Izrada platformske igre u tri dimenzije

Zvonimir, Crljenko

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Zvonimir Crljenko

IZRADA PLATFORMSKE IGRE U TRI DIMENZIJE

ZAVRŠNI RAD

Varaždin, 2019.

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Zvonimir Crljenko

Matični broj: 43504/14–I

Studij: Informacijski sustavi

IZRADA PLATFORMSKE IGRE U TRI DIMENZIJE

ZAVRŠNI RAD

Mentor:

prof. Danijel Radošević

Varaždin, rujan 2019.

Zvonimir Crljenko

Izjava o izvornosti

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

Autor/Autorica potvrdio/potvrdila prihvaćanjem odredbi u sustavu FOI-radovi

_____

Sažetak

Ovaj rad bavi se izradom platformske igre u 3 dimenzije. U uvodnom djelu ukratko je opisan vremenski pojam industrije video igara, njena povijest ukratko uz najbitnije predstavnike svakog razdoblja, koji su svojim naprednim mogućnostima napravili iskorak. Za potrebe izrade praktičnog dijela rada odabrao sam alat Unity kojeg ću detaljnije objasniti. Opisane su osnovne značajke i komponente alata, kako ih koristiti za potrebe platformerske igre i kako optimizirati rad u sučelju korištenjem određenih prečaca i navika. Opisane su i osnovne komponente platformske videoigre, iako je pojam vrlo širok, uz odabrane željene funkcionalnosti. U zaključku je dan kritički osvrt na alat i izradu igra u alatu Unity te općenito izradu videoigara u 2019. godini.

Ključne riječi: 3D; videoigra; igra; konzola; razvoj videoigre; platformer; Unity; C#; gaming; industrija videoigara.

Sadržaj

1. Uvod

U ovome radu obradio sam temu izrade trodimenzionalne platformer igre. Na odabir sam imao više alata odnosno razvojnih okruženja: Unity 3D, Unreal Engine, libGDX, Urho3D, CryEngine itd. Već prilikom odabira teme sam imao zamišljenu dilemu između dva najkonkurentnija okruženja: Unity 3D i Unreal Engine. Konačno sam odabrao okruženje Unity 3D, iz više razloga koje ću pokušati objaniti u nastavku.

Unity 3D (u nastavku: Unity) je okruženje za razvoj trodimenzionalnih igara, koji koristi C# kao primarni jezik za pisanje skripti i upravljanje komponentama, odnosno dijelovima scene unutar igre. S druge strane stoji Unreal Engine, koji kao primarni jezik koristi C++. Iako kažu da je Unreal jednostavniji za početnike, odabir Unity-a za ovu temu bio je nekako logičan: Unity je izuzetno voljen od strane zajednice igrača i razvojnih programera *indie* (skraćeno od samostalan (eng. *independent*)) igara. Obzirom da iza ovog rada ne stoji tim programera, brzina i efikasnost sučelja, kao i mogućnosti jednostavne implementacije kompliciranijih funkcija bile su glavna motivacija za odabir alata. Također, te mogućnosti su izuzetno dobro dokumentirane i pojašnjene, kako u samoj izvornoj dokumentaciji, tako i od strane zajednice na internetu. Na mnogim mjestima riješavaju se problemi zajedničkim komentiranjem pojedinih specifičnosti s kojima se pojedinac susretne, a ponajviše na vlastitim forum stranicama alata Unity.

2. Metode i tehnike rada

Za izradu ovog završnog rada nisu korištene nikakve poznate metode i tehnike. Rezultat je plod mašte, baziran isključivo na otkriću različitih mogućnosti, bilo iz dokumentacije, čistog korištenja sučelja ili pukom idejom, koja bi uz pregled atributa i metoda pojedinih klasa često dovela do rješenja. Glavna inspiracija za izradu igre bila je druga igra – *Super Mario 3D World*. Iako igru nikada osobno nisam igrao, Super Mario je svima poznata igra u dvije dimenzije, no tri dimenzije su pokazale koliko je razvoj platformera otišao naprijed u pozitivnom smislu. Igra nam jednostavno pokazuje koliko je snalaženje u trodimenzionalnom prostoru kompliciranije, možda čak i neprilagođeno za pravu platformsku videoigru, no da je cilj idalje ostvariv – zabava krajnjeg korisnika.

Izuzev samog razvojnog okruženja Unity, korišteni su pomoćni alati za obradu slike, kao što su *Paint.NET* i *GIMP*, a okušao sam se i u izradi trodimenzionalnih objekata koristeći alat *Blender* za te svrhe. Za pisanje skripti koristio sam *Microsoft Visual Studio 2019*.

3. O gaming industriji

U ovome dijelu rada opisati ću povijest i današnji doseg gaming industrije, koje su posljedice i uzroci uspjeha ove industrije, kao i moj pogled na budućnost ove specifične industrije.

3.1. Povijest

Rana povijest gaming industrije nije bila pretjerano uzbudljiva, i realno jedini primjeri vrijedni spomena iz doba prije prvih konzola (1972.) su *Bertie the Brain*, igra križić-kružić s računalom, predstavljena 1950. godine i *Spacewar!*, koji je izašao 1962. Drugospomenuti je po meni signifikantniji, obzirom da je nastao u vrijeme kada su mainframe računala postala dostupna profesorima i studentima na fakultetima, te su tako isti mogli razvijati igre koje bi se igrale putem terminala koji ima pristup mainframe-u. Dakako, u ovom razdoblju vrlo je bitno i spomenuti godinu 1964., kada je razvijen programski jezik BASIC koji je dakako olakšao razvoj ove industrije, obzirom da je bio programski jezik izuzetno jednostavan za savladavanje, naravno, uzimajući konekst toga doba (očekivanja i tadašnja ponuda).

3.1.1. Prve konzole

Prva konzola dostupna za kućnu uporabu bila je *Magnavox Oddysee*. [6] Ideja za kućnom konzolom nastala je još 1966., kada su Ralph Baer, Bill Harrisson i Bill Rusch odlučili razviti nekolicinu prototipova. U nekoliko godina, nastalo je 7 prototipova, a posljednji je prikazan brojnim proizvođačima. Posljednji prototip bio je poznatiji pod nazivom Smeđa kutija (eng. *Brown Box*), a *Magnavox* je odlučio započeti proizvodnju u siječnju 1971. godine. U rujnu 1972. godine, konzola je postala dostupna u maloprodajnim trgovinama.

Do svojeg prestanka proizvodnje, 1975. godine, prodalo se ukupno 350.000 primjeraka konzole. [6] Njen izuzetan uspjeh tvrtki je omogućio izdavanje čitave serije *Oddysee* konzola, koje su doduše bile limitirane na jednu ili više igara koje su integrirane u sami hardver uređaja. Također, 1978. godine, javnosti je predstavljen nasljednik, poznatiji pod nazivom *Magnavox Oddysee*^{2}. Konzola je imala više naziva, a europski naziv bio je *Philips Videopac G7000*.

Jedan od 28 igara dostupnih na originalnoj verziji konzole bila je ping pong igrica, vrlo slična *Atari*-jevoj uspješnici, što je dodatno potaklo uspjeh konzole. Patenti koje je Ralph Baer položio za sustav i njegove igrice, kroz idućih 20 godina donijele su tvrtki izuzetne

prihode, smatra se negdje oko 100 milijuna dolara. Dolazak ove konzole na tržište označio je završetak rane povijesti video igara.

3.1.2. **Atari VCS – Atari 2600**

Atari 2600 u svojim se početcima nazivao Atari *Video Computer* System odnosno skraćeno VCS. Kasnije, nakon 1982., poprimio je naziv 2600. U prodaji je bila od 1977., pa sve do 1992. godine (raspod od 15 godina!!). Prvotne verzije dolazile su u paketu s igrom *Combat*, a kasnije *Pac-Man,* no svaka kutija uključivala je par kontrolera (2 komada). Ova konzola bila je prvotna koja je koristila hardver temeljen na mikroprocesorima, i igrama koje su dolazile na ROM kasetama (eng. *cartridge*). Ovaj način distribucije i razvoja dozvolio je velikom broju razvojnih inženjera kreiranje raznovrsnih i naprednijih igara. Kaseta je bila izuzetno limitirana veličinom, tako je svaka igra mogla maksimalno zauzimati 2 KiB (1 kibibyte – 1024 byte-a). Iz današnje perspektive, veličina tih igrica je toliko zanemariva, da jedna snimka zaslona (eng. *screenshot*) zauzima više memorije nego čitave igre toga doba. Govoreći o kapacitetu, valja također naglasiti da u početku igre nisu zauzimale svu memoriju, već manji dio. Kasnije, razvojem drugih sustava i metoda, igre su zauzimale i 4 puta više memorije nego prvotne igre za Atari 2600.

Slika 1. Atari VCS[1]

Najveći uspjeh, doduše, doživjela je verzija *Space Invaders*-a iz 1980. godine, koja je dovela do uspjeha brojnih tvrtki, kako softverskih, tako i hardverskih, koje su preživjele sve do današnjega dana. Tako od hardverskih valja navesti recimo *Mattel*, američkog proizvođača igračaka, koji je u to vrijeme proizvodio konzole, ili recimo *Activision*, još uvijek popularnog studija za razvoj igara.

[1] - preuzeto s: https://commons.wikimedia.org/wiki/File:Atari-2600-Heavy-Sixer-FL.png

3.1.3. **Krah industrije video igara 1983.**

Kao i svaka industrija, i ova je imala svoje uspone i padove. Jedan od važnijih padova dogodio se 1983. godine, a poznat je i pod nazivom *Atari shock*, i većinom je bio osjetan u Sjedinjenim Američkim Državama. No, krah nije imao isključivo negativne posljedice. Veliki broj izdavača se odlučio povući, što je dovelo do još gore situacije, jer trgovine koje nisu mogle prodavati igre, nisu mogle vratiti robu koja nije prodana izdavačima, jer oni više nisu postojali.

Glavni razlog kraha bio je pojava osobnih računala. Godine 1979., Atari je predstavio svoja računala 400 i 800, *IBM* je predstavio *IBM 5150 PC*, a ta računala, iako građena na hardveru konzola, imala su puno veću memoriju i jače procesore, kako opće, tako i namjenske, odnosno grafičke i zvučne. Sukladno tome, igre koje su pokretala takva računala bile su naprednije i kvalitetnije nego konzolne igre.

Također, kao što je navedeno prethodno, *Activision*, tvrtka koja je nastala od nezadovoljnih zaposlenika *Atari*-ja, koji im nije dozvoljavao vlastitu promociju na izrađenim igrama, kao ni zaradu temeljenu na uspjehu pojedinog naslova, bila je prva *third-party* razvojna tvrtka (u značenju da razvoj videoigara namjenjenih toj konzoli nije isključivo od strane razvojnih programera matične tvrtke – proizvođača konzole). Sve nije išlo glatko, dapače. *Atari*, nezadovoljan što bivši zaposlenici zarađuju razvojem igara za njihov sustav, odlučili su priložiti tužbu sudu. Međutim, tužba nije imala osnovu, te je ishod bio dogovoran – *Activision* je pristao plaćati *Atari*-ju naknadu za razvoj na njihovom sustavu, a *Atari* je morao pristati legitimizirati model *third-party* razvoja. Brojne tvrtke, uvidjevši rast i razvoj *Activision*-a, krenule su u slične pohode. Međutim, uz brojne programere koji su tek ušli u industriju, i nepoznate opće algoritme ostvarivanja određenih komponenti igre, tvrtke su tražile brojna rješenja, pa čak i neka koja se danas smatraju ilegalnim. Mnoge tvrtke otkupljivale su već etablirane programere iz drugih tvrtki ili čak podvrgavali tuđe igre obrnutom inženjeringu (eng. *reverse engineering*), u smislu „dekompajliranja“ krajnjeg rezultata, odnosno pretvaranja rezultata u programski kod.

Ralph Bier, za to vrijeme je rekao da je diferencijacija najveći razlog kraha: „U pravo vrijeme za predstavu, biznis počinje zaranjati - zašto? Opća panika u industriji, male tvrtke gladuju, Midway Mfd proizvodi samo 50 uređaja tjedno, Atari se "bori", rečeno mi je... Najbolja pretpostavka uzroka jest: Svi kopiraju jedni druge u igrama - Atarijev dizajn nije dopuštao kreativno inženjerstvo, a javnost je odjednom postala sita s istom stvari, 28 puta drukčije zapakiranoj! Pouka: Nitko ne može tvrditi, ali najbolja pretpostavka jest da će se IGRE PRODAVATI NAVELIKO, AKO I SAMO AKO su različite, izazivajuće i omogućuju dvoboj između igrača, pune akcije i buke, a ne igre koje su jednostavne za naučiti.“ [1].

3.1.4. **Nintendo Entertainment System - NES**

Sve to dovelo je do brojnih skeptičnih komentara i razgovora, koji su za posljedicu imali krah koji se odvijao sve do 1985. Te je godine na globalnom tržištu (u Japanu je bio dostupan već 1983. pod nazivom *Famicom* (skraćeno od *Family Computer* – obiteljsko računalo)) postao dostupan *Nintendo Entertainment System*, poznatiji po svojoj skraćenici – *NES*. Utoliko, to možemo i navesti jednim od velikih razloga zašto se NES upravo i zove tako. Marketinški tim se pribojavao da će potrošači izbjegavati proizvod ukoliko zadrže naziv „konzola“.

Također, Nintendo je bio svjestan trenutnih uvjeta, i morali su poraditi na nagovaranju preprodavača kako bi njihovi proizvodi dospjeli na police istih. Međutim, srećom, bili su izuzetno ponosni i svjesni kvaliteta svojeg proizvoda, toliko da su čak napravili i robota zvanog *R.O.B.* kako bi preprodavači igračaka shvatili važnost ovog sustava. Ova je konzola također koristila vrstu kasete, međutim ono što ju je razlikovalo od prethodnika je to što se kaseta umetala na način vrlo sličan tada popularnim *VHS* sustavima i uređajima.

3.1.5. **PlayStation**

Razvoj ove konzole pete generacije imao je veliki utjecaj na budućnost industrije. Bila je to konzola koja je iskoristila doba prelaska na CD-ROM pogone, i 3D mnogokutnu grafiku. Konzola je postala dostupna 1994. godine u Japanu, a godinu dana kasnije i u Sjevernoj Americi i Europi, te potrajao na tržištu sve do 2006. godine. Njen proizvođač – *Sony*, u tu avanturu upustio se još 1986. godine zajedno s *Nintendo*-om. Nintendo je htio, obzirom da su već imali razvijen sustav s disketnim pogonom umjesto kaseta, razviti sustav za CD-ROM pogon za svoj *Super Nintendo Entertainment System*, poznatiji po skraćenici *SNES*. Obratili su se Sony-u, s projektom nazvanim „Play Station“ ili „SNES-CD“.[3] Sony je prvotno htio razviti vlastitu konzolu, koja bi bila kompatibilna sa SNES kasetama i novorazvijenim CD sustavom koji bi se također koristio i za SNES-CD. [4] Sony je imao jako veliku kontrolu, iako je Nintendo u to vrijeme bio svojevrni „lider“ u gaming industriji. Proizvod je trebao biti predstavljen 1991., međutim odgođen je, jer je direktor Nintendo-a Hiroshi Yamauchi pročitao originalni ugovor i shvatio da Sony ima kompletnu kontrolu nad naslovima razvijenim za SNES na optičkom mediju.

4. Razvoj 3D platformera

U ovome dijelu pokušat ću objasniti izradu platformer igre u tri dimenzije u alatu Unity. Prvo ću opisati sam alat, njegove specifičnosti i osnovne načine rada i razvoja igara. Platformska igra

4.1. O alatu Unity

Samo snalaženje u sustavu je prilično jednostavno, a kako bi lakše pratili korake u ovome radu, poželjno je prilagoditi prikaz pritiskom na padajući izbornik „Window“ iz gornjeg dijela zaslona, potom odabrati „Layouts“ u kojem se nalaze predlošci izgleda razvojnog okruženja. Odabirom opcije „2 by 3“ podjelit ćemo izgled našeg radnog prozora na 5 osnovnih sekcija, prikazanih na slici 2:

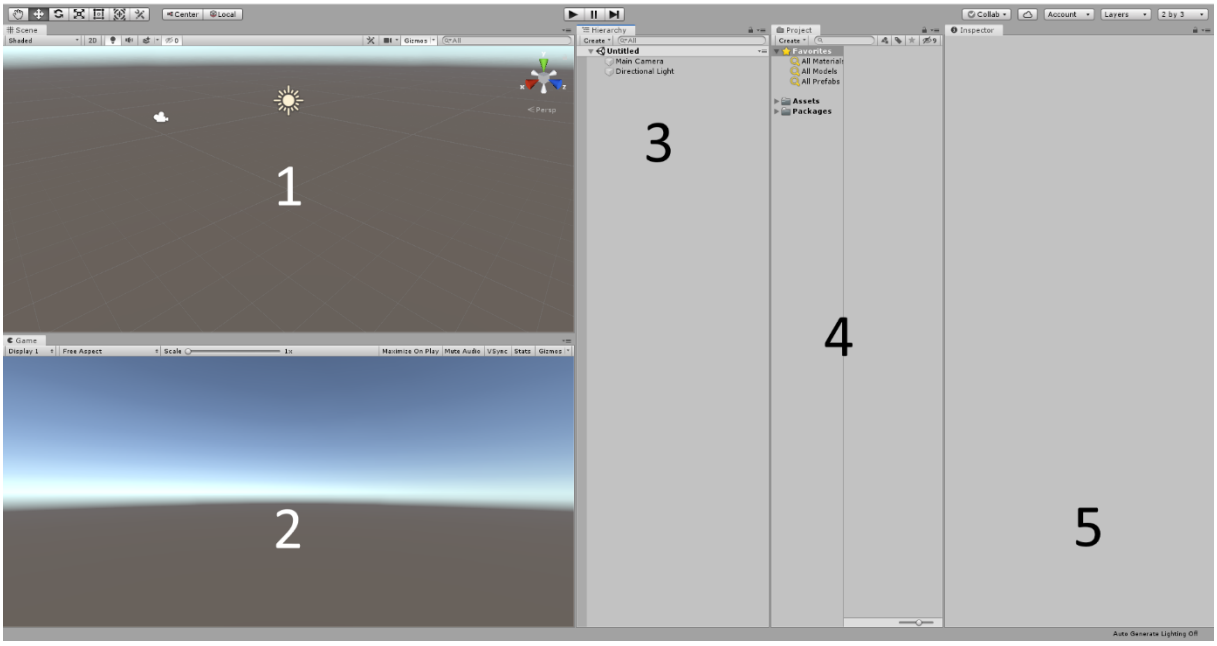
1. Pregled scene (eng. *Scene view*)
2. Pregled igre (eng. *Game view*);
3. Hijerarhijski prozor (eng. *Hierarchy*);
4. Projektni prozor (eng. *Project*);
5. Inspekcijski prozor (eng. *Inspector*);

Svaki od prozora ima svoju osnovnu namjenu upravljivosti alatom i igrom koju izrađujemo. Pregled scene nam dopušta trodimenzionalni prikaz scene koju uređujemo, postavljanje i razmještanje objekata ili modela po istoj, a ujedno može služiti i kao selektor objekta čije atribute želimo mijenjati (pozicija, rotacija, veličina (eng. *transform*), varijable iz skripti itd.), kao i hijerarhijski prozor koji te objekte prikazuje tekstualno.

Pregled igre nam pruža uvid kako finalni „proizvod“ našeg rada izgleda, i tu se pokreće scena iz perspektive igrača prilikom pritiska na tipku „Play“ u gornjem dijelu zaslona.

Projektni prozor je ustvari preglednik trenutnih mapa i datoteka koje smo uvezli u alat kako bismo ih koristili u igri. Tu se nalaze modeli, skripte, materijali, teksture i sve ostalo potrebno za igru. Obzirom da izrada igre vrlo brzo može stvoriti veliki broj potrebnih komponenti, mišljenja sam da je dobra organizacija resursa najbolji način izbjegavanja naknadnog bespotrebnog posla otkrivanja koja skripta čemu služi. Prilikom kreiranja nove scene, u projektnoj mapi će se nalaziti jedna nasljeđena mapa naziva „Scenes“, u koju ćemo spremati sve kreirane scene. Svoju sam projektnu mapu podjelio na nekoliko podmapa. Tako sam u podmapu „Scripts“ stavljao sve skripte napisane prilikom izrade. U mapu „Player“ sam postavio sve objekte potrebne za realizaciju objekta igrača i slično.

Inspekcijski prozor nam daje uvid u stanje varijabli i atributa vezanih za odabrani objekt (iz hijerarhijskog popisa ili sa pregleda scene). Tu mijenjamo vrijednosti koje se primjenjuju na željeni objekt. Iz ovog prozora također možemo kreirati oznake (eng. *tags*) kojima možemo opisati naš objekt, i po toj oznaci ga prepoznavati u kodu. Tako je recimo igrač u mojem slučaju označen zadanom oznakom „Player“, dok sam za tlo (podlogu) kreirao oznaku „Ground“ i za novčiće oznaku „Coins“.



Slika 2. "2 by 3" pogled u Unity-u [autorski rad]

Primjer korištenja ovih oznaka je recimo skripta *CoinPickup.cs*. U njoj se, ukoliko je novčić doživio sudar (eng. *collision*) s objektom koji ima oznaku „Player“, poziva metoda *PickedUp(int value)* iz skripte *GameManager.cs* kako bismo nadodali vrijednost pokupljenih novčića trenutnom stanju:

```
private void OnTriggerEnter(Collider other)
{
	if (other.tag == "Player")
	{
		FindObjectOfType<GameManager>().PickedUp(value);
		Destroy(gameObject);
	}
}
```

Osim razloga navedenih u uvodu rada, još jedan utjecajan čimbenik za odabir ovog alata bio je podrška više operativnih sustava. Iako je razlika više psihološka, jer oba okruženja dopuštaju razvoj za sve najkorištenije platforme i sustave, postoje marginalni dijelovi industrije koji pokušavaju razviti igre za specifične platforme sa specifičnim funkcionalnostima, na primjer *Nintendo Switch*, *Oculus Rift*, *Facebook Gameroom* ili *Windows Mixed Reality*.

Unity podržava razvoj igara za sljedeće platforme: *Windows, Mac, Linux, PS4, PSVITA, Xbox One, iOS, Android, Windows Phone, WebGL, Nintendo 3DS, Oculus Rift, Google Cardboard Android & iOS, Steam VR PC & Mac, PlayStation VR, Gear VR, Windows Mixed Reality, Google Daydream, Android TV, Samsung SMART TV, tvOS, Nintendo Switch, Fire OS, Facebook Gameroom, Apple ARKit, Google ARCore* i *Vuforia*.

Kako bismo jednostavnije objasnili potrebne komponente igre, potrebno je prvo definiriati osnovne pojmove u alatu Unity. Osnovna odrednica svake igre je scena. Scena se sastoji od objekata, modela ili komponenti sustava (izbornika i slično). Svaka scena u alatu Unity, kada se kreira, automatski nasljeđuje 2 komponente: glavnu kameru (eng. *Main camera*) i usmjereno svjetlo (eng. *Directional light*). Glavna kamera je osnovna komponenta, bez koje ne bismo mogli imati „pogled“ u igru. Ona međutim nema nikakvo dinamično svojstvo sama po sebi. Na početku ima lokaciju, koja se ne mijenja ovisno o lokaciji, poziciji, rotaciji igrača, zbog čega svakako moramo napraviti skriptu za automatsko upravljanje njome. Ona nam pruža prikaz iz kreirane scene, koji će biti prikazan korisniku kada pokrene scenu. Usmjereno svjetlo s druge strane pruža osvjetljenje, najsličnije objašnjivo kao Sunce u stvarnom životu. Također se može podesiti kao: reflektor (sija u jako uskom prostoru, kao onaj korišten na koncertima, eng. *spot light*), točka (oko mjesta postavljanja, iz točke se u svim smjerovima širi svjetlost) ili kao površinsko svjetlo.

4.2. Osnovni pojmovi i specifičnosti

U ovome dijelu rada objasnit ću osnovne pojmove za lakše razumijenje procesa izrade igre po pojedinim komponentama. Jedna od osnovnih pretpostavki praktične organizacije komponenti u alatu jest kreiranje mapa unutar projektne mape. Za potrebe rada sam projekt podjelio na mape: Sounds, Scripts, Scenes, Prefabs, Player, Pickupables, Materials, Images, Fonts, Animations. Prijevode pojedinih neću objašnjavati, samo ću napomenuti da mapa Prefabs sadrži „gotov“ objekt, povezan sa svime potrebnime i može se sačuvati za ponovnu upotrebu. Unutra sam stavio gotove objekte novčića, kako nebih morao ponovno odvlačiti, namještati 3D model, povezivati na njega skriptu i ostale korake potrebne da kreiramo novčić.

Scena je u alatu Unity laički rečeno prostor na koji smještamo objekte, ali u osnovnom značenju. Ona je prazan prostor, bez terena i podloge, te svakako treba kreirati ravninu koja bi poslužila kao tlo. Na tlo potom možemo stavljati razne objekte.

Skripta u Unity-u je programski kod koji dinamično utječe na rad sustava, mijenjajući vrijednosti varijabli i objekata igre. Skripta se povezuje na objekt, te „djeluje“ u njegovoj instanci. Kada god postoji instanca tog objekta, postoji i instanca skripte koja se izvršava.

Skripte mogu imati razne namjene, od upravljanja igračem, rotacije i upravljanja kamerom, osvježavanju rezultata, brojača (eng. *timer*) pa sve do višedretvenog rada. Bitno je samo napomenuti kako Unity podržava višedretveni rad, međutim dijelovi samog Unity sustava mogu se isključivo mijenjati iz glavne dretve. Tako da potreba za višedretvenim radom, barem u našem primjeru, neće biti od velike koristi. Ono što sam svakako koristio u radu jesu korutine (eng. *coroutine*). To su funkcije koje imaju mogućnost zaustavljanja izvođenja dok imaju kontrolu i povratak kontrole alatu Unity nakon završetka izvođenja kako bi se vratio točno na mjesto gdje je stao. Korutine imaju osnovno obilježje prepuštanja (eng. *yielding*) kontrole, pa tako povratak kontrole alatu Unity možemo učiniti pisanjem naredbe „yield return null;“ Kada se izvrši taj dio koda, korutina „prepušta“ kontrolu Unityu. Bitno je napomenuti kako su korutine funkcije tipa *IEnumerator*, i mogu primati arugmente. Osim običnog prepuštanja kontrole, poziv „yield“ može služiti i za prethodno „spavanje“. Kako bi naredba iz *System.Threading.Thread.Sleep()* zaustavila izvođenje svih skripti, nije je moguće koristit u Unityu. Zato, pozivom korutine i u njenoj naredbi „prepuštanja“, možemo nadodati čekanje naredbom „yield return new WaitForSeconds(int seconds);“, koja će odspavati za navedeno vrijeme i vratiti kontrolu Unityu. Također je korisnija za optimizaciju koda ukoliko se nešto mora odvijati postupno. Recimo, u igri sam načinio da se igračevo „zdravlje“ postupno umanjuje prilikom dodira neprijatelja ili tla koje je nedostupno (lava), da 2 sekunde nakon umanjenja zdravlja ga više ne možemo umanjivati, i ukoliko je pokupio sve novčiće u trenutnoj sceni, poziva se funkcija Blink() koja čini da tekst prikupljenih novčića „titra“:

```
IEnumerator Blink()
    {
        for(int i=0; i < 8; i++)
        {
            currentCoinText.enabled = !currentCoinText.enabled;
            yield return new WaitForSeconds((float)0.25);
        }
    }
    IEnumerator Invincibility()
    {
        Invincible = true;
        yield return new WaitForSeconds(2);
        Invincible = false;
    }
    IEnumerator ApplyHurt(float damage)
    {
        float endHealth = currentHealthSlider.value - damage;
        if (endHealth <= 0) currentHealthSlider.value = 0;
        while (currentHealthSlider.value > endHealth)
        {
            currentHealthSlider.value -= (float)damage *
Time.deltaTime;
            yield return new WaitForSeconds((float)Time.deltaTime);
        }
    }
```

4.3. Izrada scena i skripti

Prilikom kreiranja skripte u alatu Unity, automatski se dodaju 2 funcije – *Start()* i *Update()*. Funkcija *Start()* pokreće se jednom, prije svih *Update*-a, dok se *Update()* pali svakom sličicom (eng. *frame*). Iznad obje funkcije možemo napisati javne varijable, koje će potom biti dostupne za uređivanje iz razvojnog okruženja. Predložene funkcije moguće je izbrisati, a moguće je i dodati neke druge ugrađene funkcije poput *Awake()* koja se kao i *Start()* pokreće jednom tijekom životnog ciklusa instance, međutim pokreće se prilikom instanciranja objekta na kojem skripta „leži“, i pokrenut će se neovisno je li uključena ili isključena (eng. *enabled*). Postoji i funkcija *LateUpdate()*, koja se pokreće nakon svih *Update()* poziva i najkorisnija je za pomicanje kamere, kako bi to bio posljednji korak u kreiranju prikaza korisniku odnosno kako bi kamera izgledala prirodnije. *FixedUpdate()* je funkcija neovisna o broju sličica u sekundi, te se okida puno rijeđe od *Update()* funkcije. Najkorisnija je za izračune vezane uz fiziku objekata. Postoji velik broj ugrađenih funkcija dostupnih za korištenje, no druge su ovisne o objektima, animatoru i brojnim drugim komponentama koje potencijalno možemo imati u sceni. Sve funkcije su detaljno opisane u dokumentaciji, i imaju naveden primjer uz primjer programskog koda.

Izrada scena u alatu Unity je vrlo jednostavna: u pregledu scene možemo vršiti interakciju s objektom na sceni (pritisnuti na njega, dobiti informacije u inspekcijskom prozoru), rotirati ga, mijenjati veličinu ovisno o odabranom alatu iz alatne trake koja se nalazi iznad pregleda scene.

Pretpostavku dobre prakse organizacije istovrsnih komponenti koristio sam i pri izradi scena, međutim ovdje sam komponente organizirao po namjeni i vrsti. Tako sam u opširnijoj sceni koristio nadređene objekte: Terrain za sve objekte vezane uz podlogu i platformu podjeljen upravo na te podmape, Coins za sve novčiće, podjeljen na 3 podmape: 1, 2, 5 kuna, PlayerHUD gdje se nalaze canvasi za prikaz izbornika („game over“, pauza ili završetak nivoa) i Player gdje se nalazi model i animacije za objekt igrača.

4.3.1. Opis potrebnih komponenti

Prototip trodimenzionalne platformske igre izrađen za potrebe ovog rada sastoji se od 2 scene:

- Glavni izbornik;
- Level1;

Bitno je napomenuti da se prva scena sastoji isključivo od objekata korisničkog sučelja (eng. *UI*), odnosno gumba (eng. *button*) i slika, a svrha im je osnovna upravljivost igrom (odabir nivoa, izlaz iz igre). Druga scena je sami nivo igre i sadrži objekte igrača, kamere, terena itd.

Za potrebe ovog rada odabrao sam metode unosa koje bi odgovarale za više načina upravljanja (u smislu: koristeći kombinaciju tipkovnicu/miš, isključivo tipkovnicu, *joypad* i slično). Načinio sam sljedeće skripte:

- Timer.cs
- PlayerController.cs;
- PauseManager.cs
- NonGround.cs
- MusicPlayer.cs;
- MenuManager.cs
- GameOverManager.cs;
- GameManager.cs;
- FinishLevel.cs;
- CoinPickup.cs;
- CameraController.cs;
- AttackTrigger.cs;
- AnimationEndHandler.cs;

Ostale komponente koje su korištene u radu, a nisu spomenute ovdje, većinom su 3D modeli preuzeti iz Trgovine predložaka (eng. *Asset Store*), komponenti ugrađenoj u Unity kojoj možemo pristupati iz samog alata, te tamo možemo pronaći brojne 3D modele, teksture, gotove nivoe i sl. Modeli mogu biti besplatni ili se plaćaju, neki za potpuna prava, a neki samo za pravo korištenja. Svi modeli korišteni u ovoj igri su besplatni i dostupni na korištenje u nekomercijalne svrhe. O tome više u dijelu o izradi scena.

4.3.1.1. Izrada skripti u alatu Unity

Najjednostavniji način izrade skripte, u smislu izbjegavanja bespotrebnih koraka, jest na sljedeći način: pritisak na komponentu na koju želimo dodati skriptu lijevom tipkom miša na pregledu scene ili na hijerarhijskom prozoru, pritisak na gumb „Add Component“ – „New Script“, upišemo naziv te pritisnemo na gumb „Create and Add“. Na taj način smo dodali novu skriptu, povezali je s objektom, pravilno imenovali, otvorili u uređivaču i spremna je za uređivanje.

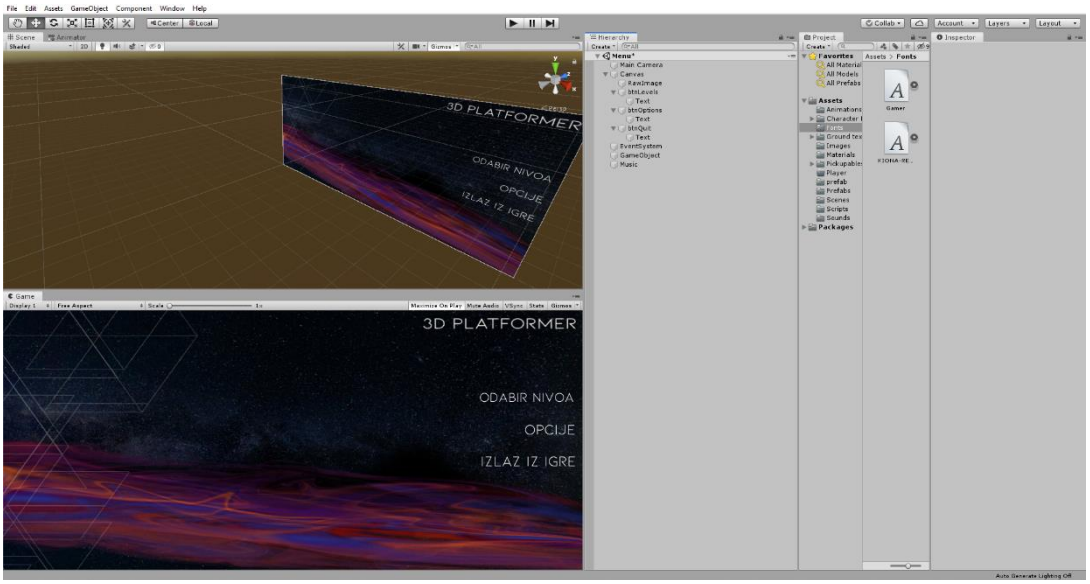
Druga, također često korištena metoda, radi praktičnosti i preglednosti, je odvajanje skripti u zasebnu mapu, najčešće naziva „Scripts“, desnim klikom na kreirani folder,

odabirom opcije „Create“ – „C# Script“. Nakon završetka uređivanja, skripta se jednostavno povuci - ispusti metodom (eng. *drag and drop*) iz projektnog preglednika iz mape „Scripts“, odvuče na objekt kojeg želimo da koristi tu skriptu. Time skripte držimo organiziranima na jednom mjestu i u svakome trenutku znamo gdje se skripta koja iziskuje određenu promjenu nalazi.

4.3.2. **Izrada scene Menu**

Ova scena je naša „početna stranica“, koja se korisniku prikazuje prilikom pokretanja igre. Na njoj se nalazi canvas koji sadrži 3 gumba, za početak igre, promjenu opcija i izlazak iz igre. Za izradu ove scene koristio sam predefinirane elemente korisničkog sučelja (eng. *UI*), koji su dostupni u alatu Unity pod „GameObjects“ – „UI“. Osim gumba, možemo dodati i razne prekidače, klizače, tekst, slike i sl.

Na slici 3. možemo vidjeti izgled menija u alatu: sve komponente korištene na sceni prikazane su u hijerarhijskom prozoru, u projektnom prozoru prikazane su sve trenutne komponente korištene dodane u projektni „build“, a trenutno na slici je otvorena mapa „Fonts“ u koju sam spremio 2 fonta: *Kiona Regular* i *Gamer*.



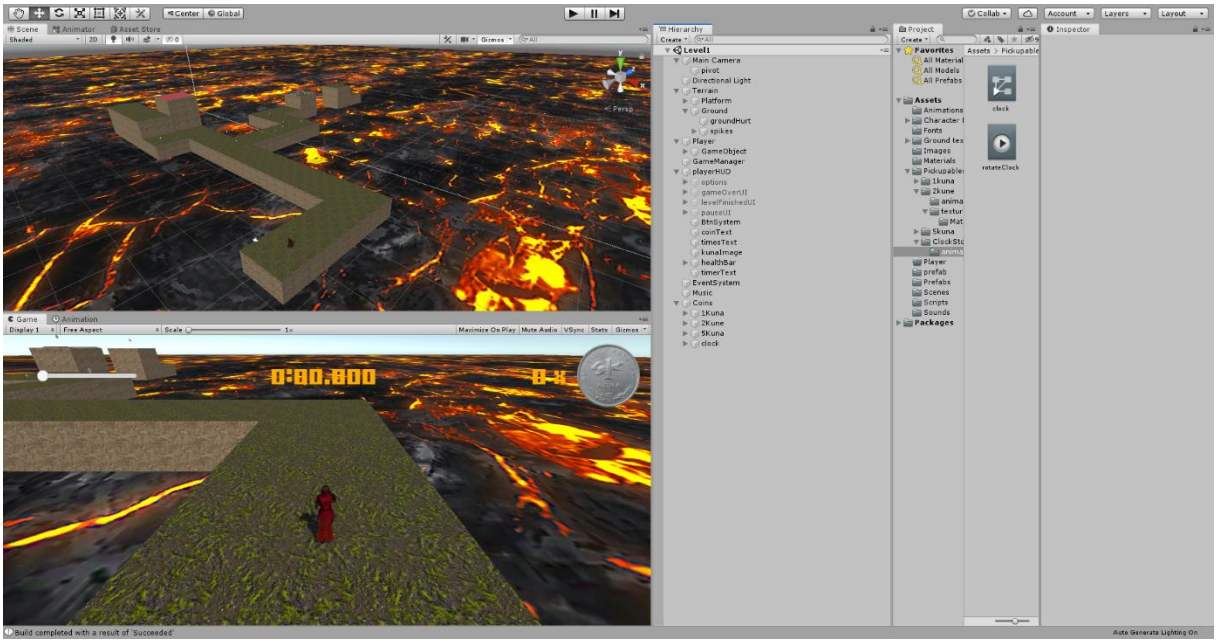
Slika 3. Izrada scene Menu [autorski rad]

4.3.3. **Izrada scene Level1**

Ova scena sadrži smisao igre i sve objekte korištene za realizaciju igre. Potreban nam je 3D model novčića i 3D model igrača. Za 3D model novčića, preuzeo sam gotov model, jedino sam promijenio teksturu, koju sam postavio na prikaz hrvatske valute – kune. Prvotna verzija bila je model *BitCoin* novčića. U igri se nalazi isti model s 3 različite teksture – 1, 2 i 5 kuna. Isto tako, vrijednost promjene ukupnog broja novčića podešena je ovisno o teksturi. Za 3D model igrača iskoristio sam model dostupan u *Asset Store*-u pod nazivom

Samurai. Uključuje 3D model samuraja i animacije napravljene u drugom alatu, zbog čega su bile potrebne preinake kako bi se iste iskoristile za Unity. O uporabi animacija u *Animator*-u bit će riječi kasnije.

Također, kako se radi o platformerskoj igri – potrebna nam je platforma. Odlučio sam se za najjednostavniji pristup izradi plaforme, od obične kocke na koju sam primjenio teksturu zemlje, i na vrh dodao ravninu koja ima teksturu trave. Napravio sam tri osnovne veličine platformi, a razlikuju se po visini. Tako imamo platformu veličine 5, 10 ili 15 jedinica veličine.



Slika 4. Izrada scene Level1 [autorski rad]

4.3.4. **Skripta Timer.cs**

Namjena ove skripte je jednostavna: vodi računa o trenutnom proteku vremena od početka igranja nivoa. Timer se zaustavlja u slučaju pauze ili završetka nivoa. Sastoji se od instance klase Text koja prikazuje trenutno vrijeme i jedne *float* varijable koja označa trenutno vrijeme. Sadrži jednu nasljeđenu metodu – *Update()* koja zbraja *Time.deltaTime* u varijablu:

```
void Update()
{
	currentTimer += Time.deltaTime;
	timerText.text = currentTimer.ToString();
}
```

4.3.5. **Skripta PlayerController.cs**

Ova skripta primarno je namjenjena upravljanju korisničkim odnosno igračevim unosima, kako bi model i objekt igrača reagirao na vanjske podražaje (korisnički unos).

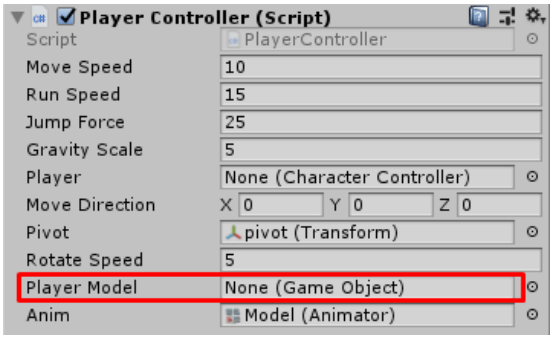
Temelji se na predefiniranoj komponenti naziva *CharacterController* odnosno upravljač lika. Sadrži 5 osnovnih varijabli tipa *float*:

Tablica 1. Prikaz varijabli skripte PlayerController.cs

Naziv	Vrsta	Opis
moveSpeed	float	Brzina kretanja objekta
runSpeed	float	Brzina trčanja igrača
jumpForce	float	Jačina skoka
gravityScale	float	Utjecaj gravitacije
rotateSpeed	float	Brzina rotacije objekta igrača

Smisao svake pojedine varijable je čitljiv iz tablice, međutim svakako ćemo spomenuti značenje: *moveSpeed* je varijabla kojom postavljamo brzinu kretanja korisnika (korisnički unos smjera množimo tom vrijednošću, kako bismo pomaknuli objekt u traženom smjeru po tlu, traženom brzinom), *jumpForce* je snaga odupiranja gravitaciji, odnosno vrijednost do koje ćemo pomicati objekt igrača u svijetu kako bismo simulirali njegov skok, množeći vrijednost s unosom (pritiskom razmaknice na tipkovnici), *gravityScale* je vrijednost kojom ćemo umanjivati visinu objekta igrača, odnosno čiju vrijednost ćemo množiti s vrijednošću same gravitacije kako bi objekt imao prividni utjecaj gravitacije, *rotateSpeed* je brzina kojom ćemo objektu igrača primjenjivati rotaciju prilikom promjene smjera kretnje i *runSpeed* je brzina trčanja, vrlo slična *moveSpeed* varijabli, no pokreće se samo kada igrač trči (kada je pritisnuta tipka *Shift* (i to lijevi)).

Skripta također sadrži i 4 složene varijable odnosno instance klasi, koje u kodu ili u alatu Unity moramo povezati na objekt čiju instancu tražimo. Tako ćemo u slučaju *moveDirection* instance, vektor graditi kroz kod ovisno o akcijama korisnika, *player* instancu ćemo dohvaćati u metodi *Start()* pomoću funkcije *GetComponent<CharacterController>*, obzirom da je instanca tog tipa, dok ćemo *playerModel* i *anim* instancu povezati povuci - ispusti (eng. *drag and drop*) metodom odvlaćenjem objekta „Model“ iz objekta „Player“ na mjesto prikaza korištene instance u skripti:



Slika 5. Povezivanje instance u skripti s instancom objekta [autorski rad]

Tablica 2. Prikaz instanci klasa skripte PlayerController.cs

Naziv	Vrsta	Opis
player	CharacterController	Komponenta za upravljanje kretnjom objekta igrača
moveDirection	Vector3	Vektor smjera kretnje igrača
pivot	Transform	Koordinate pivot točke u koju je usmjerena kamera
playerModel	GameObject	3D model igrača kojeg rotiramo
anim	Animator	Instanca animatora, kojoj želimo pristupati i mijenjati vrijednosti

Player, odnosno instanca klase CharacterController, nam daje mogućnost upravljanja igračem. Klasa CharacterController sadrži brojne korisne varijable, a jedna od glavnih koju ćemo koristiti za naše potrebe jest *.isGrounded*, boolean varijabla koja poprima vrijednost *true* samo ukoliko igrač dodiruje tlo. To će nam biti potrebno kako bismo znali je li korisnik trenutno na podu, u skoku odnosno hoćemo li dozvoliti skok i slično. Također, klasa sadrži metodu .*Move*() koja kao argument prima instancu klase *Vector3*, u našem slučaju proslijedit ćemo instancu te klase nazvanu *moveDirection*. Ta metoda pomiče igrača u smjeru navedenom u instanci koju smo proslijedili, što za sobom povlači činjenicu da prethodno pozivu funkcije, moramo okarakterizirati i zapisati smjer kretnje u instancu *moveDirection* koju prosljeđujemo. Instancom klase *Animator* dolazimo do atributa (varijabli) koje smo načinili u samom *Animator*-u te ih možemo čitati i pisati. Na početku provjeravamo izvodi li se animacija napada, i ukoliko je završena ili se ne izvodi postavljamo vrijednost atributa „isAttacking“ na neistinito (eng. *false*), a prilikom prikaza svake sličice ažurira se i vrijednost atributa *isGrounded* i *Speed* kako bi animator znao kreće li se igrač, je li prizemljen i slično. Skripta sadži 2 nasljeđene metode, Start() i Update():

```
    void Start()
    {
        player = GetComponent<CharacterController>();
    }
    void Update()
    {
        if (anim.GetCurrentAnimatorStateInfo(0).IsName("Attack") &&
anim.GetCurrentAnimatorStateInfo(0).normalizedTime > .99f)
        {
            anim.SetBool("isAttacking", false);
        }
        float lastY = moveDirection.y;
```

```
        moveDirection = (transform.forward * Input.GetAxis("Vertical")) +
(transform.right * Input.GetAxis("Horizontal"));
        if (Input.GetButton("Run") && !anim.GetBool("isAttacking"))
        {
            moveDirection = moveDirection.normalized * runSpeed;
            anim.SetBool("isRunning", true);
        }
        else
        {
            if (Input.GetMouseButtonDown(0) && Input.GetAxis("Vertical") >
0)
            {
                anim.SetBool("isAttacking", true);
            }
            moveDirection = moveDirection.normalized * moveSpeed;
            anim.SetBool("isRunning", false);
        }
        moveDirection.y = lastY;
        if (player.isGrounded)
        {
            moveDirection.y = 0f;
            if (Input.GetButtonDown("Jump")) moveDirection.y += jumpForce;
        }
        moveDirection.y += (Physics.gravity.y * gravityScale *
Time.deltaTime);
        player.Move(moveDirection * Time.deltaTime);

        if (Input.GetAxis("Horizontal") != 0 || Input.GetAxis("Vertical")
!= 0)
        {
            transform.rotation = Quaternion.Euler(0f,
pivot.rotation.eulerAngles.y, 0f);
            Quaternion newRotation = Quaternion.LookRotation(new
Vector3(moveDirection.x, 0f, moveDirection.z)).normalized;
            playerModel.transform.rotation =
Quaternion.Lerp(playerModel.transform.rotation, newRotation, rotateSpeed *
Time.deltaTime);
        }
        anim.SetBool("isGrounded", player.isGrounded);
        anim.SetFloat("Speed", Mathf.Abs(Input.GetAxis("Vertical")) +
Mathf.Abs(Input.GetAxis("Horizontal")));
}
```

4.3.6. **Skripta PauseManager.cs**

Skripta sadrži funkcije potrebne za korištenje pauze odnosno privremenog zaustavljanja igre. Sastoji se od 2 bool vrijednosti, 2 instance klase GameObject, 2 instance klase Slider, instance klase Toggle te 10 funkcija:

Tablica 3. Prikaz varijable i instanci klasa skripte PauseManager.cs

Naziv	Vrsta	Opis
isPaused	bool	Uključen/isključen mod pauze
isOptions	bool	Uključen/isključen mod opcija
pauseMenu	GameObject	Instanca klase GameObject,

		canvas koji prikazuje izbornik pauze
options	GameObject	Instanca klase GameObject, canvas koji prikazuje izbornik opcija
volume	Slider	Instanca klase Slider, klizača za glasnoću glazbe
mouseSensitivity	Slider	Instanca klase Slider, klizača za osjetljivost miša (brzina rotacije)
invertedView	Toggle	Obrnute kretnje potezom miša (miš prema dolje – pogled gore)

Funkcija Update() pali se prilikom prikazivanja svake sličice, te stoga u njoj provjeravamo pritisak tipke za pauzu – slovo P ili tipka Esc. Ukoliko je neka od tipki pritisnuta, provjeravamo je li igra pauzirana ili ne te ovisno o tome palimo Resume() ili Pause(). Pause() oslobađa pokazivač miša, kako bi se moglo upravljati izbornikom, postavlja canvas trenutno aktivnim, zaustavlja vrijeme i postavlja isPaused na vrijednost true. Resume() radi potpuno identičan zadatak, samo obrnuto (zaključava pokazivač, deaktivira canvas, pokreće vrijeme i postavlja isPaused na false). Funkcije ChangeVolume(), ChangeSensitivity() i InvertView() su funkcije namjenje radu kontroli opcija. Prilikom otvaranja menija korisniku se prikazuju klizači i prekidači kojima regulira postavke sustava igre (u primjeru sam izradio klizače za brzinu rotacije pokreta mišem odnosno osjetljivost pokazivača i glasnoću glazbe te prekidač za atribut *invertedView* u skripti *CameraController.cs* koji obrće smjer rotacije kamere prilikom pomicanja miša prema gore ili dolje. Funkcija Options() služi za otvaranje izbornika opcija, Back() služi za povratak iz opcija, Menu() za povratak u glavni izbornik, a Quit() za izlaz iz igre:

```
public void Update()
{
    if (Input.GetKeyDown(KeyCode.P))
    {
        if (!isPaused) Pause();
        else Resume();
    }
    if (Input.GetKeyDown(KeyCode.Escape))
    {
        if (isOptions) Back();
        else Resume();
```

```
        }
    }
    public void Resume()
    {
        Cursor.lockState = CursorLockMode.Locked;
        pauseMenu.SetActive(false);
        Time.timeScale = 1f;
        isPaused = false;
    }
    public void Pause()
    {
        Cursor.lockState = CursorLockMode.None;
        pauseMenu.SetActive(true);
        Time.timeScale = 0f;
        isPaused = true;
    }
    public void ChangeVolume()
    {
        FindObjectOfType<AudioSource>().volume = volume.value;
    }
    public void ChangeSensitivity()
    {
        FindObjectOfType<CameraController>().rotateSpeed =
mouseSensitivity.value;
    }
    public void InvertView()
    {
        FindObjectOfType<CameraController>().invertedView =
!FindObjectOfType<CameraController>().invertedView;
    }
    public void Options()
    {
        pauseMenu.SetActive(false);
        options.SetActive(true);
        isOptions = true;
        isPaused = true;
        volume.maxValue = 1;
        volume.value = FindObjectOfType<AudioSource>().volume;
        mouseSensitivity.maxValue = 5;
        mouseSensitivity.minValue = 2;
        mouseSensitivity.value =
FindObjectOfType<CameraController>().rotateSpeed;
        invertedView.isOn =
FindObjectOfType<CameraController>().invertedView;
    }
    public void Back()
    {
        try
        {
            isOptions = false;
            isPaused = true;
            options.SetActive(false);
            pauseMenu.SetActive(true);
        }
        catch
        {
            //prazan blok, nastavi
        }
    }
    public void Menu()
    {
```

```
    Time.timeScale = 1f;
    SceneManager.LoadSceneAsync("Menu");
}
public void Quit()
{
    Application.Quit();
}
```

4.3.7. **Skripta NonGround.cs**

Skripta primjenjena na tlo kako bi se igraču oduzimalo „zdravlje“ ukoliko ga direktno dodiruje (ukoliko upadne u lavu). Sastoji se od jedne varijable tipa float (*stepDamage*) i jedne funkcije *OnTriggerEnter(Collider other)*. Funkcija umanjuje „zdravlje“ igrača prilikom njegovog sudara s lavom, i resetira atribut *isTrigger* na tlu, kako bi se zdravlje ponovno moglo umanjiti ukoliko korisnik ostane u lavi i istekne 2 sekudne nedodirljivosti. Ponašanje tla prilikom postavljanja atributa *isTrigger* na istinitu vrijednost je takvo da igrač kroz okidač prolazi, odnosno tlo nije čvrsto i ne zadržava igrača – iz tog razloga unutar ravnine tla kreirana je još jedna ravnina, međutim njoj je atribut *isTrigger* postavljen na false, kako igrač nebi bio u mogućnosti ispasti iz granica igre:

```
private void OnTriggerEnter(Collider other)
{
      if (other.tag == "Player")
      {
            if (!FindObjectOfType<GameManager>().Invincible)
            {
                  FindObjectOfType<GameManager>().Hurt(stepDamage);
            }
            other.enabled = false;
            other.enabled = true;
      }
}
```

4.3.8. **Skripta MusicPlayer.cs**

Ova je skripta namjenjena reprodukciji glazbe odnosno zvučnog zapisa u igri. Specifičnost ove skripte je u tome što se zvučni zapis počinje reproducirati na sceni glavnog izbornika, a svakako nam je potreban i u drugim scenama, zbog čega smo iskoristili mogućnost alata Unity s funkcijom *DontDestroyOnLoad(GameObject object)* koja ne uništava instancu ni njenu djecu prilikom učitavanja druge scene. Skripta se sastoji od jednog statičkog svojstva – *Instance*, čiji *getter* je javan, a *setter* privatan, i jedne nasljeđene funkcije – *Awake()*, koja ne dozvoljava instanciranje ove skripte više od jednom (praktički *singleton* objekt):

```
void Awake()
{
    if (Instance != null && Instance != this)
    {
        Destroy(this.gameObject);
```

```
        }
        else
        {
            Instance = this;
        }
        DontDestroyOnLoad(this.gameObject);
    }
```

4.3.9. **Skripta MenuManager.cs**

Skripta služi za upravljanje glavnim izbornikom. Ima jednu instancu klase *GameObject* koja predstavlja canvas s opcijama koji je početno skriven do pritiska na gumb „Opcije“. Skripta je korištena isključivo u sceni „Meni“ i sadrži 1 instancu klase *GameObject* koja predstavlja canvas koji sadrži izbornik opcija i 3 funkcije: *PokreniIgru()*, *Options()* i *Quit()*. Prva funkcija nas vodi na scenu „Level1“, druga otvara opcije, a posljednja izlazi iz aplikacije (igre):

```
public void PokreniIgru()
{
    SceneManager.LoadScene("Level1");
}
public void Options()
{
    options.SetActive(true);
}
public void Quit()
{
    Application.Quit();
}
```

4.3.10. **Skripta GameOverManager.cs**

Ova skripta okida se kada igrač ostane bez „zdravlja“. Također se okida u slučaju da igrač stane na blok za završetak nivoa. U tom trenutku potrebno je obavijestiti korisnika da je igra završena, i ponuditi opcije povratka u izbornik, izlaza iz igre ili ponovnog igranja nivoa. Sastoji se od dvije instance klase *GameObject* koje su ustvari canvasi s prikazom statusa završenosti igre, koji je početno (eng. *default*) isključen i četiri funkcije za prethodno navedene namjene:

```
public void Restart()
{
    SceneManager.LoadScene("Level1");
    Time.timeScale = 1f;
}
public void Menu()
{
    SceneManager.LoadScene("Menu");
}
public void Quit()
{
    Application.Quit();
}
public void FinishGame(string time)
```

```
{
    gameFinished.text += time;
    gameFinishedOverlay.SetActive(true);
}
```

4.3.11. **Skripta GameManager.cs**

Glavna skripta za upravljanje igrom. Sadrži varijablu koja pohranjuje vrijednost svih novčića na nivou, vrijednost trenutno prikupljenih novčića, trenutno „zdravlje“ (eng. *health*), maksimalnu vrijednost našeg „zdravlja“, instancu klase Text, koja je ustvari dio stalnog canvasa, koji prikazuje stanje novičića prilikom igranja nivoa, instancu klase Slider koja je prikaz trenutnog „zdravlja“ te instancu klase Image koja je ustvari ispuna našeg klizača, koja mijenja boju ovisno o trenutnoj razini „zdravlja“:

Tablica 4. Prikaz varijabli i instanci klasa skripte GameManager.cs

Naziv	Vrsta	Opis
coinSum	int	Ukupna vrijednost novčića na nivou
currentCoins	int	Vrijednost prikupljenih novčića
currentHealth	float	Trenutno „zdravlje“
maxHealth	float	Maksimalna vrijednost „zdravlja“
currentCoinText	Text	Tekst s canvasa koji prikazuje stanje novčića
currentHealthSlider	Slider	Klizač s canvasa koji prikazuje stanje „zdavlja“
fill	Image	Ispuna klizača za prikaz „zdravlja“

Skripta sadži osam metoda i jedno svojstvo. Svojstvo ima osnovnu namjenu, javno je, no samo za čitanje, i prikazuje vrijednost atributa neoštetljivosti (eng *invincibility*), kako bismo znali kada igraču smijemo a kada ne smijemo oduzimati „zdravlje“:

```
public void Start()
{
    currentHealthSlider.maxValue = maxHealth;
    currentHealthSlider.value = maxHealth;
    currentHealth = maxHealth;
    GameObject[] coins = GameObject.FindGameObjectsWithTag("Coins");
    foreach (var item in coins)
    {
```

```
            coinSum += item.GetComponent<CoinPickup>().value;
        }
    }
    public void Update()
    {
        switch (currentHealthSlider.value)
        {
            case var i when i <= 0.6:
                fill.color = Color.red;
                break;
            case var i when i < (currentHealthSlider.maxValue * 76) /
100:            fill.color = Color.yellow;
                break;
        }
    }
    public void PickedUp(int value)
    {
        currentCoins += value;
        if (currentCoins == coinSum)
        {
            currentCoinText.text = "ALL";
            StartCoroutine(Blink());
        }
        else currentCoinText.text = currentCoins.ToString();
    }
    public void Hurt(float damage)
    {
        if(!Invincible)
        {
            currentHealth -= damage;
            StartCoroutine(ApplyHurt(damage));
            StartCoroutine(Invincibility());
            if (currentHealth <= 0)
            {
                Time.timeScale = 0f;
                Kill();
            }
        }
    }
    public void Kill()
    {
         Cursor.lockState = CursorLockMode.None;
         FindObjectOfType<GameOverManager>().gameOverOverlay.SetActive(t
rue);
        Time.timeScale = 0f;
    }
    public void Finish(string time)
    {
        Cursor.lockState = CursorLockMode.None;
        FindObjectOfType<PauseManager>().isPaused = true;
        FindObjectOfType<GameOverManager>().FinishGame(time);
        Time.timeScale = 0f;
    }
    IEnumerator Blink()
    {
        for(int i=0; i < 8; i++)
        {
            currentCoinText.enabled = !currentCoinText.enabled;
            yield return new WaitForSeconds((float)0.25);
        }
    }
```

```
IEnumerator Invincibility()
{
    Invincible = true;
    yield return new WaitForSeconds(2);
    Invincible = false;
}
IEnumerator ApplyHurt(float damage)
{
    float endHealth = currentHealthSlider.value - damage;
    if (endHealth <= 0) currentHealthSlider.value = 0;
    while (currentHealthSlider.value > endHealth)
    {
        currentHealthSlider.value -= (float)damage * Time.deltaTime;
        yield return new WaitForSeconds((float)Time.deltaTime);
    }
}
```

4.3.12. **Skripta FinishLevel.cs**

Ova skripta sadrži jednu funkciju, *OnTriggerEnter(Collider)* koja označava korisnički dodir finalne kocke i pokreće završetak igre:

```
private void OnTriggerEnter(Collider other)
{
    if (other.tag == "Player")
    {
       FindObjectOfType<GameManager>().Finish(FindObjectOfType<CameraC
 ontroller>().FloatTimeToString());
    }
}
```

4.3.13. **Skripta CoinPickup.cs**

Ova skripta sadrži funkcije koje se odvijaju prilikom „skupljanja“ novčića. Korektnije bi bilo, umjesto „skupljanja“, reći sudaranja – obzirom da je izvor našeg saznanja da je novčić skupljen, upravo sudar (eng. *collision*) našeg objekta igrača s objektom novčića.

Skripta se sastoji od 1 funkcije i 1 varijable tipa *int* koja označava vrijednost objekta s kojim smo se sudarili (broj za koji uvećavamo trenutno stanje novčića).

Nije potrebno imati referencu na instancu novčića, obzirom da ćemo skriptu primjeniti na novčić (*drag and drop* potezom). Tada i naredba *Destroy(gameObject);* iz skripte ima smisla: *gameObject* je nasljeđena varijabla, odnosi se na objekt u igri (sceni) na kojem se izvršava instanca skripte. Sličnu namjenu ima *transform* koji sadrži informacije o veličini, poziciji i rotaciji objekta na kojem se izvršava instanca skripte. Postoje i brojne druge nasljeđene varijable, kojima možemo saznati mnoge stvari o objektu koji trenutno izvršava skriptu. Detaljnije informacije o nasljeđenim varijablama moguće je pronaći u službenoj dokumentaciji.

Funkcija koju skripta sadrži naziva se *OnTriggerEnter* i kao argument prima instacu klase tipa *Collider* (prijevod: sudarač). Funkcija *OnTriggerEnter* je funkcija ugrađena u sustav

Unity-a i pokreće se kada *GameObject* doživi sudar s drugim *GameObject*-om, u našem slučaju igraču kojeg prepoznajemo po prethodno dodanoj oznaci (eng. *tag*) „Player“: [4]

```
private void OnTriggerEnter(Collider other)
{
    if (other.tag == "Player")
    {
      FindObjectOfType<GameManager>().pickedUp(value);
      Destroy(gameObject);
    }
}
```

4.3.14. **Skripta CameraController.cs**

U većini igara današnjice, ukoliko se radi o igrama u tri dimenzije, imamo i sustav upravljanja kamerom. Takav sustav pokušava se što više unaprijediti, kako je on osnova dobrog iskustva igre (eng. *gameplay*) [5]. Osnovna komponenta nad kojom ova skripta radi je objekt igrač, a specifično smo za ovu namjenu kreirali i pivot objekt, odnosno točku koju kamera prati. Pivot je ništa drugo nego objekt igrača, međutim sami pivot na početku nije dodjeljen igraču već kameri (unutar kamere kreirali smo prazni objekt i nazvali ga pivot). Smisao toga je da u drugim scenama ne moramo ponovno postavljati i kreirati pivote koji prate igrača.

Skripta se sastoji od 4 osnovne varijable, 3 tipa float i 1 tipa bool:

Tablica 5. Prikaz varijabli skripte CameraController.cs

Naziv	Vrsta	Opis
rotateSpeed	float	Brzina rotacije kamere
invertedView	bool	Obrnuto pomicanje kamere prema gore ili dolje
minViewAngle	float	Minimalni kut gledanja
maxViewAngle	float	Maksimalni kut gledanja
currentTimer	float	Trenutni brojač vremena

Kao što imena samih varijabli kažu, njihovo značenje je sljedeće; *rotateSpeed* je vrijednost koju množimo s trenutnom pozicijom pokazivača miša (eng. *mouse pointer*), kako bismo rotirali kameru u smjeru korisnikovog fizičkog pomaka miša, *invertedView*, ukoliko je postavljen na vrijednost *true*, pomiče kameru u okomitom smjeru, ali obrnuto (u značenju: korisnik pomiče miš prema dolje – za očekivati je pomicanje kamere prema dolje, međutim ako je vrijednost postavljena na *true*, kamera se pomiče prema gore (obrnuto)), *minViewAngle i maxViewAngle* su vrijednosti koje označavaju kut gledanja te limitiraju kameru na pomicanje u rasponu od *minViewAngle* do *maxViewAngle* u vidu okomitog

pomicanja kamere. Varijabla *currentTimer* je decimalni broj, trenutni zbroj svih *Time.deltaTime*-ova od početka izvođenja skripte (izuzevši pauze).

Skripta sadrži 4 složene varijable, odnosno instance klasi:

Tablica 6. Prikaz instanci klasa skripte CameraController.cs

Naziv	Vrsta	Opis
target	Transform	Ciljani predmet fokusa kamere
offset	Vector3	Vektor razmaka između kamere i predmeta fokusa
timerText	Text	Tekstualni okvir koji sadrži trenutni brojač
pivot	Transform	Koordinate pivot točke u koju je usmjerena kamera

Target, instanca klase *Transform*, nam dopušta pristup lokaciji objekta igrača, a potrebna nam je kako bismo kameru usmjerili na taj objekt metodom .LookAt(Transform), *offset* odnosno instanca klase *Vector3*, podešava se na početku, te označava vrijednost udaljenosti kamere od objekta igrača (X, Y i Z koordinate), *pivot* odnosno instanca klase *Transform* koja označava lokaciju kamere.

Skripta sadrži 5 metoda, od čega su 2 nasljeđene: Start() i LateUpdate(). LateUpdate() je funkcija koja se okida nakon izvršenja svih Update() poziva, te je zbog toga korisna upravo za pomicanje kamere, kada se igrač mogao pomaknuti u fazi Update() funckije:

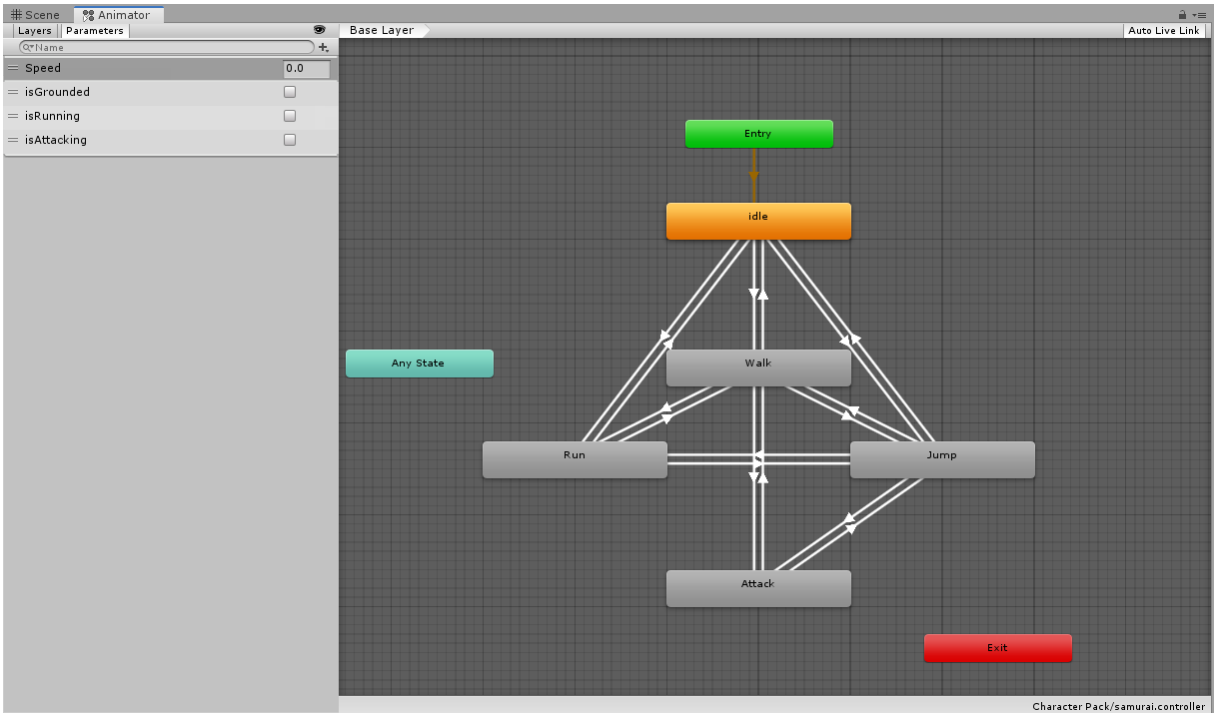
```
    public string FloatTimeToString()
    {
        System.TimeSpan t =
System.TimeSpan.FromSeconds(currentTimer);
        return string.Format("{0:D2}:{1:D2}.{2:D3}", t.Minutes,
t.Seconds, t.Milliseconds);
    }
    void Start()
    {
        Cursor.lockState = CursorLockMode.Locked;
        offset = target.position - transform.position;
        pivot.transform.position = target.transform.position;
        pivot.transform.parent = null;
    }
    void LateUpdate()
    {
        if (!FindObjectOfType<PauseManager>().isPaused)
        {
            if (!stopTime)
            {
```

```
                currentTimer += Time.deltaTime;
                timerText.text = "";
                timerText.text = FloatTimeToString();
            }

            pivot.transform.position = target.transform.position;

            float horizontal = Input.GetAxis("Mouse X") *
rotateSpeed;
            pivot.Rotate(0f, horizontal, 0f);

            float vertical = Input.GetAxis("Mouse Y") * rotateSpeed;
            if (invertedView) pivot.Rotate(-vertical, 0f, 0f);
            else pivot.Rotate(vertical, 0f, 0f);

            if (pivot.rotation.eulerAngles.x > maxViewAngle &&
pivot.rotation.eulerAngles.x < 180f) pivot.rotation =
Quaternion.Euler(maxViewAngle, pivot.rotation.eulerAngles.y, 0f);
            if (pivot.rotation.eulerAngles.x > 180f &&
pivot.rotation.eulerAngles.x < 360f + minViewAngle) pivot.rotation =
Quaternion.Euler(360f + minViewAngle, pivot.rotation.eulerAngles.y,
0f);

            float cameraAngleX = pivot.eulerAngles.x;
            float cameraAngleY = pivot.eulerAngles.y;
            Quaternion rotation = Quaternion.Euler(cameraAngleX,
cameraAngleY, 0f);
            transform.position = target.position - (rotation *
offset);
            if (transform.position.y < target.position.y)
transform.position = new Vector3(transform.position.x,
target.position.y - 0.5f, transform.position.z);
            transform.LookAt(target);
        }
    }
    public void StopTime()
    {
        StartCoroutine(Stop());
    }
    IEnumerator Stop()
    {
        stopTime = true;
        yield return new WaitForSeconds(3);
        stopTime = false;
    }
```

4.3.15. **Izrada i prenamjena animacija**

Animacije koje su preuzete zajedno s modelom *Samurai* iz *Asset Store*-a, izrađene su u drugom alatu (alatu s kojim se izrađivao 3D model), i moguće ih je uvesti u alat, međutim traže prenamjenu. Prvotno, s objekta igrača sam uklonio komponentu „Animation“ kako je ona zastarjela i ne radi na jednostavniji način na koji radi „Animator“. Animator nam pruža mogućnost vizualnog povezivanja stanja objekata i animacija, kao i dodavanje osnovnih varijabli (bool, float, int) kojima možemo regulirati status objekta i koju ćemo animaciju u slučaju koje promjene okinuti. Prozor animatora možemo otvoriti iz padajućeg izbornika s alatne trake: „Windows“ – „Animation“ – „Animator“.

Osnovna animacija je animacija naziva „idle“ i označava neaktivnost objekta, odnosno ona se izvršava u petlji dokle model igrača miruje. Kada igrač započne kretnju okida se animacija „Walk“, na pritisak tipke shift okida se animacija „Run“, na pritisak razmaknice aktivira se animacija „Jump“, a ukoliko je korisnik pritisnuo lijevu tipku miša, okida se animacija „Attack“.



Slika 6. Parametri (lijevo) i povezanost animacija [autorski rad]

4.4. Buildanje

Kao i svaka aplikacija, i ova igra mora se „kompajlirati“, odnosno prevesti u jezik razumljiv računalu (prevoditelju). Prije konačnog *buildanja* igre, potrebno je svugdje gdje smo postavljali zaključavanje kursora dodati i naredbu za isključivanje prikaza istog: *Cursor.visible = false;* i obrnuto (postavljanje na true kada se korisnik vrati u igru). Ukoliko su sve scene dodane u *Build settings* , možemo pritisnuti iz padajućeg izbornika „File“ – „Build and Run“. Odabir te opcije prikazat će nam pretraživač datoteka, i potrebno je odabrati folder gdje želimo spremiti igru. Nakon podužeg čekanja, na odabranoj lokaciji dobili smo izvršnu datoteku kreirane igre, odakle igru možemo pokretati. Finalna veličina naše igre je 73 megabajta.

5. Zaključak

Izrada igre u tri dimenzije složen je i opsežan posao. Potrebno je znanje iz više grana kako bi istu izradili sami, a ujedno je potrebno i savladati sve specifičnosti alata za izradu koji koristimo. Osobno smatram da je razvoj igara danas uvelike pojednostavljen, u odnosu na stanje početkom 2000.-ih godina. Tada je za razvoj bilo potrebno prvotno razviti jezgru igre (eng. *game engine*) kojeg se potom prilagođavalo potrebama. Taj je proces uvelike olakšan koristeći jezgre koje su dostupne na korištenje i vrlo jednostavne za promjenu, kao što je Unity ili Unreal Engine.

Iako osobno smatram da je izrada igre bez jezgre ustvari polovičan posao, ostvariti prve korake i prototipe je puno lakše u okruženju koje nudi jednostavnost ispred mogućnosti. Smatram da je ovaj pristup izuzetno doprinio razvoju industrije video igara, jer brojni su korisnici koji imaju ideje koje bi htjeli ostvariti, međutim nisu dovoljno kompetentni ni fluentni u programiranju i programskim jezicima da bi samostalno napravili jezgru, pa je time dozovljeno kreativnim pojedincima da ostvare svoje ideje, iako nemaju potpunu podlogu za izradu iste.

Osobno bih preporučio svakome tko voli kreirati vlastiti sadržaj i programirati, da apsolutno isproba svoje snalaženje u alatu Unity. Početno, stvari će ići teško i za jednostavnije korake trebat će puno vremena, ali korištenjem i usvajanjem, stvari će ići sve glađe, a na kraju ovoga rada mogu reći da neke stvari idu i automatizmom nakon nekog vremena. Kao i svaki specifični sustav, i ovaj ima svoja „pravila igre“, pa je vrlo bitno savladati osnovno snalaženje sa specifičnim funkcijama, kao npr. načina pisanja skripti i povezivanja objekata u sceni s istom. Na kraju mogu reći da mi je izrada ovog rada bila izuzetno draga i da smatram da sam naučio usmjeriti logičko (programsko) razmišljanje u vidu 3D objekata i manipulacije 3D scenama.

Popis literature

[1] Baer R., „Videogames: In the beginning“, 2005., Rolenta Press, str. 96.

[2] Huhtamo E., „*Slots of Fun, Slots of Trouble: Toward an Archaeology of Electronic Gaming*“, [Na internetu]. Dostupno: http://classes.design.ucla.edu/Fall06/10/HuhtamoGameCulture.pdf. [pristup: 25.8.2019].

[3] Nutt, C., „*Birthday Memories: Sony PlayStation Turns 15*“, 2010., [Na internetu]. Dostupno: https://www.gamasutra.com/view/feature/6122/birthday_memories_sony_.php?print=1. [pristup: 25.8.2019.]

[4] Unity Dokumentacija, [Na internetu]. Dostupno: https://docs.unity3d.com/Manual/index.html. [pristup: 27.8.2019].

[5] Nacke, Lennart E.; Drachen, Anders; Kuikkaniemi, Kai; Niesenhaus, Joerg; Korhonen, Hannu; van den Hoogen, Wouter; Poels, Karolien; IJsselsteijn, Wijnand; et al., "*Playability and Player Experience Research*", 2009., [Na internetu]. Dostupno: https://hcigames.com/wp-content/uploads/2015/01/Playability-and-Player-Experience-Research.pdf. [pristup: 1.9.2019].

[6] Edge, N. „A History on the Advancement of Games Consoles And their respective Games“, [Na internetu]. Dostupno: https://commerce3.derby.ac.uk/ojs/index.php/c4e/article/download/90/67#page=58 [pristup: 31.8.2019].

[7] Montfort, N.; Bogost I., „Racing the Beam: The Atari Video Computer System“, [Na internetu]. Dostupno: https://books.google.hr/books?hl=hr&lr=&id=DqePfdz_x6gC&oi=fnd&pg=PR5&dq=atari+vcs&ots=FSVslZNwRB&sig=AxU2zj1JB8WQA5WfGKDLXykjXEM&redir_esc=y#v=onepage&q=atari%20vcs&f=false [pristup: 31.8.2019].

Popis slika

Popis tablica