BiljAR

Tehnička dokumentacija

Verzija 1.1

Studentski tim: Bruno Banek

Simon Grgurina

Anteo Ivankov

David E. Lukšić

Filip Matijević

Ivan Sičić

Josip Sito

Nastavnik: prof. dr. sc. Željka Mihajlović

Sadržaj

1. Opis razvijenog proizvoda 3

2. Korišteni alati 4

3. Tehničke značajke 5

3.1 Pravila igre 5

3.2 Fizika 6

3.3 Modeli 7

3.4 Grafičko korisničko sučelje (GUI) 10

3.5 Računalni vid i proširena stvarnost 12

3.6 Glavni izbornik 16

3.7 Zvučni efekti 18

4. Upute za korištenje 20

5. Budući razvoj 21

6. Literatura 22

Tehnička dokumentacija

# Opis razvijenog proizvoda

U sklopu projekta razvili smo zabavnu i edukativnu računalnu igru koja omogućuje igranje biljara na svakom mjestu i u svako vrijeme. Namijenjena je mobilnim uređajima s operacijskim sustavom *Android* i koristi proširenu stvarnost kako bi pružila iskustvo što vjernije igranju pravoga biljara. Upravo zbog toga smo je nazvali *BiljAR* (*AR – augmented reality*, proširena stvarnost).

Zahvaljujući razvoju sklopovlja mobilnih uređaja te metoda raspoznavanja uzoraka, proširena stvarnost postaje vrlo zanimljivo područje za razvoj mobilnih igara i aplikacija. Prožimanjem stvarnog svijeta virtualnim elementima, stvara se dojam vjerno nadograđene stvarnosti koja se kretanjem u stvarnom prostoru može dodatno istražiti, gledati iz više pozicija i time pospješiti interakciju s virtualnim svijetom. Mobilna igra upravo zbog visoke razine interakcije korisnika i virtualne scene te pokretljivosti korisnika i uređaja daje izvrstan uvid u mogućnosti tehnologije proširene stvarnosti.

Kako bi program znao gdje iscrtati virtualne elemente u stvarnom okružju snimanom kamerom, potreban je uzorak stvarnog svijeta kojeg može prepoznati. Takav uzorak naziva se marker i pomoću njega igra može odrediti položaj scene u odnosu na stvarni svijet i položaj uređaja u odnosu na scenu. Pokretanjem programa i odabirom načina igranja, kamera snima okruženje stvarnoga svijeta, a pri ulasku markera u vidljivo polje kamere igra prepoznaje marker i iscrtava scenu biljara na poziciji markera dok je stvarni svijet sniman kamerom i dalje prikazan u pozadini. Korisnik kretanjem oko virtualne scene može pobliže ispitati stanje elemenata scene kako bi donio odluku o daljnjim potezima u igri.

Kada korisnik hoda štap je iscrtan uspravno kako bi bilo naznačeno stanje kretanja, a na pritisak gumba ciljanja štap se postavlja u položaj za ciljanje. Dok korisnik drži taj gumb pritisnutim, štap ga prati u ciljanju te kada ga pusti štap ostaje u istom položaju dok se korisnik značajno ne pomakne. U slučaju prevelikog odmaka, štap se vraća u položaj hodanja. Korisnik u stanju ciljanja može povlačenjem prsta po ekranu dodatno pomicati položaj štapa radi veće kontrole i preciznosti te dodatno može odabrati u koji dio bijele kugle želi udariti kako bi napravio spin na kugli. Konačno, korisnik dodiruje gumb za pucanje te povlači mobilni uređaj unazad kako bi odredio jačinu udarca. Kada je zadovoljan jačinom pušta gumb pucanja. Tada će štap udariti u kuglu i odaslati je u željenom smjeru.

Igra podržava igranje u dva igrača preko lokalne mreže, a ulaskom u igru kao posmatrač moguće je promatrati igru dvojice aktivnih igrača. Igra može odrediti položaje igrača te će na jednom uređaju iscrtati štap na poziciji uređaja drugog igrača.

# Korišteni alati

Kako je cilj projekta bio izraditi igru s proširenom stvarnošću, da bi smo što lakše implementirali komponentu proširene stvarnosti i što prije došli do gotovog proizvoda, koristili smo višeplatformski pogon igara (*game engine*) i razvojnu okolinu *Unity* koji se može koristiti za razvoj za osobna računala, igraće konzole, mobilne uređaje i web stranice. Unity nam je omogućio jednostavno korištenje grafičkih elemenata i raznih funkcionalnosti poput osvjetljenja i iscrtavanja scene, pomicanja očišta, učitavanja resursa i programskih dodataka, reprodukcije zvukova i izvođenja igre bez brige o implementacijskim detaljima istih.

Nijedan član tima nije imao većeg iskustva s tehnologijama proširene stvarnosti, ali smo došli do zaključka da je alat za proširenu stvarnost *Vuforia* najbolje rješenje iako nije besplatna za projekte koji izađu iz stanja razvoja, što predstavlja prepreku ukoliko u budućnosti poželimo objaviti igru.

Modeli su -izrađeni u *Blenderu,* skupu alata za izradu grafičkih komponenti, modeliranje i izradu tekstura. Otvorenog je koda i besplatan, kao i *Gimp*, alat za uređivanje slika u kojem je izrađeno prilagođavanje tekstura UV koordinatama modela.

Za praćenje razvoja i verzija programa korišten je besplatni upravitelj repozitorijima *Gitlab* koji omogućava istovremeni rad više članova na proizvodu uz kontrolu toka.

Za razvoj programa korišteni su razvojna okolina *Miscrosoft Visual Studio* i uređivač izvornog koda *Visual Studio Code* koji se mogu spojiti s upraviteljem repozitorija i tako direktno upravljati udaljenim repozitorijem.

Kako bismo mogli izvesti igru koja se može pokrenuti na uređajima s operacijskim sustavom *Android*, koristili smo skup razvojnih alata *Android SDK* za generiranje izvršnog koda.

# Tehničke značajke

## Pravila igre

Stanja igre implementirana su u obliku automata stanja. Početno stanje je “*SetupState*”, ono se koristi na početku igre za provjeravanje da su se dva igrača pridružili igri. Nakon toga ulazi se u stanje za razbijanje "*BreakState*", nakon kojega se koriste stanje za normalne udarce “*NormalShootingState*“ i stanje za ubacivanje crne lopte „*BlackBallShootingState*“. Ova tri stanja nasljeđuju “*ShootingState*“.

internal class ShootingState : EightBallGameState

{

public ShootingState(EightBallGame game, bool ballinhand)

public override void ExecuteShot(Vector3 dir, Vector2 spin, float power)

public override void BallHitRail(BallRailCollisionPair cp)

public override void BallPotted(BallPocketCollisionPair cp)

public override void BallHitBall(BallBallCollisionPair cp)

public override void BallsStopped()

public override void PlaceBall(int ballID, Vector3 pos)

protected EightBallGameState GetNextState(bool ballInHand)

protected bool PlayersBallPotted()

protected void SendMessage(MessageType messageType)

}

Slika 1. Prikaz stanja ShootingState i metoda koje implementira

"*ShootingState*" obavlja funkcije koje su zajedničke svim stanjima koja uključuju udarce poput upravljanja sučeljem i praćenja događaja. Stanja koja uključuju udarce podijeljena su u faze: igrač najprije puca, tada se ugase kontrole, i čeka se zaustavljanje loptica. Tijekom kretanja loptica stanja prate događaje poput ubačenih loptica i sudara između različitih loptica i sudara sa zidovima. Kako se pojedini događaj odvije, igra poziva događaj (engl. Event), pri čemu se u trenutnom stanju poziva odgovarajuća metoda koja na njega odgovara. Na primjer, pri ubacivanju loptice, poziva se metoda *BallPotted(BallPocketCollisionPair cp)* koja parametrom prima podatke o tome koja loptica je ubačena u koju rupu. Stanje će tada ovisno o ubačenoj loptici postaviti bool varijablu npr. “*BlackPotted*” koje označava da je u odvijanju stanja ubačena crna lopta. Kada se sve loptice ubace u rupe, poziva se metoda za “*BallsStopped*“ koja tada analizira događaje koji su se dogodili te odlučuje koje je sljedeće stanje i koji igrač je na redu. Preostaju stanja “*EndState*”, koje se koristi kada je završila igra, te “*CallShotState*” pomoću kojeg igrač izabire rupu u koju će ubaciti crnu loptu.

## Fizika

Potreba za vlastitom simulacijom fizike uočila se već u prvim koracima izrade simulacije biljarskih kugli. Naime, Unityjevo standardno rješenje za fiziku ima nekoliko nedostataka koji su nedopustivi u igri kao što je biljar. Neki od njih su sljedeći: trenje ne funkcionira dobro u malim mjerilima, odbijanje kugle od zida nije realistično, tijela usporavaju na nerealističan način, ali najgore od svega jest simulacija u fiksnim vremenskim koracima, zbog čega smo odlučili definitivno napisati vlastiti simulator.

Da bismo razumjeli zašto je koristiti simulaciju u fiksnim vremenskim koracima tako velika pogreška pri izradi simulacije biljara, zamislimo sljedeću situaciju. Lopta „1“ kreće se nekom brzinom v. Nailazi na loptu „2“ koja stoji nepomično. Ukoliko simuliramo ovu situaciju u fiksnim koracima, tj. u svakom koraku na poziciju lopte „1“ dodajemo njenu komponentu brzine (skaliranu u vremenu), dobit ćemo situaciju prikazanu na slici 1.



Slika 2. Jednoliko pravocrtno kretanje lopte prema sudaru

Zašto je ovo nedopustivo uočavamo prilikom kolizije. Događa se to da lopta „preskoči“ koliziju, a zatim je detektira. Rezultat ovakve kolizije nikako ne može biti točan te se kugle odbiju pod krivim kutom. Ovo se zapravo može izbjeći u Unityjevoj standardnoj simulaciji tako što se omogući „*Continuous collision detection*“, no nikako nismo mogli dobiti precizne odbitke kugli nakon kolizije unatoč tome.

Baš iz ovog razloga, naša simulacija fizike ima varijabilne korake i predviđanje kolizije koja uspješno rješava navedeni problem.

Ovdje nećemo navoditi sve matematičke metode predviđanja sudara koje su korištene, no najzanimljiviji slučaj nam je sudar dviju kugli, koji ima lijepo matematičko rješenje ukoliko zanemarimo deceleracija kugli uslijed trenja. Pogledajmo najprije rješenje predikcije kada ne zanemarimo trenje. U sljedećim jednadžbama vektori a,v i p su redom razlike akceleracije, brzine i pozicije dviju kugli, a R je suma radijusa.

Uočavamo da je ovo vrlo kompaktna jednadžba, koja za zadanu akceleraciju, brzinu i poziciju kugli može matematičkom točnošću odrediti vrijeme sudara tih kugli. No, uočavamo da je rješenje zapravo korijen polinoma četvrtog stupnja, kojem je eksplicitno rješenje previše kompleksno za implementaciju. Drugi način evaluacije ovakvog polinoma jest neki od poznatih algoritama traženja korijena.

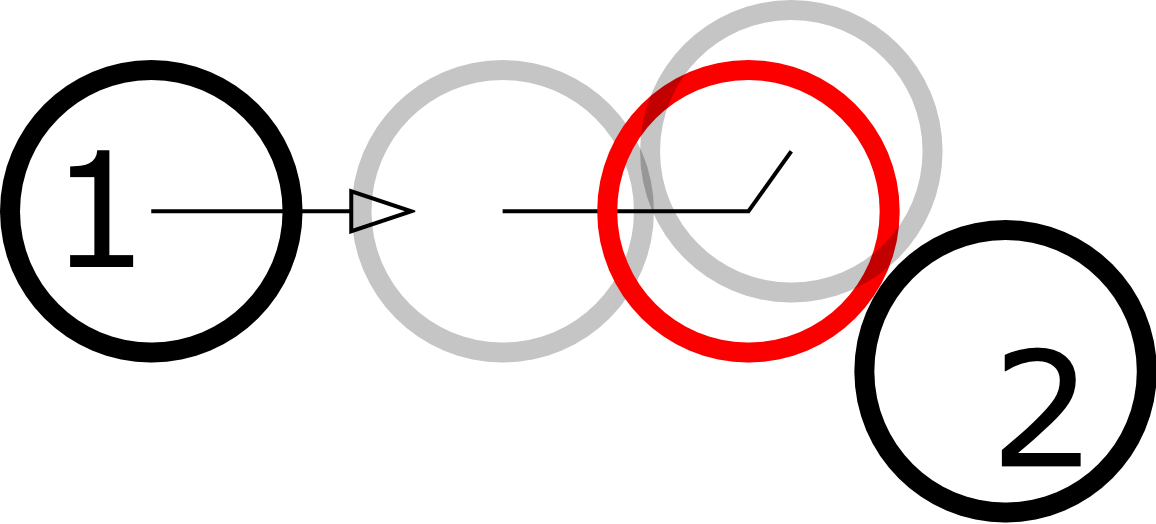
Ispostavilo se da zanemarivanjem trenja nećemo izgubiti puno na točnosti. To je zato što je vrijeme koraka vrlo malo (1/60 sekunde), a akceleracije uslijed trenja su također male po iznosu. Pojednostavljena jednadžba je sljedeća:

Na sličan način implementirani su sudari kugle s rubom (eng. Rail) stola. Svaki od njih zanemaruje akceleraciju kugle i određuje koliko vremena je potrebno kugli da se sudari krećući se trenutnom brzinom. Jednom kada smo to sve implementirali, možemo objasniti kako u konačnici izgleda sam algoritam simulacije:



Slika 3. Pseudokod simulacije

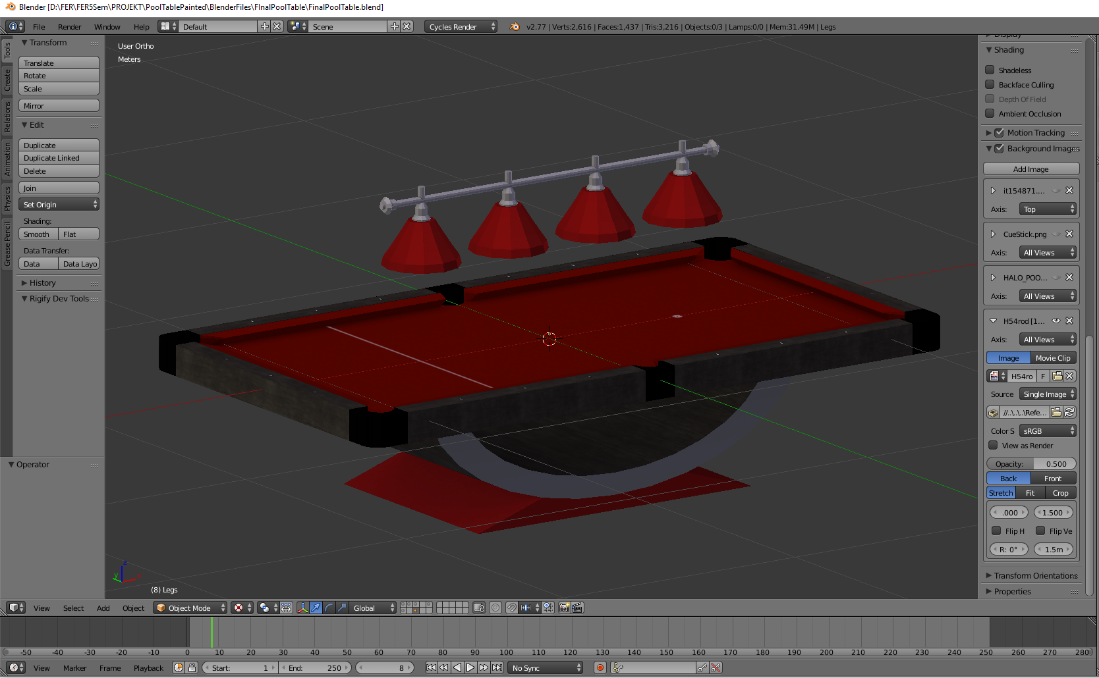
Riječima, to možemo opisati ovako: ukoliko nema kolizija, naša simulacija će raditi fiksne korake. U slučaju da unutar tog koraka predvidimo koliziju, taj korak se podijeli na manje dijelove, tako da se simulira prvo vrijeme do kolizije, zatim se izračuna sudar (eng. collision response), zatim se to ponovi za ostatak vremena tog koraka. Ako više nema kolizija, simuliramo korak do kraja i prelazimo na sljedeći korak.



Slika 4. Naš korak simulacije

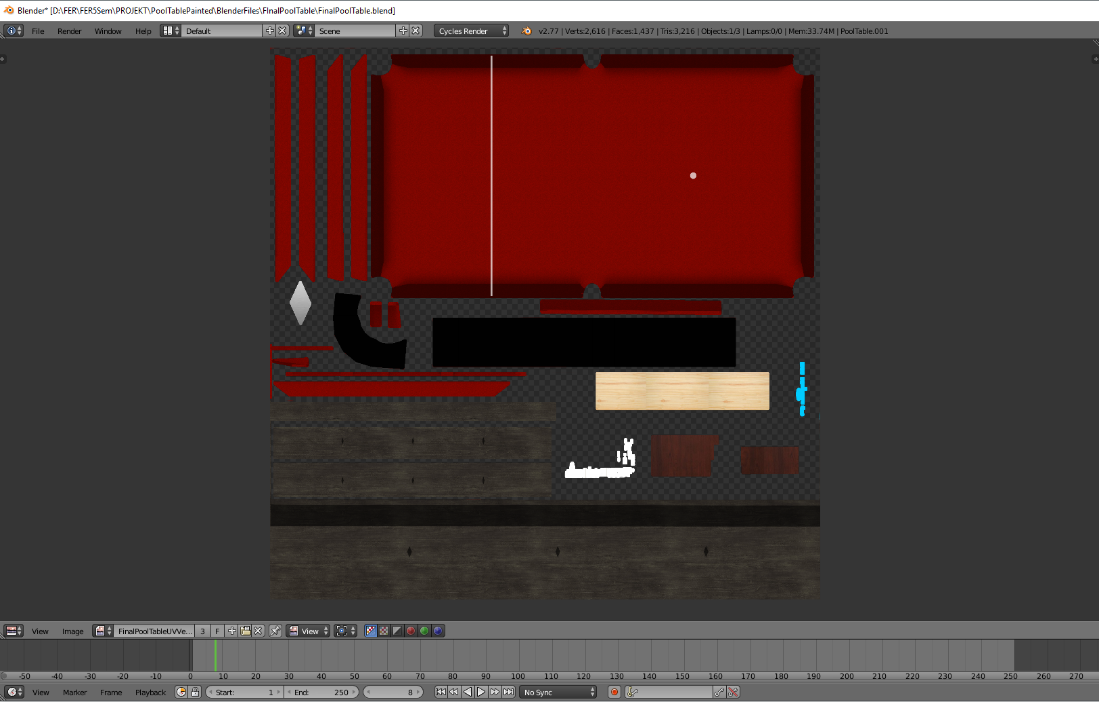
## Modeli

Potaknuti željom za stvaranjem veoma realistične igre biljara, nakon stvaranja realističnih kolizija loptica, te izgradnjom stanja (pravila) igre po WPA "8-ball-pool" standardima, i modeli su također morali biti realni i izgrađeni po WPA standardima veličina. Zbog prethodno navedenih motiva i razloga, izrađeni model biljarskog stola veličine je 2.54 x 1.27 m. Udaljenosti između vrhova rupa su 11.75 cm za rupe u kutovima biljarskog stola, odnosno 13 cm za bočne rupe stola. Visina rubova biljarskog stola su 4.75 cm za rubove koji sudjeluju u koliziji s lopticama, odnosno 5.4 cm za rubove koji ih okružuju. Model štapa također je izgrađen po WPA pravilima.



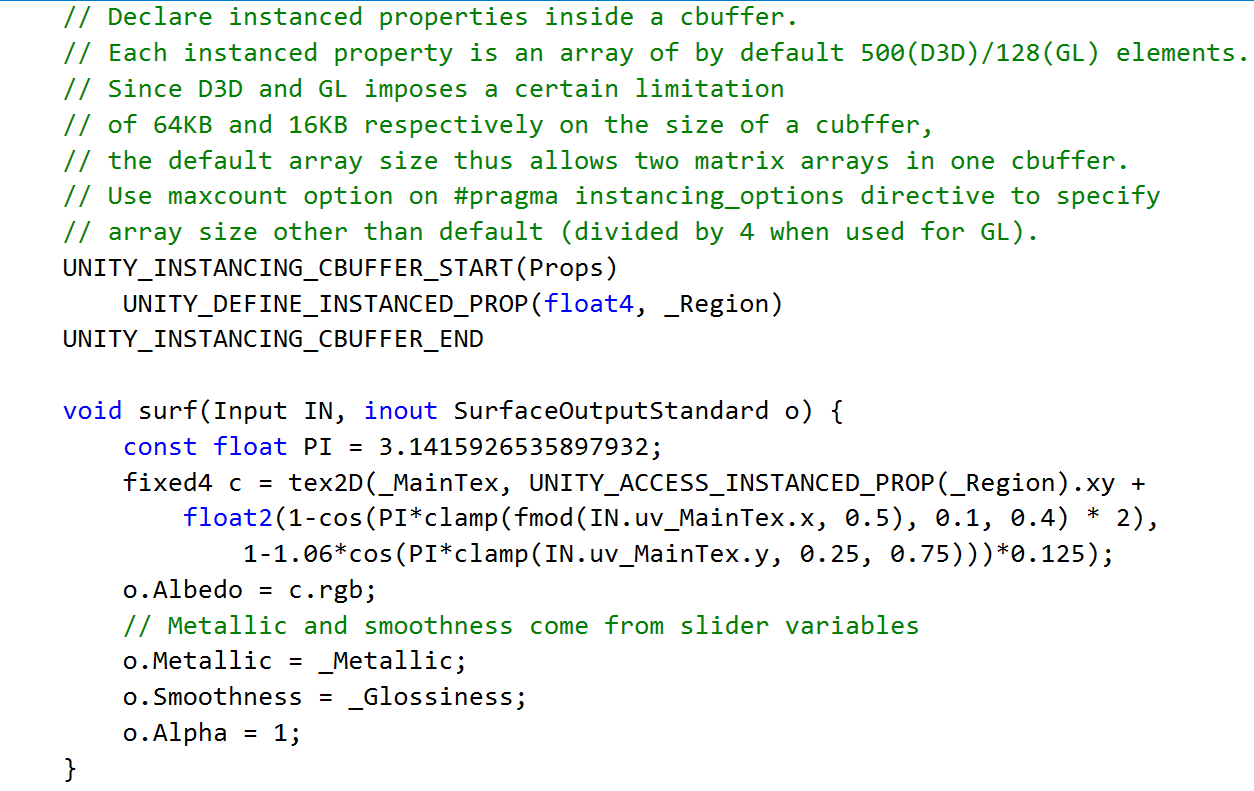
Slika 5. Prikaz izrade modela stola u Blenderu

Svi modeli izrađeni su unutar modelno-razvojnog programa Blender. Pri izgradnji modela, uz restrikcije kojih smo se morali držati radi poštivanja WPA standarda, morali smo također imati na umu da izrađujemo modele za mobilnu aplikaciju. Upravo zbog toga, broj vrhova (engl. Vertex) modela treba biti što manji mogući, te sve teksture i UV-koordinate modela trebaju biti na jednoj slici, kako bismo optimirali kasnije izvođenje unutar Unity-a. Prilagođavanje tekstura UV koordinatama modela izrađeno je uz pomoć alata za obradu slika Gimp.



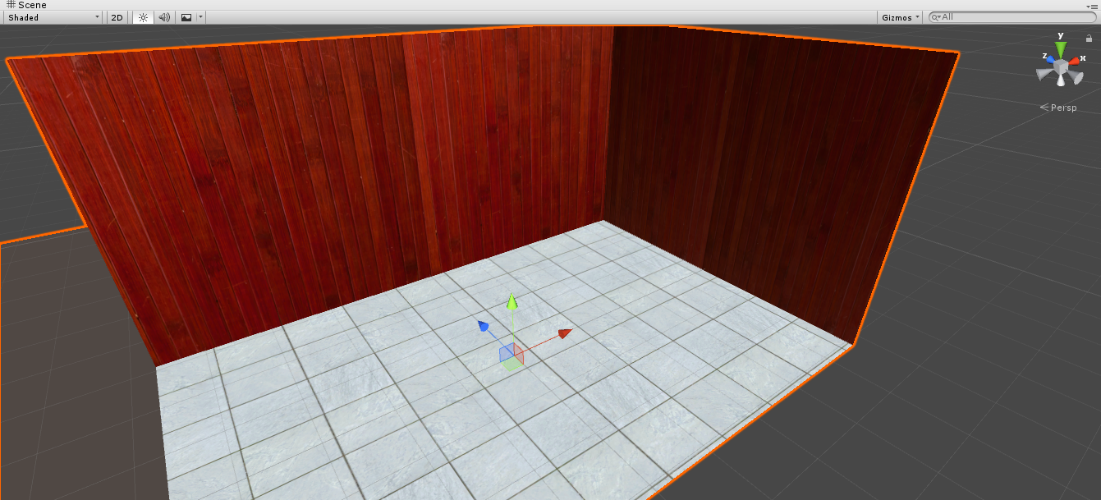
Slika 6. Prikaz raspoređivanja UV-koordinata radi poboljšavanja izvođenja

Sljedeći problem su predstavljale loptice, koje zbog samog njihovog broja ( 15 loptica za jednu partiju biljara), nikako ne smiju sadržavati svaka svoj materijal i teksturu. Pošto smo odlučili kako je Unity-eva verzija sfere ( loptice) dovoljno dobra, tj. zbog nedostatka vremena za izradu vlastitih kvalitetnih loptica, našli smo se suočeni sa problemom pridjeljivanja jednog materijala svim lopticama. Rješenje smo pronašli u pisanju vlastitog programa za sjenčanje (sjenčara, engl- shader) za loptice. Dio sjenčara prikazan na slici 7 pridjeljuje određeni dio teksture loptica pojedinoj loptici, te se brine za ispravno određivanje normala loptice i koordinata teksture trenutno vidljivih na loptici u svakom trenu izvođenja. Također, radi estetskih razloga, omogućili smo podešavanje utjecaja "metallic" odsjaja na lopticama.

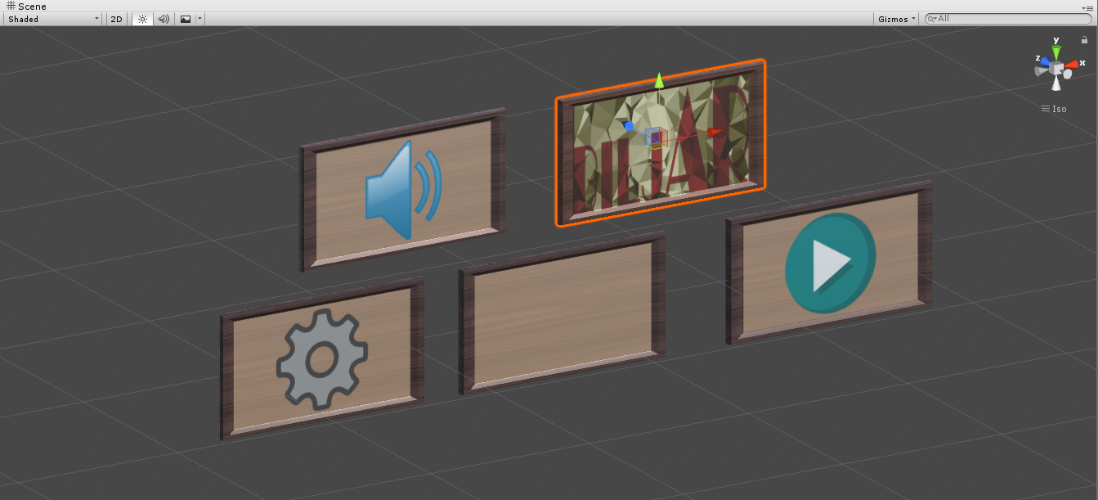


Slika 7. Isječak koda sjenčara za loptice

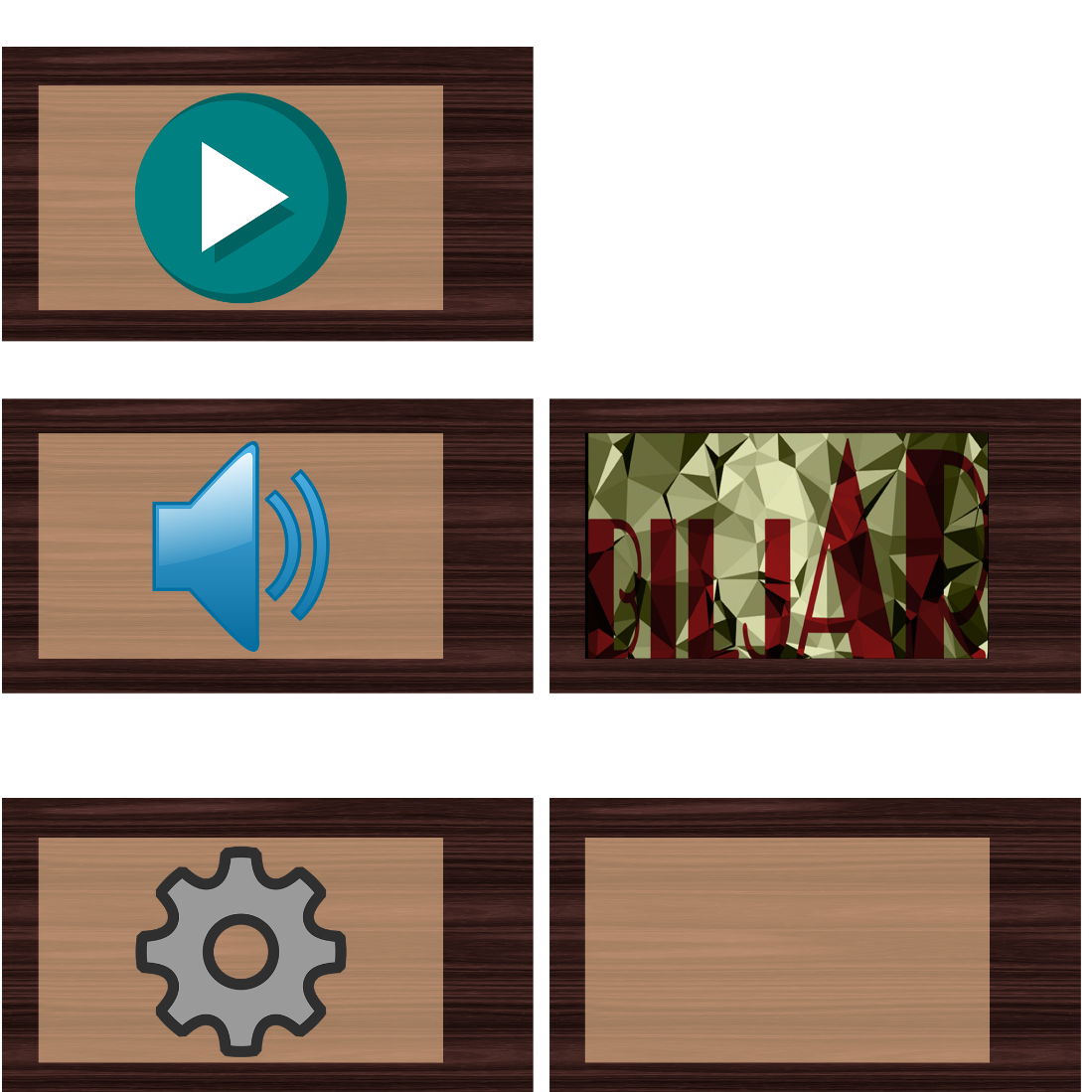
Za potrebe glavnog izbornika napravili smo jedan model prostorije (koji sadrži teksture zida i poda) i slike odnosno okvira slike od kojih svaki predstavlja jednu komponentu GUI sustava (Slike 8 i 9). Gumbi play, mute, postavke, panel za postavke i krediti sadrži isti model okvira slike, ali svi oni koriste različite teksture koje su radi optimizacije postavljeni u jednu mapu. Svakoj UV koordinati pojedinog modela pridružuje se specifična tekstura sadržana u mapi(Slika 10).



Slika 8. Model prostorije s teksturama zida i poda



Slika 9. Modeli okvira slike s njihovim specifičnim teksturama



Slika 10. Mapa sa teksturama pojedinih slika

## Grafičko korisničko sučelje (GUI)

Korisničko sučelje predstavlja glavni element interakcije s korisnikom. Budući da korisničko sučelje mora biti intuitivno i jednostavno za korištenje odlučili smo ubaciti minimalan broj komponenti sučelja potrebnih za igru.

Korisničko sučelje je u potpunosti izgrađeno koristeći Unity-eve alate namijenjene za to. Svaka komponenta grafičkog korisničkog sučelja mora biti dijete posebnog objekta platna (engl. canvas). Glavni elementi korisničkog sučelja za vrijeme igre su gumbi (engl. buttons) koji su prikazani na slici 11. Također, svaki gumb kada se pritisne promijeni svoju veličinu kako bi se naznačilo da je korisnik u interakciji s tim gumbom.



Slika 11. Prikaz glavnih elemenata grafičkog korisničkog sučelja za vrijeme igre

Svakom gumbu pridijeljena je posebna skripta koja određuje što se događa kada se određeni gumb pritisne, drži pritisnutim ili otpusti. Budući da svi gumbi imaju neke zajedničke atribute poput slike, veličine polumjera i stanja u kojem se nalaze, odlučili smo napraviti općeniti apstraktni razred za sve gumbe koju onda svaki pojedinačan gumb nasljeđuje. Na slici 12 je prikazan općeniti razred *Button* te neki njegovi atributi i metode.

public abstract class Button : MonoBehaviour

{

public ButtonState CurrentState { get; set; }

public Transform RadiusDefiner;

public Image Image;

public int Order;

internal float Radius;

internal float LerpCoef;

internal Vector3 StartPosition;

private void Start()...

protected abstract void OnStart();

protected virtual void Update()...

public abstract void Enable();

public abstract void Disable();

public virtual bool IsInside(Vector3 pos)...

}

Slika 12. Kod apstraktne klase Button

Jedan od važnijih atributa u razredu *Button* je trenutno stanje jer u ovisnosti u kojem se stanju dugme nalazi ono ima različitu ulogu u tom trenutku. Tako smo za svako dugme definirali njegova moguća stanja specifična za njega. Zajedničko svim stanjima je da nasljeđuju apstraktan razred *ButtonState*. Glavne apstraktne metode koje sadrži razred *ButtonState* su: *Pressed*, *Released* i *Hold*. Te metode onda implementiraju konkretna stanja a služe za opisivanje što se događa kad je dugme pritisnuto ili otpušteno ili se drži pritisnutim u tom određenom stanju.

Iz razloga što svi gumbi u igri ne mogu biti aktivni u isto vrijeme, napravili smo poseban objekt koji je zadužen za upravljanje svim gumbima . Tom objektu pridodana je skripta *GUIManager* u kojoj je glavna metoda *Update* a uloga joj je odrediti koji gumb je trenutno aktivan i pozvati određenu metodu aktivnog gumba. Metoda *Update* implementirana je na način da prvo iterira po svim gumbima i traži prvog od njih koji je aktivan. Nakon toga nad aktivnim gumbom poziva određenu metodu u ovisnosti o tome je li gumb pritisnut, da li se drži pritisnutim ili je otpušten. Implementacija metode *Update* prikazana je na slici 13.

void Update()

{

var button = Buttons.FirstOrDefault(b => b.CurrentState.Active);

if (button == null)

button = Buttons.FirstOrDefault(b => b.IsInside(Input.mousePosition) &&   
 b.CurrentState.TakesInput);

if (button == null) return;

if (Input.GetMouseButtonDown(0))

{

button.CurrentState.Pressed(Input.mousePosition);

}

else if (Input.GetMouseButton(0))

{

button.CurrentState.Hold(Input.mousePosition);

}

else if (Input.GetMouseButtonUp(0))

{

button.CurrentState.Released(Input.mousePosition);

}

}

Slika 13. Implementacija metode Update u razredu GUIManager

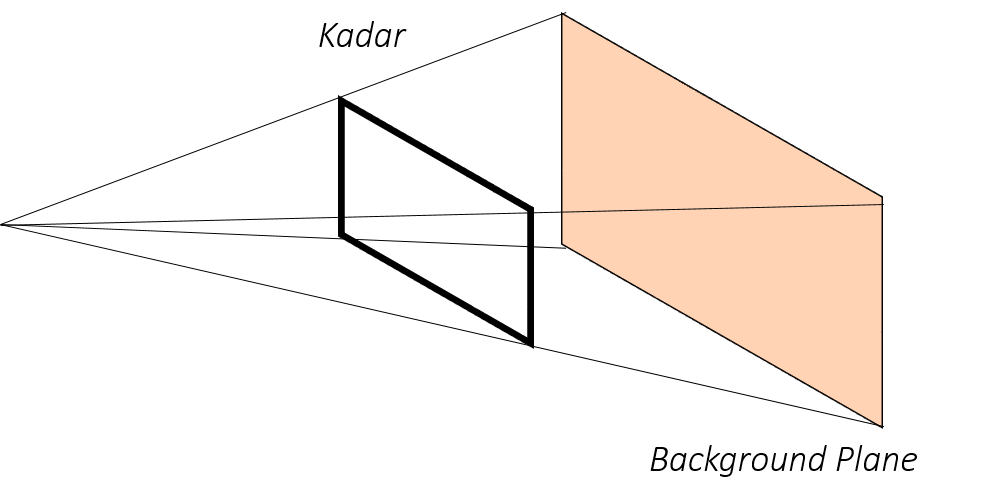
## Računalni vid i proširena stvarnost

Koristeći razvojni alat proširene stvarnosti „Vuforia“ omogućili smo korisniku da kroz svoj pametni telefon jasno vidi biljarski stol i protivničkog igrača. „Vuforia“ koristi tehnologiju računalnog vida za prepoznavanje i praćenje ravnih slika (Image Target) i jednostavnih trodimenzionalnih objekata. Primarni cilj ovog projekta je da igra bude prenosiva pa smo se zato odlučili na praćenje ravne slike. Alat nam omogućuje da stvarnome svijetu dodijelimo referentnu točku pomoću koje se možemo orijentirati i na temelju nje poznajemo koordinate svih ostalih objekata u sceni.



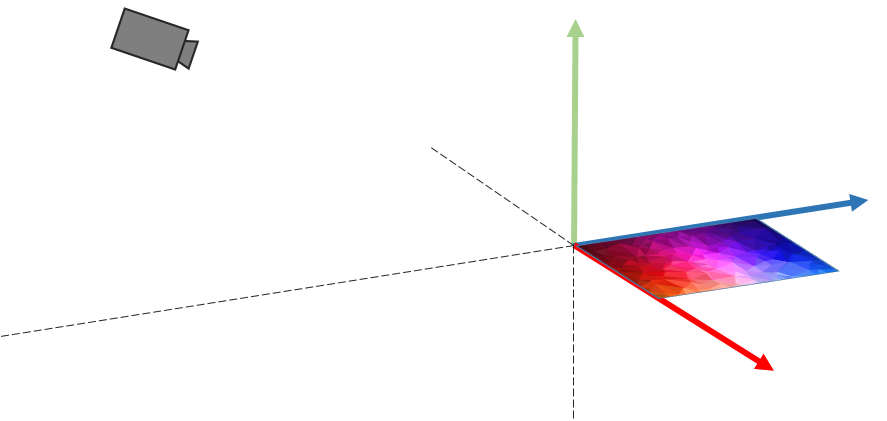
Slika 14. Slika koja se koristi kao marker

Na slici 14 prikazan je „Image Target 1“. Kada ta slika uđe u kadar kamere mobitela, na njoj se stvara biljarski stol, zajedno sa svim ostalim elementima igre. Bitno je naznačiti da pri dizajniranju markera, boje ne igraju veliku ulogu te će sustav raditi jednako ispiše li se isti marker u sivim tonovima. Također je važno da se na markeru nalazi što veći broj oštrih i jednostavnih geometrijskih oblika i nagli prijelazi u zasićenju boje. Primjetili smo da markeri koji sadrže tekst daju najbolje rezultate.



Slika 15. Standardna kamera dostupna u Unity3D

Na slici 15 je pojednostavljeni prikaz proširene stvarnosti. U razvojnom alatu „Vuforia“ priložen je objekt *ARCamera* koja se sastoji od standardne kamere dostupne u Unity3D i od pozadinske plohe. Ta pozadinska ploha je zapravo niz slika dobiven od kamere na mobilnom uređaju.



Slika 16. Referentna točka proširene stvarnosti

Slika 16 predstavlja pojednostavljen prikaz referentne točke proširene stvarnosti. Nakon što kamera registrira marker, ona može prepoznati svoje koordinate i rotaciju relativnu referentnoj točki koja je zapravo marker.

Računali vid koristili smo i pri prijenosu informacija s jednog uređaja na drugi. U želji da se u potpunosti izbjegne korištenje tipkovnice na zaslonu mobitela, dodali smo vlastito dizajnirani generator i skener koda. Kako je igra namijenjena da se igra za dva igrača, jedan igrač stvara server, dok se drugi pridružuje. Igrači moraju biti povezani na isti usmjeritelj kako bi mogli dijeliti igru.



Slika 17. Marker za dijeljenje jednog bajta

Na slici 17 prikazan je marker za dijeljenje jednog bajta informacije. Kako je raspon IP adresa u podmreži 0-255, upravo je jedan bajt dovoljan za prijenos te informacije. Korisnik koji je stvorio server može saznati i njegovu točnu IP adresu, dok igrač koji se želi pridružiti igri zna samo adresu podmreže. Kako bi klijent saznao točnu adresu servera, on će jednostavno skenirati zaslon vlasnika igre i dobiti pristup igri.

Slika se generira zacrnjivanjem šesterokuta (binaran broj).



Slika 18. Čitanje markera za dijeljenje bajta

Na slici 18 prikazano je čitanje bajta sa generirane slike. Algoritam je prilagođen da prepoznaje crnu i bijelu boju. Zahvaljujući „Vuforijinom“ sustavu praćenja slike, uvijek su nam poznate točne koordinate gdje se nalazi pojedini šesterokut. Da bi se izbjegla loša očitavanja sustav prvo za svako polje pogleda koje je boje. Postavljena je stroga granica točno na sredini sivoga spektra.

Color test = tekstura\_kamere.GetPixel((int)poz.x, (int)poz.y);

if (test.r < 0.5f && test.g < 0.5f && test.b < 0.5f)

return Math.Max(test.r, Math.Max(test.g, test.b));

else

return Math.Min(test.r, Math.Min(test.g, test.b));

Ova metoda poziva se za svako polje čija se povratna vrijednost sprema u niz. U sljedećem koraku stvaraju se dozvoljena i zabranjena područja. Niz se sortira po veličini i ispituju se udaljenosti susjednih članova.

void color\_gap()

{

int i;

float max = 0.2f;

low = 0.2f; high = 0.8f;

for (i = 0; i < 7; i++)

{

if (Mathf.Abs(boje[i] - boje[i + 1]) > max)

{

max = Mathf.Abs(boje[i] - boje[i + 1]);

high = boje[i];

low = boje[i + 1];

}

}

}

Varijabla max označava širinu sivog područja. U njemu vrijednost boje ne smije poprimiti niti „0“, niti „1“, dok varijable high i low označavaju granice visoke i niske impedancije. Iz ovoga zaključujemo da nijansa sive koja pada u interval [0, low] simbolizira logičku „0“, dok nijansa koja je iz intervala [high, 1] simbolizira logičku „1“. Konačni korak je te granice primijeniti na svih osam polja i iščitati binaran zapis bajta prikazanog na slici. Ovaj postupak se ponavlja u svakom okviru (engl. frame) kako bi se osigurala točnost.

broj = 0;  
        for (i = 0; i < 8; i++)  
            broj += broj\_bin [i] \* (int)Mathf.Pow (2, i);

Na kraju se očitan binaran broj pretvara u int i prosljeđuje skriptama odgovornim za povezivanje igrača na server.

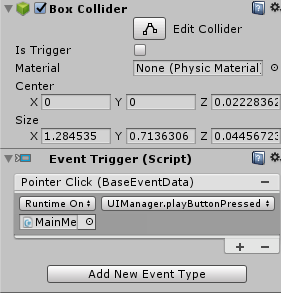
## Glavni izbornik

Scena prikazuje tipičnu biljarsku sobu koja sastoji od modela prostorije, pet slika, biljarske loptice i biljarskog stola koji su detaljnije objašnjeni u poglavlju Modeli. Razmještaj tih modela možemo vidjeti na slici 19. Za razliku od klasičnih glavnih izbornika u kojima su gumbi predstavljeni u obliku 2D tekstura, cilj ovoga glavnog izbornika je da su oni prikazani kao 3D modeli te se klikom na neke od njih pokrene animacija pomicanja glavne kamere koja zatim prikazuje sljedeći dio izbornika.



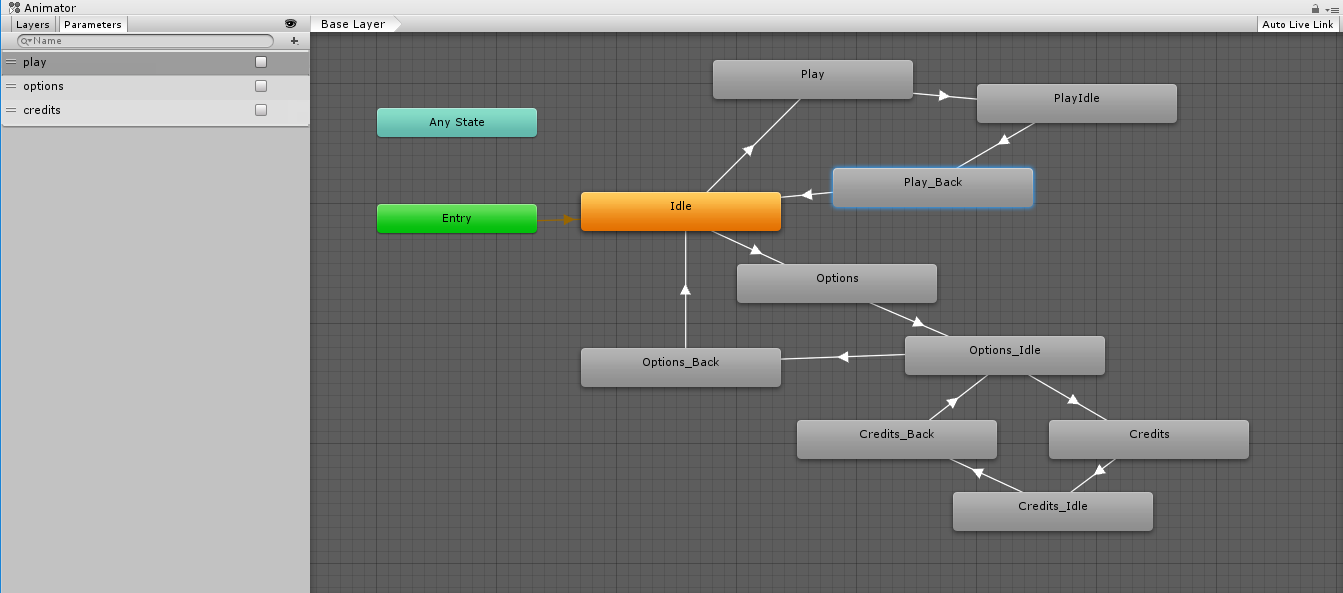
Slika 19. Scena glavnog izbornika

Da bi se omogućila izrečena funkcionalnost potrebno je glavnoj kameri dodati komponentu *GraphicRaycaster* koja omogućava slanje poruka 3D objekata koja sadrže sudarač (engl. collider) i implementiraju sučelja događaja (engl. event). Iz tog razloga objekti koji označavaju GUI komponentu implementiraju *EventTrigger* koji omogućuje da se klikom na njih pokrene specifična akcija(Slika 20).



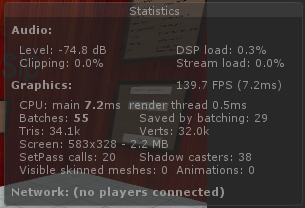
Slika 20. Implementacija Event Triggera

Animacije su ostvarene preko *Animation Controllera* koji se sastoji od stanja koja označuju pojedinu animaciju i koja su povezana tranzicijama od kojih pojedine sadrže uvjet prijelaza u drugo stanje(Slika 21).

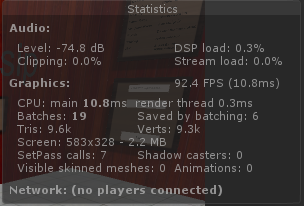


Slika 21. Animation Controller za glavnu kameru

Pri prvom ulasku u aplikaciju korisnik mora unijeti svoje novo korisničko ime koje će tijekom igre biti vidljivo ostalim korisnicima. Izbornik pruža mogućnost promijene korisničkog imena u bilo kojem trenutku. Za pohranjivanje i učitavanje korisničkog imena koristi se razred *PlayerPrefs* koji nudi već implementirane funkcije za tražene akcije. Pošto je ovo igra za mobilne uređaje, imali smo u planu što više optimizirati aplikaciju kako bi se bez problema izvršavala na njima. Iz tog razloga odlučili smo napraviti mape svijetla (engl. lightmapping) odnosno tehniku koja sprema sjajnost površine statičnih objekata u njihovu teksturu te se tada ne trebaju raditi dodatne kalkulacije. Razliku prije i poslije mapa svijetla prikazuju slike 22 i 23, te se može uočiti jasna prednost mapa svijetla odnosno smanjivanje *batches* i *SetPass* funkcijskih poziva.



Slika 22. Prikaz statistike prije lightmappinga

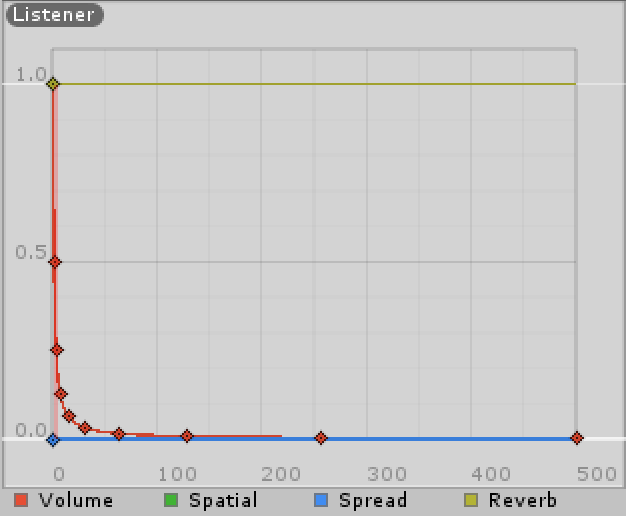


Slika 23. Prikaz statistike poslije lightmappinga

## Zvučni efekti

Izvedba zvukova u igri ostvarena je uporabom objekta koji je pretplaćen na događaje sudaranja i udarca biljarskim štapom. Kada se ostvari takav događaj, izvršava se metoda koja ovisno poziciji sudara u sceni i jačini udarca stvara *Unity GameObject* na zadanoj poziciji koji određenom jačinom zvuka izvodi zvučni zapis. Kako bi se umanjio značaj ponavljanja istog zvučnog zapisa, pri svakom izvođenju zvuka je visina tona zvuka promijenjena za nasumično odabrani koeficijent. Također, ako pri sudaru jačina udarca prijeđe određenu granicu, izvodi se drugi zvučni zapis kako bi se izrazio dojam jačeg udarca.

Izvedene zvukove prepoznaje audio primatelj koji je pridjeljen kameri. Time Unity brine o obradi dolazećih zvukova i kontrolira njihovu glasnoću i raspodjelu po kanalima ovisno o poziciji i udaljenosti od audio primatelja. Glasnoća zvuka koja se šalje na izlaz uređaja logaritamski opada ovisno o udaljenosti kako bi se ostvarilo ponašanje što bliže onom u stvarnosti (Slika 24).



Slika 24 – Prikaz karakteristika zvuka ovisno o udaljenosti od izvora

# Upute za korištenje

Kako bi se mogla pokrenuti igra, potrebno je imati dva mobilna uređaja s operacijskim sustavom *Android*, kamerom i mogućnošću povezivanja na bežičnu mrežu. Također je potreban i usmjeritelj/hotspot uređaj na kojeg će se ostali uređaji spojiti na lokalnu mrežu. Pratiti sljedeće korake:

* Pokrenuti instalaciju preuzetog programa (.apk)
* Ispisati potrebni marker u željenoj veličini
* Spojiti oba uređaja na isti usmjeritelj
* Pokrenuti instaliranu aplikaciju na oba uređaja
* Odabrati ulogu domaćina (engl. Host) na jednom te klijenta na drugom uređaju
* Na klijentskom uređaju uperiti kameru u ekran drugog uređaja kako bi se učitala adresa domaćina
* Kamere oba uređaja okrenuti prema markeru kako bi se na ekranima prikazala scena
* Za pronalazak dobre poziciju za udarac potrebno je kretati se, a scena će biti prikazana dok je marker vidljiv kameri
* Nakon pronađene dobre pozicije za gađanje pritisnuti gumb za ciljanje i namjestiti biljarski štap u željenu poziciju
* Podesiti željeni spin bijele kugle pritiskom i povlačenjem gumba za namještanje spina
* Pritisnuti gumb za pucanje i udaljavati se do željene jačine udara
* Kada je postignuta željena jačina udarca, potrebno je otpustiti gumb za pucanje kako bi bijela kugla bila pogođena biljarskim štapom
* Tijek igranja i pravila propisana su WPA "8-ball-pool" standardima
* Brisanje programa je moguće načinom uobičajenim za uređaje s operacijskim sustavom *Android*

# Budući razvoj

Tehnologija proširene stvarnosti pruža novo iskustvo. Spoj stvarnoga i virtualnoga na mobilnim uređajima može biti zanimljiv, ali i koristan. Potencijal tehnologije je uočen i iskorišten u svrhe unaprijeđenja prikaza informacija, reklamnih i edukativnih materijala. Time je moguće izraditi interaktivne programe koji vrlo detaljno prikazuju kompleksne sustave. Moderni mobilni uređaji su dovoljno jaki za podržavanje takve vrste interakcije s okolinom, a rješenja koja se mogu koristiti za izradu takvih programa moćna i jednostavna za korištenje.

Igra koju smo izradili, unatoč očekivano dobrom izvođenju zahtijeva određene dorade. Potrebno je dodati način igranja za jednog igrača kako ljudi nakon skidanja ne bi odustali u uvjerenju kako igra ne radi. Također je potrebno i provjeriti postoje li bolji zvukovi koji se mogu koristiti ili snimiti vlastiti, oblikovani prema potrebama igre. U slučaju objavljivanja igre jedna od prepreka je poprilično skupa Vuforia, barem za naše financijsko stanje i neprovjerenu igru. Rješenje toga problema je da uzmemo u obzir neke od besplatnih alata za proširenu stvarnost (npr. ARToolkit), no oni trenutno pružaju slabije mogućnosti od Vuforije.

# Literatura

[1] Vuforia Developer Portal - <https://library.vuforia.com/>

[2] Unity Manual - <https://docs.unity3d.com/Manual/index.html>

[3] Unity razvojna okolina - <https://unity3d.com/>