Android igra za vježbanje refleksa

Pjevačević, Milan

Undergraduate thesis / Završni rad

2016

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **Josip Juraj Strossmayer University of Osijek, Faculty of Electrical Engineering, Computer Science and Information Technology Osijek / Sveučilište Josipa Jurja Strossmayera u Osijeku, Fakultet elektrotehnike, računarstva i informacijskih tehnologija Osijek**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:200:937523

Rights / Prava: In copyright / Zaštićeno autorskim pravom.

Download date / Datum preuzimanja: **2024-12-23**

Repository / Repozitorij:

Faculty of Electrical Engineering, Computer Science and Information Technology Osijek

SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU
ELEKTROTEHNIČKI FAKULTET

Stručni studij

ANDROID IGRA ZA VJEŽBANJE REFLEKSA

Završni rad

Milan Pjevačević

Osijek, 2016.g.

SADRŽAJ

1. UVOD

Tema ovog završnog rada je kreiranje igre za pametne telefone koji rade na Android operativnom sustavu a koristit će ju djeca za vježbanje refleksa. Zadatak igre je bušiti balone koji se kreću prema gore iglom na vrhu ekrana kojom se upravlja senzorom pokreta (žiroskopom). Igrači će dobivati određeni broj bodova za svaki probušeni balon te će se na kraju igre pomoću tih bodova kreirati rang lista najboljih igrača. Igre na mobilnim uređajima postoje te su bile popularne i prije dolaska pametnih telefona, a pametni telefoni su ih još više popularizirali i sada zauzimaju velik dio trgovina aplikacijama raznih operativnih sustava. Kroz poglavlja ovog završnog rada pobliže će se objasniti kreiranje igara u najpopularnijem alatu za razvoj igara za Android operativni sustav, Unity 3D-u. Poglavlja će pokrivati sve od uvoda i teorijskog dijela koji je potreban za razumijevanje svih tehnologija i alata koji se koriste do samog izvođenja i stvaranja igre. Nakon uvodnog dijela u drugom poglavlju će se nalaziti teorija koja će objašnjavati platformu za koju se kreira igra, alate i programski jezik koji se koriste za kreiranje. U trećem poglavlju će biti detaljno objašnjeni najbitniji koraci u izvedbi, a u četvrtom poglavlju će biti rezultati rada iz prijašnjih poglavlja, gotov izgled igre te sve njene funkcionalnosti.

1.1 Zadatak završnog rada

Zadatak ovog završnog rada je kreirati igru sa svim njenim elementima za Android operativni sustav. Igra mora imati avatar koji će se pokretati pomoću senzora pokreta i rang listu najboljih rezultata. Za to ćemo koristiti Unity 3D razvojni alat, C# za dodjelu osobina raznim objektima i njihovo ponašanje te udaljenu bazu podataka za rang listu najboljih rezultata.

2. TEORIJA

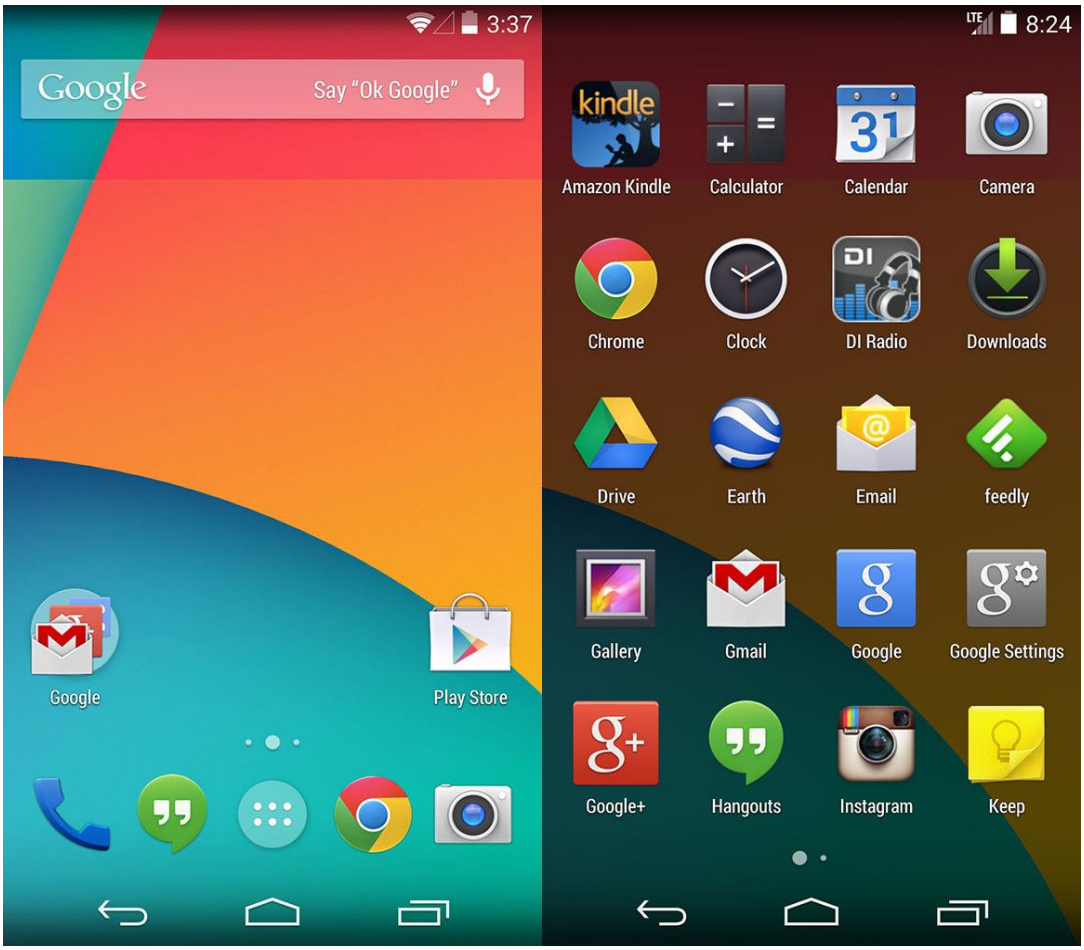
U ovom poglavlju bit će objašnjen teorijski dio završnog rada za lakše razumijevanje metoda i pojmova u samom kreiranju igre. U potpoglavljima će se pobliže objasniti i upoznati s Android operativnim sustavom, *Unity Engineom*, programskim jezikom C# korištenim za kreiranje igara.

2.1 Android operativni sustav

Android je operativni sustav za mobilne uređaje koji trenutno razvija Google. Zasnovan je na Linux jezgri i dizajniran primarno za mobilne uređaje sa zaslonom na dodir kao što su pametni telefoni i tableti. Osim uređaja sa zaslonom na dodir, Google je dodatno razvio Android TV za televizore, Android Auto za automobile i Android Wear za ručne satove, svaki s prilagođenim korisničkim sučeljem. Inačice Android operativni sustava se koriste i na igraćim konzolama, digitalnim kamerama i ostalim elektroničkim uređajima. U početku ga je razvijao Android Inc., kojeg Google kupuje 2005. godine. Android je predstavljen 2007. godine zajedno s osnivanjem Open Handset Alliance grupe čiji je cilj ubrzati inovacije na području operativni sustava za mobilne uređaje. Prvi komercijalno dostupan pametni telefon s Android operativni sustavom bio je HTC Dream, predstavljen 22.10.2008. godine.

2.1.1 Korisničko sučelje

Upravljanje korisničkim sučeljem Androida radi se dodirima po ekranu osjetljivom na dodir, oponašaju se stvarne radnje kako bi se manipuliralo objektima na ekranu zajedno s virtualnom tipkovnicom. Odziv na korisnikove naredbe je napravljen da bude bez kašnjenja te zbog toga pruža fluidno korisničko sučelje. Neke aplikacije koriste i hardver kao što su žiroskop, brzinometar i senzor udaljenosti kako bi reagirali na ostale korisničke radnje kao što su rotacija ekrana iz vodoravnog u horizontalni položaj ili za omogućivanje korisniku da upravlja vozilom u igrama okrećući uređaj. Android uređaji se pri pokretanju stavljaju na početni zaslon koji je primarno sredstvo za navigaciju u korisničkom sučelju Androida, kao što je i *desktop* na osobnim računalima. Početni zaslon Androida se uobičajeno sastoji od ikona aplikacija i widgeta, može se sastojati od nekoliko zaslona između kojih korisnik može listati te je vrlo prilagodljiv što omogućuje korisnicima prilagođavanje izgleda korisničkog sučelja prema svojim željama. Na vrhu ekrana nalazi se statusna traka koja pokazuje informacije o uređaju kao što su vrijeme, stanje baterije, snaga signala i drugo. Statusna traka se može povući prema dolje kako bi se prikazao zaslon obavijesti gdje aplikacije prikazuju razne informacije i obavijesti kao što su primljena elektronička pošta ili SMS poruka, na način koji ne smeta korisniku pri korištenju uređaja.



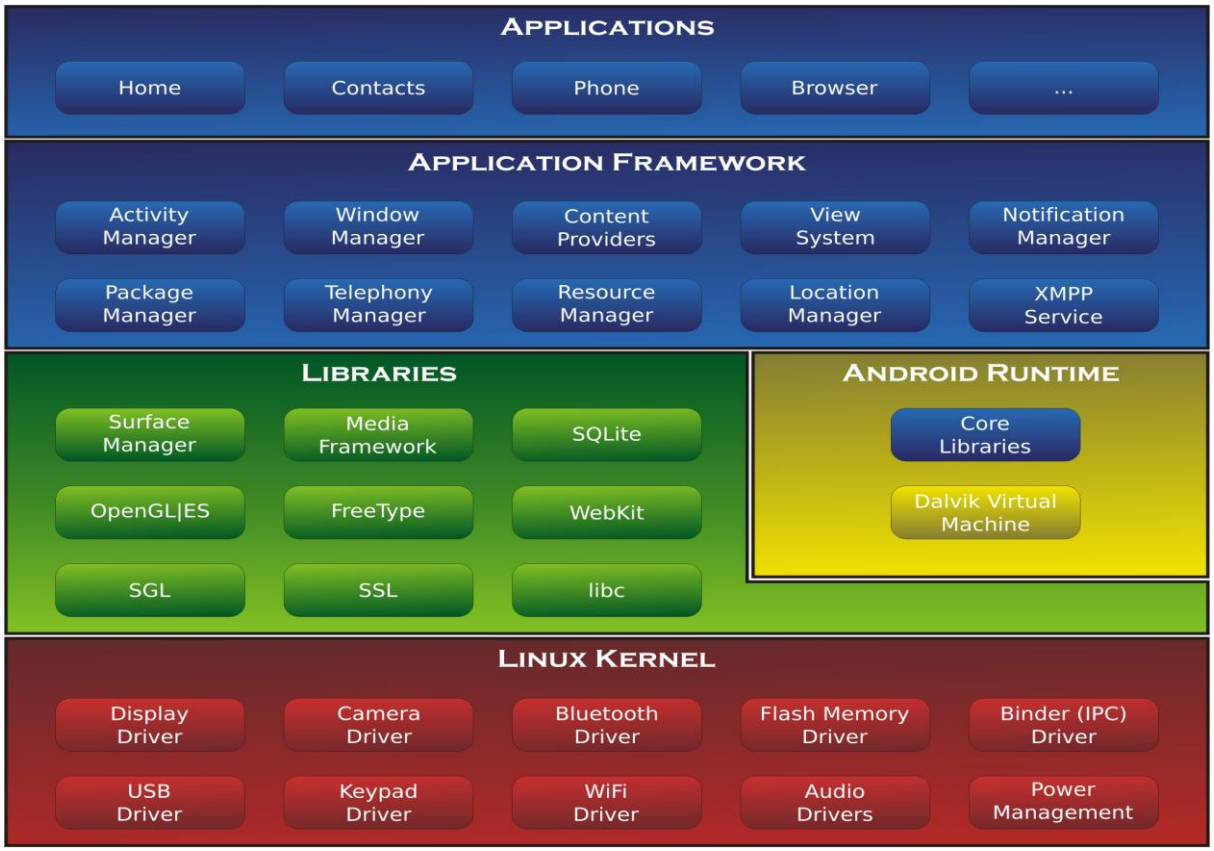
Sl. 2.1. Početni zaslon i izbornik Androida

2.1.2 Aplikacije

Aplikacije za Android su pisane pomoću Android alata za razvoj softvera (Android Software Development Kit- SDK) i često pomoću Java programskog jezika s potpunim pristupom sučelju za programiranje aplikacija. SDK dolazi s opsežnim setom razvojnih alata uključujući *debugger*, programske biblioteke, emulatore, dokumentaciju… U početku Googleovo integrirano razvojno okruženje (IDE) bio je Eclipse, a u prosincu 2014. godine predstavljaju Android Studio kao primarni IDE za razvijanje Android aplikacija. Android ima velik broj aplikacija koje su razvili sami korisnici i za koje se može skinuti i instalirati APK (Android application package) datoteka ili ih se može preuzeti pomoću trgovine aplikacija koja dopušta korisnicima instaliranje, ažuriranje i uklanjanje aplikacije s uređaja. Google Play Store je primarna trgovina aplikacija za Android uređaje a dopušta korisnicima pretraživanje, skidanje i ažuriranje aplikacija objavljene od strane Googlea ili samih korisnika. Do srpnja 2013. godine na Google Play Store postavljeno je više od milijun aplikacija s više od 50 milijardi preuzimanja.

2.1.3 Linux jezgra

Jezgra operativnog sustava brine o upravljanju memorijom, procesima, mrežnim sučeljima i ostalim sustavima na sklopovskom nivou, obrađuje osnovne usluge sustava i djeluje kao HAL (Hardware Abstraction Layer), među sloj između fizičkog sklopovlja i Android operativnog sustava. Tvorcima programske potpore jezgra nije dostupna. Android operativni sustav se temelji na modificiranoj Linux jezgri (kernel) 2.6, nekoliko upravljačkih programa i biblioteka je izmijenjeno ili novorazvijeno, ne sadrži izvorni X-Window sustav, niti podržava cijeli skup standardnih GNU biblioteka, a sve u svrhu omogućavanja učinkovitijeg rada Android mobilnih uređaja. Najveća razlika između Linux i Android sustava je u Java sloju apstrakcije ugrađenom u Android. Android aplikacije su dalje od same jezgre nego u Linuxu, imaju duži kod put do OS sloja. U Linuxu, korisničke aplikacije (preko knjižnica i podsustava za sistemske pozive) imaju izravan pristup jezgri.

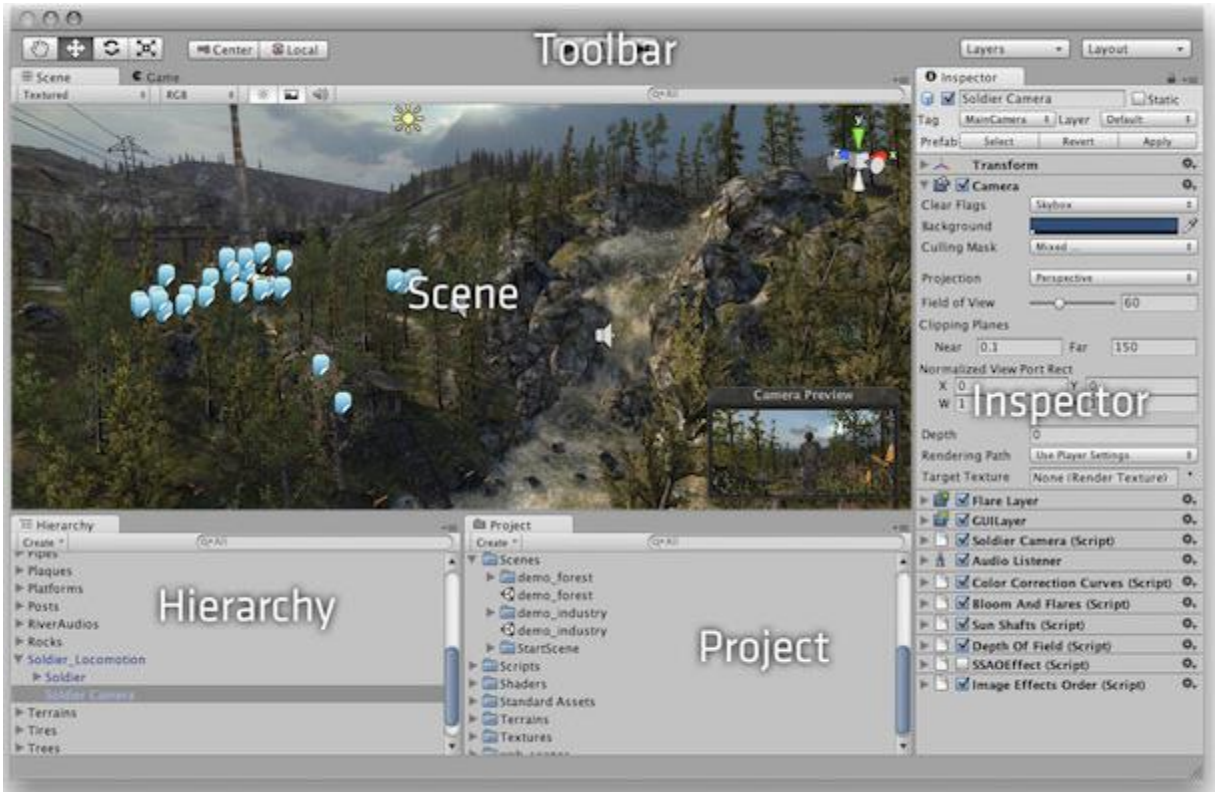


Sl. 2.2. Arhitektura Android operativnog sustava (en.wikipedia.org/wiki/Android_(operating_system))

2.2 Unity engine

Unity je potpuno integrirani *engine* za razvoj koji pruža veliku funkcionalnost i mogućnosti kod kreiranja igara i ostalog interaktivnog 3D sadržaja. U Unity-u se može složiti grafika i *assets* kako bi se napravile scene i okolina, dodalo osvjetljenje, zvuk, specijalni efekti, fizička svojstva objektima i animacija, u isto vrijeme testirao i uređivao projekt, te objaviti za željenu platformu kao što je Mac, PC i Linux desktop računala, iOS, Android, Windows Phone 8, Blackberry10, Wii U, PlayStation4, Xbox One, Unity Web Player… Unity nudi besplatnu i *Unity pro* verziju koja dolazi s više alata i mogućnosti.

Kada se u Unityu započinje novi projekt kreirat će se folder u kojemu će se nalaziti svi resursi za igru: sadržavat će foldere *assets*, *library* i *project settings*. *Assets* su svi resursi koje će igra koristiti. Oni uključuju 3D modele, materijale, teksture, audio, skripte i fontove između ostalog. Osim nekoliko osnovnih objekata kao što su kocka ili sfera, Unity ne može kreirati većinu tih asseta. Za to se mora koristiti neka druga aplikacija za 3D modeliranje nakon koje se taj model uvozi u Unity. Unity podržava sve popularne 3D podatkovne formate kao što su Maya, 3D Studio Max, Blender i FilmBox sa svim njihovim materijalima i teksturama ne promijenjenima. Također podržava i sve uobičajene slikovne podatkovne formate, uključujući PNG, JPEG, TIFF. Od audio formata Unity podržava WAV i AIF što je idealno za zvučne efekte, a MP3 i OGG za glazbu u igri. Unity ima i *Asset Store*, trgovinu gdje korisnici mogu kupiti ili besplatno naći 3D modele, teksture, zvučne efekte, glazbu, alate pa čak i skripte.



Sl. 2.3. Korisničko sučelje Unitya
(docs.unity3d.com/410/Documentation/Manual/LearningtheInterface.html)

Korisničko sučelje Unitya je jako dobro organizirano, svi paneli se mogu složiti kako odgovara korisniku. U *Project* panelu se nalaze svi *assets* koji se koriste u projektu. Kada se novi *assets* uvezu prvo se tu pojavljuju. U panelu *hierarchy* se svi *assets* organiziraju u scenu, *assets* iz *project* panela se mogu prevući u *hierarchy* panel kako bi se dodali u trenutnu scenu. U *hierarchy* panelu će se nalaziti organizirano svi objekti koji su u trenutnoj sceni. *Inspector* panel dopušta pregled i namještanje atributa odabranog *asseta*, sve od pozicije i rotacije do toga da li na njega utječe gravitacija i da li će stvarati sjene. *Scene* panel omogućuje 3D pregled gdje se fizički mogu razmjestiti *asseti* pomjerajući ih kroz 3D prostor. Scene u Unityu sadržavaju objekte igre, a mogu se koristiti kako bi se napravio glavni izbornik, zasebni leveli i slično. U svakoj sceni se zasebno stavljaju okolina, prepreke i dekoracije, gradeći igru u dijelovima. Kada se kreira novi projekt scena će sadržavati samo kameru, a sve druge objekte u igri treba dodati korisnik.

Skriptiranje je osnovni dio u svim igrama. Čak i najjednostavnije igre trebaju skripte kako bi reagirale na upute igrača te kako bi se radnje u igri izvršile. U kompleksnijim igrama skripte se koriste za kreiranje grafičkih efekata, upravljanje ponašanjem objekata pa čak i za implementiranje

umjetne inteligencije za likove u igri. Više skripti se može staviti na jedan objekt. Unity podržava tri različita programska jezika: UnityScript (sličan JavaScriptu), C# i Boo.

Sve skripte imaju *Start()* i *Update()* metodu. *Start()* metoda se pokreće samo jednom i to kada je objekt prvi puta kreiran, a *Update()* metoda se pokreće jednom po svakom okviru.

2.3 C# programski jezik

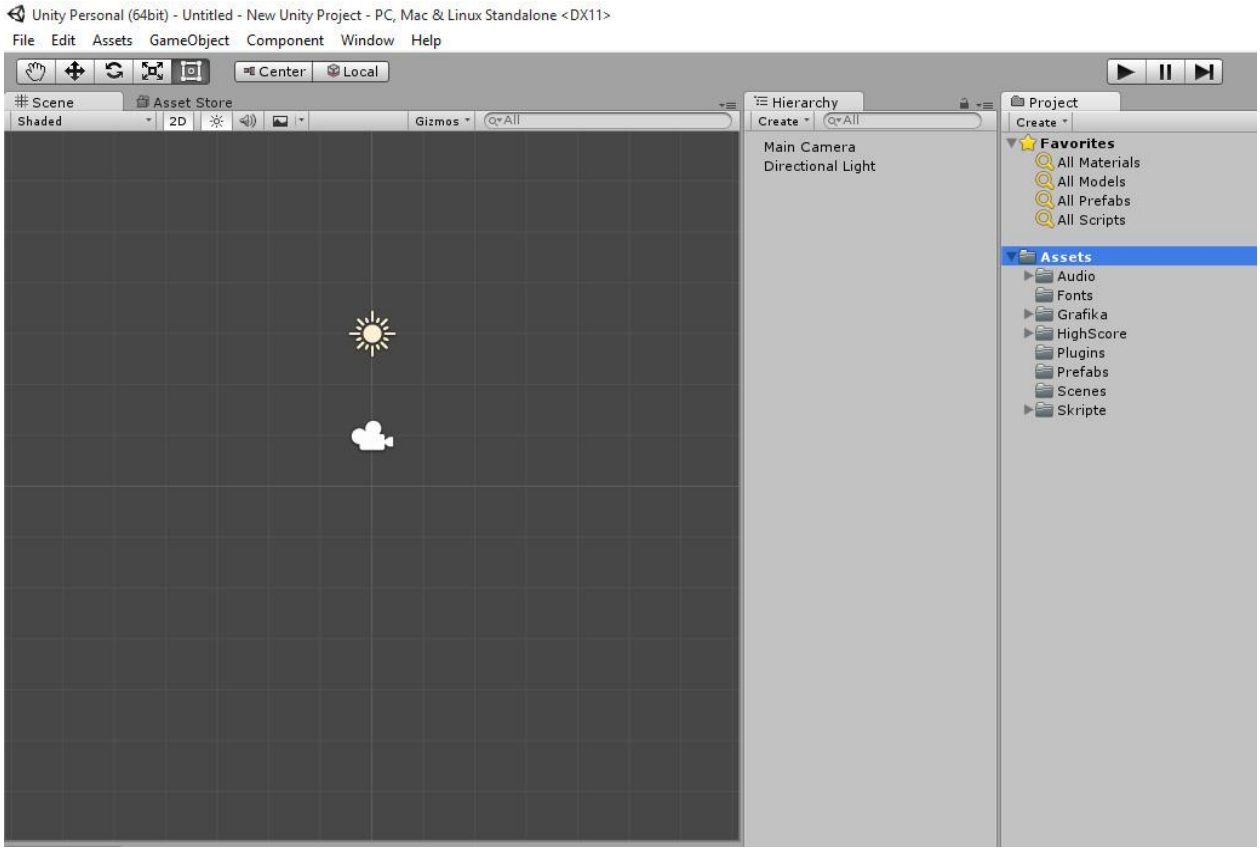
C# (CSharp) je nastao u tvrtki Microsoft i razvijen je od strane tima stručnjaka predvođenih s Anders Hejlsberg-om i Scott Wiltamuth-om. Na tržištu se pojavio 2000. godine zajedno sa .NET platformom. C# je nastao s ciljem da bude jednostavan, siguran, moderan, objektno orijentiran jezik visokih performansi za .NET platformu. C# je nastao na temelju objektnih jezika Java, C++ i Visual Basic. Vrlo je sličan Javi i C++ jeziku (sintaksa i semantika je dobrim dijelom preuzeta iz Jave, koja je kao i C# potpuno objektno orijentirani jezik). Ali C# za razliku od Jave nije neovisan o platformi, tj. operativnom sustavu, već je kreiran za izradu stolnih (desktop) i Internet aplikacija u .Microsoft .NET okruženju.

C# sadrži sve dobre odlike potpuno objektnog programskog jezika (koje većinom preuzima iz C++ i Jave), a u sklopu .NET platforme omogućava kreiranje vizualnih aplikacija čak i onim korisnicima koji nemaju programerskog iskustva. C# ima velike mogućnosti u definiranju klasa (tipova objekata), novih metoda i svojstava, te korištenju enkapsulacije, nasljeđivanja i polimorfizma kao što je to omogućeno u C++ i Javi. Također podržava XML stil unutar dokumenata, sučelja, svojstva, događaje te podržava rad s pokazivačima.

3. IZRADA PROJEKTA

U ovom poglavlju će se detaljno opisati svi koraci u kreiranju ove igre uz primjere i objašnjenja. Prije samog kreiranja igre potrebno je besplatno skinuti program Unity i dobro se upoznati s korisničkim sučeljem, mogućnostima te samim programom Unity. Potrebno je i vrlo dobro poznavanje C# jezika zato što su skripte ključan dio u kreiranju igara.

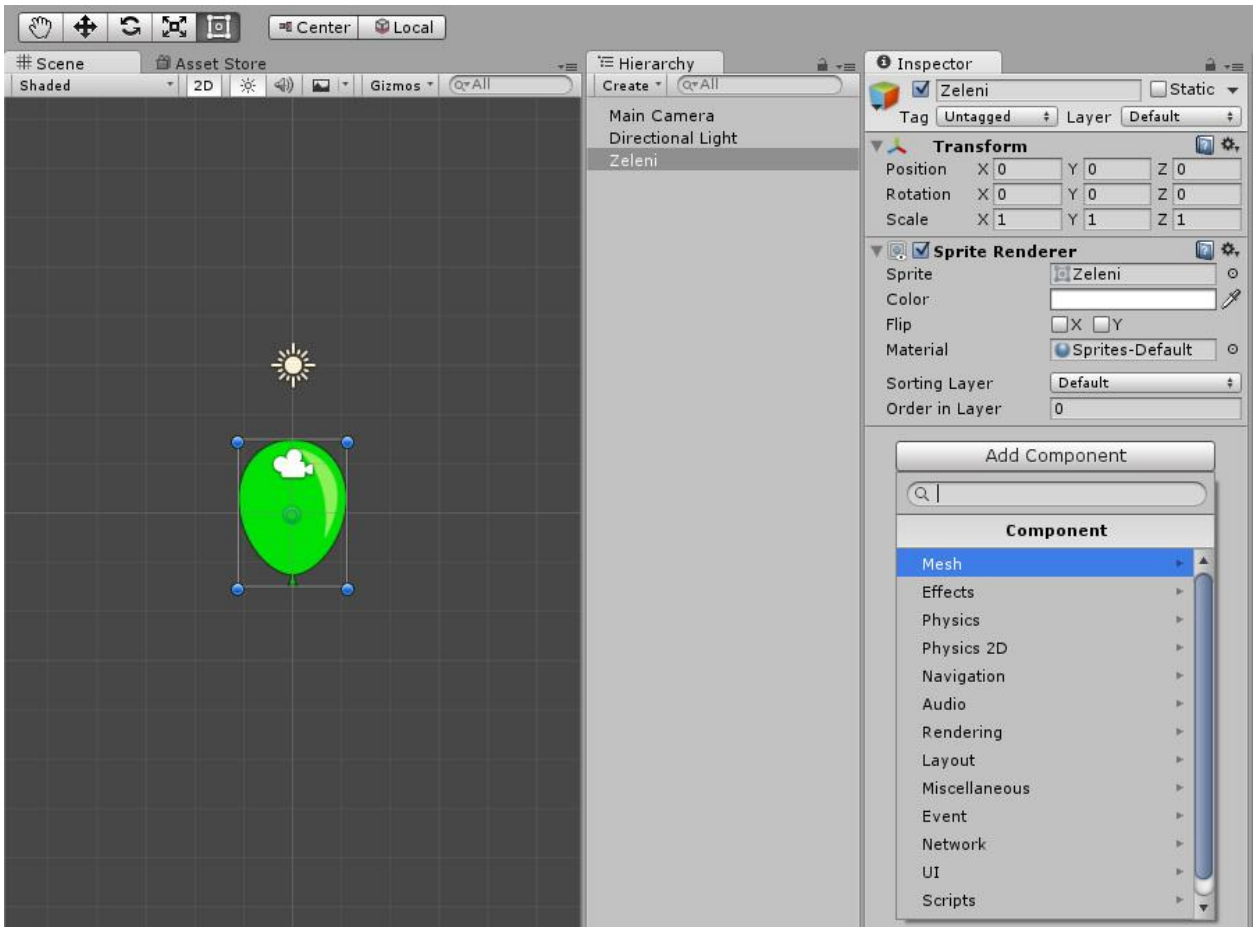
Kada se kreira novi projekt, odabere koja vrsta igre će se raditi, 2D ili 3D, prvo što se mora napraviti je uvesti sve *assets* koji će se koristiti u ovoj igri. *Assets* koji će se koristiti za ovu igru su PNG slike balona, oblaka, slike za pozadinu, pozadinska glazba, zvučni efekti, vizualni efekti, skripte te drugo.



Sl.3.1. Prikaz Assetsa u hierarchy panelu

Na slici 3.1. uvezeni *assets* se vide u *project* panelu, a u *hierarchy* i *scene* panelima se vidi, kada se kreira novi projekt da su u sceni samo dva objekta, *Main Camera* i *Directional Light*. Svi drugi objekti za igru trebaju se napraviti u Unityu ili ubaciti iz *assets*. S obzirom da je ovo 2D igra za

objekte neće trebati 3D modeli pa se 2D objekt može napraviti i iz PNG slike. Za grafike objekata sam koristio slike koje sam pronašao na internetu a dodatno sam ih obradio i prilagodio u programu Photoshop. Za kreiranje objekata iz PNG slika koje su uvezene dovoljno je samo tu sliku iz *assets* prevući u *hierarchy* ili *scene* panel. Novokreirani objekt trenutno nema nikakva svojstva i funkcije kao što su efekti, fizička svojstva, da bude izvor zvuka, skripte i drugo, a može ih se dodati dodavanjem komponenti i skripti na taj objekt.



Sl.3.2. Dodavanje komponenti na kreirani objekt

Dodavanjem komponenti objektima daju im se razne funkcije kao što su kretnja, reagiranje na korisnikove naredbe ili na neke događaje. Na slici 3.2. se u *inspector* panelu vidi opcija *Add Component*, na koju kada se klikne razlistaju se razne komponente koje se mogu dodati na objekt. Sve komponente dodane na objekt bit će prikazane u *inspector* panelu. U ovoj igri baloni bi se trebali kretati prema gore te bi se trebali probušiti kada dođu u kontakt s iglom koja je na vrhu ekrana. Za ta svojstva objektu balona se treba dodati *Physics 2D* komponentu te izabrati *Rigidbody*

2D i *Edge Collider 2D* svojstva. *Rigidbody 2D* dopušta objektu da se kreće na fizički uvjerljiv način dodavanjem sila iz skripti koje ćemo staviti na objekt, a *Edge Collider 2D* omogućuje objektu kontakt s drugim objektima koji na sebi imaju *Edge Collider*, te da detektiraju taj kontakt. Jedna od najvažnijih stvari kod kreiranja igre su skripte i skriptiranje. Unity podržava tri različita programska jezika, UnityScript, C# i Boo, a za ovu igru sve skripte sam pisao u C# jeziku. Kao zadani editor za skripte Unity koristi MonoDevelop. Prije početka programiranja potrebno je u svakoj .cs datoteci definirati bazu podataka sa sljedećim linijama koda,

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

te ukoliko skripta služi za definiranje UI (User interface) elemenata ili audia potrebno je dodati baze podataka kao što su,

```
using UnityEngine.UI;
using UnityEngine.Audio;
```

Sljedeće što u skripti treba napraviti jest da se objekt balon kreće prema gore,

```
public class BrzinaKretanja : MonoBehaviour {

    Vector2 Brzina;

    public static float BrzinaGore;

    void Start()
    {
        Brzina = new Vector2(0,BrzinaGore*Time.deltaTime);
    }

    void Update()
    {
        GetComponent<Rigidbody2D> ().velocity = Brzina;
    }

    void OnCollisionEnter2D (Collision2D col)
    {
        if (col.gameObject.name == "Needle") {

            Destroy (gameObject);

        }
    }
}
```

Definiran je vektor *Brzina* koji predstavlja 2D poziciju (x, y) objekta. Definirana je i javna varijabla *BrzinaGore*, ta varijabla će biti brzina kojom će se baloni kretati prema gore. Brzina neće

uvijek biti jednaka kako se ne bi svi baloni kretali jednakom brzinom nego različitom i varijabla je javna tako da joj se može pristupiti i iz druge skripte. Svaka skripta ima *Start()* i *Update()* funkciju. *Start()* funkcija se poziva prije nego što krene prvi okvir (*frame*) igre, a *Update()* funkcija se poziva svaki okvir. U *Start()* funkciji se postavlja da je y vrijednost vektora *Brzina* jednaka vrijednosti *BrzinaGore* puta *Time.deltaTime.* Vrijednost varijable *BrzinaGore* se uzima iz druge skripte a kada se ta vrijednost množi s *Time.deltaTime* u osnovi se želi taj objekt pomjeriti u odnosu na sekunde a ne u odnosu na okvire. Kada se objekt pomjera u odnosu na okvire on zna zapinjati i zastajkivati zato što brzina promjene okvira nije konstantna, pa ako se objekt pomjera u odnosu na sekunde kretnja bude puno fluidnija. U *Update()* funkciji s *GetComponent se* dohvaća vektor *.velocity* s komponente *Rigidbody2D* te postavlja da njena vrijednost bude jednaka vrijednosti vektora *Brzina*. Ispod funkcije *Update()* još će se napraviti jedna funkcija koja će registrirati dodir te uništiti objekt balon kada dođe u dodir s objektom igle.

Kada se ova skripta spremi i postavi na objekt balon on bi se sada trebao kretati prema gore sve dok se ne zaustavi igra ili dok se ne probuši te je s time završeno kreiranje objekta balona. Za ovu igru će biti potreban velik broj balona koji će se stvarati s dna ekrana te se kretati prema gore pa će se od ovog objekta balon morati napraviti *prefab.* U Unityu *prefab* se koristi kada je u sceni potrebno puno istih objekata s istim svojstvima kao što su metci, prepreke ili u ovom slučaju baloni. Za napraviti *prefab* iz objekta potrebno je samo prevući taj objekt iz *hierarchy* panela u *project* panel među *assets* foldere. Kada je od objekta balon kreiran *prefab* potrebno je napraviti skriptu koja će stvarati neograničen broj balona, primjer takve skripte je,

```
public class GeneriranjeBalona : MonoBehaviour {
        public GameObject BalonPrefab;
        float Vrijeme;
        public static int BrBalona;

        void Start ()
        {
            BrBalona = 0;
            StartCoroutine(Generiranje());
        }
```

U ovom dijelu koda deklarira se *GameObject* koji će se konstantno stvarati, deklarira se varijabla *Vrijeme* koja će označavati vrijeme između stvaranja dva balona te varijabla *BrBalona* koja označava ukupan broj stvorenih balona. U *Start()* funkciji varijabla *BrBalona* postavlja se na 0 te se poziva *coroutine. Coroutine* se može gledati kao funkcija koja se izvršava u intervalima, radi s posebnim *yield* izjavama koje pauziraju izvršavanje na određeno vrijeme.

```
IEnumerator Generiranje()
{
    while(true)
    {
        yield return new WaitForSeconds(Vrijeme);
        if (BrBalona < 20)

  {
            Invoke ("NapraviBalon", 1);
            Vrijeme = Random.Range (0.8f, 3f);
            BrBalona += 1;
            BrzinaKretanja.BrzinaGore = Random.Range(115,120);
        }
        if (BrBalona>=270f)
        {
            Invoke ("NapraviBalon", 1);
            Vrijeme = Random.Range (0.4f, 0.8f);
            BrBalona += 1;
            BrzinaKretanja.BrzinaGore = Random.Range(190,210);
        }
    }
}
```

IEnumerator je povratni tip za *coroutine* i ova funkcija će se uvijek izvršavati. U funkciji postoji više *if* izjava, u primjeru su navedene samo prva i zadnja *if* izjava. U prvoj će se pozivati funkcija *NapraviBalon* sve dok je varijabla *BrBalona* manja od 20, te će se varijabla *Vrijeme* nasumično postaviti na vrijednost između 0.8 i 3. Varijablom *Vrijeme* se određuje koliki će razmak biti između stvaranja dva balona. U svakoj *if* izjavi se postavlja i brzina kretanja svakog balona, to je javna varijabla iz prve skripte koja je postavljena na objekt balona. Što je veći broj balona u svakoj *if* izjavi će vrijeme između stvaranja balona biti manje i brzina kretanja balona biti veća.

```
    void NapraviBalon()
    {
         Instantiate(BalonPrefab, Pozicija(), Quaternion.identity) ;
    }

   Vector2 Pozicija()
    {
        float x,y;
        x = Random.Range(-2.5f,2.5f);
        y = Random.Range(-8,-6);
        return new Vector2(x,y);
    }

}
```

Primjer funkcije za stvaranje balona koja se poziva u svakoj if izjavi. Funkcija *Instantiate* stvara kopije originalnog objekta te traži tri parametra, objekt koji želimo kopirati, njegovu poziciju i njegovu rotaciju. Prvi parametar je *BalonPrefab*, objekt koji je deklariran na početku skripte, drugi parametar je pozicija, to će biti vektor za koji je postavljeno da mu koordinate x i y budu nasumične vrijednosti između dvije vrijednosti kako se baloni ne bi uvijek stvarali na istom mjestu. Kada je skripta gotova potrebno ju je staviti na bilo koji objekt koji će biti stalan u toj sceni kako bi se skripta mogla stalno izvršavati.

Kada su kreirani baloni koji se konstantno stvaraju te kreću prema gore sljedeći korak je da se na vrhu ekrana ti baloni probuše. To će se raditi na dva načina, prvi je s iglom koja je na vrhu ekrana i s kojom upravlja igrač, a drugi je s objektom koji će biti iznad ekrana, neće biti vidljiv i bušit će sve balone koji se ne probuše iglom. Svaki balon koji bude probušen iglom zbrajat će se u bodove, a svaki balon koji ne bude probušen iglom nego se propusti bit će probušen s objektom iznad ekrana te će se računati kao jedan izgubljen život. Objekt igle će se kreirati kao i objekt balon, željena PNG slika će se prevući iz *assets* foldera u *hierarchy* ili *scene* panel. Objektu igle će se također dodati svojstva *Rigidbody 2D* i *Edge Collider 2D*, ali će mu se dodati i svojstvo *Audio Source*. *Audio Source* komponenta će omogućiti objektu da bude izvor zvuka koji se postavi, zvuk se može emitirati od početka scene, samo na nekim događajima ili kada igrač zatraži, a emitiranjem zvuka se može upravljati i kroz skripte. Objekt igle će emitirati zvuk kada god probuši balon, zato će se na komponentu *Audio Source* staviti .wav datoteku zvuka bušenja balona. Objekt igle će se pomjerati samo po x osi u lijevo i desno naginjanjem uređaja, tako da se mora napraviti skripta koja će upravljati kretnjom objekta. Skripta koja će se staviti na objekt igle će također upravljati i emitiranjem zvuka kad god se balon probuši te s registriranjem bodova za svaki probušeni balon.

```
public class collision : MonoBehaviour {

public AudioClip BalloonPop;
AudioSource audio;

    void Start()
    {
        audio = GetComponent<AudioSource>();
    }

    void Update()
    {
        float Xos= Input.acceleration.x*8*Time.deltaTime;
        transform.Translate (Xos, 0, 0);

        Vector2 clamped = transform.position;
```

```
        clamped.x=Mathf.Clamp (transform.position.x, -2.75f, 2.75f);
        transform.position = clamped;
    }
```

U *Update()* funkciji se registrira svaki pomak uređaja te se ta registrirana brzina množi s osam kako bi se objekt pomjerao brže i fluidnije. S *Mathf.Clamp* se osigurava da se objekt može kretati samo unutar -2,75 i 2,75 vrijednosti na x osi, odnosno da se ne može kretati izvan ekrana uređaja.

```
void OnCollisionEnter2D (Collision2D col)
    {
        if (col.gameObject.name == "BBalon(Clone)") {
            ScoreManager.Score += 1;
            audio.PlayOneShot (BalloonPop, 1f);
     }
```

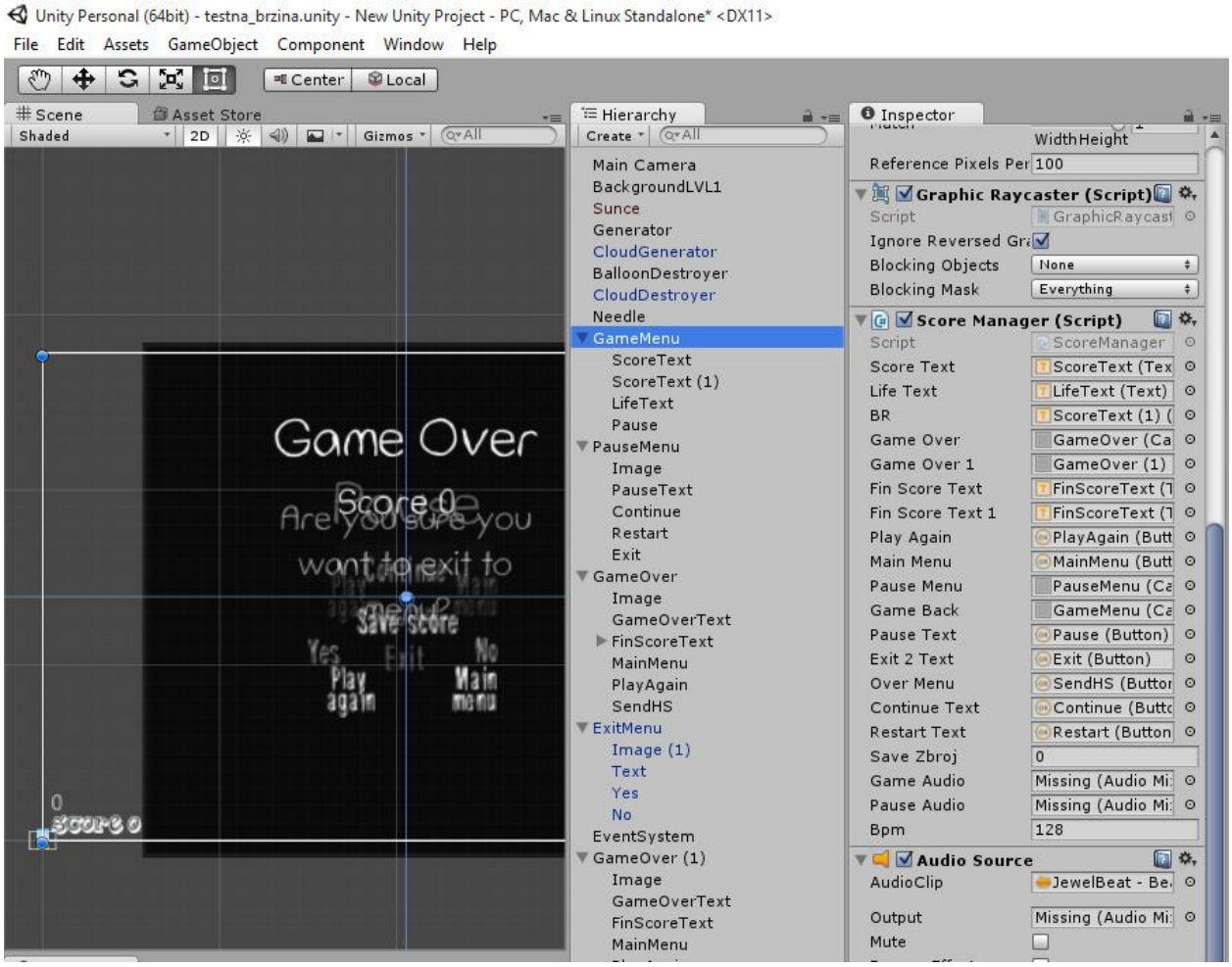
Funkcija *OnCollisionEnter2D()* registrira dodire objekta igle i objekta balona, te na svaki dodir zbraja bodove i pušta zvuk bušenja balona. U igri postoje pet vrsta balona, obični baloni koji se pojavljuju najčešće i vrijede jedan bod, baloni koji vrijede dva boda, pet bodova, deset bodova i baloni koji daju jedan život kada se probuše. Za svaku vrstu balona postoji po jedna *if* izjava, u primjeru je samo *if* izjava za balone koji daju jedan bod.

Sljedeći korak je napraviti objekt iznad ekrana koji će bušiti sve balone koji se ne uspiju probušiti s iglom. Objekt se neće vidjeti u igri pa je dovoljno napraviti pravokutnik i razvući ga da bude preko cijele širine ekrana. Treba mu staviti iste komponente kao i na objekt balon i igla, a u skripti koja će se staviti na njega dovoljno je da ima samo jednu funkciju koja će registrirati dodire.

```
void OnCollisionEnter2D (Collision2D coll)
    {
        if(coll.gameObject.name == "BBalon(Clone)"||
           coll.gameObject.name == "BCrni(Clone)"||
           coll.gameObject.name == "B5(Clone)"||
           coll.gameObject.name == "B10(Clone)"||
           coll.gameObject.name == "BZivot(Clone)")
        {
            ScoreManager.life -= 1;
            Destroy(coll.gameObject);
            Handheld.Vibrate ();
        }
    }
}
```

Za svaki registrirani dodir između objekta i balona oduzet će se jedan život, uništit će se objekt balona te će uređaj kratko vibrirati kako bi igraču pružio povratnu informaciju.

U ovoj sceni je preostalo napraviti elemente korisničkog sučelja kao što su pauza, opcije za izlazak iz igre, ulazak u glavni izbornik, za ponavljanje igre, elementi koji će pokazivati broj bodova i broj preostalih života, postaviti pozadinsku glazbu te animacije u igri. Korisničko sučelje se radi na način da u *hierarchy* panelu dodamo *Canvas*. *Canvas* je područje gdje bi svi elementi korisničkog sučelja trebali biti. Svi elementi korisničkog sučelja moraju biti djeca bilo kojeg *Canvasa.* Elementi korisničkog sučelja mogu biti gumbi, *toggle*, *slider*, *scrollbar*, tekst i drugi. Elementi se mogu potpuno prilagođavati prema željama korisnika, od veličine, boje fonta, animacija.



Sl.3.3. Prikaz Canvasa te skripte postavljene na njih

Na slici 3.3 se vidi da u ovoj sceni imaju pet zasebnih *Canvasa*, to su *GameMenu*, *PauseMenu*, *GameOver*, *ExitMenu* i *GameOver (1)*. Svaki *Canvas* ima svoju ulogu, *GameMenu* je aktivan dok traje igra i sadrži broj bodova, broj života te tipku pauza. *PauseMenu* je aktivan samo kada se pritisne tipka pauze te sadrži tekst i tipke za izlaz, nastavak ili ponavljanje igre. *GameOver* se

pojavljuje kada broj života dođe na nulu, a sadrži tekst, opciju da se upiše ime igrača te taj rezultat pošalje. Svi elementi korisničkog sučelja ne bi imali nikakvu funkciju bez skripti koje se stavljaju na njih pa se tako na slici 3.3 s desne strane vidi da je na *Canvas GameMenu* stavljena *Score Manager* skripta, ista skripta je stavljena i na ostale *Canvase.* Skripta upravlja sa svim elementima korisničkog sučelja te kada će se oni pojavljivati. Kako bi skripta mogla povezati i prepoznati elemente svaki interaktivni element se mora deklarirati što se na slici 3.3 i vidi.

```
public class ScoreManager : MonoBehaviour {

    public Text ScoreText;
    public Canvas GameOver;
    public Button MainMenu;
    public static int Score;
     public static int life;
     public int saveZbroj=0;

    void Awake ()
    {
        ScoreText = ScoreText.GetComponent <Text> ();
        GameOver = GameOver.GetComponent<Canvas> ();
        MainMenu = MainMenu.GetComponent<Button> ();

        PauseMenu.enabled = false;
        GameOver.enabled = false;
        GameOver1.enabled = false;
        GameBack.enabled = true;
        Score=0;
        life = 5;
    }
```

Prvo se deklariraju svi elementi s kojima će upravljati ova skripta, postoje primjeri za svaku vrstu, tekst, *canvas* i gumbe, a deklariraju se i varijable koje su zadužene za zbroj bodova i živote. *Awake()* funkcija se poziva samo jednom i to prije početka same igre, znači da se izvršava prije *Start()* i *Update()* funkcije. U *Awake()* funkciji dohvaćaju se komponente svakog elementa te se mora uključiti ili isključiti svaki *canvas* posebno, na počeku igre će svi biti aktivni osim ako ih se tu ne isključi. Ostavit će se uključen samo pozadinski *canvas* igre na kojemu se nalazi rezultat, broj života i tipka pauze.

```
    void Update()
    {
        ScoreText.text = "Score: " + Score;
        LifeText.text = "Life: " + life;

        if (life <= 0 )
        {
```

```
            Time.timeScale = 0;
            GameOverMenu ();
        }
```

U *Update()* funkciji je postavljeno da tekst ispisuje rezultat i broj života, tako da će se svaki okvir ažurirati te ispisivati rezultat i broj života. Također se provjerava da li je broj života manji ili jednak nuli, ako je, pauzira igru i izbacuje *GameOverMenu()* funkciju gdje igrač može vidjeti svoj konačan rezultat, spremiti rezultat, ponoviti igru ili izaći iz nje. U skripti također imaju i funkcije u kojima se određuje koji od *canvasa* će biti uključeni, da li je vrijeme u igri pauzirano i slično. Te funkcije se mogu pozvati kroz skriptu ili ih se može staviti na elemente korisničkog sučelja kao što su gumbi, pritiskom na gumb će se pozvati. Primjer takvih funkcija su,

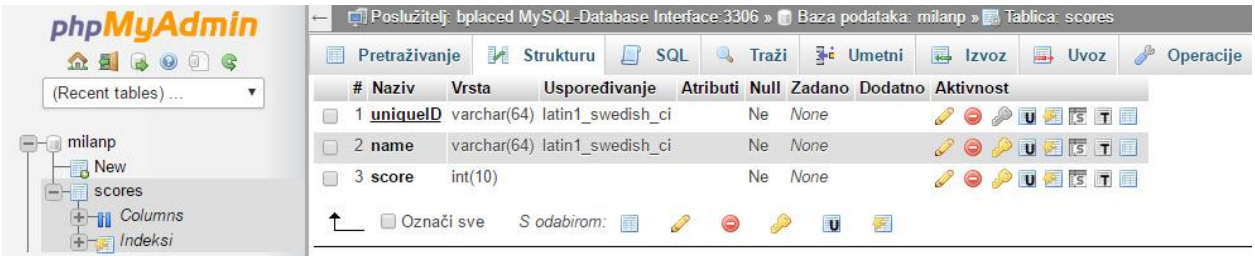
```
    public void PausePress ()
        {
            PauseMenu.enabled = true;
            PauseText.enabled = false;
            GameOver.enabled = false;
            GameOver1.enabled = false;
            Time.timeScale =0;
        }

    public void RestartPress ()
        {
            Time.timeScale =1;
            Application.LoadLevel (1);
        }
```

PausePress() funkcija se poziva kada se pritisne tipka za pauzu, vrijeme u igri će se zaustaviti te će biti aktivan samo *canvas PauseMenu.* Funkcija *RestartPress()* se poziva kada se pritisne tipka za ponavljanje igre iz izbornika pauze, s naredbom *Application.LoadLevel (1)* se učitava čitava scena ispočetka, a vrijeme se ponovno pokreće, postavlja na vrijednost 1, zato što je u izborniku pauze bilo zaustavljeno.

Nakon završetka korisničkog sučelja u ovoj sceni je ostalo samo još da se ubace estetski elementi. U pozadini, iza balona, na nebu će se kretati oblaci s lijeve na desnu stranu, princip je isti kao i kod kretnje balona, ali oblaka će biti manje i kretat će se sporije, nemaju nikakvu funkciju u igri nego su tu samo zbog estetike. Balonima su dodane i animacije, u trenutku kada se probuše bit će animacija male eksplozije te će također biti i broj koji će označavati bodovnu vrijednost tog probušenog balona. Animacije su vrlo bitna stvar kod kreiranja igara, igru čine zanimljivijom te ih se može koristiti za što god se poželi. Unity ima odličan sustav za animacije koji se zove *Particle system,* namijenjen je za animacije u igrama kao što su vatra, dim, kiša, eksplozije i slično. Ovaj

sustav nije težak za naučiti i vrlo jednostavno se mogu dobiti kompleksne animacije. Na *Asset store* postoji velik broj već gotovih animacije koje se mogu kupiti ili besplatno preuzeti te izmijeniti i prilagoditi za igru. Sa završetkom animacija gotova je i ova scena, ova scena će služiti kao sama igra te će se u nju ulaziti iz glavnog izbornika.

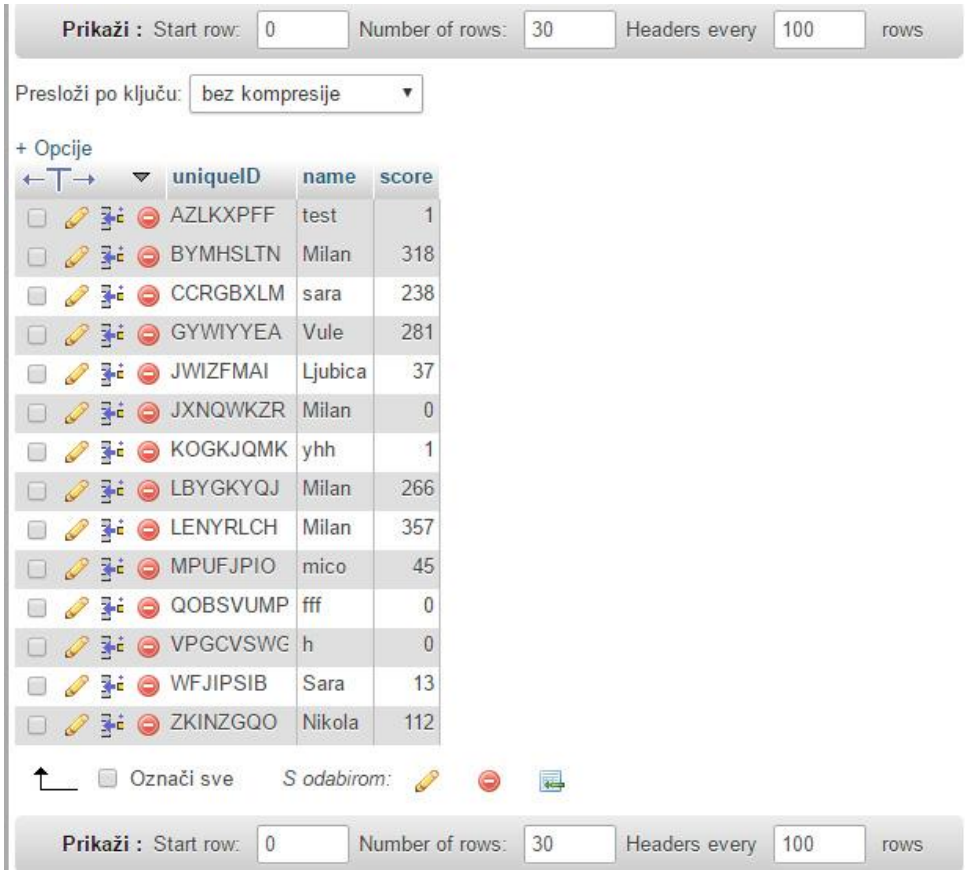
Glavni izbornik će biti zasebna scena tako da se prošla scena treba spremiti i kreirati nova, prazna scenu. Ova scena će služiti samo kao izbornik i biti će prva nakon što se uđe u igru. Sadržavati će samo pozadinsku sliku, naslov igre na vrhu ekrana, a u izborniku će biti opcije *Play*, *High Score* i *Exit*. Elementi korisničkog sučelja se rade na istom principu kao i u prošloj sceni, na isti način se piše i skripta koja će upravljati tim korisničkim sučeljem. Pritiskom na tipku *Play* pozvat će se funkcija iz skripte koja će samo s naredbom *Application.LoadLevel (1)* učitati scenu 1, gdje se nalazi sama igra. Pritiskom na tipku *Exit* pojavit će se mali izbornik koji će pitati da li ste sigurni da želite izaći, pritiskom na tipku *Yes*, pozvat će se funkcija iz skripte koja će s naredbom *Application.Quit ()* izaći iz čitave igre, a pritiskom na tipku *No* vraća u glavni izbornik. Opcija *High Score* otvara 20 najboljih upisanih rezultata, poredanih od najboljeg prema najgorem. Rezultati se ispisuju iz online baze podataka, tako da su prikazani upisani rezultati s bilo kojeg uređaja, a u slučaju da uređaj nema pristup internetu lista se neće prikazati, odnosno biti će prazna. Za ovu igru nije trebala kompliciran baza podataka, trebala je jednostavna baza podataka sa samo tri vrijednosti koje će se upisivati i iščitavati, *uniqueID*, *name* i *score*. Baza podataka je napravljena na www.bplaced.net, a za uređivanje i upravljanje bazom korištena je aplikacija phpMyAdmin.



Sl. 3.4 Tablice i struktura u bazi podataka

Kada se igra završi, izaći će *GameOverMenu* u kojemu će biti mogućnost upisivanja imena i odabira opcije *Save Score*, pritiskom na tu tipku poziva se funkcija koja će upisati u bazu podataka ime koje je igrač upisao, konačan rezultat koji je imao te će se generirati nasumičnih osam znakova koja će se upisati pod *uniqueID*. *UniqueID* osigurava da svaki zapis bude jedinstven i da se neće

upisivati jedan preko drugoga, *uniqueID* će uvijek biti različit dok vrijednosti *name* i *score* mogu biti iste.



Sl. 3.5 Primjer upisa u bazi podataka

Na slici 3.5 se vide primjeri upisanih vrijednosti, svi *uniqueID* su različiti dok neke *name* i *score* vrijednosti iste. Osnovu skripte za upravljanje i povezivanje igre s bazom podataka pronašao sam na Unity forumima te ju prilagodio za ovaj projekt. Dodavanjem baze podataka igra je sada završena, u potpunosti je funkcionalna i ima sve elemente koje bi trebala imati.

4. REZULTAT

U ovom poglavlju bit će prikazani rezultati rada iz prethodnih poglavlja, preko slika te opisa tih slika prikazat će se sve funkcionalnosti gotove igre.

Sl. 4.1 Početni izbornik

Na slici 4.1 se vidi izbornik koji se pojavi odmah nakon ulaska u igru. Na vrhu ekrana je naslov igre a ispod su tri opcije. S pritiskom na *Play* odmah se ulazi u sljedeću scenu, samu igru. Pritiskom na *High Score* ulazi se u listu 20 najboljih rezultata, a pritiskom na tipku *Exit* pojavit će se mali izbornik koji će pitati da se potvrdi izlazak iz igre, s dodatnim opcijama *Yes* i *No*.

Sl. 4.2 Prikaz najboljih rezultata

Kada se uđe u opciju *High Score* s početnog izbornika prikazat će se 20 najboljih upisanih rezultata. Rezultati će biti poredani od najboljeg prema najgorem, a svakim upisom novog rezultata koji ulazi u najboljih 20 miče se prethodni najgori. Lista je udaljena tako da će prikazivati rezultate upisane s bilo kojeg uređaja. Prikaz će se osvježiti svaki puta kada se uđe u listu.

Sl. 4.3 Opcije za izlaz

Kada se u glavnom izborniku pritisne na opciju *Exit* pojavit će se opcija, kao što se vidi na slici 4.3 lijevo, koja će pitati da se potvrdi izlazak iz igre ili da se vrati u glavni izbornik. Ista ta opcija pojavit će se kada se iz početnog izbornika na uređaju pritisne tipka *back*, a kada se iz same igre na uređaju pritisne tipka *back* pojavit će se opcija, kao što se vidi na slici 4.3 desno, koja će pitati da se potvrdi povratak u glavni izbornik ili nastavak igre.

Sl. 4.4 Izgled igre i opcije pauza

Pritiskom na opciju *Play* u početnom izborniku ulazi se u drugu scenu, samu igru. Na slici 4.4 lijevo se vidi izgled igre te korisničkog sučelja u igri. S donje lijeve strane se nalazi rezultat, s donje desne strane se nalazi broj života i tipka za pauzu. Na početku igre baloni se ne kreću jako brzo i nema ih toliko puno tako da ih nije teško probušiti, ali što se duže igra budu sve brži i sve ih je više tako da jedna igra traje samo nekoliko minuta. Na desnoj strani slike 4.4 se vidi izgled pauze, postoje 3 mogućnosti: nastavak igre, ponovno pokretanje ili izlazak u glavni izbornik.

Sl. 4.4 Game over

Game over se pojavljuje u trenutku kada igrač ne uspije probušiti pet balona, odnosno kada broj života dođe na nula. Igrač počinje igru s pet života ali tokom igre mogu se probušiti određeni baloni koji dodaju po jedan život. Kada je igra završena na ekranu piše konačan broj bodova što je igrač skupio, može upisati svoje ime i spremiti taj rezultat. Kada se klikne na *save score* rezultat se upisuje u udaljenu bazu podataka te igraču nestaje opcija za upis imena. Igrač bodove skuplja bušenjem balona, najčešći su baloni koji dolaze u raznim bojama i daju po jedan bod, crni baloni se kreću brže od ostalih i daju po dva poda, srebrni baloni ne dolaze često i daju po pet bodova, a zlatni baloni su najrjeđi i daju deset bodova. Kada igra završi i igrač spremi rezultat ima još mogućnost da ponovno igra ili da se vrati u početni izbornik.

5. ZAKLJUČAK

U ovom projektu napravljeni su svi željeni ciljevi, rezultat je funkcionalna i stabilna igra za Android operativni sustav. Igra je testirana na više uređaja sa različitim verzijama Android operativnog sustava i različitim veličinama ekrana. Ni na jednom uređaju igra nije imala problema u radu kao što su iznenadna zatvaranja igre ili da neki elementi igre ne stanu u ekran uređaja kod različitih omjera ekrana. Rad u programu Unity nije pretjerano kompliciran i vrlo brzo se mogu savladati osnove funkcionalnosti koje su potrebne za ovakvu igru. Snalaženje i učenje u Unityu je olakšano zahvaljujući korisničkom sučelju koje je prilagođeno korisnicima i brojnim objašnjenima i video uputstvima za svaku funkciju u Unityu. U projektu je najveći problem bio pronalazak odgovarajućih materijala za igru kao što je grafika, zvukovi i animacije, te bi za bolji i kvalitetniji dojam igre bilo bolje kada bi se ti materijali radili posebno za igru. Igra je prvobitno namijenjena djeci zbog svog zabavnog sadržaja, ali kako bi igra bila još privlačnija i zanimljivija može se dodatno doraditi dodavanjem novih opcija u mehaniku igre kao što su baloni koji se ne smiju probušiti, baloni koji kada se probuše usporavaju ili ubrzavaju ostale balone, uništavaju ostale balone i slično.

LITERATURA

[1] Wikipedia. Android (operating system). https://en.wikipedia.org/wiki/Android_(operating_system) [17.svibnja 2016.]

[2] Radić, D. " Informatička abeceda " Split-Hrvatska. http://www.informatika.buzdo.com/pojmovi/mobile-3.htm [17.svibnja 2016.]

[3] Unity3D. Unity manual. http://docs.unity3d.com/Manual/index.html [19.svibnja 2016.]

[4] Elektrotehnički fakultet Osijek. Objektno orijentirano programiranje. http://www.etfos.unios.hr/~lukic/oop/Auditorne_vježbe_5.pdf [11.lipnja 2016.]

SAŽETAK

Cilj ovog završnog rada je napraviti igru za pametne telefone s Android operativnim sustavom. Igra treba imati avatar kojim će se upravljati pomoću senzora pokreta i udaljenu bazu podataka za rang listu najboljih rezultata. Igra je zbog svog načina igranja i zabavnog sadržaja primarno namijenjena djeci. Za kreiranje igre koristio se alat Unity, a skripte koje su služile kao mehanika u pozadini igre pisane su u C# jeziku.

Ključne riječi: Igra, Android, Unity, 2D, C#

ABSTRACT

Android game for exercise reflexes

Goal of this final paper was to create a game for smartphones with Android operating system. Game has to have avatar which will be controlled with motion sensor and remote database for high score rankings. Game is primarily made for children because of the way it's played and its entertainig content. Unity game engine was used for creating game, and C# programming language was used for writing all game mechanics.

Key words: Game, Android, Unity, 2D, C#

ŽIVOTOPIS

Milan Pjevačević rođen je 21. listopada 1992. godine u Somboru u Republici Srbiji s prebivalištem u Belom Manastiru gdje je također pohađao osnovnu i srednju školu smjer tehničar za računalstvo u Prvoj srednjoj školi Beli Manastir. 2011 godine završava srednju školu i u Osijeku upisuje elektrotehnički fakultet.

Milan Pjevačević