale u rupe i to je nešto što smo morali znati prije izrade same igrice. Dakle, poznavanje jednog općenitog programskog jezika je dovoljno za naučiti drugi napredniji programski jezik, međutim u izradi računalne igre jako su bitne matematika i fizika i ako nemamo predznanje o navedenim znanostima, nećemo ju moći tako lako izraditi. Jedna računalna igra se dosta razlikuje od konzolne ili bilo kakve druge aplikacije budući da je to beskonačna petlja u kojoj se svaki okvir stvara i izvodi svoj dio instrukcija. Sve je to lako za shvatiti kada vas educiraju pravi ljudi i kada imate volje.

Popis literature

[1] Unity Technologies, „Unity – Scripting API“, 2020. [Na internetu]. Dostupno: https://docs.unity3d.com/ScriptReference [pristupano 28.7.2020.].

[2] Unity Technologies, „Unity - Manual: Unity User Manual (2019.4 LTS)“, 2020. [Na internetu]. Dostupno: https://docs.unity3d.com/Manual/ [pristupano 28.7.2020.]

[3] „Blender (software)“, (bez dat.). u *Wikipedia, the Free Encyclopedia.* Dostupno: https://en.wikipedia.org/wiki/Blender_(software) [pristupano 28.7.2020.]

[4] „Unity (game engine)“, (bez dat.). u *Wikipedia, the Free Encyclopedia.* Dostupno: https://en.wikipedia.org/wiki/Unity_(game_engine) [pristupano 28.7.2020.]

[5] Unity Technologies, „Unity Learn“, 2020. [Na internetu]. Dostupno: https://learn.unity.com/ [pristupano 28.7.2020.]

[6] Dawid Michalczyk, „Pool table“, 2008. [Slika]. Dostupno: https://www.art.eonworks.com/gallery/misc/pool_table-200815.html [pristupano 14.2.2020.]

[7] blended, (9.4.2017.) „How To Cut A Hole In Object In Blender? - Quick Tutorial“, *YouTube* [video datoteka]. Dostupno: https://www.youtube.com/watch?v=tkbU1bGOzWM [pristupano 14.2.2020.]

[8] „Symmetrize“, (bez dat.). u *Blender Manual* [Na internetu]. Dostupno: https://docs.blender.org/manual/en/latest/modeling/meshes/editing/mesh/symmetrize.html [pristupano 14.2.2020.]

[9] shehraz2010, „ASHSEN | shehraz2010 | Flickr“, 18.8.2010. [Slika]. Dostupno: https://www.flickr.com/photos/53111257@N04/4905511890/in/photostream/ [pristupano 14.2.2020.]

[10] axungroup, „Pool“, 28.9.2006. [Komprimirana datoteka]. Dostupno: https://www.turbosquid.com/3d-models/free-max-mode-fun/322935 [pristupano 14.2.2020.]

[11] „Dark wood plank texture“, (bez dat.). [Slika]. Dostupno: https://cutewallpaper.org/down.php?file=/21/dark-wood-texture-hd/Dark-Wood-Plank-Texture-Vector-Art-and-Graphics-freevector.com.jpg [pristupano 14.2.2020.]

[12] „Setmax Multicolor Wood Series Tiles“, (bez dat.). [Slika]. Dostupno: https://www.indiamart.com/proddetail/wood-series-tiles-16573381373.html [pristupano 14.2.2020.]

[13] Blender, (18.7.2019.) „UV Unwrapping - Blender 2.80 Fundamentals“, *YouTube* [video datoteka]. Dostupno: https://www.youtube.com/watch?v=Y7M-B6xnaEM [pristupano 14.2.2020.]

[14] Randy Van Nostrand, „How do I flip normals on this model?“, (29.4.2015.). u *Blender Stack Exchange* [Forum]. Dostupno: https://blender.stackexchange.com/questions/29132/how-do-i-flip-normals-on-this-model [pristupano 14.2.2020.]

[15] UnityBeginner, (10.7.2018.) „Unity How to : Change Skybox“, *YouTube* [video datoteka]. Dostupno: https://www.youtube.com/watch?v=KPgHGqYv17Q [pristupano 16.2.2020.]

[16] Mz3, „Move gameobject pivot“, (12.6.2015.). u *Unity Answers* [Forum]. Dostupno: https://answers.unity.com/questions/984944/move-gameobject-pivot.html [pristupano 16.2.2020.]

[17] brojevna kružnica, (bez dat.) u *Hrvatska enciklopedija*. Dostupno: https://www.enciklopedija.hr/natuknica.aspx?id=70211 [pristupano 16.2.2020.]

[18] Ash, „Return multiple values to a method caller“, (14.4.2009.). u *Stack Overflow* [Forum]. Dostupno: https://stackoverflow.com/questions/748062/return-multiple-values-to-a-method-caller [pristupano 16.2.2020.]

[19] Micorsoft, „Math.Cos(Double) Method“, (bez dat.). u *Microsoft Docs* [Na internetu]. Dostupno: https://docs.microsoft.com/en-us/dotnet/api/system.math.cos?view=netcore-3.1 [pristupano 16.2.2020.]

[20] O'Reilly, „Converting Degrees to Radians“, (bez dat.). u *C# Cookbook* [Na internetu]. Dostupno: https://www.oreilly.com/library/view/c-cookbook/0596003390/ch01s02.html [pristupano 16.2.2020.]

[21] Warman Steve, „Adjust physics to speed up/slow down?“, (3.1.2014.). u *Stack Overflow* [Forum]. Dostupno: https://stackoverflow.com/questions/20908312/adjust-physics-to-speed-up-slow-down [pristupano 17.2.2020.]

[22] Huy.V, „How to tell if a GameObject is moving or not?“, (30.4.2017.). u *Stack Overflow* [Forum]. Dostupno: https://stackoverflow.com/questions/43705000/how-to-tell-if-a-gameobject-is-moving-or-not [pristupano 17.2.2020.]

[23] greens356, „Hide/Unhide Game Object“, (15.4.2016.). u *Unity Answers* [Forum]. Dostupno: https://answers.unity.com/questions/1171111/hideunhide-game-object.html [pristupano 17.2.2020.]

[24] JNSVS, „Rigidbody - how to stop it quickly“, (13.3.2014.). u *Unity Answers* [Forum]. Dostupno: https://answers.unity.com/questions/662811/rigidbody-how-to-stop-it-quickly.html [pristupano 18.2.2020.]

[25] Bokaii, „Freeze rigidbody position in script“, (14.7.2014.). u *Unity Answers* [Forum]. Dostupno: https://answers.unity.com/questions/747872/freeze-rigidbody-position-in-script.html [pristupano 18.2.2020.]

[26] Blackthornprod, (1.5.2018.), „HOW TO MAKE COOL SCENE TRANSITIONS IN UNITY - EASY TUTORIAL“, *YouTube* [video datoteka]. Dostupno: https://www.youtube.com/watch?v=Qd2em_ts5vs [pristupano 19.2.2020.]

[27] Kapil11, „what is the difference between Update & FixedUpdate in Unity?“, (24.12.2015.). u *Stack Overflow* [Forum]. Dostupno: https://stackoverflow.com/questions/34447682/what-is-the-difference-between-update-fixedupdate-in-unity [pristupano 20.2.2020.]

[28] Ophelia, „Moving an object with vector3.MoveTowards“, (8.11.2016.). u *Stack Overflow* [Forum]. Dostupno: https://stackoverflow.com/questions/40489102/moving-an-object-with-vector3-movetowards [pristupano 20.2.2020.]

[29] Benny-1, „Fading out using render.material.color.a +=/-= doesn't work“, (24.4.2011.). u *Unity Answers* [Forum]. Dostupno: https://answers.unity.com/questions/60453/fading-out-using-rendermaterialcolora-doesnt-work.html [pristupano 20.2.2020.]

[30] Dima Kozyr, „Unity game manager. Script works only one time“, (9.3.2016.). u *Stack Overflow* [Forum]. Dostupno: https://stackoverflow.com/questions/35890932/unity-game-manager-script-works-only-one-time [pristupano 20.2.2020.]

[31] jeffery_unity, „How do you generate a line between objects that are only instantiated upon game Start ?“, (5.7.2019.). u *Unity Answers* [Forum]. Dostupno: https://answers.unity.com/questions/1645888/how-do-you-generate-a-line-between-objects-that-ar.html [pristupano 22.2.2020.]

[32] Matthew0123, „How delete or remove a component of an GameObject?“, (12.1.2013.). u *Unity Answers* [Forum]. Dostupno: https://answers.unity.com/questions/378930/how-delete-or-remove-a-component-of-an-gameobject.html [pristupano 22.2.2020.]

[33] Shrimpey, „Outlined Diffuse Shader Fixed for Unity 5.6“, (16.8.2018.) u *GitHub* [Komprimirana datoteka]. Dostupno: https://github.com/Shrimpey/Outlined-Diffuse-Shader-Fixed [pristupano 22.2.2020.]

[34] HolBol, „enable/disable specific components“, (7.9.2010.). u *Unity Answers* [Forum]. Dostupno: https://answers.unity.com/questions/26844/enabledisable-specific-components.html [pristupano 22.2.2020.]

[35] Ryan Zehm, (1.11.2017.), „Unity Scripting Tutorial: Physics.SphereCast“, *YouTube* [video datoteka]. Dostupno: https://www.youtube.com/watch?v=Nplcqwq_oJU [pristupano 22.2.2020.]

[36] Revolave, „Unity SphereCast - Getting the center of the last sphere before the hit“, (16.12.2019.). u *Unity Answers* [Forum]. Dostupno: https://answers.unity.com/questions/1684418/unity-spherecast-getting-the-center-of-the-last-sp.html [pristupano 22.2.2020.]

[37] n Beats, (12.7.2014.), „BILLIARDS SOUND EFFECT IN HIGH QUALITY“, *YouTube* [video datoteka]. Dostupno: https://www.youtube.com/watch?v=QxibcFuZQJk [pristupano 23.2.2020.]

[38] John, „How to play audio in Unity (with examples)“, 17.9.2019. u *Game Dev Beginner* [Na internetu]. Dostupno: https://gamedevbeginner.com/how-to-play-audio-in-unity-with-examples/ [pristupano 23.2.2020.]

[39] absameen, „Rotate Around Local Y Axis“, (23.7.2012.) u *Unity Answers* [Forum]. Dostupno: https://answers.unity.com/questions/288948/rotate-around-local-y-axis.html [pristupano 23.2.2020.]

[40] John, „The right way to pause a game in Unity“, 13.2.2020. u *Game Dev Beginner* [Na internetu]. Dostupno: https://gamedevbeginner.com/the-right-way-to-pause-the-game-in-unity/ [pristupano 23.2.2020.]

[41] Brackeys, (2.12.2018.), „SAVE & LOAD SYSTEM in Unity“, *YouTube* [video datoteka]. Dostupno: https://www.youtube.com/watch?v=XOjd_qU2Ido [pristupano 23.2.2020.]

[42] Unity Technologies, „QuickStart to TextMesh Pro“, 2020. u *Unity Learn* [Na internetu]. Dostupno: https://learn.unity.com/tutorial/working-with-textmesh-pro [pristupano 24.2.2020.]

[43] „Pill button green“, (27.2.2018.). [Slika]. Dostupno: https://www.iconspng.com/image/115744/pill-button-green [pristupano 24.2.2020.]

[44] BrinkHouseGames, „Inspector Assigned Variables Not In Both Scenes“, (14.10.2013.). u *Unity Answers* [Forum]. Dostupno: https://answers.unity.com/questions/555910/inspector-assigned-variables-not-in-both-scenes.html [pristupano 24.2.2020.]

[45] Rewpparo, „Button.OnClic.AddListener(delegate{method(object);}), wrong object sent“, (28.2.2015.). u *Unity Answers* [Forum]. Dostupno: https://answers.unity.com/questions/912202/buttononclicaddlistenermethodobject-wrong-object-s.html [pristupano 24.2.2020.]

[46] Unity Technologies, „Canvas Group | Unity UI | 1.0.0“, (bez dat.). [Na internetu]. Dostupno: https://docs.unity3d.com/Packages/com.unity.ugui@1.0/manual/class-CanvasGroup.html [pristupano 26.2.2020.]

[47] GutoThomas, „Slowly fades from opaque to alpha“, (9.3.2012.). u *Unity Answers* [Forum]. Dostupno: https://answers.unity.com/questions/225438/slowly-fades-from-opaque-to-alpha.html [pristupano 26.2.2020.]

[48] „Snooker pool ball vectors“, (bez dat.). [Slika]. Dostupno: https://www.clipartlogo.com/image/snooker-pool-ball-vectors_336083.html [pristupano 26.2.2020.]

[49] Unity Technologies, „Selectable.interactable“, 2019. [Na internetu]. Dostupno: https://docs.unity3d.com/2019.1/Documentation/ScriptReference/UI.Selectable-interactable.html [pristupano 26.2.2020.]

[50] Pure Phase, „Is there a global volume setting?“, (4.5.2010.). u *Unity Answers* [Forum]. Dostupno: https://answers.unity.com/questions/16603/is-there-a-global-volume-setting.html [pristupano 26.2.2020.]

[51] „Triangle“, (bez dat.). [Slika]. Dostupno: http://pngimg.com/download/87923 [pristupano 26.2.2020.]

[52] Unity Technologies, „Image.sprite“, 2018. [Na internetu]. Dostupno: https://docs.unity3d.com/2018.1/Documentation/ScriptReference/UI.Image-sprite.html [pristupano 26.2.2020.]

[53] Unity Technologies, „Unity Playground“, 2019. [Na internetu]. Dostupno: https://learn.unity.com/project/unity-playground [pristupano 27.2.2020.]

[54] Unity Technologies, „Introduction to Particle Systems Tutorial“, 2019. [Na internetu]. Dostupno: https://learn.unity.com/tutorial/introduction-to-particle-systems [pristupano 27.2.2020.]

[55] Unity Technologies, *Unity* (verzija 2019.3.0a6) (2019) [Program]. Dostupno: https://unity.com [pristupano 30.9.2019.]

[56] Microsoft, *Visual Studio Community 2019* (verzija 16.1) [Program]. Dostupno: https://visualstudio.microsoft.com [pristupano 30.9.2019.]

[57] The Blender Foundation, *Blender* (verzija 2.81a) [Program]. Dostupno: https://www.blender.org [pristupano 13.2.2020]

[58] Microsoft, *Word* (verzija 2006) [Program]. Dostupno: https://www.office.com/ [pristupano 2.7.2020.]

[59] The GIMP Team, *GIMP* (verzija 2.10) [Program]. Dostupno: https://www.gimp.org/ [pristupano 15.9.2019.]

[60] Audacity Team, *Audacity* (verzija 2.3.0) [Program]. Dostupno: https://www.audacityteam.org/ [pristupano 23.10.2019.]

Popis slika