Razvoj aplikacije za virtualnu stvarnost

Devčić, Tea

SVEUČILIŠTE U ZAGREBU

PRIRODOSLOVNO–MATEMATIČKI FAKULTET

MATEMATIČKI ODSJEK

Tea Devčić

RAZVOJ APLIKACIJE ZA PROŠIRENU STVARNOST

Diplomski rad

Voditelj rada:
Goran Igaly, v. pred. dr. sc.

Zagreb, srpanj, 2021.

Ovaj diplomski rad obranjen je dana _____ pred ispitnim povjerenstvom u sastavu:

1. _____, predsjednik
2. _____, član
3. _____, član

Povjerenstvo je rad ocijenilo ocjenom _____.

Potpisi članova povjerenstva:

1. _____
2. _____
3. _____

Sadržaj

Uvod

Proširena stvarnost je tehnologija koja povezuje virtualne sadržaje sa stvarnim svijetom i time stvara novo, drugačije korisničko iskustvo. Najčešće se koristi u područjima zabave i edukacije, no sve veću primjenu nalazi i u raznim industrijama. Veliki porast interesa za to područje tehnologije doveo je do razvoja specijaliziranih programa pomoću kojih se jednostavno mogu razvijati aplikacije za proširenu stvarnost.

U prvom poglavlju rada objasnit ćemo pojmove proširene i virtualne stvarnosti, dati kratki pregled povijesnog razvoja te opisati pojmove vezane uz način rada aplikacije za proširenu stvarnost. U drugom poglavlju navest ćemo i opisati programe koji se koriste za izradu aplikacije. U zadnjem poglavlju objasnit ćemo izradu aplikacije pomoću prethodno opisanih programa te opisati neke mogućnosti koje ti programi nude.

Poglavlje 1

Općenito

1.1 Proširena stvarnost i virtualna stvarnost

Sve brži napredak tehnologije doveo je do razvoja novih uređaja poput pametnih telefona i tableta kojima se svakodnevno koristimo te sve češće preko njih dobivamo razne informacije. Proširena stvarnost i virtualna stvarnost tako postaju dio te tehnologije koja stvara nova, uzbudljiva i interaktivna iskustva.

Proširena stvarnost (*augmented reality, AR*) je tehnologija koja omogućuje interakciju između stvarnog i virtualnog svijeta preko nekog uređaja poput pametnog telefona. Pomoću nje se objekti iz stvarnog okruženja nadopunjuju dodatnim virtualnim sadržajima i tim putem digitalne informacije postaju dio stvarnog svijeta, barem iz korisničke perspektive.

Virtualna stvarnost (*virtual reality, VR*) je oblik računalne simulacije u kojoj se korisnik nalazi u potpuno umjetno napravljenom svijetu kojeg doživljava kao stvaran. Taj prikaz se ostvaruje pomoću dodatne tehnologije poput posebnog monitora i slušalica, a ponekad i dodataka za ruke koji omogućavaju kretanje unutar virtualnog svijeta.

Dok je u virtualnoj stvarnosti korisnik u potpunosti izvan svojeg trenutnog okruženja, proširena stvarnost daje korisniku mogućnost interakcije sa svojim okruženjem u realnom vremenu.

1.2 Povijesni razvoj

Ideja fizičkog svijeta povezanog s kompjuterski generiranim informacijama prvi put se spominje 1960-ih godina. Početak razvoja polja iz kojeg će se kasnije razviti VR i AR

pripisuje se Ivanu Sutherlandu. On u svojem eseju objavljenom 1965. godine opisuje ideju imerzivnih zaslona (*immersive displays*), odnosno uranjanja korisnika u potpuno drugačiji svijet koji se doima stvarnim. U istom eseju spominje kako pomoću vizualnog zaslona (*visual display*) korisnik lako može neko čvrsto tijelo učiniti prozirnim tj. može vidjeti kroz njega. Upravo se ta ideja smatra predviđanjem razvoja područja koje će se kasnije razviti u koncept zvan proširena stvarnost. Nedugo nakon objave eseja, Sutherland konstruira prvi VR sistem. 1968. godine završava prvi zaslon postavljen na glavu (*head-mounted display, HMD*) s mogućnošću praćenja kretanja glave i prozirne optike (*see-through optics*).

Razvoj tehnologije i performansi računala 1980-ih i početkom 1990-ih godina konačno daje proširenoj stvarnosti mogućnost razvoja kao zasebnom polju istraživanja.

1992. godine po prvi put se upotrebljava izraz „proširena stvarnost“. Izraz se pojavio u radu Thomasa P. Caudella i Davida W. Mizella koji je opisivao program koji pomaže radnicima u zrakoplovnim tvornicama Boeinga gdje im se, pomoću HMD-a i prozirne optike, daje mogućnost gledanja sheme snopa žica u avionima.

Idućih godina pojavljuju se sistemi koji uključuju tzv. AR vođen znanjem (*knowledge-based AR*) koji je sposoban automatski provoditi odgovarajući niz instrukcija za popravak i održavanje, kao i zasloni koji se mogu nositi u rukama te prepoznaju prostor i okolinu. Godine 1995. pojavljuje se prvi pravi - iako privezan – ručni AR zaslon povezan s radnom stanicom, ali opremljen s prednjom kamerom koji je preko povratnih informacija videa mogao detektirati obojene markere (*markers*) i prikazati informacije pomoću prozirnog videa.

Sljedeće godine razvija se prvi suradnički AR sistem koji daje mogućnost da ga istovremeno koristi više korisnika te se virtualni objekti pojavljuju u zajedničkom dijeljenom prostoru.

Godinu dana kasnije nastaje prvi vanjski AR sistem koji uz HMD s prozirnim prikazom koristi i GPS te praćenje orijentacije, ali koristi dodatnu opremu - ruksak s računalom i raznim senzorima te tablet. Već iduće godine (1998.) konstruiran je vanjski AR navigacijski sistem Map-in-the-Hat, a njegov nasljednik Tinmith se razvija u eksperimentalnu platformu za vanjski AR koja se koristi za naprednije primjene poput 3D mjerenja okoline, a zatim prerasta u prvu vanjsku AR igru.

Sve do 1999. godine nije postojao nikakav softver dostupan izvan nekog istraživačkog laboratorija. Situacija se promijenila kad se pojavio ARToolKit, prva softverska platforma otvorenog koda za AR. Sadržavala je biblioteku s 3D praćenim modelima koji koriste crno-bijele fiducijalne[1] markere (*fiducials*) koji se lako proizvode pomoću laserskih printera.

[1] **Fiducijalan** - koji se zasniva na povjerenju ili vjerovanju

Pametan dizajn softvera, u kombinaciji s porastom dostupnosti web kamera, učinio je ARToolKit vrlo popularnim.

Iste godine u Njemačkoj se započinje na razvoju programa za korištenje proširene stvarnosti u industriji, a ideja se proširila i globalno te se sve više radi na primjeni proširene stvarnosti u raznim granama industrije.

Nakon 2000. godine razvoj mobilne industrije ubrzano raste te se 2003. godine razvija prvi ručni AR sistem koji se pokretao autonomno na tzv. dlanovniku (*personal digital assistent, PDA*) – pretečom današnjih pametnih telefona.

Nekoliko godina kasnije, 2008. godine, predstavljen je prvi pravi iskoristivi sustav za praćenje i prepoznavanje okoline za pametne telefone, a kasnije se iz njega razvila Vuforia - program za razvoj AR aplikacija. Zadnjih godina sve više se radi na poboljšanju praćenja okoline što uključuje paralelno praćenje i mapiranje koje omogućava praćenje bez pripreme u nepoznatoj okolini te korištenja senzora dubine za gradnju fiktivnog 3D reljefa.

Također, razvija se sve više softverskih platformi kojima se programeri mogu služiti u razvoju vlastitih aplikacija, a do sada je proširena stvarnost osim u razvoju igrica i raznim tehnološkim industrijama našla primjenu i u medicini, edukaciji, turizmu, kulturi, ali i u novinarstvu i oglašavanju.

1.3 Što je važno za razvoj proširene stvarnosti?

U kontekstu proširene stvarnosti mogu se pronaći tri važna pojma povezana s načinom na koji aplikacija mjeri i poravnava virtualne objekte sa stvarnom okolinom, a to su praćenje (*tracking*), kalibracija (*calibration*) i registracija (*registration*).

Praćenje je izraz koji opisuje dinamičku „svijest o okolini“ i mjerenje kojim se AR sustav služi. Kako bi se neki virtualni objekt prikazao kao dio stvarnog trodimenzionalnog svijeta, najmanje što moramo znati jest relativan položaj, odnosno položaj i orijentaciju AR zaslona u odnosu na stvarni objekt. Kako se AR koristi u stvarnom vremenu te mjere moraju se konstantno ažurirati te je bitno u svakom trenu znati i položaj i orijentaciju.

Kalibracija je proces u kojem se uspoređuju mjerenja napravljena s dva različita uređaja, jednim referentnim i jednim koji će tek biti kalibriran. Referentni uređaj može se zamijeniti s referentnom vrijednošću ili s poznatim koordinatnim sustavom. Cilj je utvrditi parametre kojima će se novi uređaj koristiti, a u kontekstu proširene stvarnosti to se odnosi na parametre za što bolje praćenje položaja objekata u stvarnoj okolini. Pri tome se u obzir uzimaju neki interni parametri kojima se koristi kamera i pogreška optičke leće kamere.

Registracija u kontekstu proširene stvarnosti se odnosi na prostorno poravnavanje objekata unutar nekog koordinatnog sustava. Cilj registracije je što preciznije spajanje stvarnog i virtualnog poštivajući pravila poput perspektive i preklapanja. Statičkom registracijom se dobiva uvid u okolinu preko kamere koja se ne pomiče, pri čemu se zahtijeva kalibracija praćenog sustava da bi se utvrdio zajednički koordinatni sustav između virtualnih i stvarnih objekata. Dinamičkom registracijom dobiva se uvid u okolinu u kojoj se korisnik s kamerom pomiče, a to zahtijeva praćenje.

Određivanje ciljanih objekata

Prilikom pronalaska i određivanja ciljanih objekata (npr. slike, 3D modela ili samog prostora), AR sustav uzima u obzir više transformacijskih sustava. Transformacija modela opisuje položaj i orijentaciju objekta koji se pomiče u statičnoj okolini. Transformacija pogleda opisuje položaj i orijentaciju kamere, nekog senzora za praćenje ili zaslona u toj istoj okolini. Obje transformacije mogu biti praćene što omogućava registraciju. Transformacija projekcije opisuje odnos 3D koordinata kamere i 2D koordinata nekog uređaja, odnosno sadržaj onoga što vidi „oko“ mapira se na ekran. Takva transformacija je obično izvanmrežno kalibrirana i treba biti zasebno napravljena za svaku kameru i za svaki uređaj.

Različiti uređaji pomoću raznih senzora prepoznaju mnoge fizičke komponente koje mogu poslužiti u praćenju kao su zvuk, gravitacija, inercija, a postoje i metode kojima se kombinacijom senzora može odrediti položaj određene površine i udaljenost neke točke na toj površini što daje mogućnost stavljanja virtualnog objekta na pravo mjesto u pravoj mjeri za što vjerniji prikaz.

Dakako, u obzir se uzimaju i moguće pogreške mjerenja te se u skladu s poznatim pogreškama poput rezolucije senzora ili preciznosti uređaja razvijaju metode i tehnologije koje će neizbježne pogreške smanjiti što je više moguće. U protivnom, u AR aplikaciji može doći do kašnjenja neke informacije ili privremenog prekida gibanja virtualnog objekta što smanjuje vjerodostojnost prikaza.

Iako se korisnik može poslužiti sve većim brojem različitih tehnoloških uređaja za primjenu proširene stvarnosti, još uvijek su najzastupljeniji i najdostupniji mobilni uređaji. Dok su s jedne strane nedostaci tih uređaja manjak memorije, korištenje jedne kamere, slabija baterija, s druge strane ovakvi uređaji imaju mnogo prednosti. Glavna prednost je ta što je korisnik neovisan o dodatnim uređajima ili žicama, slobodno se može kretati i u zatvorenom i u otvorenom prostoru. Nadalje, mobilni uređaji nisu preveliki i lako se drže u jednoj ruci što omogućava lakšu interakciju. Gotovo svi uređaji mogu se povezati s internetom što daje širok spektar informacija koje se mogu iskoristiti u različitim vrstama AR aplikacija, a opremljeni su i s nizom integriranih senzora koji pomažu u performansama.

Senzori poput GPS-a i bežičnih mreža omogućuju određivanje pozicije mobilnog uređaja koja se preko kamere unutar neke AR aplikacije može nadopuniti nekim korisnim informacijama i interaktivnim sadržajima. Također unutar mobilnih uređaja možemo naći i kompas, žiroskop i linearni akcelerometar koji služe za određivanje orijentacije i brzine kao i relativnog položaja.

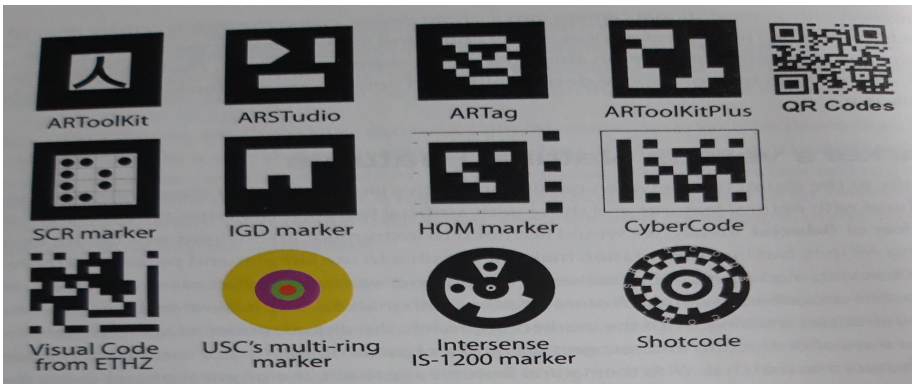
Ipak najveću ulogu igra kamera preko koje se uz pomoć tehnika računalnog vida može obavljati praćenje. Da bi se neki virtualni objekt, bio on trodimenzionalan ili dvodimenzionalan, mogao uspješno spojiti sa stvarnom okolinom potrebno je odrediti cilj, odnosno neku ciljanu sliku (*target image*), ciljani objekt (*target object*) ili detektirati različite površine u prostoru na koje možemo staviti naš virtualni model. U tome nam pomažu sustavi koji detektiraju tražene objekte ili površine pomoću nekih karakteristika poput izraženih točaka, kontrasta boja ili interakcije sa svjetlom.

Kada je riječ o prepoznavanju neke površine ili modela pomoću svjetla koristi se nekoliko metoda.
Jedna metoda je pasivno osvjetljenje koje obuhvaća prirodno i umjetno svjetlo. Pri korištenju takvog osvjetljenja praćenje se fokusira na fizička obilježja, a najveći izazov je omogućiti brzo i efikasno praćenje unatoč različitim izvorima svjetla. Objekti od interesa se traže na temelju dovoljno dobrog kontrasta koji zahtijeva istaknute značajke u zadanoj okolini i dovoljno indirektnog svjetla kako bi se iste značajke isticale na slici. Problem se stvara u zatvorenom prostoru gdje zna nedostajati prirodnog svjetla ili bude previše umjetnog što ometa rad digitalnih kamera.
Druga metoda je aktivno osvjetljenje koje prevladava prepreku vanjskog svjetla tako da spaja optičke senzore s aktivnim izvorom svjetlosti. Kako aktivna svjetlost vidljivog spektra mijenja izgled okoline, sve veći pristup rješavanja tog problema temelji se na infracrvenoj svjetlosti. Taj izvor svjetlosti osvjetljava dani prizor, ali ostane neprimijećen od strane korisnika. Takav način korištenja vrlo je efikasan kod AR aplikacija koje se odvijaju pod jakom sunčevom svjetlosti.
Treća metoda koristi strukturirano osvjetljenje tako da se na određenu okolinu projicira znani uzorak kojeg proizvode konvencionalni projektori ili laseri. Kamera tako promatra odraze koje zatim koristi kako bi detektirala geometriju okoline i objekta sadržanog u njoj te tim postupkom aktivno istakne značajke okoline. Ova metoda funkcionira pri korištenju i infracrvenog spektra i vidljivog spektra.

Slično kao i kod osvjetljenja, ciljeve koje tražimo praćenjem (*tracking targets*) možemo podijeliti na skupinu ciljeva koji sadrže neke prirodne značajke i one koje sadrže umjetne, a potonje najčešće nazivamo markerima (*markers, fiducials*).

Kao što je već spomenuto, optičko praćenje zahtijeva dovoljno kontrasta na slici kako bi se ona prepoznala. Ovisno o raznim okolnostima taj uvjet nije uvijek nužno ispunjen. U stvar-

nom svijetu postoje objekti koji su podjednako obojeni ili nemaju gotovo nikakvu teksturu, npr. bijeli zidovi, te je na takvim objektima vrlo teško pronaći ikakvu istaknutu značajku. Drugi problem koji se javlja je spekularna refleksija, odnosno refleksija pri kojoj se svjetlost odbija od nekog objekta u istoj količini i pod istim kutem pod kojim i upada. Takvi objekti su obično presjajni i jako nestabilni za kameru koja se pomiče. Također postoje i objekti koji na sebi imaju ponavljajući uzorak, kao npr. fasade s identičnim prozorima, te se u tom slučaju ne može jednoznačno odrediti neka istaknuta značajka. Takve situacije mogu se izbjeći korištenjem markera.
Markeri su poznati uzorci koji se nalaze na površini nekog ciljanog objekta ili određeni prativi oblici (*trackable shapes*) prilijepljeni na ciljanu metu, a oni sami su napravljeni tako da ih se lako i pouzdano prepozna. Kod izrade markera bitan je dobar odabir oblika koji će dati optimalni kontrast i biti lako detektiran. Najkorišteniji oblici su kvadrati i krugovi, najčešće crno-bijeli kako bi se dobio najbolji kontrast.



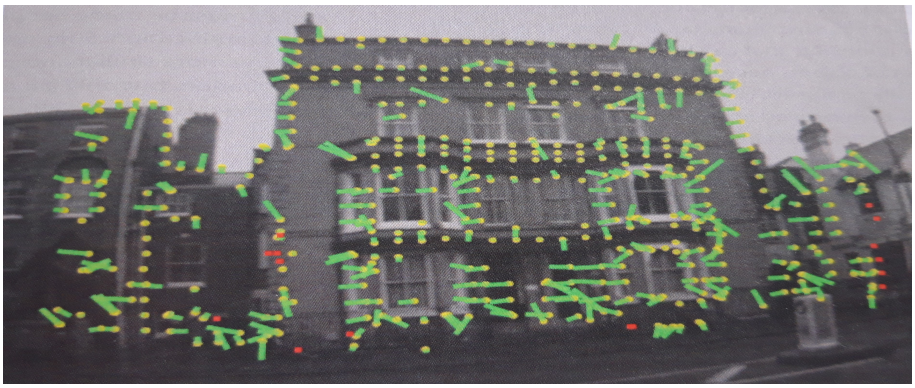
Slika 1.1: Primjeri markera

Ako nismo u mogućnosti ili jednostavno ne želimo koristiti markere, „prisiljeni“ smo koristiti se prirodnim značajkama ciljanog objekta. Takav način praćenja obično zahtijeva bolju kvalitetu slike i više računalnih resursa i tek je odnedavno postao popularan.

Najčešće korištene prirodne značajke nazivaju se točke od interesa (*interest-points*) ili ključne točke (*keypoints*), a čine ih istaknute točke značajki na ciljanom objektu. Točke od interesa trebale bi biti lako detektirane i njihova lokacija na objektu bi trebala ostati stabilna i tijekom promjene točke gledišta. U praksi, korištenje točaka od interesa zahtijeva dovoljno gustu i nepravilnu teksturu površine.

Slika 1.2: Detekcija točaka od interesa

Objekti koji ne posjeduju takvu površinu mogu se pratiti korištenjem rubnih značajki, odnosno detekcijom izraženih i lako uočljivih rubova. Međutim, pojedinačni rubovi rijetko daju neku jedinstvenu identifikaciju bez nekog dodatnog znanja o objektu, stoga se mora koristiti više rubova povezanih u neku pouzdaniju cjelinu.



Slika 1.3: Detekcija značajnih rubova

Osim pronalaska nekih lokalnih značajki poput točaka i bridova, također se može koristiti metoda ključnog kadra (*keyframe*) gdje se kadar uhvaćen kamerom u realnom vremenu uspoređuje s ključnim kadrom uhvaćenim s neke određene točke gledišta. Nažalost, ovakav pristup je teško skalirati u nekim većim okolinama.

Za neke jednostavnije AR aplikacije ili aplikacije koje ne zahtijevaju kretanje korisnika nego se interakcija odvija nad istom statičnom površinom, kao ciljani objekt može poslužiti

slika ili trodimenzionalni objekt poput kocke ili cilindra. Kako bi se objekt jednostavno i brzo detektirao, kao što je prethodno opisano, bitno je da mu se tekstura sastoji od mnogo istaknutih značajki, poput istaknutih točaka ili rubova, dobrog kontrasta bez ponavljajućih uzoraka. Iako se prethodnim definiranjem praćenog objekta možda gubi na slobodi, s druge strane se dobiva brzina i pouzdanost.

Najčešći problemi zbog kojih dolazi do sporije detekcije ili prekida sadržaja nakon detekcije cilja u zatvorenom prostoru jesu gubljenje samog praćenog objekta zbog micanja njega samog izvan kadra, brže pomicanje kamere ili objekta koje uzrokuje zamagljenost, naginjanje kamere pod kutem kojim se gubi većina prikaza ciljanog objekta, bljesak ili neki drugi odsjaj koji prekriva dio slike kao i ulazak bilo kojeg drugog objekta ispred našeg ciljanog objekta koji ga tim postupkom zaklanja.
Ipak, ove probleme korisnik može riješiti tako da u prostoriji napravi idealnije uvjete za razliku od vanjskog prostora čije uvjete korisnik ne može samostalno i u potpunosti riješiti.

Neki od problema vanjskog praćenja uključuju mobilnost, odnosno slobodno kretanje korisnika po širem području koje ponekad nije idealno. Neki senzori u takvim područjima ne mogu najpreciznije odrediti zadano, odnosno dolazi do slabljenja signala i pucanja veze. Također, i sama tekstura okoline neće uvijek biti idealna. Postoji mnogo neiskoristivih površina poput trave ili asfalta, sličnih zgrada, a u obzir se trebaju uzeti i prirodne pojave poput kiše ili magle koje mogu stvoriti zamagljenost, kao i neki objekti u pokretu poput ljudi i automobila. Posljednja stvar je tzv. lokalizacija baze podataka. Da bi naš model praćenja dobro radio u nekom poznatom vanjskom području od njega se zahtijeva da bude svjestan mnogih stvari koje detektira u okolini. Što je više stvari oko njega baza podataka raste te može doći do rušenja aplikacije zbog prevelike potrošnje memorije.

Kako što vjernije prikazati virtualni objekt

Nakon detekcije ciljanog objekta, od AR aplikacije želimo da nešto s njim napravi – da mu promijeni izgled, nadopuni pronađeni objekt nekim opisnim sadržajem ili da jednostavno doda nešto u okolinu.

Virtualni objekti kao što su dvodimenzionalne slike ili animacije, videi ili običan tekst lako se mogu dodati i ne zahtijevaju nikakva dodatna podešavanja, osim možda neke estetike, no kako živimo u trodimenzionalnom svijetu od istih takvih objekata očekujemo da se ponašaju u skladu s njim, odnosno želimo da se prilagode uvjetima vanjskog svijeta i da izgledaju što „življe“ i vjerodostojnije. Kao što je već ranije spomenuto, u procesu registracije stvarni i virtualni koordinatni sustavi se poravnavaju i preko dane relativne pozicije korisnika (ili kamere) virtualni objekt se može pravilno pozicionirati i orijentirati u stvarnoj okolini. Nakon što se virtualni objekt smjesti u okolinu očekujemo od njega da

se ponaša u skladu sa svojom strukturom. Želimo da čvrsti objekt stoji na mjestu na koje smo ga stavili i da pri tome dotiče tlo, neki lebdeći objekt želimo da miruje u zraku ili da se polako giba kako bi se dobio dojam da lebdi. Nadalje, od virtualnog objekta očekujemo da bude svjestan svoje okoline i da se prilagodi uvjetima kao što su svjetlo, sjena, okluzija i veličina.

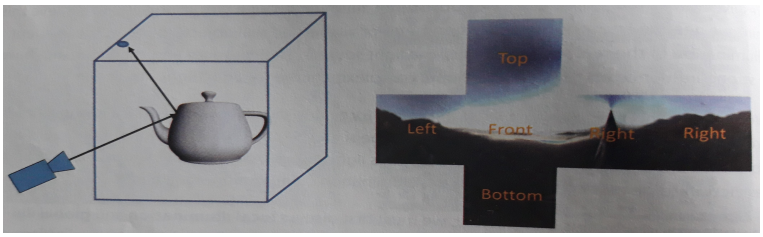
Ljudsko oko percipira dubinu na nekoliko načina. Jedan od njih je i veličina promatranog objekta. Da bi se to postiglo i virtualni objekti u AR aplikaciji ponašaju se u skladu s linearnom perspektivom. Nakon što se objekt postavi u okolinu, bilo automatski preko unaprijed definirane funkcije aplikacije ili korisnikovom interakcijom, taj objekt se najprije usidri. Sidrenje je postupak kojim se dani objekt postavlja u okolinu na određene koordinate te se nadalje, u skladu s tim koordinatama, određuje njegova relativna pozicija u odnosu na kameru i orijentacija, ali se predmet ne miče s danih koordinata kada se kamera pomiče do ili od njega ili kruži okolo. Kako bi se dobila percepcija dubine, kada se kamera približava danom virtualnom objektu njegova veličina se povećava, tekstura postaje detaljnija, a boja postaje svjetlija. Kada se kamera udaljava događa se obratno. Kada kamera kruži oko objekta na zaslonu se prikazuje dani objekt iz drugog kuta.

Druga percepcija dubine jest okluzija ili preklapanje. Smatra se da su objekti koji preklapaju neki drugi objekt bliži, a to se pokušava postići i u proširenoj stvarnosti. Okluzija igra veliku ulogu u stvaranju što vjernijih prikaza virtualnih objekta, a razlikujemo dva slučaja. Prvi slučaj jest da se naš virtualni objekt nalazi ispred stvarnih objekata. Taj slučaj je jednostavniji i rješava se tako da se virtualni objekt samo iscrta na pozadinu videa.
Drugi, kompliciraniji slučaj, jest stavljanje virtualnog objekta iza stvarnog. Osnovni algoritam koji postiže taj efekt koristi tzv. fantome (*phantoms*) koji koriste standardni z-međuspremnik (*z-buffer, depth buffer*) koji se koristi za dubinu. Fantom je virtualna reprezentacija stvarnog objekta koji se nevidljivo iscrtava tj. modificira se samo z-međuspremnik. Time se utvrđuje točna vrijednost dubine stvarnog objekta vidljivog u videu nakon čega se može iscrtati virtualni objekt pri čemu se dijelovi prekriveni stvarnim objektom ne iscrtavaju, a za to se brine algoritam za z-međuspremnik (*z-buffer algorithm*).

Slika 1.4: Postavljanje virtualnog objekta u stvarnu okolinu bez korištenja okluzije (lijevo) i sa korištenjem okluzije (desno)

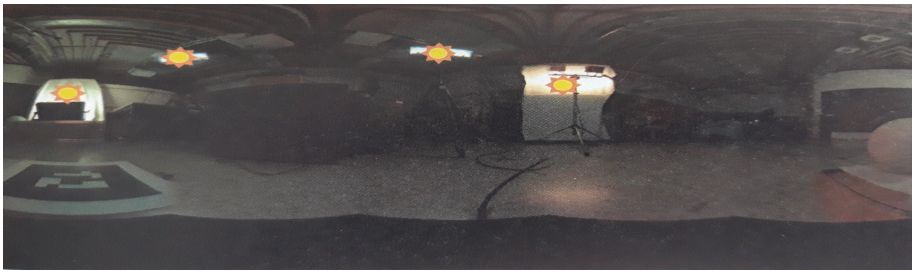
Osvjetljenje također ima ulogu u percepciji dubine. Objekti koji su bliži obično su svjetliji i bistriji dok su oni dalji tamniji i sjenovitiji. Da bi se postigao takav efekt koriste se različite metode za detekciju raznih izvora svjetlosti i u obzir se uzimaju lokalno i globalno osvjetljenje. Lokalno osvjetljenje u obzir uzima samo put svjetla od izvora do neke zadane površinske točke u danoj okolini. Globalno osvjetljenje podrazumijeva kompleksne interakcije svjetla s ostalim objektima iz okoline. Stoga takvo osvjetljenje proizvodi i refleksiju, refrakciju i sjene, a u obzir uzima i materijale objekata te na temelju njih određuje na koji način se svjetlo odbija od danog objekta.

Jedna metoda osvjetljavanja koja koristi lokalno osvjetljenje je tzv. mapa okoline (*environmental map*). Tom metodom se usmjereno svjetlo sprema u dvodimenzionalnu tablicu koja se potom primjenjuje na objekt, čime se dobije sjenčanje.



Slika 1.5: Određivanje mape okoline (lijevo) i njezin 2D prikaz (desno)

Metoda osvjetljavanja koja koristi globalno osvjetljenje pretvara svaki piksel mape svjetlosti (*radiance map*) u određeni izvor svjetlosti koristeći pretpostavku da je taj izvor svjetlosti fiksne veličine, počevši od centra dane okoline. Kako globalno osvjetljenje u obzir uzima više izvora svjetlosti, da bi se dobio realniji prikaz osvjetljenja određeni izvori svjetlosti mogu se procijeniti tako da se mapa okoline podijeli na dijelove jednakog osvjetljenja i u svakom dijelu se određuje reprezentativna točka.



Slika 1.6: Reprezentativne točke u prostoru

Još jedna komponenta koja pomaže u percepciji dubine i daje osjećaj stvarnosti je sjena koju baca objekt. Dvije metode koje se najviše koriste su volumen sjene (*shadow volume*) i mapiranje sjene (*shadow mapping*).

Volumen sjene je frustum (krnja piramida ili krnji stožac), koji okružuje objekt koji leži u sjeni u odnosu na izvor svjetlosti i dani poligon koji baca sjenu, a stranice frustuma nazivaju se poligoni volumena sjene. Nakon što se pomoću videa inicijalizira međuspremnik okvira (*frame buffer*), prvi prolaz iscrtava fantome u z-međuspremnik. Volumeni sjena virtualnog objekta iscrtavaju se u međuspremnik šablone (*stencil buffer*). Koristeći maskiranje međuspremnika šablone, kreiraju se sjene bačene s virtualnog na stvarni objekt, a to je dobiveno spajanjem svih piksela iz videa, koje je međuspremnik šablone označio kao sjene, s prozirnim zatamnjenjem (*dark transparent*) kako bi se dobio dojam sjenovitog dijela. U drugom prolazu cijela okolina, uključujući i virtualne objekte i fantome, iscrtava se u međuspremniku boje (*color buffer*). Volumeni sjene i virtualnog objekta i fantoma iscrtavaju se u međuspremnik šablone koji se koristi kao maska te se potom ponovo iscrtava cijela okolina takva da se dijelovi virtualnog objekta koji su u sjeni čine neosvijetljenima od danog izvora svjetlosti.

Mapiranje sjene je dvoprolazna tehnika koja radi tako da se u prvom prolazu prikaže međuspremnik dubine dane okoline iz perspektive izvora svjetlosti, a zatim se u drugom prolazu koristi ta mapa sjene, pri čemu je okolina prikazana iz perspektive promatrača, kako bi se odredilo je li dio prekriven iz perspektive izvora svjetlosti. Ako je dubina dijela unutar koordinata izvora svjetlosti veća nego dubina početnog razine mape sjene, taj dio je u sjeni.
Problem na koji nailazi ova metoda je tzv. problem dvostruke sjene, a odnosi se na preklapanje umjetne sjene, one koju virtualni objekt baca na stvarni objekt, i stvarne sjene na koju neki drugi objekt baca svoju sjenu. Kada dođe do takve situacije već prirodno tamni pikseli postaju još tamniji što ne izgleda dovoljno stvarno. Trenutno se radi na rješavanju tog problema, a jedna predložena tehnika je proširenje metode mapiranja sjena.

Poglavlje 2

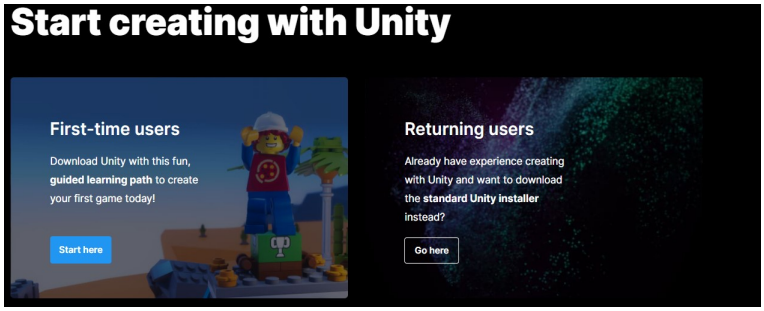
Programi za razvoj aplikacije za proširenu stvarnost

2.1 Unity

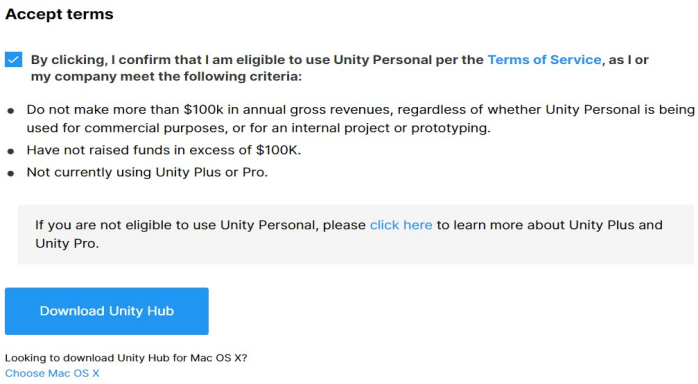
Unity je višeplatformski (*cross-platform*) softver za razvoj video igara, a razvila ga je tvrtka Unity Technologies te prvi put najavila i objavila u lipnju 2005. godine. Od tada se Unity postupno proširuje kako bi podržao različite razvojne platforme kao što su stolna računala, mobilne platforme za Android i iOS, razne konzole i platforme za virtualnu stvarnost. Može se koristiti za izradu 3D i 2D igara, a smatra se i softverom kojeg lako koriste i početnici i profesionalci. Osim u industriji za razvoj videoigara, Unity se sve više koristi i u filmskoj i automobilskoj industriji kao i za modeliranje raznih konstrukcija u arhitekturi. Također, unutar Unityja moguće je koristiti i dodatne alate koji služe za razvoj aplikacija za proširenu stvarnost.

Instalacija i početak rada

Sve potrebne programe za instalaciju Unityja možemo pronaći na službenoj stranici https://unity.com/. Kada se otvori stranica na desnoj strani glavnog izbornika nalazi se plavi gumb na kojem piše **Get started**. Pritiskom na taj gumb otvara se stranica Unity Store na kojoj se nalaze različite verzije planova za preuzimanje te opisi o plaćanju i mogućnostima koje se nude za koji izbor plana. Za studente ili početnike na izbor je dana besplatna verzija Unityja. Ponovo pritiskom na dani gumb **Get started** otvara se stranica na kojoj je moguće preuzeti Unity Hub pritiskom na gumb **Start here** za nove korisnike ili **Go here → Download Unity Hub** za stare korisnike. Pritiskom na taj gumb instalira se Unity Hub, samostalna aplikacija koja omogućava daljnje instaliranje željenih verzija Unityja, kreiranje novih projekata, preuzimanje raznih tutorijala i upravljanje licencama.

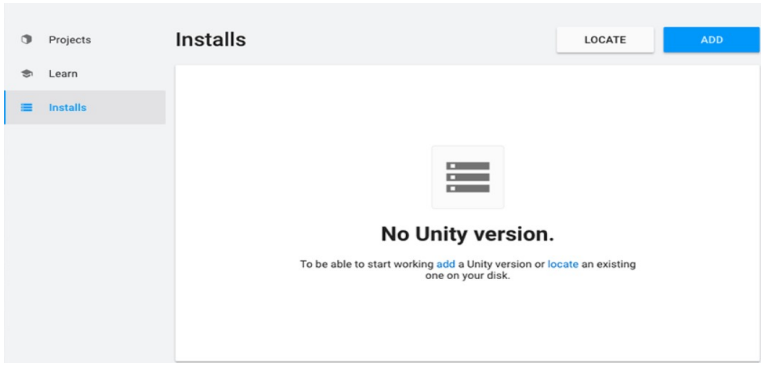


Slika 2.1: Službena stranica za preuzimanje



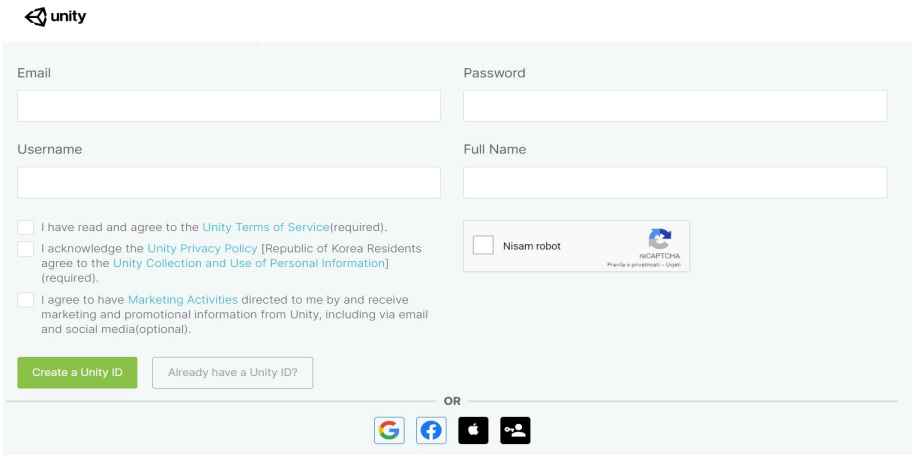
Slika 2.2: Početak preuzimanja

Nakon što je završilo preuzimanje, možemo otvoriti Unity Hub gdje su nam ponuđene instalacije raznih verzija Unityja.



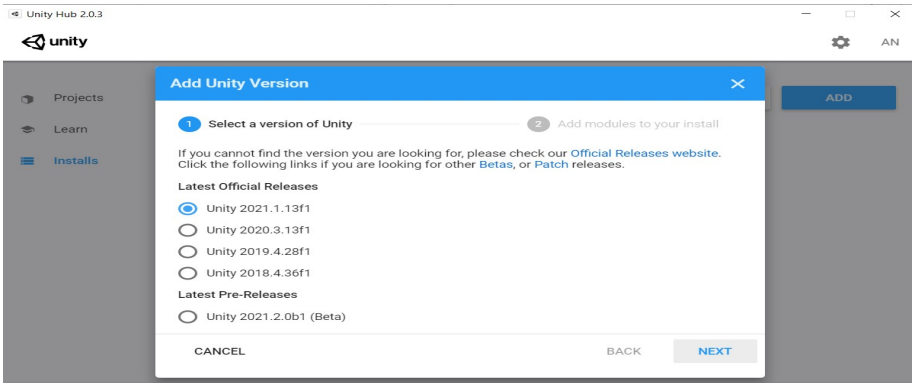
Slika 2.3: Unity Hub

Prije prve instalacije potrebno je napraviti **Unity ID**, odnosno račun za Unity. Ukoliko sam Unity Hub ne ponudi mogućnost izrade računa, on se može kreirati preko službene stranice koja u glavnom izborniku u desnom kutu ima opciju kreiranja računa ili logiranje ukoliko korisnik već ima račun. Nakon izrade računa unutar Unity Huba u desnom kutu postoji opcija za prijavu te se prijavljujemo s kreiranim Unity računom. Nakon što smo se prijavili možemo početi instalaciju Unityja.



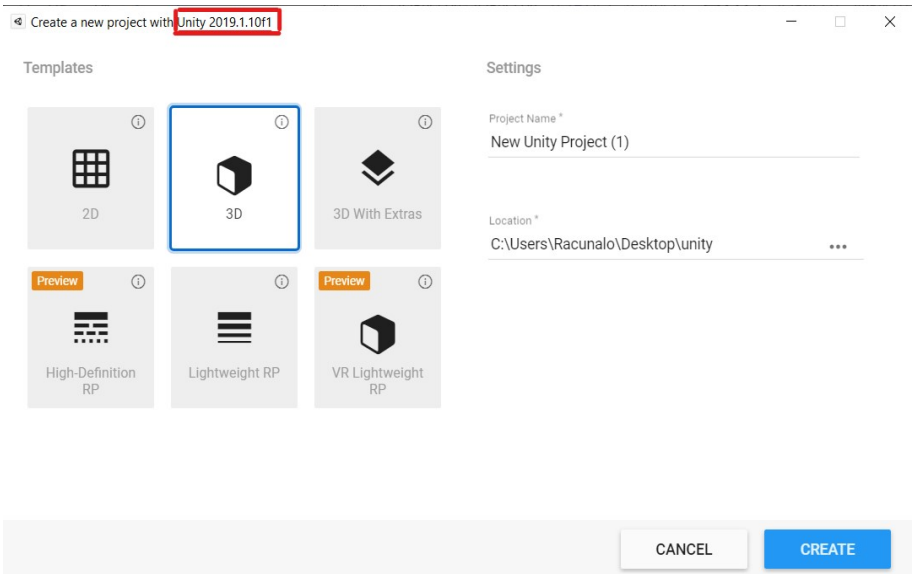
Slika 2.4: Kreiranje računa za Unity

Za instalaciju određene verzije Unityja (prvi put ili dodavanje nove verzije) idemo na **Installs** → **Add** gdje odabiremo željenu verziju. Nakon što se instalira dana verzija bit će prikazana pod **Installs**.

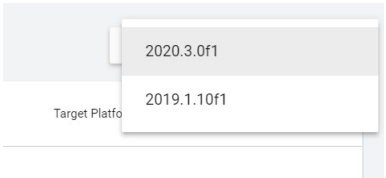


Slika 2.5: Dodavanje određene verzije Unityja u Unity Hub

Novi projekt kreiramo tako da u Unity Hubu u **Projects** odaberemo opciju **New**, izaberemo određeni template te u **Settings** upišemo ime našeg projekta i odaberemo lokaciju gdje će se on pohraniti, a zatim pritisnemo **Create** te se potom otvara naš novi projekt. Ukoliko imamo instalirano više verzija Unityja, novi projekt će se otvoriti u zadnjoj korištenoj verziji. Ako želimo odabrati neku drugu verziju odaberemo gumb s trokutićem (▽) pokraj **New** te odaberemo željenu verziju nakon čega nam se opet otvara template. Ako smo template otvorili samo preko **New** i nismo sigurni o kojoj verziji se radi, ona je napisana u gornjem lijevom kutu.



Slika 2.6: Odabir Unity templatea

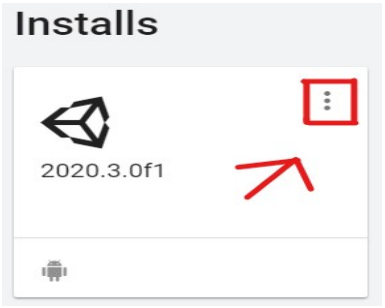


Slika 2.7: Odabir verzije Unityja za novi projekt

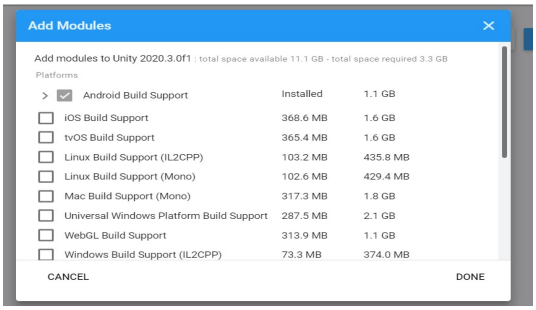
Kada otvorimo Unity Hub, u **Projects** se nalaze naši prethodni projekti, poredani kronološki počevši od najnovijeg, a otvaranje nekog već postojećeg projekta vrši se tako da se samo lijevom tipkom miša pritisne taj projekt. Ukoliko postojećeg projekta nema na popisu, možemo ga dodati preko **Add**.

Instalacija dodatnih modula

Kao što je već navedeno, Unity podržava različite module za razvoj na različitim platformama. Samom instalacijom Unityja ti moduli nisu automatski instalirani nego ih moramo instalirati ručno, ovisno o modulu koji nam treba. Instalaciju modula radimo preko Unity Huba gdje odaberemo Installs, zatim pritisnemo tri točke (⋮) kod željene verzije i odaberemo **Add Modules**. Nakon toga otvara se popis modula koje možemo instalirati te za instalaciju željenih modula pokraj naziva tog modula pritisnemo kvadratić, unutar kojeg se pokaže kvačica, a kada smo gotovi s odabirom pritisnemo **Done** i instalacija se započinje. Nakon što se odabrani moduli instaliraju ispod određene verzije Unityja prikazana je ikona instaliranog modula.



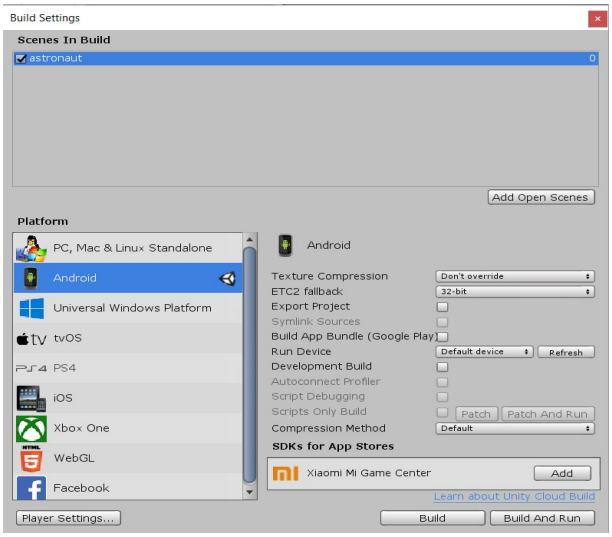
Slika 2.8: Opcija za instaliranje modula



Slika 2.9: Popis mogućih modula

Instalacija i podešavanje modula za mobilne aplikacije

Unity podržava razvoj aplikacija na mobilnim platformama za Android i iOS te su za obje platforme dostupni moduli za instalaciju. Prilikom instalacije modula za Android potrebno je instalirati **Android Build Support** te **Android SDK & NDK** Tools. Za instalaciju modula za iOS potrebno je instalirati **iOS Build Support**. Nakon kreiranja novog projekta zadana radna platforma u Unityju je platforma za razvoj aplikacije za PC, Mac i Linux. Ako želimo razviti aplikaciju koju će podržavati Android ili iOS sustav moramo u Unityju zamijeniti platforme. To radimo tako da odemo u **File → Build Settings...** i odaberemo željenu platformu, a zatim **Switch Platform**. Nakon što smo promijenili platformu, u gornjem lijevom kutu kod naziva našeg projekta trebalo bi pisati ime odabrane platforme.



Slika 2.10: Promjena platforme

Kod korištenja Android platforme treba podesiti SDK (*Software Development Kit*) i NDK (*Native Development Kit*) preko **Edit → Preferences... → External Tools** pa pod **Android** vidimo opcije za podešavanje SDK i NDK. Ako već nije automatski podešeno izaberemo opciju (kvačica) lijevo od **Android SDK Tools installed with Unity (recommended)** (isto napravimo i za NDK), te bi se trebala pojaviti putanja (*path*), odnosno hijerarhija mapa u kojoj se nalazi naša datoteka za određeni Development Kit. Ukoliko je došlo do pogreške i Unity automatski ne želi predložiti putanju, trebamo ju dodati ručno tako da isključimo opciju **... installed with Unity...** i dodamo željenu putanju preko **Browse**. Kopiramo cijelu putanju počevši od mape u kojoj je instaliran Unity.
Primjer za NDK putanju:
C:\Program Files\Unity\Hub\Editor\2020.3.0f1\Editor\Data\PlaybackEngines\AndroidPlayer\NDK
Kod korištenja iOS platforme potrebno je prethodno imati instaliran Xcode.

Nakon podešavanja platforme i SDK/NDK trebamo još otvoriti **Edit → Project Settings → Player → Other Settings → Identification → Package Name (Android) / Bundle Identifier (iOS)** i podesiti identifikator za aplikaciju koju radimo. Identifikator je zadanog oblika **com.CompanyName.ProductName**, a mi moramo zamijeniti polja CompanyName i ProductName s nazivom naše aplikacije. (pod CompanyName možemo staviti bilo što, ono samo služi za lokaciju tzv. preferencijskih datoteka (*preference files*).
Ako programiramo za Android, pod **Minimum API Level** možemo podesiti najmanju zahtijevanu razinu API-ja na korisničkom uređaju, a najmanju verziju za iOS podešavamo pod **Target Minimum iOS Version**.

2.2 Vuforia

Vuforia je platforma za razvoj AR aplikacija s podrškom razvoja na vodećim mobilnim platformama, tabletima i drugim uređajima za korištenje proširene stvarnosti. Omogućava jednostavno dodavanje naprednih funkcionalnosti računalnog vida u različite aplikacije pružajući mogućnost prepoznavanje slika i objekata kao i interakciju virtualnog sa stvarnim. Vuforia nudi različite oblike praćenja ciljanih objekata kao što su pojedinačne slike, slike omotane oko cilindra, više slika odjednom, različiti modeli, 3D skeneri, površine ili prilagođeni markera zvani VuMarkers, a u online biblioteci nudi i primjere kako izabrati i koristiti određene ciljane objekte.
Više o tome se može naći na:
https://library.vuforia.com/features/overview.html

Također Vuforia nudi i popis uređaja koji podržavaju dani SDK i koje su sve funkcionalnosti podržane za koji uređaj.
Više o tome može se pronaći na:
https://library.vuforia.com/platform-support/vuforia-engine-recommended-devices.html

Instalacija i dodavanje u Unity

Unity platforma podržava Vuforiu i ona se može instalirati unutar Unityja na dva načina. Starije verzije Unityja daju mogućnost dodavanja Vuforije preko modula. Ukoliko određena verzija Unityja posjeduje modul, onda se on instalira preko Unity Huba tako da se pod **Installs** kod određene verzije ode u **Add Modules** te se odabere **Vuforia Augmented Reality Support**. Nakon što se instalira, kod određene verzije prikazat će se ikona za Vuforiu.

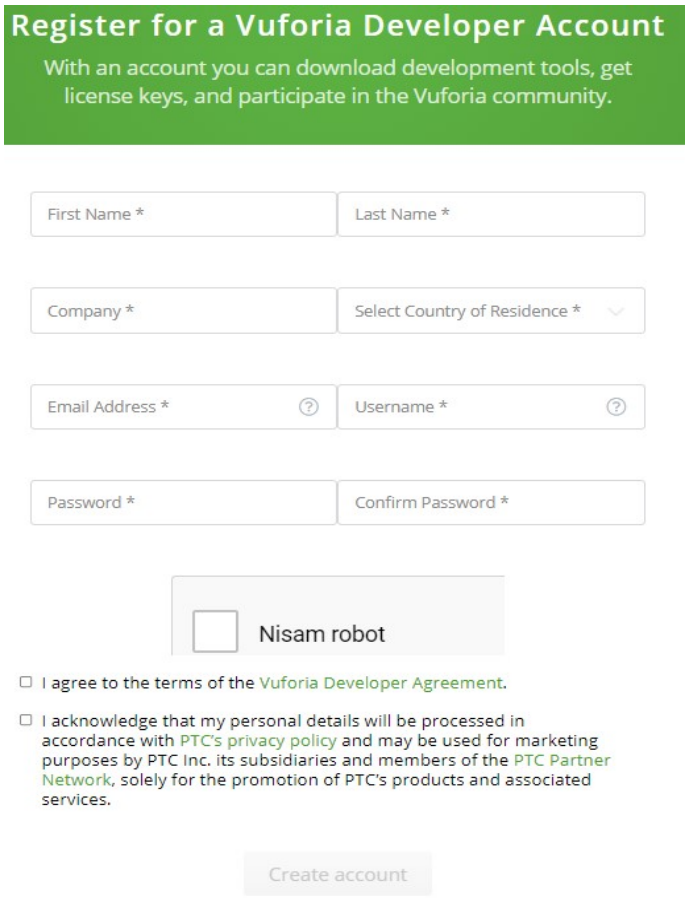
Ako neka verzija ne posjeduje direktnu instalaciju Vuforije tada je potrebno otići na službenu stranicu https://developer.vuforia.com/ i u glavnom izborniku pritisnuti **Downloads** te nakon toga odabrati **Add Vuforia Engine to a Unity Project or upgrade to the latest version** pri čemu se preuzima najnovija verzija Vuforia SDK-a. Nakon što je završilo preuzimanje, otvorimo danu verziju Unityja te izaberemo **Assets → Import Package → Custom Package...** i odaberemo preuzetu verziju Vuforie. Kada nam se u Unityju prikaže dijaloški okvir **Import Unity Package**, odaberemo **Import**.

Nakon instalacije Vuforie u Unity, moramo još omogućiti njezino korištenje. To radimo tako da odemo na **Edit → Project Settings → Player → XR Settings** i odaberemo **Vuforia Augmented Reality Support**. Nakon toga još u **Other Settings** pod **Rendering**

→ **Graphics APIs**, ukoliko se kao opcija nalazi, treba ukloniti **Vulkan** i pod **Configuration → Target Architectures**, ukoliko se nudi kao opcija, označiti **x86**.

Kreiranje računa za Vuforiu

Da bismo kreirali bazu ili preuzeli neke dodatne alate potrebno je prijaviti se na službenu stranicu Vuforie, a ako se prijavljujemo prvi put potrebno je prvo kreirati račun. Obje opcije nalaze se u desnom gornjem kutu glavnog zbornika.



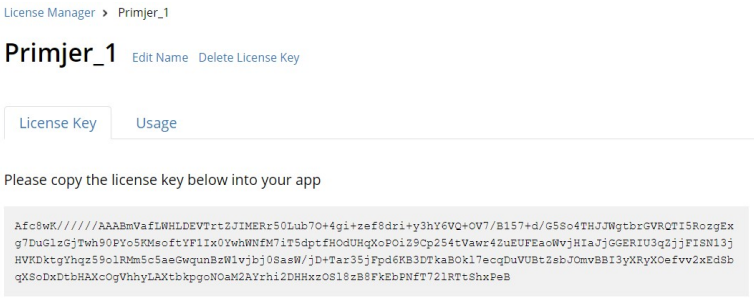
Slika 2.11: Kreiranje računa za Vuforiu

Nakon kreiranja računa i/ili prijave na stranicu u mogućnosti smo kreiranja baze ili preuzimanja određenih alata.

Kreiranje baze

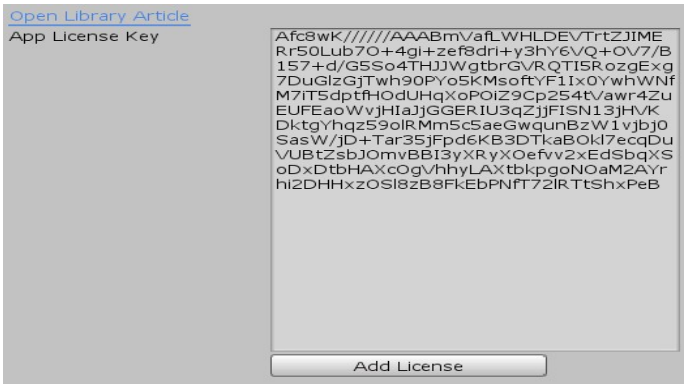
Nakon prijave na stranicu, u glavnom izborniku izaberemo **Develop** koji nas vodi na stranicu na kojoj možemo odabrati opcije **Licence Manager** ili **Target Manager**.

Licence Manager nam daje mogućnost kreiranja licenčnog ključa (*licence key*) koji nam je potreban za kreiranje aplikacije. Kreiramo ga tako da pritisnemo **Get Development Key** te nakon toga kreiramo novi licenčni ključ kojeg možemo pronaći pod danim imenom kada se vratimo na **Licence Manager**. Odaberemo naš licenčni ključ te ga kopiramo lijevim klikom miša.



Slika 2.12: Licenčni ključ u Licence Manager

Kopirani licenčni ključ dodajemo u Unity tako da odaberemo **Window** → **Vuforia Configuration** te na desnoj strani ekrana (**Inspector**) pod **Vuforia Configuration** → **Global** → **App Licence Key** u prozor zalijepimo kopirani licenčni ključ.

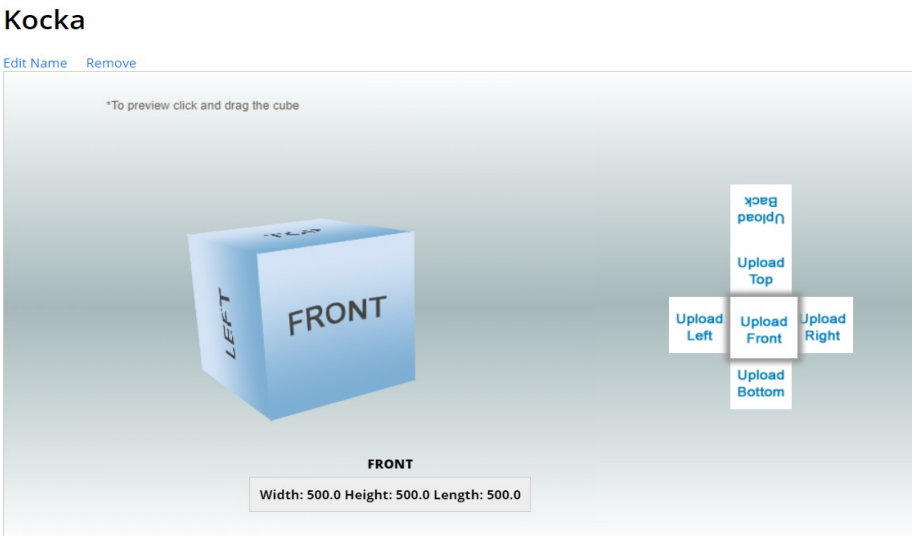


Slika 2.13: Licenčni ključ u Unityju

Također ispod kućice za dodavanje licenčnog ključa vidimo gumb **Add Licence**. Taj gumb nas odvodi direktno na stranicu Vuforie na dio **Licence Manager** preko kojeg možemo kreirati licenčni ključ na prethodno opisan način.

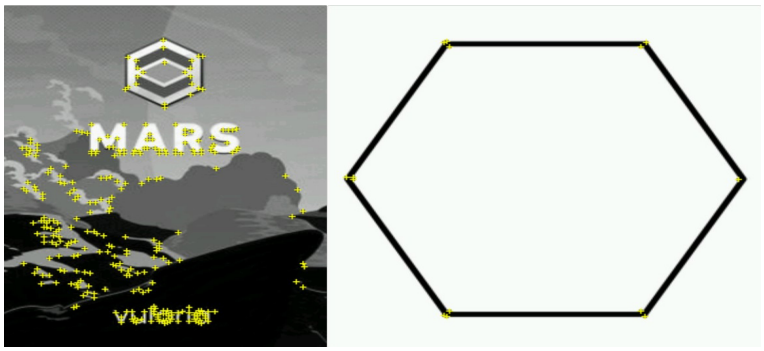
Target Manager nam daje mogućnost kreiranja baze podataka koja će nam služiti za pohranu naših ciljanih slika (*target images*). Nju kreiramo tako da najprije odemo na **Target Manager** te kreiramo novu bazu podataka pomoću opcije **Add Database**. Nakon toga nam se prikazuje okvir u koji upisujemo naziv naše baze i tip baze koji želimo. **Device** je baza koja se koristi izvanmrežno i pohranjena je u samoj aplikaciji, **Cloud** je baza koja se nalazi online, a **VuMarks** je baza kreiranih markera. Za potrebe jednostavnih aplikacija dovoljno je odabrati **Device**. Nakon kreiranja baze, pod **Target Manager** možemo vidjeti našu bazu.

Određeni ciljani objekt možemo dodati u bazu tako da najprije odaberemo bazu u koju ga dodajemo, zatim odaberemo **Add Target** nakon čega nam se prikaže okvir **Add Target** preko kojeg najprije odabiremo tip cilja, zatim upišemo dimenzije u pikselima i naziv cilja te ga dodajemo s **Add**. Ukoliko je naš cilj slika dodajemo je odmah (u .jpg ili .png formatu), a ukoliko se radi o 3D objektu, najprije ga dodamo u bazu te pokraj njega možemo vidjeti da piše **incomplete**, odnosno da moramo još definirati slike koje se nalaziti na površini objekta. To radimo tako da otvorimo dani objekt u bazi te dodamo slike na određena mjesta kako je navedeno u uputama.



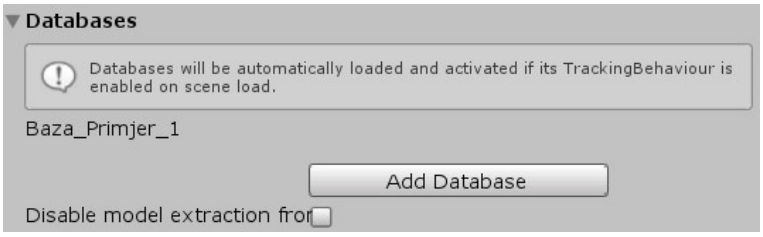
Slika 2.14: Kocka kao ciljani objekt u bazi

Kada otvorimo određenu bazu možemo vidjeti popis svih slika i objekata koje koristimo kao ciljeve te pokraj svake možemo vidjeti rejting, odnosno ocjenu koju je slika dobila na temelju broja i položaja značajnih točaka. Ako odaberemo neki cilj iz baze, s desne strane slike možemo vidjeti podatke o cilju, a ispod cilja možemo odabrati opciju **Show Features** koja nam prikazuje kako računalni vid vidi sliku, odnosno gdje se nalaze značajne točke.



Slika 2.15: Primjer dobre ciljane slike s mnogo značajnih točaka(lijevo) te loše ciljane slike s malo značajnih točaka (desno)

Nakon kreiranja baze i odabirom iste, bazu preuzimamo pritiskom na gumb **Download Database (All)** te kad se pokaže prozor **Select a development platform** odaberemo **Unity Editor**. Nakon što je gotovo preuzimanje, u Unity je dodajemo tako da se, u donji dio ekrana, u **Projects** → **Assets** ručno dovuče preuzeta baza iz mape u kojem se trenutno nalazi. Kada se pojavi okvir **Import Unity Package**, odaberemo **Import**. Ako smo sve dobro napravili, pod **Vuforia Configuration** → **Global** → **Databases** trebao bi biti naziv naše baze.



Slika 2.16: Naziv odabrane baze u Unityju

Kod opcije **Databases** vidimo gumb **Add Database**. Taj gumb nas odvodi direktno na stranicu Vuforie na dio **Target Manager** preko kojeg možemo kreirati bazu kako je prethodno opisano.

Unity Asset Store

Unity Asset Store je biblioteka koja sadrži razne 2D i 3D modele, šablone za korisnička sučelja, slike, animacije i teksture te mnogo drugih sredstava koje korisnici mogu iskoristiti unutar vlastitih programa. Svaki korisnik osim što može preuzeti neko sredstvo, može i objaviti svoje vlastito. U biblioteci se nalazi široka ponuda sadržaja, od besplatnih do onih koji se naplaćuju.U Unity Asset Store može se pristupiti direktno unutar nekih verzija

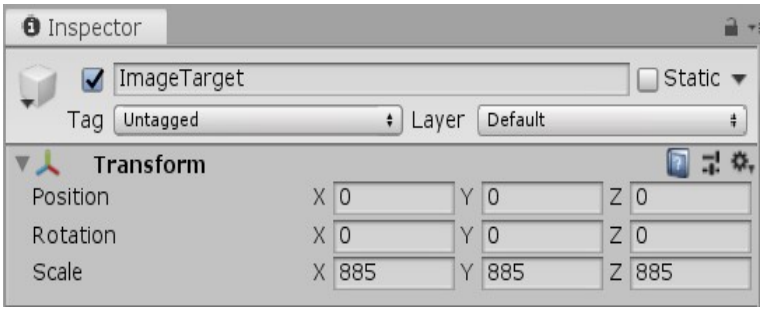
Unityja ili preko službene stranice https://assetstore.unity.com/ . Za preuzimanje ili objavu sadržaja potrebno je prijaviti se s Unity ID.
Preuzeti sadržaj možemo dovući u **Assets**, ubaciti pomoću **Import** te ga koristiti za razvoj naše AR aplikacije.

Postavljanje modela na ciljani objekt

Konstruirat ćemo rotirajuću kocku koja se pojavljuje na ekranu kada se prepozna određeni cilj. Na tom primjeru objasnit ćemo kako se u Unityju postavlja meta, na nju dodaje neki 3D model i kako se tom modelu pridodaje tekstura i, pomoću skripte, neka animacija.

U lijevom dijelu ekrana unutar Unityja nalazi se **Sample Scene**, odnosno okolina u kojoj gradimo naš model. Prema zadanim postavkama u **Sample Scene** možemo naći glavnu kameru (**Main Camera**) i usmjereno svjetlo (**Directional Light**). Za izradu proširene stvarnosti glavna kamera nam nije potrebna pa je možemo ukloniti s desnim klikom na **Main Camera** i odabirom opcije **Delete** te ponovo desnim klikom u **Sample Scene** i odabirom **Vuforia Engine → AR Camera** dodamo kameru za proširenu stvarnost. Prilikom prvog dodavanja moguće je pojavljivanje prozora **Import Vuforia Engine Assets** gdje pritisnemo **Import** kako bismo uvezli razne modele i teksture definirane u Vuforia Engineu.

Nakon što smo postavili AR kameru, ciljanu sliku dodajemo desnim klikom na **Vuforia Engine → Image** te nam se dani cilj prikaže u **#Scene** (srednji dio ekrana). **Image Target** (kao i bilo koju drugu komponentu) možemo preimenovati tako da desnim klikom odaberemo opciju **Rename** ili da tri puta kliknemo lijevim klikom na nju. Kada otvorimo naš **Image Target** uvijek će se prikazati slika koju je Unity izabrao kao zadanu sliku iz dane baze, a možemo je promijeniti u neku drugu iz iste baze tako da pritisnemo lijevim klikom **Image Target**, a zatim na desnoj strani ekrana (**Inspector**) pod **Image Target Behaviour (script)** i **Image Target** izaberemo sliku koju želimo koristiti kao naš cilj. Također, pod **Transform** možemo izmijeniti poziciju našeg cilja (**Position**), zarotirati ga (**Rotation**) ili mu promijeniti veličinu (**Scale**) upisivanjem željenih parametara. Ako to želimo napraviti ručno, u gornjem lijevom kutu nalazi se alatna traka s istim opcijama.



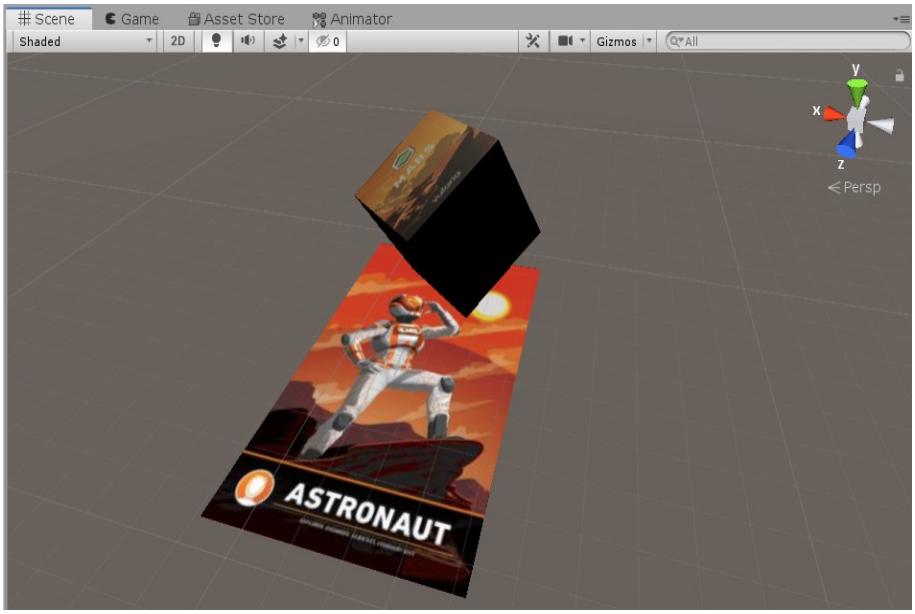
Slika 2.17: **Transform** unutar **Inspectora**

Slika 2.18: Alati za **Transform** pokraj glavnog izbornika

Kao zadano, naš **Image Target** okrenut je naopako, a ako ga želimo okrenuti „prema nama“ upišemo pod **Rotation** → **Y** parametar **180**.

Slika 2.19: **Image Target** postavljen u Unityju

Desnim klikom na naš **Image Target** te odabirom **3D Object** → **Cube**, na naš **Image Target** smo dodali model kocke koji će se pojaviti kada naša kamera prepozna dani cilj. Na isti način na koji smo transformirali cilj možemo transformirati i naš 3D model. Osim toga, našem 3D modelu možemo dodijeliti teksturu koja može biti neka zadana iz **Project** → **Assets** → **Vuforia** → **Materials** ili neka naša slika koju dodajemo tako da je ručno dovučemo u **Project** → **Assets** → **Resources**, a nakon toga je dodijelimo 3D modelu tako da tu istu sliku dovučemo na naš 3D model. U **#Scene** ćemo zatim moći vidjeti model kocke prekriven odabranom slikom.



Slika 2.20: Model dodan na **Image Target**

Za kraj našem 3D modelu kocke dodat ćemo rotaciju preko animacije. Za to ćemo morati definirati skriptu sa željenim kôdom pisanim u C#. Skriptu možemo definirati na dva načina. Prvi način jest da odemo u **Project** → **Assets** → **Scripts**, desnim klikom odaberemo **Create** → **C# Script** i damo joj određeno ime, npr. Rotate. Tu skriptu pridružujemo našem 3D modelu tako da je povučemo i stavimo na njega, a otvaramo dvostrukim lijevim klikom. Drugi način je da odaberemo naš 3D model i na desnom dijelu ekrana (**Inspector**) odaberemo opciju **Add Component** → **New Script** i pridružimo joj ime, a otvaramo je desnim klikom i odabirom **Edit Script**. Nakon što smo otvorili skriptu u nekom editoru (npr. Visual Studio), dodamo željeni kôd. Nakon dodavanja i debugiranja kôda spremimo promjene i naš novi kôd je pohranjen u dodanoj skripti.

Kôd za rotiranje kocke izgleda ovako:

```
1   using System.Collections;
2   using System.Collections.Generic;
3   using UnityEngine;
4
5   public class Rotate : MonoBehaviour
6   {
7       // Start is called before the first frame update
8       void Start()
9       {
10          //
11      }
12
13      // Update is called once per frame
14      void Update()
15      {
16          transform.Rotate(new Vector3(0, Time.deltaTime * 100, 0));
17      }
18  }
19
```

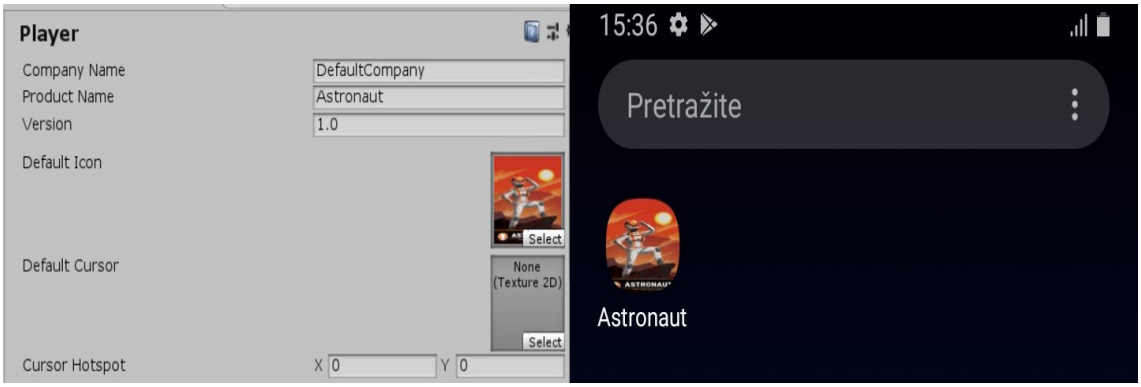
Slika 2.21: Skripta Rotate.cs

Kompajliranje i pokretanje (*Build and Run*)

Kada smo dodali sve komponente i završili s programiranjem naše aplikacije, potrebno je aplikaciju kompajlirati i pokrenuti na uređaju.

Sample Scene na kojem smo radili najprije spremimo preko **File** → **Save As...** i zatim mu dodijelimo ime i pritisnemo **Save**. Nakon toga odemo na **File** → **Build Settings...** te otvorimo **Player Settings...** . Pri tome nam se na vrhu nude opcije za promjenu naziva tvrtke, aplikacije i verzije. **Product Name** je naziv naše aplikacije koji će se prikazati na mobilnom uređaju, **Version** je broj trenutne verzije. Također možemo i pod **Default Icon** → **Select** dodati sliku koja će se kao ikona prikazati na mobilnom uređaju i predstavljati našu aplikaciju zajedno s nazivom. Slike koje se nude kao ikone su sve zadane Unityjeve slike, slike iz naše baze te sve slike koje smo dodali u **Assets**. Ukoliko želimo staviti neku sliku koja nije ponuđena, jednostavno je moramo dovući u **Assets** te će se ista pojaviti.

Ukoliko ne odaberemo niti jednu sliku za ikonu, kao ikona će se prikazati zadana Unityjeva slika njegovog loga.



Slika 2.22: Zadana ikona u **Player** (lijevo) i kao ikona aplikacije na mobilnom uređaju (desno)

Nakon podešavanja opcija u **Player** izađemo iz njega i vratimo se na **Build Settings**. Na vrhu dijaloškog okvira piše **Scenes In Build** te se u prostoru ispod nalazi popis svih **Sample Scenea** koje ćemo kompajlirati. **Sample Scene** dodajemo preko **Add Open Scenes**, a ako neku **Sample Scene** ne želimo kompajlirati uklonimo je tako da uklonimo kvačicu ispred njezina imena. Kada smo dodali sve željene **Sample Scene**, pritisnemo **Build** nakon čega nam se otvori prozor koji nam nudi opciju spremanja naše aplikacije u obliku .apk (za Android) ili .xcodeproj (za iOS) te prikazuje mjesto gdje će se ona spremiti i pri tome zahtijeva da damo ime našem paketu odnosno projektu. Nakon što pritisnemo **Save** započinje kompajliranje naše aplikacije.

Nakon što je kompajliranje završeno možemo našu aplikaciju prebaciti na mobilni uređaj. Pomoću USB kabela spojimo mobilni uređaj na računalo. Kod Androida otvorimo mapu u kojoj se nalazi naš .apk i ručno ga prebacimo u mapu koja predstavlja mobilni uređaj. Zatim na mobilnom uređaju pronađemo taj .apk te ga instaliramo. Kod iOS-a je potrebno otvoriti projekt u Xcode i podesiti **Signing** nakon čega u Xcode treba pritisnuti **Build** te se sve kompajlira i prebacuje na mobilni uređaj.

Korištenje aplikacije

Kako što je već ranije opisano, virtualni modeli koji se pojavljuju u stvarnoj okolini pozicioniraju se pomoću cilja koji mogu biti slike ili trodimenzionalni modeli. Da bi naša AR aplikacija uspješno mogla postaviti virtualni model u stvarnost potrebno je imati primjerak slike ili modela koji naša aplikacija koristi kao cilj. Kako skaliranje ne bi stvaralo

problem, najbolje je imati sliku ili model dimenzija kakve su zadane u našoj bazi s ciljevima. Računalni vid slike i modele vidi u sivim tonovima pa će aplikacija jednako raditi koristimo li u stvarnosti slike i modele koji su printani u boji ili crno-bijelo.

Slika 2.23: Rezultat pokretanja aplikacije s obojanom ciljanom slikom (lijevo) i rezultat pokretanja aplikacije s crno-bijelom ciljanom slikom (desno)

Dakle, nakon instalacije i pokretanja aplikacije na mobilnom uređaju, potrebno je kamerom pronaći cilj u stvarnoj okolini te kada je on detektiran aplikacija dalje dodaje definirane modele i funkcionalnosti.

Poglavlje 3

Izrada aplikacije za proširenu stvarnost

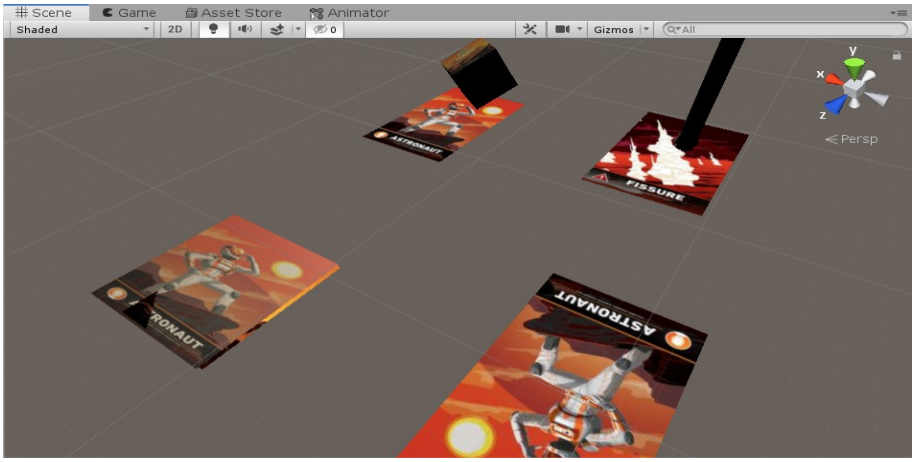
Prethodno je opisan postupak instalacije potrebnih programa, podešavanje određenih postavki te kompajliranje i pokretanje. Također je opisan postupak kojim se u Unityju, koristeći alate iz Vuforie, kreiraju osnovne potrebne komponente poput praćene slike i modela te jednostavne mogućnosti za modeliranje postavljenog modela kao što su tekstura i animacija. Nadalje opisujemo neke mogućnosti koje Vuforia nudi, njihovu izradu i opis problema i rješenja prilikom izrade.

3.1 Ciljevi (*Targets*)

Kao što je već ranije spomenuto, za postavljanje nekog virtualnog objekta u danu okolinu potrebno je najprije detektirati određeni objekt koji nam služi kao cilj. Moguće je koristiti slike, cilindre, kocke i posebne 3D modele. Aplikacija može imati samo jedan ciljani objekt ili više njih te može odjednom pratiti samo jedan ili njih nekoliko.

Dodavanje ciljeva

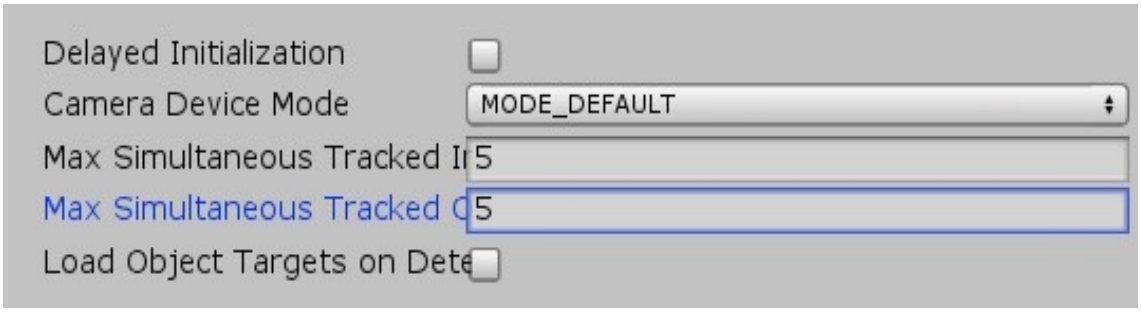
U točki **2.2**, u odjeljku **Postavljanje modela na ciljani objekt**, opisan je postupak kojim iz baze dodajemo naše ciljeve. Ukoliko želimo koristiti više ciljeva ponovimo postupak, ali kao novi cilj izaberemo drugu sliku ili model iz baze. Svi ciljevi automatski se postavljaju u ishodište koordinatnog sustava pa ako ih dodamo odjednom prekrivat će jedan drugoga. Pozicija AR kamere u Unityju ne igra nikakvu ulogu tako da problem preklapanja ciljeva rješavamo pomicanjem pojedinačnih ciljeva negdje drugdje unutar koordinatnog sustava, a kada pokrenemo aplikaciju svi ciljevi će se moći prepoznati. U Unityju to možemo napraviti pomoću opcije **Move** (alatna traka, ikona s rukom).



Slika 3.1: Više ciljanih slika u Unityju

Praćenje više ciljeva odjednom

Po zadanim postavkama, nakon što kompajliramo i pokrenemo aplikaciju, ona će prepoznati samo jedan cilj u realnom vremenu i postaviti virtualni objekt namijenjen njemu. Čak i ako je više ciljeva u kadru, prepoznat će se samo jedan kojeg kamera stavi u fokus.
Ukoliko želimo da se odjednom prepozna više ciljeva i da oni istovremeno prikažu odgovarajući virtualni objekt, unutar Unityja koristimo opciju za višestruko praćenje (*multitracking*). Podešavamo je tako da lijevim klikom u **Sample Scene** kliknemo na **AR Camera**, zatim u **Inspector** na dnu pritisnemo gumb **Open Vuforia Engine Configuration** i pod **Global** u kućicu pokraj **Max Simultaneous Tracked Images** i **Max Simultaneous Tracked Objects** upišemo željeni broj praćenih slika i objekata.



Slika 3.2: Podešavanje opcija za višestruko praćenje

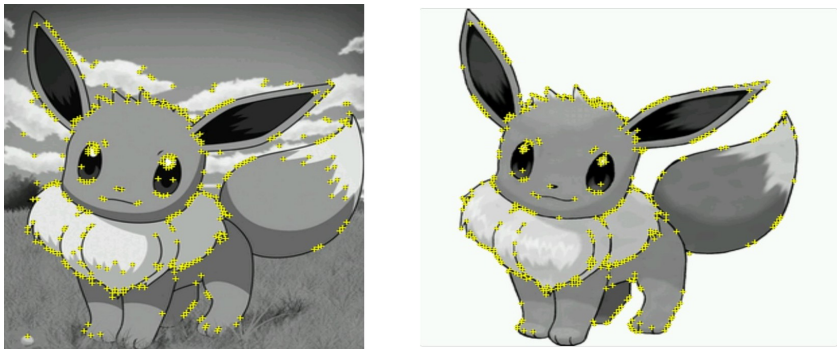
Slika 3.3: Prepoznavanje više ciljanih slika odjednom (slike likova) i prikaz virtualnih slika (slika s tekstom)

Izbor cilja

Za brzo i pouzdano praćenje potrebno je izabrati ciljeve sa što više značajnih točaka, ali u obzir se treba uzeti i pozicija tih točaka i kontrast slike ili modela.
Slike koje imaju sličan raspored značajnih točaka, npr. različite slike s tekstom na istom mjestu, računalni vid može teže razlikovati. Također slike koje na sebi imaju slične oblike, npr. jednobojni poligoni, kao značajne točke imaju samo vrhove pa sličan raspored na različitim praćenim objektima neće biti toliko jedinstven, a onda niti lako prepoznatljiv.

- **Primjer 1:**

Slika 3.4: Isti lik sa nekom nacrtanom pozadinom (lijevo) i bijelom pozadinom (desno)



Slika 3.5: Značajne točke lika s nacrtanom pozadinom (lijevo) i bijelom pozadinom (desno)

Gledano iz ljudske perspektive, obje slike sadrže isti lik i kada bi nas se pitalo da ga prepoznamo na slici i imenujemo, za obje slike rekli bismo isto. Iako se na obje slike nalazi isti lik u istoj pozi i istoj boji, značajnu ulogu ima pozadina prve slike na kojoj se prepoznaje još nekoliko značajnih točaka. Zbog tih dodatnih značajnih točaka u elementima pozadine, računalni vid tretira te dvije slike kao dvije različite slike i nije u mogućnosti razlikovati glavni lik i elemente pozadine. Razlog više jest taj da ljudi kroz iskustvo nauče kako se na dvodimenzionalnim slikama postiže perspektiva i stavlja pojedini lik u fokus, a generalno računala još nisu postigla tu razinu.

Kako računalni vid ciljani objekt vidi u nijansama sive, ono što ljudskom oku zbog percepcije boje djeluje očito različito, računalnom vidu djeluje slično.

- **Primjer 2:**

Kada bismo pogledali ove dvije slike, zbog različitih boja odmah bismo odredili da se radi o dvije različite slike. No, kada se obje slike prikažu u nijansama sive one djeluju slično.

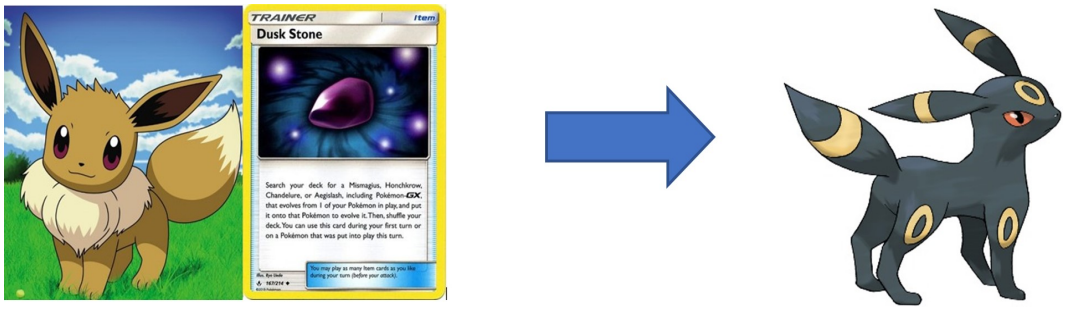
Slika 3.6: Dvije različite slike u boji

Slika 3.7: Dvije različite slike u sivim nijansama

Spajanjem dviju slika u jednu dobivamo novu sliku. Prethodno smo opisali da računalni vid naizgled istu sliku prepoznaje u potpunosti drugačije zbog dodatnih značajnih točaka. Tada bismo očekivali da će različite kombinacije slika dati različite slike s lako prepoznatljivom kombinacijom značajnih točaka, ali u praksi to nije tako jednostavno.

- **Primjer 3:**

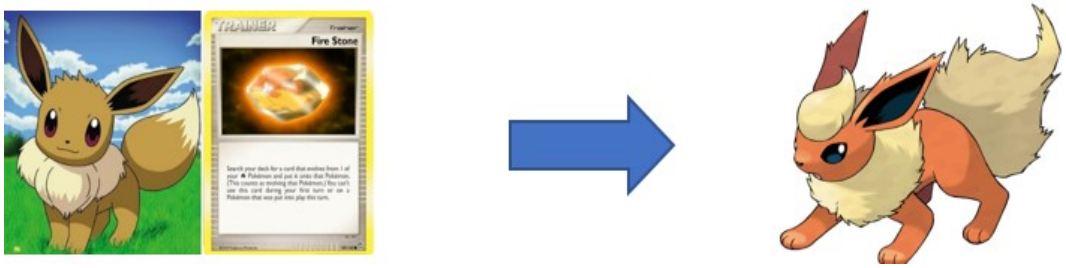
Slika 3.8: Prva cljana slika kao kombinacija dviju različitih slika i očekivani rezultat



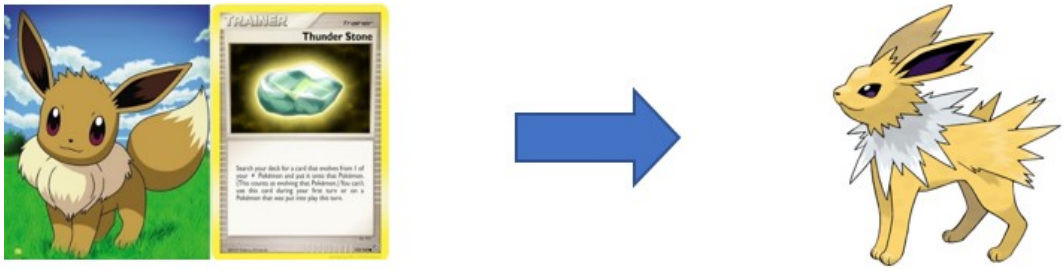
Slika 3.9: Druga cljana slika kao kombinacija dviju različitih slika i očekivani rezultat

Na danom primjeru vidimo kako ista slika glavnog lika u kombinaciji s dvije različite slike daje dva različita rezultata. Zbog korištenja dovoljno različitih drugih slika (druga boja obruba, različit položaj teksta) dobili smo željeni rezultat, ali kada su te dvije slike slične računalni vid neće lako prepoznati razliku te će dati isti rezultat u oba slučaja ili uopće neće dati nikakav rezultat.

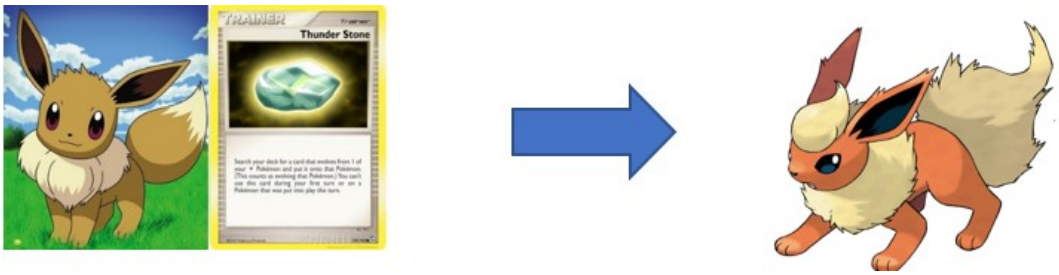
- **Primjer 4:**



Slika 3.10: Treća cljana slika kao kombinacija dviju različitih slika i očekivani rezultat



Slika 3.11: Četvrta cljana slika kao kombinacija dviju različitih slika i očekivani rezultat



Slika 3.12: Četvrta cljana slika kao kombinacija dviju različitih slika i dobiveni rezultat

Na ovom primjeru vidimo kako, iako koristimo različite slike možemo dobiti isti rezultat. Jedan od razloga kojeg smo već naveli su slične boje u sivom spektru. Dakako, kao ljudi, mogli bismo prepoznati razliku po obliku kamena i različitom tekstu, ali kao računalo i te dvije komponente su nam slične jer se radi o sličnim poligonima, a tekst se ne čita nego samo odredi značajna točka kao kontrast dviju boja (teksta i pozadine). Također, položaj teksta je podjednak na obje slike što stvara dodatni problem.

Kao što je navedeno ranije, zbog lošijeg fokusa i osvjetljenja moguće je slabije očitanje praćene slike što može rezultirati time da aplikacije uopće ne da nikakav rezultat. U danim primjerima koristi se kombinacija dviju slika, a prva slika je uvijek ista. Ako dođe do slabijeg očitanja druge slike, većina fokusa se prebacuje na prvu te se kao ciljana slika prepoznaje neka slika iz baze čiji položaj značajnih točaka je najbliži očitanju. To može rezultirati prikazom slike namijenjene toj ciljanoj slici, a ne onoj koju gledamo kroz kameru. Također se kao rezultat može pojaviti i zadnja slika koju je aplikacija pokazala jer joj je ostala pohranjena u memoriji. Oba slučaja mogu rezultirati prikazom pogrešne slike. Problem možemo riješiti tako da kao druge slike koristimo što različitije slike s obzirom na kontrast i položaj značajnih točaka.

Još jedan problem koji se zna dogoditi je pamćenje sidra, odnosno aplikacija zapamti lokaciju usidravanja virtualnog modela pa taj model ostane na toj lokaciji i nakon što uklonimo praćenu sliku ili se pojavi ponovo iako smo postavili različitu praćenu sliku. Dio tog problema nastaje zbog sporije reakcije fokusa kamere naspram promjene stvarne okoline, npr. prebrzo smo pomaknuli praćenu sliku iz kadra, a dio je do samog računalnog vida koji koristi aplikacija, ali na tome se trenutno radi te se u kasnijim verzijama Vuforie može očekivati bolji rezultat.

3.2 2D modeli

3.2.1 Slike

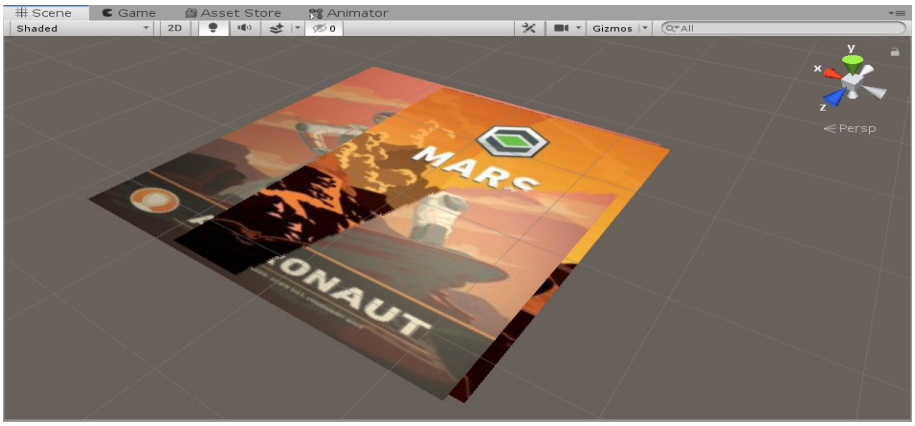
Jedna mogućnost koja se nudi jest prikaz određene slike, ili više njih, kada se detektira cilj. Tada se nova slika pokaže kao virtualni objekt te se može nalaziti u istoj ravnini kao i cilj ili u nekoj drugoj, željenoj ravnini. Također se može prikazati i više slika odjednom.

Postavljanje slika

Nakon što smo dodali **Image Target**, desnim klikom na njega odabiremo **3D Object** → **Quad**. **Quad** je pravokutnik koji se po zadanim postavkama nalazi u XY ravnini i koristi se kada željeni objekt želimo prikazati kao jednostavan zaslon s nekim sadržajem, npr. slika ili video. Dakako, dani **Quad** možemo transformirati po želji u odnosu na **Image Target**, odnosno postaviti ga u istoj ravnini kao **Image Target** ili oko njega, zarotirati ga, povećati, smanjiti itd.

Željenu sliku postavljamo tako da je dovučemo na napravljeni **Quad** te je u **#Scene** možemo vidjeti kao materijal unutar našeg **Quada**. Ukoliko želimo da naša slika bude u istoj ravnini kao i **Image Target** to možemo napraviti tako da promijenimo parametar pod **Rotation** → **X** u **90** zadanog **Quada** u kojem je naša slika. Ako želimo da se umjesto

praćene slike pojavi virtualna slika kada pokrenemo aplikaciju, nakon što smo sliku stavili u istu ravninu kao i praćenu sliku, zadani **Quad** potrebno je skalirati tako da prekriva praćenu sliku. Ako želimo da se prilikom pokretanja aplikacije uopće ne vidi praćena slika, vrijednost od **Position** → **Y** zadanog **Quada** mora biti postavljena na **0**, inače za vrijednosti veće od 0, slika će pri pokretanju aplikacije biti iznad praćene slike, a praćena slika će time biti vidljiva. U **#Scene** je vidljivo da su slike u istoj ravnini jedna preko druge kada pomičemo **Image Target** (a s njime i njemu dodijeljen **Quad**), slika koja se nalazi u **Quadu** na trenutke nestaje pa se opet pojavljuje.



Slika 3.13: Ciljana slika i virtualna slika u istoj ravnini unutar Unityja

Slika 3.14: Ciljana slika (lijevo) i virtualna slika (desno)

Ako želimo dodati više slika, svaku sliku stavimo u poseban **Quad** koji je dodijeljen pod **Image Target**.

Slika 3.15: Ciljana slika (lijevo) i više virtualnih slika (desno)

Problemi i rješenja

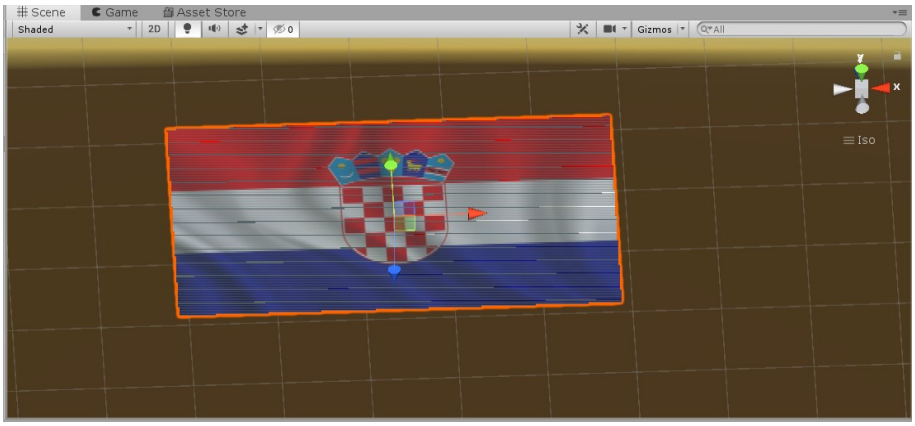
Poželjno je kod korištenja više slika ne stavljati ih jednu preko druge u istu ravninu jer se pri korištenju aplikacije te slike stalno izmjenjuju pa je onemogućen čisti prikaz svake slike. Također, poželjno je dani **Quad** sa slikom postaviti pod neki kut koji nije okomit na praćenu sliku jer se okomiti objekti najslabije očitavaju pa može doći do prekida prikazivanja virtualnog objekta.

3.2.2 Video

Osim slika, kao dvodimenzionalni virtualni objekt možemo postaviti i video, a unutar jedne aplikacije možemo koristiti i više njih. Isto kao i kod slika, odjednom možemo prikazati jedan ili više videa odjednom.

Postavljanje videa

Kao i slike, videe postavljamo unutar **Quada** koji nam služi kao zaslon na kojem se video reproducira kada aplikacija detektira praćeni objekt. **Quad** s videom postavljamo oko ili preko **Image Targeta** na isti način kao što je prethodno opisano te ga skaliramo po želji. Ako želimo da **Quad** s videom u potpunosti prekrije naš praćeni objekt kada ga se detektira, postavimo iste parametre kao i kod slika. Ako smo dobro uradili u **#Scene** ćemo pri pomicanju zadanog **Image Targeta** moći vidjeti linije preko njega.



Slika 3.16: Ciljana slika i virtualni video u istoj ravnini unutar Unityja

Video dodajemo tako da kliknemo lijevom tipkom miša na određeni **Quad**, zatim u **Inspector** pritisnemo **Add Component** i upišemo **Video Player** te kada nam se prikaže odaberemo ga. Tada ćemo u **Inspectoru** vidjeti da je našem **Quadu** dodana komponenta **Video Player**.

Video možemo dodati na dva načina, preko URL-a ili preko video klipa.

Ako koristimo video klip onda ga prethodno moramo dovući u **Assets** i on mora biti formata .mp4. Nakon što smo ga dovukli, možemo kliknuti na njega lijevim klikom te će se na desnoj strani pojaviti **Inspector** u kojem možemo na dnu vidjeti naš video te ga pokrenuti da provjerimo ispravnost i slike i zvuka.

Po zadanim postavkama **Video Player** → **Source** je postavljen na **Video Clip**. Ispod **Source** možemo vidjeti opciju **Video Clip** koja je zadana kao **None (Video Clip)**. U tu kućicu postavljamo naš video tako da ga ručno dovučemo iz **Assets** u kućicu, nakon čega ćemo vidjeti naziv našeg videa unutar nje.
Ako koristimo URL, tada **Source** promijenimo u **URL**, te ispod njega u opciji **URL** stavimo URL našeg videa.

Slika 3.17: Postavljanje videa preko **Vide Clipa**

Slika 3.18: Postavljanje videa preko **URL-a**

Prednost korištenja videa preko URL-a je manja veličina konačne aplikacije, ali nedostatak je obavezna povezanost s internetom kako bi se moglo pristupiti videu. Prednost korištenja video klipova je neovisnost o internetu i pouzdanost svih videa, ali nedostatak je da previše videa može uzrokovati preveliku veličinu konačne aplikacije.

Kod **Inspector** → **Video Player** možemo naći još neke dodatne postavke. **Play On Awake** je postavka koja omogućava našem videu da se pokrene čim se pokrene i aplikacija, **Wait For First Frame** je postavka koja pokreće video tek nakon što se učita prvi kadar, **Loop** ponavlja video nakon završetka, **Skip On Drop** omogućava zakašnjelim kadrovima da se preskoče kako bi se sinkronizirali video i zvuk. Također su tu i opcije za postavku brzine videa (**Playback Speed**), glasnoće (**Volume**) ili utišavanja zvuka (**Mute**).

Kod korištenja AR aplikacija, najčešće želimo da se naš video pokrene tek kad se prepozna dani praćeni objekt. Zato je potrebno onemogućiti **Play On Awake**. Također je potrebno podesiti u skripti pokretanje i pauziranje videa kada se praćeni objekt detektira, odnosno izgubi iz fokusa. To radimo dodavanjem sljedećeg kôda u skriptu **DefaultTrackableEventHandler**:

```
20  using UnityEngine;
21  using Vuforia;
22
23  public class DefaultTrackableEventHandler : MonoBehaviour, ITrackableEventHandler
24  {
25      PROTECTED_MEMBER_VARIABLES
32
33      UNITY_MONOBEHAVIOUR_METHODS
49
50      PUBLIC_METHODS
86
87      #region PROTECTED_METHODS
88
89      protected virtual void OnTrackingFound()
90      {
91          var rendererComponents = GetComponentsInChildren<Renderer>(true);
92          var colliderComponents = GetComponentsInChildren<Collider>(true);
93          var canvasComponents = GetComponentsInChildren<Canvas>(true);
94
95          // Enable rendering:
96          foreach (var component in rendererComponents)
97              component.enabled = true;
98
99          // Enable colliders:
100         foreach (var component in colliderComponents)
101             component.enabled = true;
102
103         // Enable canvas':
104         foreach (var component in canvasComponents)
105             component.enabled = true;
106
107         // dodati ovaj dio koda; služi za pokretanje videa nakon što se detektira cilj:
108
109         if(mTrackableBehaviour.gameObject.GetComponentInChildren<VideoSource>()!=null)
110         {
111             mTrackableBehaviour.gameObject.GetComponentInChildren<VideoSource>().Play();
112         }
113     }
```

Slika 3.19: DefaultTrackableEventHandler.cs (prvi dio)

```
115
116     protected virtual void OnTrackingLost()
117     {
118         var rendererComponents = GetComponentsInChildren<Renderer>(true);
119         var colliderComponents = GetComponentsInChildren<Collider>(true);
120         var canvasComponents = GetComponentsInChildren<Canvas>(true);
121
122         // Disable rendering:
123         foreach (var component in rendererComponents)
124             component.enabled = false;
125
126         // Disable colliders:
127         foreach (var component in colliderComponents)
128             component.enabled = false;
129
130         // Disable canvas':
131         foreach (var component in canvasComponents)
132             component.enabled = false;
133
134         //dodati ovaj dio koda; služi za pauziranje videa nakon što se izgubi cilj:
135
136         if (mTrackableBehaviour.gameObject.GetComponentInChildren<VideoSource>() != null)
137         {
138             mTrackableBehaviour.gameObject.GetComponentInChildren<VideoSource>().Pause();
139         }
140     }
```

Slika 3.20: DefaultTrackableEventHandler.cs (drugi dio)

Nakon kodiranja i spremanja skripte, pridodajemo je **Image Targetu** s **Quadom** u kojem se nalazi odabrani video. Istu skriptu dodajemo svim ostalim **Image Targetima** ukoliko imamo više videa s više različitih praćenih slika.

Prilikom pokretanja aplikacije i detekcije praćenog objekta pokreće se naš video. Kada prestane praćenje video se prekida i pauzira te ponovnom detekcijom istog praćenog objekta video se nastavlja tamo gdje je pauziran nakon zadnjeg pokretanja. U slučaju više videa, ista stvar će se dogoditi za svaki video.
Ako želimo da se naš video ponovno pokrene jednom kad završi, omogućimo opciju **Loop**, inače jednom kad video završi prikazat će se samo zadnji kadar videa.

Slika 3.21: Ciljana slika (desno) i video (lijevo)

Problemi i rješenja

Iako je moguće praćenje više videa istovremeno, bolja opcija je prikazivanje samo jednog u realnom vremenu. Ukoliko se više videa pokrene odjednom, zvuk svakog od njih se pokreće te se dobiva prevelika i neugodna buka. Ako se ipak odlučimo za istovremeno pokretanje više videa dobra opcija je podešavanje volumena, a ako želimo samo video prikaz bez zvuka, možemo utišati videe pomoću opcije **Mute**.

3.3 3D modeli

Trodimenzionalni modeli najčešći su virtualni objekti korišteni u aplikacijama za proširenu stvarnost. Zanimljivi su zbog toga što se nakon postavljana na dani cilj mogu vidjeti iz svakog kuta, sjenčanje i sjene daju osjećaj kao da pripadaju danoj okolini, a moguće je i animirati dani model što daje još jedan dojam stvarnosti. Također je moguće i stupiti u interakciju s modelom što ga čini još zanimljivijim. Vuforia nudi nekoliko standardnih 3D modela, a u Unity Asset Storeu možemo pronaći mnogo različitih modela koje možemo preuzeti, uvesti i koristiti u našoj aplikaciji. Također pomoću nekih programa za izradu 3D modela (npr. Blender) i sami možemo izraditi vlastite modele koje kasnije možemo koristiti u Unityju.

Postavljanje 3D modela na ciljanu poziciju

Nakon što smo dodali **Image Target**, pridružujemo mu jedan ili više modela. Ako koristimo osnovne modele poput kocke, cilindra, kapsule itd. njih dodajemo desnim klikom na **Image Target**, zatim pod **3D Models** odaberemo željeni model. Nakon dodavanja taj model možemo transformirati po želji. Ukoliko želimo više modela ponovimo gornji postupak.
Ako koristimo neki drugi model koji nije standardni, najprije ga moramo dovući u **Assets**, a na **Image Target** ga postavljamo tako da ga ručno dovučemo na **Image Target** u **Sample Scene**. Neki 3D modeli već imaju u potpunosti napravljenu teksturu, a neki dolaze kao „goli“ modeli s materijalima koje ručno moramo postaviti, a to radimo tako da odaberemo željeni materijal i dodamo ga komponenti našeg modela (npr. tijelu, očima itd.) preko hijerarhije u **Sample Scene** ili direktno na model u **#Scene**.

Slika 3.22: Model bez materijala

Slika 3.23: Model nakon dodavanja materijala

Nakon što smo postavili i oblikovali dani model, nakon kompajliranja i pokretanja aplikacije te detekcije željenog cilja prikazat će se naš model.

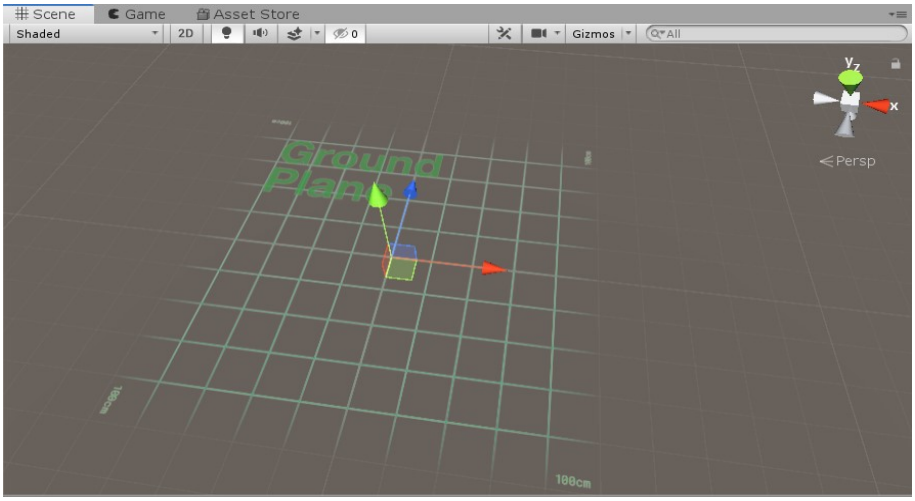
Slika 3.24: 3D model na ciljanoj slici

Postavljanje 3D modela u prostor

Vuforia nudi opciju postavljanja nekog virtualnog 3D modela u prostor na određenu površinu u okolini i postavljanja virtualnog 3D modela koji lebdi.

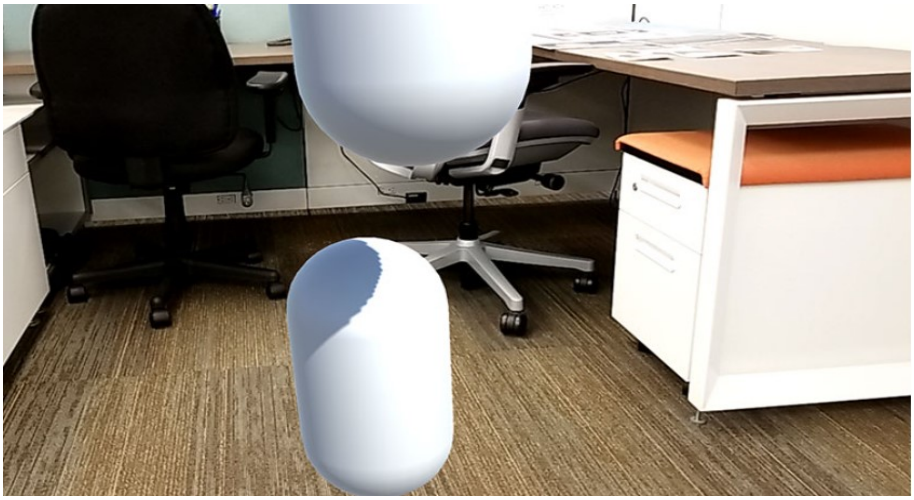
Prva opcija je tzv. **Ground Plane**. Nju koristimo tako da, nakon što smo postavili sve opcije za Vuforiu u Unityju i u **Sample Scene** dodali **AR Camera**, u **Sample Scene** desnim klikom biramo **Vuforia Engine → Ground Plane → Ground Plane Stage**, a zatim **Vuforia Engine → Ground Plane → Plane Finder**. Nakon dodavanja komponenti u **#Scene**

vidimo ravninu na kojoj piše Ground Plane. Kada lijevim klikom pritisnemo **Ground Plane Stage** u **Inspector**, **Anchor Behaviour (Script)** → **Stage Type** treba biti postavljen na **Plane**.



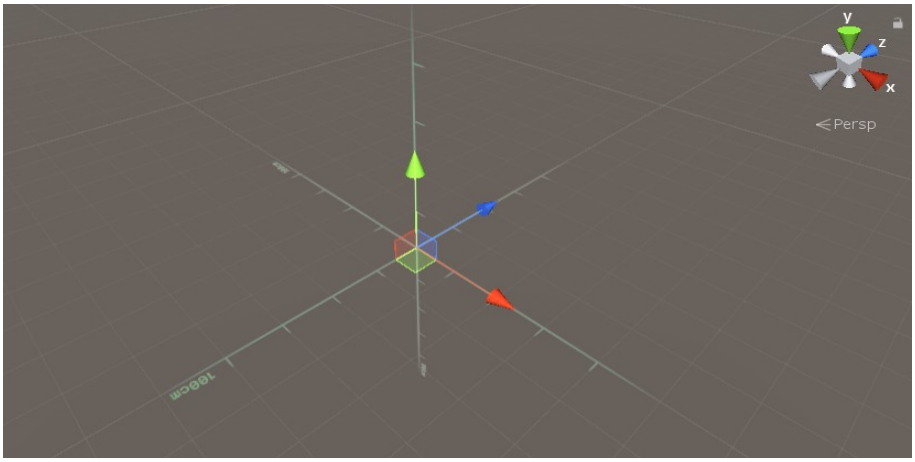
Slika 3.25: **Groun Plane** u Unityju

Željeni model dodajemo na **Ground Plane** tako da ga ručno dovučemo na **Ground Plane Stage** u **Sample Scenes** te transformiramo po želji. Taj model će se prikazati na pronađenoj površini nakon pokretanja aplikacije i detektiranja površine.
Nakon dodavanja modela, u **Sample Scene** lijevim klikom miša na **Plane Finder** otvorimo **Inspector** te pod **Content Positioning Behaviour (Script)** → **Anchor Stage** iz **Sample Scene** dovučemo **Ground Plane Stage** u kućicu pokraj natpisa **Anchor Stage**. Ako smo dobro uradili, u kućici bi trebalo pisati **Ground Plane Stage (AnchorBehaviour)**.
Ispod **Anchor Stage** imamo opciju **Duplicate Stage**. Ako je ona uključena tada naša aplikacija nakon što detektira površinu i stavi prvi virtualni objekt na nju, ponovo detektira istu površinu (duplicira je) te ponovo možemo dodati drugi virtualni objekt. Tada ćemo na našoj površini imati dva ista virtualna objekta. Postupak možemo ponoviti za više virtualnih objekata. Ako opcija nije uključena tada će se prilikom ponovne detekcije površine, na novom dijelu površine prikazati novi virtualni objekt, a stari će nestati, odnosno na površini će se uvijek nalaziti samo jedan virtualni objekt.
Nakon što smo postavili sve komponente i podesili opcije, naš **Sample Scene** možemo spremiti, kompajlirati i pokrenuti na mobilnom uređaju. Kada pokrenemo aplikaciju, kameru usmjerimo prema nekoj površini. Ako je aplikacija detektirala neku površinu, na njoj će se pokazati oznaka gdje će se postaviti virtualni objekt, a korisnik postavlja objekt tako da dodirne zaslon svojeg uređaja.



Slika 3.26: **Groud Plane** s dupliciranim virtualnim objektima u prostoru

Druga opcije je tzv. **Mid Air**.
Nakon postavljanja svih opcija za pokretanje Vuforie u **Sample Scene** desnim klikom miša najprije dodamo **AR Camera**, a zatim, također desnim klikom miša, dodamo **Vuforia Engine → Mid Air → Mid Air Stage** te **Vuforia Engine → Mid Air → Mid Air Positioner**. Nakon dodavanja komponenti u **#Scene** vidimo 3D-koordinatni sustav. Kada lijevim klikom pritisnemo **Mid Air Stage** u **Inspector**, **Anchor Behaviour (Script) → Stage Type** treba biti postavljen na **Mid Air**.



Slika 3.27: **Mid Air Stage** u Unityju

Model kojeg želimo koristiti prethodno dovedemo u **Assets**, a nakon toga ga od tamo dovučemo ručno na **Mid Air Stage** te transformiramo po želji. Nakon dodavanja modela, u **Sample Scene** lijevim klikom miša na **Mid Air Positioner** otvorimo **Inspector** te pod **Content Positioning Behaviour (Script)** → **Anchor Stage** iz **Sample Scene** dovučemo **Mid Air Stage** u kućicu pokraj natpisa **Anchor Stage**. Ako smo dobro uradili, u kućici bi trebalo pisati **Mid Air Stage (AnchorBehaviour)**. Kao i kod **Ground Plane** i ovdje možemo postaviti ili ukloniti opciju **Duplicate Stage**.
Nakon spremanja, kompajliranja i pokretanja, kad otvorimo aplikaciju na zaslonu uređaja prikazat će nam se zelena kugla. Ona označava mjesto na kojem ili oko kojeg će se prikazati naš virtualni objekt.

Slika 3.28: **Mid Air** prije postavljanja modela

Slika 3.29: **Mid Air** nakon postavljanja modela u prostor

Problemi i rješenja

Za što bolji i vjerodostojniji doživljaj virtualnog modela u prostoru potrebno je voditi brigu o modelu kojeg koristimo. Od modela koje postavljamo u zraku očekujemo da se ponašaju kao da ostaju u zraku, da lebde ili lete. Mnogi modeli iz Unity Asset Storea već imaju podešene animacije koje daju taj efekt, a ako koristimo obične modele, poželjno je odabrati one koji u stvarnosti mogu lebdjeti u zraku, a ne neke modele čvrste građe koji će izgledati neprirodno.

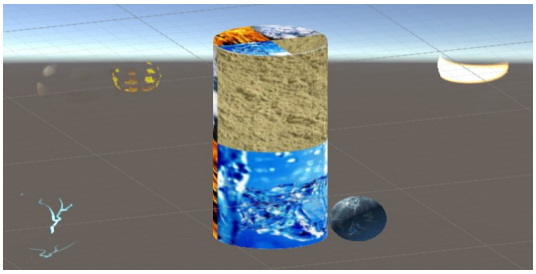
Sama izrada **Ground Plane** i **Mid Air** aplikacija u Unityju nije teška. Međutim, pri pokretanju aplikacije može se dogoditi da aplikacija ne detektira nikakve površine i time ne postavlja nikakav virtualni objekt. Ponekad je razlog da su površine previše neprepoznatljive, odnosno ne mogu se detektirati rubovi. Zato je najbolje koristiti neke površine s dobrim kontrastom.
Još jedan razlog zbog kojeg je moguć neispravan rad detekcije jest i taj da je greška u samom uređaju koji koristi danu aplikaciju, kao i greška u samom sustavu Unityja ili Vuforie, iako nam se sve kompajlira bez greške.

3D modeli kao ciljani objekti

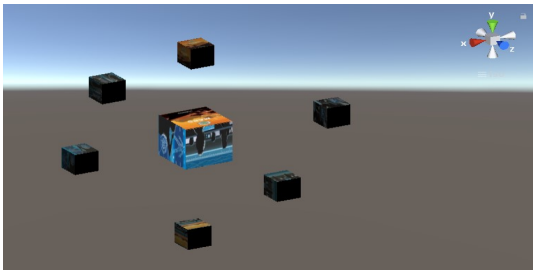
Kao što je navedeno u točki **2.2**, odjeljak **Kreiranje baze**, osim slike kao ciljanog objekta, mogu se koristiti i standardni 3D modeli poput kocke ili cilindra, ali i posebni modeli ispisani 3D printerom. Ovdje ćemo opisati samo standardne modele.

Ako želimo dodati takav objekt kao ciljani to radimo tako da u Sample Scene desnim klikom miša dodamo **Vuforia Engine → Cylindrical Image** ako želimo cilindar, a **Multi Image** za kocku. Oba ciljana objekta moraju se nalaziti u prethodno postavljenoj bazi.

Nakon što smo ih postavili, željene virtualne objekte dodajemo na isti način kao i kod ciljanih slika. Jedina razlika je ta da ih možemo postaviti posvuda oko našeg ciljanog objekta i sami virtualni objekti će se prikazivati u skladu s tim kada se pokrene aplikacija.



Slika 3.30: Cilindar kao 3D ciljani objekt



Slika 3.31: Kocka kao 3D ciljani objekt

Kod cilindra, odnosno **CylinderTargeta**, prate se sve slike postavljene na tom cilindru i kada se one detektiraju pojavljuju se virtualni objekti, a moguće je i kretati se oko njega, pri čemu ćemo u svakom kadru vidjeti druge virtualne objekte, ovisno o dijelu cilindra kojeg očitavamo.
Kod kocke, odnosno **MultiTargeta**, prati se svaka strana kocke posebno i ovisno o strani na zaslon se prikazuje određeni virtualni objekt.

Slika 3.32: Cilindar kao ciljani objekt i virtualni objekti

Slika 3.33: Kocka kao ciljani objekt i virtualni objekti

3.4 Interakcija s virtualnim objektom

Poželjno je da AR aplikacija daje korisniku mogućnost interakcije s virtualnim objektom. Osim što neki virtualni model možemo postaviti u stvarnu okolinu, možemo od njega zatražiti da nešto napravi kada mu to naredimo. Te naredbe mogu se odvijati preko virtualnih gumba ili zaslona, odnosno interaktivnog korisničkog sučelja.

3.4.1 Virtualni gumbi

Virtualni gumb je virtualni objekt koji služi kao gumb i korisniku daje mogućnost neke radnje. On ne postoji u stvarnoj okolini i vidljiv je samo kada se pokrene AR aplikacija, a stvori se zajedno sa svim virtualnim objektima. Dodiruje se u stvarnoj okolini, iako sâm ne postoji, a preko zaslona možemo vidjeti njegovu lokaciju i dodirnuti to mjesto, odnosno pritisnuti gumb. Kao i obični gumbi korišteni u aplikacijama, virtualni gumb reagira na dodir, a taj efekt se postiže tako da aplikacija detektira određeno prekriveno područje i to smatra pritiskom.

Postavljanje virtualnog gumba

Kako nam je za detekciju virtualnog gumba potrebno da aplikacija prepozna radnju na određenom dijelu slike, virtualne gumbe postavljamo na zadani ciljani objekt na kojem je i naš virtualni model. Ovisno o verziji Vuforie, virtualni gumb možemo postaviti na dva načina.

U starijim verzijama, virtualni gumb postavljamo tako da odemo u **Project** → **Assets** → **Vuforia** → **Prefabs** i izaberemo **Virtual Button**. U novijim verzijama možemo ga pronaći tako da lijevim klikom u **Sample Scene** odaberemo **Image Target** kojem dodjeljujemo virtualni gumb, a zatim u **Inspector** → **Advanced** odaberemo gumb **Add Virtual Button**. U oba slučaja na našem **Image Targetu** pojavit će se plavi pravokutnik. Taj pravokutnik označava područje koje će se detektirati kao virtualni gumb, a nakon dodavanja po želji ga možemo transformirati. Lijevim klikom na **Virtual Button** otvara se **Inspector** u kojem možemo pod **Virtual Button Behaviour (script)** našem virtualnom gumbu dodijeliti ime (**Name**) i osjetljivost (**Sensitivity Setting**). Kao ime je najbolje odabrati isto ono kako smo nazvali virtualni gumb u **Sample Scene**, a osjetljivost postaviti na **HIGH**.

Slika 3.34: Virtualni gumb u Unityju

Iako unutar **#Scene** vidimo virtualni gumb, odnosno područje koje se detektira na dodir, ono neće biti vidljivo tijekom pokretanja aplikacije. Kako bismo napravili vidljiv gumb koji će uputiti korisnika na to mjesto koristimo neki model, na primjer kocku. Dakle našem virtualnom gumbu pridružimo kocku tako da u **Sample Scene** desnim klikom odaberemo **Virtual Button** → **3D Object** → **Cube** te je potom transformiramo tako da veličina kocke odgovara veličini ravnine koja predstavlja virtualni gumb. Naravno, kocki možemo smanjiti visinu da dobijemo standardniji oblik gumba.
Nakon što smo transformirali kocku da nam predstavlja gumb, možemo joj dodati neki tekst koji će opisivati rad toga gumba. Njega dodajemo tako da desnim klikom miša na

Cube u **Sample Scene** odaberemo **3D Object** → **3D Text**. Lijevim klikom miša na **3D Text** otvaramo **Inspector** te možemo izabrati opcije za uređivanje teksta. Pod **Transform** → **Rotate** → **X** možemo postaviti parametar na **90** da tekst bude u istoj ravnini kao i **Image Target** te ga postaviti na kocku koja predstavlja određeni gumb. Pod **Text Mash** imamo opciju **Text** i kućicu u koju možemo upisati željeni tekst. Po zadanim postavkama taj tekst je **Hello World**, a mi možemo upisati željeni tekst koji će se prikazati u **#Scene** onako kako smo ga upisali, uključujući i razmake i novi red. Također imamo i opcije za poravnavanje teksta (**Alignment**), promjenu veličine slova (**Font Size**), stila slova (**Font Style**), boje (**Color**) itd.

Slika 3.35: Vizualni prikaz područja virtualnog gumba

Za kraj trebamo još našem **Image Targetu** dodijeliti C# skriptu koja će opisivati rad naših virtualnih gumba. Skriptu dodijelimo lijevim klikom miša na **Image Target** unutar **Sample Scene** te u **Inspector** odabirom **Add Component** → **New Script** imenujemo danu skriptu. Zatim unutar nekog editora otvorimo tu skriptu i dodamo potreban kôd.

Ovisno o broju gumba, dajemo dva primjera kôda.
Jedan gumb i pripadne funkcije je moguće je direktno definirati. U slučaju više virtualnih gumba, kako bi se izbjeglo ponavljanje istog kôda, bolje je koristiti definirani niz virtualnih gumba te funkcije unutar **OnButtonPressed()** i **OnButtonReleased()** razdvojiti na slučajeve.

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.Events;
using Vuforia;              // ako već nije, treba dodati using Vuforia jer je tamo definirano sučelje za virtualne gumbe

public class VirtualButtonScript: MonoBehaviour,
    IVirtualButtonEventHandler // sučelje za virtualne gumbe (definira funkcije OnButtonPressed() i OnBUttonReleased())
{
    //definiramo varijable koje će nam služiti kao virtualni gumb i model kojeg transformiramo
    private GameObject virtualButton;
    private GameObject model;

    //funkcija koja se pokreće kada se pokrene program
    void Start()
    {
        //pronalazi komponentu s imenom "Virtualni gumb" koja je definiran u Unityju (ime određenog virtualnog gumba)
        virtualButton = GameObject.Find("Virtualni gumb");

        //komponenti s tim imenom pridruži komponentu tipa Virtual Button i označi je kao aktivan događaj (event)
        virtualButton.GetComponent<VirtualButtonBehaviour>().RegisterEventHandler(this);

        //pronalazi komponentu s imenom "ime modela" koja je definiran u Unityju (ime određenog modela)
        model = GameObject.Find("ime modela");

    }
```

Slika 3.36: VirtualButtonScript.cs za jedan gumb (prvi dio)

```
    //funkcija koja se pozove kada se "dodirne" ili "drži" virtualni gumb
    public void OnButtonPressed(VirtualButtonBehaviour vb)
    {
        //funkcija koja se obavlja nad modelom nakon "dodira" virtualnog gumba (u ovom slučaju rotacija)
        model.transform.Roatate(new Vector3(0, Time.deltaTime * 1000, 0));

    }

    //funkcija koja se pozove kada se "pusti" virtualni gumb
    public void OnButtonReleased(VirtualButtonBehaviour vb)
    {
       //kako samo koristimo jedan dodir, nije potrebno dodavati nikakav kod unutar ove funkcije

       //ako bismo imali neku radnju koja se odvija sve dok je virtualni gumb "pritisnut" npr. animacija ili pokretanje videa,
            //onda ovdje dodamo što se dogodi nakon što se virtualni gumb "pusti" npr. prekine se animacija ili pauzira video
    }

}
```

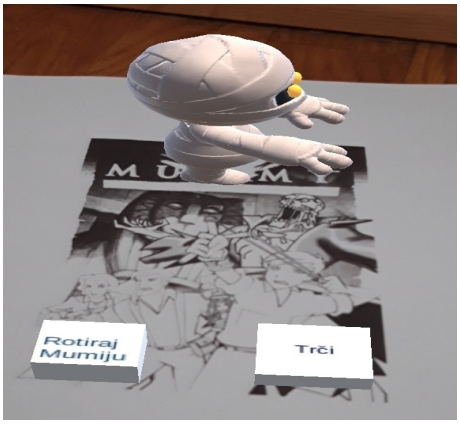
Slika 3.37: VirtualButtonScript.cs za jedan gumb (drugi dio)

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.Events;
using Vuforia;              // ako već nije, treba dodati using Vuforia jer je tamo definirano sučelje za virtualne gumbe

public class VirtualButtonScript: MonoBehaviour,
    IVirtualButtonEventHandler // sučelje za virtualne gumbe (definira funkcije OnButtonPressed() i OnBUttonReleased())
{
    //definiramo varijablu koja će nam služiti kao model kojeg transformiramo
    private GameObject model;

    //funkcija koja se pokreće kada se pokrene program
    void Start()
    {
        //pronalazi sve komponente od nekog Image Targeta definirane u Unityju tipa Virtual Button i stavlja ih u niz
        VirtualButtonBehaviour[] vbs = GetComponentsInChildren<VirtualButtonBehaviour>();
        for (int i = 0; i < vbs.Length; ++i)
        {
            vbs[i].RegisterEventHandler(this);  // svaku komponentu iz niza označi je kao aktivan događaj (event)
        }

        //pronalazi komponentu s imenom "ime modela" koja je definiran u Unityju (ime određenog modela)
        model = GameObject.Find("ime modela");

    }
```

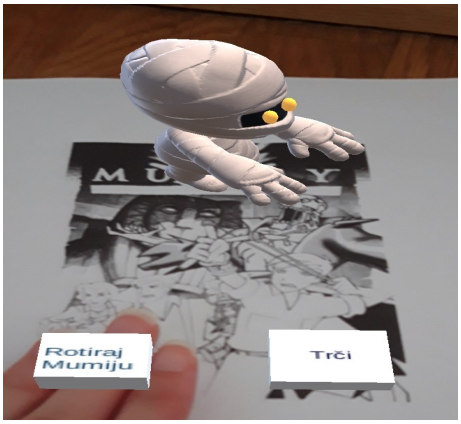
Slika 3.38: VirtualButtonScript.cs za više gumba (prvi dio)

```
    //funkcija koja se pozove kada se "dodirne" ili "drži" virtualni gumb
    public void OnButtonPressed(VirtualButtonBehaviour vb)
    {
        //kada se "pritisne" virtualni gumb, odredi naziv gumba koji je "pritisnut" te obavi funkcijiu
            //koja se obavlja nad modelom nakon "pritiska" virtualnog gumba

        switch (vb.VirtualButtonName)
        {
            case "RotateMummyButtonY":
                {
                    mummy.transform.Rotate(new Vector3(0, Time.deltaTime * 1000, 0));
                }
                 break;
            case "RotateMummyButtonX":
                {
                    mummy.transform.Rotate(new Vector3(Time.deltaTime * 100, 0, 0));
                }
                break;
        }
    }

    //funkcija koja se pozove kada se "pusti" virtualni gumb
    public void OnButtonReleased(VirtualButtonBehaviour vb)
    {
        //kako samo koristimo jedan dodir, nije potrebno dodavati nikakav kod unutar ove funkcije

        //ako bismo imali neku radnju koja se odvija sve dok je virtualni gumb "pritisnut" npr. animacija ili pokretanje videa,
            //onda ovdje dodamo što se dogodi nakon što se virtualni gumb "pusti" npr. prekine se animacija ili pauzira video
                //također definirano kao switch - case
    }
}
```

Slika 3.39: VirtualButtonScript.cs za više gumba (drugi dio)



Slika 3.40: Model prije virtualnog gumba



Slika 3.41: Model kada se pritisne virtualni gumb

Nakon dodavanja i pisanja skripte te postavljanja virtualnog modela na **Image Target**, možemo spremiti, kompajlirati i pokrenuti naš program. Kada pokrenemo aplikaciju na definiranom cilju vidimo virtualni model i virtualne gumbe. Možemo ih pritisnuti, odnosno dodirnuti područje u stvarnoj okolini gdje se dani gumb nalazi, i vidjeti koja radnja se poduzima nad našim virtualnim modelom.

Problemi i rješenja

Kako virtualni gumbi rade na dodir, odnosno detektiraju se kao određeno prekriveno područje, poželjno ih je postaviti na područje na kojem se nalazi dovoljno značajnih točaka korištenog cilja kako bi se jednostavno i jedinstveno očitali. Također je poželjno i odrediti dovoljnu veličinu zadane ravnine koja predstavlja virtualni gumb i pobrinuti se da je cijela ravnina unutar ciljanog objekta. U obzir treba uzeti želimo li da korisnik pritišće virtualni gumb samo prstom ili većim dijelom dlana.
Problem nastaje kod korištenja više virtualnih gumba na istom ciljanom objektu. U tom slučaju treba izbjegavati slaganje virtualnih gumba u stupac, tj. prema virtualnom objektu, kako bi korisnik mogao podjednako lako dodirnuti sva područja. Slaganje virtualnih gumba u stupac može pri korisnikovom dodiru uzrokovati preveliko prekrivanje površine što smanjuje jedinstveno određivanje ciljanog gumba, a može dovesti i do gubitka svih virtualnih objekata s danog cilja zbog ometanog praćenja.
Dakle, za što lakšu i bolju detekciju virtualnih gumba u obzir treba uzeti područje sa što više značajnih točaka, područje sa što većim kontrastom, veličinu i položaj svakog virtualnog gumba te njihov međusobni odnos, tj. razmještaj takav da je između svaka dva gumba dovoljan razmak i takav da korisnik pri korištenju nekog gumba ulazi u što manje područje drugog.

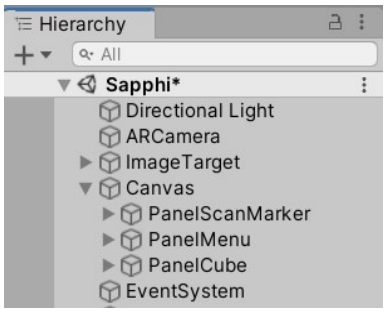
3.4.2 Korisničko sučelje

Korisničko sučelje je poveznica između korisnika i uređaja koja korisniku daje mogućnost upravljanja uređajem i pristupa aplikacijama preko dodira zaslona. Korisničko sučelje kod aplikacija za proširenu stvarnost osim što daje izgled samoj aplikaciji, može poslužiti i kao metoda interakcije korisnika s virtualnim objektima. Preko dodira na zaslonu korisnik može transformirati neki virtualni objekt, pokrenuti neku animaciju ili neku drugu funkciju. Samo praćenje i detekcija cilja ima mogućnost promjene izgleda korisničkog sučelja.

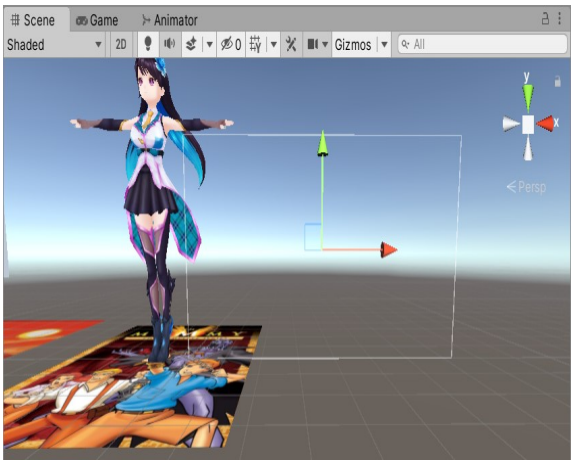
Postavljanje korisničkog sučelja

Vuforia kao još jednu mogućnost daje i dizajniranje korisničkog sučelja, promjenu korisničkog sučelja ovisno o detekciji cilja te interakciju s virtualnim objektom preko korisničkog sučelja.

Komponente sučelja možemo izraditi sami u nekom programu za dizajn (npr. Photoshop) ili možemo preuzeti iz Unity Asset Storea, a prije korištenja moramo ga dovući u **Assets**. Nakon što smo u **Sample Scene** dodali **AR Cameru**, **Image Target** i dodijelili mu željeni model, u **Sample Scene** desnom tipkom miša dodajemo korisničko sučelje preko **UI** → **Panel**. Kada to napravimo, u **Sample Scene** možemo vidjeti da se pojavio **Canvas** s komponentom **Panel** te **EventSystem**, a unutar **#Scene** pravokutnik koji predstavlja sučelje uređaja postavljeno vodoravno.



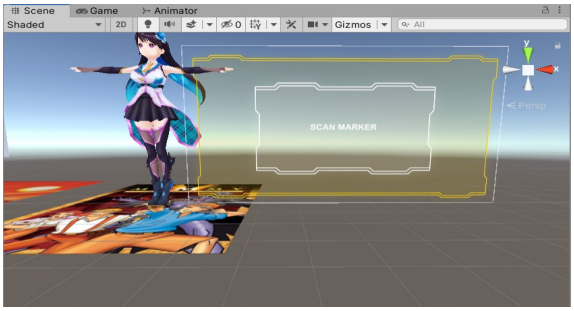
Slika 3.42: Definirane komponente u **Sample Scene** hijerarhiji



Slika 3.43: Prikaz **Panela** u **#Scene**

Dizajnirane komponente sučelja, npr. pozadinu, gumbe, okvir itd., pridružujemo željenom **Panelu** pritiskom lijeve tipke miša te nakon što se otvori **Inspector** u dio **Image** → **Source Image** dovučemo željenu slikovnu komponentu koja se nalazi u **Assets**. Također pod

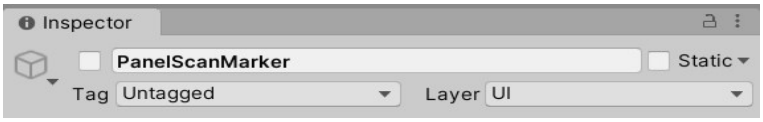
Color možemo i odabrati boju sučelja. Zbog lakšeg praćenja komponenata, predlaže se preimenovanje **Panela** u neko smisleno ime, npr. funkciji kojoj služi (izbornik, početni zaslon itd.)
Ako unutar našeg **Panela** želimo imati još neku komponentu, dodajemo je pritiskom na **Panel** desnom tipkom miša te odabirom na **UI** → **Image**. Isto kao i kod **Panela**, dodanoj komponenti **Image** možemo na isti način pridružiti neku slikovnu komponentu u **Source Image**, te je i samu možemo transformirati ili joj promijeniti boju. Ako nekoj komponenti želimo dodati tekst to radimo pritiskom desne tipke miša na određenu komponentu, a zatim biramo **UI** → **Text** kojeg u **Inspectoru** možemo transformirati po želji. Pokraj opcije **#Scene** nalazi se i opcija **Game**. Preko nje možemo vidjeti kako izgleda pojedini **Panel** kada se pokrene na korisnikovom uređaju.



Slika 3.44: Prikaz **Panela** u **#Scene**

Slika 3.45: Prikaz **Panela** u **Game**

Jednom **Canvasu** možemo pridružiti više **Panela** desnom tipkom miša na njega, a zatim odabirom **UI** → **Panel**. Novi **Panel** u **#Scene** će biti postavljen na isto mjesto kao i prvi. Ako želimo da novi Panel ima iste komponente kao prvi, npr. želimo da se okvir ili isti gumbi nalaze na istom mjestu, otvorimo **Inspector** prvog **Panela** te pod **Rect Transform** u desnom dijelu pritisnemo (⋮) te odaberemo **Copy Component**. Zatim otvorimo **Inspector** drugog **Panela** te na istom mjestu odaberemo **Paste Component Values**.
Kako se oba **Panela** nalaze na istom mjestu, kako bismo neometano mogli dizajnirati samo jedan **Panel** bez drugog u pozadini, **Panele** koje trenutno ne koristimo možemo privremeno ukloniti iz **#Scene** tako da otvorimo **Inspector** određenog **Panela** te pokraj njegova imena uklonimo kvačicu.



Slika 3.46: Primjeri isključenog **Panela**

Ako želimo da se jedan **Panel** prikazuje kada se traži cilj, a neki drugi kada se cilj pronađe potrebno je postaviti aktivnosti za pojedini **Panel**, a to postavljamo modificiranjem skripte **DefaultTrackableEventHandler**. Dani dijelovi kôd izgledaju ovako:

```
9   using UnityEngine;
10  using Vuforia;
11
12  public class DefaultTrackableEventHandler : MonoBehaviour, ITrackableEventHandler
13  {
14      #region PROTECTED_MEMBER_VARIABLES
15
16      // svi korišteni Paneli kao varijable
17      public Transform ime_prvog_panela;
18      public Transform ime_drugog_panela;
19
```

Slika 3.47: DefaultTrackableEventHandler.cs - dodane potrebne varijable

```
99
100         //postavka aktivnosti pojedinog Panela
101         ime_prvog_panela.gameObject.SetActive(true);  //kada se pronađe ciljani objekt prikazuje se ovaj Panel
102         ime_drugog_panela.gameObject.SetActive(false);
```

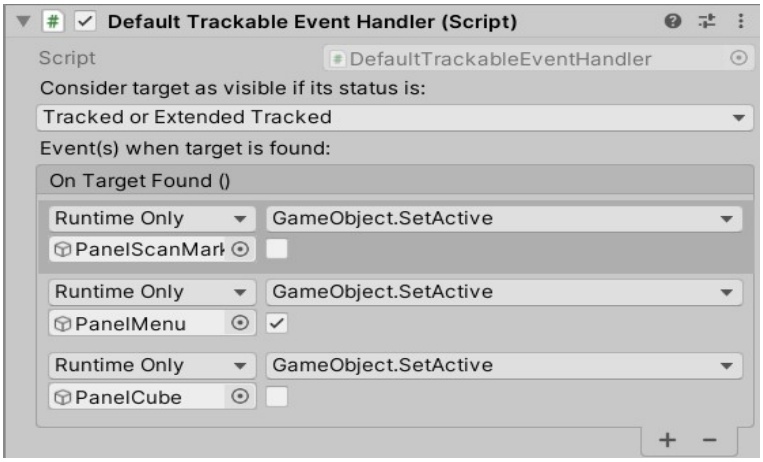
Slika 3.48: DefaultTrackableEventHandler.cs - dodatne funkcije u OnTrackingFound()

```
123
124         //postavka aktivnosti pojedinog Panela
125         ime_prvog_panela.gameObject.SetActive(false);
126         ime_drugog_panela.gameObject.SetActive(true); //kada se izgubi ciljani objekt prikazuje se ovaj Panel
127
```

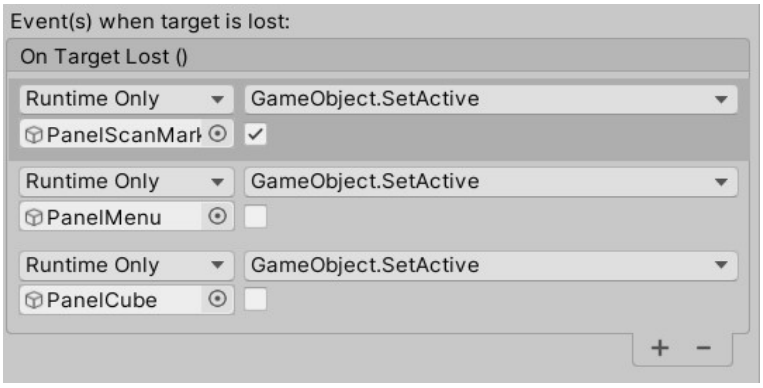
Slika 3.49: DefaultTrackableEventHandler.cs - dodatne funkcije u OnTrackingLost()

Novije verzije Unityja nude podešavanje aktivnosti unutar njega samog bez dodavanja koda u skriptu. Unutar Unityja to činimo tako da otvorimo **Inspector** od **Image Targeta** te pod **Default Trackable Event Handler (Script)** → **On Target Found ()** u danu listu elemenata, pritiskom na znak +, dodamo novi element. U polje ispod **Runtime Only**, koje je po zadanim postavkama **None (Object)**, dovučemo iz **Sample Scene** određeni **Panel** ili ga pritiskom na ⊙ u kućici potražimo kada se ponudi **Select Object**. U kućicu desno od **Runtime Only**, u kojoj piše **No Function**, odaberemo **Game Object** → **Set Active**. Sve **Panele** koje želimo koristiti, odnosno prikazati kao aktivne, uključimo pritiskom na kvadratić ispod funkcije. Svi objekti pokraj kojih se nalazi kvačica smatraju se aktivnima,

a oni bez nje neaktivnima, odnosno u skripti je funkcija **Set Active** automatski postavljena kao **true**, odnosno **false**.
Isti postupak napravimo za **On Target Lost** ().



Slika 3.50: Podešavanje aktivnosti u OnTrackingFound()



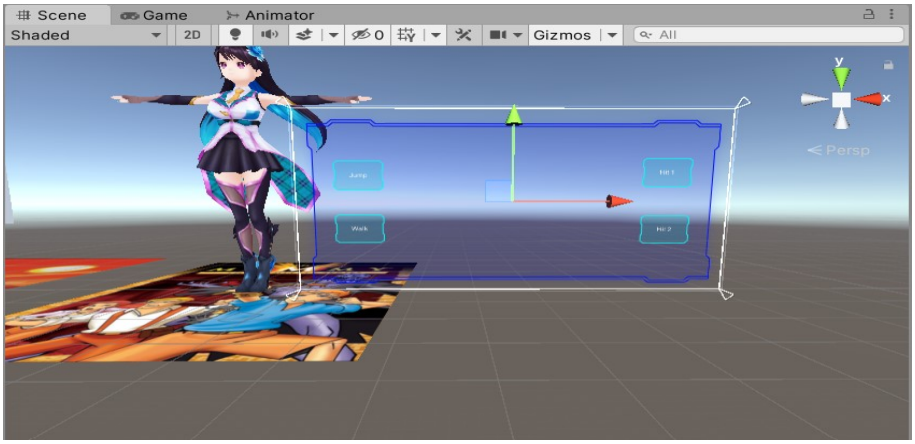
Slika 3.51: Podešavanje aktivnosti u OnTrackingLost()

Interakcija s modelom

Kao što je već spomenuto, preko korisničkog sučelja moguće je stupiti u interakciju s virtualnim objektom. Najjednostavniji način je korištenje gumba i definiranje određene radnje koju virtualni objekt čini jednom kada se pritisne na njih.

Gumbe dodajemo tako da pritisnemo određeni **Panel** desnom tipkom miša i odaberemo **UI** → **Button**. Na našem **Panelu** će se pojaviti pravokutnik koji predstavlja određeni gumb, a po želji ga možemo preimenovati i transformirati. Kao i kod **Panela** i **Imagea**, i **Buttonu** možemo pridružiti neku slikovnu komponentu preko **Source Image** i dodijeliti tekst preko **UI** → **Text**.

Ako želimo neki **Button** učiniti nevidljivim, to napravimo tako da odemo u **Inspector** pa postavimo parametar u **Image** → **Color** → **A** na **0**. Vrijednost **A** (*alpha*) označava prozirnost slike, a ako je parametar postavljen na 0, tada je slika u potpunosti prozirna.



Slika 3.52: **Panel** s gumbima u **#Scene**

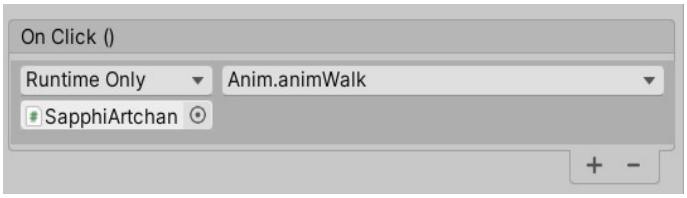
Slika 3.53: **Panel** s gumbima u **Game**

Povezivanje određenog gumba s određenom radnjom, koju će virtualni objekt napraviti, vršimo preko skripte. Definiramo novu skriptu te preko editora dodajemo određeni kôd u kojem definiramo funkcije:

```
1   using System.Collections;
2   using System.Collections.Generic;
3   using UnityEngine;
4
5   public class Anim : MonoBehaviour
6
7   {   //varijabla koja će sadržavati model koji se animira
8       private Animator anim;
9
10      void Start()
11      {
12          anim = GetComponent<Animator>();  //varijabli s modelom se pridodaje
13      }                                          //komponenta animator
14
15      public void animWalk()              //funkcija se zove kada se pritisne
16      {                                       //određeni gumb
17
18          anim.Play("walk", -1, 0f);      //pokretanje animacije iz Animatora
19      }                                       //pod imenom "walk"
20
21      public void animHit1()
22      {
23          anim.Play("hit01", -1, 0f);
24      }
25
26      public void animHit2()
27      {
28          anim.Play("hit02", -1, 0f);
29      }
30
31      public void animJump()
32      {
33          anim.Play("jump", -1, 0f);
34      }
35  }
```

Slika 3.54: Buttons.cs - funkcije definirane za pritisak pojedinog gumba

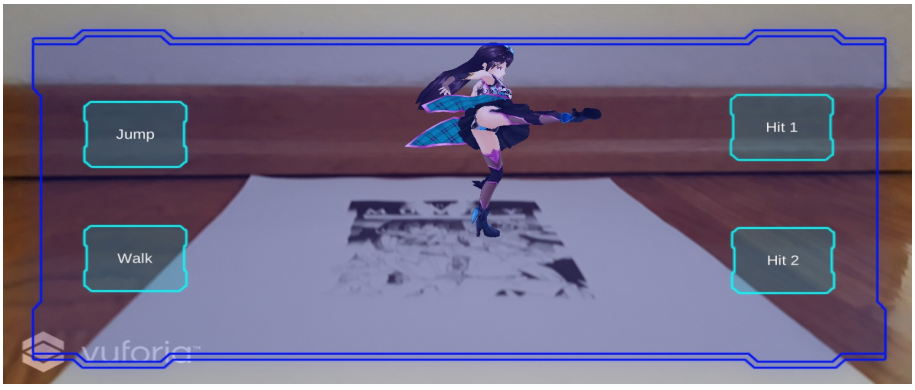
Nakon što spremimo skriptu, pridružimo je određenom modelu. To možemo učiniti pritiskom lijeve tipke miša na model, potom u **Inspector → Add Component** odaberemo željenu skriptu.
Nakon toga, za svaki **Button** podesimo funkciju **On Click ()** (otvaramo **Inspector → Button → On Click()**). U kućicu gdje piše **None (Object)** dovučemo naš model, a u kućicu gdje piše **No Function** pronađemo ime naše skripte, odaberemo je i odaberemo funkciju koju želimo dodijeliti tom gumbu.



Slika 3.55: Funkcija OnClick() unutar Unityja

Nakon što smo to napravili spremimo **Sample Scene**, kompajliramo kod i pokrenemo ga na mobilnom uređaju. Iako je u samom programu korisničko sučelje napravljeno u vodoravnom položaju zaslona, iste komponente u istom položaju pojavit će se i u horizontalnom položaju.

Slika 3.56: Korisničko sučelje prije detekcije ciljane slike



Slika 3.57: Korisničko sučelje nakon detekcije cilja

Ako želimo drugačiji izgled korisničkog sučelja kada pronađemo neki drugi cilj unutar iste aplikacije, potrebno je podesiti aktivnosti svakog **Panela** u svakom **Inspectoru** od **Image Targeta** u **On Target Found ()** i **On Target Lost ()** ili preko skripte.

Kako bi se izbjegle greške u kodu pri pozivanju funkcije za praćenje i postavljanje aktivnosti, preporučuje se korištenje neke novije verzije Unityja (npr. Unity 2020) koji nudi podešavanje danih funkcija unutar samog Unity Editora.

Problemi i rješenja

Prilikom promjene **Panela** tijekom korištenja aplikacije, može se dogoditi da se prikazuje još uvijek isti **Panel** koji se pojavljuje kada detektiramo određeni cilj, iako smo prestali pratiti taj cilj. Slično kao i kod slika, razlog je pamćenje lokacije usidravanja virtualnog objekta prethodnog cilja te aplikacija još uvijek smatra da se prati taj isti cilj. Rješenje je prebacivanje fokusa kamere na neku drugu podalju lokaciju ili „prisilno“ preusmjeravanje, odnosno prekrivanje kamere uređaja kako bi se izgubio cilj.
Također treba pripaziti da su aktivnosti u funkcijama **On Target Found ()** i **On Target Lost ()** dobro postavljene da ne bi dolazilo do konflikata.

3.5 Animacije

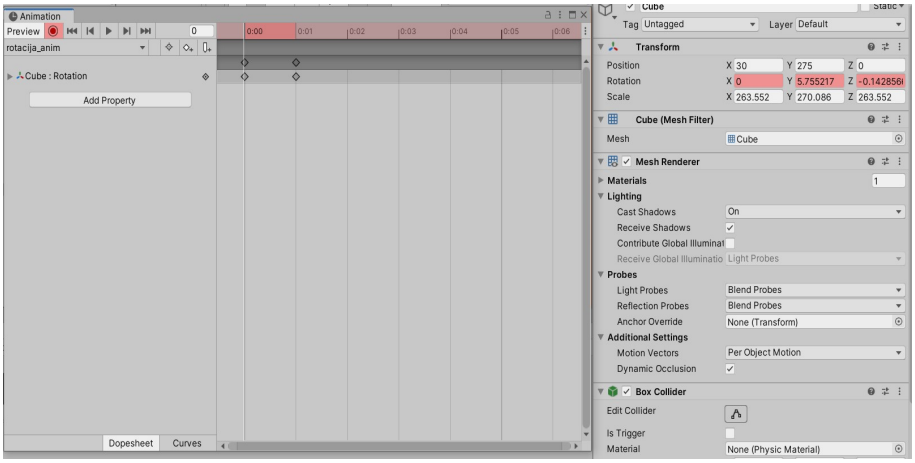
Mnogi modeli iz Unity Asset Storea imaju već definirane animacije koje možemo koristiti u našem programu. Ako želimo vlastitim modelima dodati neku animaciju, to je moguće u samom Unityju.

Postavljanje animacije

Game Objectu kojeg želimo animirati animaciju dodajemo tako da ga pritisnemo lijevom tipkom miša, a zatim u glavnom izborniku izaberemo **Window → Animation → Animation**. Tada nam se na zaslonu otvori prozor **Animation** koji nudi opciju **Create**, a pritiskom na nju imenujemo i spremamo našu animaciju u formatu .anim. Nakon spremanja na desnoj strani možemo vidjeti vremensku traku, a na lijevoj ime našeg **Game Objecta** i dodijeljene animacije nakon što je dodamo pomoću **Add Property**, koji nudi nekoliko različitih mogućnosti animiranja. Također, u gornjem lijevom kutu možemo pronaći alatnu traku s opcijama **Record**, **Play**, **Pause** itd. Po zadanim postavkama, u vremenskoj traci jedan **Keyframe**, odnosno ključni kadar, postavljen je na 1 sekundu. Po želji ga možemo smanjiti ili povećati. Animacija se izvodi tako da se ključni kadar ponavlja.

U točki **2.2**, odjeljak **Postavljanje modela na ciljani objekt**, opisali smo kako možemo kocku rotirati preko koda iz skripte. Ovdje ćemo opisati kako možemo rotirati kocku preko animacije.

Nakon otvaranja prozora **Animation** za dani **Game Object**, u ovom slučaju to je kocka, odaberemo **Add Property** → **Transform** → **Rotation**. Zatim odaberemo u alatnoj traci **Record** (●). Kada smo to napravili možemo primijetiti da se dio vremenske trake i dio **Inspector** → **Transform** zacrvenio. To znači da će animacija biti definirana promjenom tih parametara.

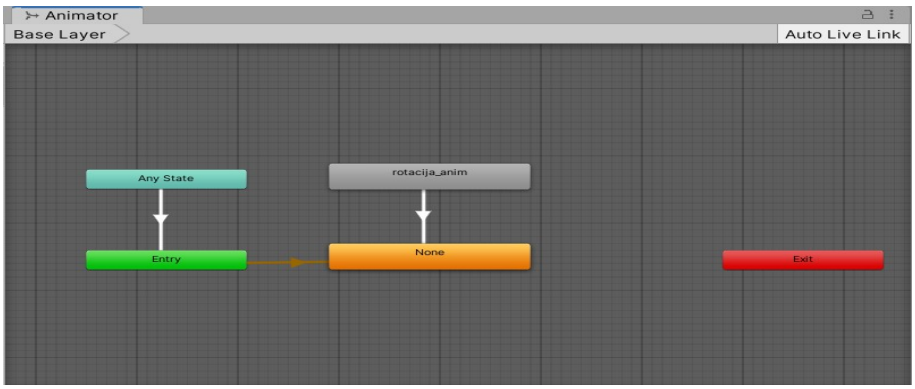


Slika 3.58: Snimanje animacije (**Record**)

Nakon uključivanja **Recorda**, u **Inspector** → **Transform** → **Rotate** → **Y** upisujemo parametar 359 te isključimo **Record**. Time smo napravili animaciju kocke koja se rotira oko Y osi.

Pokraj opcija **#Scene** i **Game** možemo pronaći i opciju **Animator**. Pritiskom na nju otvara nam se prozor koji sadrži stanja u kojima se može naći naš animirani **Game Object** te definirane prijelaze između njih. Kao stanja su navedene sve animacije te još nekoliko stanja poput **Any State**, **Entry** i **Exit**. Novo stanje možemo dodati pritiskom desne tipke miša i odabirom **Create State** → **Empty** te mu dodijelimo neko ime. Neko stanje možemo povezati tako da ga odaberemo desnom tipkom miša, i izaberemo **Make Transition** te pritisnemo stanje u koje želimo da ide, što će se prikazati kao usmjereni brid.
Najčešće želimo kreirati neko novo stanje koje će nam služiti kao stanje u kojem model neće raditi ništa, npr. nazovemo to stanje **None**. Ono nam služi kao stanje u koje model prelazi nakon što želimo da se neka animacija prekine, npr. želimo da se kocka rotira samo kada pritisnemo gumb, a inače ne. Takvo stanje želimo postaviti kao početno i željenu

animaciju zatim povezati s njim. Kao početno stanje, postavljamo ga pritiskom desne tipke miša na njega te izaberemo **Set as Default Layer State**.



Slika 3.59: Animator

Našu animaciju nekom **Buttonu** dodajemo u **On Click ()** na isti način kao što je prethodno opisano. U kućicu gdje piše **No Function** ovaj put odaberemo **Animator → Play(string)** te u kućicu ispod napišemo ime naše animacije. Ako dodajemo animaciju preko skripte, potrebno je dodati varijablu tipa **public Animator** te je dohvatiti pomoću **GetComponent< Animator>()**. Kada je želimo pokrenuti koristimo funkciju s parametrom tipa string - **Play("naziv animacije")**, a zaustavljamo je prebacivanjem u stanje **None** tj. zovemo funkciju **Play("None")**.

Problemi i rješenja

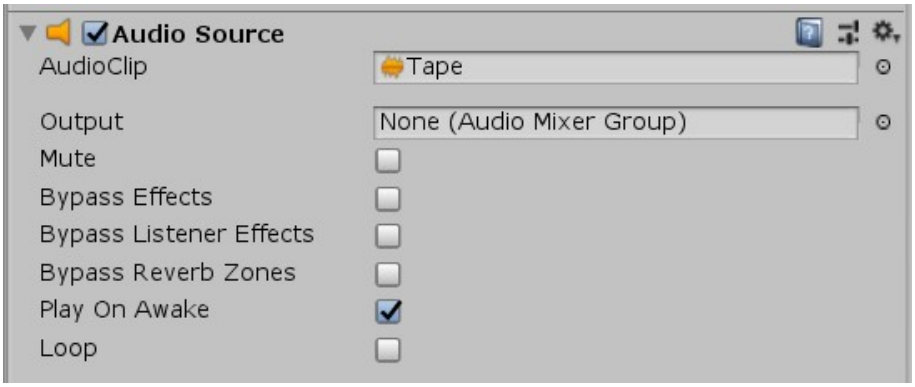
Prilikom definiranja animacije moramo pripaziti da se kraj prethodnog ključnog kadra i početak sljedećeg podudaraju. Ukoliko taj prijelaz nije dobro definiran, virtualni objekt prilikom animacije može na trenutak zastati ili naglo prijeći iz jednog položaja u drugi što djeluje neprirodno.
Također treba i dobro definirati prijelaze iz jednog stanja u drugo kako prilikom pokretanja aplikacije ne bi dolazilo do istovremenog prijelaza u više različitih stanja što može rezultirati neispravnim radom ili rušenjem cijele aplikacije.

3.6 Zvuk

Unutar naše aplikacije možemo koristiti i zvučne pozadine koje će svirati kada se pokrene aplikacija, kada se detektira neki cilj, a možemo ga dodijeliti i nekoj komponenti. Sve zvučne datoteke koje koristimo u našoj aplikaciji dovučemo u **Assets** u nekom zvučnom formatu (npr. .mp3).

Postavljanje zvuka

Ako želimo koristiti neku zvučnu pozadinu, zvučnu datoteku dodajemo tako da u **Sample Scene** desnom tipkom miša odaberemo **Audio → Audio Source**. Kada otvorimo njegov **Inspector** u **Audio Source → AudioClip** dodajemo željenu zvučnu datoteku. Po zadanim postavkama uključena je opcija **Play On Awake** što znači da će se naš zvuk reproducirati čim se aplikacija pokrene, što u slučaju pozadinske glazbe i želimo. Također možemo i uključiti opciju **Loop** kako bi se zvuk nanovo reproducirao kada prvi put dođe do kraja.



Slika 3.60: **Audio Source** u Unityju

Ako želimo da se određeni zvuk reproducira tek kada detektiramo cilj, zvuk dodajemo željenom **Image Targetu** tako da u **Inspectoru** odaberemo **Add Component → Audio Source** te na prethodno opisani način, dodajemo zvučnu datoteku. Ovaj put ne želimo da se zvuk reproducira čim se aplikacija pokrene nego tek kada detektiramo cilj. Stoga isključimo opciju **Play On Awake**. Potrebno je i podesiti skriptu **DefaultTrackableEventHandler** na isti način kako je opisano u točki **3.2.2**, odjeljak **(Postavljanje videa)** , samo zamijenimo **VideoSource** s **AudioSource**. Ovime možemo postići da svaki Image Target reproducira drugi zvuk.

Zvučnu datoteku možemo pokrenuti i pritiskom na neki gumb. Ako se radi o virtualnom gumbu, u skriptu koja se koristi za upravljanje virtualnim gumbima, u funkcije **OnButtonPressed()** i **OnButtonReleased()** dodamo liniju koda koja pokreće ili pauzira naš zvuk, a zvučnu komponentu, kao komponentu, dodajemo **Image Targetu**. Ako se radi o gumbu korisničkog sučelja, zvuk dodajemo preko funkcije **OnClick()**.

Problemi i rješenja

Kod korištenja više zvučnih datoteka potrebno je voditi brigu o ponavljanju zvuka i mogućem preklapanju više zvučnih datoteka koje može stvoriti buku. Stoga treba voditi brigu o svrsi korištenja pojedine zvučne datoteke kao i o mjestu na kojem će doći do reproduciranja.
U obzir treba uzeti i glasnoću svake zvučne datoteke i podesiti je na prave parametre kako bi i zvučni efekti davali dojam stvarne okoline, da bliži objekti budu glasniji, a oni udaljeniji tiši.

Bibliografija

[1] Unity — Creating AR apps in Unity, https://unity3d.com/how-to/create-AR-games-in-Unity-efficiently, posjećena 26.10.2019.

[2] D. Schmalstieg , T. Hollerer, *Augmented Reality: Principles and Practice (Usability)*, Addison-Wesley Professional, 1 edition, June 1, 2016.

[3] Vuforia — Vuforia Engine in Unity, https://library.vuforia.com/articles/Training/getting-started-with-vuforia-in-unity.html, posjećena lipanj, 2021.

[4] M. Sidiq, T. Lanker, K.Makhdoomi, *Augmented Reality vs Virtual Reality,*, International Journal of Computer Science and Mobile Computing, Vol. 6, Issue. 6 (lipanj 2017), 324–327.

[5] Unity — Unity tutorial, https://learn.unity.com/tutorials, posjećena 26.10.2019.

Sažetak

U ovom radu opisan je koncept proširene stvarnosti, programi koji se koriste za razvoj aplikacija za proširenu stvarnost te izrada pojedinih komponenti pomoću tih programa.
U prvom poglavlju opisani su pojmovi proširene i virtualne stvarnosti te njihove razlike. Zatim je dan kratki povijesni razvoj koncepta proširene stvarnosti i tehnologija potrebnih za razvoj aplikacija za proširenu stvarnost. Nakon toga opisani su pojmovi praćenja, kalibracije i registracije u kontekstu proširene stvarnosti i način na koji aplikacija za proširenu stvarnost prepoznaje ciljane objekte. Zadnji dio opisuje metode kojima se aplikacija služi kako bi u danoj okolini što vjernije prikazala virtualni objekt.
U drugom poglavlju opisali smo programe za izradu aplikacija za proširenu stvarnost – „Unity“ i „Vuforia“. Pri tome smo objasnili instalaciju oba programa, podešavanje postavki za korištenje, kreiranje korisničkih računa, upoznavanje sa sadržajem i njegovim korištenjem te kompajliranje i pokretanje aplikacije.
U trećem poglavlju smo naveli neke komponente koje Unity i Vuforia nude na korištenje. Pri tome smo opisali kako se one koriste u samim programima te naveli moguće probleme koji se mogu dogoditi prilikom korištenja aplikacije te metode i objašnjenja kojima se ti problemi mogu riješiti.

Summary

In this thesis, we described the concept of augmented reality, programs used for the development of augmented reality applications, and how to create individual components using those programs.
In the first chapter, we explained the terms augmented reality and virtual reality and described the differences between them. Then, we gave a brief history of developing the concept of augmented reality and technologies needed for the development of AR apps. After that, we described the concepts of tracking, calibration, and registration in the context of augmented reality, as well as the way in which an augmented reality application recognizes the target objects. The last part of the chapter describes methods that an application uses for showing virtual objects in a real environment as convincingly as possible.
In the second chapter, we described programs used for the development of augmented reality apps – Unity and Vuforia. We explained how to install both programs, set up the settings, create user accounts, and we introduced their content and usage, as well as how to build and run the apps.
In the third chapter, we listed some components available in Unity and Vuforia. Thereby, we described how to use them within the programs and provided some examples of possible problems that may occur while using the application, as well as the methods for solving them.

Životopis

Rođena sam 21. rujna 1993. u Zagrebu. U istom gradu sam 2008. godine završila osnovnu školu te upisala XV. gimnaziju. 2012. godine, nakon završetka srednje škole, upisala sam preddiplomski studij Matematika na Prirodoslovno-matematičkom fakultetu u Zagrebu. Nakon završetka preddiplomskog studija, 2018. godine upisala sam diplomski studij Računarstvo i matematika.
Tijekom studija nekoliko godina sam sudjelovala kao volonter na Otvorenim danima matematike gdje sam vodila nekoliko različitih radionica.
2018. godine, kao student sam počela raditi u Microblinku u Data Annotation timu.