Izrada 2D platformera u programskom alatu Unity

Mihalić, Petar

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Petar Mihalić

IZRADA 2D PLATFORMERA U PROGRAMSKOM ALATU UNITY

ZAVRŠNI RAD

Varaždin, 2021.

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Petar Mihalić

Matični broj: 0016136283

Studij: Informacijski sustavi

IZRADA 2D PLATFORMERA U PROGRAMSKOM ALATU UNITY

ZAVRŠNI RAD

Mentor:

Doc. dr. sc. Marko Konecki

Varaždin, kolovoz 2021.

Petar Mihalić

Izjava o izvornosti

Izjavljujem da je moj završni/diplomski 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 potvrdio prihvaćanjem odredbi u sustavu FOI-radovi

Sažetak

Tema ovog rada je izrada vlastite igre u programskom alatu Unity. Vrsta igre je 2D platformer te se prije same izrade igre upoznajemo s potrebnim alatima za izradu projekta i sa žanrom igre. Pregledom poznatih 2D igara iz povijesti i današnjice te istraživanjem o načinima izrade i različitim rezultatima koje je moguće ostvariti uz pomoć odabranih alata odlučio sam se za izradu 2D platformera korištenjem spriteova (*eng. sprites*) umjesto pločica (*eng. tiles*). Spriteovi po mojem mišljenju omogućuju veću slobodu u dizajniranju igre te pridodaju realizmu igre.

Nakon odabira načina rada definirana je tema igre u kojoj imamo glavnog agenta kojim upravlja igrač čija je misija uništiti neprijatelje, izbjeći metke i ponore te pokupiti ukradene stvari tijekom noćne misije. Izrada igre podijeljena je na nekoliko dijelova, te će stoga prvo biti prikazano dizajniranje nivoa, zatim animiranje te povezivanje istog u programskog alatu Unity, stvaranje platforme po kojoj se igrač može kretati i povezivanje više različitih nivoa zajedno. S kreiranim virtualnim svijetom nove igre potrebno je stvoriti igrača te ostale entitete s kojima glavni lik ima interakcije, poput neprijatelja, kovčega i oružja. Izrada likova slijedi slične korake kao i nivo, kreće se s dizajnom pa animacije i zatim implementacija u samu igru gdje je opisan kod za razne moguće akcije poput kretanja, pucanja i slično.

Za završavanje igre stvoreno je korisničko sučelje, dodani zvukovi i razni ostali efekti koji poboljšavaju kvalitetu i dojam igre. Na samom kraju kreirana je izvršna datoteka koja se može pokrenuti kao i svaka druga aplikacija.

Ključne riječi: 2D; platformer; Unity; sprite; skripte; dizajn; animacija; implementacija

Sadržaj

1. Uvod

Ovaj rad obradit će temu izrade 2D platformera u programskom alatu Unity. Odabrao sam ovu temu jer me zanima svijet izrade igara a još se u obrazovanju nisam ozbiljno susreo s tom temom. Također želim naučiti raditi u novom programu, koji komponira programiranje s kojim sam dobro upoznat zajedno sa dizajnom i određenim umjetničkim elementom koji čini svaku igru posebnom. Budući da je crtanje, dizajniranje i skiciranje jedan od mojih hobija smatram da je ova tema vrlo prikladna i da mogu puno naučiti u procesu izrade ovog rada.

Tema rada je razgrađena u dijelove na sličan način kako sam i ja istraživao o izgradnji i karakteristikama ovakvih igara. Prije izgradnje same igre prikazat ću povijest, utjecaj i trenutnu popularnost 2D platformera prisjećajući se nekih od najpopularnijih klasika pa sve do današnjih predivnih platformera čija slava leži na kvalitetnoj implementaciji i odličnoj priči. Naravno uz ideju potrebni su razni programski alati koji omogućuju realizaciju ideja u gotove proizvode, ovdje ću ukratko opisati korištene alate, od kojih je glavni Unity, snažan i popularan alat za izradu ne samo 2D već i najvećih 3D hitova u industriji igara. Dizajn i ilustracije bit će izrađene u Photoshop alatu dok će skripte za upravljanje i ponašanje dijelova igre biti napisane u C# jeziku koristeći Visual Studio. Usporedit ću i dva glavna načina na koje se izgrađuju 2D platformeri, koristeći spriteove (*eng. sprites*) i pločice (*eng. tiles*), između kojih sam se ja odlučio za spriteove jer mi omogućuju veću slobodu kod dizajniranja nivoa te sveukupne atmosfere igre. Kod izgradnje same igre prikazat ću dizajniranje, animiranje, implementaciju i povezivanje prvo okoline nivoa a zatim glavnog heroja, neprijatelja te ostalih elemenata i interaktivnih objekata. Objasnit ću programsku logiku kretanja, interakcije, borbe i ostalih funkcionalnosti implementiranih u igru. Za završavanje igre dodat ću zvukove, te naravno potrebno korisničko sučelje.

Nadam se da ću ovim radom ostvariti svoj cilj te naučiti raditi u programskom alatu Unity i da ću uspješno povezati znanje u programiranju sa novim načinom razmišljanja u smislu stvaranja zanimljive i funkcionalne 2D igre. Neka od glavnih pitanja na koje ću ovim radom probati pronaći odgovor su težina izrade 2D platformera od nule i važnost dobrog dizajna u usporedbi s pričom te kvalitetnim korisničkim upravljanjem.

2. Metodike i tehnike rada

Pristupio sam ovoj temi na način da sam prvo odlučio dobro se upoznati sa procesom razvoja te samim programima koji se koriste, cilj mi je bio saznati što je korisno i nužno a što opcionalno. U procesu testiranja, isprobavanja funkcionalnosti te stvaranja manjih jednostavnih nivoa i igara za vježbu shvatio sam i razne mogućnosti te opcije koje su developeru dostupne kod izrade igara. Kao pomoć za usmjeravanje i vođenje kroz osnovne i jednostavne funkcionalnosti 2D platformera koristio sam razne članke i video materijale dostupne na internetu. Postepeno učenje i bolje razumijevanje procesa razvoja igara odvijalo se postupkom kojim je strukturiran i ovaj rad, stoga ću u nastavku detaljnije opisati već spomenute programe od kojih je glavni Unity za izradu same igre odnosno povezivanje skripata pisanih unutar Visual Studio-a sa dizajnom nivoa i likova koji će biti izrađeni u Adobe Photoshop-u.

2.1. Unity

Tvrtka Unity Techologies je objavila prvu verziju Unity-a, razvojnog okruženja za izradu igara, 2005. godine, no razgovori i rad na alatu sežu sve do 2002. godine kada su se dvojca programera našla na forumu pokušavajući pronaći rješenje za svoje individualne projekte. Odlučili su spojiti snage, maknuti fokus sa svojih projekata te stvoriti novo razvojno okruženje koje će kasnije postati jedno od najpopularnijih i najkorištenijih alata za izradu 2D i 3D igara. [1][2]

Unity danas koristi više od 2.5 milijuna registriranih programera, što ga stavlja u sam vrh sa ostalim dobro poznatim i respektabilnim alatima poput Unreal Engine-a i Godot-a [3]. Jedan od razloga zašto je Unity toliko popularan danas je potpora za različite platforme, iako je prvo osmišljen i pušten u rad samo na Mac OS X platformi, danas je proširen na čak 25 različitih platformi [3]. 2D, 3D, VR i AR su također snažni atributi koji stavljaju Unity ispred ostalih alata, u svijetu 2D, VR i AR igara, Unity dominira tržište sa postocima korištenja [3]. Alat je poznat po kvalitetno implementiranom renderiranju i izgledu finalnog proizvoda te brojnim funkcionalnostima koje omogućuju programerima ostvarivanje njihovih ideja [3].

Što je mene osobno impresioniralo je jednostavnost korištenja, pogotovo za početnike. Sučelje programa je vrlo intuitivno iako na prvi pogled izgleda komplicirano i natrpano opcijama. Pisanje koda je također puno jednostavnije nego što sam očekivao, skripte su odlično integrirane u sam alat te nije potrebno napredno znanje u programiranju kako bi se

stvorila jednostavna igra. Odlično je što postoji i više verzija programa, te je tako omogućeno stvaranje igara svima koji su željni uložiti trud i učiti, budući da postoji besplatna verzija Unity-a koja sadrži više nego dovoljno funkcionalnosti i opcija za stvaranje kvalitetne igre. Uz besplatnu opciju postoje naravno i profesionalne opcije koje nude bolje i naprednije funkcionalnosti. Posljednja komponenta koju smatram važnom za početak stvaranja u Unity-u je Asset Store u kojem su dostupni brojni gotovi modeli, dizajni, predlošci, skripte i slično.

2.2. Visual Studio

Visual Studio je integrirano razvojno okruženje (*eng. Integrated developmet environment*, IDE) koje ima brojne korisne funkcionalnosti za pisanje koda, vrlo je proširivo okruženje te podržava izradu aplikacija za najpopularnije platforme [4]. Iako nije nužno koristiti Visual Studio za izradu 2D platformera, odlučio sam uzeti ovaj alat za pisanje skripti u C# jeziku zbog toga jer sam već vrlo dobro upoznat s alatom i jer sadrži korisne funkcionalnosti poput IntelliSense-a, otklanjanja pogrešaka i jednostavnu navigaciju. Napomenuo bih i da se u Visual Studio mogu instalirati razni programski dodaci za rad s određenim programima ili stvaranje određenih aplikacija. Stoga je za Unity potrebno instalirati modifikaciju pod nazivom „*Game development with Unity*“ što će uvelike olakšati pisanje skripata. Koristit ću Visual Studio Community 2019, besplatnu verziju koja sadrži sve što mi je potrebno za ovaj rad.

2.3. Adobe Photoshop

Budući da sam odlučio sam dizajnirati nivoe, likove i ostale elemente, za to ću koristiti Adobe Photoshop. Razlog tome je sličan kao i kod Visual Studio-a, već sam od prije upoznat s programom i smatram da je jedan od najkvalitetnijih u svojem području. Photoshop je danas standard u digitalnom uređivanju slika, koristi se za rasterske slike, grafički dizajn i stvaranje digitalne umjetnosti [5]. Photoshop je najpoznatiji alat za uređivanje slika a jedna od glavnih karakteristika programa su slojevi. Alat omogućuje izradu i obradu grafičkih elemenata u slojevima koje možemo zamisliti kao prozirne papire naslagane jedan na drugog, gdje možemo na svakom od njih raditi zasebno. Ta funkcionalnost je pretežito korisna kod dizajniranja 2D platformera jer omogućuje izradu scenarija pomoću spomenutih slojeva koji se zatim mogu ubaciti u Unity te animirati tako da se dobije osjećaj dubine i realizma. Vrlo je korisno i za kreiranje likova jer se anatomski dijelovi mogu odvojiti u slojeve te tako lagano animirati unutar Unity-a.

3. 2D platformeri

2D platformeri su žanr igara poznat po svojoj jednostavnosti, lakom učenju i jednostavnom pokretanju na gotovo svim uređajima. Iako su kvaliteta, detalji i kompleksnost igara napredovali zajedno s tehnologijom, 2D platformeri i dalje su u osnovi ostali isti. Koncept gotovo svakog 2D platformera je mogućnost kretanja uz pomoć jednostavnih kontrola koje može svatko lako i brzo shvatiti. Mogućnost kretanja iz ekrana u ekran na kojem svaki donosi nekakav izazov ili krije tajne za otkrivanje je ono što drži igrače vezane uz ovaj žanr. Osobno su 2D platformeri zanimljivi zbog testiranja preciznosti, vremenske točnosti i brzine reakcija, no naravno ovaj žanr nudi puno više. Od zapanjujućih zagonetki, zanimljivih avantura i istraživanja pa do predivnih i smirujućih igara koji svoju slavu postižu nevjerojatnom pričom ovaj žanr nudi nešto za svakoga što je dokazano i svevremenskom popularnošću 2D platformera.

3.1. Povijest i popularnost

Žanr 2D platformera je nastao u ranim osamdesetima, te je jedna od prvih igara bila Space Panic tvrtke Universal [6]. Navedena igra bila je vrlo jednostavna, bez skakanja i čak bez pomicanja kroz ekrane kada igrač dotakne rub nivoa. To se ubrzo promijenilo poznatom igrom tvrtke Nintendo izdanom 1981. godine, Donkey Kong, ta igra se smatra prvim pravim platformerom [7]. 90-te su bile zlatno doba 2D platformera, dominirane nekim od najpopularnijih igara ovog žanra poput Super Mario World tvrtke Nintendo i Sonic The Hedgehog izdan godinu dana kasnije od tvrtke Sega [7]. Za one koji su željeli malo drugačiji tempo i stil igre, postojali su poznati Metroid i Castlevania serijali igara [7].

Razvojem tehnologije i pojavom 3D igara, 2D igre su izgubile svoju veliku popularnost te je cilj svih tadašnjih konzola bio da sadrže najbolje i najnovije 3D naslove [7]. Usprkos tome platformeri nisu nestali već su se prilagodili novim trendovima te su zaživjeli 3D platformeri i postali standard za sve konzole. [7] Neki od poznatijih naslova ovog doba 3D platformera su Super Mario 64 te Crash Bandicoot. 2D igre su i dalje nastavile održavati svoj žanr i igrače, uz popularne 3D igre nalazili su se i neki od 2D klasika: Metroid Fusion, Sonic Advance, Castlevania: Aria of Sorrow, New Super Mario Bros [7].

U zadnjih nekoliko godina pojavio se ponovni rast u popularnosti 2D platformera zahvaljujući brojnim indie naslovima koji su vratili žanr na scenu [6]. Neki od danas najpopularnijih novijih igara su Cave Story, Shovel Knight, Hollow Knight te osobno najdraži

Ori and the Blind Forest i Cuphead, dvije igre koje su vrlo vizualno impresivne, jedna u modernom i detaljnom stilu dok je druga vođena starim i nostalgičnim dizajnom. Popularnost i ocjene navedenih i mnogih drugih 2D platformera je dokaz da žanr nije ostao u povijesti već je dobio novi vjetar u leđa te primamljuje nove generacije i budućnost 2D platformera izgleda uzbudljivo.

3.2. Načini izrade

Kod izrade 2D platformera dolazi do podijele načina izrade, koji se diskutiraju na raznim forumima i gotovo su prva bitna odluka svakog programera kod kreiranja nove igre. Podjela se odnosi na stvaranje nivoa ili virtualnog svijeta igre s naglaskom na dizajn, naime riječ je o podijeli na dizajn baziran na pločicama (*eng. tile-based*) i dizajn baziran na spriteovima (*eng. sprite-based*).

Stvaranje igre korištenjem pločica (*eng. tiles*) omogućava brže i jednostavnije dizajniranje nivoa no ograničava slobodu dizajniranja. Za dizajniranje igre uz pomoć pločica u programskom alatu je potrebno prikazati mrežu (*eng. grid*) koja će predstavljati mjesta na koja će se stavljati pločice, veličina mreže i pločica može biti lako određena i mijenjana od strane programera. „Tilemaps“ je Unity-eva funkcionalnost koja omogućava rad sa pločicama. [8] setovi pločica (*eng. tilesets*) se koriste za dizajniranje i kreiranje pločica, set je zapravo jedna slika koja sadrži sve manje pločice u sebi te se prije stavljanja u prije spomenutu mrežu treba taj set rastaviti na manje dijelove odnosno individualne pločice. Nakon rastavljanja Unity ima vrlo jednostavan i intuitivan način izrade nivoa tako da se odabere željena pločica i klikom na polje u mreži programer može crtati svoj nivo [8].

Drugi način dizajniranja igre je uz pomoć spriteova (*eng. sprites*), iako se i pločice smatraju kao spriteovi, u ovom kontekstu se misli na veće grafičke dijelove koji su neovisni o mreži te ne moraju biti dizajnirani za međusobno spajanje. Moguće je dizajnirati cijeli nivo odjednom u nekom zasebnom programu za obradu slika kao što je Adobe Photoshop te već gotove slike staviti u Unity te tako dizajnirati svaki željeni detalj i poziciju na nivou. Ova metoda omogućava puno veću slobodu u dizajniranju te može stvoriti realističniji rezultat no potrebno je puno više vremena kako bi se realizirala.

Razlike između metoda što se tiče ostatka izrade igre gotovo i nema, „*2D collider*“ odnosno objekt koji se stavlja na gotov dizajn kako bi predstavljao područje po kojem se igrač može kretati ne ovisi o načinu dizajniranja te se koristi isto u oba slučaja. Mala razlika je moguća ovisno o vrsti igre u tome što su performanse bolje kod korištenja setova pločica.

4. Opis vlastite igre

Igru koju sam izradio kao praktični dio ovog rada zove se Watch Out, naziv je igra riječi jer upozorava igrača da bude oprezan a s druge strane opisuje da svaki od neprijatelja nosi ručni sat koji igra ulogu u cijeloj priči i mehanici igre. Žanr igre je avantura i akcija.

Igra se odviju noću te započinje na rubu grada, agent se nalazi ispred noćnog kafića kada dobiva novu poruku od šefa. Zadana mu je nova misija, agent dobiva informaciju da je ukradeno pola milijuna dolara i kolekcija od 12 skupih satova. Glavni lopov iza operacije se trenutno nalazi na drugom brdu izvan grada gdje ima svoje skladište i sjedište. Kako bi sačuvao ukradene vrijednosti odlučio je novac podijeliti u kovčege a kolekciju satova raspodijeliti svojim ljudima tako da svatko čuva jedan sat. Kovčezi sa novcem su raspodijeljeni kroz spomenuta dva brda izvan grda te su nadzirani cijelo vrijeme. U poruci koju agent dobiva saznaje da ga njegovo oružje čega na krovu zgrade te ga treba pokupiti kako bi mogao ubijati lopove. Na putu van grada agent prelazi most sa kojeg se može vidjeti traženo skladište u daljini. Sljedeće agent dolazi do prvog brda gdje mu je zadaća da ubije neprijatelje, uzme njihove satove te pronađe sve kovčege sa novcem. Posljednja scena je kada agent dolazi do drugog brda gdje se nalazi skladište. U skladištu se nalaze već viđeni neprijatelji zajedno sa njihovim šefom koji ima pancirku pa ga je teže ubiti. Nakon što agent riješi posao u skladištu misija završava tako da se popne na sigurno mjesto iza skladišta. Kroz igru igrač mora paziti da izbjegne neprijateljske metke tako da se sagne ili ih preskoči, putem kojim agent prolazi misiju postoje i ponori u koje agent može pasti pa treba i na to paziti.

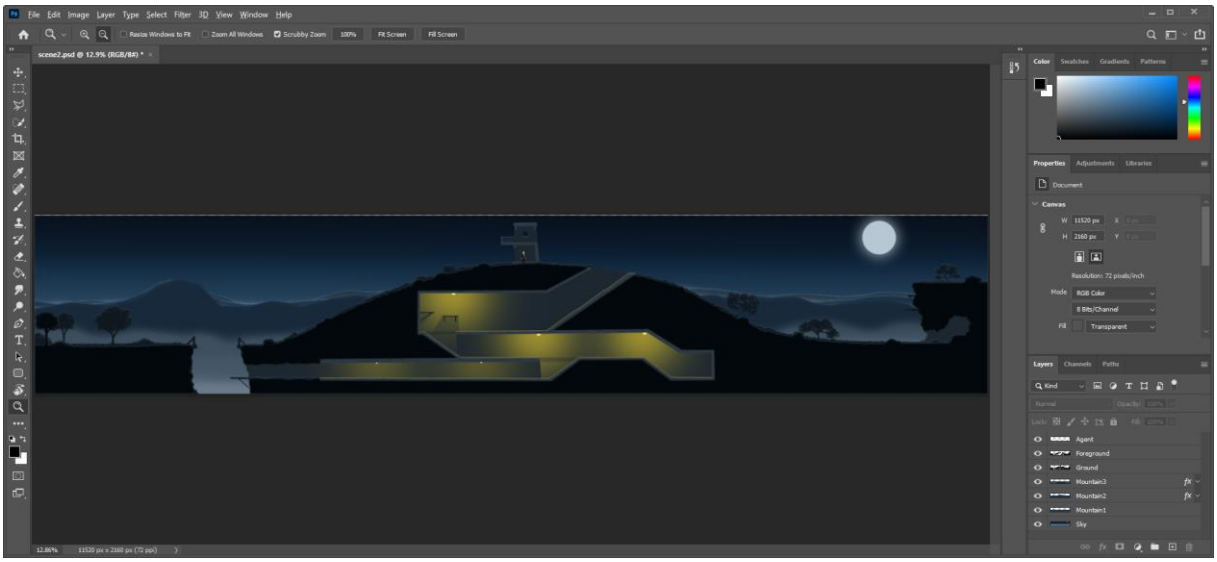
U glavnom izborniku igrač ima mogućnost pokrenuti igru, prilagoditi opcije kao što su glasnoća pozadinske glazbe i zvučnih efekata te zatvoriti igru. Uz opcije nalazi se i animacija agenta kojim igrač upravlja kroz igru. Tijekom same igre igrač može pauzirati igru i otići natrag na glavni izbornik. Ako agent umre igrač ima opciju pokušati ponovno ili se vratiti u glavni izbornik. Nakon uspješno obavljene misije prikazuje se ekran sa statistikom prikupljenih satova i novca te gumb za vraćanje u glavni izbornik gdje igrač može opet započeti igru ako želi poboljšati rezultat.

5. Izrada nivoa

Ova igra se odvija preko tri različita nivoa, u nastavku ću prikazati izradu drugog nivoa kao primjer, no svi su izrađeni istim postupkom.

5.1. Dizajniranje nivoa

Dizajnirao sam svaki grafički dio igre u Adobe Photoshop-u, pa tako i nivoe. Bitno je da je okolina podijeljena u slojeve kako bi mogao kasnije kontrolirati micanje daljnjih i bližih slojeva za efekt paralakse. Kada je proces dizajniranja gotov i slojevi željeno podijeljeni bitno je da se svaki sloj zasebno izveze kao PNG datoteka, za to sam koristio funkcionalnost Photoshop-a „*Layers to files…*“. Kod uvoza datoteka u Unity potrebno je maknuti kompresiju kako bi dobili punu oštrinu slike u igri.

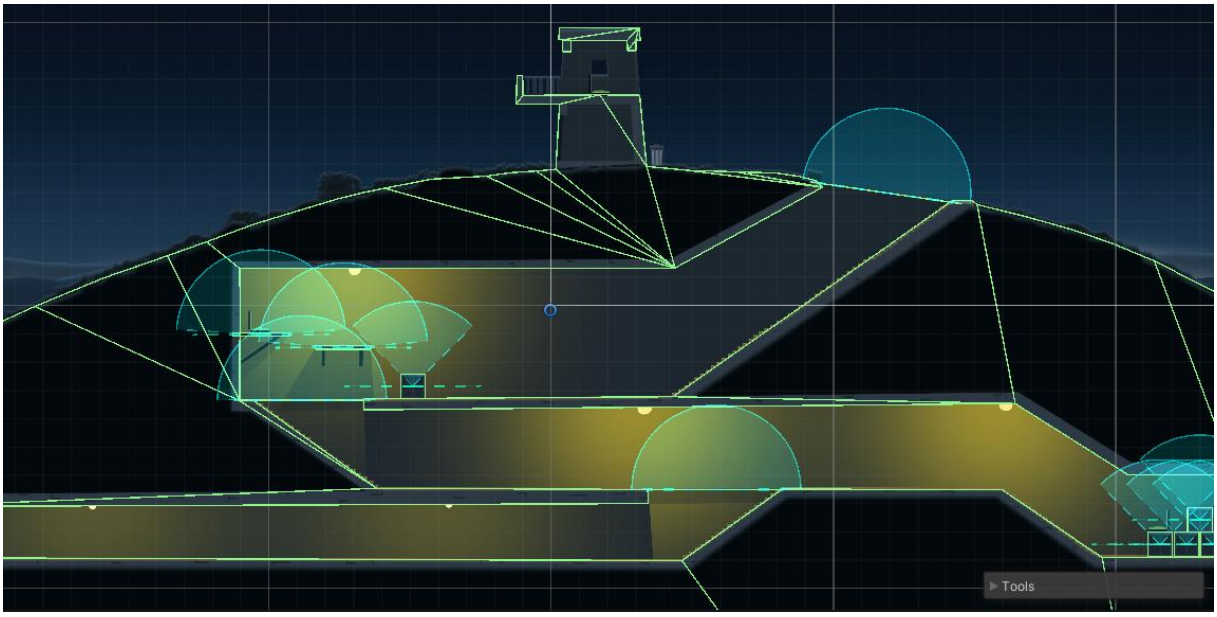


Slika 1: Drugi nivo u Photoshop-u i prikaz slojeva

5.2. Stvaranje platforme

Kada se slike okoline uvezu u Unity, poželjno ih je sortirati u grupe kao što su *Background*, *Player* i *Foreground* za lakše upravljanje te ih poredati unutar grupa kako bi slojevi bili prikazani pravilnim redoslijedom. Iako okolina nakon toga izgleda gotova Unity i dalje ne zna gdje bi igrač trebao hodati pa bi kod dodavanja objekta sa fizikom isti objekt samo

pao s ekrana zbog gravitacije, zbog toga se dodaje *Collider2D*, u ovom slučaju točnije *PolygonCollider2D*, to je komponenta koja se dodaje na objekt na kojem želimo da se sve odvija, u ovom primjeru to je objekt *Ground.* Dodavanjem komponente nastat će zeleni peterokut kojeg lagano možemo urediti da s njime napravimo oblik po kojem će se igrač kretati. Ako u nivou postoje platforme ili dijelovi koji fizički nisu spojeni sa zemljom tada se može dodati više *Collider2D* komponenta i objekata koji ih sadrže ovisno o potrebi, za lakše micanje dodatne objekte sam postavio kao djecu u hijerarhiji glavnog objekta *Ground*.



Slika 2: Prikaz *Collider2D* i *Platfrom Effector 2D* komponenti

Uz glavni teren po kojem se igrač može kretati dodao sam zidne platforme poput polica, te drvene kutije i kantu za smeće. Na spomenute objekte sam uz *Collider2D* dodao i *Platform Effector 2D* komponentu, kako bi se *Effector* povezao na željeni *Collider,* potrebno je u *Collider* komponenti staviti kvačicu na svojstvo „*Used By Effector“ [9]. Platform Effector* omogućuje određivanje dijela *Collider 2D* komponente koji će zapravo blokirati objekte koji dođu u susret [10]. Na slici 2 kao primjer prikazana je drvena kutija i označeno je da će samo gornji dio blokirati igrača i to samo s gornje strane što znači da igrač može hodati ispred kutije i ako želi može skočiti na nju. Određene objekte poput zidnih platforma i podova prvi vrhu stepenica označio sam sa posebnom oznakom (*eng. tag*) kako bi kasnije uz pomoć skripte omogućio igraču da rotira *Platform Effector 2D* komponentu i tako se spusti s objekta.

5.3. Efekt paralakse i prijelaz nivoa

Kako okolina ne bi bila potpuno statična i da u 2D svijet dodam malo dubine što pomaže sa realizmom svijeta, na svaki prethodno spomenuti sloj dodao sam jednostavnu skriptu koja će micati slojeve na temelju određenog faktora i pomicanja kamere:

```
5  public class ParallaxLayer : MonoBehaviour
6  {
7      [SerializeField] private Vector2 parallaxLayerSpeed;
8
9      private Transform cameraTransform;
10     private Vector3 lastCameraPosition;
11
12     private void Start()
13     {
14         cameraTransform = Camera.main.transform;
15         lastCameraPosition = cameraTransform.position;
16     }
17     private void LateUpdate()
18     {
19         Vector3 deltaMovemet = cameraTransform.position -
             lastCameraPosition;
20         transform.position += new Vector3(deltaMovemet.x *
             parallaxLayerSpeed.x, deltaMovemet.y * parallaxLayerSpeed.y);
21         lastCameraPosition = cameraTransform.position;
22     }
23 }
```

Skriptu je potrebno dodati na svaki sloj koji se treba micati, nakon što je skripta dodana na objekt u sučelju Unity-a se pojavi nova komponenta u kojoj se mogu postaviti vrijednosti varijabli koje su u skripti označene kao javne (*eng.* public) ili sa `[SerializeField]` svojstvom. Skripta radi tako da množi pomak kamere sa brzinom koja je unesena kako bi se promijenila pozicija određenog sloja pozadine na kojem je skripta postavljena.

Prijelaz nivoa se aktivira tako da igrač dođe blizu desnog ruba trenutnog nivoa. Kako bi to postigao dodan je novi objekt u scenu koji sadrži *Collider2D* komponentu no ovaj puta je u komponenti označeno svojstvo „*Is Trigger*“ što znači da neće blokirati objekte nego ga možemo koristiti kao okidač za željene radnje. Za prijelaz na sljedeći nivo odnosno scenu na objekt je dodan sljedeći kod:

```
13 private void Awake()
14     {
15         GameObject panel = Instantiate(fadeInPanel, Vector3.zero,
             Quaternion.identity) as GameObject;
16         Destroy(panel, 1);
17     }
18     public void OnTriggerEnter2D(Collider2D collision)
19     {
20         if (collision.CompareTag("Player"))
```

```
21          {
22              StartCoroutine(Fade());
23          }
24      }
25
26      public IEnumerator Fade()
27      {
28          Instantiate(fadeOutPanel, Vector3.zero, Quaternion.identity);
29          yield return new WaitForSeconds(fadeWait);
30          AsyncOperation asyncOperation =
              SceneManager.LoadSceneAsync(nextSceneIndex);
31          while (!asyncOperation.isDone)
32          {
33              yield return null;
34          }
35      }
```

Kako bi u prikazanom kodu mogli upravljani sa scenama potrebno je uključiti biblioteku „Unity.SceneManagement“. U osnovi ova skripta koristi ugrađenu funkciju koja se pokreće kada objekt uđe u okidač kojeg smo postavili u Unity-u. Provjerava se kod ulaza objekta u okidač da objekt ima oznaku „Player“ kako se ne bi promijenila scena kada na primjer metak prođe kroz okidač. Ako objekt ima pravilnu oznaku učitava se sljedeća scena. U skriptu sam odlučio dodati dodatan kod kako bi ublažio prijelaz uz pomoć dva platna (*eng. canvas*) koja su animirana da prolaze iz bezbojne u crnu boju i obratno. Animacija iz crne u bezbojnu se pokreće kod učitavanja instance skripte u funkciji `Awake()`. Ako objekt zadovoljava spomenuti uvjet u funkciji ulaza objekta u okidač tada se pokreće *Coroutine*, funkcija koja se označava sa `IEnumerator` i omogućuje pauziranje funkcije dok se klasična funkcija odvije cijela pri svakom pozivu [10]. U ovoj funkciji se nalazi naredba koja stvara instancu drugog dijela spomenutog animiranog prijelaza te se aktivira minimalno čekanje u sekundama. Ekran će biti crne boje dok se scena ne učita i tada se nastavlja animacija za efekt blagog prijelaza.

6. Izrada likova

U igri se nalaze tri vrste likova, glavni agent kojim upravlja igrač, neprijatelji koji su lopovi i dio veće organizirane skupine koju vodi posljednji lik, njihov šef. Svi likovi su vizualno drugačije dizajnirani ali u istom stilu. U nastavku ću pobliže objasniti dizajniranje, animiranje te svojstva i upravljanje likovima.

6.1. Dizajniranje likova

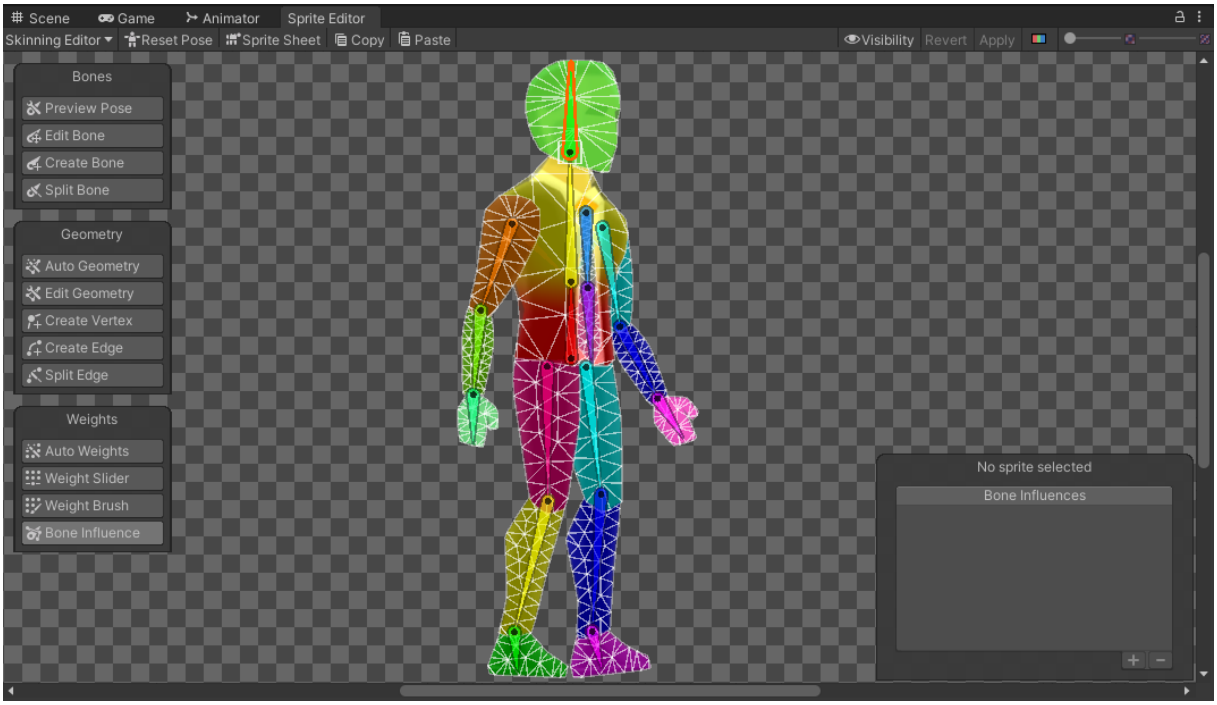
Likove sam dizajnirao u Adobe Photoshop alatu, bitno mi je kod dizajniranja da odvojim dijelove koje želim individualno micati u posebne slojeve, odmah sam slojeve prikladno imenovao kako bi se lakše kasnije snalazio u hijerarhiji lika. Također sam slojeve odmah poredao po smislenom redoslijedu i postavio ih međusobno u finalnu poziciju. Izgled likova prikazan je na sljedećoj slici.

Slika 3: Izgled agenta, običnog neprijatelja i glavnog neprijatelja

Svaki lik je dizajniran u posebnoj Photoshop datoteci te spremljen kao PSB datoteka umjesto standardne PSD datoteke poput prijašnje prikazanog nivoa. U Unity-u je potrebno uvesti „2D PSD Importer“ službeni paket koji omogućuje uvoz PSB datoteka te od njih stvori grupu sa zasebnim slikama od svih slojeva i zapamti poziciju svakog sloja isto kako je spremljeno u Photoshop-u što uvelike olakšava animiranje i upravljanje likovima.

6.2. Animiranje likova

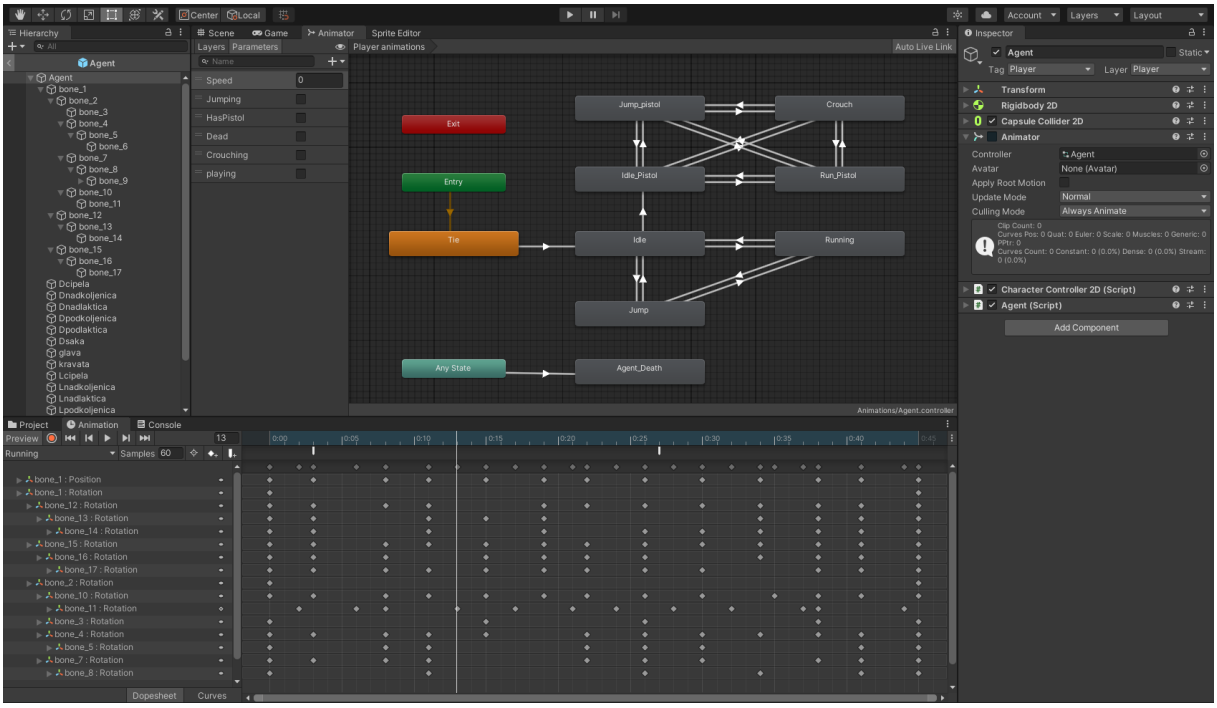
Prije animiranja treba povezati prethodno uvezene dijelove likova. Unity za ovo ima po mojem mišljenju odlično i intuitivno rješenje tako da se na lika crtaju kosti i izradi kostur. Prvo je potrebno otvoriti uvezenu datoteku u *Sprite Editor* prozoru te u istom odabrati *Skinning Editor* opciju, u tom pogledu imamo opcije kao što je vidljivo na sljedećoj slici. Nakon dodavanja kosti generira se distribucija upravljanja, odnosno Unity pokuša odrediti koja kost upravlja sa kojim dijelom tijela, ako se dijelovi preklapaju tada je potrebno urediti upravljanje tako da svaka kost upravlja samo željenim dijelom tijela kao što to je prikazano na slici.



Slika 4: *Skinning Editor* sučelje (uređivanje kostura agenta)

Nakon kreiranja kostura, sve kosti i njihova hijerarhija je vidljiva u hijerarhiji scene, sljedeće je potrebno dodati *Animator* komponentu i otvoriti *Animator* i *Animation* prozore u sučelju kao što je prikazano na sljedećoj slici. U *Animation* prozoru se stvara novi animacijski isječak (*eng. Animation Clip*) te se pritiskom na tipku za snimanje mogu micati kosti lika kako bi se kreirala i snimila animacija. Kreirane animacije se automatski dodaju u *Animator* prozor gdje se mogu dodati parametri i logika kojom će se animacije izmjenjivati s jedne na drugu. Dodane parametre će kasnije biti potrebno čitati i izmjenjivati uz pomoć skripte. Klikom na

animacijski isječak u spremljenoj mapi može se odabrati hoće li se animacija izvesti jednom (npr. animacija skoka) ili će se izvoditi u petlji (*eng. loop)* što je potrebno za animacije poput trčanja ili stajanja na mjestu.



Slika 5: Hijerarhija kostura, prozori *Animator* i *Animation*, komponenta *Animator*

6.3. Upravljanje agentom

Kako bi ova igra napokon oživjela potrebno je sve dosad napravljeno spojiti i omogućiti igraču da upravlja agentom. Za početak je potrebno na objekt agenta koji za sad sadrži dodani *Animator*, dodati dvije nove komponente, *Rigidbody 2D* i *Capsule Collider 2D*. *Rigidbody 2D* je komponenta koja na objekt primjenjuje fiziku i ovom komponentom ćemo upravljati za kretanje dok *Capsule Collider 2D* služi da označi prostor koji zauzima igrač, ova komponenta će spriječiti igrača da propadne kroz nivo i služi za detektiranje kontakta s ostalim objektima. Za ovu primjenu se specifično koristi *Capsule Collider 2D* zbog toga što ima zaobljene rubove pa kretanje po kosinama i rubovima platformi izgleda prirodnije [9]. Za upravljanje agentom kreirao sam dvije različite skripte zbog preglednosti, jedna je specifično za kontroliranje kretanja agenta od strane igrača dok se druga bavi s oružjem, pucanjem, primanjem štete, umiranjem te interakcijom s objektima za sakupljanje. Također ću opisati skripte koje se nalaze na objektima koje igrač skuplja jer su direktno vezane za upravljanje kretanjem igrača.

6.3.1. Kretanje

Skripta za kretanje je jedna od najbitnijih u cijeloj igri jer određuje osjećaj igre i željene funkcionalnosti koje želimo da igrač može kontrolirati. Što se tiče kretanja, želio sam da kontrole budu vrlo odzivne, bez kašnjenja i zaglađivanja brzine. Kod testiranja sam shvatio da postoje dva glavna načina micanja igrača koristeći ranije dodanu *Rigidbody 2D* komponentu, a to su da se koristi naredba *AddForce* ili da se direktno mijenja vrijednost parametra *velocity* [10]. Druga opcija rezultira trenutnim korištenjem zadanih vrijednosti dok prva koristi fiziku i kontrolira objekt ovisno o njegovoj težini, gravitaciji i slično, stoga sam odabrao koristiti drugu metodu. Nije mi se svidjelo što je taj način rezultirao vrlo velikom kontrolom agenta u zraku pa sam odlučio smanjiti kretanje po x osi ako agent nije na platformi. Izgled skripte i bitnije dijelove objasnit ću u nastavku.

```
5  public class CharacterController2D : MonoBehaviour
6  {
7      public float movementSpeed;
8      public float speedInAir;
9      public float jumpForce;
10     public Transform groundCheck;
11     public PhysicsMaterial2D noFriction;
12     public PhysicsMaterial2D fullFriction;
13     float movement;
14     float verticalMovement;
15     float speedOnLadder = 6f;
16     float originalSpeed;
17     bool isLadder;
18     bool isClimbing;
19     bool isGrounded;
20     bool wasGrounded;
21     Rigidbody2D rb;
22     Animator animator;
23     Collision2D twoWayPlatform;
24
25     void Start()
26     {
27         rb = GetComponent<Rigidbody2D>();
28         animator = GetComponent<Animator>();
29         originalSpeed = movementSpeed;
30     }
```

Varijable koje imaju svojstvo `public` pojavljuju se u komponenti skripte na objektu agent te se vrijednosti unose i mijenjaju u sučelju Unity-a što je vrlo korisno kod testiranja i određivanja vrijednosti. Varijable koje su objektnog tipa kao linijama 10-12 se određuju tako da se jednostavno u Unity-u povuku objekti na potrebno mjesto u komponenti skripte. Druge komponente istog objekta se mogu referencirati kao što je prikazano u `Start()` funkciji.

```
32      void Update()
33      {
34          wasGrounded = isGrounded;
35          isGrounded = false;
36          movementSpeed = speedInAir;
37
38          Collider2D[] colliders =
              Physics2D.OverlapCircleAll(groundCheck.position, 0.2f,
              1 << LayerMask.NameToLayer("Walkable"));
39          for (int i = 0; i < colliders.Length; i++)
40          {
41              isGrounded = true;
42              movementSpeed = originalSpeed;
43              if (!wasGrounded) animator.SetBool("Jumping", false);
44          }
45
46          if(movement == 0 && isGrounded) rb.sharedMaterial = fullFriction;
47          else rb.sharedMaterial = noFriction;
48
49          if (!Mathf.Approximately(0, movement))
50          {
51              transform.rotation = movement < 0 ? Quaternion.Euler(0, 180,
                    0) : Quaternion.identity;
52          }
```

Funkcija `Update()` se izvršava za svaku sliku igre (*eng. frame*), naredba u liniji 38 stvara krug zadanog radijusa na mjestu *groundCheck* objekta koji je postavljen direktno ispod agenta. Naredba sprema objekte koji se nalaze u sloju (*eng. layer*) „Walkable“. Ako se spremi neki objekt, točnije njegov *Collider 2D* tada znamo da je agent na platformi a ne u zraku. Kod u linijama 46 i 47 služi da mijenja materijal agenta, ako je agent na zemlji tada se ne smije klizati po kosinama, a ako je zraku ne smije se zalijepiti bočnu stranu drugog objekta. U linijama 49-52 se provjerava smjer kretanja kako bi se pravilno rotirao agent oko y osi.

```
54          verticalMovement = Input.GetAxis("Vertical");
55
56          if (isLadder && Mathf.Abs(verticalMovement) > 0f)
              isClimbing = true;
57
58          if (Input.GetKey(KeyCode.A) || Input.GetKey(KeyCode.LeftArrow))
59          {
60              movement = -movementSpeed;
61              rb.velocity = new Vector2(movement, rb.velocity.y);
62          }
63          else if (Input.GetKey(KeyCode.D) ||
              Input.GetKey(KeyCode.RightArrow))
64          {
65              movement = movementSpeed;
66              rb.velocity = new Vector2(movement, rb.velocity.y);
67          }
68          else
69          {
70              rb.velocity = new Vector2(0, rb.velocity.y);
71              movement = 0;
72          }
```

```
73          animator.SetFloat("Speed", Mathf.Abs(movement));
74
75          if (Input.GetKeyDown(KeyCode.Space) && isGrounded &&
              !animator.GetBool("Crouching"))
76          {
77              rb.velocity = new Vector2(rb.velocity.x, jumpForce);
78              animator.SetBool("Jumping", true);
79          }
80
81          if (Input.GetKeyDown(KeyCode.LeftControl) ||
              Input.GetKeyDown(KeyCode.C))
82          {
83              rb.constraints = RigidbodyConstraints2D.FreezePositionX |
                    RigidbodyConstraints2D.FreezeRotation;
84              animator.SetBool("Crouching", true);
85          }
86          if (Input.GetKeyUp(KeyCode.LeftControl) ||
              Input.GetKeyUp(KeyCode.C))
87          {
88              rb.constraints = RigidbodyConstraints2D.FreezeRotation;
89              animator.SetBool("Crouching", false);
90          }
91          if ((Input.GetKeyDown(KeyCode.DownArrow) ||
              Input.GetKeyDown(KeyCode.S)) && twoWayPlatform != null)
92          {
93              StartCoroutine(keepEffectorRotated());
94          }
95          if (isClimbing)
96          {
97              rb.gravityScale = 0f;
98              rb.velocity = new Vector2(rb.velocity.x, verticalMovement *
                    speedOnLadder);
99          }
100          else rb.gravityScale = 3f;
101     }
```

Ostatak `Update()` funkcije sastoji se od jednostavnih uvjeta koji provjeravaju unos preko tipkovnice i kontroliraju kretanje agenta i parametara animatora kako bi se prikazivale pravilne animacije. Napomenuo bih da se kod čučnja onemogućuje kretanje po x osi a smanjivanje visine *Capsule Collider 2D* komponente se odrađuje putem animacije.

```
103     public void OnCollisionEnter2D(Collision2D collision)
104     {
105         if (collision.gameObject.CompareTag("TwoWayPlatform"))
106         {
107             twoWayPlatform = collision;
108         }
109     }
110
111     private void OnTriggerEnter2D(Collider2D collision)
112     {
113         if (collision.CompareTag("Ladder"))
114         {
115             isLadder = true;
116         }
117     }
```

```
118
119     private void OnTriggerExit2D(Collider2D collision)
120     {
121         if (collision.CompareTag("Ladder"))
122         {
123             isLadder = false;
124             isClimbing = false;
125         }
126     }
127
128     IEnumerator keepEffectorRotated()
129     {
130         if (twoWayPlatform.gameObject.CompareTag("TwoWayPlatform"))
131         {
132             twoWayPlatform.gameObject.GetComponent<PlatformEffector2D>().
                    rotationalOffset = 180;
133             yield return new WaitForSeconds(0.3f);
134             twoWayPlatform.gameObject.GetComponent<PlatformEffector2D>().
                    rotationalOffset = 0;
135         }
136     }
137}
```

U skripti se nalaze još 4 funkcije, od kojih su 3 koje započinju na linijama 103, 111 i 119 vrlo jednostavne ugrađene funkcije koje provjeravaju nalazi li se agent na platformi sa koje se može spustiti ili blizu ljestva. Posljednja funkcija je tipa *IEnumerator* što označava da se definira *Courutine*, ova funkcionalnost je objašnjena prethodno kod prijelaza nivoa. U ovoj skripti se koristi za rotiranje *Platform Effector 2D* komponente kako bi se agent mogao spustiti s platforme, izvršavanje koda se pauzira na 0.3 sekunde kako bi agent stigao pasti kroz platformu prije nego se resetira da se opet može skočiti na nju. Funkcija se poziva u liniji 93 ako igrač pritisne tipku „S“ ili strelicu prema dolje.

6.3.2. Pucanje

Pucanje se provodi u različitoj skripti od prethodno opisanog kretanja, definiranje varijabli se provodi na isti način kao i u prethodno opisanoj skripti pa ću se fokusirati samo na elemente pucanja.

```
34 private void Update()
35     {
36         healthbarTr.eulerAngles = new Vector3(healthbarTr.eulerAngles.x,
             0, healthbarTr.eulerAngles.z);
37
38         if (Input.GetButtonDown("Fire1") &&
             agentAnimator.GetBool("HasPistol"))
39         {
40             Shoot();
41             pistolFlash.SetActive(true);
42         }
43         else
```

```
44          {
45              pistolFlash.SetActive(false);
46          }
47      }
48
49      void Shoot()
50      {
51          SoundManager.PlaySound("Pistol");
52          Instantiate(bulletPrefab, firePoint.position,
              firePoint.rotation);
53      }
```

Slično kao u skripti za kretanje agentom, provjerava se unos igrača preko tipkovnice u liniji 38. Uz pritisak na tipku provjerava se je li igrač pokupio pištolj, ukoliko je izvršavaju se linije 51-52, linija 51 bit će objašnjena u kasnijem poglavlju a ovdje je bitna izrada instance metka na poziciji koja je dodana na vrh pištolja u Unity-u i u smjeru okreta agenta. *GameObject* `bulletPrefab` predstavlja objekt koji se nalazi u projektu no nije dodan ni u jednu scenu, taj objekt na sebi sadrži jednostavnu skriptu iz koje bih izdvojio dvije linije koda:

```
rb.velocity = transform.right * speed;

Physics2D.IgnoreCollision(ladder.GetComponent<Collider2D>(),

       GetComponent<Collider2D>());
```

Prva linija postavlja smjer i brzinu kretanja metka, a druga linija se koristi više puta, naravno sa drugačijim prvim parametrom naredbe kako bi se zanemario sudar ako metak dotakne ljestve, kovčeg sa novcem ili ako metak uđe u područje u kojem neprijatelj detektira da je agent blizu.

Slika 6: Dizajn pištolja, blijeska i metka za pištolj

6.3.3.Interakcije s objektima

Igrač na početku treba otvoriti prvi kovčeg i uzeti pištolj kojeg će koristiti kroz misiju, zatim kada ubije neprijatelje treba uzeti satove koje nose na ruci i pronaći ostale kovčege te uzeti novac u njima.

Slika 7: Dizajn stvari koje se mogu pokupiti

U igri stoga postoje tri različita objekta, kovčeg sa pištoljem, kovčeg sa novcem i neprijatelj koji na sebi sadrže skriptu sa kodom za detektiranje kada je igrač blizu objekta te što se dogodi sa likom igrača, objektom i trenutnim rezultatom kada igrač djeluje na određeni objekt. Za početak ću objasniti kod za otvaranja i uzimanje novca iz kovčega.

```
31 void Update()
32     {
33         if ((Input.GetKeyDown(KeyCode.E) || Input.GetKeyUp(KeyCode.F)) &&
             SuitcaseAnimator.GetBool("playerIsClose"))
34         {
35             if (SuitcaseAnimator.GetBool("Open") == false)
36             {
37                 SoundManager.PlaySound("Suitcase");
38                 SuitcaseAnimator.SetBool("Open", true);
39             }
40             else if(!SuitcaseAnimator.GetBool("Take"))
41             {
42                 SoundManager.PlaySound("Money");
43                 SuitcaseAnimator.SetBool("Take", true);
44                 counter.AddToCount(moneyInSuitcase);
45             }
46         }
47     }
```

Sva tri navedena objekta imaju na sebi *Collider 2D* komponentu koja je postavljena da djeluje kao okidač (*eng. Is Trigger*) te se ugrađenim naredbama `OnTriggerEnter2D` i `OnTriggerExit2D` provjerava je li agent blizu objekta odnosno unutar okidača te se zapisuje podatak u parametar animatora objekta isto kao što agent provjerava je li blizu ljestva u

prijašnje prikazanom kodu za kretanje agenta. U priloženom dijelu koda u liniji 33 provjerava se je li igrač pritisnuo tipku za interakciju s objektom te nalazi li se unutar spomenutog okidača, ako je uvjet zadovoljen provjerava se je li kovčeg zatvoren, ako i taj uvjet prođe tada se mijenja parametar u animatoru i kovčeg se otvara. Kada je kovčeg otvoren igrač treba još jednom pritisnuti traženu tipku i ponavlja se sličan postupak, ovaj puta agent uzima novac/pištolj. Priložen kod je specifičan za kovčeg sa novcem no glavna razlika s kovčegom u kojem je pištolj da kod uzimanja pištolja agent dobiva pištolj u ruke sljedećim isječkom koda:

```
47          weapon.SetActive(true);
48          PlayerPrefs.SetInt("pistol", 1);
49          playersAnimator.SetBool("HasPistol", true);
```

Razlika koja je napravljena kod interakcije s neprijateljem prikazana je u sljedećem isječku koda:

```
48 if ((Input.GetKeyDown(KeyCode.E) || Input.GetKeyUp(KeyCode.F)) &&
    AgentIsClose && !watchLooted && enemyAnimator.GetBool("Dead"))
49      {
50          SoundManager.PlaySound("Watch");
51          enemyAnimator.SetBool("hasWatch", false);
52          counter.IncrementCount();
53          watchLooted = true;
54      }
```

Dodatno se provjerava je li sat već uzet s neprijatelja i je li neprijatelj mrtav, zatim ako je uvjet zadovoljen, parametar u animatoru neprijatelja se izmjeni tako da se uz pomoć animacije makne slika sata sa ruke mrtvog neprijatelja.

6.4. Upravljanje neprijateljima

U igru su za sad implementirane dvije vrste neprijatelja, klasični neprijatelj i glavni neprijatelj, dizajn im je različit kako bi ih igrač morao razlikovati no upravljanje njima je jednako. Stvaranje kostura te animiranje neprijatelja izrađeno je istim postupkom kao i za agenta. Ono što čini glavnog neprijatelja posebnim i težim za ubiti je to što može primiti više štete i puca puno brže od klasičnog neprijatelja. Obje vrste neprijatelja su zamišljene kao statični neprijatelji, gotovo kao straža jer smatram da nije prikladno za mehaniku i temu igre da neprijatelj hoda ili trći po nivou. Osim animacija htio sam neprijateljima dodati dinamičnu funkcionalnost koja će uvijek okrenuti neprijatelja prema igraču ako se igrač nađe iza leđa neprijatelja i ako mu je dovoljno blizu. U nastavku ću pobliže opisati kod koji upravlja okretanjem i pucanjem neprijatelja.

6.4.1. Okretanje

Za implementiranje okretanja sam isprobavao razne načine no nikako nisam mogao dobiti željene rezultate. Ideja je bila da je neprijatelj u početku okrenut onako kako je postavljen na nivo u Unity-u, zatim kada igrač uđe u područje koje je postavljeno kao okidač da je agent blizu tek onda se provjerava je li neprijatelj okrenut prema igraču. Na kraju sam uspio napisati jednostavno i efikasno rješenje:

```
56    if (AgentIsClose && !enemyAnimator.GetBool("Dead"))
57        {
58            if (this.transform.position.x < target.transform.position.x
                  && this.transform.eulerAngles.y == 180f)
59            {
60                this.transform.Rotate(0f, 180f, 0f);
61            }
62            if (this.transform.position.x > target.transform.position.x
                  && this.transform.eulerAngles.y < 1f)
63            {
64                this.transform.Rotate(0f, 180f, 0f);
65            }
66        }
```

Prikazani isječak koda nalazi se u skripti neprijatelja i koristi iste funkcije `OnTriggerEnter2D` i `OnTriggerExit2D` za izmjenu stanja varijable `AgentIsClose` kao i implementacija za uzimanje sata s neprijatelja. Blok koda za rotiranje se izvršava ako je agent blizu i ako neprijatelj nije mrtav što sprječava rotiranje neprijatelja ako se izvršila animacija umiranja. U linijama 58 i 62 provjerava se s koje strane je agent u odnosu na neprijatelja i kako je neprijatelj okrenut, ako se te dvije stvari ne poklapaju tada se izvršava jednostavno rotiranje objekta oko y osi.

6.4.2. Pucanje

Pucanje neprijatelja koristi isti kod kao i pucanje kod agenta, razlika je naravno u tome što igrač ne upravlja sa ispucavanjem metka već je postavljen tajmer koji upravlja pucanjem. Funkcija `Shoot()` gotovo je ista kao u skripti koja se nalazi na objektu agenta, razlika je jedino u zvuku koji puška proizvodi. Dio koda koji upravlja pozivanjem funkcije `Shoot()` nalazi se unutar `Update()` funkcije i izgleda ovako:

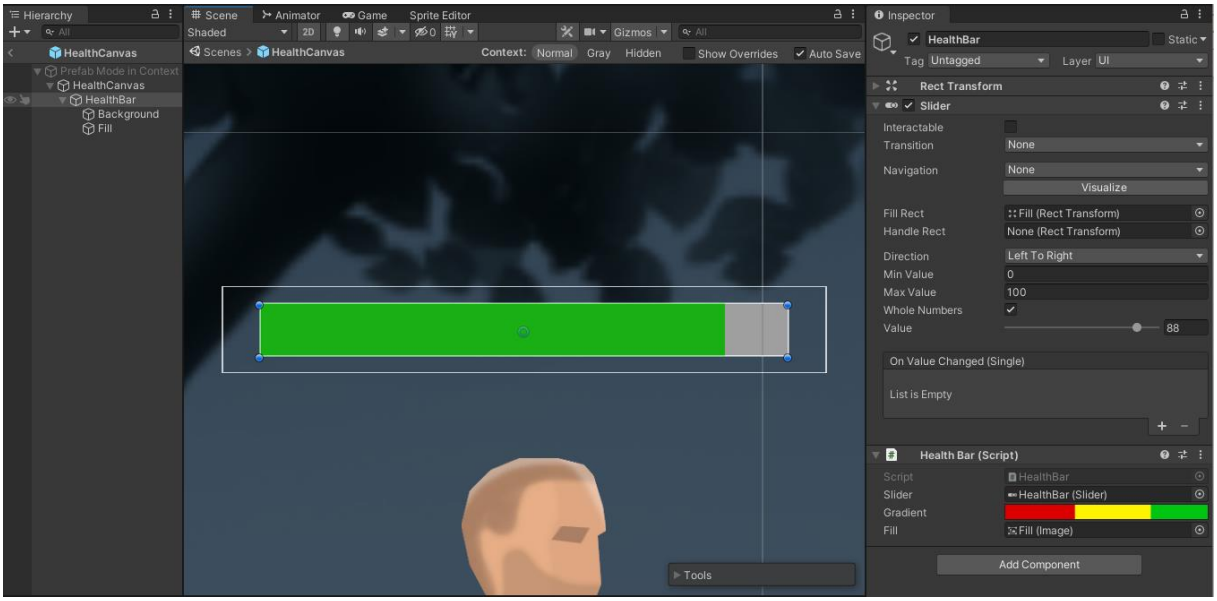
```
38          time += Time.deltaTime;
39          if(time >= timeBetweenShots)
40          {
41              time = 0;
42              if (AgentIsClose && !enemyAnimator.GetBool("Dead"))
43              {
44                  Shoot();
45              }
46          }
```

Varijabla `time` je privatna varijabla tipa *float* te se uvećava za vrijeme koje prolazi u sekundama, ako je vrijednost varijable jednaka ili veća zadanoj vrijednost to znači da je vrijeme da neprijatelj ispali metak pa se `time` resetira na nulu i ukoliko je neprijatelj živ i agent blizu, poziva se funkcija `Shoot()` koja stvara kopiju metka. Metak koji ispucava neprijatelj je po dizajnu drugačiji od metka agenta ali skripta koja njime upravlja je gotovo jednaka uz naravno prikladno zamijenjene objekte tako da neprijatelj radi štetu agentu. Jedna od glavnih promjena u skripti metka je da pronalazi sve objekte koji imaju oznaku (*eng. Tag*) „Enemy“ i ignorira sudar sa drugim neprijateljima ili njihovim okidačima za provjeru je li agent u blizini što je riješilo problem da neprijatelji mogu pucati jedni druge ili da se metak uništi u dodiru s okidačem.

Slika 8: Dizajn puške, bljeska i metka za pušku

6.5. Upravljanje životom likova

Iznad svakog lika u igri nalazi se *HeathBar*, vizualna reprezentacija stanja lika koja prikazuje koliko štete je primio i koliko još može primiti prije nego umre. *Healthbar* je izrađen pomoću elemenata korisničkog sučelja (*eng. User Interface*), koristi se pojednostavljen klizač (*eng. Slider*) koji sadrži dvije slike (*eng. Image*) od kojih je jedna postavljena na solidnu sivu boju i koristi se kao pozadina klizača a druga koristi gradijent kako bi se ovisno o vrijednosti klizača mijenjala boja ispune.



Slika 9: *HealthBar* objekt, hijerarhija, izgled i komponente

Za upravljanje *HealthBar* objektom izrađena je skripta kao što je vidljivo pod komponentama na slici 6. Skripta sadrži dvije vrlo jednostavne funkcije koje se kasnije koriste za upravljanje vrijednostima klizača za ispravno prikazivanje i pomicanje.

```
6 public class HealthBar : MonoBehaviour
7 {
8     public Slider slider;
9     public Gradient gradient;
10    public Image fill;
11    public void SetMaxHealth(int health)
12    {
13        slider.maxValue = health;
14    }
15
16    public void SetHealth(int health)
17    {
18        slider.value = health;
19        fill.color = gradient.Evaluate(slider.normalizedValue);
20    }
21}
```

Nakon implementiranog objekta *Healthbar*, on se može odvući iz hijerarhije u mapu projekta kako bi se stvorio *Prefab*, *Prefab* predstavlja objekt koji smo stvorili zajedno sa svim komponentama i hijerarhijom kako bi se lakše prenio na ostale likove [9]. Nakon što je objekt dodan na agenta i pravilno referenciran mogu se koristiti prethodno izrađene funkcije na sljedeći način:

```
20      health = PlayerPrefs.GetInt("health");
21      healthbarTr = healthBar.transform;
22      healthBar.SetMaxHealth(maxHealth);
23      healthBar.SetHealth(health);
```

Prikazani isječak koda (linije 20-23) nalaze se u `Start()` funkciji agenta, `PlayerPrefs` je klasa koja je koristi kako bi se željene vrijednosti spremile na računalo tokom igranja što je vrlo korisno za prijenos podataka između scena jer ne bi bilo ispravo da se prilikom mijenjanja nivoa vrijednost *HealthBar* klizača resetira na početnu već da zadrži vrijednost iz prethodnog nivoa [10]. Za razliku od implementacije unutar skripte agenta, u `Start()` funkciji skripte neprijatelja nalazi se sljedeći blok naredbi:

```
26      if (this.name.Equals("Boss"))
27      {
28          maxHealth = 200;
29          health = 200;
30      }
31      healthBar.SetMaxHealth(maxHealth);
```

Razlika je u tome što nije potrebno koristiti `PlayerPrefs` klasu i budući da neprijatelj i glavni neprijatelj imaju istu skriptu na sebi, provjerava se ako se radi o glavnom neprijatelju kako bi se njemu dodijelile veće vrijednosti da ga bude teže ubiti.

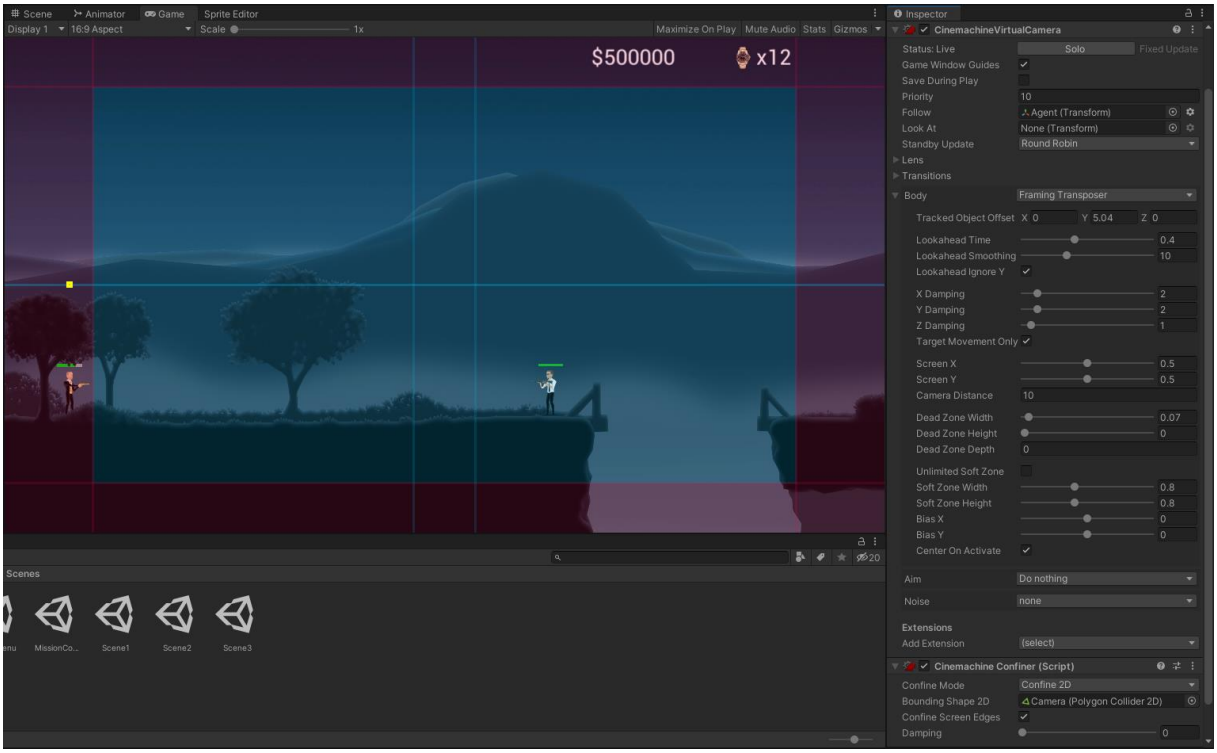
Za kraj je potrebno kod agenta i neprijatelja stvoriti funkciju koja će se pozivati kod dodira metka sa likom kako bi se napravila šteta i smanjila vrijednost *HealthBar* elementa.

```
55      public void TakeDamage(int damage)
56      {
57          health -= damage;
58          healthBar.SetHealth(health);
59          PlayerPrefs.SetInt("health", health);
60          if (health <= 0)
61          {
62              Die();
63          }
64      }
```

Prikazana funkcija ista je kod agenta i neprijatelja jedino što kod neprijatelja nije potrebna linija 59. Ako je varijabla `health` jednaka ili manja od nule tada se izvršava linija 62, poziv funkcije `Die()` koja upravlja animacijom i kod agenta aktivira korisničko sučelje.

7. Kamera

Za kontroliranje kamere sam odlučio koristiti Unity-ev paket *Cinemachine* koji se može lako i brzo dodati unutar Unity-a. *Cinemachine* paket dodaje novu karticu u sučelje alata preko koje se mogu dodati razne vrste kamera od kojih je za ovu igra potrebna 2D kamera. Dodana *Cinemachine* kamera postoji u sceni zajedno sa već postojećom kamerom te zapravo samo kontrolira ponašanje glavne kamere. Prvo je potrebno povući iz hijerarhije scene objekt agenta na mjesto u postavkama kamere „Follow“ kako bi kamera pratila agenta. Na slici 10 je prikazana pokrenuta igra zajedno sa postavkama u kojima se može vrlo detaljno podesiti željeno kretanje kamere u odnosu na poziciju praćenog objekta.



Slika 10: Postavke *Cinemachine* kamere

Prvo sam spustio agenta niže u kadru, zatim je postavljeno da kamera kod kretanja agenta ubrza i krene ispred njega kako bi se igraču bolje prikazalo što se nalazi ispred njega. Dvije okomite linije vidljive na sredini kadra označuju mjesto na x osi gdje se kamera neće micati ako je agent unutar oznaka. Posljednje što je potrebno napraviti je kreirati novi objekt u sceni i na njega dodati *Polygon Collider 2D* komponentu te ju oblikovati po rubu cijele scene. Ta komponenta će se koristiti kao granica koju kamera nikad neće prijeći tako da odvučemo objekt na mjesto „Bounding Shape 2D“ postavke.

8. Korisničko sučelje igre

Iako je srž igre uspješno implementirana, kako bi igrač mogao upravljati igrom a ne samo agentom, potrebno je intuitivno, efikasno i jednostavno korisničko sučelje. U ovom poglavlju opisat ću izradu i funkcionalnosti različitih elemenata korisničkog sučelja, neki dijelovi poput glavnom i završnog menija su napravljeni u posebnim scenama koje se nalaze prije i poslije samih nivoa, dok su sučelja poput menija pauze, smrti i konstantnog vođenja rezultata implementirana unutar nivoa te se prikazuju uvijek ili se pozivaju putem koda. Navedeni dijelovi korisničkog sučelja koji sadrže slične opcije u meniju koriste zajedničku skriptu koja sadrži funkcije svih menija zajedno pa se lagano može kod pritiska gumba pozvati pripadajuća funkcija te se mogu iste funkcije iskoristiti više puta. Za izradu korisničkog sučelja dodao sam u projekt besplatan Unity-ev paket, *TextMeshPro*, koji omogućuje detaljnije uređivanje teksta od ugrađenih opcija. Kod izrade skripte je važno u ovom slučaju uključiti dvije biblioteke, *UnityEngine.UI* i *UnityEngine.SceneManagement*. U nastavku ću opisati početni dio skripte a zatim postepeno dijelove skripte vezane uz pojedine menije koji koriste ovu skriptu.

```
4 using UnityEngine.SceneManagement;
5 using UnityEngine.UI;
6
7 public class MainMenu : MonoBehaviour
8 {
9     static bool Paused = false;
10    public GameObject pauseMenu;
11    public Slider musicVolumeSlider;
12    public Slider effectsVolumeSlider;
```

Na početku su uključene spomenute biblioteke te se definirani potrebni parametri, iako su parametri javnog svojstva, ukoliko provjeravamo postoje li u određenoj instanci skripte, možemo koristiti istu skriptu za više menija.

8.1. Glavni meni i opcije

Glavni meni i meni opcija nalaze se u sceni sa indexom 0 koja se pokreće pri pokretanju igre. Na ova dva menija je vidljiv logo igre te animirani agent što je po mojem mišljenju podiglo izgled i kvalitetu cijele igre. Kao pozadinu menija sam stavio zamućenu sliku prvog nivoa tako da izgleda kao se meniji nalaze iznad.

Slika 11: Glavni meni i opcije

Na Slici 11 su prikazana oba menija i vidljivo je da glavni meni nudi mogućnosti za pokretanje igre, otvaranje menija opcije te izlazak iz cijele igre dok u meniju opcija igrač može podesiti glasnoću pozadinske glazbe i ostalih zvukova uz pomoć dva klizača. Dio koda korišten u prikazana dva menija izgleda ovako:

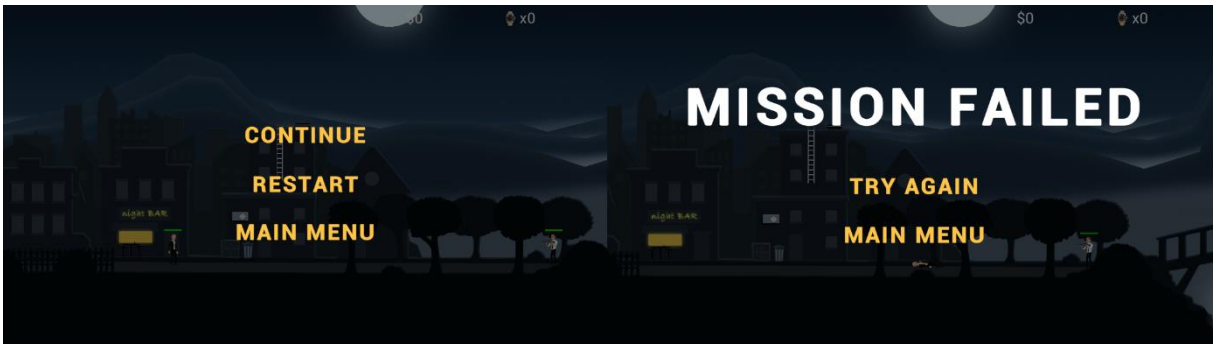
```
14  private void Start()
15      {
16          Time.timeScale = 1f;
17          if (musicVolumeSlider != null && effectsVolumeSlider != null)
18          {
19              musicVolumeSlider.value = PlayerPrefs.
                    GetFloat("MusicVolume");
20              effectsVolumeSlider.value = PlayerPrefs.
                    GetFloat("EffectsVolume");
21          }
22      }
23      public void Play()
24      {
25          PlayerPrefs.SetInt("playing", 1);
26          SceneManager.LoadScene(1);
27      }
28
29      public void Quit()
30      {
31          Application.Quit();
32      }
33
34      public void BackToMainMenu()
35      {
36          Time.timeScale = 1f;
37          SceneManager.LoadScene(0);
```

```
38      }
39
40      public void ChangeMusicVolume()
41      {
42          PlayerPrefs.SetFloat("MusicVolume", musicVolumeSlider.value);
43      }
44
45      public void ChangeEffectsVolume()
46      {
47          PlayerPrefs.SetFloat("EffectsVolume", effectsVolumeSlider.value);
48      }
```

U `Start()` funkciji se postavljaju vrijednosti klizača na pravilne vrijednosti ukoliko se igrač nalazi u glavnom izborniku, funkcije u redovima 23-34 pozivaju se kod pritiska na pripadajući gumb, gumb i funkcija povezuju u opcijama guma tako da se funkcija doda u *On Click()* događaj. Funkcije u linijama 40-48 se na isti način dodaju u događaj pripadnog klizača označen sa *On Value Changed (Single).*

8.2. Meni pauze i smrti

Tijekom igranja igre moguće je prikazati dva menija, meni pauze i smrti. Oba izbornika su dizajnirana tako da iskoriste trenutni izgled igre i dodaju zatamnjenje kako bi se naglasilo da je otvoren meni i da fokus više nije na igri. Meni pauze se otvara pritiskom na tipku Escape (Esc), a meni smrti se otvara automatski kada igrač ne može primati više štete i umre.



Slika 12: Meni pauze i smrti

Prikazani meniji koriste prethodno definirane funkcije uz nekoliko dodataka:

```
50  private void Update()
51      {
52          if(Input.GetKeyDown(KeyCode.Escape) && pauseMenu != null)
53          {
54              if (Paused)
55              {
56                  Continue();
57              }
58              else
```

```
59              {
60                  Pause();
61              }
62          }
63      }
64
65      public void Continue()
66      {
67          pauseMenu.SetActive(false);
68          Time.timeScale = 1f;
69          Paused = false;
70      }
71
72      void Pause()
73      {
74          pauseMenu.SetActive(true);
75          Time.timeScale = 0f;
76          Paused = true;
77      }
78  }
```

Gumbi „Restart“ i „Try Again“ koriste prethodno definiranu funkciju `Play()` dok tipke „Main menu“ na oba menija koriste `BackToMainMenu()` funkciju. Dodatne funkcije prikazane u priloženom kodu služe za pauziranje i nastavljanje igre koristeći `Time.timeScale` narebnu kao što je vidljivo u linijama 68 i 75.

8.3. Praćenje rezultata i završni meni

Tijekom igranja igre igrač sakuplja novac i satove, kako bi igrač vidio brojač cijelo vrijeme u gornji desni kut sam dodao sučelje koje je ažurira svaki put kada se nešto pokupi. Kako bi igrač završio igru na kraju trećeg nivoa se nalazi okidač jednak onima na prvom i drugom nivou, no ovaj ne učitava sljedeći nivo nego sljedeću scenu koja sadrži završi meni gdje je prikazan konačni rezultat i gumb za vraćanje na glavni meni.

Slika 13: Praćenje rezultata i završni meni

Praćenje rezultata je implementirano tako da se svaki broj nalazi u posebnom objektu koji sadrži skriptu sa funkcijom za ažuriranje prikaza koja se poziva kod interakcije s pripadnim objektom za skupljanje. Brojač novca i brojač satova su gotovo jednake skripte tako da ću objasniti koncept na skripti za brojanje novca:

```
4 public class moneyCounter : MonoBehaviour
5 {
6     public Text counterText;
7     private int count = 0;
8
9     private void Start()
10    {
11        count = PlayerPrefs.GetInt("moneyCounter");
12    }
13    void Update()
14    {
15        counterText.text = "$" + count.ToString();
16    }
17
18    public void AddToCount(int  money)
19    {
20        count += money;
21        PlayerPrefs.SetInt("moneyCounter", count);
22    }
23}
```

Ovo je vrlo jednostavna skripta koja sadrži varijablu sa pripadnom količinom koja se povećava uz pomoć funkcije i pretvara u tekst za prikaz u igri. Kod brojača se koristi `PlayerPrefs` klasa kako bi se rezultat prenosio iz nivoa u nivo te na završni meni.

Kod završnog menija nalaze se dvije skripte, za gumb koji vraća igrača na početni meni koristi se skripta objašnjena u prethodnim poglavljima kako bi se ponovno iskoristila `BackToMainMenu()` funkcija, a za prikaz konačnog rezultata kreirana je nova skripta koja sadrži ovaj jednostavan kod:

```
7 public class FinalScore : MonoBehaviour
8 {
9     public GameObject moneyText;
10    public GameObject watchText;
11    private int mCount;
12    private int wCount;
13    void Start()
14    {
15        SoundManager.PlaySound("Completed");
16        mCount = PlayerPrefs.GetInt("moneyCounter");
17        wCount = PlayerPrefs.GetInt("watchCounter");
18        moneyText.GetComponent<TextMeshProUGUI>().text = "$" + mCount +
            "/$500000".ToString();
19        watchText.GetComponent<TextMeshProUGUI>().text = wCount +
            "/12".ToString();
20    }
21}
```

9. Pozadinska glazba i zvučni efekti

Posljednji no vrlo bitan element koji sam dodao u ovu igru je zvuk. Pozadinska glazba može odrediti atmosferu cijele igre dok neki zvučni efekti pomažu kod osjećaja realizma i povezanosti sa svijetom igre. Smatram da je bitno staviti zvučne efekte primarno kod radnji koje obavlja igrač kao u ovom slučaju skupljanje novca i satova kako ne bi bilo potrebno svaki puta pogledati brojač u gornjem desnog kutu već zvuk indicira da je radnja uspješno obavljena. Osim pozadinske glazbe u ovaj projekt sam dodao zvukove pucanja puške i pištolja, otvaranja kovčega i uzimanja novca, zvuk padanja tijela na tlo kod umiranja, zveckanja kod uzimanja sata s mrtvog neprijatelja, zvuk koraka kod trčanja agenta te obavijesti dobivanja poruke i završavanja cijele misije. Sve zvučne datoteke korištene u projektu sam preuzeo besplatno sa različitih online izvora. Za početak rada sa audio datotekama, potrebno je kreirati novu mapu u projektu i nazvati ju „Resources“, u toj mapi će se nalaziti sve zvučne datoteke. Korištenje tako imenovane mape omogućuje lako referenciranje datoteka u skripti [10]. Za zvuk u igrici sam kreirao dvije različite skripte, jedna je specifična samo za pozadinsku dok druga sadrži kod za upravljanje ostalim zvučnim efektima.

Razlog zašto nisu svi zvukovi kontrolirani iz iste skripte je zato što jedna skripta kontrolira jedan izvor zvuka (*eng. Audio Source*) koji je postavljen na novi prazan objekt koji će sadržati skriptu i taj objekt će biti postavljen na svaku scenu gdje je zvuk potreban, druga komponenta kod implementacije zvuka u igru je „slušatelj zvuka“ (*eng. Audio Listener*) koji se najčešće kod 2D platformera postavlja na kameru kako bi zvuk uvijek dolazio „iz sredine ekrana“. Kao što je prethodno prikazano u opcijama igre postoji mogućnost zasebnog podešavanja glasnoće pozadinske glazbe i zvučnih efekata, što znači da će jedna skripta kontrolirati izvor zvuka koji će sadržavati samo pozadinsku glazbu, a druga će kontrolirati drugi izvor zvuka na različitom objektu koji će moći puštati više različitih zvukova pomoću jednostavne funkcije. Prvo ću objasniti kako funkcionira upravljanje pozadinskom glazbom:

```
3 public class MusicManager : MonoBehaviour
4 {
5     private static MusicManager managerInstance;
6     public AudioSource audioSource;
7     private void Awake()
8     {
9         if(!PlayerPrefs.HasKey("MusicVolume"))
10        {
11            PlayerPrefs.SetFloat("MusicVolume", 0.1f);
12            PlayerPrefs.SetFloat("EffectsVolume", 0.2f);
13        }
14        DontDestroyOnLoad(this);
15
```

```
16          if(managerInstance == null)
17          {
18            managerInstance = this;
19          }
20          else
21          {
22              Destroy(gameObject);
23          }
24      }
25
26      private void Update()
27      {
28          audioSource.volume = PlayerPrefs.GetFloat("MusicVolume");
29      }
30  }
```

Priloženi kod skripte postavlja glasnoću na definiranu vrijednost ukoliko ne postoji drugi podatak o glasnoći spremljen pomoću `PlayerPrefs` klase, također se kod promjene klizača u postavkama ažurira spremljena vrijednost kako bi postavljene postavke zvuka ostale zapamćene između sesija igranja. Sljedeća bitna stvar u ovoj skripti je blok koda u linijama 14-23 koji omogućuje da se objekt koji pošta glazbu i sadrži ovu skriptu postavlja samo jednom i to u glavni izbornik te se prilikom pokretanja igre ne uništava već nastavlja svirati glazbu bez prestanka, kako prilikom vraćanja u glavni izbornik ne bi nastao još jedan izvor zvuka, novi objekt se uništava (linija 22).

Druga skripta se naziva *SoundManager* i sadrži vrlo jednostavno referenciranje datoteka iz mape „Resources“ te ih spremna u varijable tipa *AudioClip*. Funkcija `PlaySound (string clip)` sadrži *switch-case* grananje u kojem se na temelju unesenog naziva datoteke pokreće pripadni zvuk. Prikaz isječka funkcije:

```
28      public static void PlaySound (string clip)
29      {
30          switch (clip)
31          {
32              case "Pistol":
33                  audioSource.PlayOneShot(pistolFire);
34                  break;
. . .
56              case "Completed":
57                  audioSource.PlayOneShot(completed);
58                  break;
59          }
60      }
```

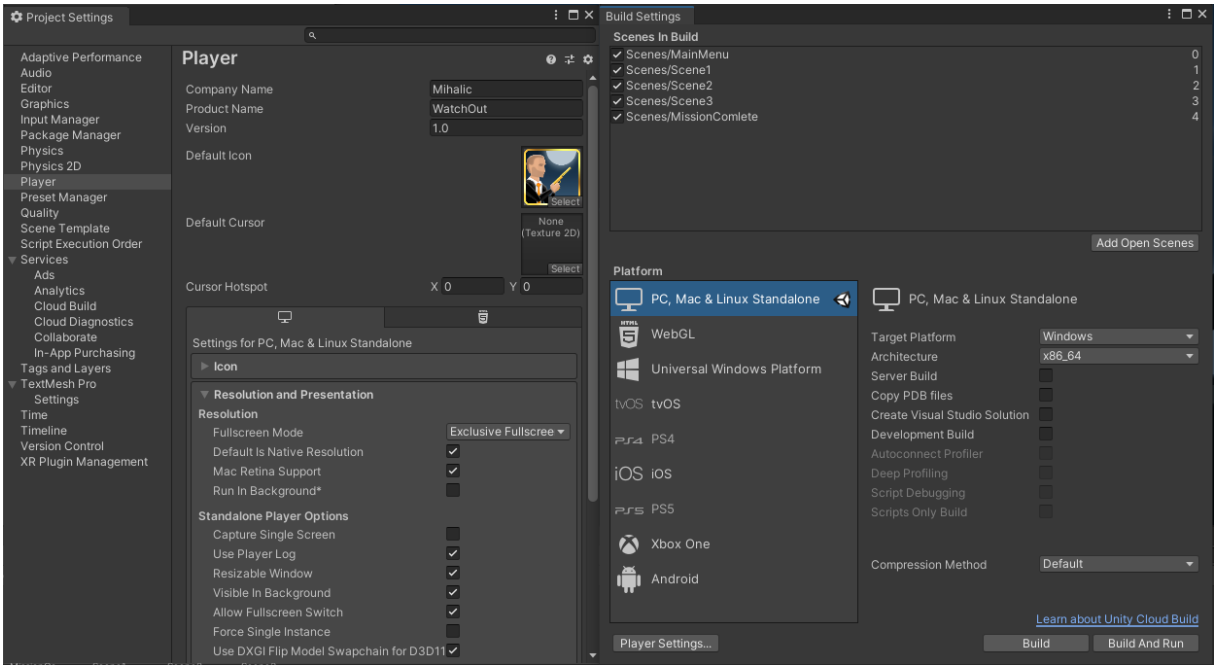
Objekt sa skriptom *SoundManager* se nalazi svakom nivou, kada je potrebno svirati određen zvuk, skripta se direktno referencira te se pozove prikazana funkcija sa ispravnim unosom argumenta. Za zvukove poput koraka u skripti objekta koji se kreće kreirana je jednostavna funkcija koja poziva `PlaySound (string clip)` funkciju te se ista može pozvati na željenom mjestu u animaciji dodavanjem događaja (*eng. Event*) u vremensku traku animacije.

10. Izrada izvršne datoteke

Posljednji korak u izradi igre u Unity-u je izrada izvršne datoteke. Prije pritiska na tipku „Build“ potrebno je u postavkama na putanji „Project Settings > Player“ upisati i odrediti osnovne podatke i postavke igre. Stavio sam da ime igre bude „WatchOut“ i dodao ikonu igre koju sam kreirao u Photoshop-u.

Slika 14: Ikona igre

Unity nakon dodavanja ikone generira različite veličine koje će biti potrebne. Posljednja stvar koju sam izmjeno u ovom prozoru su *Fullscreen Mode, Resizable Windows* i *Allow Fullscreen Switch.* Testirao sam s raznim kombinacijama no postavke prikazane na slici 15 omogućuju da se igra pokrene preko cijelog zaslona a ako igrač želi može se sa „Alt + Enter“ komandom prebaciti u pogled prozora kojeg zatim može podesiti po želji. Na kraju je potrebno u *Build Settings* prozoru dodati sve scene, odabrati željeni platformu i kliknuti „Build“.



Slika 15: *Project Settings* i *Build Settings* prozori

11. Zaključak

Ovim radom sam obradio osnove 2D platformera kao žanra igre, prisjetio se starih klasičnih igara koje su dominirale tržištem te upoznao sa novijim naslovima koji održavaju popularnost žanra. U praktičkom dijelu sam izradio vlastitu igru u Unity programskom alatu koristeći dodatne alate poput Adobe Photoshop-a za kreiranje svih slika korištenih u igri te Visual Studio za pisanje mnogih skripti potrebnih za uspješnu izradu igre.

Postupak izrade vlastitog 2D platformera je bio vrlo zanimljiv, budući da mi je ova igra bila prvi projekt u Unity-u puno toga sam naučio pokušavajući realizirati isplaniranu ideju. Na početku sam odlučio da ću koristiti vlastite slike za sve elemente u igri i iako nije bilo lako sretan sam što sam odabrao taj način rada jer sam se upoznao sa dijelova i funkcionalnostima Unity-a koje vjerojatno ne bih koristio da sam uvezao gotove dijelove poput likova i okoline. Kreiranjem likova od nule upoznao sam se sa stvaranjem kostura i kreiranjem različitih animacija. Programski alat Unity me pozitivno iznenadio jer sam prije izrade ovog rada smatrao da je to vrlo kompleksan alat sa kompliciranjem sučeljem no uz malo prakse shvatio sam princip rada te koliko je zapravo sam rad prilagođen svim razinama znanja. Pisanje skripti je bilo intuitivno jedino je u početku potrebno naučiti osnove funkcioniranja objekata i različitih tipova podataka koji se mogu pojaviti u Unity-u. Prijašnje znanje C# jezika uvelike je pridonijelo pisanju skripti te je pravilno postavljanje radne okoline u Visual Studio-u ključ za vrlo lako i brzo implementiranje željenih funkcionalnosti.

Izradom ovog rada smatram da sam ostvario svoje ciljeve te iskoristio svoje umjetničke sposobnosti i znanje programiranja u C# jeziku kako bi uspješno izradio svoju prvu igru sa osnovnim elementima 2D platformera. U procesu izrade sam naučio koristiti programski alat Unity te smatram da sam stekao mnogo znanja o kreiranju igara što je nešto s čim se prije nisam susreo. Zadovoljan sam konačnim rezultatom rada te smatran da Unity jedan od najkvalitetnijih alatu za izradu igara.

Popis literature

[1] J. Haas, „A History of the Unity Game Engine,“ Electronic Projects Collection, ožujak 2014. [Na internetu]. Dostupno: https://web.wpi.edu/Pubs/E-project/Available/E-project-030614-143124/unrestricted/Haas_IQP_Final.pdf [pristupano 30.06.2021.].

[2] MVC Staff, „United they stand“, MCV Development News, prosinac 2009. [Na internetu]. Dostupno: https://www.mcvuk.com/development-news/united-they-stand/ [pristupano 30.06.2021.].

[3] Arnia software, „What Makes Unity So Popular in Game Development?“, 10.08.2020. [Na internetu]. Dostupno: https://www.arnia.com/what-makes-unity-so-popular-in-game-development/ [pristupano 30.6.2021.]

[4] Microsoft, „Visual Studio Community“, (bez dat.) [Na internetu]. Dostupno: https://visualstudio.microsoft.com/vs/community/ [pristupano 30.6.2021.]

[5] Techopedia, „Adobe Photoshop“, (bez dat.) [Na internetu]. Dostupno: https://www.techopedia.com/definition/32364/adobe-photoshop [pristupano 30.6.2021.]

[6] Nodwin Gaming, „The evolution of platform games in 9 steps“, 23.03.2017. [Na internetu]. Dostupno: https://www.redbull.com/in-en/evolution-of-platformers [pristupano: 01.07.2021.]

[7] H. Kantilaftis, „Genre Revival: 2D Platformers“, studeni 2014. [Na internetu]. Dostupno: https://www.nyfa.edu/student-resources/genre-revival-2d-platformers/ [pristupano: 01.07.2021.]

[8] A. Touch, „2D Tilemap Asset Workflow: From Image to Level“, 25.01.2018. [Blog post]. Dostupno: https://blog.unity.com/technology/2d-tilemap-asset-workflow-from-image-to-level [pristupano: 01.07.2021.]

[9] Unity Documentaion, „Unity User Manual 2020.3 (LTS)“, 23.08.2021. [Na internetu]. Dostupno: https://docs.unity3d.com/Manual/index.html [pristupano: 24.08.2021.]

[10] Unity Documentaion, „Welcome to the Unity Scripting Reference!“, 23.08.2021. [Na internetu]. Dostupno: https://docs.unity3d.com/ScriptReference/ [pristupano: 24.08.2021.]

Popis slika