Izrada videoigre uloga u programskom alatu Unity

Oroz, Marin

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Marin Oroz

Izrada videoigre uloga u programskom alatu Unity

ZAVRŠNI RAD

Varaždin, 2022.

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Marin Oroz

Matični broj: 0016142597

Studij: Informacijski sustavi

Izrada videoigre uloga u programskom alatu Unity

ZAVRŠNI RAD

Mentor:

Doc. dr. sc. Mladen Konecki

Varaždin, listopad 2022.

Marin Oroz

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

Ovaj rad se bavi detaljnim prolaskom kroz izradu igre uloga u programskom alatu Unity. Velika većina rada se temelji na samoj izradi igre u programskim alatima te na načinima kojim sam došao do konačnog rezultata. Prvi dio rada je potkrepljen teorijom iza izrade igre. U početku objašnjavam svoje metode te tehnike rada koje sam upotrebljavao. Nakon toga prelazim na teoriju na kojoj sam bazirao svoju igru te brzo nakon toga prelazim na praktični dio ovog rada. U tom dijelu detaljno objašnjavam izradu igre uz mnogobrojne vizualne prikaze te pojašnjavanje koda kako bih što bolje predočio načine na koje sam sve napravio. Dakle, krenuo sam od samih osnova bez kojih nema igre, a to su igrač i mapa. Detaljan prikaz izrade te samog koda koji je implementiran. Nakon toga se prelazi na dijelove kao što su izrada izbornika te sistem borbe. Praktični dio je objašnjen na način da svaka cjelina vuče za sobom sljedeću te da su povezane kako bi se lakše zamislio izgled i funkcioniranje igre.

Ključne riječi: unity, igra uloga, logika, implementacija, kod, igrač, izbornik, borba, sistemska logika

Sadržaj

1. Uvod

Tema ovog završnog rada je izrada igre uloga u programskog alatu Unity, uz pomoć programa Microsoft Visual Studio. Naziv teme loše zapravo predstavlja ovaj rad jer se doima kao da je to samo izrada primjera igre. Tokom svog rada na ovoj igri, naučio sam puno toga te sam većinu vremena proveo programirajući funkcionalnosti igre te cjelokupnu logiku igranja. To je zapravo najnaporniji i najdetaljniji i osjetljiviji dio izrade videoigre pošto je to u pozadini te se zapravo ne primijeti igrajući. Od malena igram videoigre zahvaljujući svom bratu koji me uveo u taj svijet te dosad igrajući igre razmišljaš jedino o ovome što vidiš, odnosno o grafičkim aspektima igre. Logički dio videoigre kao da se podrazumijeva, no zapravo to je dio za kojim se provede najveći dio vremena pišući stotine i stotine linija koda. Sad kad sam završio sa praktičnim dijelom, igra je spremna za bilo kakvo proširivanje pošto je cijela logika implementirana te je samo potrebno primijeniti u Unityju.

Ovaj rad je značajan zbog toga što gaming industrija eksponencijalno raste u svakom aspektu. Kao primjer, mogu navesti jednu od najvećih tvrtki ovog sektora, CD Project Red, proizvođači klasika poput *The Witcher* serijala te nešto novijeg *Cyberpunk-a*. Prema istraživanjima Gamespot-a, imaju godišnji porast zaposlenika oko 20% te se prema tome može ravnati i razvoj same industrije [6]. Kod nas gaming industrija nije na lošoj nozi zbog tvrtke Croteam koja je napravila poznati serijal *Serious Sam*. Vjerujem da, ako se odlučim ići ovim putem, da će mi ovaj radi pomoću pri zapošljavanju, no i u samom radu.

2. Metode i tehnike rada

Metode i tehnike rada koje sam koristio u izradi ovog završnog rada, odnosno videoigre koja je sama po sebi i tema završnog rada, su „Unity“ i „Microsoft Visual Studio“ te programski jezik C#. Kako bih pravilno i ispravno izradio videoigru, koristio sam tečaj na web stranici „Udemy“ [1] pomoću kojeg sam uspješno napravio željenu videoigru od početka do kraja.

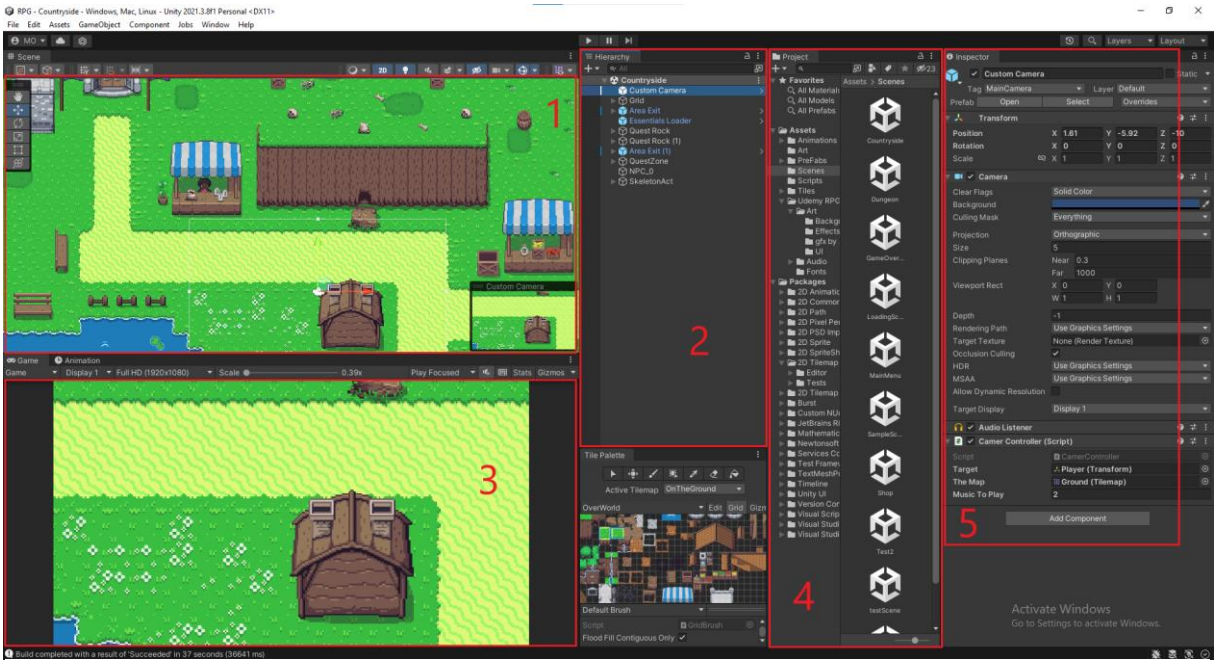
Tečaj „Learn To Create An RPG Game In Unity“ ima ukupno 18 sati video materijala kroz koje se prođu sve potrebne stavke koje su potrebne pri kreiranju videoigre. Prolaženjem kroz sate tečaja, osim što sam napravio videoigru, naučio sam razmišljati kao game developer te razvio programersku logiku potrebnu za kreiranje videoigre. Zahvaljujući predznanju s fakulteta, osobito kolegija kao što je „Programsko inženjerstvo“, kroz većinu tečaja sam prolazio s razumijevanjem te bez većih poteškoća. Međutim, u trenutcima kada se nakupi veći broj noviteta u oboje, programiranju i radu u Unityju, znalo se odužiti te mi je zato usavršavanje igre te samo finaliziranje trebalo duže vremena. Pri pisanju koda u Microsoft Visual Studiju koristio sam programski jezik C# pošto je isti bio preporučen od strane kreatora tečaja. Taj dio mi se prvotno svidio kod ovog tečaja pošto smatram da sam najvještiji upravo u C#. Tečaj je organiziran na način da se prvo prođu osnove kao što su kreiranje igrača, scena i mapa, a nakon toga izrada menija te implementacija istog u dijelove mape te za kraj veći dio je zauzeo „battle system“ koji je razrađen u detalje [1].

Nakon više od mjesec dana rada na videoigri, uz pomoć tečaja te uloženog vlastitog truda, završio sam svoj praktični dio koji sam izradio u sljedećim alatima.

2.1. Unity

„Unity (NYSE: U) is the world’s leading platform for creating and operating real-time 3D (RT3D) content. Creators, ranging from game developers to artists, architects, automotive designers, filmmakers, and others, use Unity to make their imaginations come to life.„ [2]. Kao što je i citirano, Unity je vodeća platforma za kreiranje videoigara te ostalog sličnog sadržaja te je tako većini game developera prva platforma u kojoj su kretali sa izradom videoigara. Definirao bih Unity kao platformu koja pomaže implementirati naš kod u obliku konkretnih „game objekata“ koji onda imaju svoju funkciju koja je definirana u skripti. Unity uvelike pomaže pri izradi igara sa svojom jednostavnošću te tako određene stvari koje bi se laiku činile kompleksnima, zapravo se riješe u nekoliko klikova. Pomoću ove platforme su kreirane razne videoigre (Pokemon Go, Call of Duty: Mobile, Cuphead…), no koristio se i, primjerice, u

filmskoj industriji pri izradi pozadina (The Lion King) [3]. Kako bih što bolje dočarao Unity i kako se koristi, odlučio sam priložiti sliku izgleda rasporeda unutar programa.



Slika 1. Raspored unutar programa „Unity Editor“ [autorski rad]

Na slici su crvenim brojkama označeni segmenti programa u kojima se radi:

1. **Scena** – prozor u kojem se uređuje scena u igri, dakle dodaju objekti, uređuje mapa pomoću „tileset-ova“ itd.
2. **Hierarchy** – hijerarhija dijelova, odnosno objekata u sceni. Ovdje se nalazi sve što možemo ili ne možemo vidjeti u sceni.
3. **Game** – primarno služi da možemo vidjeti što nam kamera prikazuje te kada pokrenemo igricu da možemo testirati.
4. **Project** – sadrži naš folder koji se nalazi negdje pohranjen na računalu te gdje se sve sprema. Preko toga dolazimo do drugih skripti, prefabova i ostalog.
5. **Inspector** – služi kao „properties“ tabu za sve objekte. Možemo dodavati skripte objektima te na taj način implementirati kod.

2.2. Microsoft Visual Studio

Visual Studio je program tvrtke Microsoft koji se koristi za razvoj projekata kao što su web stranice, web aplikacije, računalni programi, mobilne aplikacije i slično. Jedan je od modernijih razvojnih okruženja za programere te, primjerice, imaju „IntelliCode“, alat za automatsko ispunjavanje koda pri čemu uzima kontekst i pretpostavlja naše radnje. Microsoft je omogućio lako povezivanje s drugih platformama kao što je i Unity, no o tome ću kasnije kada dođemo do praktičnog dijela završnog rada. [4]

3. Igre uloga

Igre uloga, odnosno nama poznatiji naziv *„role-playing game“,* je kategorija igara u kojima igrač poprima ulogu određenog lika iz izmišljenog okruženja. Kroz RPG igre igrač je u ulozi jednog ili više likova s kojima prolazi kroz igru te izvršava određene misije zadane od strane developera [7]. Likovi mogu imati različite osobine, ovisno o igri, neki mogu biti dominantni i istaknuti, a drugi u pozadini te rjeđe zapaženi. Developeri će dizajnirati igru tako da igrač dobije određeni željeni dojam o njoj, odnosno o njenim likovima. Tako će u velikoj većini slučaja glavni lik igre imati sve poželjne osobine koje se igraču na prvu svide što vodi tome da igrač uživa u igranju uloge glavnog lika.

3.1. Likovi

Likovi su jedan od najvažnijih dijelova jedne RPG igre. U većini modernih RPG igara, igrač sam izgrađuje svog lika uz pomoć mehanizama unutar igre kao što su jačanje određenih *skillova,* dodjeljivanje osobnih karakteristika liku kroz odrađivanje određenih radnji. Svaki lik ima svoju pozadinu koja ga krasi, dakle tko je taj lik bio prije igre i koje su mu aspiracije tokom igre [8]. Brojne nove igre čak imaju i mogućnost razvoja svog igrača prema različitim ciljevima s čim tvoja igra može imati drugačiji završetak od nečije druge. Često pozadina jednog lika aludira na njegovu budućnost. To se postiže prolaženjem lika kroz razne izazove u kojima on mora donijeti odluke koje ga opisuju kao lika i osobu. Kroz takve izazove se zbližavamo s likom te se poistovjećujemo s njim.

3.1.1. Neprijatelji

Neprijatelji su također likovi s kojima ćemo se kada tad susresti tijekom igre. Oni čine ključan dio igre pošto igri dodaju posebno uzbuđenje i izazov. Neprijatelji guraju priču da ide dalje te likove da napreduju kroz priču.

3.2. Priča

Priča RPG igre je ono što krasi videoigru. Ovdje nema kraja kreativnosti i mašti jer je sve ostvarivo i moguće za implementirati. Svijet jedne igre može biti ogroman, što je između ostalog i trend u modernoj industriji. Kako bi igra imala dobru priču, mora imati dobru podlogu za tu priču, a to je sami svijet u kojoj se odrađuje priča. Pri kreaciji takvog svijeta treba se zapitati niz pitanja poput: Koliko kontinenata želim da svijet ima? Koliko gradova? Kakav reljef

želim da bude u mom svijetu? Tko živi u ovom svijetu? U kakvom su odnosu različiti narodi? Koliko naroda ima? [9] … Kada se pređe na priču, mora se postaviti osnovna priča oko koje se svodi cijela igra. To je određena priča vezana uz glavnog lika koju ne želimo cijelu otkriti u početku već tokom igre s ciljem da igrač malo po malo saznaje o cilju igre. Tokom igre se uvode novi likovi koji u trenutnom kontekstu nose određeno novo značenje, odnosno dodaju još jedan aspekt igri. Što igra dalje ide, igrač sa svim likovima postaje sve bliži i bliži. Zna sve više o njima jer ih gleda kako žive, kako prolaze prepreke i slično. U nastavku igre je se potrebno držati osnova priče igre kako se ne bi izgubila srž same igre. Kroz priču se moramo pitati napreduje li ova igra te dodajemo li nove vrijednosti igri preko ove priče.

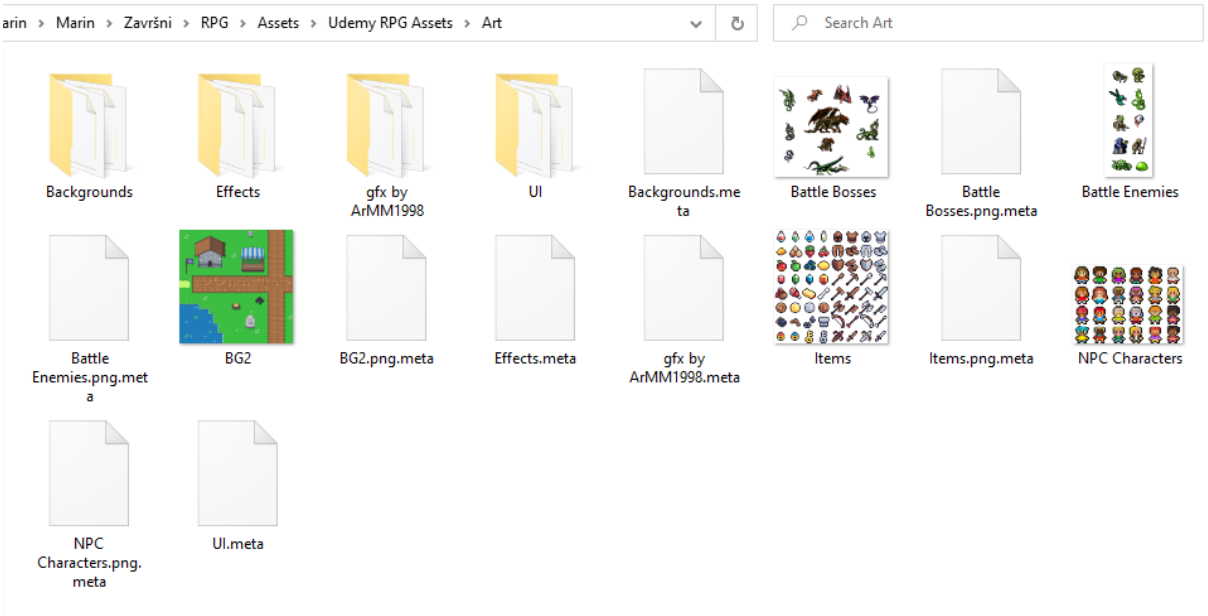
3.3. Sustav borbe

Svaka RPG igra ima određeni sustav borbe. Ne mora to biti nužno borba u kojoj dolazi do dvoboj između likova već način suprotstavljanja neprijateljima kroz koje lik napreduje. U većini modernih RPG igara likovi odrađuju misije kako bi pomogli nekome ili došli do nekog svog cilja koji mu pomaže pri napredovanju kroz igru. U tim misijama se igrač susreće s preprekama koje mora prijeći kako bi nastavio s misijom. To može biti borba s neprijateljem, određeni izazov, odabir točnog odgovora i slično. Sad ću se posvetiti sistemu borbe u igrama. U borbama likovi skupljaju iskustvo (eng. *Experience)* koji im je potreban za napredovanje lika i njegovih osobina kao što su: *Stamina, Wisdom, Strength, Defence, Magic Power, Speed, Luck* [7]… U mnogim RPG igrama mi možemo odrediti koje vještine će naš lik razvijati te s tim oblikujemo osobnost lika te se posvećujemo određenoj klasi. Postoje mnogobrojne klase koje su inače na volju developera, no neke osnovne su: The Fighter Class, The Rogue Class, The Magician Class, The Rangers Class, The Clerics Class [10]. Igrač razvija svog lika u klasi koja se njemu najviše sviđa te se tako poistovjećuje liku.

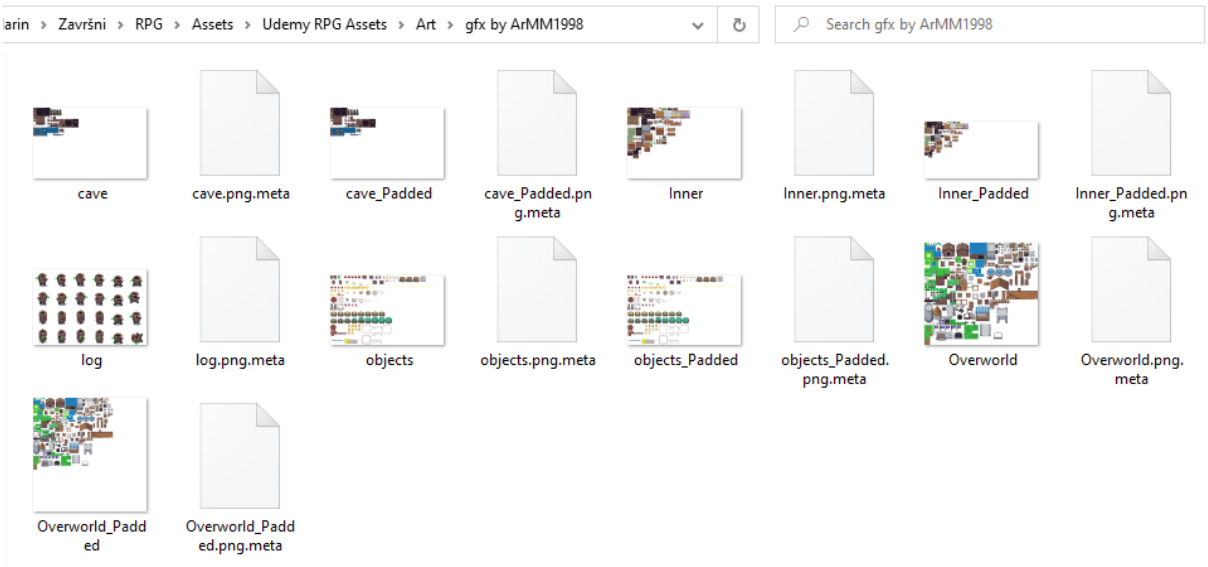
4. Izrada igre uloga u programu Unity

Nakon što sam prošao samu teoriju koja stoji iza praktičnog dijela, sljedeći je zadatak izraditi videoigru uloga (RPG) u programu Unity. Za izradu sam koristio Unity Editor verziju 2021.3.8f1.

Osnovni dijelovi jedne RPG igrice su igrač i mapa te su po tome to i prve instance koje je potrebno kreirati kako bi se imalo od čega krenuti. Za sve video aspekte igre koristio sam „*Udemy RPG Assets*“ koje sam dobio uz tečaj. Unutar *package-a* sam imao sav potreban materijal za izradu mape, igrača, NPC-ova (Non-player character), efekata i sličnog.



Slika 2. Udemy RPG Assets, Art datoteka [autorski rad]



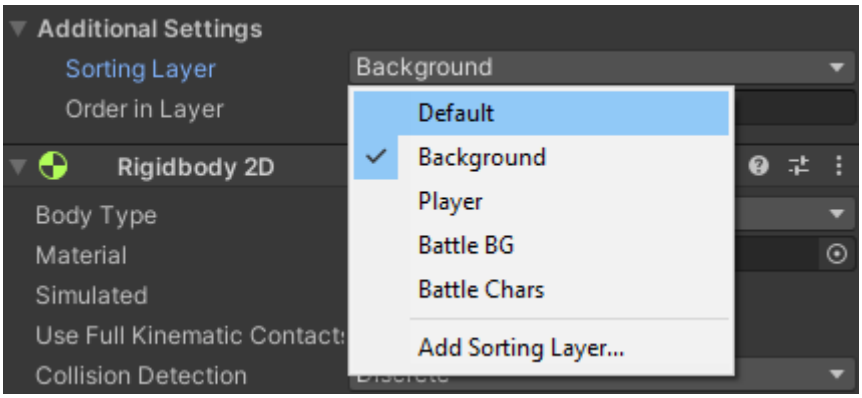
Slika 3. Udemy RPG Assets, GFX datoteka [autorski rad]

Nakon izrade igrača i mape, kreiraju se funkcionalnosti igre kao što su korištenje *item-a,* prelaženje iz scene u scenu, interakcija s NPC-evima, sistem borbe… Jedan od većih dijelova je sama izrada menija te povezivanje gumbova s njihovim funkcijama te ću tako i to malo detaljnije prijeći.

4.1. Izrada slojeva za sortiranje

Sloj za sortiranje (eng. *Sorting Layer*) služi tome da se svakom objektu u igri može dodijeliti određeni sloj kojemu pripada. Odnosno, po mom primjeru postoje slojevi (Slika 4):

- Background
- Player
- Battle BG
- Battle Chars



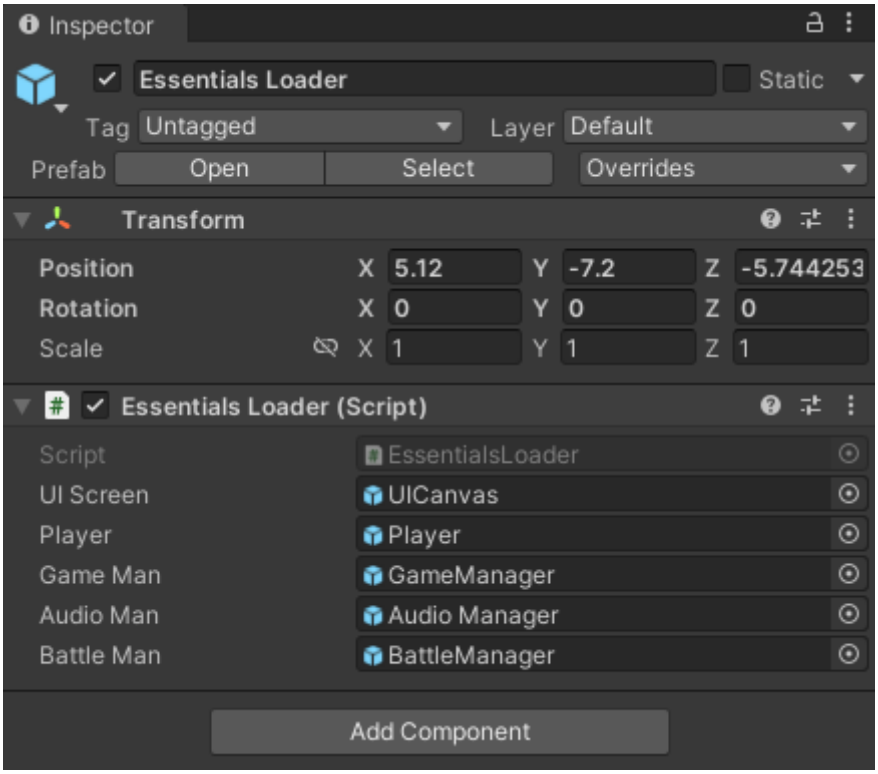
Slika 4. Slojevi za sortiranje [autorski rad]

Uloga slojeva za sortiranje je da se određeni objekti nalaze iznad drugih u igri. Dakle, po mom primjeru *Player* sloj će biti iznad *Background* sloja, *Battle BG* iznad *Player* sloja te *Battle Chars* iznad *Battle Chars*. Ovim se postiže to da se određeni dijelovi mape ne prikazuju onda kad ne trebaju, npr. tijekom borbe.

4.2. Izrada igrača

Izgled igrača te sve njegove pozicije sam uzeo iz „Udemy RPG Assets“ datoteke. Prvo što se radi je ubacivanje samog igrača u mapu s čim on postaje objekt unutar svijeta. Naknadno, igraču možemo pridodati sloj za sortiranje *Player*. Igrač se koristi u svakoj sceni te je zbog toga *prefab.* Prefab je element u igri koji nam je potreban u više scena te s toga ga se pohrani kao prefab i ubacujemo u kojoj god sceni nam je potreban. Ovim se izbjegava konstantno ponavljanje istih operacija te se posao jednog game developera puno pojednostavljuje. Prefabovi koji su mi potrebni u svakoj sceni se nalaze u „Essentials Loader“.

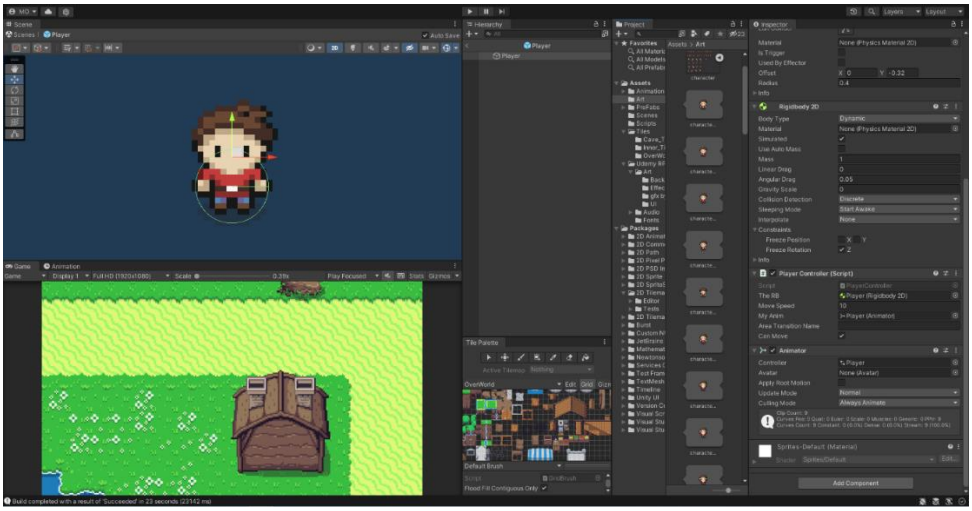
Tako se u njemu nalaze „UICanvas“, „Player“, „Game Manager“, „Audio Manager“ i „BattleManager“ o kojima ćemo kasnije. Kao što se može vidjeti na slici 4, svi imaju plavu kutiju ikonu koja označava da su prefab.



Slika 5. Essentials Loader [autorski rad]

4.2.1. Dodavanje komponenata igraču

Na slici 5 možemo vidjeti *layout* Unityja kod uređivanja igrača. U *Inspector* prozoru se mogu vidjeti karakteristika dodijeljene igraču. Prvo se dodijelio *Circle Collider 2D* koji omogućuje igraču sudaranje s ostalim dijelovima mape. Sljedeće je *Rigidbody 2D* koji daje igraču gravitaciju, odnosno ponašanje čovjeka u svijetu i svu fiziku iza toga.



Slika 6. Unity prikaz igrača [autorski rad]

4.2.1.1. Implementacija skripte „PlayerController“ za upravljanje igračem

Igraču se dodjeljuje C# skripta „PlayerController“ u kojoj se dodjeljuje igraču sve što mu je potrebno za kretnju unutar svijeta.

```
    public Rigidbody2D theRB;
    public float moveSpeed;

    public Animator myAnim;
    public static PlayerController instance;

    public string areaTransitionName;

    private Vector3 bottomLeftLimit;
    private Vector3 topRightLimit;

    public bool canMove = true;

    void Awake()
    {
        if (instance == null)
        {
            instance = this;
        }
        else if (instance != null)
        {
            Destroy(instance);
        }

        DontDestroyOnLoad(gameObject);
    }

    void Update()
    {
        if (canMove)
        {
                    theRB.velocity = new
      Vector2(Input.GetAxisRaw("Horizontal"), Input.GetAxisRaw("Vertical"))
      * moveSpeed;
        }
        else
        {
            theRB.velocity = Vector2.zero;
        }

        myAnim.SetFloat("moveX", theRB.velocity.x);
        myAnim.SetFloat("moveY", theRB.velocity.y);

       if (Input.GetAxisRaw("Horizontal") == 1 ||
      Input.GetAxisRaw("Horizontal") == -1 || Input.GetAxisRaw("Vertical")
      == 1 || Input.GetAxisRaw("Vertical") == -1)
        {
            if (canMove)
            {
                myAnim.SetFloat("lastMoveX",
Input.GetAxisRaw("Horizontal"));
                myAnim.SetFloat("lastMoveY", Input.GetAxisRaw("Vertical"));
            }
        }
```

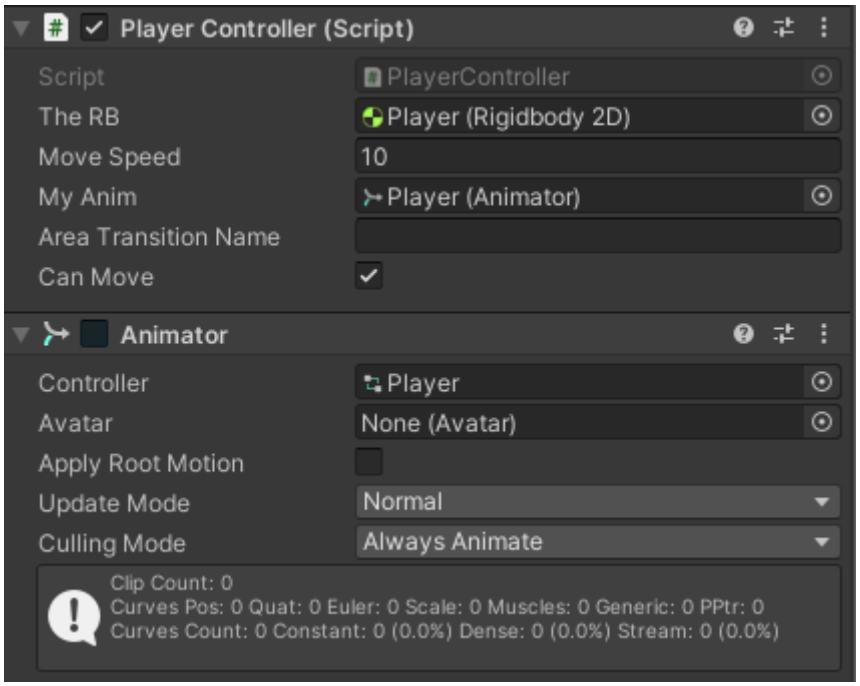
```
        transform.position = new Vector3(Mathf.Clamp(transform.position.x,
      bottomLeftLimit.x, topRightLimit.x),
      Mathf.Clamp(transform.position.y, bottomLeftLimit.y,
      topRightLimit.y), transform.position.z);

    }

    public void SetBounds(Vector3 botLeft, Vector3 topRight)
    {
        bottomLeftLimit = botLeft + new Vector3(.5f, 1f, 0f);
        topRightLimit = topRight + new Vector3(-.5f, -1f, 0f);
    }
```

Isječak koda 1. Skripta „PlayerController“ [autorski rad]

Kao što se vidi u prvom isječku koda, sve `public` varijable se mogu vidjeti u Unityju pod komponentom *PlayerController.* Dakle, `public` varijable su javne varijable koje se mogu vidjeti i koristiti izvan skripte, dok su `private` one koje se mogu jedino koristiti i vidjeti unutar skripte. Varijabla `instance` je `static` pošto postoji samo jedna instanca igrača koja se koristi u više skripti. U funkciji `Awake()` se kreira instanca igrača, u slučaju da već nije kreirana što vidimo u `if` selekciji.

Putem funkcije `Update()` se omogućuje kretnja igrača. Ako je vrijednost varijable `canMove` (mijenja se tokom igre) jednaka `true` onda se čita pritisak tipaka na tipkovnici te se množi s `moveSpeed` varijablom. *Velocity* unutar Rigidbodyja označava brzinu kretnje igrača. Varijabli `myAnim` se dodjeljuju vrijednosti *velocity* od x i y osi da se može ispravna animacija prikazati. Kasnije, kao što vidimo u isječku koda 1, se provjerava zadnji položaj igrača tako da se zna u kojoj poziciji igrač treba ostati nakon kretnje.

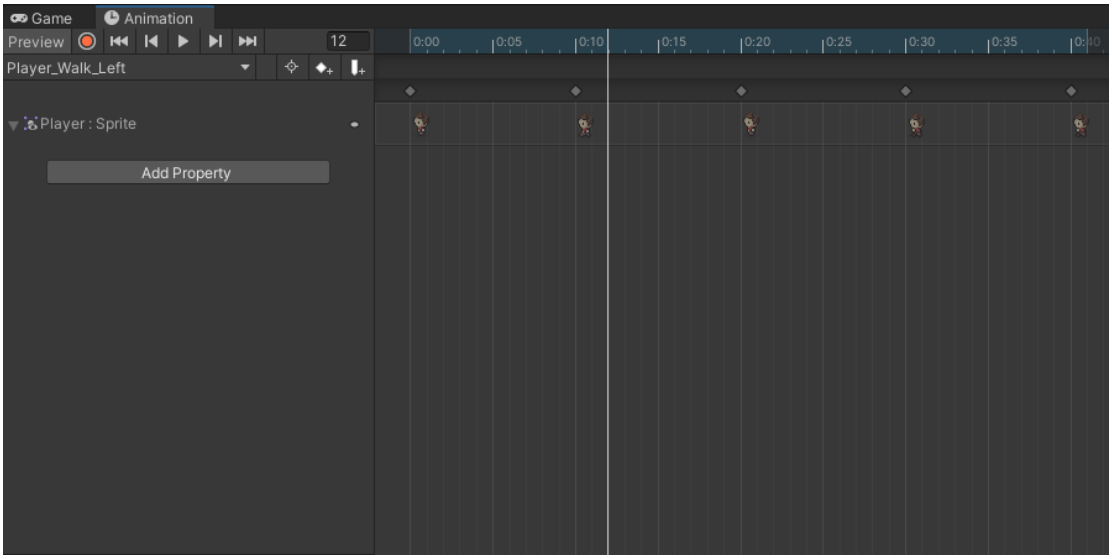
Funkcijom `SetBounds()` se pomoću varijabli `bottomLeftLimit` i `topRightLimit` postavljaju najdalji rubovi mape izvan kojih se igrač ne smije naći. Ova funkcija se poziva u skripti „CameraController“.



Slika 7. Unity Player Controller [autorski rad]

4.2.1.2. Animator

Na slici 6 se može vidjeti kako sam dodijelio varijablama određene vrijednosti, primjerice, dodijelio vlastiti Rigidbody, brzina igrača je 10 te `My Anim` koji ima vrijednost *Animator*. Igraču je dodijeljena animacija njegove kretnje koja je kreirana u animatoru. Na slici 7 je prikazan *Animator* u kojem se izrađuju sve animacije. Animacije se jednostavno rade po principu dodavanje sličica pozicije igrača po logici prvih načina izrade filmova. Dakle, na slici možemo vidjeti animaciju kretnje ulijevo gdje se svakih 10 milisekundi mijenja pozicija igrača što u konačnici izgleda kao da se igrač kreće.



Slika 8. Unity Animator [autorski rad]

4.3. Izrada scena

Scena je prostor u kojem se igrač nalazi, u kojem se kreće te obavlja svoje zadaće. Scena može biti dvodimenzionalna ili trodimenzionalna, no za moju igru sve sam radio s dvodimenzionalnim scenama. Igrač može ići iz jedne u drugu scenu te se s tim postiže određena dinamika u igri. Na osnovi primjera ću prikazati ostatak.

4.3.1. Popis scena u igri

U svojoj igri imam 7 scena od kojih su 4 s igračem, a ostale 3 bez te služe više kao tranzicijske scene. Scene s igračem:

- Countryside
- Town
- Dungeon
- Shop

Tranzicijske scene:

- Main Menu
- Game Over
- Loading Screen

Slika 9. Scena „Town“ [autorski rad]

Slika 10. Scena „Countryside“ [autorski rad]

Slika 11. Scena „Dungeon“ [autorski rad]

Slika 12. Scena „MainMenu“ [autorski rad]

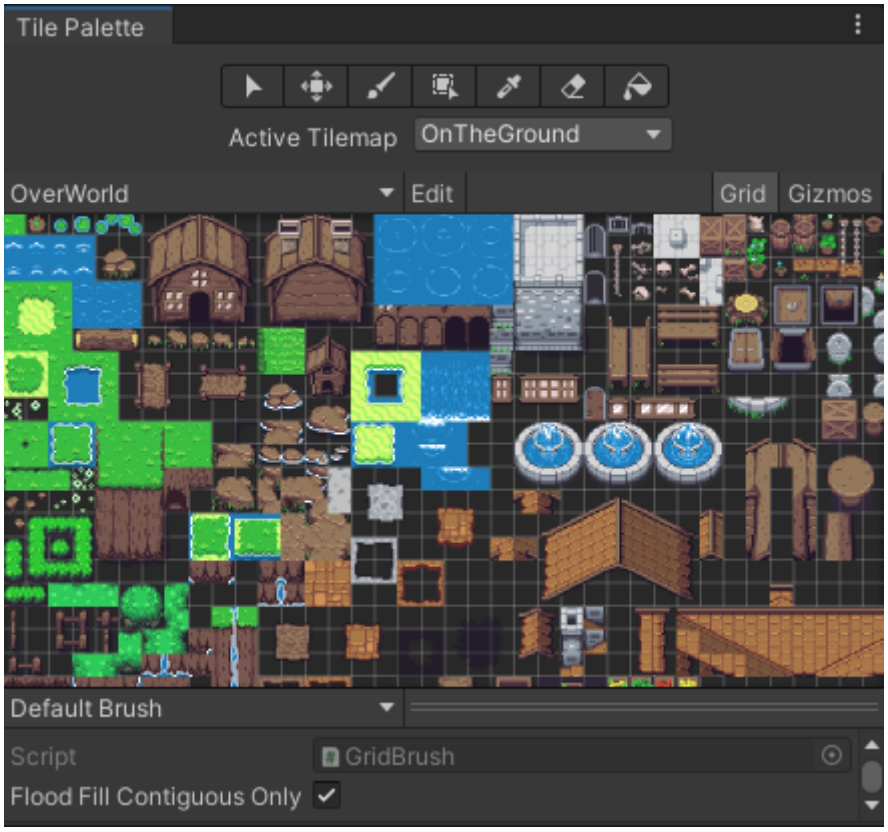
Slika 13. Scena „GameOver“ [autorski rad]

Slika 14. Scena „Shop“ [autorski rad]

Scena *LoadingScreen* je samo crna pozadina koja služi za tranziciju iz scena s igračem u scene bez igrača i suprotno. Izradu scene ću prikazivati na *Countryside* sceni.

4.3.2. Izgled scene

Scenu uređujemo pomoću *Tile Palette* što je u suštini kolekcija pločica (eng. *tile*) kojima „bojamo“ scenu. Paletu kreiramo na osnovu PNG dokumenta koji sam dobio u sklopu „Udemy RPG Assets“ datoteke. Imam 3 palete, *OverWorld* pomoću koje sam kreirao *Countryside* i *Town* scene, *Inner* pomoću koje sam kreirao *Shop* scenu te *Cave* pomoću koje sam kreirao *Dungeon* scenu.



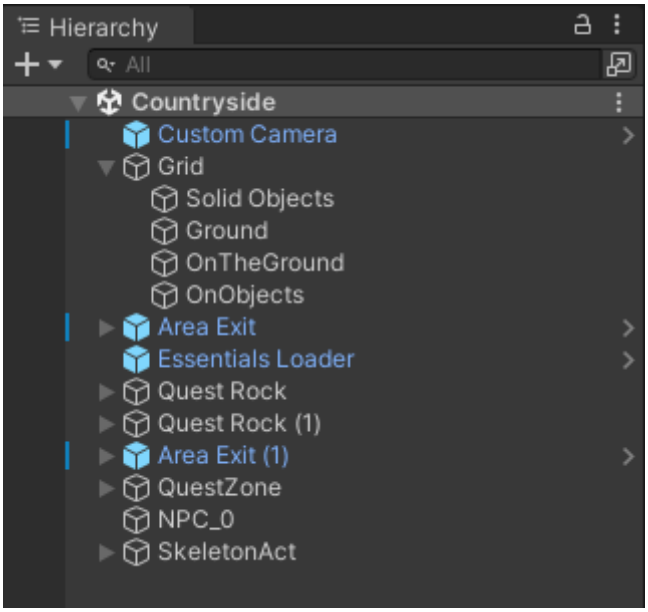
Slika 15. Paleta „Overworld“ [autorski rad]

Na slici 15 se je prikazana *Overworld* paleta te vidimo da je pod *Active Tilemap* odabrano *OnTheGround*. Radi se i mapama pločica (eng. *tilemap*) koji služe kao slojevi scene, a pomažu nam kod sortiranja što treba biti iznad čega i čemu može igrač pristupiti. Na slici 16 se može primijetiti kako imam ukupno 4 *tilemape* za scenu *Countryside*:

- *Solid Objects*
- *Ground*
- *OnTheGround*
- *OnObjects*

Mapa *Ground* predstavlja sam pod scene, odnosno tlo po kojem igrač hoda. Mapa *OnTheGround* služi za prikaz elemenata scene koji se nalaze na tlu, ali se po njima može hodati (cvijeće i slično). Mapa *Solid Objects* predstavlja objekte u sceni s kojima se igrač može sudariti. To je omogućeno pomoću dodanih komponenata *TileMap Collider 2D, Rigidbody 2D*

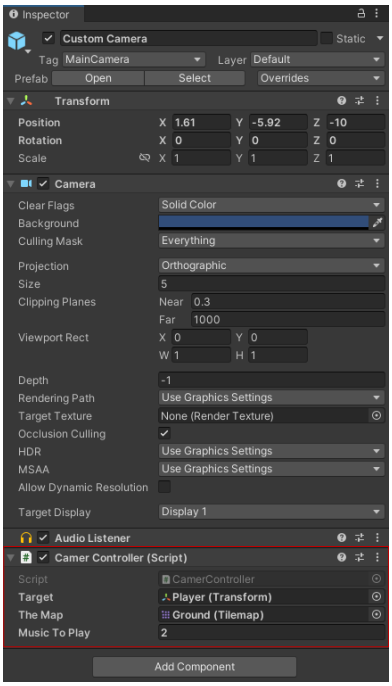
i *Composite Collider 2D.* Pod ovu mapu spadaju kuće, brda, štandovi, jezero, klupe… Zadnja mapa je *OnObjects* koja predstavlja objekte koji se nalaze u sklopu *Solid Objects*, a prikazuju se na tim objektima.



Slika 16. Hijerarhija objekata u sceni „Countryside“ [autorski rad]

4.3.3. Uspostavljanje kamere

Pri kreiranju scene, zajedno sa scenom već dobijemo kameru zvanu *Main Camera,* no ona nije od koristi kod scena gdje postoji igrač jer ne prati igrača u kretnji. Zbog toga se dodaje nova kamera kojoj se dodjeljuju komponente: *Audio Listener* i *Camera Controller* (Slika 17).



Slika 17. Inspector kamere [autorski rad]

4.3.3.1. Implementacija skripte „CameraController“

Skripta *CameraController* je C# skripta preko koje se dodjeljuju kameri mogućnosti praćenja igrača te ostajanja unutar granica scene. Na slici 17 se mogu uočiti varijable *Target, The Map* i *Music To Play* koje su definirane u ovoj skripti.

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.Tilemaps;

public class CamerController : MonoBehaviour
{
    public Transform target;

    public Tilemap theMap;
    private Vector3 bottomLeftLimit;
    private Vector3 topRightLimit;

    private float halfHeight;
    private float halfWidth;

    public int musicToPlay;
    private bool musicStarted;

    void Start()
    {
        target = FindObjectOfType<PlayerController>().transform;

        halfHeight = Camera.main.orthographicSize;
        halfWidth = halfHeight * Camera.main.aspect;

       bottomLeftLimit = theMap.localBounds.min + new Vector3(halfWidth,
      halfHeight, 0f);
       topRightLimit = theMap.localBounds.max + new Vector3(-halfWidth, -
      halfHeight, 0f);

       FindObjectOfType<PlayerController>().SetBounds(theMap.localBounds.mi
      n, theMap.localBounds.max);
    }

    void LateUpdate()
    {
        transform.position = new Vector3(target.position.x,
      target.position.y, transform.position.z);


       transform.position = new Vector3(Mathf.Clamp(transform.position.x,
      bottomLeftLimit.x, topRightLimit.x),
      Mathf.Clamp(transform.position.y, bottomLeftLimit.y,
      topRightLimit.y), transform.position.z );

       if (!musicStarted)
       {
           musicStarted = true;
           AudioManager.instance.PlayBGM(musicToPlay);
       }
    }
}
```

Isječak koda 2. Skripta „CameraController“ [autorski rad]

Skriptom *CameraController* se postiže to da kamera nikada ne ide vam granica mape. Dakle, moramo kreirati varijablu tipa `Tilemap` koja se zove `theMap`. Pomoću te varijable možemo odrediti donji lijevi kut (`bottomLeftLimit`) i gornji desni kut mape (`topRightLimit`) pomoću kojih preko `SetBounds()` funkcije određujemo granice mape. Varijable `halfHeigth` i `halfWidth` imaju vrijednost pola iznosa visine i širine kamere te se s njima definiraju nove **`Vector3`** vrijednosti koje se dodaju `localBounds.min` (donji lijevi kut) i oduzimaju od `localBounds.max` (gornji desni kut). Razlog ovog računa je taj da ne možemo raditi s granicama kamere, nego s središnjom točkom kamere koja te joj tako oduzimamo ili zbrajamo pola visine i širine kamere. Kako bi to sve radilo potrebno je u *Inspectoru* kamere varijabli *The Map* dodati *TileMap Ground* koja sadrži sve dijelove mape do koje igrač može doći. Kako bi mogli upravljati s karakteristika mape, potrebno je uključiti biblioteku `UnityEngine.Tilemaps`.

Kreirana je varijabla `target` koja je zapravo sam igrač u igri. Postavljanje igrača u središte se radi na način da mu se za početak pridoda trenutna vrijednost `transform`. *Transform* vrijednost određenog objekta sadrži vrijednosti *Position, Rotation* i *Scale* koje opisuju gdje se nalazi objekt te u kojem je položaju (Slika 18). Nakon svakog ažuriranja igre (svaki „frame“) se *Transform* kamere postavlja da je *Transformu* igrača. Na taj način se postiže to da je igrač uvijek u sredini kamere. Isto kao i kod mape, kako bi sve imalo smisla potrebno je u *Inspectoru* u varijablu *Target* prenijeti *Transform* od igrača.

Slika 18. Transform [autorski rad]

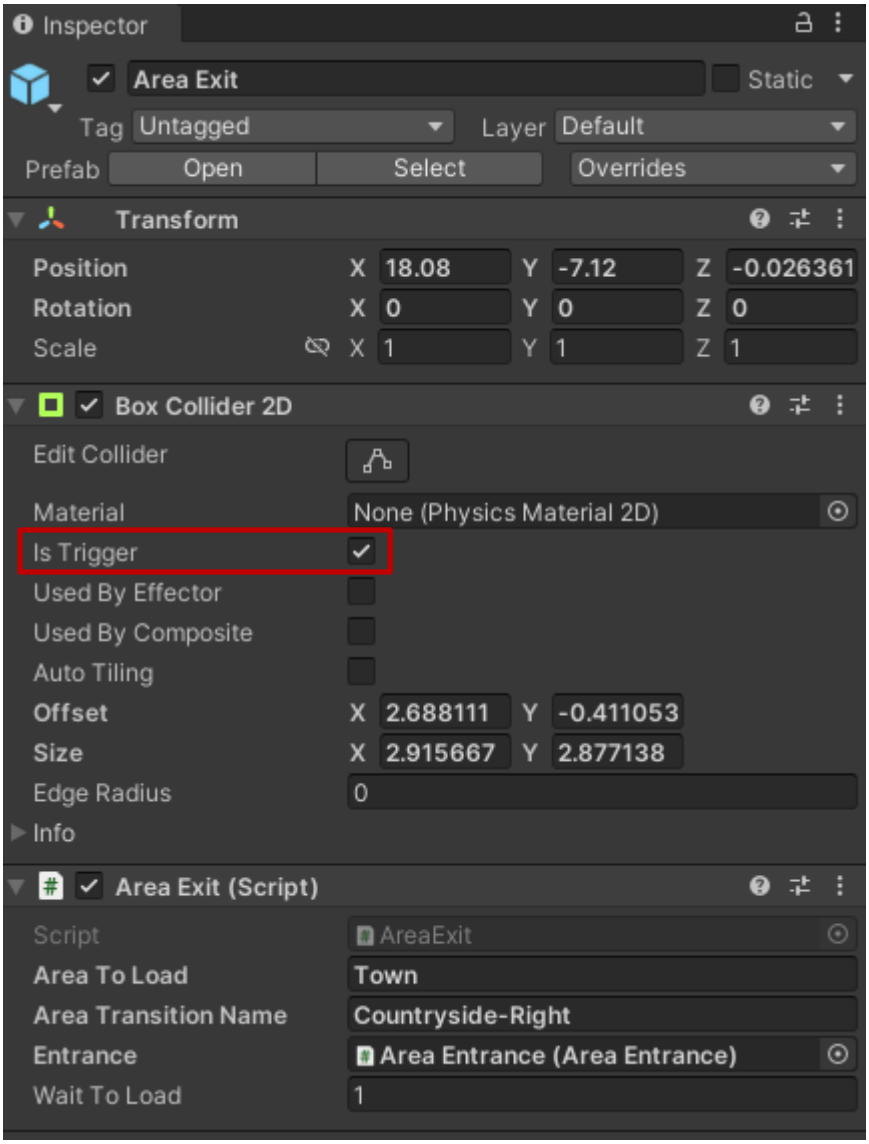
Zadnja stavka koja je implementirana u kameru je audio igre. Kako igra ne bi bila bez ikakvih zvukova i na kraju dosadna, svaka scena ima svoju melodiju dok pritisci na gumbove i napadi imaju svoj „sound effect“. Zbog toga sam kreirao `public` varijablu `musicToPlay` u koju se u Unityju unosi broj melodije za svaku scenu te se preko skripte *Audio Manager* pokreće čim se pokrene scena.

4.3.4. Tranzicija između scena

Pošto u mojoj igri postoji više scena između kojih se igrač mora moći kretati, zbog toga je potrebno napraviti sistem preko kojeg će igrač moći izaći iz jedna te ući u drugu scenu, ako se nalazi na određenoj poziciji. Ako se prisjetimo slike 16, među svim objektima se može zamijetiti dva objekta zvana *Area Exit.*

4.3.4.1. Kreiranje objekata tranzicije

Ovaj objekt predstavlja izlaz iz scene, a isti takav objekt se nalazi i na ulazu u drugu scenu jer svaki *Area Exit* ima svoj *Area Entrance* kao ulaz u scenu. Na slici 19 vidimo kako smo objektu *Area Exit* dodali komponentu *Box Collider 2D* koja služi kao područje u koje kad uđe igrač, ide u sljedeću scenu (Slika 20). Obavezno se mora označiti kučica *Is Trigger* jer time taj četverokut dobiva svoju funkciju. Ulaz u scenu, odnosno objekt *Area Entrance* je jednostavni objekt koji se nalazi nešto ispred *Area Exit-a* kako igrač ne bi zapeo u jednoj sceni. Od komponenata mu je dodana skripta *AreaEntrance* preko koje se realizira premještanje igrača.



Slika 19. Inspector „Area Exit“ [autorski rad]

Slika 20. „Area Exit“ na mapi [autorski rad]

4.3.4.2. Implementacija skripte „AreaExit“ za prijelaz među scenama

Pošto u ovoj skripti radim s scenama, potrebno je dodati biblioteku `UnityEngine.SceneManagement`. Kao što se vidi u isječku koda 3 sve `public` varijable se vide na slici 19, jedino je `entrance` varijable tipa `AreaEntrance`. Ta varijabla predstavlja ulaz koji je u sklopu tog izlaza.

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.SceneManagement;

public class AreaExit : MonoBehaviour
{

    public string areaToLoad;

    public string areaTransitionName;
    public AreaEntrance entrance;

    public float waitToLoad = 1f;
    private bool shouldLoadAfterFade;

    void Start()
    {
        entrance.transitionName = areaTransitionName;
    }

    void Update()
    {
        if (shouldLoadAfterFade == true)
        {
            waitToLoad -= Time.deltaTime;
            if (waitToLoad <= 0)
```

```
            {
                shouldLoadAfterFade = false;
                SceneManager.LoadScene(areaToLoad);
            }
        }
    }

    private void OnTriggerEnter2D(Collider2D other)
    {
        if (other.tag == "Player")
        {
            shouldLoadAfterFade = true;
            GameManager.instance.fadingBetweenAreas = true;
            UIFade.instance.FadeToBlack();

            PlayerController.instance.areaTransitionName =
areaTransitionName;
        }
    }
}
```

Isječak koda 3. Skripta „AreaExit“ [autorski rad]

Dakle, u `Start()` funkciji se varijabli `transitionName` ulaza dodjeljuje vrijednost naziva izlaza. Nakon svakog `Update()` provjerava se je li varijabla `shouldLoadAfterFade` istinita, ako je onda se nastavlja s učitavanjem nove mape nakon odbrojavanja vremena `waitToLoad` koje po defaultu postavljen na 1 sekundu. Nova scena se učitava funkcijom `LoadScene()` koja dobiva `string` vrijednost varijable `areaToLoad` unutar biblioteke `SceneManagement`. Vrijednost varijable `areaToLoad`, `areaTransitionName`, `entrance` i `waitToLoad` se postavlja unutar Unityja za svaki izlaz posebno.

Kako bi se išta dogodilo, igrač prvo mora doći u područje *Collidera,* a kada se to dogodi poziva se funkcija `OnTriggerEnter2D(Collider 2D)`. Prvo dolazi do zatamnjivanja ekrana `UIFade.instance.FadeToBlack()` te do premještanja igrača što se odvija u skripti *AreaEntrance.*

4.3.4.3. Implementacija skripte „AreaEntrance“ za prijelaz među scenama

U ovoj skripti je jedino bitna `Start()` funkcija koja se odvija odmah jer se igrač premješta u drugu scenu odmah.

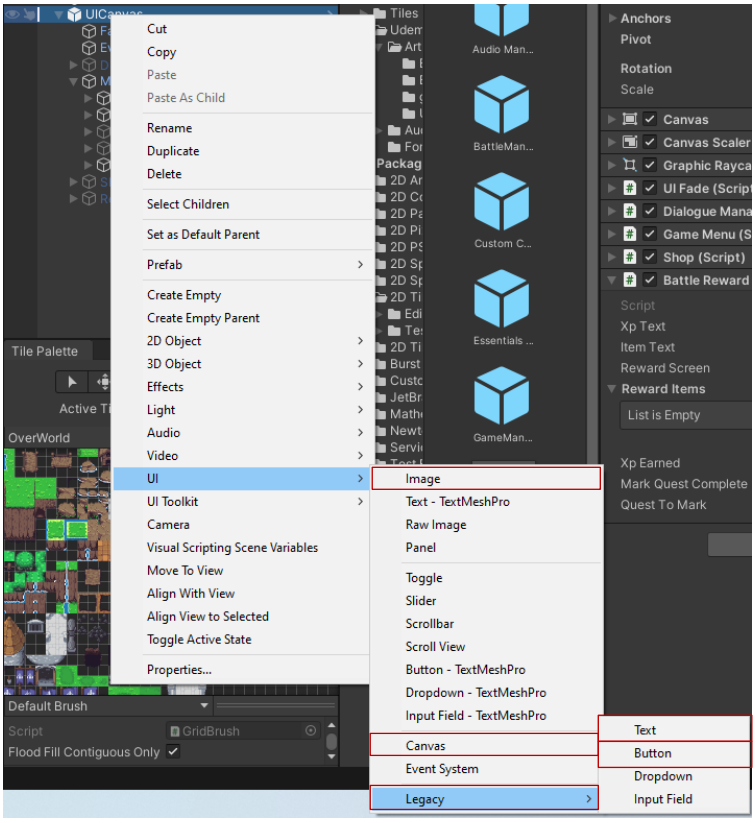
```
    public string transitionName;
    void Start()
    {
        if (transitionName == PlayerController.instance.areaTransitionName)
        {
            PlayerController.instance.transform.position =
transform.position;
        }
        UIFade.instance.FadeFromBlack();
    }
```

Isječak koda 4. Skripta „AreaEntrance“ [autorski rad]

Dakle provjeravaju se nazivi izlaza te ako su jednaki `transform.position` igrača postaje `transform.position` ulaza u scenu. Odnosno pozicija igrača dobiva vrijednost pozicije ulaza. Poziva se funkcija `FadeFromBlack()` iz skripte *UIFade* kojom se laganim prijelazom vraćamo u mapu iz crnog ekrana. Obje funkcije skripte *UIFade* funkcioniraju na način da se *alpha* vrijednost boje zaslona postepeno povećava ili smanjuje, odnosno ide do 1 (crno) ili do 0 (prozirno).

4.4. Izrada izbornika

Kao što i u svakoj igri postoji, igrač mora imati mogućnost odlaska u određeni meni. Ovisno o igri, u meniju se nalaze brojni podaci o igri, igraču, gameplay-u igre i slično. Meni sam izradio pomoću raznih alata za uređivanje korisničkog sučelja (eng. *user interface - UI*).



Slika 20. UI alati [autorski rad]

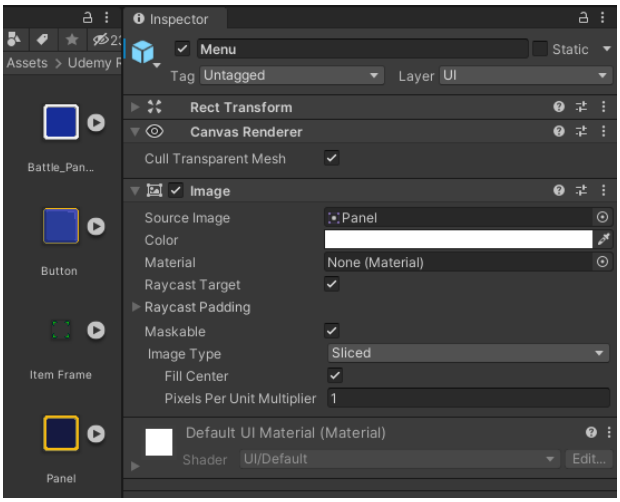
Izbornik se kreira kao i sve drugo, odnosno kao objekt. Ja sam se odlučio raditi preko *canvas-a* koji ima ulogu običnog platna u boji te je zato pogodan za moje potrebe. Na slici 20 je prikazano kako se dođe do potrebnih UI alata. Za izradu sam koristio *Image, Canvas, Text* i *Button.*

4.4.1. Pozadina izbornika

Obični *canvas* dolazi u obliku plave pozadine, no kako bi uskladili meni s ostatkom igre, mora imati određeni stil. U materijalima „Udemy RPG Assets“ u datoteci „UI“ nalaze se pikselizirani stilovi za panele i gumbove. Nakon što sam kao *Source Image* pozadine stavio stiliziranu sliku panela, pozadina menija izgleda ovako:

Slika 21. Canvas, pozadina menija [autorski rad]

Na slici 22 prikazan je *Inspector* od pozadine gdje se može vidjeti kako je *Panel* kojeg se može vidjeti u donjem lijevom kutu, ubačen na mjesto *Source Image.*



Slika 22. Inspector canvas-a [autorski rad]

4.4.2. Gumbovi izbornika

Na lijevoj strani pozadine izbornika nalaze se gumbovi kojima se orijentiramo u izborniku. Kao i pozadina, dobili su pikseliziran izgled koji sam dobio uz materijale tečaja. Ukupno imam 5 gumbova: *Items, Stats, Save, Close, Exit* (Slika 23). Svaki gumb ima svoju funkciju o čemu ćemo kada dođe red na funkcije.

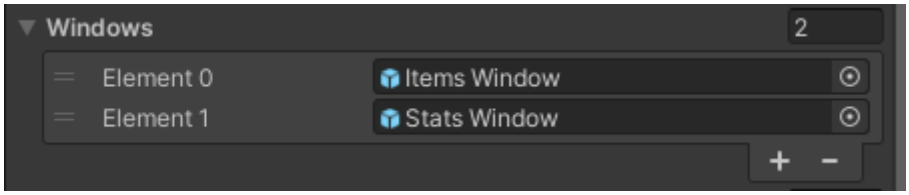
Slika 23. Gumbovi izbornika [autorski rad]

4.4.2.1. Gumbovi „Items“ i „Stats“

Gumbovima *Items* i *Stats* su se funkcije dodijelile jednom funkcijom unutar skripte *GameMenu.* U isječku koda broj 5 prikazana je funkcija `ToggleWindow(int windowNumber)`. Radi se o jednostavnoj funkciji koja prikazuje prozor s brojem koji je poslan u funkciju. Brojevi opcijama dodjeljuju se u Unityju (Slika 24).

```
public void ToggleWindow(int windowNumber)
{
    UpdateMainStats();

    for (int i = 0; i < windows.Length; i++)
    {
        if (i == windowNumber)
        {
            windows[i].SetActive(!windows[i].activeInHierarchy);
        }
        else
        {
            windows[i].SetActive(false);
        }
    }
    itemCharChoiceMenu.SetActive(false);
}
```

Isječak koda 5. Skripta „GameMenu“ [autorski rad]



Slika 24. Brojevi opcija [autorski rad]

4.4.2.2. Gumb „Close“

Gumb *Close* se također ostvaruje pomoću jedne jednostavne funkcije, CloseMenu(). Prethodna dva prozora se zatvaraju kroz for petlju, a naknadno se sam meni i informacije o igračima također deaktiviraju (isječak koda 6).

```
    public void CloseMenu()
    {
        for (int i = 0; i < windows.Length; i++)
        {
            windows[i].SetActive(false);
        }
        theMenu.SetActive(false);
        GameManager.instance.gameMenuOpen = false;

        itemCharChoiceMenu.SetActive(false);
    }
```

Isječak koda 6. Skripta „GameMenu“ [autorski rad]

4.4.2.3. Gumb „Save“

Gumb *Save* se realizira funkcijom `SaveData()` iz skripte *GameManager.* Igra se sprema u nešto zvano `PlayerPrefs`, a prema uputama Unityja, radi se o klasi koja pohranjuje preference igrača između sesija igranja te može pohraniti `string`, `float` i `integer` vrijednosti u registre korisnika [5].

```
    public void SaveData()
    {
        PlayerPrefs.SetString("Current_Scene",
SceneManager.GetActiveScene().name);
        PlayerPrefs.SetFloat("Player_Position_x",
PlayerController.instance.transform.position.x);
        PlayerPrefs.SetFloat("Player_Position_y",
PlayerController.instance.transform.position.y);
        PlayerPrefs.SetFloat("Player_Position_z",
PlayerController.instance.transform.position.z);
        for (int i = 0; i<playerStats.Length; i++)
        {
PlayerPrefs.SetInt("Player_"    +    playerStats[i].charName    +    "_level",
playerStats[i].playerLevel);
        }
    }
```

Isječak koda 7. Skripta „GameManager“ [autorski rad]

U isječku koda 7 se vidi način pohrane vrijednosti u `PlayerPrefs` gdje je prva vrijednost naziv pref-a, a druga vrijednost. Ostali podaci o svim igračima se pohranjivao kroz petlju, no u isječku koda je samo primjer za pohranu levela igrača jer se sve ostalo radi na isti način.

4.4.2.4. Gumb „Quit“

Izlaz iz igre se realizira preko *Quit* gumba koji je realiziran preko `QuitGame()` funkcije unutar skripte *GameMenu.* U isječku koda 8 se može vidjeti način izlaza iz igrice. Prvo se učitava scena *MainMenu,* nakon toga se uništava sve instance objekata koje smo kreirali za vrijeme igranja.

```
public void QuitGame()
{
    SceneManager.LoadScene(mainMenuName);

    Destroy(GameManager.instance.gameObject);
    Destroy(PlayerController.instance.gameObject);
    Destroy(AudioManager.instance.gameObject);
    Destroy(gameObject);
}
```

Isječak koda 8. Skripta „GameMenu“ [autorski rad]

4.4.3. Informacije o igračima

Nakon što sam kreirao pozadinu menija, sljedeće su na redu početne informacije o igračima kojih ukupno može biti tri. Na kraju je to ispalo kao što se može vidjeti na slici broj 24. Na slici su trenutno default podaci koji se mijenjaju dok se pokrene igra jer se podaci o igračima ažuriraju preko C# skripte. Kako bih došao do ovog rezultata, prvo sam dodao dodatna tri *canvas-a* u kojima će se nalaziti informacije. Prvo sam dodao *Image* igrača te sve tekstualne elemente kao što su ime igrača, HP (Health Points), MP (Mana Points), trenutni level te broj XP-a (Experience) do idućeg. Za prikaz potrebnog XP-a upotrijebio sam *Slider* koji se pomiče u skladu s brojkom koja piše na njemu. Informacije o igračima su početni zaslon menija kad se upali.

Slika 25. Informacije o igračima [autorski rad]

4.4.3.1. Implementacija skripte „GameMenu“ za ažuriranje informacija

Preko ove skripte se ažuriraju podaci prikazani u informacijama o igračima. Pomoću funkcije `UpdateMainStats()` se dobavljaju informacije od svih aktivnih igrača i prikazuju na ekranu.

```
    public void UpdateMainStats()
    {
        playerStats = GameManager.instance.playerStats;

        for (int i = 0; i < playerStats.Length; i++)
        {
            if (playerStats[i].gameObject.activeInHierarchy)
            {
                charStatHolder[i].SetActive(true);

                nameText[i].text = playerStats[i].charName;
                hpText[i].text = "HP: " + playerStats[i].currentHP + "/" +
playerStats[i].maxHP;
                mpText[i].text = "MP: " + playerStats[i].currentMP + "/" +
playerStats[i].maxMP;
                lvlText[i].text = "Lvl: " + playerStats[i].playerLevel;
                xpText[i].text = "" + playerStats[i].currentXP + "/" +
playerStats[i].expToNextLevel[playerStats[i].playerLevel];
                xpSlider[i].maxValue =
playerStats[i].expToNextLevel[playerStats[i].playerLevel];
                xpSlider[i].value = playerStats[i].currentXP;
                charImage[i].sprite = playerStats[i].charImage;
            }
            else
            {
                charStatHolder[i].SetActive(false);
            }
        }
        goldText.text = GameManager.instance.currentGold.ToString();
```

```
    }
```

Isječak koda 9. Skripta „GameMenu“ [autorski rad]

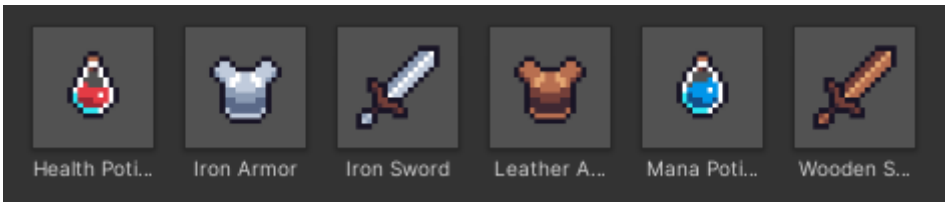
Aktivnost igrača se provjerava pomoću `IsActiveInHierarchy()` koji vraća bool vrijednost `true`, ako je object aktivan te `false`, ako je neaktivan. Podaci o igračima se pohranjuju u polje `playerStats`, tipa `CharStats`. Kroz polje se prolazi pomoću for petlje te uključujemo `charStatHolder` (informacije o igračima *canvas*) za svakog igrača koji je aktivan. Za svakog aktivnog igrača se u Text elementima u meniju prikazuju trenutni njegovi podaci.

4.4.4. Opcija „Items“

Opcija *Items* na meniju služi za prikaz svi *item-a* koje igrač posjeduje. Dakle, igrač kroz igru sakuplja razne *item-e* koji mu mogu biti korisni za preživljavanje te jačanje.

4.4.4.1. Popis „Item-a“

Od *item-a* u igri imam: *Health Potion, Mana Potion, Iron Sword, Wooden Sword, Iron Armor, Leather Armor.*



Slika 26. „Item-i“ [autorski rad]

- *Health Potion* – daje igraču 50 HP
- *Mana Potion* – daje igraču 25 MP
- *Wooden Sword* – snaga mača je 10
- *Iron Sword* – snaga mača je 15
- *Leather Armor* – snaga oklopa je 10
- *Iron Armor* – snaga oklopa je 15

4.4.4.2. „Item“ gumbovi

Prvo što sam napravio kod popisa *itema* je izrada prozora koji će dobro prikazivati same *iteme* te biti intuitivan korisniku. U njemu samo pomoću *Grid Layout Group* napravio popis gumbova s podacima o *item-ima,* odnosno s slikom i brojkom dostupne količine.

Slika 27. „Item“ gumbovi [autorski rad]

Svakom gumbu je dodijeljena funkcija `Press()` iz skripte „ItemButton“ preko koje se vrše određene funkcije koje želimo.

```
    public void Press()
    {
        if (GameMenu.instance.theMenu.activeInHierarchy)
        {
            if (GameManager.instance.itemsHeld[buttonValue] != "")
            {

GameMenu.instance.SelectItem(GameManager.instance.GetItemDetails(GameManage
r.instance.itemsHeld[buttonValue]));
            }
        }

    }
```

Isječak koda 10. Skripta „ItemButton“ [autorski rad]

Preko funkcije prikazane u 10. isječku koda, o *item-u* na koji se pritislo se prikazuju informacije te mogućnost korištenja ili brisanja. Informacije o *item-u* kao što su naziv i opis se prikazuju ispod popisa te se prikazuju dva dodatna gumba *Use* i *Discard*.

Slika 28. „Item“ informacije [autorski rad]

Na slici 28 se prikazan primjer odabira prvog *item-a*, *Health Potion-a*. Prikazane su informacije o odabranom *item-u* te gumbovi za korištenje i brisanje. Dohvaćanje informacija se vrši preko funkcije `GetItemDetails(string itemToGrab)` u kojoj se uspoređuju nazivi svih *item-a* i traženog naziva. Ako se radi *item-u*, prikazuje se gumb *Use*, a ako se radi o oružju (eng. *weapon*) prikazuje se gumb *Equip*. Pritiskom na gumb *Use*, pokreće se funkcija `Use(int charToUseOn)` preko koje se igraču pridodaju efekti *item-a.*

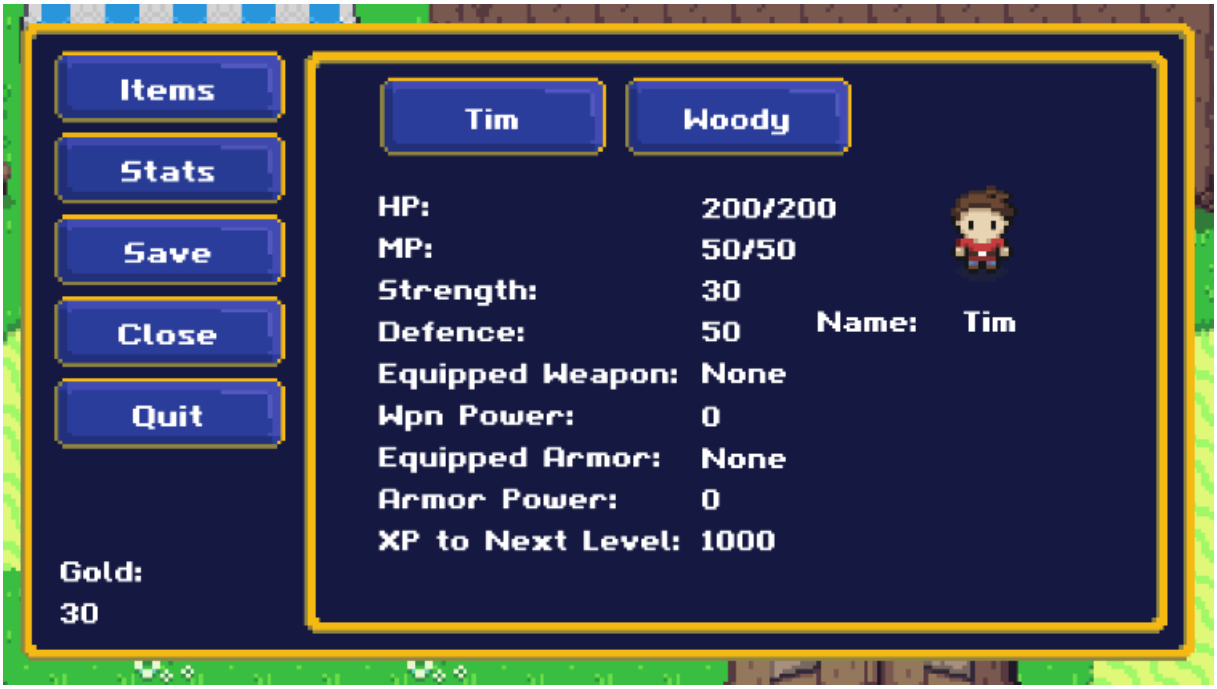
```
public void Use(int charToUseOn)
{
    CharStats selectedChar = GameManager.instance.playerStats[charToUseOn];

    if (isItem)
    {
        if (selectedChar.currentHP != selectedChar.maxHP)
        {
            if (affectHP)
            {
                selectedChar.currentHP += amountToChange;
                if (selectedChar.currentHP > selectedChar.maxHP)
                {
                    selectedChar.currentHP = selectedChar.maxHP;
                }

            }

        }

            GameManager.instance.RemoveItem(itemName);
    }
```

Isječak koda 11. Skripta „Item“ [autorski rad]

U isječku koda 11 prikazan je dio funkcije `Use()` koji je korišten za item-e koji utječu na HP igrača. U varijablu `selectedChar` se pohranjuju podaci odabranog igrača te se provjerava je li *item* zapravo *item* ili je nešto drugo, npr. oružje. Ako *item* utječe na HP igrača, `currentHP` igrača se povećava za `amountToChange` tog *item-a* te ako je HP veći od maksimalnog iznosa, onda je jednak tom maksimalnom iznosu. Na kraju se *item* briše iz popisa, odnosno smanjuje mu se količina za jedan.

4.4.5. Opcija „Stats“

Opcija na meniju, *Stats,* služi za detaljan pregled informacija o igraču kao što su *HP, MP, Strength, Defence, Equipped Weapon, Equipped Armor, Weapon Power, Armor Power* i *XP to Next Level.* Na vrhu prozora se nalaze gumbovi odabira igrača za kojeg želimo vidjeti informacije, a prikazuju se samo za aktivne igrače te tako na slici 29 se vide samo dva gumba.



Slika 28. Opcija „Stats“ [autorski rad]

Informacije se prikazuju pomoću funkcije `StatusChar(int selected)` unutar skripte *GameMenu*. Tekst svakog tekstualnog elementa prozora se mijenja jer mu se dodjeljuju informacije o igraču kao što je prikazano na isječku koda broj 12. Varijabla selected predstavlja igrača za kojeg se prikazuju informacije te je sve ostalo jasno u kodu. Ono što se mora provjeriti je ima li igrač već nekakvo oružje ili oklop. U slučaju da nema, piše se „None“.

```
    public void StatusChar(int selected) {
        statusName.text = playerStats[selected].charName;
        statusHP.text = "" + playerStats[selected].currentHP + "/" +
playerStats[selected].maxHP;
        statusMP.text = "" + playerStats[selected].currentMP + "/" +
playerStats[selected].maxMP;
        statusStrength.text = playerStats[selected].strength.ToString();
        statusDef.text = playerStats[selected].defence.ToString();
        if (playerStats[selected].equippedWpn != "")
        {
            statusWpnEq.text = playerStats[selected].equippedWpn;
        }
        statusWpnPwr.text = playerStats[selected].weaponPower.ToString();
        if (playerStats[selected].equippedArmr != "")
        {
            statusArmrEq.text = playerStats[selected].equippedArmr;
        }
        statusArmrPwr.text = playerStats[selected].armorPower.ToString();
        statusXp.text =
(playerStats[selected].expToNextLevel[playerStats[selected].playerLevel] -
playerStats[selected].currentXP).ToString();
        statusImg.sprite = playerStats[selected].charImage;
    }
```

Isječak koda 12. Skripta „GameMenu“ [autorski rad]

4.5. Sistem borbe

Zadnja stavka izrade moje igre je bila izrada sistema borbe u igri, odnosno kreiranje borbi, načina napadanja te uostalom i započinjanja borbi.

4.5.1. Korisničko sučelje borbe

Prvo što sam napravio je izabrao pozadinu koja će biti prikazana tokom borbe. Pozadinu sam uzeo iz materijala dobivenih uz tečaj. Sljedeće je bilo potrebno napraviti određeni meni kojim će se korisnik koristiti tijekom borbe. Pošto sam objasnio kako se radi meni, sad ću samo prikazati kako to finalno izgleda. Kao što se vidi na slici broj 29, korisnik ima opciju napasti (*Attack*), koristiti magiju (*Magic*) te pobjeći (*Flee*). Osim ovih mogućnosti, korisnik vidi i informacije o igračima kao što su *HP* i *MP*. Ukupno u borbi može biti uključeno 3 igrača te 6 protivnika koji svaki ima svoju poziciju. Pozicije su napravljene kao prazni objekti pravilno razmješteni kako bi, u slučaju borbe 3v6 svi imali mjesta te ne bi izgledalo nagurano.

Slika 29. „Battle UI“ [autorski rad]

4.5.2. „Attack“ opcija

Pritiskom na *Attack* gumb, otvara se *Target Menu* u kojemu su igraču ponuđeni protivnici koje može napasti (slika 30). Ovo se izvršava pomoću funkcije `OpenTargetMenu(string moveName)` unutar skripte *BattleManager* u kojoj se općenito nalazi velika većina funkcija vezane za sistem borbe.

Slika 30. „Target Menu“ [autorski rad]

```
public void OpenTargetMenu(string moveName)
{
    targetMenu.SetActive(true);

    List<int> enemies = new List<int>();

    for (int i=0; i < activeBattlers.Count; i++)
    {
        if (!activeBattlers[i].isPlayer)
        {
            enemies.Add(i);
        }
    }

    for (int i = 0; i < targetButtons.Length; i++)
    {
```

```
            if (enemies.Count > i && activeBattlers[enemies[i]].currentHP >
0)
            {
                targetButtons[i].gameObject.SetActive(true);

                targetButtons[i].moveName = moveName;
                targetButtons[i].activeBattlerTarget = enemies[i];
                targetButtons[i].targetName.text =
activeBattlers[enemies[i]].charName;
            }
            else
            {
                targetButtons[i].gameObject.SetActive(false);
            }
        }
    }
```

Isječak koda 13. Skripta „BattleManager“ [autorski rad]

Kreira se lista protivnika u koju idu svi sudionici borbe koji nisu igrači što se realizira u `if` selekciji s uvjetom `!activeBattlers[i].isPlayer`. Sljedeće se provjerava imaju li protivnici više od 0 HP te ako imaju aktivira se njihov gumb u *Target Menu*. Pritiskom na jednog od ponuđenih protivnika igrač napada te smanjuje HP za određeni iznos, ovisno o snazi igrača i napada.

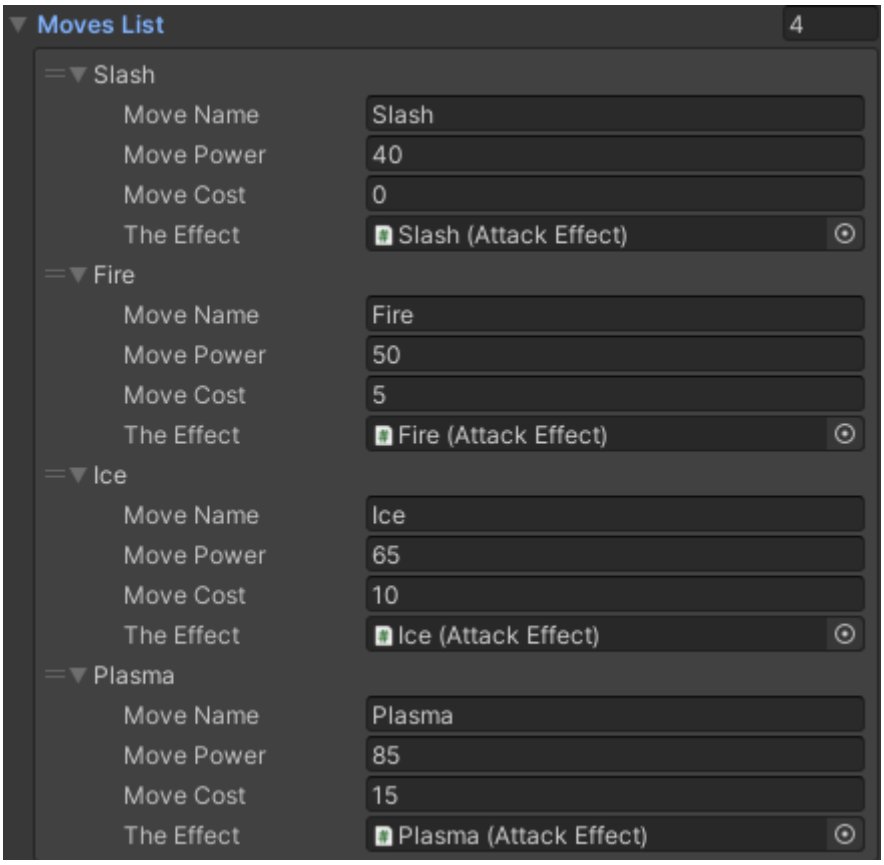
4.5.3. „Magic“ opcija

Pritiskom na Magic gumb, otvara se Magic Menu u kojemu su popisani magični napadi koje igrač može izvesti. Navedeni potezi se mogu naći te mijenjati u karakteristikama igrača u listi *Moves Available.* Svaki napad košta određeni broj MP-a, *Fire* je 5 MP, *Ice* je 10 MP i *Plasma* je 15 MP.

Slika 31. „Magic Menu“ [autorski rad]

Popis magičnih napada se generira na isti način kao i *Target Menu* samo što se prolazi kroz polje *Moves Available*. Nakon odabira napada, provjerava se ima li igrač dovoljno MP za taj napad, ako nema pokazuje se obavijest, a ako ima prelazi se na *Target Menu*.

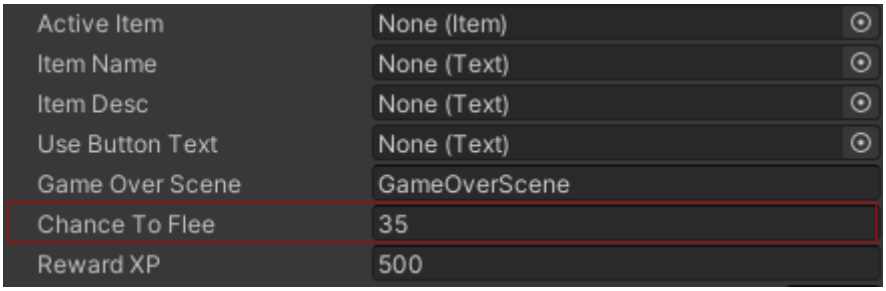
Unutar *BattleManager* prefaba se nalazi lista *Moves List* u koju se dodaju prefabi napada (slika 30). U toj se listi podešava *Power* napada te koliko MP košta. Svaki napad ima svoj efekt koji se prikazuje na protivniku kada ga igrač napadne s određenim napadom. Efekti su napravljeni na isti način kao i animacija kretnje igrača, pomoću *Animatora*.



Slika 32. „Moves List“ [autorski rad]

4.5.4. „Flee“ opcija

Pritiskom na opciju *Flee*, igrač ima određeni postotak mogućnosti da pobjegne od protivnika. Ova mogućnost može biti korisna, ako je protivnik dosta veći rank od igrača te igrač nema šanse dobiti taj duel. Postotak se unosi *u BattleManager-u* pod *Chance To Flee* (slika 33).



Slika 33. Postotak bijega iz borbe [autorski rad]

4.5.5. Funkcija napadanja

Nakon odabira protivnika kojeg igrač napada, izvršava se napad pomoću `PlayerAttack (string moveName, int selectedTarget)` funkcije unutar *BattleManager* skripte.

```
    public void PlayerAttack (string moveName, int selectedTarget)
    {
        int movePower = 0;
        for (int i = 0; i < movesList.Length; i++)
        {
            if (movesList[i].moveName == moveName)
            {
                Instantiate(movesList[i].theEffect,
activeBattlers[selectedTarget].transform.position,
activeBattlers[selectedTarget].transform.rotation);
                movePower = movesList[i].movePower;
            }
        }
        Instantiate(enemyAttackEffect,
activeBattlers[currentTurn].transform.position,
activeBattlers[currentTurn].transform.rotation);

        DealDamage(selectedTarget, movePower);

        uiButtonsHolder.SetActive(false);
        targetMenu.SetActive(false);
        NextTurn();
    }
```

Isječak koda 14. Skripta „BattleManager“ [autorski rad]

Kao što je prikazano u isječku kod broj 14, prolazi se kroz listu napada te se traži odabrani napad. Ako je pronađen, napad se instancira na poziciji odabrane mete. Odnosno, prikazuje se animacija napada nad `selectedTarget`. Nakon toga se isto tako instancira efekt izvođenja napada nad igračem ili protivnikom koji vrši napad. Još što je potrebno napraviti je da taj napad zapravo naškodi protivniku, a upravo to radi funkcija `DealDamage (int target, int movePower)`. *Damage* se izračunava pomoću tri linije koda:

```
        float attackPower = activeBattlers[currentTurn].strength +
activeBattlers[currentTurn].wpnPwr;
        float defPower = activeBattlers[target].defence +
activeBattlers[target].armPwr;

        float damageCalc = (attackPower / defPower) * movePower *
Random.Range(.9f, 1.1f);
```

Isječak koda 15. Skripta „BattleManager“ [autorski rad]

Dakle, snaga igrača uz snagu oružja, ako ga ima, se dijeli s snagom obrane protivnika zbrojenu s snagom oklopa, ako ga ima. Za točan izračun *damage-a*, podijeljen iznos se množi s snagom napada. Na kraju, kako bi dodao mali udio sreće i slučajnosti u igru, stavio sam da se taj iznos množi s nasumičnim brojem između 0.9 i 1.1.

5. Zaključak

Kao zaključak bi htio iznijeti da je izrada videoigre puno teža nego što se čini te da je potrebno uložiti veću dozu truda kako bi bili zadovoljni s konačnim rezultatom. Rezultat moje igre je videoigra uloga u kojoj se igrač može slobodno kretati po mapi te započinjati misije u kojima je nagrađen brojnim nagradama. Igrač ima mogućnosti započinjanja dijaloga s drugim likovima u igri te započinjati borbe protiv protivnika. Sve to sam napravio pomoću tečaja, programskih alata Unity i Microsoft Visual Studio. Ovaj rad sam temeljio na mojoj percepciji što je više važno, a što manje te sam zbog toga većinu rada posvetio samom praktičnom dijelu jer sam htio da sve bude jasno objašnjeno kako je što realizirano. Zbog toga teorijskog dijela nema previše, no dovoljno da njime potkrijepim praktični dio.

U uvodu navodim kako je najopširniji dio ovog rada zapravo funkcionalna logika same igre. Kao što se vidi kroz rad, najveći dio zauzima objašnjavanje upravo te logike pomoću koje igra funkcionira. U početku sam objasnio kako se povezuje kod s Unityjem te kako se on implementira da mu se dodijeli određena vrijednost i funkcija. Kasnije se nisam puno obazirao na taj dio jer je u principu ostatak sličan za implementirati, moguće su određene male promjene. Kao rezultat rada kroz rad je cjelokupno funkcioniranje igre koja je spremna za proširenje te izradu ozbiljnije igre čak i za komercijalne svrhe. Radeći na ovoj igri pokupio sam mnoštvo znanja koje će mi dobro doći i u nastavku studiranja te sam izrazito zadovoljan odabirom ove teme jer sam se natjerao naučiti nešto novo, a znanje ne mogu izgubiti.

Popis literature

[1] James Doyle, „Learn To Create An RPG Game In Unity“, 2020. [Na internetu]. Dostupno: https://www.udemy.com/course/unity2drpg/ [pristupano 14.7.2022. – 24.8.2022.]

[2] LinkedIn, „Unity“ (bez dat.) [Na internetu]. Dostupno: https://www.linkedin.com/company/unity/ [pristupano 24.8.2022.]

[3] „Unity“, (bez dat.) u *Wikipedia, the Free Encyclopedia.* Dostupno: https://en.wikipedia.org/wiki/Unity_(game_engine) [pristupano: 24.8.2022.]

[4] Microsoft Visual Studio, (bez dat.) „Code faster Work smarter“ [Na internetu]. Dostupno: https://visualstudio.microsoft.com/vs/ [pristupano: 25.8.2022.]

[5] Unity Documentation, (bez dat.) „PlayerPrefs“ [Na internetu]. Dostupno: https://docs.unity3d.com/ScriptReference/PlayerPrefs.html [pristupano: 27.8.2022.]

[6] E. Makuch, Gamespot, (8.4.2020.) „Cyberpunk 2077 Dev Shares Key Diversity Details About Its Workforce“ [Na internetu]. Dostupno: https://www.gamespot.com/articles/cyberpunk-2077-dev-shares-key-diversity-details-ab/1100-6475835/ [pristupano: 30.8.2022.]

[7] „Računalne igre uloga“, (bez dat.) u *Wikipedia, the Free Encyclopedia.* Dostupno: https://hr.wikipedia.org/wiki/Ra%C4%8Dunalne_igre_uloga_ [pristupano: 30.8.2022.]

[8] P. Palmer, PCGamer, (20.11.2021.) „How to make a great RPG character“ [Na internetu] Dostupno: https://www.pcgamer.com/how-to-make-a-great-rpg-character/ [pristupano: 30.8.2022.]

[9] Paladin, (6.8.2022.) „How to write a good game story“ [Na internetu]. Dostupno: https://paladinstudios.com/2012/08/06/how-to-write-a-good-game-story-and-get-filthy-rich/ [pristupano: 30.8.2022.]

[10] Plarium, (3.8.2022.) „RPG Classes: How to Find Your Character“ [Na internetu]. Dostupno: https://plarium.com/en/blog/rpg-classes/ [pristupano: 30.8.2022.]