Razvoj računalne igre u proširenoj stvarnosti

Drulak, Ivan

SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA

Sveučilišni diplomski studij računarstva

RAZVOJ RAČUNALNE IGRE U PROŠIRENOJ STVARNOSTI

Diplomski rad

Ivan Drulak

Osijek, 2019.

Sadržaj

1. UVOD

Tema ovog diplomskog rada je 3D računalna igra koja uključuje povezivanje virtualnog i stvarnog svijeta korištenjem programskog alata Vuforia[1]. Tip igre je 3D AR (*augmented reality*[2]) i pripada akcijskom žanru, first-person shooter[3] (FPS). Za ovakav tip igre bitno je da korisnik ima kameru na svom mobilnom uređaju kako bi se mogla „kreirati" prividna stvarnost (na ekranu uređaja) u odnosu na stvarno okruženje (fizičkom prostoru u kojem se korisnik nalazi). Svaki stvarni pokret korisnika, točnije mobilnog uređaja, utječe na rezultat stvaranja proširene stvarnosti. Naziv igre koja će biti detaljnije objašnjena u ovom diplomskom radu je Legacy. Tema igre je borba protiv zlih kostura i njihovog glavnog vođe koji zajedno žele osvojiti kip koji sadrži energiju života. Igra je napravljena u programu Unity koji predstavlja *game engine*, pri čemu je korišten C# objektni programski jezik za pisanje skripti. Vizualna okolina (scena) igre napravljena je na temelju besplatnih 3D modela koji su preuzeti [1], kao i besplatne glazbe [2] te zvučnih efekata [3]. Izgled korisničkog sučelja je napravljen u programu GIMP 2. Unutar Unityja implementirana je Vuforia kao paket koji omogućuje kreiranje proširene stvarnosti. Kako se to ostvaruje i koji su problemi koji nastaju prilikom korištenje ove tehnologije biti će objašnjeno u poglavlju koje slijedi kao i kratki osvrt na povijest i razvoj te tehnologije. Najpoznatija i najigranija igra za mobilne uređaje koja uključuje proširenu stvarnost je Pokémon GO[4] koja je u travnju 2018. godine imala preko 140 milijuna aktivnih korisnika, igrača [4].

1.1. Zadatak diplomskog rada

Za zadatak diplomskog rada definirana je izrada 3D računalne igre u proširenoj stvarnosti s kojom se treba ostvariti interakcija stvarne i virtualne okoline. Pritom igra mora imati svoju fizičku „podlogu" (*Image Target*[5]) na temelju koje se stvara odnos sa stvarnim i virtualnim svijetom. Za uspješnu realizaciju ovog rada potrebno je znanje iz područja 3D programiranja korištenjem Unity Enginea, osnovno znanje objektno orijentiranog programiranja baziranog na C# programskog jeziku te razumijevanje rada programskog alata Vuforia.

[1] *Vuforia – programski razvojni alat za stvaranje proširene stvarnosti na mobilnim uređajima*
[2] *Augmented reality – odnos između stvarnog i virtualnog okruženja koje je kreirano i prikazano pomoću računala*
[3] *First – person shooter - žanr računalne igre u kojoj igrač igra iz prvog lica pri čemu je oružje centrirano na ekranu*
[4] *Pokémon GO – mobilna igra za Android i iOS uređaje koja je službeno izašla 2016. godine, a razvila ju je tvrtka Niantic*
[5] *Image Target – slika koju Vuforia Engine može detektirati i pratiti*, na temelju koje se stvara proširena stvarnost

2. VUFORIA I POVIJEST PROŠIRENE STVARNOSTI

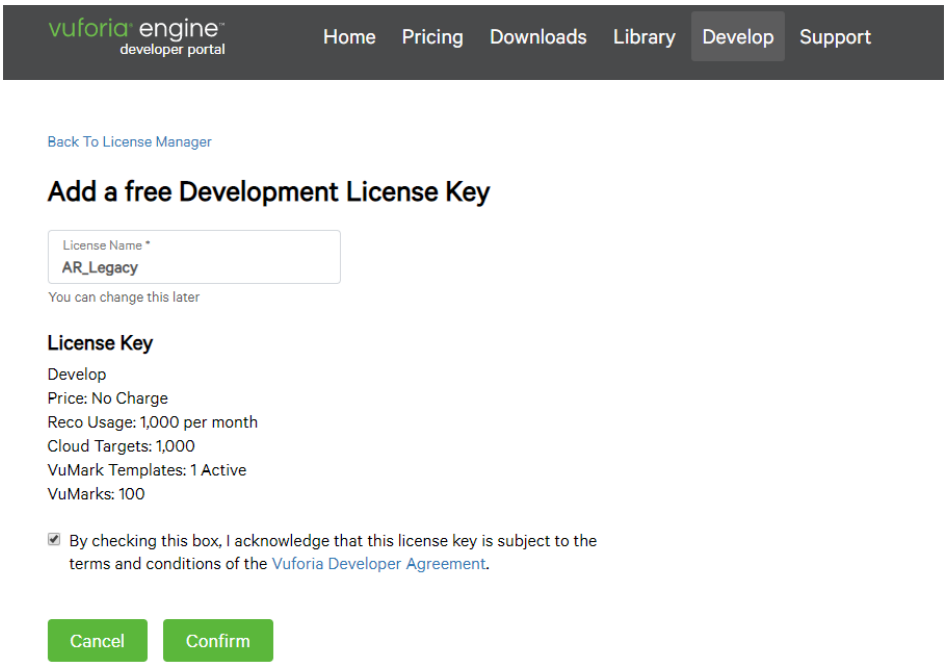
Vuforia se može definirati kao višeplatformski programski razvojni alat koji omogućuje stvaranje proširene stvarnosti i „pomiješane stvarnosti“ (*Mixed Reality*) [5]. Alat je namjenjen za mobilne uređaje pri čemu je potrebno da uređaj ima kameru ili projektor [6]. Povijest razvoja proširene stvarnosti započeo je 1968. godine kada je profesor s Hardvarda, Ivan Sutherland u suradnji sa studentom, Bobom Sproullom napravio prvi uređaj koji stvara virtualni svijet u stvarnoj okolini. Značajnija primjena AR tehnologije bila je 1999. godine kada je NASA koristeći ovu tehnologiju testirala i poboljšavala navigaciju za njihov svemirski brod X-38. Veliki napredak dogodio se 2000. godine kada je Hirokazu Kato objavio softver pod nazivom ARToolKit. Ovim programom je omogućeno praćenje stvarnih aktivnosti i njihovo kombiniranje s virtualnim objektima što je bio velik i bitan skok u razvoju alata za proširenu stvarnost. 2003. godine NFL (National Football League) je koristio Skycam[6] sustav kako bi se prikazao teren sa virtualnim down markerima. Napredniji oblik korištenja proširene stvarnosti pokušao se ostvariti sa Google Glass naočalama koje sadrže projektor koji stvara virtualnu sliku i projicira u oko korisnika. Godine 2016. Microsoft je predstavio svoju inačicu naočala HoloLens koje omogućuje manipulaciju virtualnim objektima u stvarnom svijetu. [7]

2.1. Vuforia i Unity

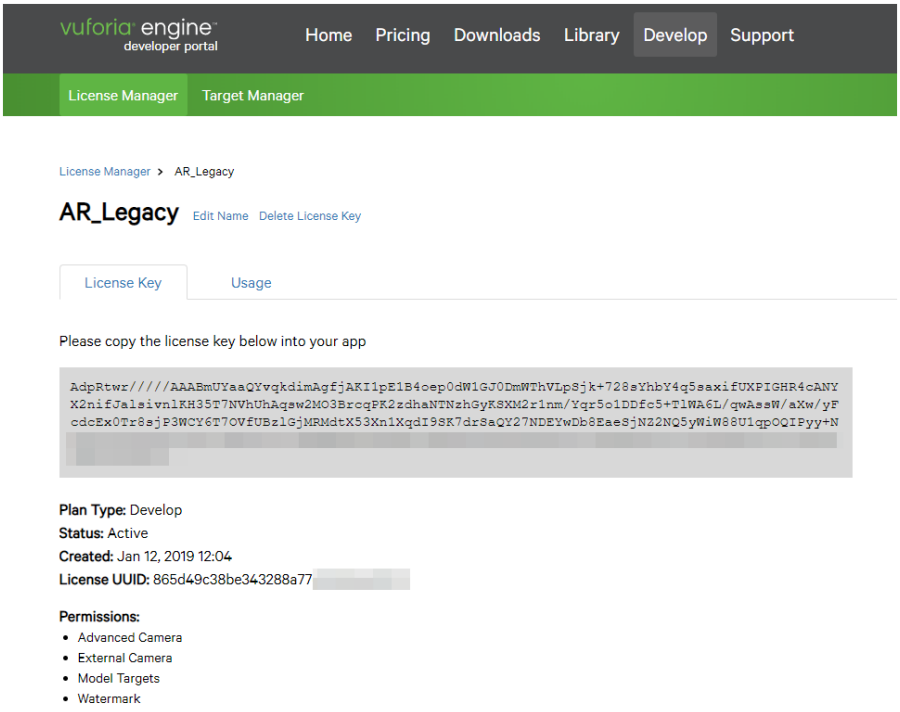
Zajedno korištenje Unityja i Vuforije ne predstavlja nikakav problem. Postupak implementacije ovog alata unutar Unityja biti će ukratko objašnjen u ovom potpoglavlju. Prvo je potrebno izraditi korisnički račun i zatim se prijaviti. Nakon toga se unutar prikaza za licence dodaje novi ključ za aplikaciju. Potrebno je samo definirati naziv licence i prihvatiti uvjete korištenja kao što se može vidjeti na slici (Sl. 2.1.). Nakon kreiranja licence generira se ključ koji je potrebno upisati prilikom korištenja Vuforije unutar Unityja. Kako izgleda dobiveni rezultat može se vidjeti na slici (Sl. 2.2.). Sada kada postoji ključ, potrebno je definirati ime baze (potprozor *Target Manager*) koja će sadržavati željene mete (*targets*). U ovom prozoru se dodaju željene mete koje Vuforia treba prepoznati i na temelju njih stvoriti proširenu stvarnost. Za ovaj rad kao mete koriste se slike (*Image Targets*) pri čemu je bilo potrebno dodati sliku (.jpg ili .png formata) te definirati širinu slike i naziv. Zatim slijedi proces pronalaženja značajki slike i ocjenjivanje

[6] *Skycam – sustav koji kontrolira računalo za snimanje pri čemu se koriste posebne kamera koje se kreću po kabelima*

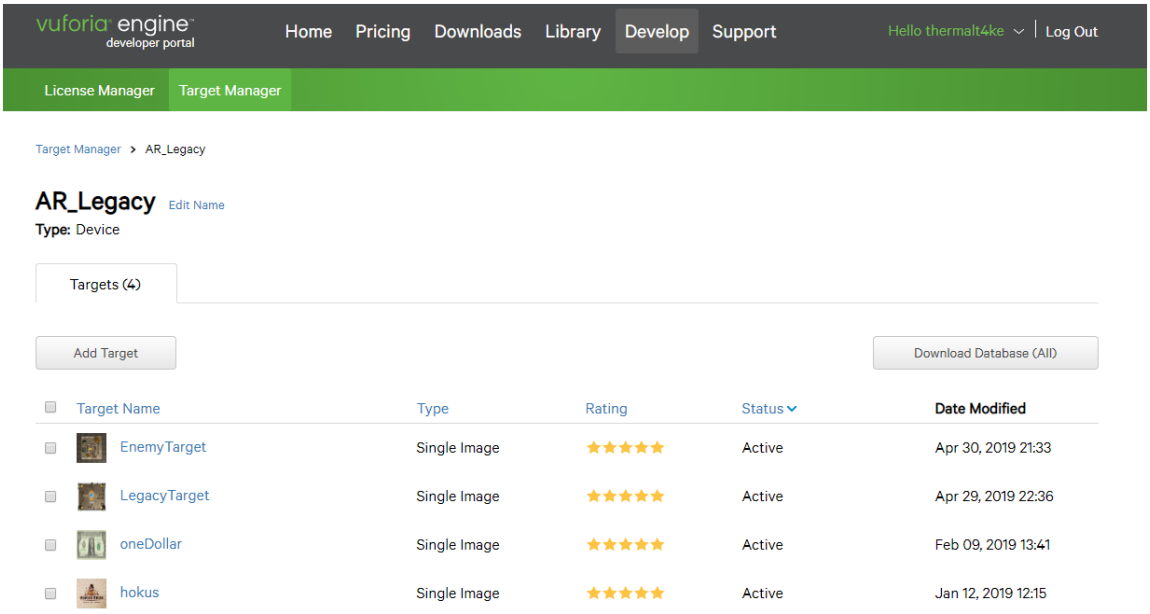
kvalitete prepoznavanja koje Vuforia automatizirano odrađuje. Primjer baze pod nazivom AR_Legacy koja je korištena u ovome radu može se vidjeti na slici (Sl. 2.3.).



Sl. 2.1. Prikaz prozora za izradu licence



Sl. 2.2. Prikaz prozora nakon kreirane licence s generiranim ključem



Sl. 2.3. Prikaz prozora nakon kreirane baze AR_Legacy sa ubačenim slikama (metama)

Na slici iznad se također može vidjeti procjena (*Rating*) kvalitete prepoznavanja svake pojedine slike. Svaka meta (slika) može se otvoriti zasebno kako bi se vidjele detaljnije informacije o slici te značajke (žuti plusevi) koje su karakteristične za prepoznavanje. Kako to izgleda kada se otvore pojedinosti za sliku LegacyTarget može se vidjeti u nastavku (Sl. 2.4.). Detaljnije o implementaciji definirane baze biti će objašnjeno u potpoglavlju 4.1. Ideja i Vuforia implementacija

Sl. 2.4. Detaljan prikaz informacija za sliku LegacyTarget i prikaz njezinih značajki

3. UNITY

Unity game engine proizvod je tvrtke Unity Technologies [8] koja ga je javno objavila 2005. godine. Unity predstavlja višeplatformsko razvojno okruženje koja je prvobitno bilo namijenjeno isključivo za izradu računalnih igara i simulacija na računalu. Primjena u današnje vrijeme puno je šira. Uz već navedena područja/kategorije koristi se kao pomoćni alat za razvoj u: automobilskoj i filmskoj industriji, inženjerskim i graditeljskim projektima. [9]. Prva platforma (operacijski sustav) koja je bila podržana je OS X (*Apple Inc*) te je zatim uslijedila podrška za Windows i Linux operacijski sustav, te za brojne druge platforme. Danas Unity podržava preko 25 različitih platformi [10], što mu omogućava sve veću popularnost i razlog da baš on bude izabran od strane korisnika/programera. Uz navedene platforme, on omogućuje rad i razvoj igara ili bilo kojeg ranije navedenog proizvoda stvorenog kao: 2D, 3D, AR ili VR (*Virtual reality*[7]) okruženje. Ovaj game engine pisan je u C, C#, UnityScript, Boo i C++ programskom jeziku, pri čemu se za pisanje skripti koristi isključivo C# programski jezik. Sadrži brojne alate i sadržaje kao što su: Unity Asset Store, Ads, Everyplay, Analytics, Cloud Build, Certification te Vuforia kao najbitniji alat za ovaj rad. Nakon instalacije ovaj *engine* omogućava korisniku da izabere hoće li ga koristiti bez korisničkog računa (*offline*) ili s prijavljenim korisničkim računom (*online*). Prilikom stvaranja novog projekta, osim definiranja naziva projekta i mjesta spremanja, može se odabrati da to bude isključivo projekt za 2D ili 3D, kao i mogućnost dodavanja paketa i uključivanja Unity analitike. Kada je projekt napravljen otvara se prozor u kojem se mogu dodavati različiti objekti u sceni, dizajnirati sam izgled scene i brojne druge mogućnosti. Osim scene, postoji pod-prozor za pregled svih mapa (*Assets*) koji su potrebni za izradu projekta. To mogu biti mape s podacima za 3D modele koji se koriste u sceni, mape sa slikama, zvučnim efektima/melodijama, animacijama i slično. Bitno je naglasiti da osnovni element u sceni predstavlja objekt koji na sebi može imati različite komponente. Objekt se može pomicati bilo gdje u sceni, rotirati u svim smjerovima, može mu se povećati ili smanjiti veličina, itd. Kako bi objekti poprimili oblik i funkcionalnost za koju se koriste potrebno je osim vizualnih manipulacija koristiti skripte (jedna od komponenti) kojima se ostvaruje funkcionalnost i interakcija s drugim objektima u sceni. Pri korištenju i radu s objektom unutar scene s desne strane nalazi se prozor s karticom *Inspector* koji prikazuje bitne informacije o samom objektu kao i komponente koje su dodane na tom objektu. Neke osnovne informacije i opcije koje omogućuje su: definiranja imena objekta, nadimka, razine/sloja (*Layer*) u kojem se objekt treba nalaziti i koordinate po kojima se definira

[7] *Virtual reality – predstavlja virtualnu stvarnost, računalnu simulaciju odnosno virtualno okruženje u kojem se sudionik nalazi*

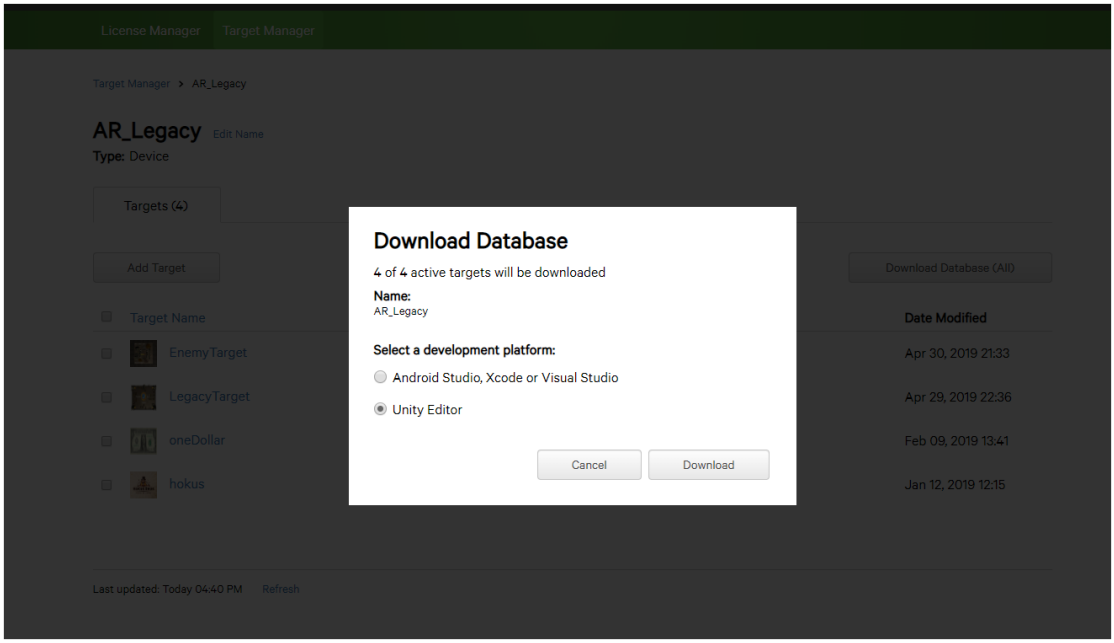
položaj objekta u prostoru. Moguće je dodavati i otvarati druge prozore u sceni koji su bitni za razvoj igre, a to su: *Console*, *Animator*, *Animation* i *Game*. Prvi navedeni prozor omogućuje ispis teksta koji može pomoći prilikom razvoja igre, pri čemu se može ispisivati vrijednost pojedine varijable. Drugi prozor služi za kontrolu animacija te se mogu postavljati različiti prijelazi iz jedne animacije u drugu ovisno o definiranim uvjetima. Više o načinu rada animatora biti će objašnjeno u potpoglavlju (4.9. Animacije i Animator). Zatim prozor za animacije služi za dodavanje i uređivanje animacija za odabrani objekt. Posljednji prozor *Game*, služi za prikaz scene igre kakvu bi igrač vidio prilikom igranja pri čemu je bitno da postoji barem jedna kamera u sceni. Ovaj prozor sadrži prvi vrhu više kartica koje se koriste za: definiranje rezolucije koja se koristi u igri (*Display*), povećanje prikaza scene (*Scale*), prikaz scene preko cijelog prozora editora (*Maximaze on Play*), mogućnosti isključenja zvuka u igri (*Mute Audio*), prikaz informacija o performansama (*Stats*) i *Gizmos* koji se može koristiti za prikaz i označavanje objekata koji se ne mogu vidjeti u Game kartici ili uredniku. Kako je već ranije spomenuto da se objekt kontrolira pomoću skripte sada će biti kratko objašnjeno što je osnova svake skripte i što Unity kao *engine* nudi. Svaka skripta se piše u C# programskom jeziku i svaka novostvorena skripta sadrži dvije glavne funkcije: *Start()* i *Update()*. Prva funkcija se koristi za definiranje početnih stanja varijabli i poziva se samo jednom prilikom stvaranja objekta u sceni. S druge strane, funkcija *Update()* izvršava se jednom po slici (*frameu*) i koristi se za izvršavanje koda koji se treba često pozivati kao što je na primjer kod za kontrolu igrača. Ukoliko takvo izvršavanje nije dovoljno brzo, postoji dodatna funkcija *FixedUpdate()* koja se izvršava svakih 0.02 sekunde i time se ne oslanja na vrijeme promjene slike. Svaku skriptu je posebno pisati u nekakvom uredniku skripti (*Code Editor*) i Unity po tom pitanju daje slobodu odabira. Postoji ugrađeni urednik skripti, MonoDevelop koji dolazi zajedno prilikom instalacije te je moguće promijeniti da se koristi bilo koji drugi urednik skripti kao na primjer Atom. Prilikom izrade ove igre korišten je Unity 2017.3.0f3, kao urednik skripti Visual Studio 2017 te isključivo C# kao programski jezik.

4. RAZVOJ IGRE

4.1. Ideja i Vuforia implementacija

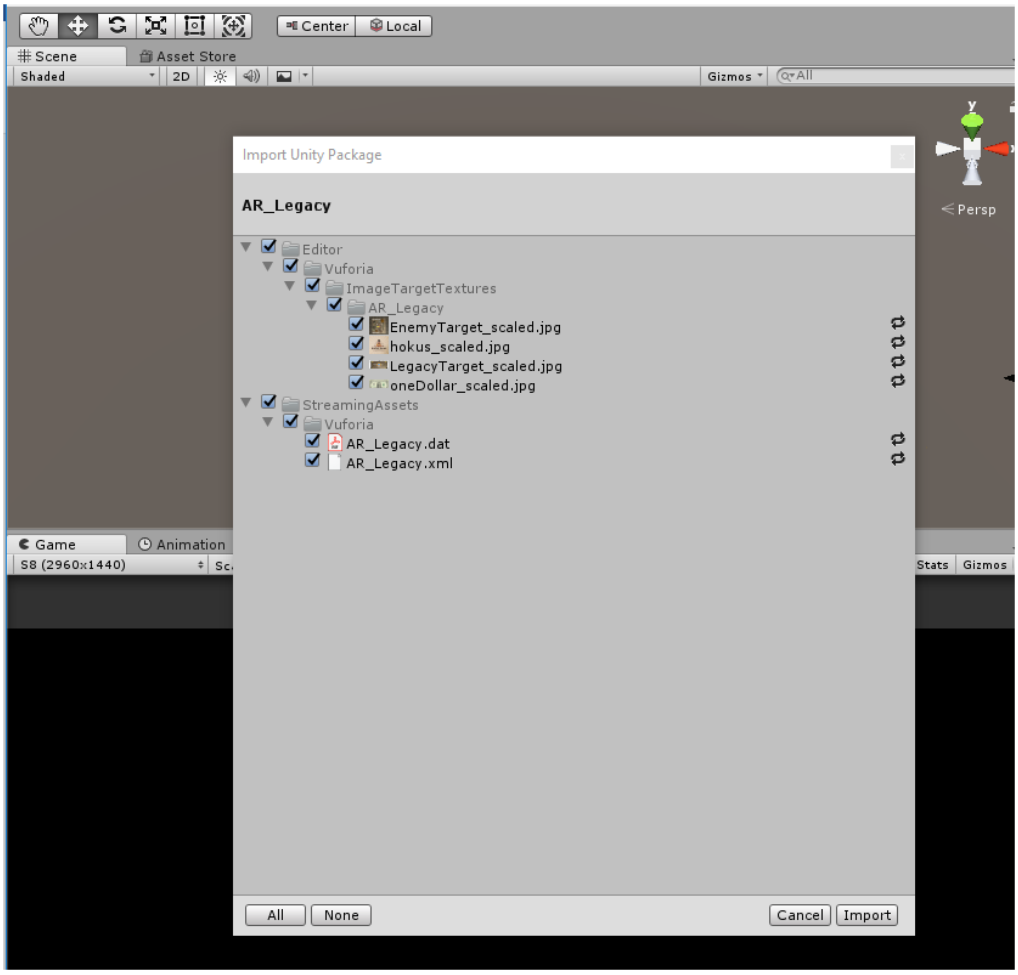
Temeljna ideja rada je izrada 3D računalne igre za mobilne uređaje s proširenom stvarnošću korištenjem Vuforia alata. Igra se sastoji od nivoa koji se može smjestiti bilo gdje u stvarnom svijetu s time da postoji meta (*Image Target*) na temelju koje se stvara scena. Cilj igre je obraniti kip s energijom koja predstavlja izvor života, a koju neprijatelji žele uništiti i osvojiti. Tok igre čine višestruki napadi neprijatelja (kosturi), pri čemu se na svaki novi val napada njihov broj povećava. Kada igrač ubije sve kosture preostaje još glavni *boss* nakon čijeg poraza igra završava pobjedom, a u suprotnom, porazom. Igrač neprijatelje ubija na način da puca iz jednog od četiri vrsta oružja (pištolj, automatska puška, strojnica i snajper). Postoje dva bitna kontrolna gumba s lijeve i desne strane koji se nalaze pri dnu korisničkog sučelja (ekrana). Gumb s lijeve strane služi za odabir oružja dok gumb s desne strane za pucanje iz oružja. Mehanizam ciljanja oružjem temelji se na fizičkom pomicanju mobilnog uređaja (gore-dolje, lijevo-desno) pri čemu se ciljnik nalazi u središtu ekrana.

Za implementaciju Vuforija alata unutar Unityja prvo je potrebno imati definiranu bazu kako je objašnjeno u prijašnjem poglavlju (2.1. Vuforia i Unity). Nakon definirane baze i odabranih slika potrebno je skinuti bazu pri čemu je potrebno odabrati Unity Editor za platformu kako je prikazano na slici ispod (Sl. 4.1.). Idući korak je implementiranje unutar samog Unity projekta.



Sl. 4.1. Preuzimanje baze definiranih meta/slika kao paket za Unity Editor

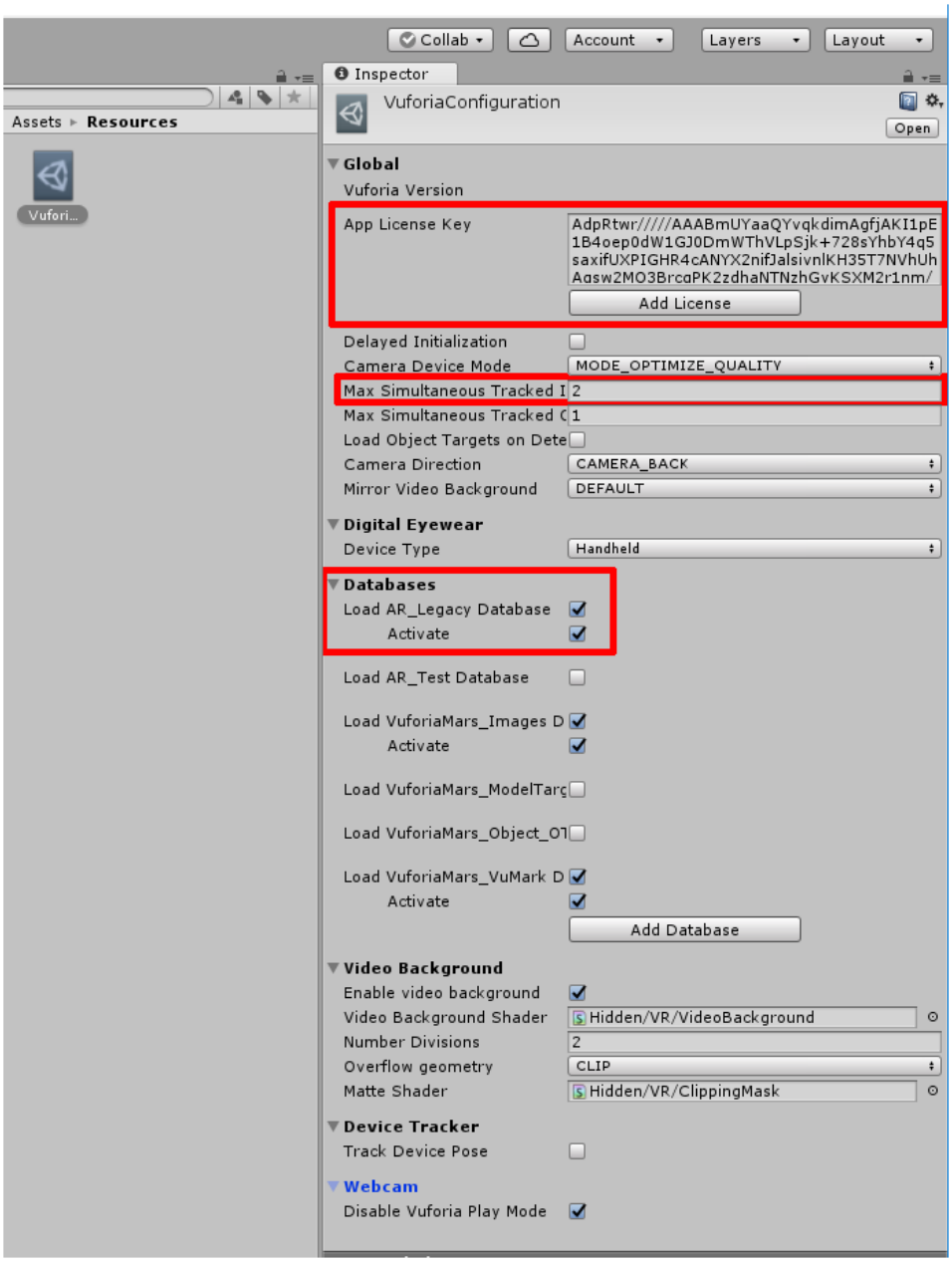
Nakon što se otvori projekt u Unity Editoru potrebno je unutar Asset Storea upisati Vuforija i unijeti paket (s kojim dolazi i AR kamera) kako bi se mogao koristi Vuforia alat unutar projekta. Sada slijedi korak u kojem se unosi ranije preuzeta baza u projekt koja sadrži slike (mete). Baza se unosi kao i svaki drugi paket koji nije predefiniran unutar Unityja (Assets - Import Package - Custom Package) pri čemu se odabire paket s računala. Prikaz kako izgleda kada se odabere paket može se vidjeti na slici koja slijedi (Sl 4.2.). Paket se sastoji od više mapa koje su generirane automatski prilikom preuzimanja s Vuforija Developer Portala koje sadrže mete i potrebne datoteke za rad.



Sl. 4.2. Prikaz prozora nakon odabranog AR_Legacy paketa za unos u projekt

Nakon unosa baze potrebno je napraviti konfiguraciju (Window – Vuforia Configuration). Prvo je potrebno dodati ključ za licencu i postaviti istovremeni broj praćenja slika, a za ovaj projekt taj broj je postavljen na 2 pošto će postojati dvije slike (jedna za stvaranje glavne scene, a druga za stvaranje jačih kostura). Sljedeće je potrebno odabrati koje će baze biti korištene i aktivne.

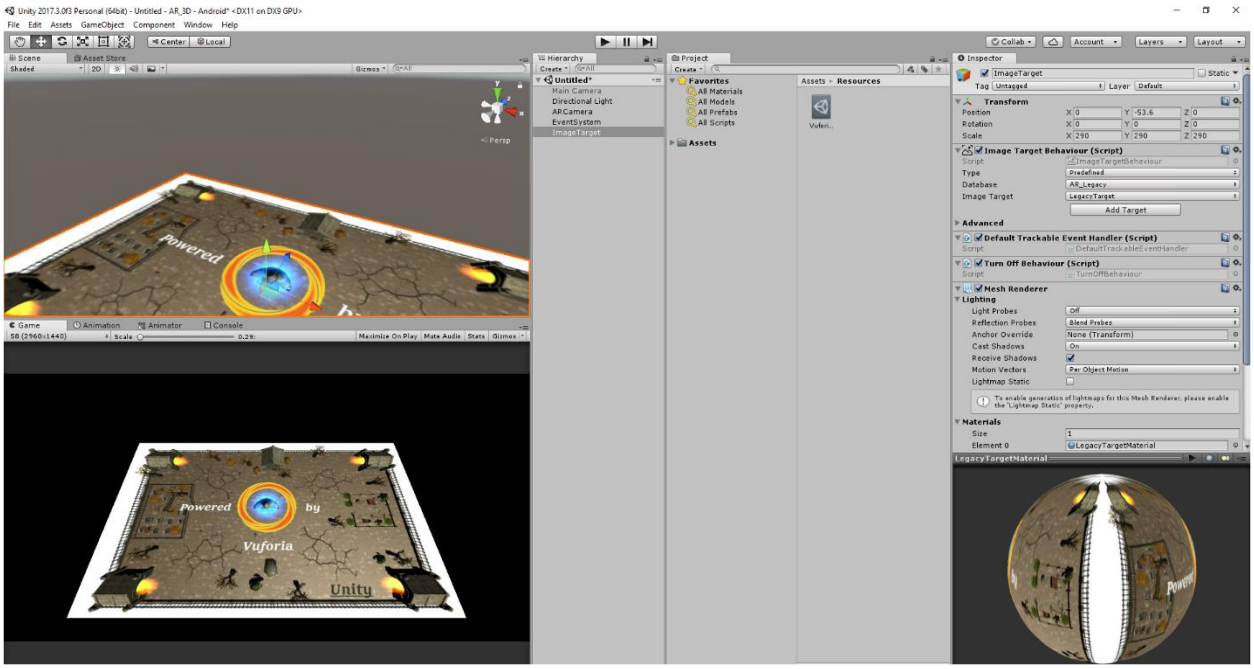
Navedene postavljene postavke na slici su uokvirene crvenom bojom dok su ostale postavke ostavljene na predefinirane vrijednosti. Postavljena konfiguracija nalazi se na slici (Sl. 4.3.).



Sl. 4.3. Prikaz postavljene Vuforia konfiguracije sa uokvirenim bitnim postavkama

Nakon složene konfiguracije potrebno je dodati Vuforia kameru (GameObject – Vuforia - ARCamera) te metu (*Image Target*) na temelju koje će se stvarati scena (Gameobject – Vuforia - Image). Objekt ImageTarget sadrži dvije predefinirane skripte: *Default Trackable Event Handler* i *Turn Off Behaviour*. Prva skripta sadrži metode koje detektiraju status meta, jesu li pronađene i u stanju praćenja ili su izgubljene, odnosno nisu pronađene. Druga skripta služi za kontrolu vidljivosti mete. Kada se radi u Unity Editoru (prilikom razvoja projekta) zbog ove skripte meta

će biti vidljiva u sceni kako bi se olakšala percepcija scene u odnosu na metu, dok će ova skripta sakriti metu prilikom pokretanja aplikacije na uređaju pošto se očekuje ista takva meta, ali u stvarnom okruženju (fizičkom prostoru). Ovaj objekt sadrži još jednu bitnu Vuforia skrptu, a to je *Image Target Behaviour* na temelju koje se postavlja baza i meta. Kako to izgleda kada su svi navedeni objekti dodani u sceni može se vidjeti na slici ispod (Sl. 4.4.).

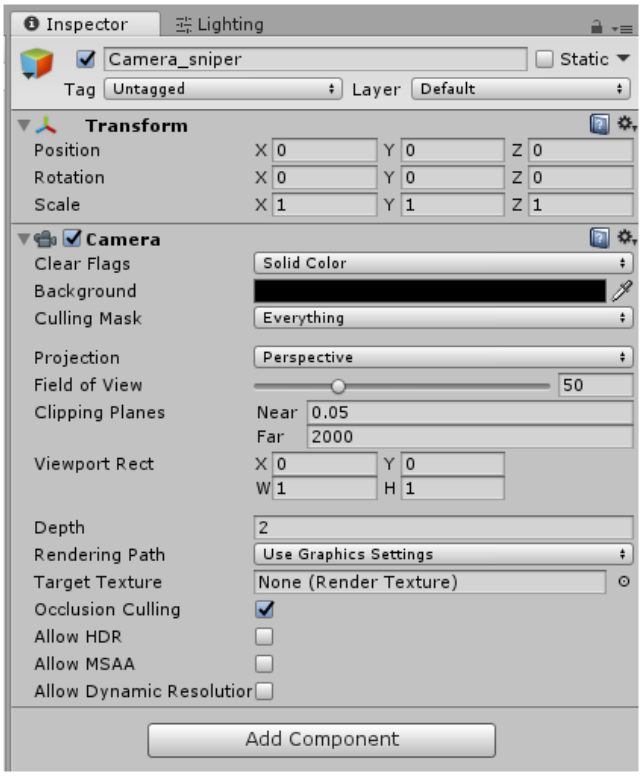


Sl. 4.4. Prikaz scene unutar koje je dodana meta i postavljena konfiguracija

Nakon ovih koraka implementirana je Vuforija unutar Unity projekta i ovaj alat je spreman za korištenje. Dalje se scena stvara dodavanjem različitih objekata na temelju mete kao i kod običnog 3D Unity projekta.

4.2. Kamera

Kamera predstavlja glavni objekt koji omogućuje vizualiziranje scene i u svakoj sceni je potrebna barem jedna kamera. Ona prikazuje scenu kakvu će vidjeti igrač tijekom igranja i zbog toga je jako bitna tijekom izrade scene. Unutar jedne scene može biti više kamera ovisno o potrebi. U ovom radu postoje najviše dvije kamere koje su aktivne istovremeno, a razlog je prikazivanje scene kada se koristi oružje snajper koje ima mogućnost ciljanja, približavanja i udaljavanja (*zoom in/out*), korištenje „optike“. Kamera kao i svaki drugi objekt može imati više skripti koje definiraju određeno ponašanje i mijenjaju postavke kamere u stvarnom vremenu. Osnovne postavke kamere su: Clear *Flags*, *Background*, *Culling Mask*, *Projection*, *Field of View*, *Clipping Planes* i *Depth*. Sada će biti redom objašnjene navedene postavke na temelju kamere za snajper kao konkretan primjer (Sl. 4.5.).



Sl. 4.5. Prikaz postavki za kameru koja se koristi za prikaz optike snajpera

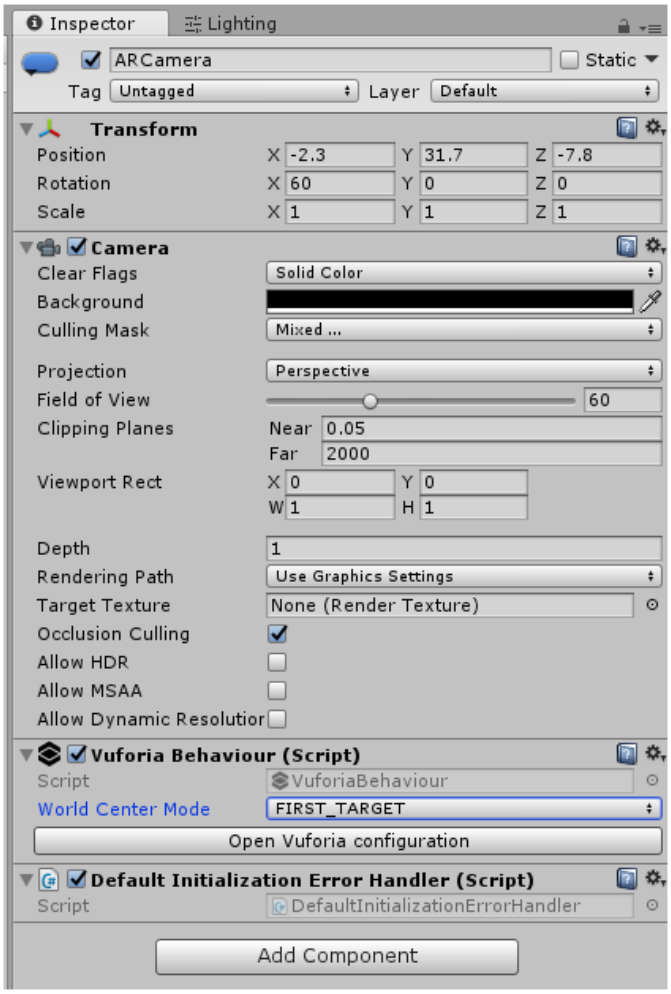
Prva postavka *Clear Flags* služi za definiranje okoline koja će se prikazivati u pozadini scene. Predefinirana vrijednost je *Skybox* i ukoliko je ona postavljena onda će okruženje biti kao da je nebo u pozadini koje je postavljeno unutar samog Unityja. Sa slike je vidljivo da je postavljeno na *Solid Color* čime je definirano da pozadina bude ono što vidi druga kamera. Ukoliko nema druge kamere onda će pozadina biti boje koja je definirana prema postavki *Background*. Iduća

postavka određuje boju pozadine koju je moguće mijenjati, *Background*. *Culling Mask* se koristi kako bi se odredio sloj (*Layer*) koji se želi prikazivati pomoću kamere (svakom objektu može biti postavljen različiti sloj, inicijalni je *Default*). Za ovu kameru postavljen je sloj na vrijednost *Everything* pri čemu je cilj da se prikazuju svi objekti u sceni. Sljedeća postavka je projekcija kojom se definira način prikazivanja. Postoje dva tipa projekcije: *Perspective* (za 3D) i *Orthographic* (za 2D). Projekcija je postavljena na *Perspective* pošto se radi o 3D igri te je za cilj prikazati i dubinu prostora, a ne samo širinu i visinu. *Filed of View* predstavlja vrijednost širine kuta koji kamera prikazuje (koristi se za efekt približavanja i udaljavanja unutar scene), a definirana je u stupnjevima (od 1 do 179). Vrijednost je 50 koja je određena testiranjem te se pokazala kao najbolja za ovu scenu. Kako bi se odredilo koliko najbliže i najdalje može prikazivati objekte u sceni koristi se *Clipping Planes*. Za najmanju udaljenost (*Near*) postavljena je vrijednost 0.05, dok za najdalju vrijednost 2000 pri čemu se ti brojevi definiraju kao mjerne jedinice koje se koriste unutar same scene Unityja (*Transform* vrijednosti, nisu niti metri niti centimetri). Kao zadnja bitna postavka *Depth* koristi se za definiranje poretka kamere koja će prikazivati scenu. Kamera koja ima veću vrijednost koristit će se za prikazivanje zbog toga je postavljena vrijednost na 2. Razlog je što se ova kamera koristi samo kod prikazivanja ciljanja pomoću snajpera pri čemu se *ARCamera* koristi za prepoznavanje mete i prikazivanje scene.

4.3. AR Kamera

Za razliku od prošle kamere ova kamera služi za prepoznavanje/detekciju mete (*Image Target*) na temelju koje će se stvoriti scena (proširena stvarnost). Kada se doda u sceni ona također sadrži komponentu *Camera* sa svim mogućnostima postavki kao i „obična“ kamera unutar Unityja (Sl. 4.5.). Glavna razlika je što ova kamera, objekt na sebi sadrži dvije inicijalne Vuforia skripte: *Vuforia Behaviour* i *Default Initialzation Error Handler*. Prva skripta služi za prepoznavanje mete dok druga za ispisivanje poruka o grešakama ukoliko se pojave. Jedina razlika u postavkama ove kamere u odnosu na „kameru za snajper“ su: *Field of View* i *Depth*. Kod ove kamere te vrijednosti iznose 60 za *Field of View* te 1 za *Depth* pošto se želi postići efekt da ova kamera uvijek prikazuje scenu. Dok se koristi optika snajpera, *ARCamera* prikazuje svu pozadinu dok se korištenjem „obične“ kamere omogućuje *zoom in/out* optike. Razlog zbog čega se koristi druga kamera za postizanje efekta približavanja/udaljavanja (pri čemu se mijenja vrijednosti postavke: *Field of View*) je taj što Vuforia ne dozvoljava da se vrijednost *Field of View* mijenja na AR kameri tijekom stvarnog vremena te dolazi da prikazivanja titrajuće slike što predstavlja izuzetno loše korisničko

iskustvo. Na ovaj način je ostvaren efekt korištenja optike snajpera, pri čemu se koristi Camera_sniper kao kamera kojoj se mijenja *Filed of View* te koja prva prikazuje scenu, pri čemu je *Clear Flags* postavljen na *Solid Color*, čime je prepušteno da pozadinu prikazuje *ARCamera*. Kako izgledaju postavke AR kamere može se vidjeti na slici ispod (Sl. 4.6.).



Sl. 4.6. Prikaz postavki za AR kameru pomoću koje se detektira meta (Image Target)

4.4. Problemi korištenja AR kamere i kako su riješeni

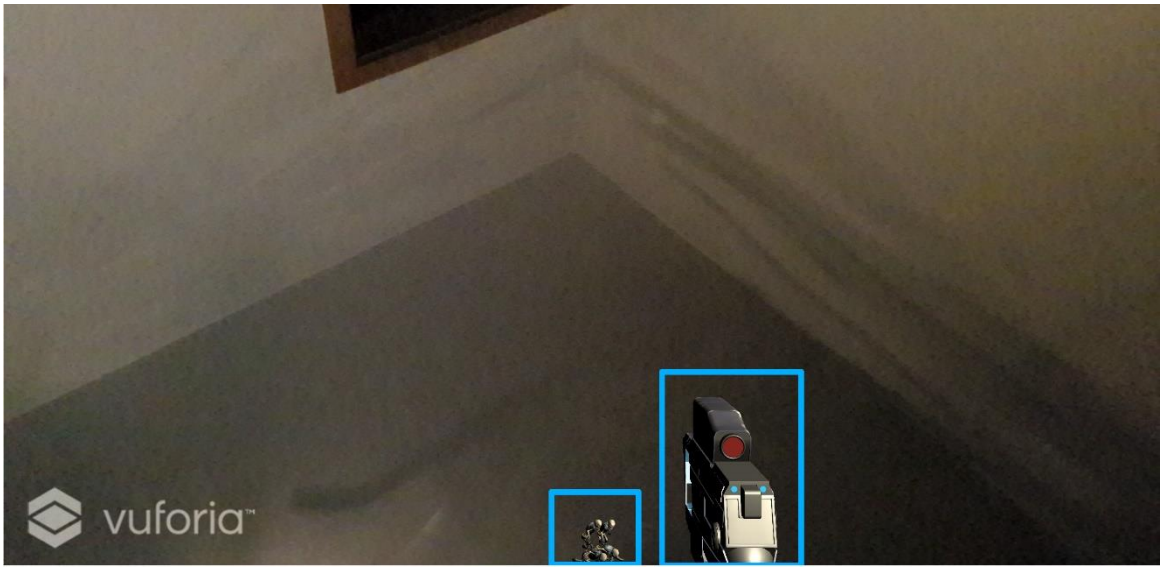
Neki od problema koji su nastali prilikom izrade ove igre su: definiranje položaja oružja iz prvog lica, objekti iz scene su prikazani i kada meta više nije u fokusu kamere mobilnog uređaja, položaj mete u stvarnom prostoru, korištenje više meta odjednom.

a) Položaj oružja

Kako bi se ostvario pogled, perspektiva iz prvog lica, potrebno je da položaj oružja bude na određenoj udaljenosti od kamere. Ukoliko bi se oružje pozicioniralo na iste koordinate u prostoru kao i AR kamera vidjela bi se samo cijev oružja što nije smisleno. Stoga je bilo potrebno pozicionirati svako oružje da se vidi kao da igrač „drži“ oružje. Iterativnim postupkom povećavanja udaljenosti oružja od kamere dobivena je željena perspektiva u prvom licu. Konačna udaljenost iznosi 8 jedinica u prostoru po Z osi.

b) Vidljivost objekata kada nema mete/slike (*Image Target*)

Prilikom prve detekcije slike scena bi se učitala bez nekih posljedica ili grešaka. Ukoliko bi se igrala igra sa stalnim fokusom kamere mobilnog uređaja prema meti ne bi bilo nikakvih neželjenih efekata. Ukoliko bi se maknuo fokus kamere objekti bi i dalje bili vidljivi na ekranu što nije poželjno. Kako izgleda primjer takvog efekta može se vidjeti na slici u nastavku (Sl. 4.7.). Rješenje je definiranje skripte ShowWeapon koja nasljeđuje ITrackableBehaviour interfejs.



Sl. 4.7. Prikaz postojanja objekata (oružja i kostura) kada je maknut fokus kamere sa mete

```
public class ShowWeapon : MonoBehaviour, ITrackableEventHandler {

    private TrackableBehaviour mTrackableBehaviour;
    public GameObject[] weapons;
    public bool isImageTargetDetected = false;

    void Start() {
    mTrackableBehaviour = GetComponent<TrackableBehaviour>();

    if (mTrackableBehaviour)
        mTrackableBehaviour.RegisterTrackableEventHandler(this);
    }


    public void OnTrackableStateChanged(TrackableBehaviour.Status previousStatus, TrackableBehaviour.Status newStatus)
    {
        if (newStatus == TrackableBehaviour.Status.DETECTED || newStatus == TrackableBehaviour.Status.TRACKED ||
newStatus == TrackableBehaviour.Status.EXTENDED_TRACKED)
        {
            weapons[0].SetActive(true);
            isImageTargetDetected = true;
        }
        else
        {
            if(weapons[0] != null)
                weapons[0].SetActive(false);
            isImageTargetDetected = false;
        }
    }

}
```

Sl. 4.8. Skripta ShowWeapon

Skripta iznad sadrži kod koji omogućuje detektiranje mete te manipulaciju vidljivosti objekata. Na početku se nalaze tri varijable. Prva varijabla *mTrackableBehaviour* je instanca klase *TrackableBehaviour* i koristi se za detekciju promjena događaja je li meta detektirana ili ne. Druga varijabla predstavlja polje objekata oružja, *weapons* unutar koje su definirana 4 vrsta oružja. Zadnja varijabla *isImageTargetDetected* je tipa *boolean* i predstavlja zastavicu kada je detektirana meta. Ova varijabla se dohvaća u svim drugim skriptama kod kojih objekti trebaju nestati kada se makne fokus kamere mobilnog uređaja s mete. Prva funkcija koja se poziva prilikom inicijalizacije klase je *Start()* unutar koje se nalazi kod za dohvaćanje *TrackableBehaviour* komponente i kod za provjeru ukoliko je već registriran rukovoditelj događajima za detekciju mete. Zatim slijedi javna funkcija *OnTrackableStateChanged()* koja sadrži dva argumenta: *previousState* i *newState* koji

predstavljaju prijašnje stanje i novo stanje. Unutar tijela funkcije slijedi provjera novog stanja, je li meta detektirana i ako je da se postavi prvo oružje vidljivo i da se postavi zastavica na *true*. U suprotnom, ukoliko nije detektirana meta neka se makne oružje iz scene i neka se postavi zastavica na *false*. Rezultat ove skripte može se vidjeti na slici pri čemu je još dodana vizualizacija traženja mete (Sl. 4.9.). Sada se može vidjeti da niti oružje, a niti kosturi nisu vidljivi izvan scene.

Sl. 4.9. Rezultat nakon korištenja skripte ShowWeapon i implementacije vizualizacije traženja mete

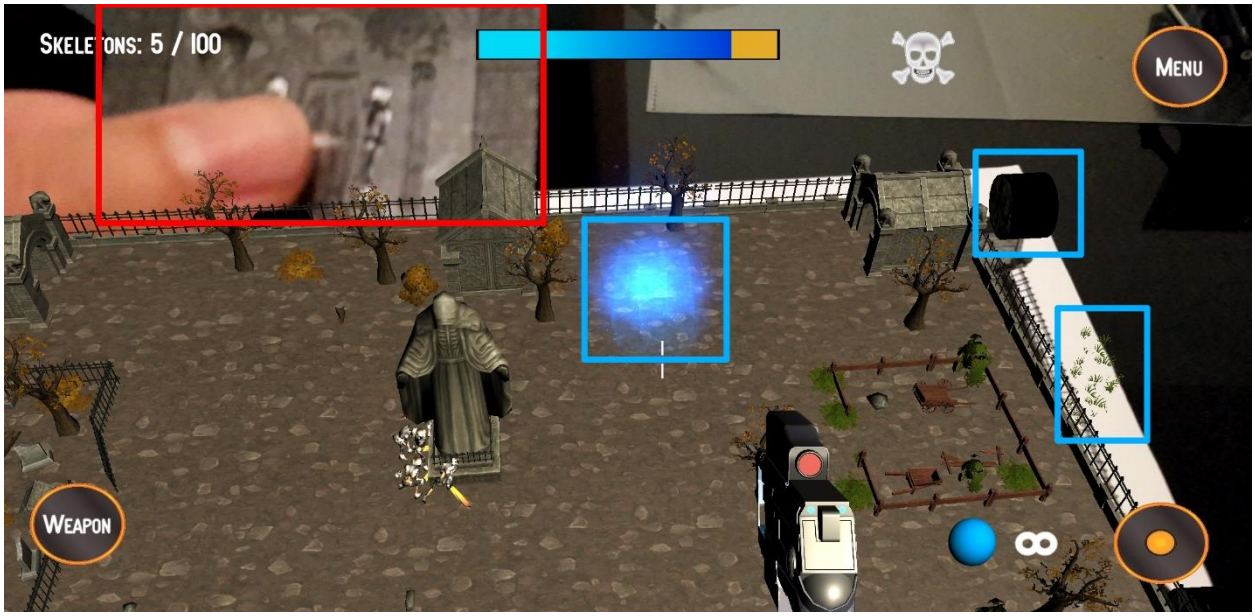
c) Položaj slike u stvarnom prostoru

Prilikom korištenja slike/mete (koja je isprintana na A4 papiru) bitno je da bude na ravnoj podlozi. Ako bi papir na kojem je meta bio iskrivljen, podignut, udubljen ili ukošen scena se ne bi pravilno prikazivala. Posljedica bi najčešće bila titranje scene, a kod većih deformacija mete i gubitak prikazivanja scene. Rješenje je korištenje ravne podloge poput stola pri čemu neće doći do iskrivljenja papira, a kao jedno od rješenja je i korištenje čvrste pozadine (npr. karton) na kojoj je zalijepljena meta. Kako izgleda kada je meta odnosno papir iskrivljen može se vidjeti na idućoj stranici pri čemu dolazi i do titranja cijele scene (Sl. 4.10.).

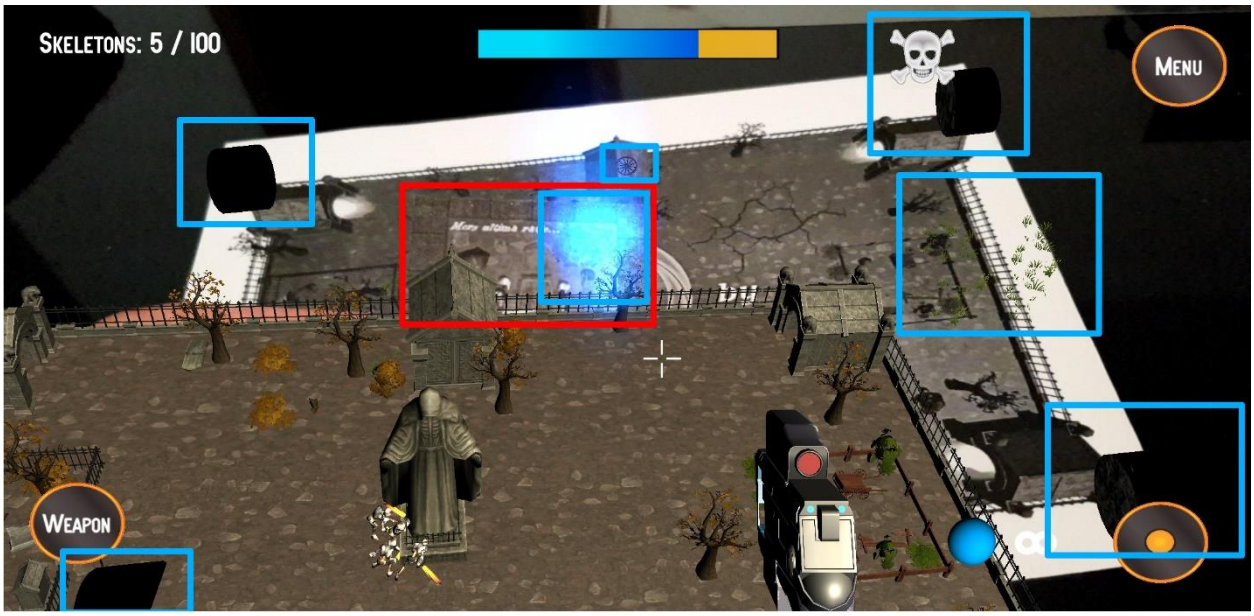
Sl. 4.10. Rezultat iskrivljenja slike/mete pri čemu dolazi do zakretanja cijele scene i titranja

d) Korištenje više meta odjednom

Kod korištenja dvije mete odjednom često je došlo do „lomljenja“ scene i mnogi bi objekti bili prikazani izvan scene. Rezultat se može vidjeti na slikama koje slijede (Sl. 4.11. i Sl. 4.12.), a crvenom bojom je uokvirena druga meta (EnemyTarget, na temelju koje se trebaju stvarati jači, napredniji kosturi) dok su plavom bojom uokvireni objekti koji su se pomaknuli ili izašli iz scene.

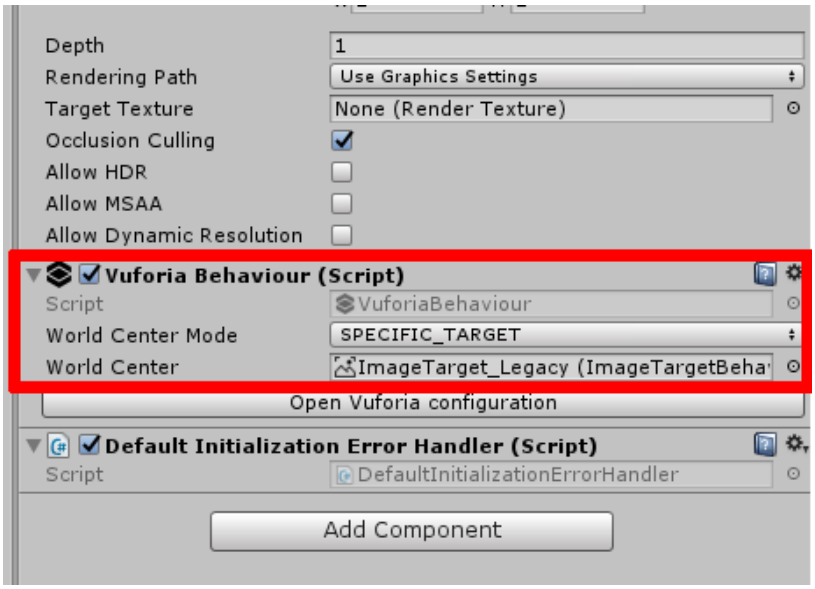


Sl. 4.11. Posljedica nepravilnog detektiranja dviju meta odjednom



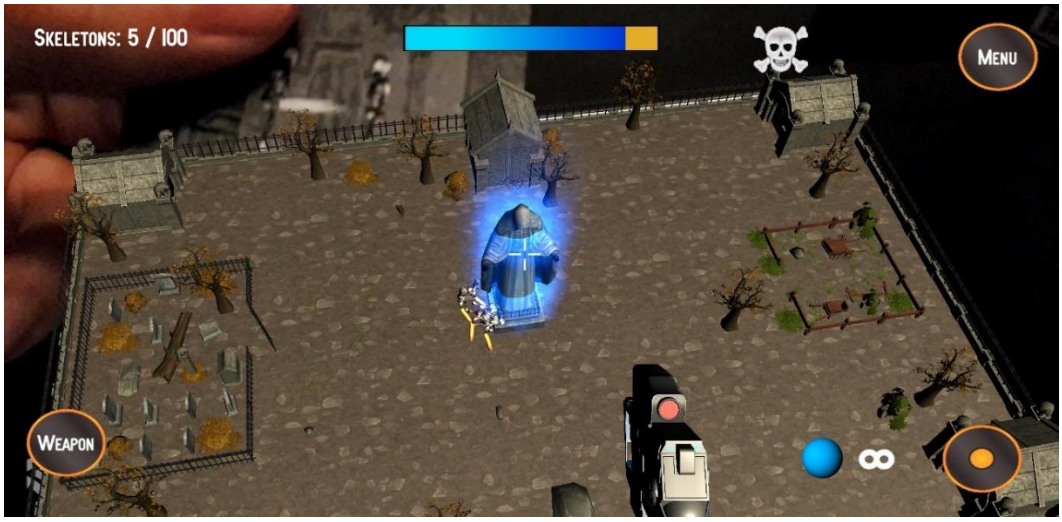
Sl. 4.12. Drugi primjer nepravilnog detektiranja dviju meta odjednom

Rješenje za ovakav problem vrlo je jednostavan, potrebno je samo promijeniti konfiguraciju AR kamere, pri čemu se postavke: *World Center Mode* i *World Centre* trebaju postaviti na *SPECIFIC_TARGET* i odabrati željenu metu kao „baznu“. Izmijenjena konfiguracija nalazi se na slici (Sl. 4.13.), dok se prvobitna konfiguracija može vidjeti na prijašnjoj slici (Sl 4.6.).



Sl. 4.13. Prikaz izmijenjene konfiguracije na AR kameri

Sada kada je izmijenjena konfiguracija više ne utječe druga meta na stvaranje scene na način da ima za posljedicu „lomljenje“ scene ili „ispadanje“ objekata iz scene. Ovom konfiguracijom je postavljena prva meta LegacyTarget kao „bazna/centralna“ i ukoliko je ova meta uspješno detektirana dolazi do prikazivanja scene u suprotnom se traži meta i ne može više doći do ranije spomenutih neželjenih efekata. Rezultat slično postavljene druge mete kao kod slike (Sl. 4.11.) gdje je prije došlo do ispadanja tri objekta sada se može vidjeti na slici ispod (Sl. 4.14.).



Sl. 4.14. Rezultat pravilnog prikazivanja scene nakon konfiguracije AR kamere unatoč velikoj blizini druge mete (EnemyTarget)

Sa slike iznad se sada jasno može vidjeti da su svi objekti unutar scene na svome mjestu i da druga meta koja je poprilično blizu scene nije negativno utjecala. Također je uspješno detektirana druga meta pri čemu je indikator, slika kosturske glave u desnom gornjem kutu. Ukoliko bi se druga meta nalazila jako blizu kameri mobilnog uređaja i dalje ne bi došlo do „lomljenja“ scene nego bi se aktivirao prikaz za učitavanje centralne mete (LegacyTarget). Primjer se nalazi ispod (Sl. 4.15.).

Sl. 4.15. Rezultat traženja centralne mete ukoliko bi se druga meta nalazila preblizu

4.5. Likovi

U igri postoje ukupno tri vrste neprijatelja, dvije vrste kostura i vođe/*bossa* (Diamond Lord). Za interakciju i kontrolu navedenih likova potrebno je da kao objekti u sceni imaju dodanu skriptu kojom će se to sve ostvariti. Također, skripte se koriste zajedno sa komponentom *Animator* kako bi se ostvarila kontrola i izvršavanje određenih animacija. U ovom potpoglavlju biti će objašnjene i skripte koje su ključne za pojedinog lika. Sada slijedi pregled pojedinih likova.

4.5.1. Igrač

Sl. 4.16. Prikaz igrača iz prvog lica s oružjem, kontrolama i ciljnikom u središtu ekrana

Igrač je postavljen u prvom licu, pri čemu vidi oružje koje koristi sa ciljnikom u sredini ekrana. S lijeva i desna nalaze se dva kontrolna gumba koja može koristiti. Lijevi gumb (*Weapon*) je za biranje oružja te igrač može birati između ukupno četiri vrste oružja (pištolj, automatska puška, strojnica i snajper). Na suprotnoj strani, nalazi se gumb koji služi za pucanje iz oružja, a lijevo do njega nalazi se indikator (plavi krug) za prikaz broja metaka. Na slici iznad (Sl. 4.16.) indikator za vrijednost ima znak za beskonačno pošto pištolj nema ograničenje za metke, dok preostala tri oružja imaju. Za pomicanje oružja lijevo-desno, gore-dolje potrebno je fizički pomicati mobilni uređaj, pri čemu se koristi ugrađeni senzor za žiroskop i akcelerometar [11]. Tijekom igre igrač ima ograničeni broj metaka na početku. Meci se stvaraju nasumično u sceni, a skupljaju se na način da ih igrač nacilja. U nastavku slijedi objašnjenje glavnih funkcija skripte

ShootingController. Zbog veličine skripte radi bolje preglednosti biti će podijeljena na tri pojedinačne slike.

```
public class ShootingController : MonoBehaviour {

    public float damage = 10f; // definiranje varijable za iznos jačine metka
    public float range = 1000f; // inicijalno definiranje raspona duljine pucanja

    public Camera fpsCamera; // definiranje objekta kamere

        public GameObject hitEffect, bossHitEffect; // definiranje objekata za efekte prilikom          pucanja

    public AudioSource[] shootSound; // definiranje polja zvučnih efekata oružja

    public ParticleSystem[] weaponEffect; // definiranje polja vizulanih efekata oružja
    [SerializeField]
        private GameObject impactEffect; // definiranje objekta za efekt prilikom pucanja u objekte

    // postavljanje oruzja
        public GameObject[] weapons; // definiranje polja koje sadrži objekte oružja, ukupno ih ima četiri
    // polje stringova unutar kojeg se nalaze četiri vrste oružja
    private string[] allWeapons = { "pistol", "rifle", "automatic", "sniper" };
        private string selectedWeapon = "pistol"; // definiranje početnog oružja koje se koristi, pištolj

    //victory/defeat
        private SculptureEnergy sculptureEnergy; // definiranje varijable na temelju koje će se dohvatiti
    // javne varijable iz klase SculptureEnergy
        private spawnCounter spawnCt; // definiranje varijable spawnCt na temelju koje će se dohvatiti
    // javne varijable iz klase spawnCounter

    private Animator crosshairAnimator; // definiranje varijable za kontrolu animatora

    //ammo
        private AmmoController ammoController; // definiranje varijable na temelju koje će se dohvatiti
    // javne varijable iz klase AmmoController
        public AudioSource ammoCollect; // definiranje varijable za zvučni efekt prilikom skupljanja metaka
        public GameObject ammoCollectText; // definiranje objekta koji sadrži tekst sa iznosom metaka
```

Sl. 4.17. Prvi dio skripte ShootingController

Na početku definiranja naziva klase potrebno je operatorom : naslijediti MonoBehaviour klasu kako bi se mogle koristiti ključne funkcije kao što su: *Start()*, *Update()*, *OnEnable()*, *OnDisable()*, itd. Prvi dio skripte sadrži javne i privatne varijable koje definiraju svojstva kao što su: jačina metka, kamera na temelju koje je definirano ciljanje, objekti oružja, pristupanje animatoru, itd. Komentarima je objašnjena uloga svake pojedine varijable. Sada slijedi drugi dio koda u kojem se definirane funkcije *Start()* i *Update()*.

```
// funkcija koja se poziva samo jednom prilikom incijalizacije klase
void Start() {
            // definirane varijable unutar koje se sprema pronađen objekt koji sadrži
            // određenu skriptu, kao što je za prvu varijablu skripta SculptureEnergy
            sculptureEnergy = FindObjectOfType<SculptureEnergy>();
            spawnCt = FindObjectOfType<spawnCounter>();
            ammoController = FindObjectOfType<AmmoController>();
}
```

```
// funkcija koja se poziva jednom po sličici / frame-u
void Update() {
        // provjera je li došlo do poraza ili pobjede da se onemogući
        // skupljanje metaka
        if (!sculptureEnergy.GetDefeatFlag || !spawnCt.GetVictoryFlag)
                AmmoCollection(); // pozivanje funkcije za skupljanje metaka
}
```

Sl. 4.18. Drugi dio skripte ShootingController

Prva funkcija *Start()* poziva se samo jednom prilikom stvaranja objekta u sceni i sadrži tri ranije definirane varijable unutar kojeg se spremaju objekti prema definiranim skriptama. Nakon uspješnog pronalaska navedenih objekata moguće je pristupati javnim varijablama i metodama koje te skripte sadrže. Ovim načinom postiže se upravljanje drugim skriptama (izmjena vrijednosti varijabli i pozivanje metoda) iz jedne skripte, vrsta odnosa roditelj-dijete (parent-child) odnosno način nasljeđivanja (referenciranje). Druga funkcija *Update()* poziva se na promjenu sličice (framea) na ekranu. Unutar tijela ove funkcije postoji jedno *if* grananje (provjera) kod kojeg se izvršava provjerava. Ukoliko je varijabla *GetDefeatFlag* iz skripte SculptureEnergy postavljena na istinu (*true*) ili druga varijabla *GetVictory* također postavljena na istinu više se neće pozivati metoda *AmmoCollection(),* u suprotnom se svakim pozivanjem funkcije *Upadate()* poziva i navedena metoda. Kao što se može vidjeti sa slike (Sl. 4.18.), za pristupanje javnim varijablama drugih skripta potrebno je koristiti operator "." (točka) i zatim slijedi naziv varijable ili metode. Na stranici koja slijedi nalazi se treći dio koda (Sl. 4.19.) koji sadrži funkciju *Shoot()* koja prima dva argumenta. Prvi argument je cjelobrojnog tipa i predstavlja indeks oružja, a drugi argument je istog tipa, ali predstavlja indeks zvučnog efekta oružja. Ova metoda služi za pucanje i detekciju neprijatelja kada igrač pogodi. Za detekciju objekta koristi se sturuktura *RaycastHit* koja sadrži informacije o objektu koji se pogodi. Ključna metoda koji se koristi unutar funkcije *Shoot()* je *Raycast()* koja se poziva iz klase Physics, a omogućuje detekciju pogođenog objekta. Ukoliko je neki objekt pogođen vraća vrijednost *true*. Za parametre prima izvor „zrake" što predstavlja kamera igrača kao i drugi argument smjer zrake. Treći argument je varijabla tipa *RaycastHit,* a četvrti argument je domet „zrake". Više o načinu rada ove metode komentirano je u kodu na sljedećoj stranici.

```
 // funkcija kojom se definirna pucanje iz oružja, prima dva argumenta

// prvi argument index je cjelobrojnog tipa i odnosi se na index oružja iz polja varijable weapons
public void Shoot(int index, int soundIndex)
{
   // provjera ukoliko nije doslo do pobjede/gubitka i broj metaka je veci od nule ili
   // je izabrano oružje pištolj onda se izvršava kod unutar if provjere
```

```
        if ((!sculptureEnergy.GetDefeatFlag && !spawnCt.GetVictoryFlag && ammoController.ammunition > 0) ||
        selectedWeapon == "pistol")
        {   // provjera da se ne aktivira animaciju pucanja nišana kada je sniper ~ tada je ciljnik isključen
    if (selectedWeapon != "sniper" && crosshairAnimator != null)
        crosshairAnimator.Play("shooting", 0);

    weaponEffect[index - 1].Play(); // pokretanje efekta pucanja iz oružja
    shootSound[soundIndex].Play(); // pokretanje zvučnog efekta pucanja iz oružja

    // provjera ukoliko odabrano oružje nije pištolj, umanjuje se broj metaka
    if (selectedWeapon != "pistol")
        ammoController.ammunition--;
    // provjera ukoliko je broj metaka jedak ili manji od nule da se postavi na nulu
    if (ammoController.ammunition <= 0)
        ammoController.ammunition = 0;

        RaycastHit hit; // definiranje varijable hit u kojoj će biti sadržane informacije strukture RaycastHit
    // koja se koristi u ovom slučaju za prepoznavanje objekta koji se pogodi
        // if grana za provjeru ukoliko je zraka pogodila objekt, pri čemu su postavljeni argumenti:
       // položaj kamere kao izvor zrake, smjer - također kamera, ali smjer po Z osi, treci argument je RaycastHit varijabla
      // za dohvaćanje informacije što je pogođeno i zadnji argument je udaljenost do koje zraka ide (Unity units)
     if (Physics.Raycast(fpsCamera.transform.position, fpsCamera.transform.forward, out hit, range))
{
      // definiranje varijable za svakog pojedinog neprijatelja na temelju zrake koja pogađa objekt
       // i dohvaćanje komponente - skripte pojedinog objekta
      SkeletonController skeletonTarget = hit.transform.GetComponent<SkeletonController>();
      Skeleton2Controller skeleton2Target = hit.transform.GetComponent<Skeleton2Controller>();
 BossController bossTarget = hit.transform.GetComponent<BossController>();
      // provjera ukoliko je barem jedna od varijabli ima vrijednost trazličitu od null - ne postoji
  if (skeletonTarget != null || skeleton2Target != null || bossTarget != null)
  {
            // ukoliko vrijednost varijable skeleton2Target je različita od null - znači da je pogođen objekt zrakom ~
            RaycastHit
   // i sadrži traženu komponentu izvršava se kod unutar tijela grananja
          // analogno se izvršava kod za preostale dvije varijable: bossTarget i skeletonTarget
    if (skeleton2Target)
     {
                 skeleton2Target.TakeDamage(damage); // pozivanje metode za samnjenje ukupnog života kosutra
                // stvaranje novog objekta s efektom pogotka, određenom pozicijom i "nultom" orijentacijom (0,0,0) po x,y,z,
                osi
                Instantiate(hitEffect, skeleton2Target.transform.position + skeleton2Target.transform.up * 1.5f,
                Quaternion.identity);
          }
          else if (bossTarget)
          {
             bossTarget.TakeDamage(damage);
                              Instantiate(bossHitEffect, hit.point, Quaternion.LookRotation(hit.normal));
    }
    else
    {
     skeletonTarget.TakeDamage(damage);
                 Instantiate(hitEffect, skeletonTarget.transform.position + skeletonTarget.transform.up * 1.5f,
                 Quaternion.identity);
          }
               } // ukoliko sve tri varijable imaju vrijednost null onda se stvara objekt s efektom: impacEffect
     else
     {
       Instantiate(impactEffect, hit.point, Quaternion.LookRotation(hit.normal));
     }
   }
       } // ukoliko nije odabran pištolj i nema metaka izvršava se kod za pokretanja zvučnog efekta za prazno oružje
 else
   shootSound[4].Play();
}
```

Sl. 4.19. Treći dio skripte ShootingController

Sada još preostaje četvrti dio skripte unutar kojeg su definirane dvije metode: *ShootOnButton()* i *AmmoCollection()*. Prvom metoda se poziva prilikom pritiska gumba za pucanje. Zatim slijedi niz provjera koje je trenutno odabrano oružje i na temelju odabranog oružja poziva se funkcija *Shoot()* s odgovarajućim parametrima. *AmmoCollection()* metoda koristi se za prikupljanje metaka i povećavanje ukupnog broja metaka. Ukupno postoje dvije skupine metaka: obična i bonus. Ako se sakupe obični meci tada taj broj iznosi 50 metaka, a ako bonus onda 100 metaka. Ova funkcija se poziva unutar metode *Update()*. Kako su definirane navedene metode može se vidjeti na slici (Sl. 4.20.).

```
// funkcija za pucanje prilikom pritiska gumba za pucanje
public void ShootOnButton()
{   // provjere odabranog oružja i pozivanje funkcije Shoot() s odogovarajućim parametrima
    if (selectedWeapon == "pistol")
        Shoot(1, 0);
    else if (selectedWeapon == "rifle")
        Shoot(2, 1);
    else if (selectedWeapon == "automatic")
        Shoot(3, 2);
    else if (selectedWeapon == "sniper")
        Shoot(4, 3);
}

// funkcija za skupljanje metaka
void AmmoCollection()
{
    RaycastHit hit; // definiranje varijable tipa RaycastHit

    // provjera ukoliko je zraka pogodila objekt
            if (Physics.Raycast(fpsCamera.transform.position, fpsCamera.transform.forward, out hit, range))
    {
        // definiranje varijable tipa Ammo i dohvaćanje objekta s istoimenom skriptom
        Ammo ammoTarget = hit.transform.GetComponent<Ammo>();
        AmmoBonus ammoBonusTarget = hit.transform.GetComponent<AmmoBonus>();
                    // ukoliko je naciljan objekt koji predstavlja metke izvršava se kod unutar if provjere
        // analogno se sve izvršava i za provjeru ispod osim što predstavlja bonus metke
        if (ammoTarget != null)
        {
            ammoCollect.Play(); // pokretanje zvučnog efekta skupljanja metaka
                        Destroy(ammoTarget.gameObject); // pozivanje metode za uništenje objekta, micanje iz scene
            ammoController.ammunition += 50; // povećanje ukupnog broja metaka za 50
                        ammoCollectText.GetComponent<Text>().text = "+50"; // ispis teksta količine skupljenih metaka
                        ammoCollectText.SetActive(true); // prikazivanje objekta koji sadrži iznad definiran tekst
        }
        else if (ammoBonusTarget != null)
        {
            ammoCollect.Play();
            Destroy(ammoBonusTarget.gameObject);
            ammoController.ammunition += 100;
            ammoCollectText.GetComponent<Text>().text = "+100";
            ammoCollectText.SetActive(true);
         }
       }
    }
}
```

Sl. 4.20. Četvrti, zadnji dio skripte ShootingController

4.5.2. Neprijatelji

Ukupno postoje tri vrste neprijatelja i u ovom potpoglavlju biti će objašnjeni svaki tip pojedinačno. Redom neprijatelji su:

a) Kosturi

Sl. 4.21. Prikaz modela dvije vrste kostura

Kosturi su vrsta neprijatelja koji su oklopljeni i za oružje koriste mač. Model je preuzet iz *Asset Storea* [12]. Kako je već ranije spomenuto, postoje dvije vrste kosutra: obični i napredni. Na slici iznad (Sl. 4.21.) na lijevoj strani se nalazi napredna vrsta kostura, dok s desne obična vrsta. Razlika između te dvije vrste je osim vizualne ta da napredna vrsta uvijek trči dok obična vrsta hoda ili trči. Također, napredniju vrstu potrebno je pogoditi 6 puta (osim snajperom koji predstavlja najjače oružje), dok su za običnu vrstu dovoljna 3 pogotka. Oni se stvaraju iz četiri takozvana *spawnera* koji su aktivni ovisno o trenutnoj rundi koja je u tijeku i ukupnom broju kosutra. Više o toku igre i *spawnerima* biti će objašnjenu u potpoglavlju 4.6. Tok igre i oružja. Obje vrste neprijatelja imaju skriptu kojom su kontrolirani. Napredni kosturi imaju skriptu naziva Skeleton2Controller, a obični SkeletonController. Skripte su gotovo jednake, ali postoje manje razlike koje definiraju kontrolu animacija i količinu životne energije. U nastavku slijedi prikaz i objašnjenje skripte SkeletonController. Slika ispod prikazuje prvi dio skripte i njezine definirane varijable.

```
public class SkeletonController : MonoBehaviour {

    public Transform[] targets; // definiranje polja varijabli tipa Transforms koje predstavlja mjesto napada
  private int index = 0; // varijabla koja definira na kojem mjestu će kostur napasti
    private NavMeshAgent navAgent; // definiranje varijable tipa NavMeshAgent koja se koristi za AI hodanje
    private Vector3 targetDestination; // definiranje varijable u koju će se spremiti položaj napada

  private Animator anim; // definiranje varijable tipa Animator za kontrolu animacija
    private float speed; // definiranje varijable tipa float za određivanje brzine kretanja kostura

  public float health = 30f; // definiranje varijable tipa float za ukupnu energiju kosutra
    private AudioSource hurt; // definiranje varijable tipa AudioSource koja sadrži zvučni efekt

  private float turnSpeed = 5f; // definiranje varijable za brzinu okreta kostura
    public GameObject capColliders; // definiranje varijable tipa objekta koja se koristi za aktivaciju/deaktivaciju svih collider-
    a

    public int firstAnimation; // definiranje cjelobrojne varijable kojom se određuje prva animacija
    private bool isHurt = false; // definiranje varijable tipa boolean kao zastavica je li kostur pogođen

  //imageTarget za vidljivost
  private ShowWeapon showWeapon; // definiranje varijable tipa klase ShowWeapon
  public GameObject body, weapon; // definiranje objekata za tijelo i oružje kostura

  //defeat
    private SculptureEnergy sculptureEnergy; // definiranje varijable tipa klase SculptureEnergy
  private spawnCounter spawnCounter; // definiranje varijable tipa klase spawnCounter

  public AudioClip deathClip, swordAttack; // definiranje zvučnih isječaka
```

Sl. 4.22. Prvi dio skripte SkeletonController

```
void Start () {
    navAgent = GetComponent<NavMeshAgent>(); // dohvaćanje komponente NavMeshAgent
    anim = GetComponent<Animator>(); // dohvaćanje komponente Animator
    index = Random.Range(0, 5); // postavljanje nasumične vrijednosti index-a između 0 i 5

        targetDestination = targets[index].position; // postavljanje vrijednosti mjesta na temelju index-a
    navAgent.SetDestination(targetDestination); // postavljanje odredišta napada kostura

    hurt = GetComponent<AudioSource>(); // dohvaćanje komponente AudioSource

        firstAnimation = Random.Range(0, 6); // postavljanje nasumične vrijednosti na temelju raspona
    // između 0 i 6 kojom će biti definirana prva animacija

        showWeapon = FindObjectOfType<ShowWeapon>(); // dohvaćanje objekta iz scene koji sadrži skriptu ShowWeapon

        sculptureEnergy = FindObjectOfType<SculptureEnergy>(); // dohvaćanje objekta koji sadrži skriptu SculptureEnergy
        spawnCounter = FindObjectOfType<spawnCounter>(); // dohvaćanje objekta koji sadrži skriptu spawnCounter

  }

  // funkcija koja umanjuje životnu energiju kosutra (HP)
  public void TakeDamage(float amount)
  {
    isHurt = true; // postavljanje zastavice da je kostur pogođen
    navAgent.isStopped = true; // definiranje da kostur stoji
    navAgent.speed = 0f; // postavljanje brzine kretnje kostura na 0

    anim.Play("Hurt", 0); // pokretanje animacije za ozljedu iz prvog sloja animatora
    hurt.Play(); // pokretanjue zvučnog efekta
    health -= amount; // smanjenje životne energije za definiran iznos - amount

    if (health <= 0) // provjera je li životna energija manja ili jednaka 0
    {
```

```
            capColliders.SetActive(false); // gašenje svih collider-a koji se nalaze na kosturu
        anim.Play("Death", 0); // pokretanje animacije za smrt, prvi sloj animatora
        hurt.volume = 1f; // postavljanje vrijednosti zvuka na 1f - 100%
        hurt.clip = deathClip; // postavljanje novog zvučnog isječka - deatClip
        hurt.Play(); // pokretanje novopostalvjenog zvučnog efekta
        StartCoroutine(Death()); // pokretanje coroutine koja izvršava uništenje objekta
    }
}

// funkcija tipa IEnumerator koja omogućuje kontrolu izvođenja koda
// ovisno o vremennu
IEnumerator Death()
{   // definiranje akcije da se odgodi izvršavanje koda koji slijedi
    // za 2.4 sekunde
    yield return new WaitForSecondsRealtime(2.4f);
    Destroy(gameObject); // uništavanje objekta, kostura iz scene
}

// funkcija za smanjene energije kipa
public void DealDamage()
{
    sculptureEnergy.energyCurrentState -= 2f;
}
```

Sl. 4.23. Drugi dio skripte SkeletonController

Glavni dio kontrole kostura nalazi se u funkciji *Update()*. Na početku se provjerava koliko životne energije ima kostur te je li došlo do poraza igrača. Ako je, izvršava se kod unutar prve if grane kojim se definira da kostur stane i da se prestanu izvoditi navedene animacije. Ukoliko nije zadovoljen prvi uvjet prelazi se na drugu granu *else if* pri čemu se provjerava udaljenost kostura od mjesta napada. Uvjet je da udaljenost mora biti manja ili jednaka 2 te da je životna energija kostura veća od 0. Tek tada se ulazi u izvršavanje ove grane pri kojoj se kostur zaustavlja i aktivira se animacija za napad. Zatim slijedi još jedno grananje s razlikom je li udaljenost veća od 2. Posljedica ispravnosti uvjeta je da se izvršava kod koji postavlja animaciju napada na *false* i provjerava se broj varijable za animaciju. Ako je broj veći od 3 izvršava se animacija trčanja i postavlja vrijednost kretanja kostura na 5, ali ako je broj manji do 4 izvršava se animacija hodanja i postavlja brzina kretanja kostura na 2. Zatim, neovisno o uvjetima uvijek se izvršava kod za rotaciju kostura prema meti unutar scene (kipu). To se ostvaruje izračunavanjem razlike položaja između kipa i kostura. Nakon toga, izvršava se kod u kojem se provjerava je li meta (*Image Target*) LegacyTarget detektirana. To se provjerava na način da se ispituje vrijednost *boolean* varijable *isImageTargetDetected* iz skripte *ShowWeapon*. Ukoliko je uvjet istinit, meta je detektirana, tijelo i oružje kostura postavlja se da je vidljivo u sceni, u suprotnom se deaktiviraju (ne prikazuju se u sceni) ti dijelovi kostura. Time je riješen problem koji je spomenut ranije u potpoglavlju (4.4. Problemi korištenja AR Kamere i kako su riješeni, problem pod b)). Na kraju se još provjerava je li igrač pobijedio i ako je uvjet istinit izvršava se kod kojim se pokreće animacija za smrt i *coroutina* za uništenje objekta iz scene. Opisani dio koda nalazi se na ispod (Sl. 4.24.).

```
    void Update () {
        // provjera ukoliko kostur ili skulptura ima nula ili manje životne energije
        if (health <= 0 || sculptureEnergy.GetDefeatFlag)
        {
            navAgent.speed = 0f; // postavlanje brzine kretanja kosutra na 0
                        anim.SetBool("walk", false); // postavljanje vrijednosti animacije hodanja na false
            anim.SetBool("run", false); // postavljanje vrijednosti animacije trčanja na false
            anim.SetBool("attack", false); // postavljanje vrijednosti napadanja na false

                } // ukoliko je udaljenost do mjesta za napad manja ili jednaka od 2 i životna energija veća od 0
        else if (navAgent.remainingDistance <= 2f && health > 0)
        {
          navAgent.isStopped = true; // definiranje da kostur stoji
                    anim.SetBool("walk", false); // postavljanje vrijednosti animacije hodanja na false
          anim.SetBool("run", false); // postavljanje vrijednosti animacije trčanja na false
          anim.SetBool("attack", true); // postavljanje vrijednosti napadanja na true
              } // ukoliko je udaljenost do mjesta za napad veća ili jednaka od 2 i životna energija veća od 0
        else if (navAgent.remainingDistance > 2f && health > 0)
        {
            anim.SetBool("attack", false); // postavljanje vrijednosti napadanja na false
            // provjera broja prve animacije je li veća od 3 i da pritom kostur nije pogođen
            if (firstAnimation > 3 && !isHurt)
            {
             navAgent.speed = 5f; // postavljanje brzine kretanja kostura na 5
             anim.SetBool("run", true); // postavljanje vrijednosti animacije trčanja na true
            }
            else if(firstAnimation < 4 && !isHurt)
            {
             navAgent.speed = 2f;
             anim.SetBool("walk", true); // postavljanje vrijednosti animacije hodanja na true
            }
        }
              // definiranje varijable tipa strukture Quaternion na temelju koje se kontrolira rotacija kostura
        // LookRotation() metoda se koristi za zakretanje kosutra prema skulpturi koju napada
              Quaternion target_Rotation = Quaternion.LookRotation(targets[index].transform.position - transform.position);
        target_Rotation.x = 0f; // postavljanje rotacije na x osi na vrijednost 0
        target_Rotation.z = 0f; // postavljanje rotacije na y osi na vrijednost 0

              // definiranje rotacije objekta pri čemu se koristi metoda Lerp() koja prima 3 parametra
              // pri čemu se radi zakretanje između kosutra (objekta a), mete (objekt b) i definiranom brzinom u vremenu - turnSpeed
              transform.rotation = Quaternion.Lerp(transform.rotation, target_Rotation, turnSpeed * Time.deltaTime);
        // provjera ukoliko je meta LegacyTarget pronađena
        if (showWeapon.isImageTargetDetected)
        {
            body.SetActive(true); // postavljanje tijela kostura vidljivim u sceni
            weapon.SetActive(true); // postavljanje oružje kostura vidljivim u sceni
        }
        else // u suprotnom meta nije pronađena
        {
            body.SetActive(false); // postavljanje tijela kostura nevidljivim u sceni
            weapon.SetActive(false); // postavljanje oružje kostura nevidljivim u sceni
        }
        // provjera ukoliko je igrač pobjedio, porazio bossa
        if (spawnCounter.GetVictoryFlag)
        {
            capColliders.SetActive(false); // deaktiviranje svih collider-a na kosturu
            anim.Play("Death", 0); // pokretanje animacije za smrt, 0 - bazni sloj animatora
            StartCoroutine(Death()); // pokretanje coroutine za smrt kojom se uništava objekt
        }
    }
}
```

Sl. 4.24. Treći, zadnji dio skripte SkeletonController

b) Diamond Lord (*boss*)

Sl. 4.25. Prikaz modela vođe, Diamond Lord

Diamond Lord predstavlja vođu neprijatelja koji se stvara u sceni kada se poraze svi kosturi koji su stvoreni iz *spawnera*. Model je besplatan i preuzet je također iz *Asset Storea* [13]. Kao napad koristi ruke („kanđe“) i ono što ga čini jakim je mogućnost regeneriranja tri puta. Ukoliko mu vrijednost energije padne ispod 30% počinje proces regeneriranja pri čemu poziva napredne kosture da mu pomognu (stvaraju se iz „zemlje“ unutar scene). Ukoliko ga igrač porazi, zajedno s njim umiru i svi kosturi koji su stvoreni te je time definirano da je igrač pobijedio. Kako su ostvarene navedene posebnosti vođe i njegova kontrola biti će prikazano kodom ispod (Sl. 4.26.). U prvom dijelu skripte nalaze se definicije svih potrebnih varijabli.

```
public class BossController : MonoBehaviour
{
    public float health = 300f; // definiranje varijable ukupne energije života na 300
    private Animator anim; // definiranje varijable za kontrolu animatorom
        private SculptureEnergy sculptureEnergy; // definiranje varijable tipa klase SculptureEnergy
        private NavMeshAgent navAgent; // definiranje varijable tipa NavMeshAgent koja se koristi za AI hodanje

        public Transform[] targets; // definiranje polja varijabli tipa Transforms koje predstavlja mjesto napada
    private int index; // varijabla koja definira na kojem mjestu će boss napasti
        private Vector3 targetDestination; // definiranje varijable u koju će se spremiti položaj napada

    public GameObject colliders; // definiranje objekta koji predstavlja collider-e boss-a

    //imageTarget za vidljivost
    private ShowWeapon showWeapon; // definiranje varijable tipa klase ShowWeapon
    public GameObject body; // definiranje objekata za tijelo boss-a

        public bool bossDead = false; // definiranje varijable tipa boolean kao zastavica je li boss poražen
        public Image healthImage; // definiranje varijable tipa Image za prikaz životne energije boss-a
        private float currentHealth; // definiranje varijable tipa float za definiranje trenutne energije
```

```
public int timeToHeal = 0; // definiranje brojača za brojanje obnavljanja energije
    private bool isBossHeal = false; // definiranje varijable tipa boolean kao zastavica je li se boss trenutno oporavio

    public HelpSpawner[] helpSpawner; // definiranje varijable tipa klase HelpSpawner koji se koristi za stvaranje
// novih kostura, kao pomoć boss-u

    public GameObject healthBar; // definiranje objekta kojim se aktivira/deaktivira prikaz energije na ekranu
    public bool bossIsSpawned = false; // definiranje varijable tipa boolean kao zastavica je li boss stvoren u sceni

// definiranje različitih zvučnih efekata
public AudioSource roarSound, deathSound, spawnSound, spawnBossRoar, attackSound;
    private bool canStart = false; // definiranje varijable tipa boolean koja služi kao zastavica može li
// se izvoditi određeni kod unutar funkcije Update()
```

Sl. 4.26. Prvi dio skripte BossController

Drugi dio skripte sadrži funkcije *Start()*, *OnEnable()*, *OnDisable()* i *Update()* koje su potrebne za kontrolu vođe. U prvoj funkciji se postavljaju početne vrijednosti varijabli i dohvaćaju se potrebni objekti, skripte. Zatim slijede dvije funkcije *OnEnalbe()* i *OnDisable()* kojima se kontrolira prikaz energije i je li stvoren *boss*. Unutar funkcije *Update()* nalazi se kod koji se definiraju potrebne animacije i upravljanje. Kod je sličan kao i kod skripte *SkeletonController*, znatnija razlika je to što se ovisno o vrijednosti energije *boss* može tri puta regenerirati i pozvati stvaranje kostura. Kako je to ostvareno može se vidjeti u kodu koji slijedi (Sl. 4.27.).

```
void Start()
  {
    anim = GetComponent<Animator>(); // dohvaćanje komponente Animator
          sculptureEnergy = FindObjectOfType<SculptureEnergy>(); // dohvaćanje objekta koji sadrži skriptu SculptureEnergy
    navAgent = GetComponent<NavMeshAgent>(); // dohvaćanje komponente NavMeshAgent

    index = Random.Range(0, 2); // postavljanje nasumične vrijednosti index-a između 0 i 2
          targetDestination = targets[index].transform.position; // postavljanje vrijednosti mjesta na temelju index-a
    navAgent.SetDestination(targetDestination); // postavljanje odredišta napada kostura
          showWeapon = FindObjectOfType<ShowWeapon>(); // dohvaćanje objekta iz scene koji sadrži skriptu ShowWeapon

                currentHealth = health / 300f; // definiranje početne vrijednosti energije healthImage.fillAmount = currentHealth;
          // definiranje vizualnog prikaza količine energije
  }
  // funkcija koja se poziva kada objekt postane aktivan u sceni
  private void OnEnable()
  {
    healthBar.SetActive(true); // vidljiv prikaz energije
    bossIsSpawned = true; // postavljanje zastavice da je boss stvoren
    StartCoroutine(SpawnBoss()); // pozivanje coroutine za stvaranje boss-a
  }
  // funkcija koja se poziva kada se objekt deaktivira iz scene
  private void OnDisable()
  {
    healthBar.SetActive(false); // prikaz energije se više ne prikazuje u sceni
  }

  void Update()
  {
    if (canStart) // provjera zastavice da se može izvoditi kod
    {  // provjera je li energija boss-a manja ili jednaka od 0 ili je igrač izgubio
      if (health <= 0 || sculptureEnergy.GetDefeatFlag)
      {
```

```
 navAgent.speed = 0f; // postavljanje brzine kretnje boss-a na 0
            anim.SetBool("attack", false); // postavljanje vrijednosti animacije napada na false
            anim.SetBool("walk", false); // postavljanje vrijednosti animacije hodanja na false
 anim.SetBool("win", true); // postavljanje vrijednosti animacije pobjede na true
 // dodatna provjera je li igrač izgubio
 if (sculptureEnergy.GetDefeatFlag)
                StartCoroutine(DelayCelebrate()); // pokretanje coroutine za slavljenje pobjede boss-a
            } // provjera je li udaljenost manja ili jednaka od 3.5 i je li energije boss-a veća od 0
else if (navAgent.remainingDistance <= 3.5f && health > 0)
        {   // provjera je li vrijednost energije manja ili jednaka od 70 i koliko se je puta regenerirao boss
   if (health <= 70f && timeToHeal < 3)
   {
      navAgent.isStopped = true; // definiranje da boss stoji
                anim.SetBool("walk", false); // postavljanje vrijednosti animacije hodanja na false
                anim.SetBool("attack", false); // postavljanje vrijednosti animacije napada na false
      anim.Play("relax", 0); // postavljanje vrijednosti regeneriranja na true
   }
   else // ukoliko je energija veća od 70
   {
      navAgent.isStopped = true; // definiranje da boss stoji
                anim.SetBool("walk", false); // postavljanje vrijednosti animacije hodanja   na false
                anim.SetBool("attack", true); // postavljanje vrijednosti animacije napada na true
   }
} // provjera je li udaljenost veća od 3.51 i energija veća od 0
else if (navAgent.remainingDistance >= 3.51f && health > 0)
{   // slijedi provjere koji je isti kao i u lese if grani iznad
   if (health <= 70f && timeToHeal < 3)
   {
      navAgent.isStopped = true;
      anim.SetBool("walk", false);
      anim.SetBool("attack", false);
      anim.Play("relax", 0);
   }
   else
   {
      navAgent.isStopped = false; // definiranje da boss više ne stoji
                anim.SetBool("attack", false); // postavljanje vrijednosti animacije napada na false
      navAgent.speed = 2.5f; // postavlanje brzine kretanja boss-a na 2.5
               anim.SetBool("walk", true); // postavljanje vrijednosti animacije hoda na true
   }
}
        // definiranje varijable tipa strukture Quaternion na temelju koje se kontrolira rotacija boss-a
        // LookRotation() metoda se koristi za zakretanje boss-a prema skulpturi koju napada
Quaternion target_Rotation = Quaternion.LookRotation(targets[index].transform.position - transform.position);
target_Rotation.x = 0f;
target_Rotation.z = 0f;


           // definiranje rotacije objekta pri čemu se koristi metoda Lerp() koja prima 3 parametra
           // pri čemu se radi zakretanje između boss-a (objekta a), mete (objekt b) i definiranom brzinom u vremenu - 5f
           transform.rotation = Quaternion.Lerp(transform.rotation, target_Rotation, 5f * Time.deltaTime);

// provjera ukoliko je meta LegacyTarget pronađena
if (showWeapon.isImageTargetDetected)
{
   colliders.SetActive(true); // postavljanje collider-a boss-a aktivnim u sceni
   body.SetActive(true); // postavljanje tijela boss-a vidljivim u sceni
}
else
{
            colliders.SetActive(false); // postavljanje collider-a boss-a neaktivnim u sceni
 body.SetActive(false); // postavljanje tijela boss-a nevidljivim u sceni
}

            currentHealth = health / 300f; // postavljanje trenutne vrijednosti energije na temelju iznosa energije
            healthImage.fillAmount = currentHealth; // vizualno postavljanje količine energije, ispunjenost slike
```

```
        if (health <= 0) // provjera je li energije jednaka ili manja od 0
            bossDead = true; // postavljanje zastavice da je boss mrtav
    }
    else // ukoliko zastavica canStart nije istinita
    {
        anim.Play("stand", 0); // pokretanje animacije stajanja iz prvog sloja animatora
        navAgent.isStopped = true; // definiranje da boss stoji
        navAgent.speed = 0f; // postavljanje brzine kretnje boss-a na 0
    }
}
```

Sl. 4.27. Drugi dio skripte BossController

Treći dio koda sadrži funkcije kojima je ostvarena kontrola regeneriranja, pokretanja različitih animacija. U ovoj skripti se koriste funkcije koje se pozivaju prilikom određenih događaja definiranih unutar animacija. Tako se prilikom pokretanja animacije *relax* aktivira događaj kojim se poziva funkcija *HealingStart()* kojom počinje proces regeneriranja. Dok se pri kraju iste animacije u određenom trenutku nalazi događaj koji poziva funkciju *HealingEnd()* čime je završen proces regeneriranja. Istim principom koriste se metode: *RoarSoundOnEnemyWin()*, *AttackSound()*, *DealDamage()* i *SpawnSkeletons()*. Kod se nalazi na idućoj stranici (Sl. 4.28.).

```
// funkcija koja umanjuje životnu energiju boss-a (HP)
public void TakeDamage(float amount)
{
    navAgent.isStopped = true; // definiranje da boss stoji
    navAgent.speed = 0f; // postavljanje brzine kretnje boss-a na 0
    if (!isBossHeal) // provjera ako se boss ne regenerira
        health -= amount; // smanjenje energije za definirani iznos
    // provjera je li energija manja ili jednaka od 0
    if (health <= 0)
    {
        colliders.SetActive(false); // postavljanje collider-a boss-a neaktivnim u sceni
        anim.Play("die", 0); // pokretanje animacije za umiranje iz prvog sloja animatora
        deathSound.Play(); // pokretanje zvučnog efekta za smrt
    }
}
// funkcija tipa IEnumerator kojom se uništava objekt - boss
IEnumerator Death()
{   // definiranje zastoja izvođenje koda na 2.4 sekunde
    yield return new WaitForSecondsRealtime(2.4f);
    Destroy(gameObject); // uništavanje objekta
}
// funkcija tipa IEnumerator kojom se zaustavljanje kretanje boss-a
IEnumerator HurtDelay()
{
    yield return new WaitForSecondsRealtime(0.5f);
    navAgent.isStopped = false;
}
// funkcija za stvaranje boss-a
IEnumerator SpawnBoss()
{
    isBossHeal = true; // postavljanje zastavice renegeriranja na true
    yield return new WaitForSecondsRealtime(2f);
    isBossHeal = false; // postavljanje zastavice regeneriranja na false
    canStart = true; // postavljanje zastavice canStart na true
}
// funkcija za smanjene energije kipa
```

```
    public void DealDamage()
    {
        sculptureEnergy.energyCurrentState -= 5f;
    }
    // funkcija za stvaranje 4 napredna kostura na predefiniranim mjestima
    public void SpawnSkeletons()
    {
        StartCoroutine(helpSpawner[0].DelaySpawn());
        StartCoroutine(helpSpawner[1].DelaySpawn());
        StartCoroutine(helpSpawner[2].DelaySpawn());
        StartCoroutine(helpSpawner[3].DelaySpawn());
    }

    // funkcija za početak regeneriranja
    public void HealingStart()
    {
        isBossHeal = true;
        roarSound.Play();
    }
    // funkcija za kraj regeneriranja
    public void HealingEnd()
    {
        isBossHeal = false;
        health = 300f; // postavljanje energije boss-a na 100%
        timeToHeal += 1; // povećanje brojača za broj regeneriranja za 1
    }
    // funkcija tipa IEnumerator kojom se pokreće animacija slavljenja
    IEnumerator DelayCelebrate()
    {
        yield return new WaitForSecondsRealtime(3f);
        anim.SetBool("celebrate", true);
    }
    // funkcija za pokretanje zvučnog efekta slavlja
    public void RoarSoundOnEnemyWin()
    {
        roarSound.Play();
    }
    // funkcija za pokretanje zvučnog efekta za napad
    public void AttackSound()
    {
        attackSound.Play();
    }
}
```

Sl. 4.28. Treći dio skripte BossController

4.6. Tok igre i oružja

Igra je bazirana na ukupno četiri runde/vala napada kostura. Kao indikator koristi se tekst kojim je ispisan broj runde i prikazuje se na početku svakog vala. U ovisnosti o trenutnoj rundi ovisi i oružje koje je dostupno igraču. Kako se povećavaju runde tako se i otključavaju nova oružja. U prvoj rundi stvaraju se kosturi iz jednog *spawnera*, pri čemu se broji ukupan broj stvorenih kostura. Tijekom ove runde jedino dostupno oružje je pištolj. Ukoliko je broj ukupno stvorenih kostura veći od 10 aktivira se novi *spawner* i prikazuje se tekst da je u tijeku druga runda i tada se pojavljuje indikator da je novo oružje otključano, automatska puška. U drugoj rundi postoje dva aktivna *spawnera* koji su aktivni sve dok se ne stvori ukupni broj od 40 kostura. Pošto je otključano

oružje koje ovisi o municiji (automatska puška) počinju se stvarati metci po sceni koje igrač može skupiti tako što ih treba nanišaniti. Oni se stvaraju u grupama po 50 i 100 metaka u razmaku od 10 do 18 sekundi. Nakon što broj kostura prijeđe 40, aktivira se još jedan *spawner* (treći po redu), ispisuje se tekst da je runda 3 u tijeku i pojavljuje se indikator da je otključano novo oružje (strojnica). Ukoliko igrač uspješno porazi do tada svih 60 stvorenih kostura započinje četvrta runda koja je zadnja. Ponovo se ispisuje tekst runde, pojavljuje se indikator za novo oružje (snajper). Od tog trenutka aktivna su četiri *spawnera* sve dok se ne dođe do ukupnog broja kostura koji iznosi 100. Nakon toga oni se gase i počinje faza u kojoj igrač čeka da se stvori *boss*. Tijekom te faze, igrač ima oko 40 sekundi da se pripremi i da pokupi metke koji se stvaraju po sceni kako bi imao dovoljno metaka za zadnju borbu. Da bi igrač znao što se događa, ova faza je popraćena promjenom melodije i ispisivanjem teksta kojim se igrač informira da se pripremi na nadolazeće. Kada se stvori *boss* počinje zadnja faza igre nakon koje će igrač pobijediti ili izgubiti. Ako igrač ubije *bossa* i uspješno obrani kip ispisuje se poruka za pobjedu, a ako ne, ispisuje se poruka da je igrač izgubio. Svaki ishod je popraćen i različitom melodijom i prikazom konačnog rezultata igre koji vraća igrača u glavni izbornik igre.

Kako je implementiran tok igre opisan na prethodnoj stranici može se vidjeti u skripti spawnCounter koja će biti prikazana u četiri dijela. U prvom dijelu definirane su varijable i funkcija *Start()*.

```
public class spawnCounter : MonoBehaviour
{
    public GameObject[] spawners; // definiranje polja varijable tipa objekta

    public int enemyCounter = 0; // definiranje i inicijaliziranje brojača
        public int totalSpawnedEnemies = 0; // definiranje i inicijaliziranje ukupnog broja stvorenih neprijatelja
    public Spawner[] spawner; // definiranje polja varijable tipa klase Spawner

    public GameObject[] bloomEffect; // definiranje polja varijable tipa objekta
    public bool canStart = false; // definiranje zastavice tipa boolean

        private SculptureEnergy sculptureEnergy; // definiranje varijable tipa klase SculptureEnergy
    public GameObject[] bloomEffectVictory; // definiranje polja varijable tipa objekta

        public BossController[] bossController; // definiranje polja varijable tipa klase BossController
    public GameObject[] bossBloom; // definiranje polje varijable tipa objekt
    public GameObject[] boss; // definiranje polja varijable tipa objekt

       public GameObject bossSpawnTrigger; // definiranje objekta koji aktivira stvaranje boss-a
    // definiranje indeksa boss-a koji ce se stvoriti
    public int bossIndex; // postoje 4 predefinirana mjesta na koja se boss može stvoriti

    public AudioSource bossCallMelody; // definiranje varijable za melodiju
    public GameObject mainMelody, bossMelody; // definiranje objekata koji sadrže melodije

    public Text roundText; // definiranje varijable tipa Text za prikaz rundi
        private bool[] roundTextFlag = { true, true, true, true }; // zastavice da definiranje trenutne runde

    public Text EnemyCounterText; // definiranje teksta za ispis stvorenih kostura
```

```
//ammo
    public GameObject ammoObject, ammoBonusObject; // definiranje objekata koji predstavljaju metke
    public GameObject[] ammoSpawnPosition; // definiranje polja objekata koji predstavljaju mjesta stvaranja metaka
    private bool bossDead = false, gameLost = false; // zastavice za detekciju pobjede i poraza
public AmmoController ammoController; // definiranje varijable tipa klase AmmoController
public AudioSource ammoSpawnSound; // definiranje varijable za zvučni efekt

void Start()
{
      sculptureEnergy = FindObjectOfType<SculptureEnergy>(); // dohvaćanje objekta koji sadrži skriptu SculptureEnergy
      bossIndex = Random.Range(0, 3); // postavljanje nasumične vrijednosti bossIndex-a između 0 i 3
}
```

Sl. 4.29. Prvi dio skripte spawnCounter

Drugi dio skripte zadrži funkcije *OnEnable()* i *Awake()* za stvaranje kostura i metaka te inicijalno gašenje svih *spawnera*. Zatim slijedi funkcija *Update()* unutar koje je definirano: brojanje kostura, ispisivanje njihovog broja, ispisivanje broja rundi, pozivanje funkcije za stvaranje *bossa*, kontrola aktivnih *spawnera*. Detaljnije objašnjenje svake linije nalazi se na slici ispod (Sl. 4.30.).

```
// funkcija koja se poziva kada objekt postane aktivan u sceni
private void OnEnable()
{
  StartCoroutine(StartDelay()); // pozivanje coroutine za početak stvaranja kostura
  StartCoroutine(SpawnAmmo()); // pozivanje coroutine za početak stvaranja metaka u sceni
}
// funkcija koja se prva poziva prilikom inicijalizacije klase
private void Awake()
{   // for petlja kojom se deaktiviraju svi spawner-i u sceni
     for (int i = 0; i < spawners.Length; i++)
    {
       spawners[i].SetActive(false);
    }
}
// Update is called once per frame
void Update()
{   // dohvaćanje objekata s nazivom EnemyTarget i spremanje njihovog broja u varijablu
    enemyCounter = GameObject.FindGameObjectsWithTag("EnemyTarget").Length - 1;
          // postavljanje ukupnog broja kostura na temelju brojača koji je definiran unutar srkipte Spawner
          totalSpawnedEnemies = spawner[0].counter + spawner[1].counter + spawner[2].counter + spawner[3].counter; //
          pridruzivanje od svakog spawner-a koji ima svoj lokalni brojac
    // ispisivanje broja stvorenih kostura
    EnemyCounterText.text = "Skeletons: " + totalSpawnedEnemies.ToString() + " / 100";

    if (canStart) // provjera zastavice za početak izvođenja koda
    {   // provjera ima li preko 99 stvorenih kostura ili je došlo do poraza
       if (totalSpawnedEnemies > 99 || sculptureEnergy.GetDefeatFlag)
       {
          ShutDownSpawners(); // pozivnje funkcije za gašenje spawner-a
          if (!bossController[bossIndex].bossIsSpawned && !sculptureEnergy.GetDefeatFlag && enemyCounter == 0)
          {   // aktiviranje objekta za stvaranje boss-a
             bossSpawnTrigger.SetActive(true);
          }
          // provjera ukoliko je boss poražen, pobjeda
          if (bossController[bossIndex].GetVictoryStatus)
          {  // zastavica kojom je definirano da je boss mrtav
            bossDead = true;
            bossMelody.SetActive(false); // deaktiviranje objekta koji sadrži melodiju za boss-a
          }
          // provjera ukoliko je igrac izgubio, lose
          if (sculptureEnergy.GetDefeatFlag)
          {  // postavljanje zastavice poraza na true
```

```
            gameLost = true;
                        bossMelody.SetActive(false); // deaktiviranje objekta koji sadrži melodiju za boss-a
            }
        } // provjera je li broj kostura veći od 60
        else if (totalSpawnedEnemies > 60)
        {   // pozivanje funkcije za aktiviranje spawner-a, argumenti redom:
                        // maksimalan broj kostura u sceni, broj kosutra koji definira aktivnost spawner-a
            // zadnji argument definira ukupan broj aktivnih spawner-a
            SetSpawner(15, 10, 4);
            if (roundTextFlag[3]) // provjera ukoliko se treba ispisivati tekst za rundu 4
                StartCoroutine(ShowRoundText("Four", 3));
        } // analogno se izvršavaju i ostale provjere za aktivaciju spawner-a
        else if (totalSpawnedEnemies > 40)
        {
            SetSpawner(12, 8, 3);
            if (roundTextFlag[2])
                StartCoroutine(ShowRoundText("Three", 2));
        }
        else if (totalSpawnedEnemies > 10)
        {
            SetSpawner(8, 5, 2);
            if (roundTextFlag[1])
                StartCoroutine(ShowRoundText("Two", 1));
        }
        else
        {
            SetSpawner(5, 2, 1);
            if (roundTextFlag[0])
                StartCoroutine(ShowRoundText("One", 0));
        }
    }
}
```

Sl. 4.30. Drugi dio skripte spawnCounter

U trećem dijelu koda nalaze se tri funkcije: *SetSpawner()*, *ShutDownSpawner()* i *StartDelay()*. Prva funkcija prima tri argumenta: najveći dopušteni broj kostura, najmanji broj kostura i ukupni broj *spawnera* koji trebaju biti aktivni u sceni. Druga funkcija, kako i samo ime naglašava, koristi se za deaktiviranje svih *spawnera* i ne prima nikakve argumente. Posljednjom funkcijom stvara se odgoda od dvije sekunde za stvaranje kostura (aktiviranje *spawnera*) i aktiviranje glavne melodije u igri. Kako izgleda opisani kod može se vidjeti u nastavku. (Sl. 4.31.).

```
// funkcija za aktiviranje broja spawner-a, prima tri argumenta
void SetSpawner(int enemyNumberMax, int enemyNumberMin, int totalSpawners)
{   // provjera ukoliko je trenutni broj kostura veći od maksimalno dozvoljenog
    if (enemyCounter >= enemyNumberMax)
    {   // ako je, gase se svi spawner-i
        for (int i = 0; i < totalSpawners; i++)
        {
            spawners[i].SetActive(false);
            bloomEffect[i].SetActive(false);
        }
    } // ukoliko je trenutni broj kostura manji od najmanje dopuštenog
    else if (enemyCounter <= enemyNumberMin)
    {   // pale se spawner-i
        for (int i = 0; i < totalSpawners; i++)
        {
            spawners[i].SetActive(true);
            bloomEffect[i].SetActive(true);
        }
    }
```

```
}

// funkcija za gašenej spawner-a
void ShutDownSpawners()
{   // ukoliko je nije poraz i boss nije pobijedio
    if (!sculptureEnergy.GetDefeatFlag && !bossController[bossIndex].GetVictoryStatus)
    {   // gase se svi spawner-i
        for (int i = 0; i < spawners.Length; i++)
        {
            spawners[i].SetActive(false);
            bloomEffect[i].SetActive(false);
        }
    } // ukoliko je igrač izgubio, poraz
    else if (sculptureEnergy.GetDefeatFlag)
    {   // zato je potrebno ugasiti spawner-e da se ne stvaraju kosturi
        // gasenje spawner-a
        for (int i = 0; i < spawners.Length; i++)
        {
            spawners[i].SetActive(false);
        }
    }
}

// funkcija tipa IEnumerator sa odgodom od dvije sekunde za početak stvaranja kostura
IEnumerator StartDelay()
{
    yield return new WaitForSecondsRealtime(2f);
    canStart = true;
    mainMelody.SetActive(true);
}
```

Sl. 4.31. Treći dio skripte spawnCounter

Posljednji dio skripte kojim je definirano stvaranje *bossa*, prikazivanje broja runde i stvaranje metaka vidljivo je u četvrtom dijelu skripte. Navedene funkcionalnosti definirane su redom metodama: *SpawnBoss()*, *ShowRoundText()* i *SpawnAmmo()*. Kod kojim je to ostvareno nalazi se ispod (Sl. 4.32.).

```
// funkcija tipa IEnumerator za stvaranje boss-a
public IEnumerator SpawnBoss()
{
   mainMelody.SetActive(false); // deaktiviranje objekta koji sadrži glavnu melodiju
   bossCallMelody.Play(); // pozivanje zvučnog efekta za stvaranje boss-a
         yield return new WaitForSecondsRealtime(bossCallMelody.clip.length - 6f); // odgoda oko 40 sekundi
   bossMelody.SetActive(true); // aktiviranje objekta koji sadrži melodiju za boss-a
   bossController[bossIndex].spawnSound.Play(); // pokretanje zvučnog efekta za stvaranje
         bossBloom[bossIndex].SetActive(true); // aktiviranje vizualnog efekta tijekomstvaranja
   yield return new WaitForSecondsRealtime(3f); // odgoda na 3 sekunde
   bossController[bossIndex].spawnBossRoar.Play(); // pokretanje efekta za glasanje boss-a
    boss[bossIndex].SetActive(true); // aktiviranje objekta - boss-a
}
// funkcija tipa IEnumerator za prikazivanje rundi, priva dva argumenta:
// naslov teksta i broj runde
IEnumerator ShowRoundText(string title, int flagIndex)
{
    roundText.enabled = true; // aktiviranje teksta
    roundText.text = "Round: " + title; // postavljanje teksta
    yield return new WaitForSecondsRealtime(3f); // odgoda na 3 sekunde
    roundText.enabled = false; // deaktiviranje teksta
    roundTextFlag[flagIndex] = false; // postavljanje zastavice trenutne runde na false
}
// funkcija tipa IEnumerator za stvaranje metaka
public IEnumerator SpawnAmmo()
```

```
{   // stalno izvršavanje while petlje
    while (true)
    {
            // provjera da je barem jedno oruzje otkljucano, da nije lose ili win, da nije pauza i da nema ukupno vise od 1000
            metaka, ako je zadovoljeno onda moze instanciranje metaka
            if (totalSpawnedEnemies > 10 && !gameLost && !bossDead && Time.timeScale == 1f &&
            ammoController.ammunition < 1000)
      {   // definiranje nasumičnog broja između 0 i 400
        var randomNumber = Random.Range(0, 400);
                var spawnerIndex = Random.Range(0, 41); // definiranje nasumičnog broja između 0 i 41, broj mjesta za
                stvaranje metaka
                var time = Random.Range(10f, 18f); // nasumično vrijeme stvaranja metaka između 10 i 18
                yield return new WaitForSecondsRealtime(time); // odgoda na temelju vremena, između 10 i 18 sekundi
        ammoSpawnSound.Play(); // pokretanje zvučnog efekta stvaranja metaka

        if (randomNumber % 2 == 0) // provjera ukoliko je broj paran
                {   // stvaranje metaka prema definiranom indeksu/mjestu pri čemu je rotacija   postavljena na vrijednost
                0,0,0
                    Instantiate(ammoObject, ammoSpawnPosition[spawnerIndex].transform.position,  Quaternion.identity);
        }
        else // ukoliko broj nije paran stvaraju se bonus metci
                    Instantiate(ammoBonusObject, ammoSpawnPosition[spawnerIndex].transform.position,
                    Quaternion.identity);
      }
      yield return null;
    }
  }

}
```

Sl. 4.32. Četvrti dio skripte spawnCounter

U drugom dijelu ovog potpoglavlja biti će objašnjeno svako oružje pojedinačno koje postoji u igri. Modeli oružja i teksture koje se koriste preuzeti su besplatno [14].

a) Pištolj

Ovo oružje igrač posjeduje od početka igre i jedino je oružje koje nema ograničenje na metke, odnosno ima beskonačno metaka. Zbog toga igrač uvijek može koristiti pištolj kao oružje ukoliko nema metaka. Pištolj može pucati metak po metak i brzina pucanja ovisi o igračevoj brzini stiskanja gumba za pucanje. Kako izgleda model pištolja može se vidjeti na slici ispod.

Sl. 4.33. Prikaz modela pištolja

b) Automatska puška

Automatska puška je prvo oružje koje koristi metke pri čemu igrač početno ima 200 metaka. Ovo oružje se otključava na početku druge runde i puca po tri metaka odjednom. Uzastopna brzina pucanja ovim oružjem definirana je na svakih pola sekunde. Nakon otključavanja ovog oružja počinje stvaranje metaka u sceni. U nastavku slijedi model ovog oružja.

Sl. 4.34. Prikaz modela automatske puške

c) Strojnica

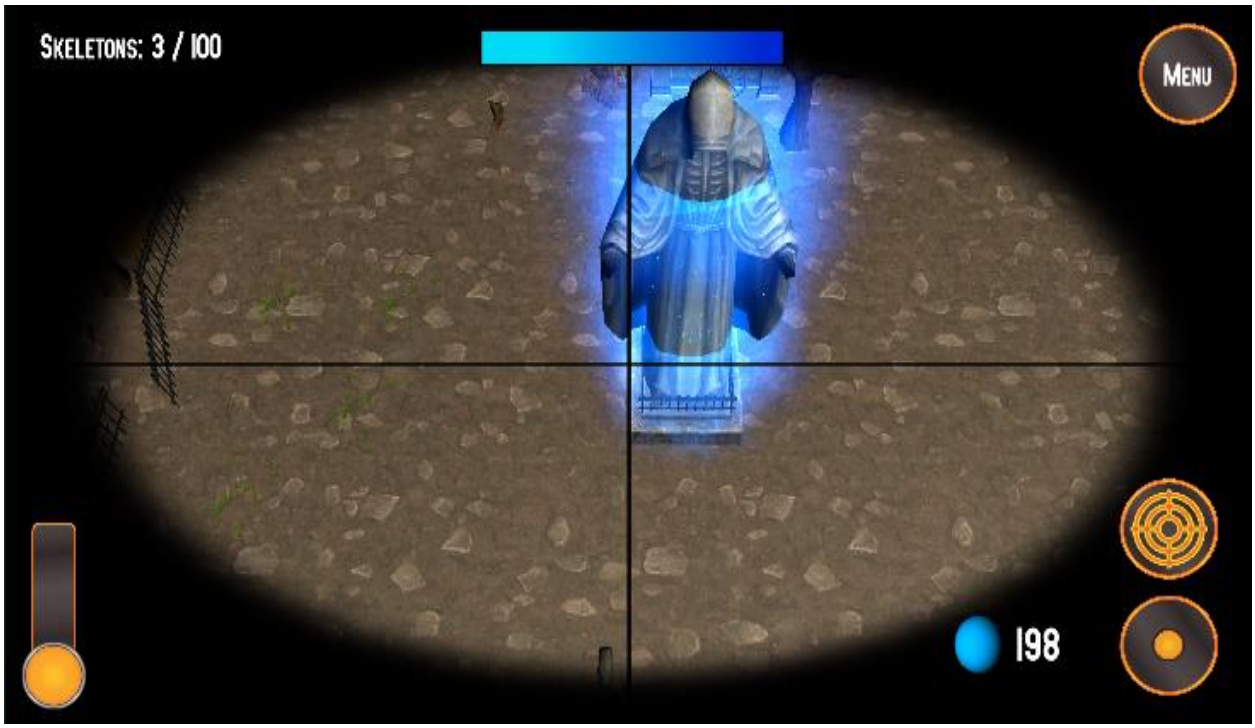
Strojnica je jedino oružje koje puca kontinuirano prilikom držanja gumba za pucanje. Ovo oružje također ima indikator topline. Zbog toga se može pregrijati ukoliko igrač drži gumb za pucanje bez prestanka i zatim mora pričekati nekoliko sekundi da se oružje ohladi. Ukoliko igrač pusti prije gumb nego što se oružje pregrije odmah počinje proces hlađenja. Ovo oružje također troši metke, a otključava se na početku treće runde. Kako izgleda strojnica može se vidjeti na slici (Sl. 4.35.).

Sl. 4.35. Prikaz modela strojnice

d) Snajper

Ovo oružje predstavlja najjače oružje u igri i ima mogućnost podešavanja optike. Dovoljan je jedan metak da se ubije bilo koja vrsta kostura. Snajper je zadnje oružje koje igrač može otključati i to na početku četvrte runde. Ovim oružjem se jedino može pucati kada se koristi optika. Kao i prethodna dva oružja i ovo koristi metke. Kako izgleda model snajpera može se vidjeti na slici (Sl. 4.36.), a kako izgleda kada igrač koristi optiku vidi se na slici (Sl. 4.37.).

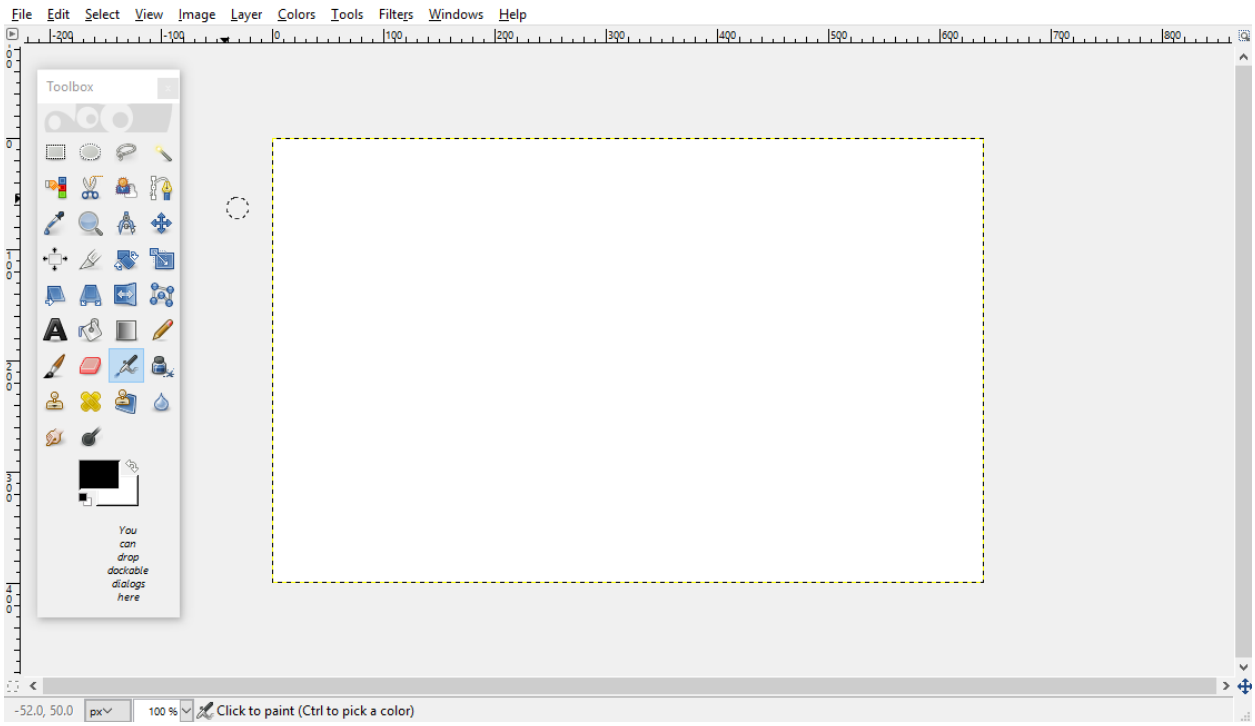
Sl. 4.36. Prikaz modela snajpera



Sl. 4.37. Prikaz korištenja optike snajpera

4.7. Grafika

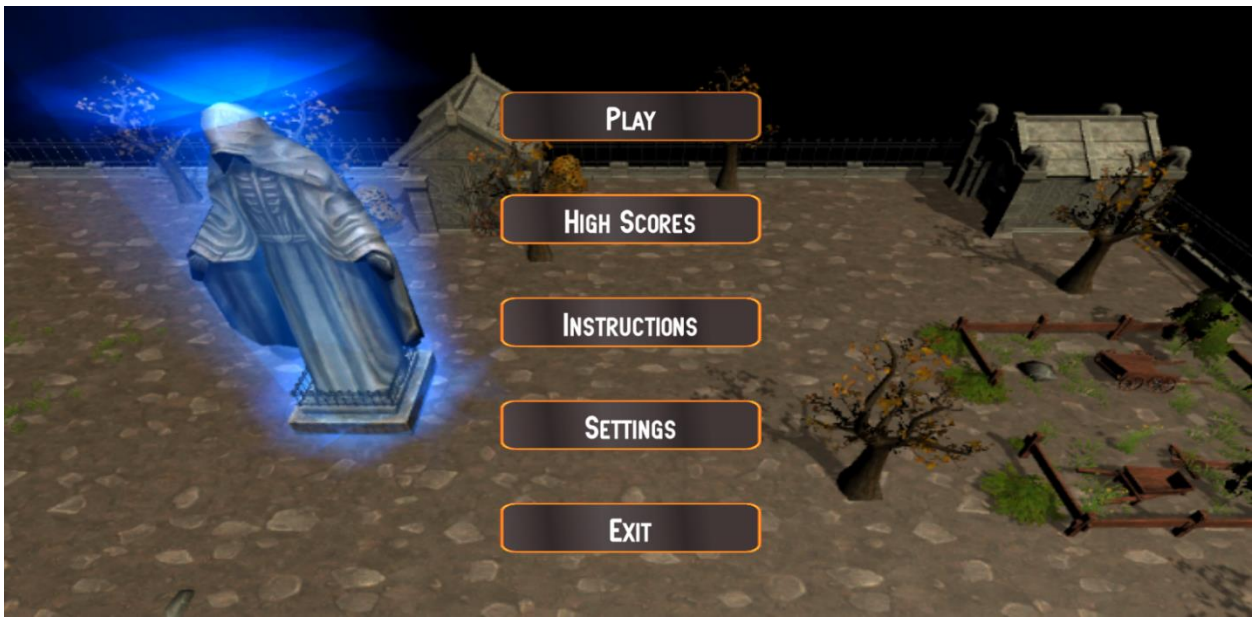
Cijeli izgled igre (scena) napravljen je korištenjem Unity Editora dok je za dizajn korisničkog sučelja (GUI, eng., *graphical user interface)* [15] korišten GIMP 2. Program je besplatan i preuzet je sa službene stranice [16]. Korisničko sučelje je izrazito bitan element igre jer omogućuje interakciju igrača za različitim prozorima u igri, izbornicima, korištenje kontrola, izmjenu postavki igre i slično. Kako izgleda glavni prozor programa GIMP 2 može se vidjeti na slici (Sl. 4.38.). Ovaj program omogućuje razne mogućnost rada sa slikama (dorade, uređivanje, crtanje, konvertiranje formata slike, itd.). Više o uputama kako se koristi ovaj program može se pročitati na stranici službene dokumentacije [17].



Sl. 4.38. Prikaz glavnog prozora s alatom za crtanje u programu GIMP 2

4.7.1. Glavni izbornik

Kada se igra učita, prva scena koju igrač vidi je glavni izbornik. Sastoji se od pozadinske slike i pet gumbova. Prije nego što bude objašnjena funkcija svakog pojedinog gumba, slijedi slika glavnog izbornika.



Sl. 4.39. Prikaz glavnog izbornika

Gledano od gore prema dolje, prvi gumb koji se nalazi u izborniku je Play. Pritiskom na ovaj gumb počinje učitavanje nivoa. Ispod njega se nalazi gumb High Scores kojim se otvara panel koji prikazuje trenutno stanje prvih 5 rezultata koji sadrži i gumb za povratak u glavni izbornik. Nakon njega slijedi gumb Instructions koji ukoliko se pritisne također otvara novi panel sa uputama o igri i mogućnosti povratka u glavni izbornik. Četvrti po redu je gumb Settings kojim se otvara panel unutar kojeg se mogu mijenjati postavke zvuka, kvaliteta grafike (detalji) i postojanje sijena u sceni. Zadnji gumb je Exit kojim se izlazi iz igre. Bitno je naglasiti da Unity prilikom korištenja sučelja stvara objekt *EventSystem* koji služi za detekciju događaja kod GUI-a kao što je pritisak na gumb. Navedene funkcionalnosti definirane su u skripti MainMenu (Sl. 4.40).

```
public class MainMenu : MonoBehaviour {

    private AudioSource btnSound; // definiranje varijable za zvuk gumba
    public GameObject[] buttons; // definiranje varijable polja objekata - gumbovi
    public GameObject settingsPanel; // definiranje varijable tipa objekt za panel postavki
    public GameObject instructionsPanel; // definiranje varijable tipa objekt za panel uputa
    public GameObject highScorePanel; // definiranje varijable tipa objekt za panel rezultata
    public Text qualityText, shadowsText; // definiranje varijabli tipa Text za ipis postavki

    void Start () {
        btnSound = GetComponent<AudioSource>(); // dohvaćanje komponente za zvuk
    }
    // funkcija tipa IEnumerator za odgodu prilikom pritiska određenog gumba
    // prima jedan parametar tipa string kojim se difinira namjena gumba
    IEnumerator ButtonDelay(string option)
    {
        switch (option)
        {   // za slučaj "start"
            case "start": // odgoda na 0.2 sekunde
                yield return new WaitForSecondsRealtime(0.2f);
                                LevelLoader.instance.LoadScene(1); // pozivanje funkcije iz skripte LevelLoader
                break; // za učitavanje scene nivoa (broj scene je 1)

            case "exit":
                yield return new WaitForSecondsRealtime(0.2f);
                Application.Quit(); // izlazak iz aplikacije
                break;

            case "settings":
                yield return new WaitForSecondsRealtime(0.2f);
                foreach (var button in buttons)
                {   // deaktiviranje svih gumbova u sceni
                    button.SetActive(false);
                }
                settingsPanel.SetActive(true); // aktiviranje panela za postavke
                break;

            case "instructions":
                yield return new WaitForSecondsRealtime(0.2f);
                foreach (var button in buttons)
                {
                    button.SetActive(false);
                }
                instructionsPanel.SetActive(true);
                break;

            case "highScore":
                yield return new WaitForSecondsRealtime(0.2f);
                foreach (var button in buttons)
                {
                    button.SetActive(false);
                }
                highScorePanel.SetActive(true);
                break;

            case "backToMenu":
                yield return new WaitForSecondsRealtime(0.2f);
                settingsPanel.SetActive(false);
                instructionsPanel.SetActive(false);
                highScorePanel.SetActive(false);
                foreach (var button in buttons)
                {
                    button.SetActive(true);
                }
                break;

                    default:
```

```
                break;
            }
        }
        // funkcija za učitavanje/pokretanje nivoa
        public void StartGame()
        {
            btnSound.Play(); // pokretanje zvučnog efekta za zvuk gumba
            StartCoroutine(ButtonDelay("start"));
        }
        // funkcija za prikaz rezultata
        public void HighScore()
        {
            btnSound.Play();
            StartCoroutine(ButtonDelay("highScore"));
        }
        // funkcija za prikaz uputa igre
        public void Instructions()
        {
            btnSound.Play();
            StartCoroutine(ButtonDelay("instructions"));
        }
        // funkcija za prikaz postavki
        public void Settings()
        {
            btnSound.Play();
            StartCoroutine(ButtonDelay("settings"));
        }
        // funkcija za povratak u glavni izbornik
        public void BackToMenu()
        {
            btnSound.Play();
            StartCoroutine(ButtonDelay("backToMenu"));
        }
        // funkcija za izlazak iz igre
        public void ExitGame()
        {
            btnSound.Play();
            StartCoroutine(ButtonDelay("exit"));
        }
    }
```

Sl. 4.40. Skripta MainMenu

Najbitnija funkcija u skripti je *ButtonDelay()* koja prima jedan argument *option* tipa *string*, a omogućuje jednostruko grananje na temelju uvjeta (korištenjem *switch-case* naredbe). Navedenim argumentom definira se koji slučaj će se izvršiti unutar kojeg je definirana cijela funkcionalnost za pojedini gumb. Tako se prilikom pritiska na gumb Start poziva funkcija *StartGame()*, unutar koje se osim aktiviranja zvučnog efekta gumba poziva i funkcija *ButtonDelay()* s argumentom „start“ kojim se definira da se treba izvršiti kod za učitavanje nivoa. Na isti se način pozivaju funkcije prilikom pritiska ostalih gumbova, pri čemu se razlikuje jedino vrijednost argumenta *option*.

4.7.2. Nivo

Nivo predstavlja glavnu scenu u kojoj igrač igra, pri čemu mu je omogućena interakcija s objektima (neprijateljima). Za izradu nivoa korišteni su objekti koji su preuzeti besplatno za stranice [18]. Postoji samo jedan nivo koji se sastoji od raznih objekata među kojima su najbitniji: *spawneri* i kip. *Spawner* predstavlja objekt iz kojeg se stvara kostur, ukupno ih ima četiri. Ovisno o trenutnom valu napada kostura, *spawneri* se aktiviraju. Drugi objekt kip, predstavlja središte scene i on predstavlja izvor energije života koju neprijatelji žele uništiti. Kip je statičan objekt oko kojeg se kreće energija u smjeru od podnožja kipa do vrha. Cilj je da igrač svojom vještinom uspije spasiti kip od navale neprijatelja i tako sačuvati energiju života koju kip posjeduje. Prikaz nivoa s upaljenim prvim *spawner*om nalazi se na slici ispod.

Sl. 4.41. Prikaz nivoa

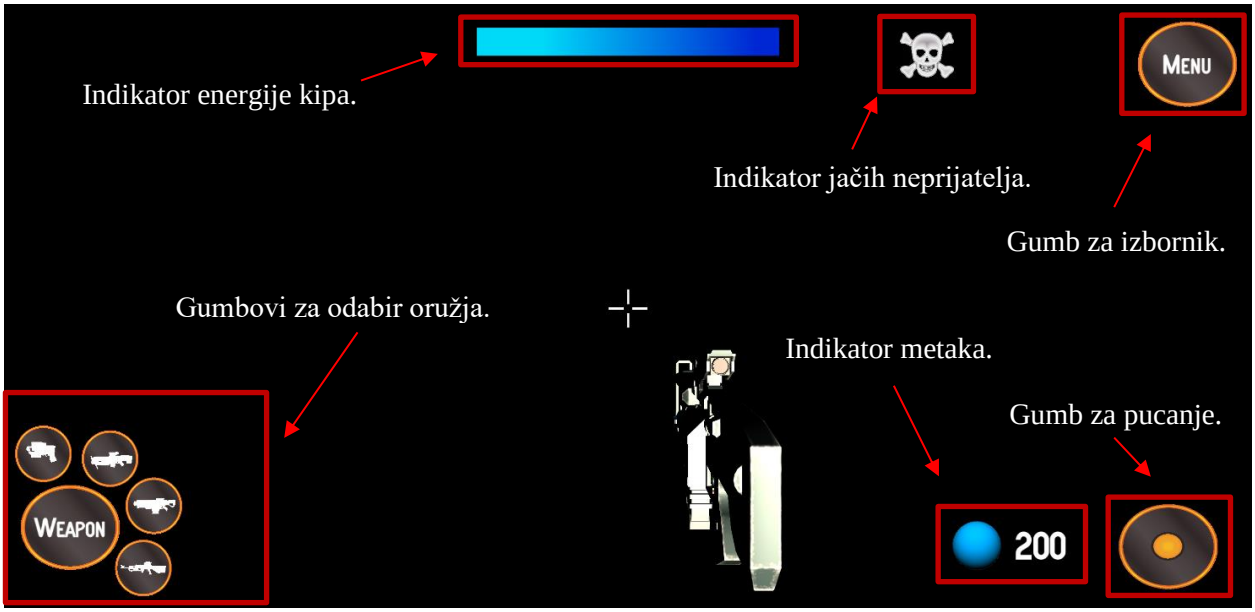
4.7.3. Sučelje unutar nivoa

Unutar scene ovim sučeljem su prikazane informacije o statusu nivoa, gumbovi za kontrolu oružja, indikator jače vrste kostura i broja metaka te gumb za pauzu. Prikaz izbornika u sceni definirano je u skripti StartGameController, a aktivira se na gumb Menu. Na stranici koja slijedi nalazi se kod skripte.

```
public class StartGameController : MonoBehaviour {

    private bool paused = false; // definiranje varijable tipa boolean kao zastavice pauze
    public GameObject menuPanel; // definiranje varijable tipa objekt za kontrolu izbornika
    // definiranje objekata - gumbova (kontrola vidljivosti)
    public GameObject weaponButtons, shootButton, scopeButton, mainMenuButton;

    public AudioSource btnSound; // definiranje zvučnog efekta prilikom pritiska gumba
    public GameObject statusPanel, showScoreButton; // definiranje objekta za kontrolu panela
    public bool scoreFlag = false; // definiranje zastavice za prikaz konačnog rezultata

    void Start () {
        menuPanel.SetActive(false); // deaktiviranje prikaza izbornika u sceni
    }

    void Update () {
        if (paused) // provjera je li vrijednost zastavica paused istinita
        {
            Time.timeScale = 0f; // postavljanje vremena igre na 0, pauza
            menuPanel.SetActive(true); // aktiviranje objekta za prikaz izbornika
        }
        else // ukoliko nije vrijednost zastavice true
        {
            Time.timeScale = 1f; // igra se normalno izvodi
            menuPanel.SetActive(false); // deaktiviranje prikaza izbornika u sceni
        }
    }
    // funkcija za postavljanje zastavice paused na true
    public void MenuButton()
    {
        btnSound.Play();
        StartCoroutine(ButtonDelayPause(true, 0.2f));
    }
    // funkcija za nastavak igre, paused - false
    public void Resume()
    {
        btnSound.Play();
        StartCoroutine(ButtonDelayPause(false, 0.2f));
    }
    // funkcija tipa IEnumerator za odgodu prilikom pritiska gumba
    IEnumerator ButtonDelayPause(bool isPaused, float time)
    {
        yield return new WaitForSecondsRealtime(time);
        paused = isPaused;
    }
    // funkcija za povratak u glavni izbornik igre
    public void ReturnToMenu()
    {
        btnSound.Play();
        StartCoroutine(ReturnToMainMenuDelay());
    }

    IEnumerator ReturnToMainMenuDelay()
    {
        yield return new WaitForSecondsRealtime(0.2f);
        LevelLoader.instace.LoadScene(0);
    }
    IEnumerator ShowScore() funkcija za prikaz konačnog rezultata igre
    {
        scoreFlag = true;
        yield return new WaitForSecondsRealtime(0.2f);
        showScoreButton.SetActive(false);
        statusPanel.SetActive(true);
    }
}
```

Sl. 4.42. Skripta StartGameController

Unutar scene nalazi se jedan objekt tipa *canvas*[8] (platno) koji služi za prikaz navedenih panela, gumbova i igračevih kontroli. Primjer glavnog platna može se vidjeti na slici (Sl. 4.43.).



Sl. 4.43. Prikaz korisničkog sučelja

Zatim platno koje se prikazuje prilikom pritiska gumba Menu. (Sl. 4.44.).

Sl. 4.44. Prikaz izbornika u nivou

[8] *Canvas – predstavlja platno unutar kojeg moraju biti definirani svi UI elemetni kako bi se ispravno prikazivali*

Zadnji panel koji je bitan unutar nivoa je za prikaz konačnog rezultata koji je igrač ostvario tijekom igranja. Ovo platno se prikazuje neovisno o ishodu, prikazuje se i ako je igrač pobijedio i ako je izgubio. Na dnu panela nalazi se gumb Return To Main Menu. Nakon što se pritisne taj gumb igrač se preusmjerava u prvu scenu u kojoj je glavni izbornik pri čemu se spremaju rezultati u lokalnu bazu. Više o implementaciji lokalne baze biti će objašnjeno u potpoglavlju koje slijedi. Izgled platna koje prikazuje konačni rezultat može se vidjeti ispod (Sl. 4.45.).

Sl. 4.45. Prikaz panela s konačnim rezultatom

4.8. Lokalna baza i spremanje rezultata

Kako bi se pohranili rezultati koje je igrač ostvario potrebno je imati mjesto spremanja. Postoje razni načini spremanja podataka, pri čemu se mogu koristiti različite baze podataka. Za spremanje rezultata u ovoj je igri korišten *plugin*[9] SimpleJSON [19] koji sprema podatke u formatu JSON[10]. Ovaj format je jednostavan za čitanje i sastoji se od para vrijednosti: ključ-vrijednost. Ključ je uvijek pisan unutar dvostrukih navodnika, dok se vrijednost također može zapisivati unutar dvostrukih navodnika, pri čemu tipovi podataka: *number*, *null* i *boolean* mogu pisati i bez

[9] *Plugin – može se definirati kao posebni programski dodatak koji sadrži različite biblioteke, programski kod u svrhu rješavanja različitih problema*
[10] *JSON (JavaScript Object Notation) – predstavlja tip formata koji se najčešće koristi za prijenos podataka između servera i aplikacija*

dvostrukih navodnika. Rezultati igre spremaju se u jedan JSON objekt, kako to izgleda može se vidjeti na primjeru ispod (Sl. 4.46.).


```
{
  "score:1":540,
  "score:2":460,
  "score:3":300,
  "score:4":220,
  "score:5":60,
  "date:1":"10/06/2019",
  "date:2":"28/05/2019",
  "date:3":"26/05/2019",
  "date:4":"21/05/1019",
  "date:5":"18/05/2019"
}
```


Sl. 4.46. Prikaz JSON objekta koji sadrži podatke koji se spremaju lokalno

U primjeru iznad može se vidjeti da postoji ukupno 10 podataka, u obliku ključ-vrijednost. Prvih pet podataka predstavljaju podatke koji se odnose na rezultat (*score*) koji je igrač ostvario, pri čemu je vrijednost cjelobrojan broj (ne moraju biti zapisani unutar dvostrukih navodnika). Idućih pet vrijednosti odnose se na datum kada je igrač te rezultate postigao. Datumi su zapisani u engleskom formatu (*EN-gb*) pošto je cijela igra napravljena na engleskom jeziku. Ti podaci prikazuju se unutar panela za prikaz svih rezultata. Prikaz unutar glavnog izbornika nalazi se na slici ispod. Lokalno spremanje rezultata ostvareno je unutar skripte LevelLoader (Sl. 4.48.).

Sl. 4.47. Prikaz ispisa spremljenih rezultata u igri iz lokalne baze

```
public class LevelLoader : MonoBehaviour {

   // spremanje u lokalnu bazu u JSON formatu
   // prosljeđuje se argument - rezultat koji se želi spremiti
   public void SaveData(int score)
   {   // spremanje vrijednosti koje vraća funkcija LoadData()
      var storedData = LoadData();
      var highScore = score; //postavljanje prosljeđenog rezultata kao trenutno najveći

      DateTime today = DateTime.Now; // dohvaćanje trenutnog datuma
      JSONObject data = new JSONObject(); // kreiranje JSON objekta
      // provjera ukoliko je broj spremljenih podatak veći od 0, postoje u lokalnoj bazi
      if (storedData.Count > 0)
      {   // korištenje brojača da se spreme rezultati koji već postoje u bazi
         for (int i = 1; i < 6; i++)
         {   // spremanje rezultata i datuma kao ključevi: score:1, 2,...; date: 1, 2,...
            data.Add("score:"+i, storedData["score:"+i]);
            data.Add("date:"+i, storedData["date:"+i]);
         }
                  // provjera spremljenog prvog rezultata je li manji od prosljeđenog rezultata - highScore
                  // ujedino se i parsira u cjelobrojni tip kako bi zasigurno zapis bio pročitan kao broj
         if (int.Parse(storedData["score:1"]) <= highScore)
         {   // ako je veći trenutni rezultat postavlja se kao prvi u bazi, najveći
            data.Add("score:1", highScore);
            // spremanje datuma u formatu dan/mjesec/godina
            data.Add("date:1", today.Date.ToString("dd/MM/yyyy"));
                        // ostali se rezultati samo pomiću za jedno mjesto iz lokalne baze i ponovo spremaju
            for (int i = 2; i < 6; i++)
            {
               data.Add("score:"+i, storedData["score:"+(i-1)]);
               data.Add("date:"+i, storedData["date:"+(i-1)]);
            }
         } // ukoliko je spremljeni broj manji od prvog u bazi, ali veći od drugog
         else if (int.Parse(storedData["score:2"]) <= highScore)
         {  // isti postupak kao i u prvom slućaju samo se sprema pod rezultat 2 - score:2
            data.Add("score:2", highScore);
            data.Add("date:2", today.Date.ToString("dd/MM/yyyy"));

            for (int i = 2; i < 6; i++)
            {
               if(i == 2)
               {
                  data.Add("score:"+1, storedData["score:" + (i - 1)]);
                  data.Add("date:"+1, storedData["date:" + (i - 1)]);
               }
               else
               {
                  data.Add("score:"+i, storedData["score:"+(i - 1)]);
                  data.Add("date:"+i, storedData["date:"+(i - 1)]);
               }
            }
         } // ukoliko je spremljeni broj manji od prvog i drugog u bazi, ali veći od trećeg
         else if (int.Parse(storedData["score:3"]) <= highScore)
         {
            data.Add("score:3", highScore);
            data.Add("date:3", today.Date.ToString("dd/MM/yyyy"));

            for (int i = 1; i < 5; i++)
            {
               if (i < 3)
               {
                  data.Add("score:" + i, storedData["score:" + i]);
                  data.Add("date:" + i, storedData["date:" + i]);
               }
               else
               {
                  data.Add("score:" + (i+1), storedData["score:" + i]);
```

```
                data.Add("date:" + (i+1), storedData["date:" + i]);
            }
        }
    } // ukoliko je spremljeni broj manji od 1,2, i 3 u bazi, ali veći od četvrtog
    else if (int.Parse(storedData["score:4"]) <= highScore)
    {
        data.Add("score:4", highScore);
        data.Add("date:4", today.Date.ToString("dd/MM/yyyy"));

        for (int i = 1; i < 5; i++)
        {
            if (i < 4)
            {
                data.Add("score:" + i, storedData["score:" + i]);
                data.Add("date:" + i, storedData["date:" + i]);
            }
            else
            {
                data.Add("score:" + (i + 1), storedData["score:" + i]);
                data.Add("date:" + (i + 1), storedData["date:" + i]);
            }
        }
                } // ukoliko je spremljeni broj manji od 1,2,3 i 4 ali veći od petog, zadnjeg rezultata
    // koji se zapisuje u bazu i prikazuje u ispisu rezultata
    else if (int.Parse(storedData["score:5"]) <= highScore)
    {
        data.Add("score:5", highScore);
        data.Add("date:5", today.Date.ToString("dd/MM/yyyy"));

        for (int i = 1; i < 5; i++)
        {
            data.Add("score:" + i, storedData["score:" + i]);
            data.Add("date:" + i, storedData["date:" + i]);
        }
    }
}
else // ukoliko još nema spremljenih podataka
{   // postavljaju se svi rezultati na nulu
// ovaj se blok koda ne bi trebao nikada izvršavati pošto je to osigurano u
// funkciji LoadData(), ali postoji kao dodatno osiguranje
    data.Add("score:1", highScore);

    for(int i = 2; i< 6; i++)
    {
        data.Add("score:" + i, 0);
    }
}
// definiranje putanje gdje će se spremiti lokalna baza pri čemu se koristi
// Application.persistenDataPath o kojem Unity sam brine i pronalazi za svaku
        // platformu, kod Windows-a to je C:\Users\Ivan\AppData\LocalLow\DefaultCompany\AR_Legacy
// unutar koje se nalazi lokalna baza GameDataBase.json
string path = Application.persistentDataPath + "/GameDataBase.json";
        File.WriteAllText(path, data.ToString()); // zapisivanje cijelog teksta na temelju putanje
}

// ucitavanje iz lokalne baze, dohvaćanje JSON objekta
public JSONObject LoadData()
{
// ako postoje podaci da ih vrati u JSON formatu
        // prvo se pokušava pristupiti podacima u try bloku, ako ih nema prelazi se u catch blok
try
{   // definiranje putanje gdje se nalazi baza, podaci
    string path = Application.persistentDataPath + "/GameDataBase.json";
    string jsonString = File.ReadAllText(path); // čitanje zapisa
            JSONObject data = (JSONObject)JSON.Parse(jsonString); // parsiranje i cast-anje u JSONObject tip podatka
    return data; // vraćanje spremljenih podataka
}
```

```
        catch
        {// ukoliko ne postoje podaci da se inicijalno svi postave na 0 i stvori JSON file
         // (prvo učitavanje na gumb High Scores kada još nema spremljenih rezultata)

            JSONObject data = new JSONObject(); // definiranje novog JSON objekta
            // inicijalno postavljanje svih rezultata na 0 i datuma na: ~ pomoću for petlje
            for (int i = 1; i < 6; i++)
            {
                data.Add("score:"+i, 0);
                data.Add("date:"+i, "~");
            }
                    // definiranje putanje gdje će se spremiti podaci sa nazivom baze i formatom (GameDataBase.json)
            string path = Application.persistentDataPath + "/GameDataBase.json";
            File.WriteAllText(path, data.ToString()); // zapisivanje svih podataka
            return data; // vraćanje svih podataka koji su u JSON formatu
        }
    }
}
```

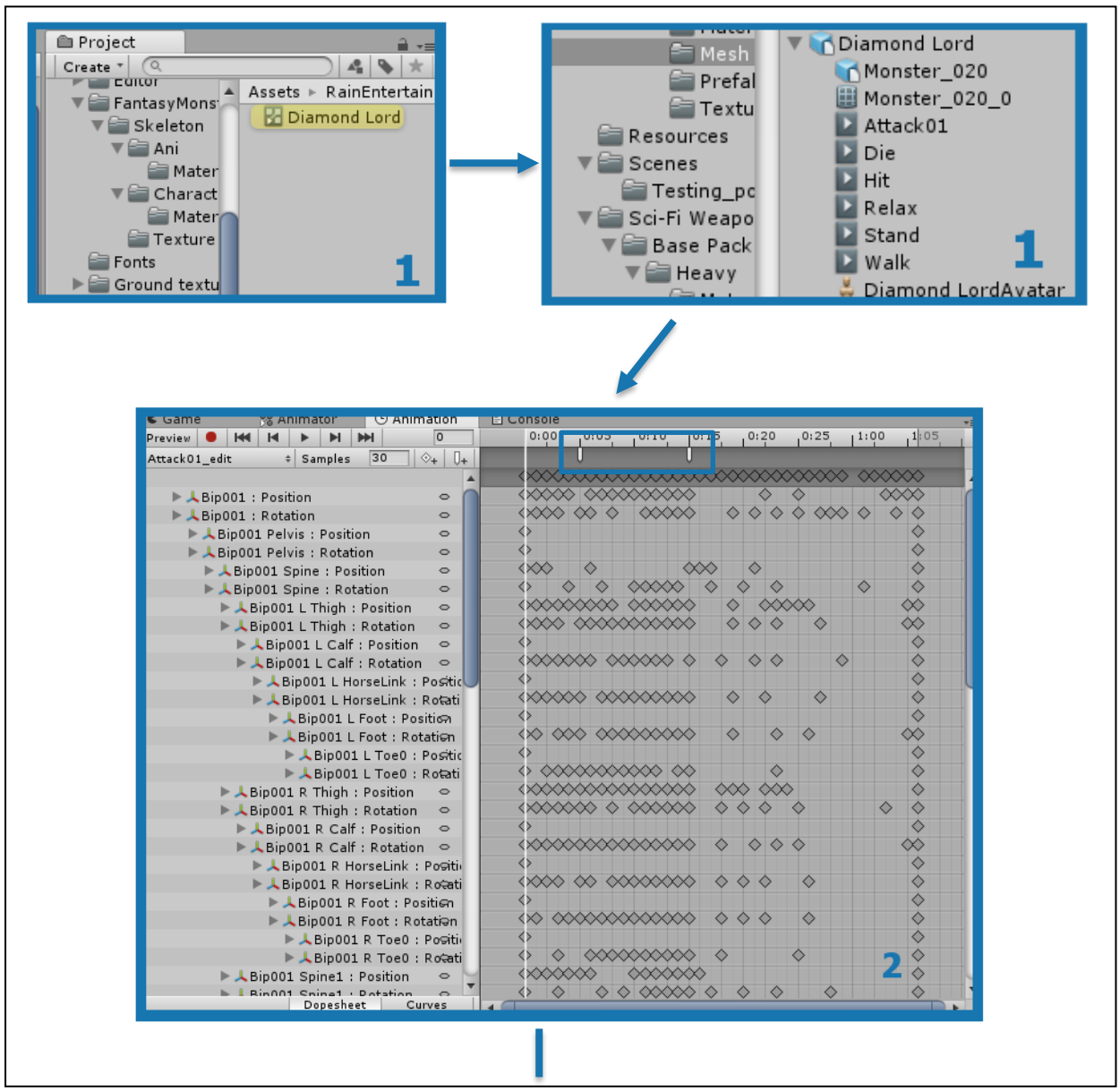
Sl. 4.48. Prikaz koda iz skripte LevelLoader koji se koristi za spremanje rezultata u lokalnu bazu

4.9. Animacije i Animator

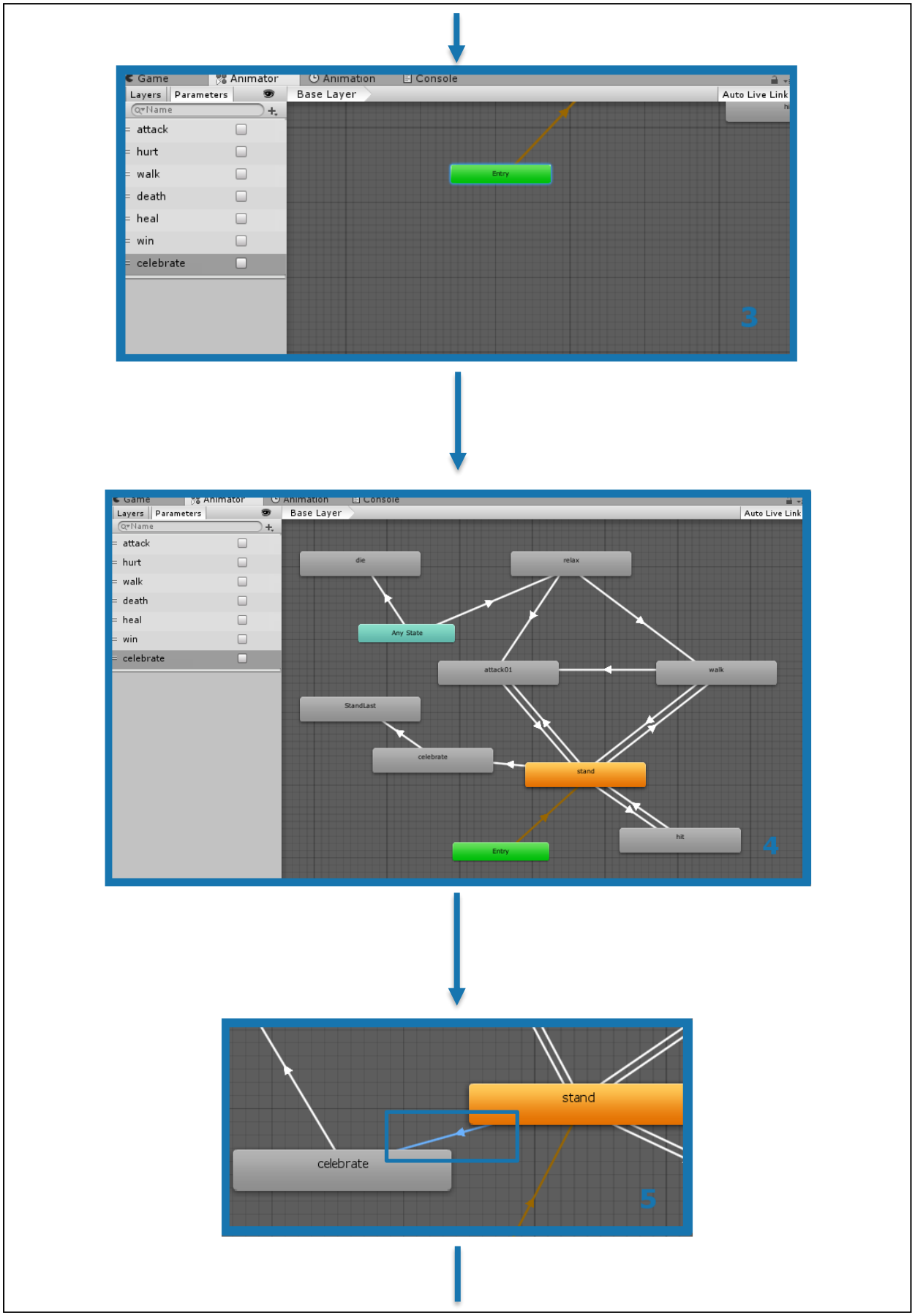
Svaku igru animacije vizualno obogaćuju i čine ju uvjerljivijom i igraču zanimljivijom. Razlog njihovog korištenja nije samo da bude vizualno bogatija već i da se ostvari dojam stvarnih pokreta likova u igri. U ovoj igri koriste se animacije koje su preuzete zajedno s modelima te su uređivane kako bi bile prilagođene tematici igre. Za njihovu kontrolu korišten je Animator.

Unity kao *game engine* u sebi ima već ugrađen sustav za animacije – ranije spomenuti Animator. Ovaj sustav definiran je konceptom korištenja animacijskih isječaka (*Animation Clips)*. Kontrola animacija ostvaruje se parametrima koji se mogu definirati unutar prozora animatora. Svaki objekt koji želi koristiti animacije mora imati svoj animacijski kontroler (*Animation Controller*). Kontroler izmjenjuje slijed izvođenja animacija na principu rada konačnog automata gdje se pri određenim uvjetima prelazi iz jednog stanja u drugo te izvršavaju određene animacije. Prvo je potrebno kreirati kontroler animacija, zatim postaviti animacijske isječke koji se trebaju izvršavati prilikom određenih uvjeta. Drugi je korak uređivanje animacija, podešavanje brzine izvođenja i dodavanje događaja (*eventa*) koji se trebaju izvršiti kod određenog trenutka animacije. Treći korak je definiranje parametara koji mogu biti tipa: *Float*, *Int*, *Bool* i *Trigger*. Četvrti korak je postavljanje prijelaza (*Transitions*) iz jednog stanja (animacije) u drugo stanje. Nakon postavljenih prijelaza između svih stanja, peti korak bi bio definiranje uvjeta na prijelazima (tranzicijama) na temelju kojih će se prijeći iz jednog stanja u drugo (iz jedne animacije u drugu animaciju). Šesti korak je dodavanje komponente *Animator* s ranije definiranim kontrolerom animacija. Navedeni slijed definiranja animatora i animacija je korišten kod izrade ove igre i ne predstavlja nikakav „standard“ za izradu animacija i korištenje animatora. U nastavku slijedi niz slika koje su

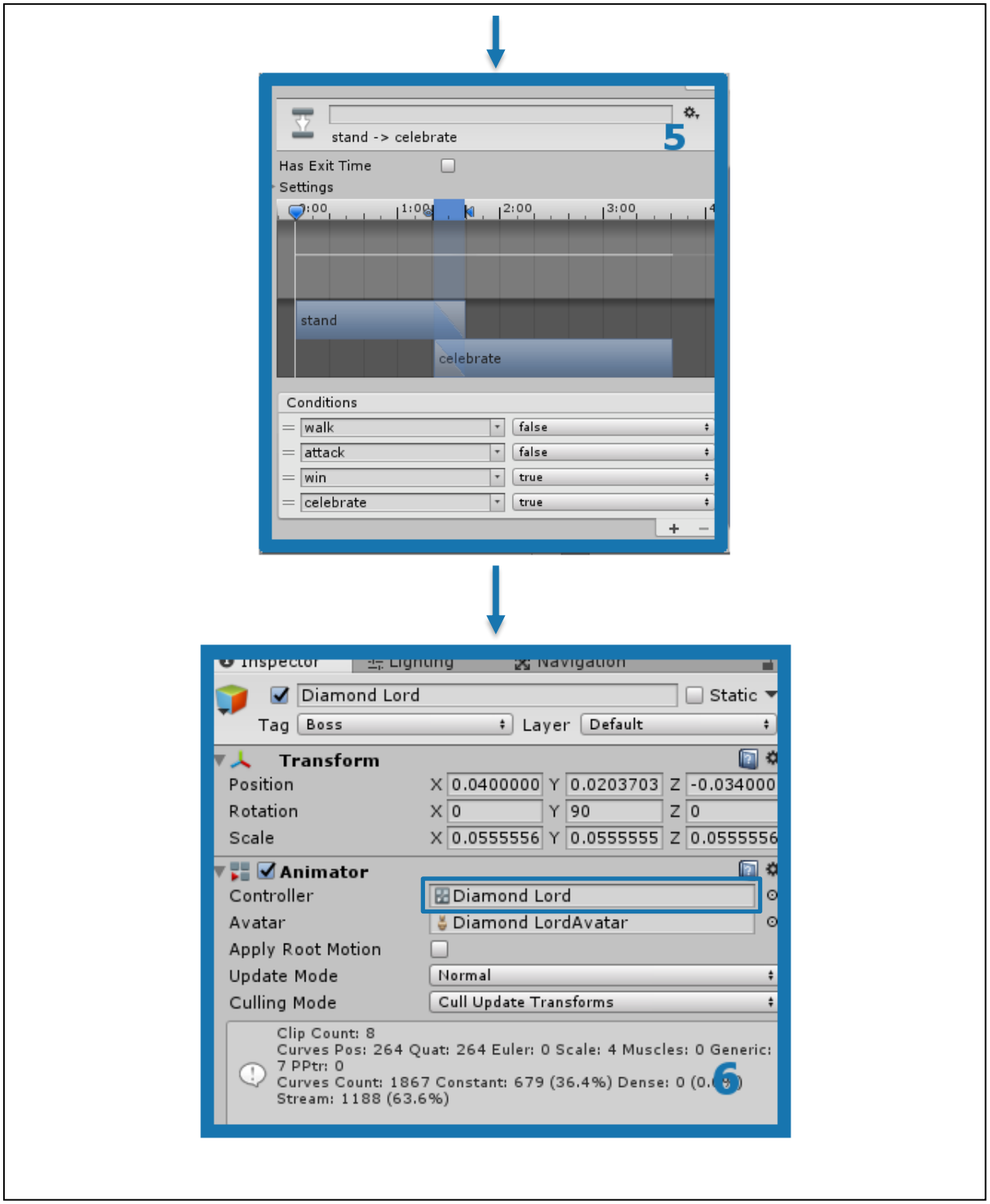
obilježene brojevima (koracima) kojim je ranije objašnjeno implementiranje animatora i animacija.



Sl. 4.49. Prikaz prva tri koraka prilikom implementacije animatora i animacija



Sl. 4.50. Prikaz iduća tri koraka prilikom implementacije animatora i animacija

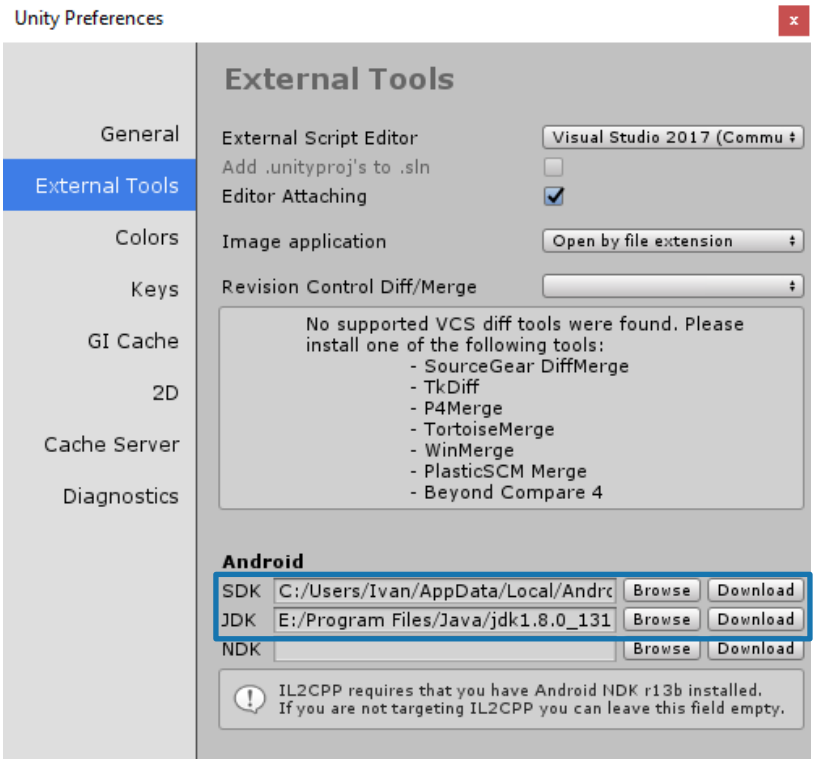


Sl. 4.51. Prikaz posljednja dva koraka prilikom implementacije animatora i animacija

Na ovaj je način implementiran animator i animacije za *bossa*, na isti način su implementirani animatori i animacije za kosture. Za pokretanje animacija, kontrolu animatora koristi se pristup iz skripte kao što je objašnjeno komentarima u skripti BossController (potpoglavlje 4.5.2. Neprijatelji).

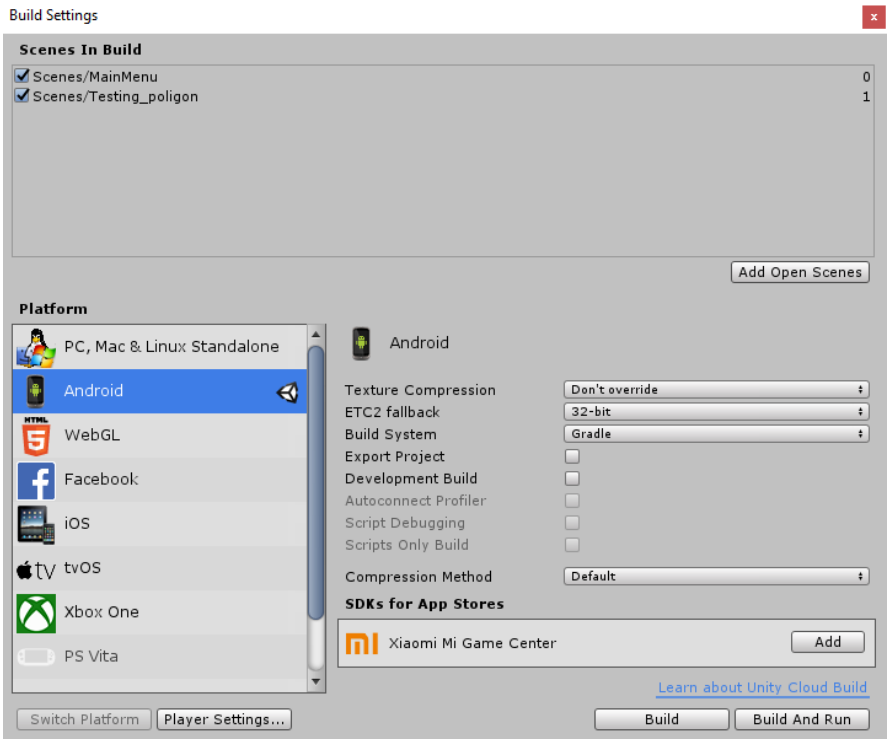
4.10. Game build - Android

Nakon što je igra napravljena unutar Unity Editora potrebno je napraviti build (export) igre za ciljanu platformu/platforme. Za ovu igru ciljana platforma je za mobilne uređaje, točnije Android. Kako bi se mogla koristiti, odabrati ova platforma, potrebno je imati instaliran modul Android unutar Unityja. Također, korištenje ove platforme zahtjeva da je na računalu instaliran program Android Studio s određenim verzijama android operacijskih sustava (*API levels*). Zatim je potrebno u postavkama (Edit – Preferences – External Tools) odabrati mjesto gdje se nalazi Android SDK (Software Development Kit) i java JDK (Java Development Kit). Izgled namještenih postavki može se vidjeti na slici ispod (Sl. 4.52.).



Sl. 4.52. Prikaz postavki SDK i JDK unutar Unityja za Android OS

Nakon što su uspješno postavljene putanje za SDK i JDK potrebno je unutar Unityja odabrati Android za trenutnu platformu (File – Build Settings). Prebacivanje s jedne na drugu platformu proces je koji traje oko desetak sekundi ovisno o brzini računala. Poslije ovog koraka potrebno je još samo namjestiti *Player Settings*. Na dnu istog prozora s lijeve strane se nalazi gumb Player Settings za otvaranje postavki. Primjer kako izgleda prozor u kojem se prebacuje platforma nalazi se na idućoj stranici.

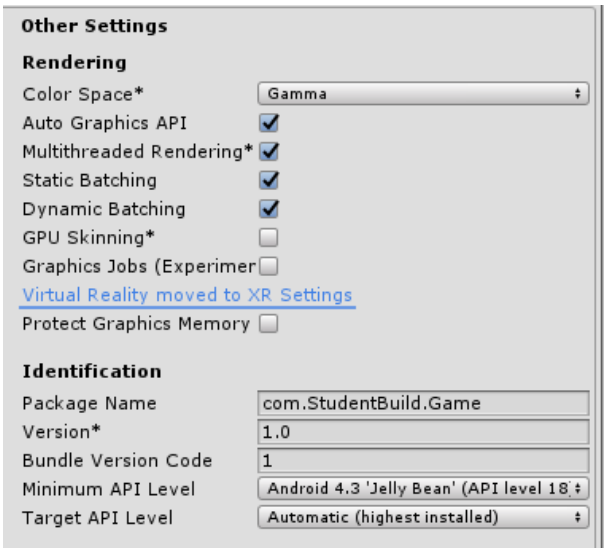


Sl. 4.53. Prikaz prozora Build Settings u kojem je odabrana platforma Android

Sada još preostaje samo definiranje postavki koje uključuju definiranje: rezolucije, naziva kompanije, verzije igre, ikona, prijelaznih ekrana, XR postavki, postavki za definiranje verzije Androida i ime paketa. Ovdje će biti prikazane samo najbitnije postavke. Slijedi slika (Sl. 4.54.) s omogućenim postavkama korištenja AR (*Augmented*) tehnologije koja mora biti odabrana kako bi bila implementirana tijekom *builda* igre.

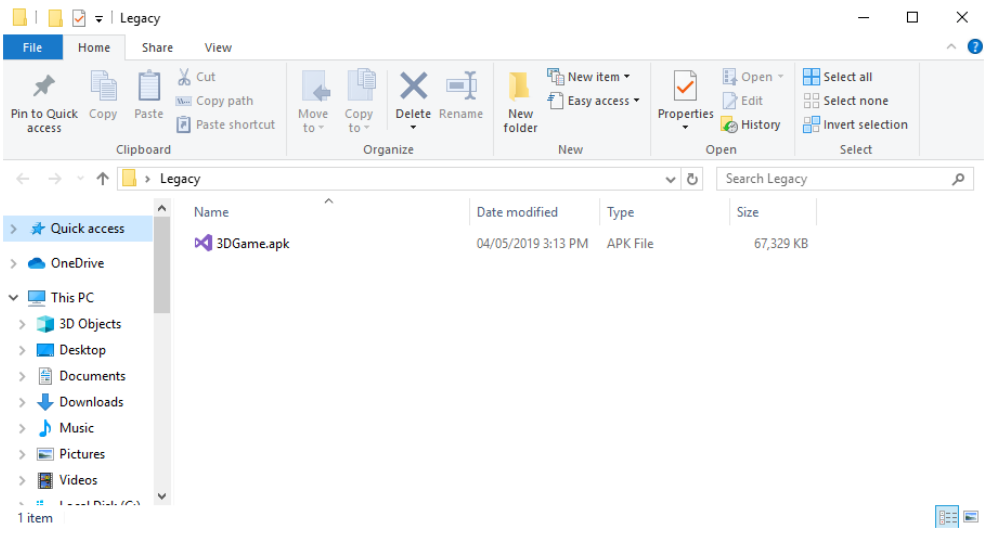
Sl. 4.54. Prikaz prozora Player Settings u kojem je odabrana tehnologija AR

Najbitnije postavke nalaze se u kategoriji *Other Settings* gdje se podešava ime paketa, verzija, broj *builda* te najmanji API level i ciljani. Bez ovih postavki nije moguće napraviti *build*. Plavim kvadratom su označene navedene postavke pri čemu je za ciljani API level postavljen Android Oreo (API 27). Primjer se nalazi u nastavku.



Sl. 4.55. Prikaz postavki za Android

Nakon ovog koraka može se napraviti *build* pri čemu je potrebno otvoriti prozor kao na slici (Sl. 4.53.) i pritisnuti tipku Build. Zatim se otvara novi prozor za odabir mjesta spremanja, a nakon toga proces kreiranja *builda* koji traje nekoliko minuta. Ispod se nalazi slika gotovog *builda* unutar mape Legacy pri čemu je ekstenzija .apk te je potrebno prebaciti igru na mobilni uređaj i instalirati.



Sl. 4.56. Prikaz gotovog builda naziva 3DGame.apk u mapi Legacy

5. ZAKLJUČAK

U ovom diplomskom radu napravljena je igra sa proširenom stvarnošću za mobilne uređaje (Android) korištenjem Vuforia programskog alata. Igra je napravljena u Unity Engineu unutar kojeg je naknadno implementirana Vuforia pri čemu je korišten C# programski jezik. Korištenje Vuforija alata nije jednostavno, te s njim dolaze i različiti problemi koji su navedeni i uspješno riješeni. U ovom radu obuhvaćen je cijeli proces izrade igre s proširenom stvarnošću od kreiranja Vuforija baze i rukovanja s tim alatom do implementacije, definiranja ideje i razrade cijele igre. Osim navedena dva programa, za realizaciju korisničkog sučelja korišten je program GIMP 2 kojim se može crtati i uređivati postojeće slike na jednostavan način, a opet je dovoljno dobar za potrebe izrade ove igre. Za spremanje i prikazivanje rezultata postignutih u igri implementirana je lokalna baza koja se automatski stvara na mobilnom uređaju. Kod izgradnje scene i dodavanja likova u igri korišteni su gotovi modeli na stranicama besplatnog sadržaja koji su zatim oblikovani i modificirani zbog potrebe, toka igre. Za animaciju likova unutar igre i njihovu kontrolu korišten je animator, sustav koji je ugrađen unutar Unityja. Kako bi igra bila zanimljivija igraču i „uvjerljivija“ popraćena je raznim melodijama i zvučnim efektima koji je obogaćuju. Na kraju, kada je igra napravljena (završena), napravljen je *build* kako bi bila dostupna za korisnike/igrače. Za izradu ovakve igre potrebno je posjedovati osnovno znanje objektnog programiranja, poznavanje programskog jezika i razumijevanje koncepta kako se objekti mogu kontrolirati pomoću skripte. Bitno je imati i osnovno znanje iz područja matematike i fizike radi lakšeg razumijevanja odnosa između više objekata. Ovaj rad se može još proširiti i upotpuniti dodavanjem više nivoa u igri, implementacijom sustava za spremanje rezultata pri čemu se uspoređuju s drugim igračima preko interneta (*online*). Kako bi igra bila prilagođenija korisnicima moguće je dodati opciju za biranje jezika (internacionalizacija). Zanimljivo bi bilo kada bi igru istovremeno mogla igrati dva igrača (*co-op*[11]) i na taj način zajedno prelaziti nivo.

[11] *Co-op (eng, Cooperative gameplay) – odabir načina igre kod koje više igrača može zajedno prelaziti igru, a pritom nije preko interneta*

LITERATURA

[1] Unity Asset Store, Unity Technologies, 22.06.2019.
https://assetstore.unity.com/packages/3d/environments/fantasy/make-your-fantasy-game-lite-8312

[2] SoundsCrate, ProductionCrate, 18.05.2019.
https://soundscrate.com/epic.html

[3] Freesound, Music Technology Group - Freesound, 06.03.2019.
https://freesound.org

[4] BusinessofApps, Business of Apps, 10.06.2019.
http://www.businessofapps.com/data/pokemon-go-statistics

[5] Unity Documentation, Unity Technologies, 22.06.2019.
https://docs.unity3d.com/Manual/vuforia-sdk-overview.html

[6] A designer's guide to hardware and software for mobile AR, UX Collective, 12.06.2019.
https://uxdesign.cc/designers-guide-to-hardware-and-software-for-mobile-ar-5c800503676d

[7] The History of Augmented Reality, Colocation America, 12.06.2019.
https://www.colocationamerica.com/blog/history-of-augmented-reality

[8] Unity Technologies, Wikipedia, 14.06.2019.
https://en.wikipedia.org/wiki/Unity_Technologies

[9] Unity for all, Unity Tecnologies, 14.06.2019.
https://unity.com

[10] Build once, deploy anywhere, Unity Tecnologies, 15.06.2019.
https://unity3d.com/unity/features/multiplatform

[11] Motion sensors, Google Developers, 15.06.2019.
https://developer.android.com/guide/topics/sensors/sensors_motion.html

[12] Unity Asset Store, Unity Technologies, 16.06.2019.
https://assetstore.unity.com/packages/3d/characters/humanoids/fantasy-monster-skeleton-35635

[13] Unity Asset Store, Unity Technologies, 16.06.2019.
https://assetstore.unity.com/packages/3d/characters/humanoids/diamond-lord-139721

[14] DevAssets, Unity Brackeys, 16.06.2019.
http://devassets.com/assets/sci-fi-weapons

[15] Graphical User Interface, Techopedia Inc, 16.06.2019.
https://www.techopedia.com/definition/5435/graphical-user-interface-gui

[16] Downloads, The GIMP Team, 17.06.2019.
https://www.gimp.org/downloads

[17] User Manual, The GIMP Team, 17.06.2019.
https://docs.gimp.org/2.10/en

[18] User Manual, The GIMP Team, 17.06.2019.
http://wiki.unity3d.com/index.php/SimpleJSON

SAŽETAK

U ovom diplomskom radu obrađena je 3D mobilna igra sa proširenom stvarnošću korištenjem programa Unity Engine i Vuforia programskog alata. Glavni cilj igre je obraniti kip koji sadrži energiju života, pri čemu je potrebno poraziti sve neprijatelje (kosture) i njihovog glavnog vođu (Diamond Lord). Igrač igra iz prve perspektive, pri čemu može koristit četiri vrste oružja (pištolj, automatska puška, strojnica i snajper). Igračeve kretnje su definirane fizičkim pomicanjem mobilnog uređaja (korištenjem ugrađenog akcelerometara, žiroskopa) u odnosu na metu (*Image Target*) koja je definirana korištenjem Vuforija alata. Ovaj alat je implementiran kao paket u Unity Editoru. Definirana scena se prikazuje u odnosu na prepoznatu metu (2D sliku) u prostoru i na taj način je ostvarena interakcija između stvarnog i virtualnog svijeta. Unatoč implementacije Vuforija alata, korištene su skripte za upravljanje objektima koje su pisane u C# programskom jeziku. Grafičko, korisničko sučelje izrađeno je u GIMP 2 programu, dok su za izradu scene korišteni modeli koji su preuzeti sa stranice besplatnog sadržaja, kao i melodije te zvučni efekti. Prilikom izrade animacija korišten je ugrađeni sustav Unityja, *Animator*. U svrhu spremanja rezultata u lokalnu bazu korišten je JSON format, a implementacija je ostvarena korištenjem SimpleJSON *plugina*. U igri ukupno postoje dvije scene. Prva predstavlja glavni izbornik u kojem igrač može vidjeti trenutno ostvarene rezultate, podesiti postavke i započeti s igrom. Druga scena je nivo koji se učitava kada je prepoznata meta (*Image Target*) u kojoj ishod može biti pobjeda ili poraz.

Ključne riječi: 3D mobilna igra, Image Target, SimpleJSON plugin, Unity Engine, Vuforia alat

ABSTRACT

In this final paper 3D augmented reality mobile game was processed. It was made combining Unity Engine and Vuforia software. The main purpose of the game is to defend sculpture which contains energy of life, while player needs to defeat all enemies (skeletons) and their boss (Diamond Lord). The player plays from the first perspective and can use four types of weapons (pistol, automatic gun, machine gun and sniper). The player movements are defined by moving mobile phone physically (using embedded accelerometer, gyroscope) in relation to the target (*Image Target*) which is defined by Vuforia software tool. This tool is implemented into Unity Editor as package. Defined scene is rendered after target (2D image) is recognized in the physical surrounding and that is the way of accomplishing interaction between real and virtual world. Despise using Vuforia tool, bunch of scripts were used for controlling objects which are coded in C# programming language. Graphics, user interface was made by using GIMP 2 software, while for making scenes the models were used. The models were downloaded from free-content sites, same as melodies and sound effects. During process of creating animations, embedded system of Unity, *Animator* was used. In order to save the results to the local base, JSON format was used, and implementation was accomplished using the SimpleJSON plugin. There are two scenes in the game. The first scene represents the main menu in which player can see current achieved results, setup settings and start with the game. The second scene represents level which is loaded, rendered when target (*Image Target*) is recognized. The outcome of level can be victory or defeat.

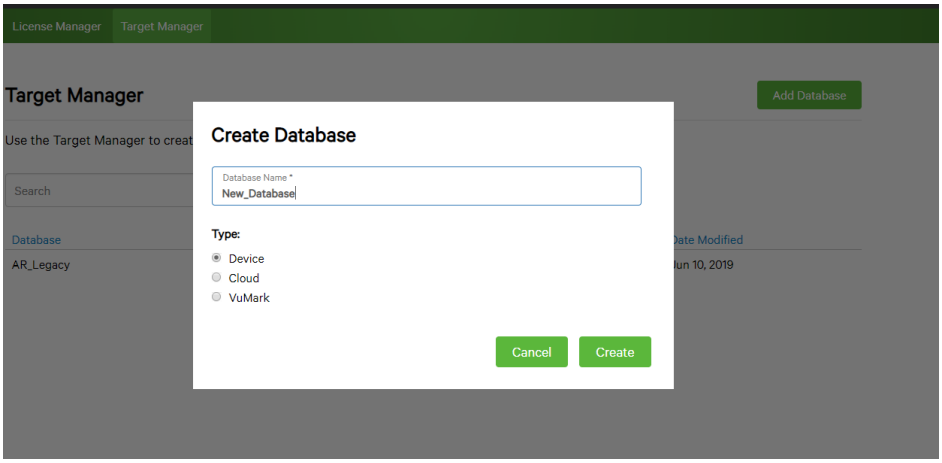
Key words: 3D mobile game, Image Target, SimpleJSON plugin, Unity Engine, Vuforia software tool

ŽIVOTOPIS

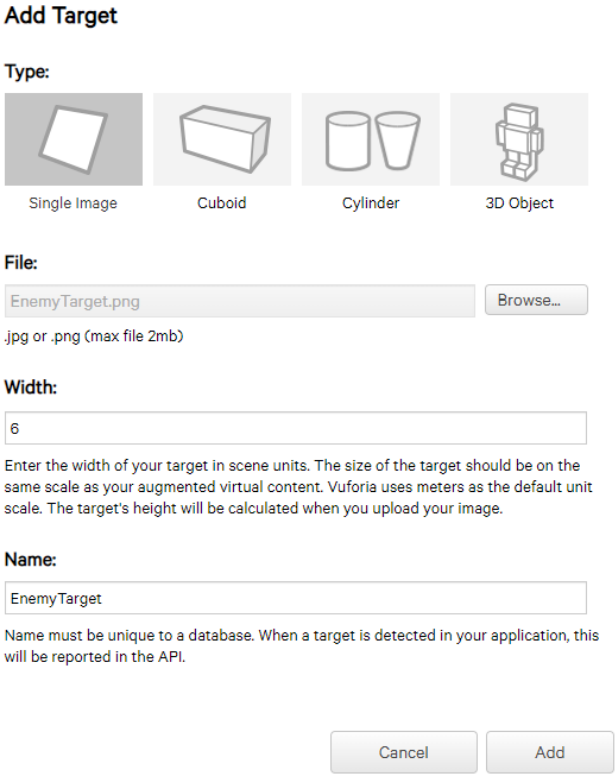
Ivan Drulak rođen je 26. rujna 1995. godine u Virovitici. Od 2002. do 2006. pohađao je OŠ Ivana Gorana Kovačića, a od 2006. do 2010. Vladimira Nazora u Virovitici. Nakon završene osnovne škole upisao je srednju školu, gimnaziju Petra Preradovića u Virovitici koju je završio 2014. godine. Također je te iste godine upisao redovni preddiplomski studij računarstva na Elektrotehničkom fakultetu u Osijeku. 2017. godine završio je preddiplomski studij računarstva i upisao diplomski studij, izborni blok Informacijske i podatkovne znanosti koji još uvijek pohađa. Još kao dječak bio je ljubitelj računalnih igara te ljubitelj sporta i fizičkih aktivnosti poput vožnje biciklom.

Vlastoručni potpis: _____

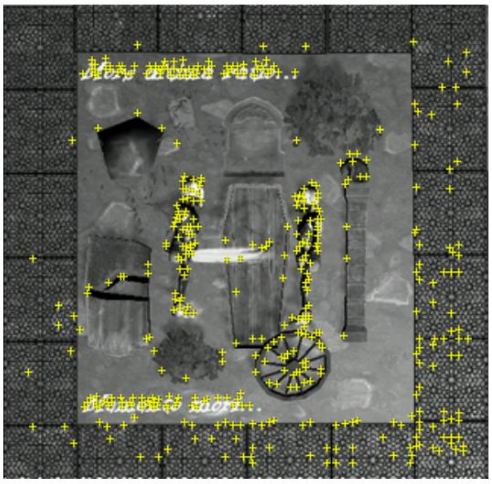
PRILOZI



Slika 1. Prikaz kreiranja nove baze



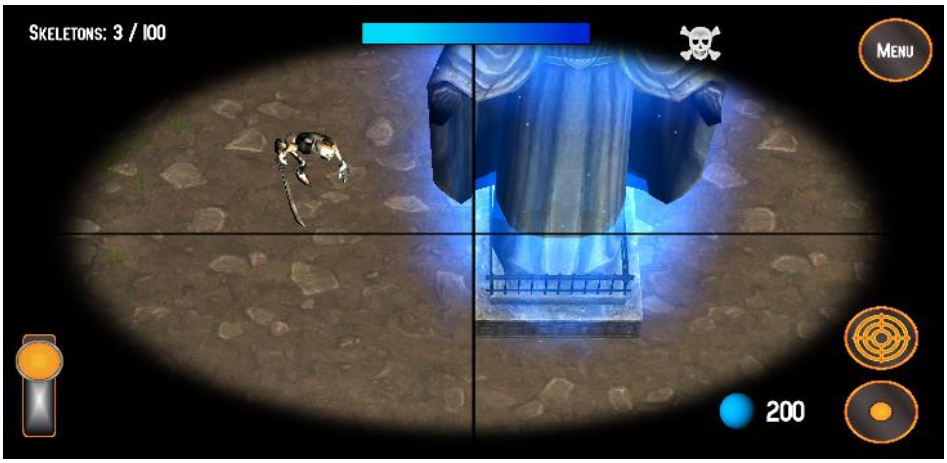
Slika 2. Prikaz dodavanja nove mete s definiranom širinom



Slika 3. Prikaz značajki mete EnemyTarget

Slika 4. Pozadinska slika koja se prikazuje prilikom učitavanje scene

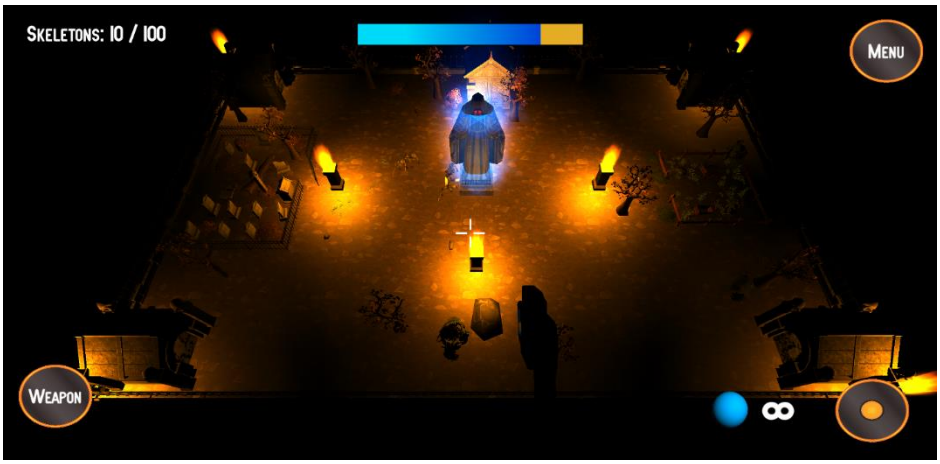
Slika 5. Prikaz početka treće runde s indikatorom da je otključano novo oružje, strojnica



Slika 6. Prikaz korištenja optike

a) *b)*

Slika 7. Prikaz odabira oružja a) u prvoj rundi i b) četvrtoj



Slika 8. Prikaz nivoa kada je u sceni noć

Slika 9. Prikaz nivoa u trenutku kada je vođa (boss) stvoren

Slika 10. Prikaz trenutka kada se vođa (boss) regenerira i inicira stvaranje novih kostura

Slika 11. Prikaz napada kostura i vođe (Diamond Lorda) kada je u sceni dan

Slika 12. Prikaz napada kostura i vođe (Diamond Lorda) kada je u sceni noć

Slika 13. Prikaz scene kada je igrač izgubio

Slika 14. Prikaz scene kada je igrač pobijedio