Razvoj JRPG digitalne igre Shadow Tomb Valley korištenjem Unity platforme

Babović, Tomislav

Sveučilište u Rijeci – Odjel za informatiku

Preddiplomski jednopredmetni studij informatike

Tomislav Babović

Shadow Tomb Valley

Završni rad

Mentor: izv. prof. dr. sc. Marina Ivašić-Kos

Rijeka, rujan 2020.

Sadržaj

ODJEL ZA INFORMATIKU
SVEUČILIŠTE U RIJECI

Rijeka, 17. veljače 2020.

Zadatak za završni rad

Pristupnik: **Tomislav Babović**

Naziv završnog rada: **Razvoj JRPG digitalne igre Shadow Tomb Valley korištenjem Unity platforme**

Naziv završnog rada na eng. jeziku: **Development of the Shadow Tomb Valley JRPG digital game using the Unity platform**

Sadržaj zadatka: Proučiti i opisati različite vrste video igara s naglaskom na specifične karakteristike RPG i JRPG igara. Proučiti osnovne elemente dizajna i razvoja video igara te odgovarajuće alate za dizajn i razvoj video igara kao što je Unity.

Osmisliti vlastitu JRPG igru, smisliti priču, scene, likove, neprijatelje, prepreke, zadatak i cilj igre te način ostvarivanja i gubljenja bodova i života. Izraditi i opisati razvoj prototipa vlastite igre u Unity-u. Opisati scene i način generiranja scena, predmete na sceni, poziciju kamere, likove, spriteove i animaciju likova, implementaciju zvukova u igri. Opisati kao su u igri implementirane prepreke i neprijatelji, njihove karakteristike i ponašanje, učestalost pojavljivanja, način borbe. Objasniti način stjecanja i gubljenja bodova, računanje rezultata i praćenje napretka u igri te mogućnost napredovanja kroz igru.

Opisati osnovne kontrole igrača, korisničko sučelje i HUI te njihovu implementaciju u igri. Opisati algoritme i skripte koje su razvijene kako bi se ostvarila dodatna funkcionalnost objekata u igri te dati prijedlog nadogradnje igre i mogućnost poboljšanja.

Mentor

izv. prof. dr. sc. Marina Ivašić-Kos

Voditelj za završne radove

doc. dr. sc. Miran Pobar

Zadatak preuzet: 17.6.2020.

Babović T.

(potpis pristupnika)

1 Sažetak

U ovom radu se opisuje razvoj „japanske igre uloga(JRPG)“ video igre „Shadow Tomb Valley“ u Unity razvojnom alatu. Koncept igre podsjeća na starije JRPG-ove za gameboy ili Sony playstation.

Igra se sastoji od „Heroja“ s kojim igrač upravlja te pokušava doći do „ShadowTomba“. U igri postoje sigurne zone i zone u kojima su neprijatelji. Borba je napravljena tako da svatko ima svoj potez tokom kojeg može izvoditi akciju. Prilikom pobjede u borbi igrača se nagrađuje zlatom i iskustvom te u rijetkim slučajevima i predmetom. Igrač sakupljanjem iskustva povećava svoj nivo. Na taj način otključava jednu od dvije klase te vještine. U jednoj od sigurnih zona se nalaze prodavač i gostioničar kod kojih je moguće kupiti, prodati predmete te prespavati noć te tako u potpunosti obnoviti zdravlje i manu.

Ključne riječi: Unity, objekt, skripta, klasa, metoda, igrač, protivnik

2 Uvod

Povijest igara seže sve u prahistoriju kada su predmeti za igru izrađivani od kostiju [1]. Razvitkom čovječanstva razvijale su se i igre. Tako je razvoj tehnologije doveo do razvoja video igara. Prve video igre se pojavljuju 50-ih godina prošlog stoljeća ali zbog cijene računala video igre postaju popularne tek 70-ih godina. Među prvim popularnijim igrama je „Pong“ [2] nastao 1972. godine od strane tvrtke Atari. Popularnost ove igre dovela je do razvoja novih igara. Razvoj igara traje i danas. Tako danas imamo skoro 3 milijarde ljudi koji uživaju u čarima video igara [3]. Video igre su se vrtoglavo razvile te je sve više turnira posvećeno istima. Nagradni fondovi znaju dosezati i do preko 30 milijuna dolara. Također postoji mogućnost da eSport postane sastavni dio olimpijskih igara.

Video igre se dijele u mnogo žanrova, ali za razliku od filmova i knjiga, svijet u kojem se događa radnja ne uvjetuje žanr. Neki od popularnijih žanrova su: akcija, pucačka igra, sportske igre, igranje uloga…[4]. JRPG je podžanr igri uloga. Jedna od specifičnosti ovakvih igara je borba koja se odvija u potezima, tako zvana „turn-based combat“. Među najprodavanijim naslovima igri uloga su „Pokemon“ koji je prodan u više od 300 milijuna primjeraka [5]. Tu se nalazi još nekoliko igara iz istog podžanra kao što su „Final Fantasy“ i „Dragon Quest“. Razlog tome je što su to serijali nastali tokom zlatnog doba igara uloga. Zlatno doba igara uloga je bilo od 1990. do sredine 2000-ih. Jedan od razloga je što je JRPG više fokusiran na priču od klasičnih igri uloga. U tom razdoblju JRPG su se brzo razvijale te su postajale sve više fokusirane na priču. Problem u tome je što su postale donekle zatvorene za zapadno tržište jer je postalo skupo prevoditi tekst sa japanskog na engleski [6].

Cilj ovog rada je osmisliti JRPG igru „Shadow Tomb Valley“ i prikazati njen razvoj korištenjem razvojnog alata Unity i programskog jezika C#. Igra je napravljena za jednog igrača te se igra isključivo upotrebom tipkovnicom. U igri su sadržani osnovni koncepti svake JRPG igre kao što su borba bazirana na potezima, vještine, klase, različite vrste predmeta, slučajne borbe te borba sa šefom.

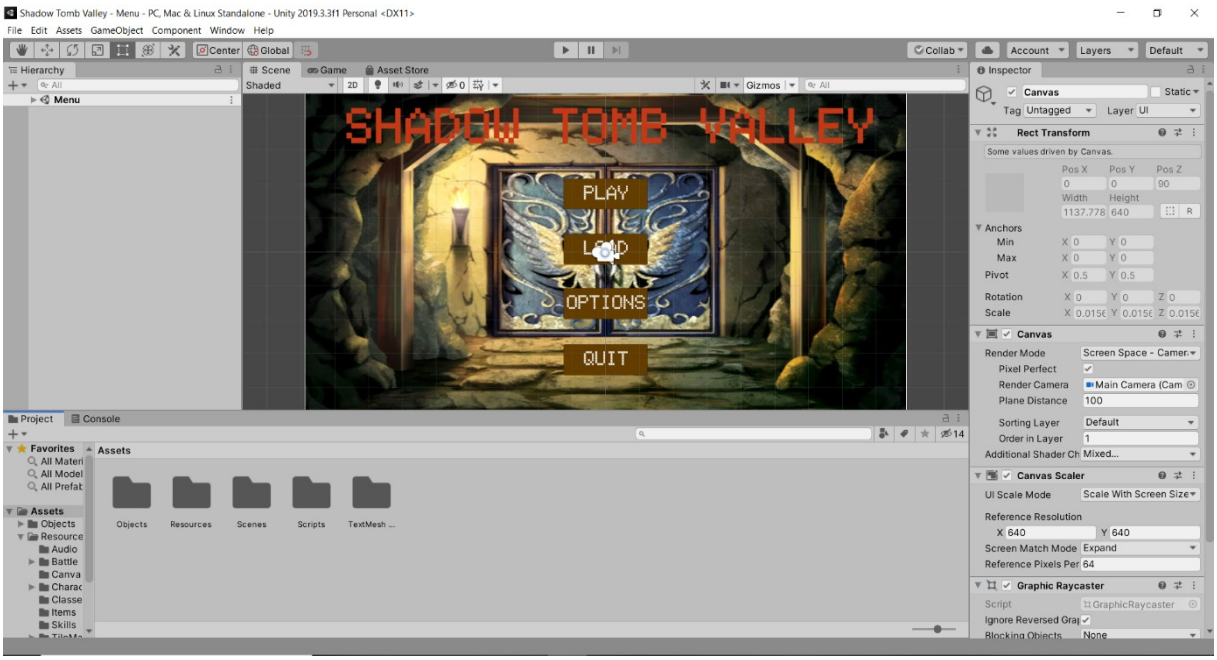
3 Unity razvojno okruženje

Unity je višeplatformski softver za razvoj 2D i 3D video igara, virtualne stvarnosti i simulacija. Osim video industrije koristi se i u drugim industrijama, kao što su filmska i arhitektonska. Izradila ga je kompanija Unity Tehnologies i predstavila 2005. godine na Worldwide Developers konferenciji. Na izlasku je bio namijenjen isključivo za platformu OS X, dok danas ima podršku za preko 25 platformi [7]. Unity podržava C# programski jezik u kojemu su pisane skripte za za izradu video igre „Shadow Tomb Valley“ koja je tema ovog rada. Osim C# Unity podržava JavaScript i Python. Unity je potpuno besplatan za korištenje, međutim sadrži svoj „Assets store“ na kojem postoji mogućnost kupovine različitih pomagala za razvoj igre. Unity verzija korištena u izradi video igre je 2019.3.3f1.

3.1 Unity sučelje

Prilikom pokretanja programa prikazuje se početno sučelje prikazano na slici 1. Sastoji se od više dijelova te je u potpunosti prilagodljiv. Početno sučelje se sastoji od:

- Hierarchy
- Scene
- Game
- Assets Store
- Inspector
- Project
- Console

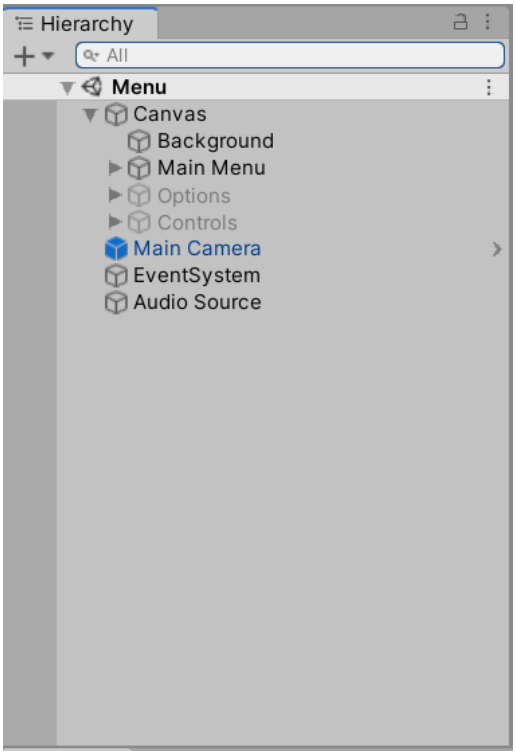


Slika 1 Unity početno sučelje

Naravno ovo su samo neki od prozora programa te se ostali mogu uključiti u padajućem izborniku „Window“.

3.1.1 Hierarchy

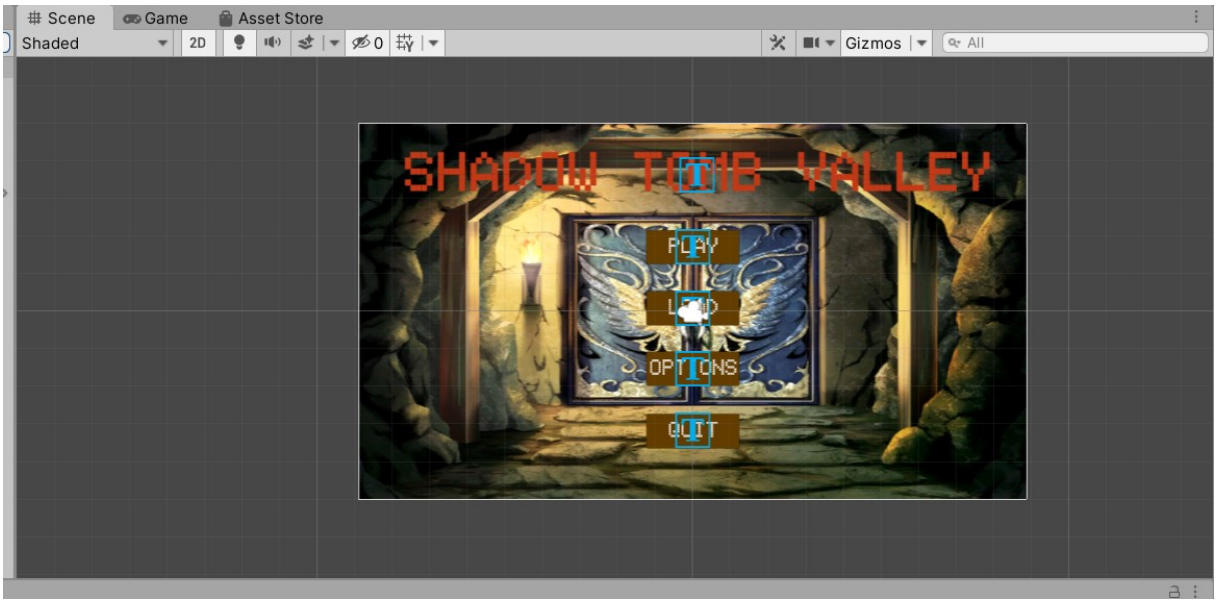
Hierarchy [8] (hrv. hijerarhija) služi za hijerarhijski prikaz svih objekata na trenutnoj sceni. Koristi koncept roditeljstva, tj. objekti se mogu grupirati unutar jednog objekta roditelja. Taj koncept je vrlo važan jer unutar sučelja možemo postaviti objekte koje želimo napraviti vidljivima na sceni, odnosno koje ne. Grupiranjem možemo isključiti roditelja te na taj način i djecu, npr. objekt „Options“ na slici 2. To svojstvo je vrlo važno u radu sa korisničkim grafičkim sučeljem jer na taj način možemo sakriti više objekata koji su usko povezani te ih sve prikazati odjednom uključivanjem roditelja.



Slika 2 Hierarchy

3.1.2 Scene

Scene [9] (hrv. Scena) je interaktivni prozor u svijet kojeg korisnik kreira. Na sceni možemo mijenjati rotaciju, poziciju te veličinu objekata. Svaka nova scena dolazi sa jednom komponentom, to je kamera. Ona služi za prikaz naše igre na ekran te ju na sceni možemo uređivati kao svaki drugi objekt u igri. Na slici 3 vidimo početnu scenu igre.



Slika 3 Scene

3.1.3 Game

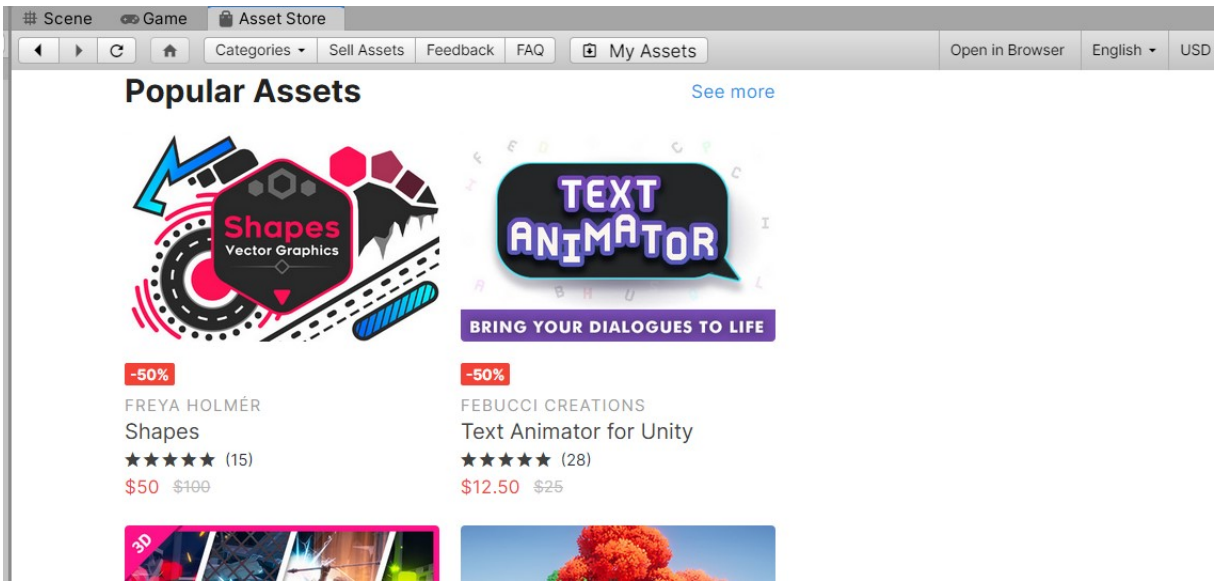
Game [10] (hrv. Igra) služi za prikaz igre. Tu se može uključiti „Play mode“ kako bi se pokrenula trenutna scena, te vidjelo kako se ponaša unutar igre. Da bi na game sučelju bila prikazana scena potrebna je kamera na sceni. Na slici 4 vidljivo je game sučelje i mogućnosti koje isto nudi. Tako na primjer se može mijenjati rezolucija te vidjeti kako se igra prikazuje na različitim rezolucijama. To je zapravo jako bitno jer Unity ne nudi opciju mijenjanja rezolucije pomoću koda osim za gotovu igru.

Slika 4 Game

3.1.4 Assets Store

Assets store [11] (hrv. Dućan sredstava) sadrži biblioteke besplatnih i komercijalnih objekata

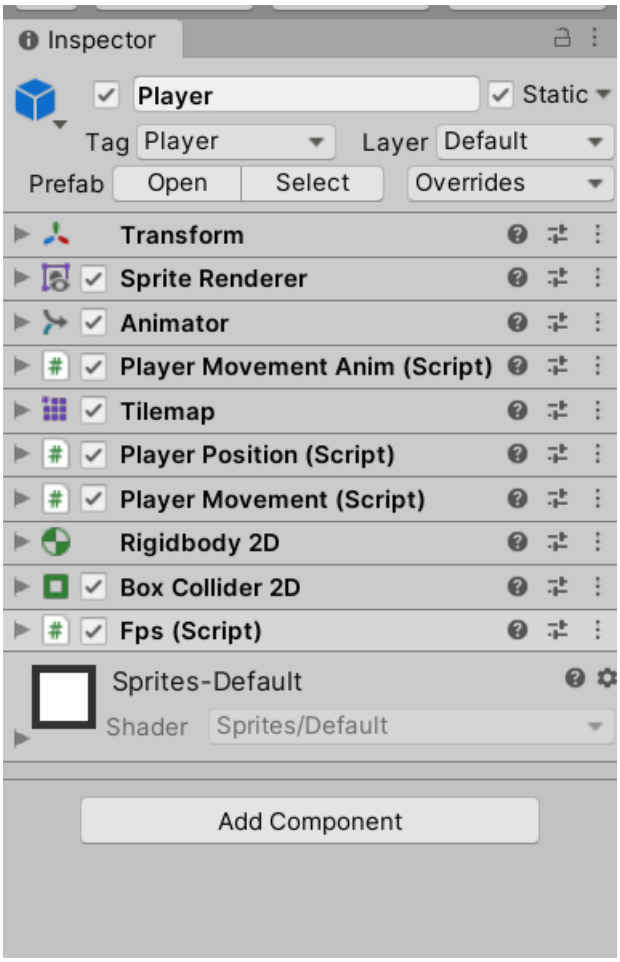
koji se mogu koristiti u igri. Sadrži mnoštvo tekstura, modela, animacija, ali i već gotovih projekata i vodiča. Na slici 5 se vide dva dodatka koje su napravili drugi korisnici ovog softvera.



Slika 5 Assets store

3.1.5 Inspector

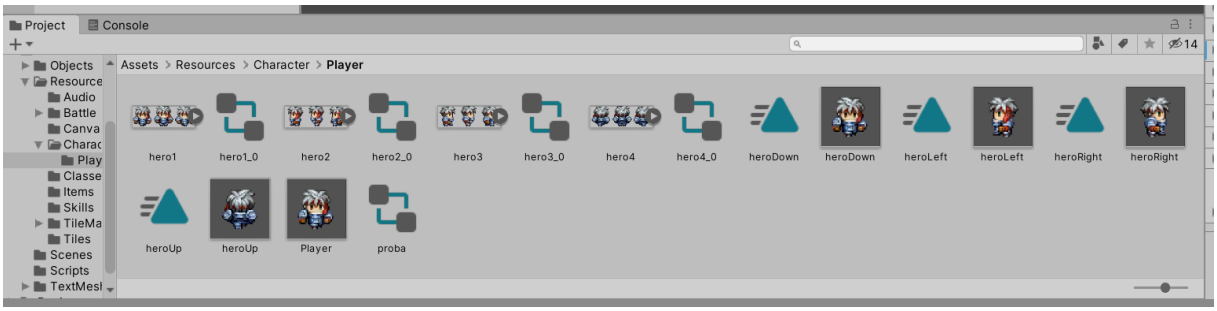
Inspector [12] (hrv. Inspektor) služi za inspekciju trenutno odabranog objekta. Tu su prikazane sve komponente i svojstva trenutnog objekta te ih se može mijenjati bez potrebe za kodom. Svi objekti na sceni dolaze sa komponentom „transform“ tj. „rect transform“ ako je objekt vezan za grafičko sučelje. Ta komponenta nam govori gdje na sceni je taj objekt smješten, te koja mu je veličina, rotacija i sidrište. Objektima je moguće dodati nove komponente kao što su „box collider 2D“ i „Rigidbody 2D“ koji u kombinaciji služe za simulaciju tijela objekta. U slučaju da korisnik spremi neki objekt za ponovno korištenje, tzv. „prefabs“ u inspektoru se nude dodatne opcije. Slika 6 prikazuje „prefab“ objekta Player.



Slika 6 Inspector

3.1.6 Project

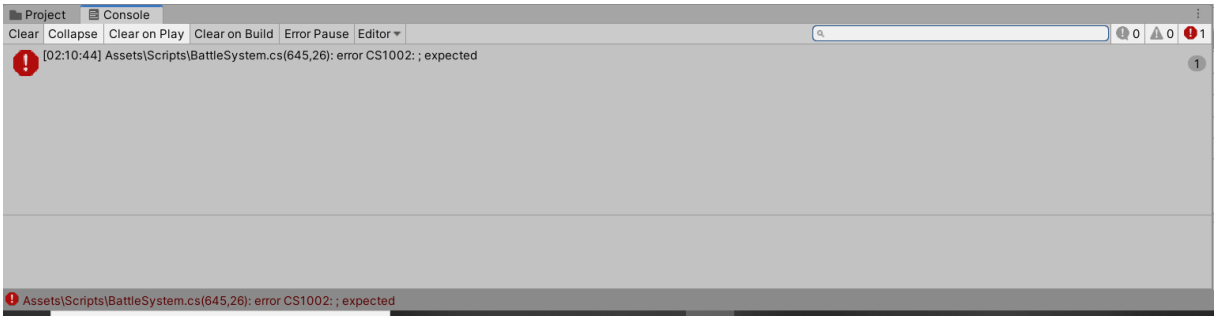
Project [13] (hrv. Projekt) prikazuje sve datoteke i mape u trenutno otvorenom projektu (slika 7). Sastoji se od dva dijela, lijevi dio prikazuje hijerarhijski odnos datoteka i mapa u projektu. Desni dio prikazuje sadržaj trenutno odabrane mape. Prilikom izrade projekta potrebno je grupirati i raspodijeliti datoteke u mape kako bi se uvijek moglo lako pristupiti određenoj datoteci.



Slika 7 Project

3.1.7 Console

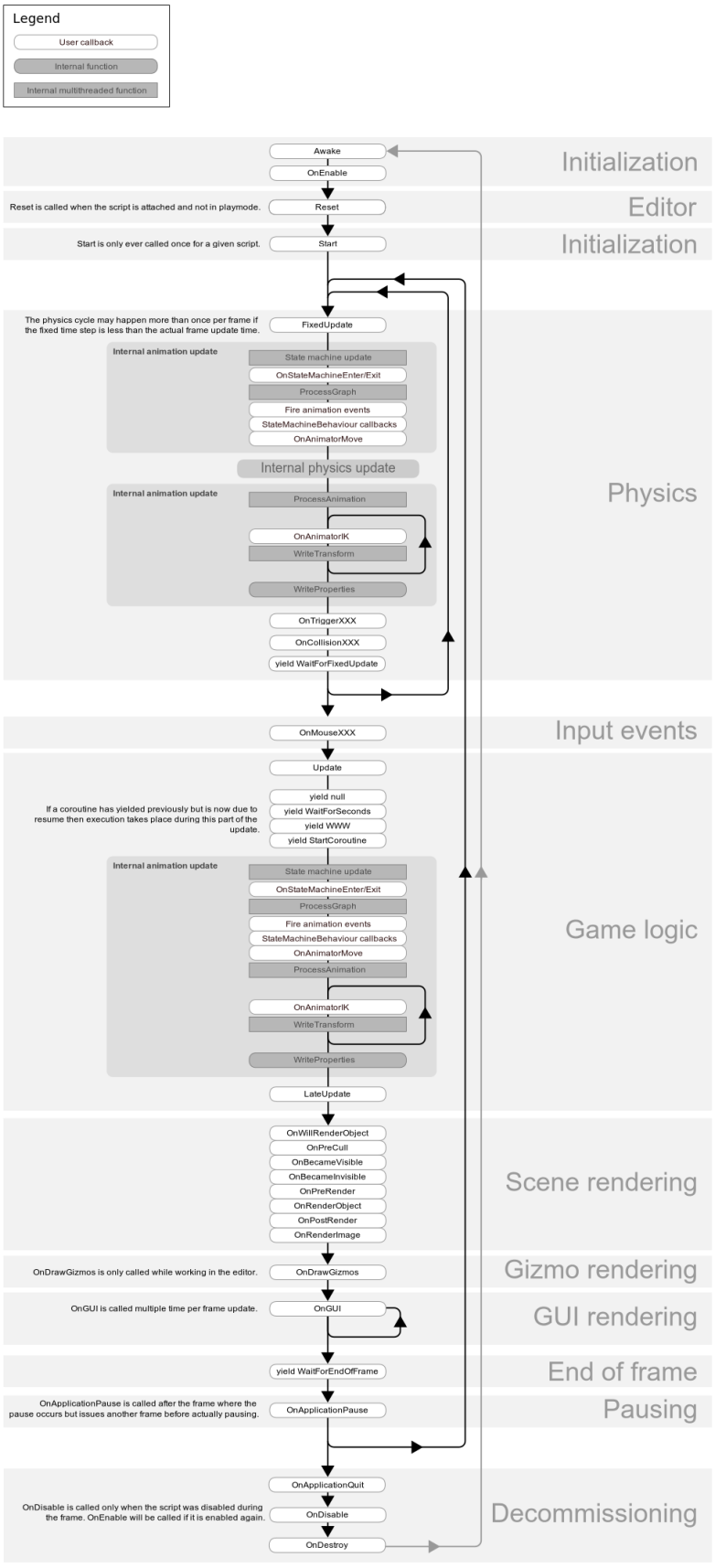
Console [14] (hrv. Konzola) služi za prikaz pogreški, upozorenja te poruka koje korisnik generira sam u svrhu otklanjanja pogreški ili Unity kad dovrši neku radnju. Slika 8 prikazuje konzolu koja obavještava korisnika o pogrešci nastaloj u skripti „BattleSystem.cs“ u redu broj 645.



Slika 8 Konzola sa porukom o pogrešci

3.2 MonoBehaviour

MonoBehaviour [15] je bazna klasa iz koje proizlaze sve druge skripte u Unityju. Jedino skripte koje proizlaze iz te klase mogu se dodati objektima. Osim toga nudi mogućnost korištenja ko-routina, koje služe za asinkrono izvođenje koda, te mogućnost izvođenja koda na osnovu događaja u projektu.



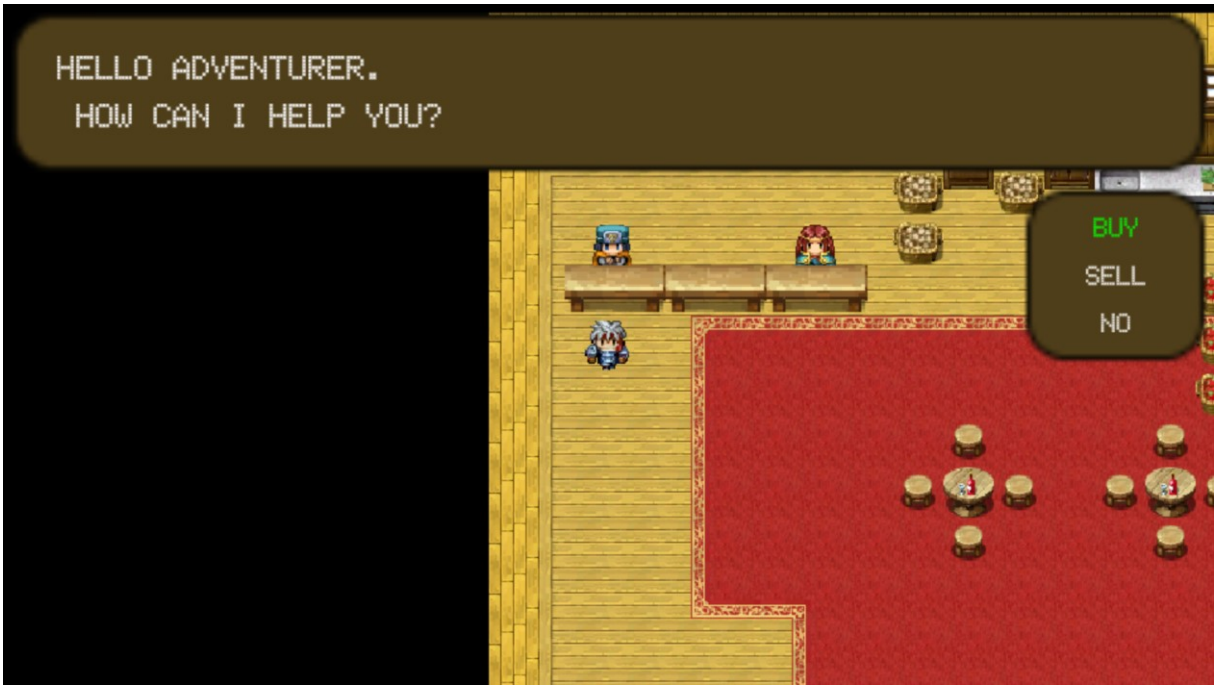
Slika 9 Tijek izvođenja metoda u MonoBehaviour klasi

Neke od važnijih metoda su „Start“, „Awake“, „OnEnable“, „Update“ „FixedUpdate“ i „OnCollisionEnter“. „Start“ i „Awake“ imaju skoro istu funkcionalnost te se obje metode izvode jednom. Razlika je da se „Awake“ izvodi kad se objekt aktivira, dok se „Start“ izvodi tek kad se skripta aktivira. „OnEnable“ se razlikuje od „Start“ po tome što se izvodi uvijek kad se skripta aktivira. „Update“ i „FixedUpade“ se razlikuju po tome što se „Update“ poziva za svaku sličicu dok se „FixedUpdate“ poziva za svaki određeni vremenski interval. Iz tog razloga se „FixedUpdate“ koristi za fiziku. „OnCollisionEnter“ je samo jedna od metoda iz familije OnCollision/OnTrigger te služi za izvođenje koda kada se dva objekta dotaknu.

4 Razvoj igre

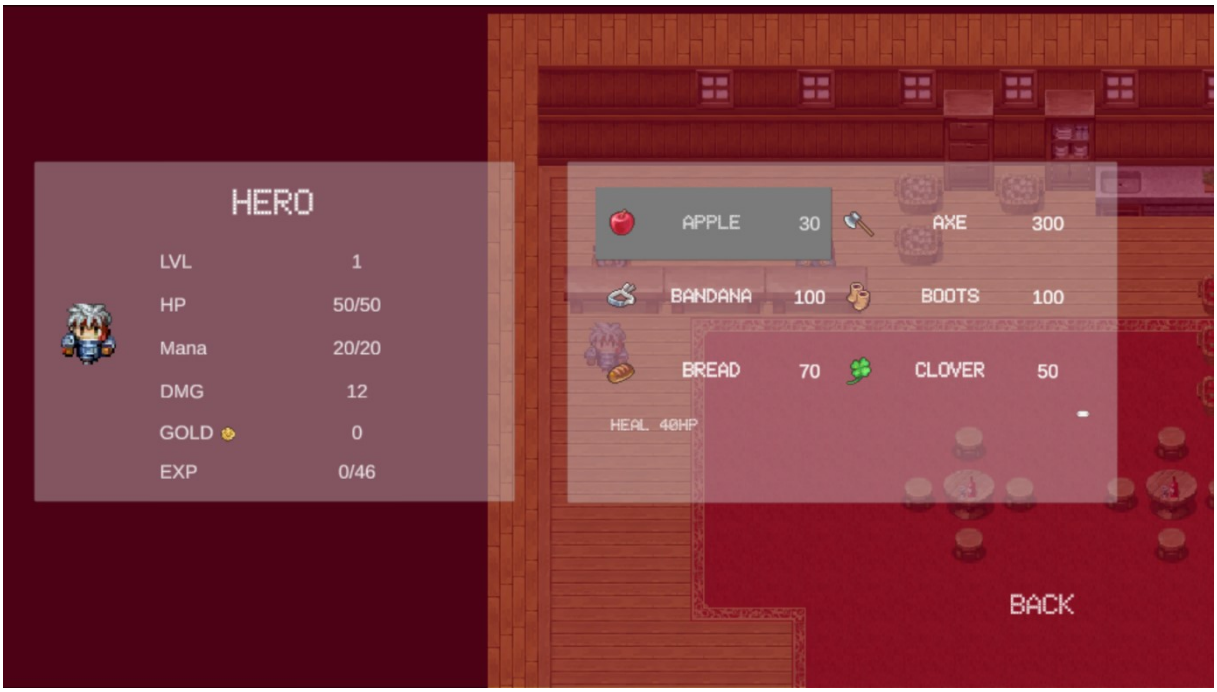
U nastavku se opisuje proces razvoja video igre „Shadow Tomb Valley“. Inspiracija za izradu ove igre je video igra „Final Fantasy“ nastala 1987. godine. Po uzoru na „Final Fantasy“ i u ovoj igri igrač ima slučajne borbe u kojima sakuplja iskustvo i zlato. Skupljanjem iskustva igrač otključava klasu i vještine kako bi bio u mogućnosti poraziti „Shadow kinga“.

Igra počinje buđenjem igrača u svojoj kući. Igrač otkriva svijet u kojem se našao te ima na izbor istražiti grad ili izaći iz njega. Istraživanjem grada nailazi na NPC-a koji kroz dijalog (slika 10) nudi igraču opciju kupovine i prodaje itema.



Slika 10 Primjer dijaloga

Izbornik za kupovinu (slika 11) i prodaju sadrži iteme koje igrač može kupiti, tj. ako posjeduje item, prodati.

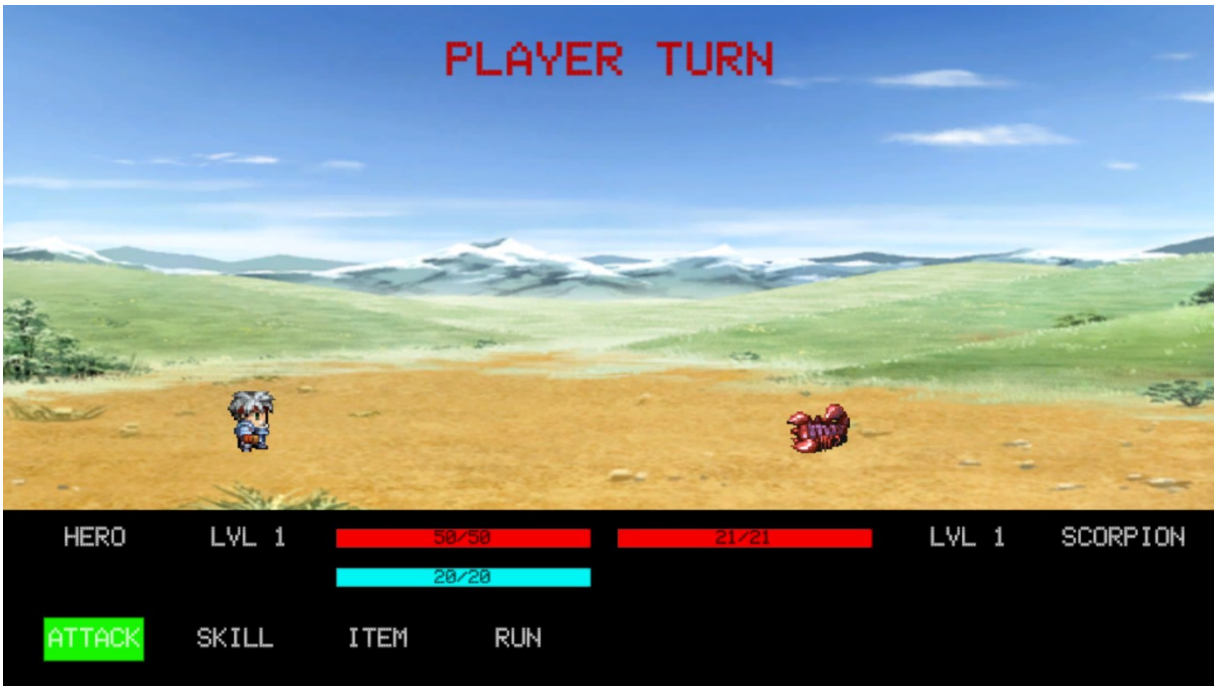


Slika 11 Prikaz izbornika kupovine

Izlaskom iz grada igrač se nalazi na sceni svijeta (slika 12).

Slika 12 Scena svijeta

Istražujući svijet oko sebe igrača napadaju protivnici. Prilikom ulaska u borbu sa protivnicima pokreće se scena borbe (slika 13).



Slika 13 Scena borbe

Ukoliko igrač pobijedi u borbi biva nagrađen iskustvom i zlatom. Zlato služi za kupovinu itema te za odmaranje u gostionici kako bi si obnovio zdravlje i manu. Iskustvo je potrebno kako bi igrač podigao svoj nivo te time otključao klase i vještine. Osim toga podizanjem nivoa igraču se povećaju atributi. Povećanjem atributa igrač postaje jači.

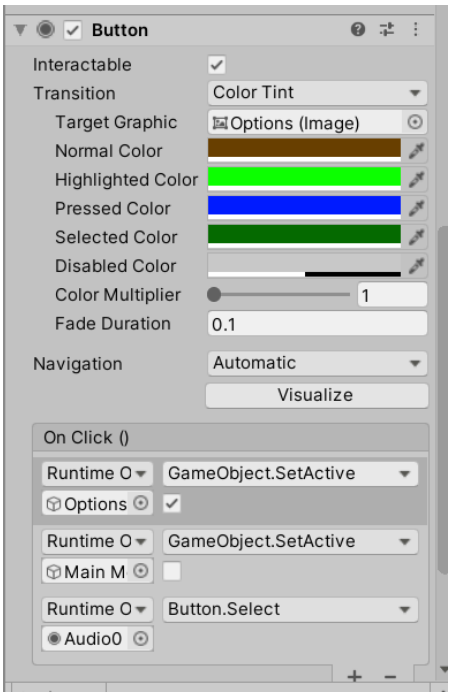
Cilj igre je kroz nasumične borbe sa protivnicima podići nivo igrača, sakupiti zlato, te kupiti iteme. Nakon što se igrač opremio itemima i podigao svoj nivo dovoljno visoko, može se zaputiti u borbu sa „Shadow Kingom“, te pobjedom nad njim otključati ulaz u „Shadow Tomb“ i pobijediti igru.

Igrač se sastoji od 12 spriteova podijeljenih u 4 animacije. U borbi kao i protivnici je prikazan jednim spriteom te ima dodatnu animaciju za prikaz napada.

4.1 Main menu

Main menu (hrv. Glavni izbornik) scena je sastavni dio svake video igre. Postavke koje nudi glavni izbornik su opcije, pregled kontrola s kojima se upravlja igra, postavljanje glasnoće zvuka, mijenjanje rezolucije, pokretanje prijašnje spremljene igre te izlazak iz nje. Na slici 4 je prikazan glavni izbornik igre implementiran interaktivnim gumbima. Gumb je unaprijed generiran objekt koji dolazi sa komponentom „Button“ te djetetom „Text“.

4.1.1 Options



Slika 14 Options gumb

Slika 14 prikazuje gumb „Options“ glavnog izbornika. Neke od postavki koje ta komponenta nudi su mijenjanje boje gumba u različitim stanjima, navigacija, te dodavanje događaja koji se izvršavaju kada se gumb pritisne. Ako scena nije kompleksna, automatska navigacija koju Unity nudi je sasvim dovoljna za kretanje po gumbima korištenjem tipkovnice. „OnClick“ događaji koji se dodaju gumbu mogu biti sve „public void“ metode koje primaju najviše jedan jednostavan argument. „Public void“ metoda je zapravo dio koda koji ne vraća rezultat. Jedna od korištenijih metoda u ovom radu, „SetActive“, postoji za svaki objekt u Unityju te služi za aktiviranje objekta na sceni. Tako „Options“ gumb deaktivira trenutno prikazane gumbe i aktivira ploču „Options“ na kojoj postavlja gumb „Audio0“ kao trenutno odabrani objekt. Mogućnosti koje nudi ovaj dio glavnog izbornika su promjena glasnoće zvuka, pregled kontrola te promjenu rezolucije. Glasnoća zvuka u Unityju je u rasponu od 0 do 1. Na slici 15 se vidi da postoje tri gumba za promjenu glasnoće, pritiskom na gumb postavlja se glasnoća na vrijednost koja piše na gumbu.

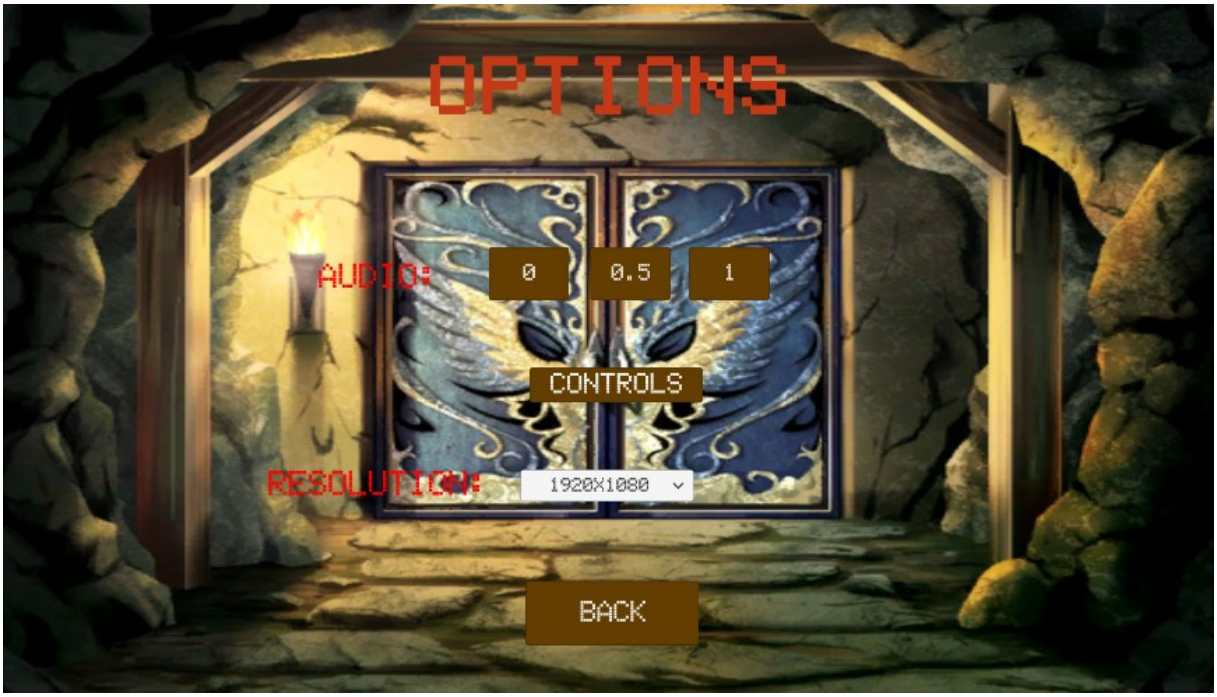
```
    public void ChangeVolume(float volume)
    {
        audioSource.volume = volume;

    }
```

Igra je napravljena tako da se zvuk konstantno reproducira tako da na početnoj sceni postoji objekt sa skriptom koja onemogućava njegovo uništavanje osim ako već ne postoji objekt sa istom skriptom na sceni.

```
if (audioPlayer == null)
        {
            audioPlayer = this;
            DontDestroyOnLoad(this);
            audioSource.Play();
        }
        else
        {
            Destroy(this);
        }
```

Osim glasnoće može se promijeniti i rezolucija odabirom odgovarajuće rezolucije u padajućem izborniku.



Slika 15 Game options

Pritiskom na gumb „CONTROLS“ prikazuju se kontrole igre. Svi objekti koji prikazuju tekst imaju komponentu „TextMeshPro – Text (UI)“. Ta komponenta zamjenjuje komponentu „Text“ . Obje komponente služe za prikaz teksta uz razliku da „TextMeshPro – Text (UI)“ nudi puno veću mogućnost uređivanja izgleda.

4.1.2 Play game

Pritiskom na gumb „Play“ se pozivaju metoda „SetUpGame“. Metoda služi za postavljanje početnih vrijednosti igre kao što su igračev nivo i igračevo zdravlje:

```
PlayerUnit.unitLvl = 1;
PlayerUnit.maxHP = 50;
```

Nakon što se postave početne vrijednosti spremi se ime trenutne scene i pokrene se nova:

```
PlayerPrefs.SetString("_last_scene_", SceneManager.GetActiveScene().name);
SceneManager.LoadScene("PlayerHouse", LoadSceneMode.Single);
```

Analogno tome gumb „Load game“ poziva metodu „Load Game“ s kojom se učitavaju spremljene vrijednosti igre. Scene se pokreću preko imena scene te se prije svakog pokretanja nove scene sprema scena na kojoj se igrač prethodno nalazio. To služi za provjeru kod pokretanja scene „Player house“ i kod povratka sa scene „Battle“. U prvom slučaju služi kod pokretanja nove igre kako bi igrač započeo igru uvijek na istoj poziciji, ali prilikom povratka na scenu se ne postavlja na tu poziciju. Na ostalim scenama prije pokretanja nove scene se sprema pozicija na kojoj će se pozicionirati igrač na novoj sceni.

Provjerava se je li prethodna scena početna scena igre, te ako je postavlja igrača na zadanu poziciju:

```
if (PlayerPrefs.GetString("_last_scene_") == "Menu")
            transform.position = new Vector2(3.5f, 3.5f);
        else
            transform.position = new Vector2(PlayerPrefs.GetFloat("x"), Player
Prefs.GetFloat("y"));
```

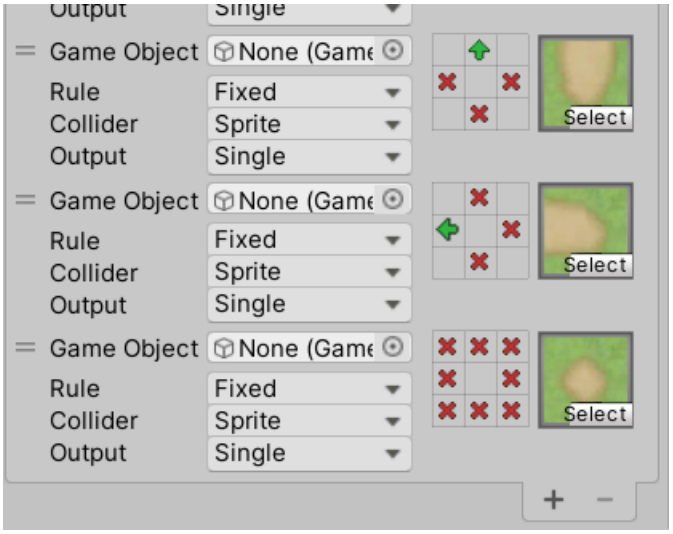
Transform.position se odnosi na poziciju objekta koji sadrži skriptu kao svoju komponentu. Pozicija u Unityju je vektorska udaljenost u smjeru x, y i z osi od ishodišta koordinatnog sustava. S obzirom da se prikazuje razvoj 2D igre veličina udaljenosti u smjeru z osi je uvijek 0 te se koristi new Vector2 metoda koja prima argumente samo za x i y os.

4.2 Scene svijeta

Igra se igra unutar šest scena od čega su pet scena po kojima se igrač može kretati. Tih pet scena jednim imenom se nazivaju scene svijeta. Sve su realiziranje korištenjem objekta grid koji služi kao roditelj za objekt tilemap. Objekt grid služi za prikaz koordinatnog sustava razdijeljenog u kvadratiće veličine 1x1. Objekt tilemap je dijete grida te služi za popločavanje grida. Ta dva objekta su važna za razvoj 2D igara jer na brz i jednostavan način omogućuju uređivanje izgleda scene. Objekti ne dolaze u instalaciji sa Unityjem te ih je potrebno preuzeti sa „Package Manager“. Da bi se mogao koristit sistem popločavanja također je potrebno napraviti i ploče koje se žele postaviti. Prvi korak je stvoriti „Tile Palette“ (Slika 16) u kojeg se dodaju spriteovi koji se žele postaviti na mapu.

Slika 16 Tile Palette

Također osim običnih ploča moguće je koristiti i ploče sa pravilima „Rule Tile“. To su posebna vrsta ploča kojima se dodaju određena pravila, te prilikom postavljanja Unity automatski prema pravilima mijenja izgled ploča. Na slici 17 je prikazano nekoliko pravila jedne takve ploče korištene u izradi igre. Pravila se sastoje od kvadrata podijeljenog u 9 manjih kvadratića. Svaki od tih manjih kvadrata služi za vizualizaciju grida, središnji kvadrat označava poziciju na koju se smješta ploča dok ostali kvadrati označavaju susjede. Susjedni kvadrati mogu biti prazni, sadržavati ili zelenu strelicu ili crveni x. Na primjeru prvog pravilo će se objasniti kako pravila funkcioniraju. Strelica označava da mora postojati gornji susjed, a x-evi kako ti susjedi ne smiju postojati. Prazni kvadrati označavaju da za izgled nije bitan susjed na toj poziciji. Razlog tome je što za generiranje potpunog rule tilea za 2D igru se koristi 47 spriteova dok rule tile omogućava puno više pravila. Nakon što se postavi svih 47 pravila dovoljno je otvoriti „Tile Palette“, izabrati napravljeni rule tile, te pritiskom miša na željenu lokaciju postaviti tile. Ukoliko se zadovolje pravila iz rule tilea, spriteovi na tim pozicijama se mijenjaju u sprite postavljen u rule tileu.



Slika 17 Rule tile

Za izradu igre je korišteno nekoliko rule tileova te nekoliko tilemapova. Razlog korištenja više tilemapova je taj što omogućuje dodatne opcije uređivanja izgleda mape. Na slici 18 se vidi

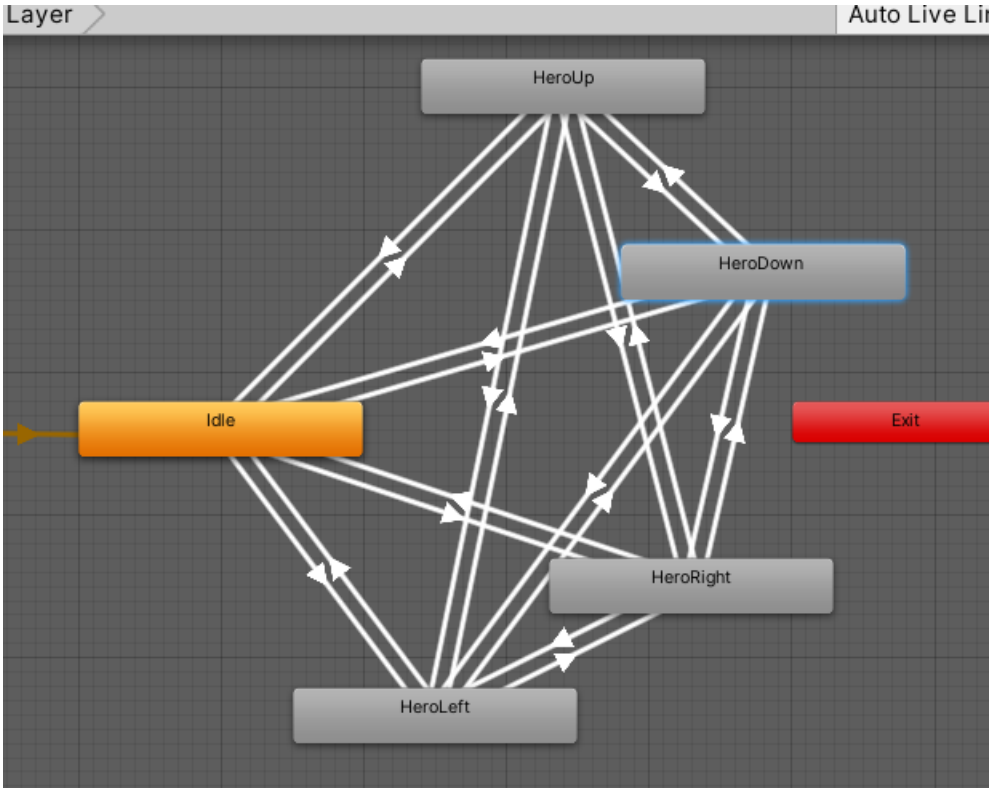
primjer toga. Korištenjem odvojenih tilemapova moguće je postaviti bazu mape i stolove i stolice odvojeno. Korištenjem jednog tilemapa bi se morao napraviti sprite koji sadrži bazu, stol i bocu.

Slika 18 Primjer više tilemapova

Također je moguće dodavanje komponente „Tilemap Collider 2D“ te na taj način napraviti tilemap koji prikazuje objekte s kojima se igrač sudara. Na primjeru slike 18 igrač se može kretati po crvenoj površini dok ne može po stolicama i stolu.

4.3 Igrač

Igrač se sastoji od nekoliko komponenti. Sprite renderer koji je zadužen za prikaz spritea koji predstavlja igrača, animatora koji je zadužen za upravljanje animacijama za kretnju igrača. Animacije se rade tako da se u „Animation“ prozoru dodaju spriteovi koje se žele animirati. Može se birati u kojoj milisekundi animacije se želi aktivirati pojedini sprite. Na taj način se dobiva niz spriteova koji se u određenom vremenskom intervalu prikažu. Jedan objekt može sadržavati više animacija, te su sve animacije sadržane u animatoru. Animator je zapravo deterministički automat kojem se šalje signal da pređe u određeno stanje i na taj način pokrene animacija. To znači da animator u svakom trenutku može biti samo u jednom stanju. Na slici 19 se vidi animator korišten za upravljanjem animacijama kretanja igrača.



Slika 19 Animator komponenta igrača

Da bi animator funkcionirao potrebno je dodati skriptu koja će upravljati prijelazima:

```
{
if (Input.GetAxis("Vertical") < 0) anim.SetTrigger("Down");
else if (Input.GetAxisRaw("Vertical") > 0) anim.SetTrigger("Up");
else if (Input.GetAxisRaw("Horizontal") < 0) anim.SetTrigger("Left");
else if (Input.GetAxisRaw("Horizontal") > 0) anim.SetTrigger("Right");
else if (Input.GetAxisRaw("Horizontal") == 0 && Input.GetAxis("Vertical") == 0
)  anim.SetTrigger("Stop");
    }
```

U skripti se provjerava je li igrač pritisnuo tipku za kretanje s pomoću Input.GetAxis („ime osi“) te ako je, pokreće se animacija vezana za smjer u kojem se kreće. Ako igrač nije pritisnuo tipku ili je otpustio tipku za kretnju šalje se animatoru poruka da prijeđe u stanje u kojem ne prikazuje animacija. Na sličan način se implementira i kretanje igrača.

Prvo se na isti način provjeri i spremi vrijednost u kojoj se igrač želi kretati. Zatim, da bi se dobio efekt kretanja, igračev položaj se mijenja ovisno o smjeru u kojem se želi kretati pomnožen sa brzinom i vremenskim intervalom u kojem se metoda poziva:

```
float h = Input.GetAxisRaw("Horizontal");
float v = Input.GetAxisRaw("Vertical");

gameObject.transform.position = new Vector2(transform.position.x + (h * speed
* Time.deltaTime),transform.position.y + (v * speed * Time.deltaTime));
```

Brzina je veličina koja određuje koliko brzo će se igrač kretati. Vremenski interval (Time.deltaTime) je vrijeme proteklo između dvije sličice. Kako je već spomenuto položaj objekta je određen koordinatnim sustavom te s new Vector2 metodom se stvara novi vektor. Vrijednosti novog vektora su položaj igrača u prošloj sličici promijenjene za dobiveni izračun za obje osi. Uz kretnju igrača su bitni „Box Collider 2D“ i „RigidBody 2D“ koji kako je spomenuto služe za simulaciju fizičkih zakona igrača. Postoje i komponente „Box Collider“ i „RigidBody“, ali ovoj igru nisu potrebni jer je igra 2D.

4.3.1 PlayerUnit

PlayerUnit je skripta usko vezana za igrača, ali zbog potrebe postojanja kroz cijelu igru nije komponenta igrača. Skripta služi kako bi se u njoj spremali podaci o igraču te kao takva ne nasljeđuje „MonoBehaviour“.

```
public static void SetExpNeededForNextLvl()
{
   expForLvlUp = (int)(100 * (Mathf.Pow(unitLvl+1,1.3f)) - 100 * (unitLvl+1));
}

public static void LvlUp()
    {
        unitLvl++;
        if (unitLvl == 2)
            pickUpClass = true;
        damage += UnityEngine.Random.Range(2, 5);
        var hp = UnityEngine.Random.Range(4, 7);
        maxHP += hp;
        currentHP += hp;
        var mana = UnityEngine.Random.Range(3, 7);
        manaAmount += mana;
        maxMana += mana;
        if (unitLvl == 2) return;
        foreach(var skill in Inventory.playerClass.skills)
        {
            if (skill.lvlReq == unitLvl)
                Inventory.AddSkill(skill);
      }
}
```

Skripta sadrži sve podatke o igraču koji se pokretanjem igre postavljaju na početnu vrijednost. Kako bi igrač imao osjećaj da napreduje kroz igru tako za svaki idući nivo treba sakupiti sve više iskustva. Također ako poželi igrati igru od početka osim potrebnog iskustva za idući nivo ostali atributi mu neće biti jednaki kao u prethodnoj igri.

4.3.2 Inventory

Još jedna skripta koja postoji kroz cijelu igru te kao takva nije proizašla iz „MonoBehaviour“. Ova skripta sadrži sve informacije o objektima koje igrač posjeduje. Tu spadaju predmeti

(itemi), vještine, klasa, oprema i zlato. Itemi i vještine su realizirani kao liste objekata u koje se učitavaju objekti koje igrač posjeduje:

```
public static List<ItemObject> playerInventory = new List<ItemObject>();
```

Oprema koju igrač koristi je realizirana kao „Dictionary“ (hrv. Rječnik). U rječnik se spremaju parovi „Ključ->Vrijednost“ te jedan ključ može posjedovati samo jednu vrijednost. Na taj način je igrač ograničen na samo jedan predmet koji može nositi u ruci, na glavi,…

U koda se provjerava postoji li već oprema istog tipa kao ona s kojom se igrač želi opremiti:

```
if (equipedItems.ContainsKey(newEquip.eType.ToString()))
{
   PlayerUnit.damage -= equipedItems[newEquip.eType.ToString()].dmg;
   PlayerUnit.maxHP -= equipedItems[newEquip.eType.ToString()].hp;
   PlayerUnit.currentHP -= equipedItems[newEquip.eType.ToString()].hp;
   AddItem(equipedItems[newEquip.eType.ToString()]);
   equipedItems.Remove(newEquip.eType.ToString());
}
equipedItems.Add(newEquip.eType.ToString(), newEquip);
PlayerUnit.damage += equipedItems[newEquip.eType.ToString()].dmg;
PlayerUnit.maxHP += equipedItems[newEquip.eType.ToString()].hp;
PlayerUnit.currentHP += equipedItems[newEquip.eType.ToString()].hp;
```

Ako postoji prvo se igraču umanje atributi za vrijednost stare opreme te se stara oprema premjesti u listu predmeta koje igrač posjeduje. Neovisno o tome je li igrač imao opremu istog tipa, igrač se oprema novom opremom te mu se povećavaju atributi za vrijednost nove opreme.

4.3.2.1 Predmeti

Predmeti (itemi) su objekti koji proizlaze iz klase „ScriptableObject“ te kao takvi ne mogu biti komponente objekata na sceni. „ScriptableObject“ objekti služe primarno kao spremište podataka. Ti objekti se spremaju kao assetsi te kao takvi se mogu učitavati u igru bez potrebe stvaranja objekta na sceni koji će ih predstavljati. Za razliku od običnih klasa ne moraju se stvarati objekti koji će sadržavati skriptu. To olakšava pronalazak podataka jer se ne traži na sceni objekt koji ih sadrži već ih se direktno učitava iz igre.

```
public abstract class ItemObject : ScriptableObject
{
    public string itemName;

    public string description;
    public enum ItemType
    {
        Default,
        Equipment
    }
    public ItemType type;
    public Sprite sprit;
    public int price;
```

```
    public virtual void Use()
    {
        int index = Inventory.playerInventory.IndexOf(this);

        Inventory.itemQuantity[index]--;
        if (Inventory.itemQuantity[index] == 0)
        {
            Inventory.itemQuantity.RemoveAt(index);
            Inventory.playerInventory.RemoveAt(index);
        }
    }
}
```

Uz nasljeđivanje „ScriptableObject“ klase predmeti su apstraktna klasa. Korištenjem ključne riječi „abstract“ onemogućuje se stvaranje predmeta koji ne posjeduju tip (type) i omogućuje se korištenje virtualne metode „Use“ u svim klasama koje nasljeđuju „ItemObject“. Virtualna metoda je zajednička svim izvedenim klasama klase „ItemObject“, te uklanja potrebu da se isti kod piše u svakoj izvedenoj klasi. Kad se predmet iskoristi, smanjuje se količina tog predmeta koju igrač posjeduje. Ako je količina nakon što se predmet iskoristi jednaka nuli taj se predmet uklanja iz „Inventory“ klase. U igri postoje dva tipa predmeta. To su „Default“ i „Equipment“. Za oba tipa predmeta stvara se nova klasa koji nasljeđuje „ItemObject“. „Default itemi sadrže još i vrijednosti zdravlja i mane koji se obnove igraču prilikom korištenja:

```
public class DefaultItem : ItemObject
{
    public int heal;
    public int mana;
    private void Awake()
    {
        type = ItemType.Default;
    }

    public override void Use()
    {
        base.Use();
        PlayerUnit.HealUnit(this.heal,this.mana);
    }
}
```

Osim toga, prilikom korištenja predmeta također se dodaje dodatna funkcionalnost te se koristi ključna riječ override. To omogućuje da se nadjača virtualna metoda „Use“ iz „ItemObject“ klase. Uz nadjačavanje se izvodi i kod iz virtualne metode sa „base.Use()“ . Osim toga se igraču obnavlja zdravlje i mana za vrijednosti koje predmet sadrži. Drugi tip predmeta je „Equipment“. Sadrži vrijednosti zdravlja i štete koje se dodaju igraču ukoliko se igrač opremi s istim. Dijele

se u četiri podtipa: Head, Chest, Legs, Hands. Ti podtipovi su ključevi u rječniku klase „Inventory“ koji služi kao spremište o opremljenim itemima:

```
public enum EquipType
    {
        Head, Chest, Legs, Hands
    }
    public EquipType eType;
    public int dmg;
    public int hp;
```

Kao i default itemi i Equipment sadrže override Use metodu s kojom se nadjačava virtualna metoda roditelja. Pozivanjem Use metode kod ovih predmeta se izvodi virtualna metoda roditelja te metoda „Equip“ klase „Inventory“:

```
public override void Use()
    {
        base.Use();
        Inventory.Equip(this);
    }
```

Equipment sadrži i metodu „StatChange“ koja služi za prikaz promjene igračevih atributa ako se opremi sa odabranim predmetom. Pojašnjenje ove metode se nalazi u poglavlju o „Pause Game“.

```
public void StatChange()
    {

        if (Inventory.equipedItems.ContainsKey(this.eType.ToString()))
            EquipmentStatDifferenceShow.equipmentStatDifferenceShow.ShowStatDi
fferences(this.hp, this.dmg, Inventory.equipedItems[this.eType.ToString()].hp,
 Inventory.equipedItems[this.eType.ToString()].dmg);
        else
            EquipmentStatDifferenceShow.equipmentStatDifferenceShow.ShowStatDi
fferences(this.hp, this.dmg);
    }
```

4.3.2.2 Klasa igrača

Igrač sakupljanjem iskustva i podizanjem svojeg nivoa može otključati klase. Klase su implementirane na isti način kao i predmeti te kao takvi nisu vezani za objekt igrača već postoje u igri:

```
public abstract class ClassObject : ScriptableObject
{
    public string cName = "New Classe";
    public List<SkillObject> skills = new List<SkillObject>();
    public int changeHP;
    public int changeDMG;
```

```
    public string description;

}
```

Klase ne sadrže podklase te postoji samo jedna skripta koja ih realizira. Klase sadrže informacije o imenu klase, zdravlju i šteti koje pojačavaju igrača, opisa, te liste vještina koje igrač može naučiti u toj klasi.

4.3.2.3 Vještine

Vještine su implementirane na identičan način. Sadrže informacije o potrebnom nivou igrača koji je potreban da bi se otključale, šteti koju igrač nanosi protivniku, te potrebnoj mani koju igrač treba imati kako bi ih mogao iskoristiti:

```
public abstract class SkillObject : ScriptableObject
{
    public string skillName = "new skill";
    public int skillDmg;
    public string description;
    public Sprite img;
    public int lvlReq;
    public int manaCost;
}
```

4.4 Pause Game

Kako bi igrač imao pregled nad atributima, predmetima te vještinama napravljen je i „Pause game“ (hrv. Pauza) sistem. Igrač pritiskom na tipku „escape“ aktivira platno pauze. S obzirom da ne postoji način za u potpunosti pauzirati igru, na korisniku je da osmisli svoj sistem kako bi prividno to napravio. Jedan od načina za to je u „Update“ metodi napraviti provjeru. Provjerava se je li igrač pritisnuo tipku „escape“ te nalazi li se trenutno u situaciji u kojoj je dozvoljena pauza:

```
void Update()
    {
        if (Input.GetKeyDown("escape") && canPause)
        {


            if (GamePaused==0)
            {
                PauseGameplay();
                GamePaused = 1;
            }
            else
            {
                ContinueGame();
```

```
                GamePaused = 0;
            }
        }
    }
```

Ako se uvjeti ispune slijedi daljnja provjera. Iduća provjera provjerava je li igra već pauzirana, te ako nije se poziva metoda „PauseGameplay“ i postavlja vrijednost varijable “GamePaused“ na 1. Ta vrijednost govori je li igra pauzirana ili nije.

Metoda „PauseGameplay“ postavlja brzinu kojom vrijeme protječe u igri na 0 i aktivira platno pauze:

```
private void PauseGameplay()
    {
        Time.timeScale = 0;
        pausePanel.SetActive(true)
    }
```

Slika 20 prikazuje aktivirano platno. Svaka scena svijeta sadrži identično platno te se pokretanjem svake od njih platno deaktivira. Razlog tome je osiguravanje da se objekti sa platna učitaju u skripte prilikom pokretanja scene.

Slika 20 Pause game platno

Prilikom aktiviranja pauze metodom „OnEnable“ se učitavaju vrijednosti iz „Inventory“ klase. Pauza se sastoji od nekoliko ploča od kojih je prilikom pokretanja vidljiva samo jedna. Na njoj su prikazani podaci o igraču te sprite koji prikazuje igrača. Osim te ploče na platnu se nalaze nekoliko gumbova sa različitim funkcionalnostima. Pritiskom na prvi gumb se prikazuje lista itema koje igrač posjeduje.

```
Button button = Instantiate(itemButton) as Button;
button.gameObject.SetActive(true);
```

Itemi i vještine su prikazani korištenjem gumba. Skripta „PlayerOptionsControl“ sadrži referencu na te gumbe. Gornji kod se poziva za svaki item kojeg igrač posjeduje. Gumb button označava da će item biti prikazan kao gumb. „Instantiate“ metoda inicijalizira kopiju već napravljenog gumba na sceni koji služi za prikaz itema. Originalni gumb je sakriven na sceni jer služi samo kao plan po kojom se prikazuju itemi. Kopija gumba se aktivira jer se sve kopije međusobno razlikuju te svaka prikazuje po jedan item koji igrač posjeduje. Potom se mijenja vrijednosti gumba na vrijednosti itema kojeg se inicijalizira.

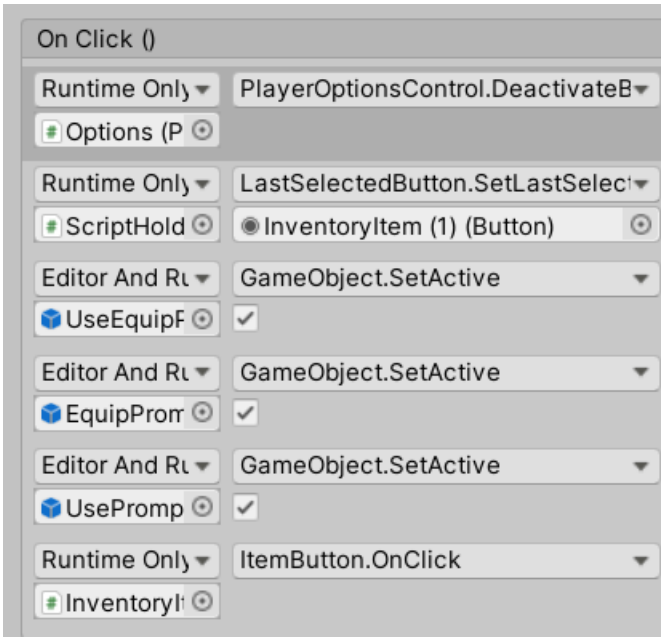
Osim inicijalizacije i aktivacije kopije, postavlja se roditelj na vrijednost roditelja originala:

```
button.transform.SetParent(itemButton.transform.parent, false);
inventoryButtons.Add(button);
```

Sa vrijednošću „false“ se osigurava da pozicija svakog gumba bude različita. To je osigurano korištenjem „scrollview“ komponente koja služi za prikaz veće količine objekata. Prikazuje ih kao listu kroz koju se može kretati (slika 21). Gumbi se nalaze u djetetu „Content“ koji je postavljen kao roditelja gumba. Bitno je da roditelj gumba bude „Content“ zbog komponente „Grid Layout Group“. Komponenta služi za automatsko postavljanje djece u rešetku. Na slici 20 se vidi da korištena rešetka postavlja djecu u 2 stupca. Osim liste prikazuje se i opis trenutno odabranog itema.

Slika 21 Itemi

Osim pregleda itema kojih igrač posjeduje, na pauzi se mogu iskoristiti itemi, te se opremiti sa opremom.



Slika 22 Item OnClick pozivi

Kako bi se osigurala funkcionalnost korištenja itema dodane su metode koje se izvode prilikom pritiska na item (slika 22). Prvo se poziva metoda „DeactivateButton“ koja služi za postavljanje zastavice interaktivnosti itema na 0. Na taj način igrač ne može slučajno pritisnuti 2 itema odjednom. Potom se postavlja posljednji item koji je stisnut. Metoda prima jedan argument te ako se kao argument postavi gumb koji poziva metodu Unity osigurava da sve kopije referenciraju sebe, a ne original. Potom se aktivira ploča na kojoj se nalaze gumbi za korištenje itema. Na posljetku metodom „OnClick“ se provjerava tip itema koji je pritisnut:

```
public void OnClick()
    {
        if (item.type == ItemObject.ItemType.Default)
        {
            GameObject.FindGameObjectWithTag("Equip").SetActive(false);
            GameObject.FindGameObjectWithTag("Use").GetComponentInChildren<But
ton>().Select();
        }
        else
        {
            GameObject.FindGameObjectWithTag("Use").SetActive(false);
            GameObject.FindGameObjectWithTag("Equip").GetComponentInChildren<B
utton>().Select();
        }
    }
```

Ako je odabran item tipa „Default“ deaktivira se ploča koja sadrži gumbe za opremiti igrača itemom. Potom se postavlja gumb za korištenje itema kao odabranog gumba na platnu. Ukoliko je odabrana oprema i igrač se odluči opremiti tim itemom poziva se metoda „ChangeStats“ koja osvježi podatke na ploči na kojoj su prikazani igračevi atributi.

Na ovaj način se promijeni vrijednost teksta koji prikazuje igračev nivo, zdravlje i manu:

```
        lvl.text = PlayerUnit.unitLvl.ToString();
        hp.text = PlayerUnit.currentHP.ToString() + "/"
        + PlayerUnit.maxHP.ToString();
        mana.text = PlayerUnit.manaAmount.ToString() + "/" +
        PlayerUnit.maxMana.ToString();
```

Kako su gumbi pozicionirani svuda po platnu korištene su tri vrste navigacije. Prva, horizontalna, se koristi za gumbe koji su vidljivi prilikom pokretanja pauze. Druga, automatska, se koristi za gumbove korištenja itema, prilikom prikaza tih gumba svi drugi su ili deaktivirani ili ne interaktivni. Treća, eksplicitna, je korištena za navigaciju kroz iteme i vještine. Eksplicitna navigacija služi za postavljanje vlastite navigacije. Razlog za korištenje eksplicitne navigacije je taj što kad je previše gumba na sceni, Unity ne može predvidjeti navigaciju koja se želi postići.

```
navigation = items[i ].GetComponent<Button>().navigation;
navigation.mode = Navigation.Mode.Explicit;
navigation.selectOnLeft = items[i - 1];
items[i ].navigation = navigation;
```

Eksplicitna navigacija se postavlja metodom „ResetNavigation“. Način na koji se postavlja navigacija je slijedeći. Prvo se preuzme navigacija gumba tako da se za svaki gumb iz liste gumba koji predstavljaju iteme dohvati komponenta „Button“. To se radi sa pozivom metode nad svakim članom liste. „items[i].GetComponent<Button>()“ dohvaća komponentu „Button“ te se od cijele komponente preuzme samo vrijednost navigacije sa .navigation. Potom se mijenja način navigacije na eksplicitni sa „navigation.mode = Navigation.Mode.Explicit“. Zatim se postavlja vrijednost „selectOnLeft“ na prethodni gumb iz liste. Potom se sprema promjena navigacije nazad u gumb. Osim toga prethodnom gumbu na isti način se mijenja vrijednost „selectOnRight“ na trenutni gumb. Na taj način se osiguralo da pritiskom lijevo i desno igrač odabire prethodni odnosno slijedeći gumb. Analogno tome provjerava se postoji li gumb koji se na sceni nalazi poviše trenutnog gumba, te ako postoji mijenja mu se vrijednost „selectOnUp“ na gumb koji se nalazi dva mjesta prije trenutnog u listi. Također se tom gumbu promjeni vrijednost „selectOnDown“ na trenutni. Osim toga posljednjem gumbu ne listi se mijenja vrijednost na gumb „back“ koji služi za deaktivaciju svih otvorenih ploča te vraćanje pauze u originalno stanje. Tokom pauze se u „Update“ metodi skripte „PlayerOptionsControl“ cijelo vrijeme provjerava koji objekt je trenutno odabran.

Prva provjera koja se radi je provjera ako je odabran objekt, te ako nije prekida se taj poziv metode:

```
if (!eventSystem.currentSelectedGameObject) return;
            if (eventSystem.currentSelectedGameObject.GetComponent<ItemButton>
() != null)
            {
```

Ako postoji odabrani objekt idućom se provjerom ispituje radi li se o itemu ili vještini.
Ukoliko je riječ o itemu provjerava se je li default ili equipment item. Ako je equipment item,

poziva se funkcija „StatChange“ koja na ploči sa igračevim atributima prikazuje promjenu atributa ukoliko se igrač opremi s njim.

```
if (eventSystem.currentSelectedGameObject.GetComponent<ItemButton>().item.ty
      == ItemObject.ItemType.Equipment)
                {
                    Resources.Load<EquipmentItem>("Items/" + eventSystem.curre
ntSelectedGameObject.
                    GetComponent<ItemButton>().item.itemName).StatChange();
                }
```

Ukoliko je običan item, gase se objekti koji prikazuju promjenu atributa. U oba slučaja se poziva metoda „Scrollin“.

```
scroll.Scrolling(inventoryButtons, eventSystem.currentSelectedGameObject.GetCo
mponentInParent<ScrollRect>());
```

Unity nema ugrađen način za skrolanje korištenjem tipkovnice. Stoga korisnik mora izgraditi svoj sistem koji će koristiti za to.

```
public void Scrolling(List<Button> listOfButtons, ScrollRect scrollRect)
    {
        selected = EventSystem.current.currentSelectedGameObject;
        float scrollChange = (float) ((listOfButtons.IndexOf(selected.GetCompo
nent<Button>()) / 2))
        / Mathf.CeilToInt((float)((listOfButtons.Count - 6) / 2f));

        if (1 - scrollChange + (1f / ((int)((listOfButtons.Count - 4) / 2))) <
 0)
            scrollRect.verticalScrollbar.value = 0;
        else if (1 - scrollChange + (1f / ((int)((listOfButtons.Count - 4) / 2
))) > 1)
            scrollRect.verticalScrollbar.value = 1;
        else
            scrollRect.verticalScrollbar.value = 1 - scrollChange  +(1f / ((in
t)((listOfButtons.Count - 4) / 2)));
    }
```

Metoda „Scrolling“ prima dva argumenta, prvi je lista tipki kroz koju se želi skrolati, drugi je komponenta „ScrollRect“ koji sadrži vrijednosti o skrolu kojeg sadrži trenutni „Scrollview“. Prvi korak je spremiti vrijednost trenutno odabranog objekta. Potom se izračuna za koliko je potrebno promijeniti vrijednost vertikalnog skrola. Vrijednosti skrola su između 0 i 1, 0 predstavlja dno liste dok 1 predstavlja vrh.

```
        float scrollChange = (float) ((listOfButtons.IndexOf(selected.GetCompo
nent<Button>()) / 2))
        / Mathf.CeilToInt((float)((listOfButtons.Count - 6) / 2f));
```

Promjena vrijednosti skrola se dobije tako da se indeks itema na listi podijeli sa 2. S time se dobije u kojem redu liste je prikazan. Potom se dijeli brojem itema na listi umanjenog za 6 i podijeljenog sa 2. Potom se provjerava ako je nova vrijednost skrola manje ili veća od 0 tj. 1 te ako je se postavlja na 0 tj. 1. Inače se to ne bi trebalo raditi, ali postoji greška u računu koja dozvoljava da vrijednost skrola prijeđe granice. Vrijednost skrola se postavlja na novu vrijednost tako da se od 1 oduzme izračunata promjena i doda joj se broj itema umanjen sa 4 i podijeljen sa 2. Razlog tog dodavanja je kako bi se odabrani objekt nalazio u sredini liste tokom skrolanja. Da se izostavi uvećavanje vrijednosti odabrani item bi uvijek bio prikazan na vrhu liste osim zadnja 4 itema koji bi bili prikazani na svojim položajima.

Prikaz vještina je napravljen na identičan način kao itemi, sa razlikom da se gumb ne može pritisnuti jer su vještine napravljene samo za borbu. Na pauzi osim itema i vještina se nalazi i tipka za aktivaciju ploče na kojoj se nalazi popis trenutno opremljenih itema, tipka za spremiti igru, tipka za učitavanje spremljene igre te tipka za povratak na scenu glavnog izbornika.

4.4.1 Save/Load Game

Unity ima već ugrađene načine za serializirati podatke. Naravno ograničen je na osnovne tipove podataka te zbog toga nije moguće serializirati cijelu klasu već se moraju sve vrijednosti spremiti zasebno. Ukoliko klasa sadrži tip podataka koji Unity ne može serializirati treba se osmisliti način na koji će se spremiti ti podatci te kasnije učitati natrag u igru. Za spremanje podataka napravljena je klasa „SaveData“ koja služi kao spremište podataka koji se žele spremiti ili učitati nazad u igru. Svi tipovi podataka te klase moraju biti jednostavni. Spremanje vrijednosti igračevih atributa se vrši njihovim kopiranjem u ovu klasu.

```
        public int unitLvl = PlayerUnit.unitLvl;
        public int damage = PlayerUnit.damage;
        public int maxHP = PlayerUnit.maxHP;
```

Do problema dolazi kada se žele spremiti podaci o itemima i vještinama. Unity nije u mogućnosti serializirati objekt koji ih predstavlja te ih se zbog toga treba spremiti na drugi način. Podaci o itemima se spremaju tako da se napravi lista stringova koja sadrži imena svih itema koje igrač posjeduje:

```
public List<string> playerItems=new List<string>();
foreach (var item in Inventory.playerInventory)
            {
                this.playerItems.Add(item.itemName);
            }
```

Prilikom kreiranja pomoćne klase koja predstavlja podatke koje treba spremiti, kreira se prazna lista stringova. Potom se za svaki item kojeg igrač posjeduje spremi njegovo ime u listu. Analogno tome spremi se i lista vještina koje igrač posjeduje. Još jedan tip podataka koje Unity ne može serializirati je rječnik.

```
foreach( KeyValuePair<string,EquipmentItem> item in Inventory.equipedItems)
            {
```

```
                playerEquip.Add(item.Value.itemName);
            }
```

Analogno itemima i vještinama, oprema se sprema u listu stringova. Sprema se na način da se za svaki tip opreme koju igrač ima spremi samo ime itema. To je dovoljno iz razloga što igrač može imati samo jedan isti tip opreme. Također nije moguće spremiti vektor koji predstavlja igračevu poziciju. Igračeva pozicija se sprema tako da se vrijednost x i y osi spreme u zasebne varijable. I za kraj se sprema ime scene na kojoj se igrač nalazi prilikom pritiska gumba „Save“.

Pritiskom gumba „Save“ na pauzi se poziva metoda „SaveGame“ koja prvo učitava podatke koje je potrebno spremiti:

```
public void SaveGame()
    {
        SaveData save = CreateSaveGame();
        BinaryFormatter bf = new BinaryFormatter();
        FileStream file = File.Create(Application.persistentDataPath
        + "/gamesave.save");
        bf.Serialize(file, save);
        file.Close();
    }
```

Potom stvara binarni format i datoteku koja će sadržavati binarni zapis podataka. Potom binarni format oblikuje podatke koji se spremaju u binarni zapis. Korisna stvar ovog načina spremanja podataka je što pruža nivo zaštite nad podacima. Za kraj se zatvara datoteka u kojoj su podatci spremljeni. Analogno se izvodi i metoda „LoadGame“ koja se poziva pritiskom na gumb „Load“. Prvo se učitaju podaci iz spremljene datoteke te se svi jednostavni tipovi podataka kopiraju igru.

Složeni podaci se učitavaju na način da se za svako spremljeno ime objekta iz igre učita taj objekt u odgovarajuću varijablu. Na primjeru je prikazano kako se za svako spremljeno ime itema koje je igrač posjedovao prilikom spremanja podataka iz igre učita objekt sa tim imenom i doda ga se u klasu „Inventory“:

```
foreach (var item in this.playerInventory.playerItems)
        {

            Inventory.AddItem(Resources.Load<ItemObject>("Items/" + item));
        }
```

Važna stavka kod učitavanja podataka iz igre je da su podaci dobro organizirani kako bi se znalo odakle učitati određeni podatak. Grupiranjem itema u mapu „Items“ omogućuje se da Unity učita iteme samo iz te mape. Da datoteke nisu organizirane, Unity bi morao za svaku datoteku provjeriti njen tip, pa ako jesu, učitati je.

4.5 Non-player charater(NPC)

Non-plaer charater (NPC) su svi likovi u igri kojima igrač ne može upravljati. Mogu se podijeliti u dvije vrste, prijateljski i neprijateljski likovi. Prijateljski likovi su likovi koji igrača usmjeravaju ili mu nude uslugu. U „Shadow Tomb Valley“ od prijateljskih likova se nalaze samo oni koji nude usluge. Usluge koje nude su usluge noćenja u gostioni te kupovine itema u dućanu. Od neprijateljskih likova postoje obični protivnici koji igrača napadaju dok se nalazi na scenama koje predstavljaju vanjski svijet, te šef protivnik koji čuva ulaz u „Shadow Tomb“. Skripta „NPC“ sadrži referencu na platno na kojem se prikazuju podaci listi stringova, koji predstavljaju dijalog kojeg lik posjeduje. „NPC“ klasa proizlazi iz „MonoBehaviour“ te kao takva ima pristup metodama za detekciju kolizije.

S pomoću te metode provjerava se je li igrač taj koji je izazvao koliziju te ako je postavlja se zastavica „triggerNpc“ na istinu. Ona služi kao jedna od provjera za započeti dijalog:

```
    private void OnTriggerEnter2D(Collider2D collision)
    {
        if (collision.CompareTag("Player"))
        {
            this.triggerNpc = true;
        }
    }
```

U metodi „OnCollisionStay2D“ koja se poziva svaki okvir u kojoj je realizirana kolizija provjerava se može li igrač započeti razgovor, je li pritisnuo tipku za započeti razgovor, te je li igrač taj koji je u koliziji sa NPC-em. Ako su svi uvjeti ispunjeni onda se vrijednost koja označava kojom brzinom vrijeme protječe u igri postavlja na 0, te sa time prividno pauzira igra.

```
if (this.triggerNpc && Input.GetButtonDown("Submit") &&
            collision.collider.CompareTag("Player"))
        {
            Time.timeScale = 0;
            StartCoroutine(SetChat());
        }
```

Osim toga poziva se koroutina „SetChat“. Svaki NPC ima implementiranu vlastitu koroutinu „SetChat“ sa zajedničkim dijelom aktivacije platna na kojem se dijalog prikazuje.

4.5.1 Gostioničar

Gostioničar se nalazi u sjeveroistočnom dijelu grada. Pruža uslugu noćenja u gostioni po cijeni od 30 zlata. Prilikom pokretanja dijaloga sa gostioničarom pokazuje se poruka igraču u kojoj ga se pita želi li prenoćiti po cijeni od 30 zlata. Igrač ima na izbor dvije opcije, prihvatiti ponudu ili je odbiti.

Provjerava se je li igrač odabrao želi li prenoćiti i ako nije onda se nakon sekunde ponovno provjeri:

```
        while (yesNoPrompt.activeInHierarchy)
        {
            yield return new WaitForSecondsRealtime(1);
        }
```

Ovisno o odabranom odgovoru se dijalog mijenja. Ako je igrač odabrao da želi prenoćiti prvo se provjerava posjeduje li igrač dovoljno zlata za platiti gostioničaru. Ako ne posjeduje, dijalog se postavlja na poruku koja govori igraču da ne posjeduje dovoljno zlata. Ako posjeduje, pokreče se koroutina „Wait“ koja služi kako bi se pričekalo da se animacija noćenja izvrši. Animacija noćenja je napravljena tako da se crnoj slici mijenja prozirnost iz prozirne do neprozirne pa natrag u prozirnu. Tako je dobiven efekt „prolaska“ vremena. Osim koroutine „Wait“ pokreče se i metoda „SpendNight“.

```
if (Inventory.goldAmount < nightSpendCost)
                i = 2;
            else
            {
                StartCoroutine(Wait());
                SpendNight();
            }
```

Ona aktivira animaciju noćenja, smanjuje zlato koju igrač posjeduje za cijenu noćenja, te u potpunosti obnavlja igračevo zdravlje i manu.

```
        img.GetComponentInChildren<Animator>().SetTrigger("Trigger");
        Inventory.goldAmount -= nightSpendCost;
        PlayerUnit.HealUnit(PlayerUnit.maxHP, PlayerUnit.maxMana);
```

Potom se postavlja dijalog na pozdrav gostioničara, te se čeka pritisak tipke „space“ za zatvaranje dijaloga. Prilikom zatvaranja dijaloga poziva se metoda „EndDialogue“ koja deaktivira platno na kojem je prikazan dijalog, vraća brzina protjecanja vremena na 1, te omogućuje pauziranje igre. Osim toga korištenjem Unityjeve klase „EventSystem“ [16] postavlja se vrijednost trenutno odabranog elementa na null vrijednost. „EventSystem“ služi za rad sa događajima unutar scene. Svaka promjena odabira objekta, pritisak objekta itd. su događaji koje Unity može obraditi.

```
    public void EndDialogue()
    {
        dialoguePanel.SetActive(false);
        EventSystem.current.SetSelectedGameObject(null);
        Time.timeScale = 1;
        PauseGame.canPause = true;
    }
```

4.5.2 Prodavač

Prodavač se nalazi na istoj sceni kao gostioničar te su sve funkcionalnosti bitne za dijalog napravljene na isti način. Prodavač nudi opciju za kupovinu i prodaju itema. Ovisno o odabranoj

opciji aktivira se ploča koja prikazuje koje je iteme moguće kupiti odnosno prodati. Ovisno o odabiranoj opciji se poziva odgovarajuća metoda koja postavlja izgled platna.

4.5.2.1 Kupovina

Odabirom opcije za kupovine aktivira se platno te se postavljaju itemi za kupnju. Itemi za kupnju su realizirani na sličan način kao i igračevi itemi prikazani na pauzi. Razlika je što se ovdje učitavaju svi itemi u igri.

Varijable „merchandise“ i „merchandiseCopy“ sadrže sve iteme u igri koji su dohvaćeni pomoću metode „Resources.LoadAll<ItemObject>(„Items“):

```
        merchandise = new List<Button>();
        merchandiseCopy = new List<Button>();
        ItemObject[] items;
        items = Resources.LoadAll<ItemObject>("Items");
```

Pozivanjem te metode dohvaćaju se objekti koji predstavljaju iteme. Kako su itemi u igri predstavljeni gumbima potrebno je te objekte dodati gumbu. Varijabla „merchandise“ predstavlja gumbe koje je moguće pritisnuti, dok varijabla „merchandiseCopy“ predstavlja kopiju gumba „merchandise“ koje igrač ne može pritisnuti. Nakon dohvaćanja svih item objekata iz igre, isti se spremaju u obje liste. Nakon inicijalizacije podataka mijenja se boja gumbima u listi kopija.

Boja se mijenja na način da se prvo učita komponenta boja sa gumba. Sa color.selectedColor = Color.gray se mijenja boja kada je gumb odabran u sivu. Po završetku mijenjanja boja iste je potrebno spremiti nazad u gumb:

```
            var color = button2.colors;
            color.selectedColor = Color.gray;
            color.pressedColor = Color.gray;
            button2.colors = color;
```

Po tom se poziva metoda „SetMerchandiseNavigation“. Kako je već opisano Unity ne može predvidjeti željenu navigaciju pa je potrebno implementirati vlastitu. Ovdje se također koristi eksplicitna navigacija. Prvi korak kod postavljanja navigacije je odrediti koji gumb možemo odabrati. Određuje se na način da varijabla „merchandise“ predstavlja iteme koje igrač može kupiti, dok njena kopija predstavlja iteme koje igrač ne može kupiti. Provjerava se ako trenutni item košta više od količine zlata koju igrač posjeduje. Ako košta, onda se gasi mogućnost interakcije sa originalnim itemom, a pali se za kopiju. Ako ne košta, kopiji se ugasi ta mogućnost:

```
for (int i = 0; i < merchandise.Count; i++)
        {
            merchandise[i].GetComponent<Button>().interactable = true;


            if (merchandise[i].GetComponent<ItemButton>().item.price > Invento
ry.goldAmount)
            {
```

```
                merchandise[i].GetComponent<Button>().interactable = false;
                merchandise[i].GetComponentsInChildren<Button>()[1].interactab
le = true;

            }
            else
                merchandise[i].GetComponentsInChildren<Button>()[1].interactab
le = false;
        }
```

Potom se za svaki item, na osnovu kojem gumbu je upaljena mogućnost interakcije, preuzima varijabla koja predstavlja navigaciju. Sa „GetComponent<Button>():interactable“ se dohvaća zastavica o interakciji te ako je postavljena na true se uzima original, inače kopija:

```
for (int i = 0; i < merchandise.Count; i++)
            {

                if (i > 0)
                {
                    if (merchandise[i].GetComponent<Button>().interactable)
                        navigation =merchandise[i].navigation;
                    else
                        navigation =merchandise[i].GetComponentsInChildren<But
ton>()[1].navigation;
```

Daljnja izrada navigacije je identična kao na prikazu itema koje igrač posjeduje koja je opisana u poglavlju Pause Game. Razlika je jedino što se bira između originala i kopije itema na osnovi zastavice interaktivnosti. Interakcija sa itemima je identična kao i interakcija opisana u poglavlju Pause Game. Razliku čine kopije itema koji služe za prikaz itema koje si igrač ne može priuštiti. Kako igrač ne može kupiti te iteme tako ne postoji događaj koji se izvodi prilikom pritiska na taj item. Prilikom kupnje itema poziva se metoda „BuyItem“. Metoda prvo pozivom metode „AddItem“ u skripti „Inventory“ dodaje kupljeni item u listu igračevih itema. Potom se igraču oduzima zlato koje je potrebno za kupnju tog itema. Te se ponavlja postavljanje ploče koja prikazuje iteme za kupnju i mijenjanje ploče na kojem su prikazani igračevi atributi, kako bi se osvježio prikaz zlata kojeg igrač ima.

```
        Inventory.AddItem(LastSelectedButton.item.
        GetComponent<ItemButton>().item);
        Inventory.goldAmount -
= LastSelectedButton.item.GetComponent<ItemButton>().item.price;
        SetUpMerchandise();
        ChangeStats();
```

4.5.2.2 Prodaja

Odabirom opcije prodaje se poziva metoda „SetUpSellableItems“. Za razliku od metode „SetUpMerchandise“ ovdje je potrebno prvo osigurati da se unište svi objekti koji predstavljaju iteme, koje je igrač u mogućnosti prodati, sa scene:

```
foreach(var item in sellableItems)
        {
            Destroy(item.gameObject);
        }
```

To je potrebno iz razloga što se i ova metoda poziva više puta, ali za razliku od „SetUpMerchandise“ neće uvijek imati isti broj itema za prikazati. Potom se za svaki item koji igrač posjeduje inicijalizira gumb, te mu se postavi navigacija na već opisan način. Razlika između gumba koji predstavlja item za kupiti i gumba koji predstavlja item za prodati je što na gumbu koji predstavlja item za kupiti piše njegova cijena, dok na gumbu koji predstavlja item za prodati piše količina koju igrač posjeduje. Osnovna funkcionalnost im je ista. Pritiskom na gumb se otvara izbornik koji nudi opciju prodaje ili odustajanja ukoliko je igrač krivo stisnuo gumb. Ako se igrač odluči prodati item poziva se metoda „SellItem“:

```
    public void SellItem()
    {
        Button item = LastSelectedButton.item;
        ItemObject itemItem = item.GetComponent<ItemButton>().item;
        Inventory.itemQuantity[Inventory.playerInventory.indexOf(itemItem)]--;
        if (Inventory.itemQuantity[Inventory.playerInventory.indexOf(itemItem)
]==0)
        {
            Inventory.itemQuantity.Remove(Inventory.playerInventory.indexOf(it
emItem));
            Inventory.playerInventory.Remove(itemItem);
        }
        LastSelectedButton.item = back;
        Inventory.goldAmount += itemItem.price / 2;
        SetUpSellableItems();
        ChangeStats();
    }
```

Ta metoda prvo dohvaća zadnje pritisnut item. Zatim se smanji količina tog itema koju igrač posjeduje za jedan. Potom se provjerava ako je količina koju igrač i dalje posjeduje jednaka 0, te ako je se item izbacuje iz liste itema koje igrač posjeduje. Potom se igraču uvećava količina zlata za vrijednost itema podijeljeno sa 2. Na kraju se osvježava prikaz ploča koje prikazuju atribute igrača i itema koje igrač posjeduje za prodaju.

„Update“ metoda prodavača kroz sličnu provjeru kao „Update“ metoda skripte „PlayerOptionsControl“ provjerava ako je trenutno odabrani objekt na sceni iz klase itema, te ako je poziva metodu „Scrolling“ šaljući joj odgovarajuće argumente. Poziv te metode je razlog

zbog kojeg postoji varijabla „merchandiseCopy“, jer kako je već opisano skripta provjerava indeks trenutno odabranog objekta u listi.

4.5.3 Protivnici

U igri postoje dvije vrste protivnika. Prva su obični protivnici koji se susreću u borbi, druga je protivnik šef koji se susreće na zadnjoj sceni svijeta. Prilikom interakcije sa šefom igraču se prikazuje dijalog kojeg šef posjeduje. Taj dijalog kao i ostali dijalozi su napravljeni pozivanjem koroutine „SetChat“. Razlika od ostalih je ta što na platnu dijaloga nema gumba već je prikazan samo dijalog. Kako igrač nema mogućnost daljnje interakcije sa šefom, u koroutini se provjerava ako igrač stisne tipku „space“ te postavlja dijalog na iduću rečenicu iz liste. Provjera se događa sve dok se ne prikaže cijeli dijalog kojeg protivnik šef posjeduje:

```
        while(i<dialogue.Count)
        {
            while(!Input.GetKeyDown(KeyCode.Space))
            {
                yield return null;
            }
            SetDialogue(i);
            i++;
            yield return null;
        }
```

Potom se postavlja zastavica „bossFight“ koja služi za provjeru kod postavljanja scene borbe. Ako je zastavica postavljenja scena borbe će se postaviti za borbu sa „Shadow Kingom“, inače će se postaviti na običnu borbu.

Potom se sprema igračeva pozicija korištenjem klase „PlayerPrefs“ i njezine metode „SetFloat“:

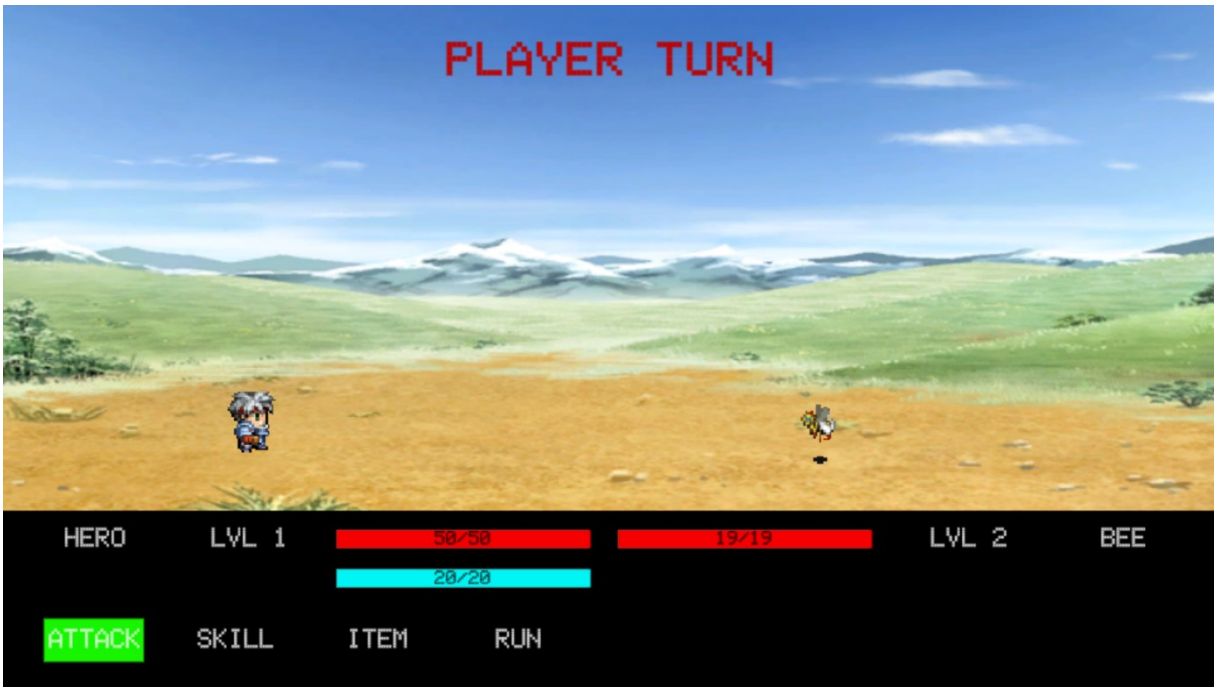
```
        PlayerUnit.bossFight = true;
        PlayerPrefs.SetFloat("x", gameObject.transform.position.x);
        PlayerPrefs.SetFloat("y", gameObject.transform.position.y);
        PlayerPrefs.SetString("_last_scene_", SceneManager.GetActiveScene().na
me);
        SceneManager.LoadScene("Battle1", LoadSceneMode.Single);
```

Klasa „PlayerPrefs“ može spremiti samo jednostavne tipove podataka, te funkcionira na isti način kao rječnik. Podaci su spremljeni kao parovi ključ->vrijednost te za svaki ključ može postojati jedna vrijednost. Ova klasa se također može koristiti za spremanje igre te njeno ponovno učitavanje. Za razliku od binarnog formatora objašnjenog u poglavlju „Save/Load Game“ podaci se ne spremaju u datoteku te nisu zaštićeni. To omogućuje igraču lako mijenjanje vrijednosti spremljene igre. Ovdje se koristi samo za spremanje pozicije i trenutne scene, te igrač nije u mogućnosti mijenjanjem ovih vrijednosti povećati svoje atribute te sebi olakšati prolaz igre. Osim spremanja igračeve pozicije sprema se i ime trenutne scene kako bi se

prilikom završetka borbe mogla ponovo učitati te postaviti igrača na poziciju na kojoj se i nalazio prije borbe.

4.6 Borba

Borba je jedan od najvažnijih elemenata svake igre uloga. Ovdje će se prikazati izgradnja jednostavnog sustava borbe koja se odvija u potezima.



Slika 23 Scena borbe

Slika 23 prikazuje izgled scene borbe. Scena se sastoji od spritea igrača (lijevo), imena igrača, nivo igrača te trake koje prikazuju igračevo zdravlje i manu. Desna polovica scene prikazuje spriteove protivnika, na slici 23 je samo jedan protivnik, ali moguće je imati borbu sa do tri protivnika. Osim spriteova protivnika također su prikazani imena, nivoi te trake zdravlja protivnika. U prikazu scene borbe koriste se dvije skripte, „BattleHUD“ i „BattleSystem“. Skripta „BattleHUD“ je zadužena za vizualizaciju podataka o igraču i protivnicima. Skripta proizlazi iz klase „MonoBehaviour“, ali za razliku od većine takvih skripti, ona ne sadrži ugrađene metode koje „MonoBehaviour“ omogućuje. Razlog tome je što u potpunosti ovisi o događaju u skripti „BattleSystem“. Također ju je potrebno dodati objektima na sceni, a kao što je već spomenuto to se može samo sa skriptama koje proizlaze iz „MonoBehaviour“. Za simulacije borbe je zadužena skripta „BattleSystem“. Koristi kolekciju konstanti kao prikaz stanja u kojem je borba trenutno.

```
public enum BattleState { START, PLAYERTURN, ENEMYTURN, WON, LOST, WAIT, RUN}
```

Borba se može nalaziti u sedam stanja koji označavaju: početak borbe, igračev potez, protivnički potez, pobjeda, poraz, čekaj, bježanje. Prilikom pokretanja scene borba se postavlja u stanje početka.

```
    void Start()
    {
```

```
        PauseGame.canPause = false;
        state = BattleState.START;
        if (PlayerUnit.bossFight)
            StartCoroutine(SetupBossBattle());
        else
        StartCoroutine(SetupBattle());
    }
```

Osim toga onemogućuje se mogućnost pauziranja igre, te se vrši provjera treba li učitati običnu borbu ili borbu protiv šefa. Opisati će se proces postavljanja obične borbe, jer za razliku od borbe sa šefom, nije uvijek jednak:

```
        clipTime = Audio.audioPlayer.audioSource.time;
        Audio.audioPlayer.audioSource.clip = Resources.Load<AudioClip>("Audio/
battle");
        Audio.audioPlayer.audioSource.time = 0;
        Audio.audioPlayer.audioSource.Play();
```

Na početku postavljanja borbe mijenja se zvuk. Prvo se učita vremenska oznaka trenutnog zvuka kako bi se mogla nastaviti njegova reprodukcija prilikom završetka borbe. „Audio.audioPlayer.audioSource“ predstavlja objekt na sceni zadužen za reprodukciju zvuka. Komponenta „clip“ tog objekta označuje zvuk koji se trenutno reproducira. Mijenja se zvuk kojeg objekt reproducira, učitavanjem drugog zvuka pomoću naredbe „Resources.Load<AudioClip>()“. Potom se vremenska oznaka novog zvuka postavlja na 0 kako bi se osiguralo da se zvuk reproducira od početka. Naredbom „Play“ se pokreće reprodukcija zvuka. Potom se pozivaju metode „SetUpItems“ i „SetUpSkills“ koje učitavaju iteme i vještine koje igrač posjeduje. U listi itema će biti prikazani samo itemi koje igrač može iskoristit, tj. neće biti prikazana oprema koju posjeduje. Vještine su prikazane na identičan način kao itemi koje prodavač prodaje. Ako igrač posjeduje dovoljno mane odabire se original tipka vještine inače se odabire kopija.

```
if (PlayerPrefs.GetString("_last_scene_") == "World")
            background.sprite = Resources.Load<Sprite>("Battle/BackgroundImage
s/worldbkgr");
        else
            background.sprite = Resources.Load<Sprite>("Battle/BackgroundImage
s/forestbkgr");
```

Nakon postavljanja itema i vještina se učitava pozadina na sceni. Učitana pozadina ovisi o sceni na kojoj se igrač nalazio prilikom pokretanja borbe.

```
        enemyPrefLabs = Resources.LoadAll<EnemyUnit>("Battle/Enemies");
        if (PlayerPrefs.GetString("_last_scene_") == "World")
        {
            numberOfEnemies = UnityEngine.Random.Range(0, 1000);
            if (numberOfEnemies < 700)
                numberOfEnemies = 1;
            else if (numberOfEnemies < 900)
```

```
                numberOfEnemies = 2;
            else
                numberOfEnemies = 3;
        }
        else
        {
            numberOfEnemies = UnityEngine.Random.Range(0, 1000);
            if (numberOfEnemies < 500)
                numberOfEnemies = 1;
            else if (numberOfEnemies < 800)
                numberOfEnemies = 2;
            else
                numberOfEnemies = 3;
        }
```

Zatim se učitavaju svi stvoreni protivnici, te bira broj protivnika koji biti prisutan u borbi. Broj protivnika se bira slučajnim odabirom te je na prvoj sceni na kojoj su borbe omogućene, manja šansa za veći broj protivnika.

```
for (int i = 0; i < numberOfEnemies; i++)
        {
            enemyBattleStation[i].SetActive(true);
            enemies.Add(Instantiate(enemyPrefLabs[UnityEngine.Random.Range(0,
3)], enemyBattleStation[i].transform));
```

Nakon što je određeno koliko će protivnika biti aktivira se objekt koji služi za prikaz protivnika. Potom se tom objektu doda komponenta koja sadrži podatke o protivniku tako da se na slučajan odabir od svih učitanih protivnika učita jedan.

```
            if (PlayerPrefs.GetString("_last_scene_") == "World")
                enemies[i].SetStats(UnityEngine.Random.Range(1, 3));
            else
                enemies[i].SetStats(UnityEngine.Random.Range(2, 5));
```

Svaki protivnik sadrži atribute koji predstavljaju njegovo zdravlje, štetu, nivo, te iskustvo i zlato koje igrač dobije pobjedom nad njim. Svaki protivnik ima svoje početne vrijednosti te se metodom „SetStats“ te vrijednosti mijenjaju. Metoda prima kao argument nivo protivnika te na osnovu tog nivoa poveća ostale atribute:

```
    public void SetStats(int lvl)
    {
        this.SetLvl(lvl);
        this.damage += Mathf.RoundToInt(1 + (this.unitLvl - 1) * 0.15f + (1 *
(this.unitLvl - 1)));
        this.maxHP += Mathf.RoundToInt(1 + (this.unitLvl - 1) * 0.15f + (3 * (
this.unitLvl - 1)));
```

```
        this.exp += Mathf.RoundToInt(1 + (this.unitLvl - 1) * 0.15f + (10 * (t
his.unitLvl - 1)));
        this.gold += Mathf.RoundToInt(1 + (this.unitLvl - 1) * 0.15f + (10 * (
this.unitLvl - 1)));
        this.currentHP = maxHP;

    }
```

Nakon što su postavljene vrijednosti atributa koje protivnik posjeduje potrebno je pomoću skripte „BattleHUD“ prikazati neke od njih na scenu.

```
            enemyBattleStation[i].GetComponent<BattleHUD>().SetEnemyHud(enemie
s[i]);
            enemyBattleStation[i].GetComponent<Image>().sprite = enemies[i].Ge
tComponent<Image>().sprite;
        }
        playerBattleStation.GetComponent<BattleHUD>().SetHUD();
```

Taj postupak se ponavlja za svakog protivnika. Nakon što su svi protivnici učitani postavlja se izgled objekata koji predstavljaju igrača korištenjem iste skripte „BattleHUD“. Za kraj postavljanja borbe se postavlja stanje borbe u „PLAYERTURN“ i pokreće se metoda „PlayerTurn“. Metoda služi za osvježavanje prikaza vještina ukoliko je igrač u prijašnjem potezu iskoristio jednu od vještina koju posjeduje. Osim toga tekst na vrhu scene promijeni se u „Player turn“ kako bi se igraču pojasnilo da je njegov red za igru. Za kraj se izvodi funkcionalnost gumba back koji osigurava da će svaki početak igračeva poteza izgledati kao na slici 18. Igrač tokom svog poteza ima izbor želi li napasti protivnika običnim napadom, iskoristiti vještinu, item ili želi li pobjeći iz borbe. Ukoliko igrač odluči napasti protivnika ili iskoristiti vještinu spremamo vrijednost gumba koji je odabran. Potom igrač bira kojeg protivnika želi napasti. Odabirom protivnika poziva se koroutina „PlayerAttack“ koja je zadužena za simulaciju igračevog napada.

```
        state = BattleState.WAIT;
        if (playerAttack.GetComponent<SkillButton>())
        {
            manaChanged = true;
            PlayerUnit.SetMana(-
playerAttack.GetComponent<SkillButton>().skill.manaCost);
            playerBattleStation.GetComponent<BattleHUD>().SetMana(PlayerUnit.m
anaAmount);
            isDead = lastSelectedEnemy.GetComponentInChildren<EnemyUnit>().Tak
eDmg(Mathf.RoundToInt(playerAttack.GetComponent<SkillButton>().skill.skillDmg
* PlayerUnit.damage));
            lastSelectedEnemy.GetComponentInChildren<Animator>().SetTrigger(pl
ayerAttack.GetComponent<SkillButton>().skill.skillName);
        }
        else
        {
```

```
            manaChanged = false;
            isDead = lastSelectedEnemy.GetComponentInChildren<EnemyUnit>().Tak
eDmg(PlayerUnit.damage);
            lastSelectedEnemy.GetComponentInChildren<Animator>().SetTrigger("A
ttack");

        }
```

Koroutina prvo postavi stanje borbe na „WAIT“, tako se osigura da se igračev napad izvede u cijelosti prije nego se borba nastavi. Zatim se provjeri ako je igrač iskoristio vještinu te ako je, postavi se zastavica „manaChange“ i igraču se oduzme mana potrebna za izvođenje te vještine. Neovisno o vrsti napada koju je igrač izveo protivniku se nanosi šteta i osvježi se prikaz protivničkog zdravlja na sceni. Također se pokreće animacija ovisno o napadu kojeg je igrač izveo. Svaki napad odnosno vještina ima svoju animaciju.

```
        if (isDead)
        {
            gold += lastSelectedEnemy.GetComponentInChildren<EnemyUnit>().gold
;
            exp += lastSelectedEnemy.GetComponentInChildren<EnemyUnit>().exp;
            foreach (var enemy in enemyBattleStation)
            {
                if (enemy.GetComponent<Button>() == lastSelectedEnemy)
                {
                    enemies.RemoveAt(enemyBattleStation.IndexOf(enemy));
                    enemyBattleStation[enemyBattleStation.IndexOf(enemy)].GetC
omponent<BattleHUD>().DestroyHud();
                    Destroy(enemyBattleStation[enemyBattleStation.IndexOf(enem
y)].gameObject);
                    enemyBattleStation.Remove(enemy);

                    break;
                }
            }
            if (enemies.Count == 0)
            {
                state = BattleState.WON;
                StartCoroutine(EndBattle());
            }
        }
```

Potom se provjerava je li protivnik kojeg je igrač napao mrtav. Ako je povećavaju se vrijednosti iskustva i zlata koje igrač dobiva prilikom završetka borbe. Također se protivnik briše sa scene te se uklanja njegovo postojanje iz skripte. Ako je to bio zadnji protivnik na sceni borba prelazi u stanje pobjede i poziva se koroutina „EndBattle“. Ako postoji još protivnika na sceni borba prelazi u stanje „ENEMYTURN“ te se poziva odgovarajuća koroutina. Ako se igrač odlučio da ne želi napasti već iskoristiti item poziva se metoda „OnItemUse“ koja osvježava izgled scene kako bi se prikazala obnova igračevog zdravlja odnosno mane. Također poziva se metoda

„SetUpItems“ koja osvježava koje iteme igrač može koristiti u idućem potezu. Igrač ima 33% šanse da pobjegne iz borbe odabirom gumba „Run“. U slučaju da je uspješan stanje borbe prelazi u „RUN“ inače prelazi u „ENEMYTURN“.

Pozivom koroutine „EnemyTurn“ prikazuje se text „Enemy turn“ na sceni kako bi igrač znao da trenutno nije njegov potez već protivnikov. Protivnici imaju opciju samo običnog napada. Protivnički potez se sastoji od nanošenja štete igraču te osvježavanju prikaza igračevog zdravlja na sceni.

```
        if (isDead)
        {
            state = BattleState.LOST;
            StartCoroutine(EndBattle());
        }
        else if (enemyToAttack == enemies.Count-1)
        {
            state = BattleState.PLAYERTURN;
            PlayerTurn();
        }
        else
        {
            enemyToAttack++;
            StartCoroutine(EnemyTurn());
        }
```

Na kraju poteza provjerava se je li igrač mrtav. Ako je borba prelazi u stanje poraza, ako nije postoje dvije mogućnosti. Ako su svi protivnici u borbi napali, započinje igračev potez, te ako nisu svi protivnici napali koroutina se izvodi ponovno kako bi se prikazao napad idućeg protivnika.

```
        if (state == BattleState.RUN)
        {
            Audio.audioPlayer.audioSource.clip = Resources.Load<AudioClip>("Au
dio/main");
            Audio.audioPlayer.audioSource.time = clipTime;
            Audio.audioPlayer.audioSource.Play();
            SceneManager.LoadScene(PlayerPrefs.GetString("_last_scene_"), Load
SceneMode.Single);



        }
```

Koroutina „EndBattle“ se poziva u 3 slučaja. Prvi je ako je igrač uspješno pobjegao iz borbe. Bježanjem iz borbe igrač ne dobiva iskustvo i zlato te je potrebno samo učitati zvuk koji je svirao prije početka borbe i pokrenuti ga od trenutka vremenske oznake koja je preuzeta na početku borbe. Osim toga se pokreće scena na kojoj se igrač nalazio prije borbe. Drugi slučaj je ako je igrač izgubio borbu. Razlika je što se neće pokrenuti scena na kojoj se igrač nalazio prije borbe već se pokreče scena koja označava izgubljenu igru. Na toj sceni igraču se prikazuje

poruka da je izgubio igru te mu se nudi povratak na glavni izbornik. Treći slučaj je ako je igrač pobijedio u borbi.

```
            if (PlayerUnit.bossFight)
            {
                DestroyOnLoad.boss = true;
                PlayerUnit.bossFight = false;
            }
```

Ukoliko je borba bila protiv šefa postavljaju se zastavice koje govore da se šef ne učita na scenu svijeta, te zastavica koja onemogućuje daljnju borbu protiv šefa. Osim toga aktivira se ploča na kojoj se nalaze podaci o iskustvu i zlatu koje je igrač osvojio.

```
for (int j = 0; j < numberOfEnemies; j++)
            {
                int i = UnityEngine.Random.Range(0, 1000);
                if (i < 850)
                    battleRewards[2].gameObject.SetActive(false);
                else if (i < 950)
                {
                    battleRewards[2].gameObject.SetActive(true);
                    var item = Resources.LoadAll<DefaultItem>("Items");
                    var itemToPickUp = item[Random.Range(1, item.Length)];
                    Inventory.AddItem(itemToPickUp);
                    battleRewards[2].text += "An enemy droped  " + itemToPickU
p.itemName+"\n";
                }
                else
                {
                    battleRewards[2].gameObject.SetActive(true);
                    var item = Resources.LoadAll<EquipmentItem>("Items");
                    var itemToPickUp = item[Random.Range(1, item.Length)];
                    Inventory.AddItem(itemToPickUp);
                    battleRewards[2].text += "An enemy droped  " + itemToPickU
p.itemName+"\n";
                }
            }
```

Osim iskustva i zlata postoji 10% šanse da igrač osvoji obični item i 5% šase da osvoji opremu. Osvojeno zlato i iskustvo se pridodaju igraču te ukoliko je igrač postigao nivo potreban za odabir klase se prikazuje ploča za izbor klase. Na toj ploči su prikazane klase te igrač bira jednu od njih. Osim klasa prikazan je i opis svake. Za kraj borbe se također učitava zvuk te se pokreće scena na kojoj se igrač nalazio prije borbe.

4.7 Victory

Ukoliko je igrač uspješno porazio šefa te ušao u „Shadow Tomb“ prikazuje se scena pobjede (Slika 24.). Na sceni pobjede se nalazi gumb koji omogućuje vračanje igrača na glavni izbornik i ploča koja prikazuje vrijeme koje je igraču bilo potrebno da pobijedi u igri, nivo koji je postigao, te broj protivnika koje je pobijedio.

Slika 24 Scena pobjede

Podaci o nivou igrača i broju protivnika koje je pobijedio se preuzimaju direktno iz skripte koja sadrži podatke o igraču. Za prikaz vremena je potrebno zbrojiti vrijeme spremljeno u skripti igrača, koje je postavljeno na 0 prilikom početka nove igre, sa vremenom koje je proteklo od početka trenutnog početka igranja. To vrijeme je u sekundama te ga je potrebno pretvoriti u sate i minute.

```
float timeToScale = PlayerUnit.time + Time.realtimeSinceStartup;
        int h = (int)(timeToScale / 3600);
        timeToScale -= h * 3600;
        int m = (int)(timeToScale  / 60);
        timeToScale -= m * 60;
        int s = (int)(timeToScale);
        time.text = h.ToString()+":" + m.ToString() +":"+ s.ToString();
```

5 Zaključak

Unity razvojno okruženje je pogodno za razvoj 2D i 3D igara. Najpopularniji jezik koji se koristi uz Unity je C# te je korišten za izradu ovog rada. Za razvoj igre u Unityju je potrebno naučiti rad sa objektima koje nudi i kako sa skriptama dodati funkcionalnost tim objektima. Također važna stavka razvoja igre je i organizacija datoteka kako bi se smanjilo potrebno vrijeme dohvaćanja istih. Uz to je bitna i organizacija objekata na sceni kako bi se brže i efikasnije moglo izvršiti obrada istih. Treba zapamtiti da je objektima moguće dodati kao komponentu jedino skriptu koja proizlazi iz klase „MonoBehaviour“. Ta klasa sadrži mnoštvo metoda koje olakšavaju izradu igre jer prate događaje na sceni te sukladno njima se poziva odgovarajuća metoda. Osim njih bitne su i skripte koje proizlaze iz klase „ScriptableObject“ jer na taj način se omogućuje stvaranje mnoštva objekata koji predstavljaju tip podataka koji sadrži određene vrijednosti koje korisnik želi.

Igra prikazana u radu sadrži mnoštvo prostora za proširenja. Osim dodavanja novih scena, mogu se dodati likovi s kojima igrač može upravljati, dodati vještine protivnicima ili čak proširiti sistem klasa tako da igrač može kombinirati klase.

6 Bibliografski navodi

[1] [Mrežno] https://en.wikipedia.org/wiki/History_of_games [Pristupano 15. rujan 2020.]

[2] [Mrežno] https://en.wikipedia.org/wiki/Pong [Pristupano 15. rujan 2020.]

[3] [Mrežno] https://equestris.hr/blog/ubrzani-razvoj-gaming-industrije/ [Pristupano 15. rujan 2020.]

[4] [Mrežno] https://en.wikipedia.org/wiki/Video_game_genre [Pristupano 15. rujan 2020.]

[5] [Mrežno] https://en.wikipedia.org/wiki/Role-playing_video_game#Popularity [Pristupano 15. rujan 2020.]

[6] [Mrežno] https://en.wikipedia.org/wiki/History_of_Eastern_role-playing_video_games#Golden_Age_(1990s%E2%80%93mid-2000s) [Pristupano 15. rujan 2020.]

[7] [Mrežno] https://en.wikipedia.org/wiki/Unity_(game_engine) [Pristupano 15. rujan 2020.]

[8] [Mrežno] https://docs.unity3d.com/Manual/Hierarchy.html[Pristupano 16. rujan 2020.]

[9] [Mrežno] https://docs.unity3d.com/Manual/UsingTheSceneView.html [Pristupano 16. rujan 2020.]

[10] [Mrežno] https://docs.unity3d.com/Manual/GameView.html [Pristupano 16. rujan 2020.]

[11] [Mrežno] https://docs.unity3d.com/Manual/AssetStore.html [Pristupano 16. rujan 2020.]

[12] [Mrežno] https://docs.unity3d.com/Manual/UsingTheInspector.html [Pristupano 16. rujan 2020.]

[13] [Mrežno] https://docs.unity3d.com/Manual/ProjectView.html [Pristupano 17. rujan 2020.]

[14] [Mrežno] https://docs.unity3d.com/Manual/Console.html [Pristupano 17. rujan 2020.]

[15] [Mrežno] https://docs.unity3d.com/Manual/class-MonoBehaviour.html [Pristupano 17. rujan 2020.]

[16] [Mrežno] https://docs.unity3d.com/2018.1/Documentation/ScriptReference/EventSystems.EventSystem.html [Pristupano 17. rujan 2020.]

7 Popis slika

8 Prilozi

Uz ovaj rad je priložena razvijena igra, te Unity projekt koji sadržava sve komponente korištene u razvoju iste.