Izrada 2D računalne igre u Unity Game Engineu

Ognjanović, Dario

Sveučilište u Rijeci – Odjel za informatiku

Preddiplomski jednopredmetni studij informatike

Dario Ognjanović

Izrada 2D računalne igre u Unity Game Engineu

Završni rad

Mentorica: doc. dr. sc. Marina Ivašić Kos

Rijeka, rujan 2018.

Zadatak za završni rad

Sažetak

U ovom radu prikazan je proces izrade jednostavne 2D TurnBased RPG igre u Unity alatu. Cilj igre je proći sve zone na mapi, naći sve škrinje s blagom te postati najjači borac. Igra se sastoji od 3 scene: Main Menu, Game i Battle. Iz svake scene izdvojene su najbitnije stvari i pojašnjene u detalje.

Ključne riječi

Unity, igra, player, kamera, skripta, C#, neprijatelji, mapa, animacije, animator, borba, playerparty, enemyparty, turn system.

Sadržaj

Uvod

Videoigre se igraju pomoću računala ili mobitela ili konzola. Igre su napravljena kao proizvod koji će biti zabavan za korištenje. U vrlo malom razdoblju igre su postale ključan dio života svakog djeteta te su postale jedan od najpopularnijih oblika zabava danas. Prva igra je napravljena 1958. godine s nazivom „Tenis za dvoje", te ubrzo nakon toga krenuo je drastičan porast u izradi.

RPG igre odnosno TurnBasedRPG [1] igre su specifičan žanr igre koji je postao najzastupljeniji i najpopularniji žanr. Ime RPG dolazi od načina igranja, igrač upravlja likom ili družinom(engl. Party), gdje svaki lik ima oružja, napade, život, skuplja iskustvo, bori se s neprijateljima. Općenito je vrlo teško odrediti sve stvari koje RPG čine onime što jest. Uz sve stvari koje su u RPG-u postoje i podvrste RPG-a. U ovom slučaju to je TurnBased RPG. To je prva vrsta RPG-a koja je napravljena. TurnBased bazira se na borbu između družine i neprijatelja potezima. Svaka strana ima određeni broj napada po potezu te kada završi čeka da druga strana napadne i obrnuto.

U ovom završnom radu prikazan je proces izrade jednostavne 2D *TurnBased RPG* [1] igre, te postupak korištenja alata Unity [2] u kojemu je izrađena igra. Igra je napravljena za jednog igrača s idejom da se igrač zabavi prilikom igranja te da istraži svijet u kojemu se nalazi.

Igrač istražuje mapu, otkriva razne predmete postavljene na mapi i bori se s neprijateljima koji su skriveni na različitim dijelovima mape. Cilj igre je pobijediti što više neprijatelja kako bi lik postao što jači te skupiti sve škrinje koje sadrže razne predmete koje igrač može koristiti te pri tome ostati živ.

Igra je napravljena za Windows [3], Linux [4] i Mac OS [5] operacijske sustave. Prilikom programiranja skripti korišten je C# [6] programski jezik. Skriptama se postižu mehanike kretanja, borbe, korištenja predmeta i slično. Grafika [7] korištena u svrhu izrade likova, mape te animacija, a preuzeta je s raznih izvora. Isto vrijedi i za zvuk [8].

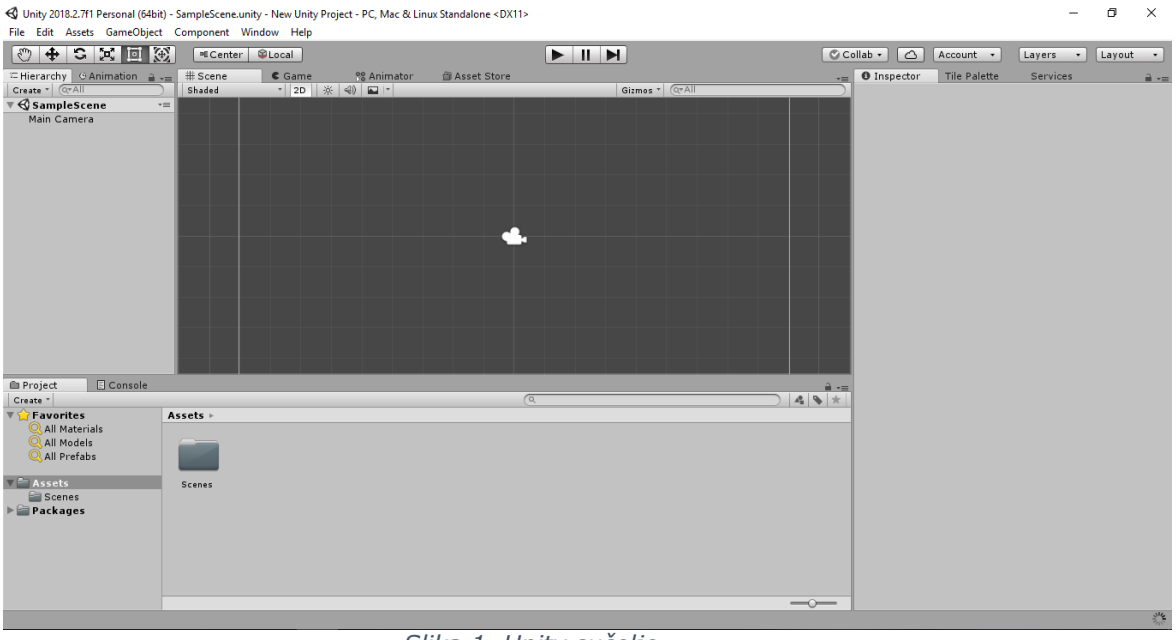
Motiv za izradu igre bio je i iskoristiti i proširiti znanje iz programiranja u radu s novim alatima i zadacima kako bi se stečena iskustva mogla primijeniti prilikom izrade sličnim projekata.

Unity

Unity game engine je program za izradu igara za razne platforme. Razvijen je od tvrtke Unity Technologies [9]. U početku je bio najavljen za platformu OS X, no ubrzo se proširio i na ostala računalna i mobilna tržišta. Razlog razvoja Unity enginea je omogućavanje široj populaciji korištenje dostupnijeg proizvoda za izradu igara. U vrlo kratkom periodu postao je najkorišteniji *engine* za izradu video igara. Razvojem iPhonea, Unity je postao jedini program koji je nudio sve mogućnosti operacijskog sustava, zbog toga je broj igara koje su se izrađivale u Unityu drastično porastao za čak 50% [5]. Unity podržava većinu programa za 3D modeliranje, video montažu, izradu slika, itd. Neki od programa koje Unity podržava su Adobe PS, Adobe Animator, 3D studio, Blender, Maya.

Unity sučelje

Unity sučelje, prikazano na Slici 1., sastoji se od prozora koji služe za izradu komponenata i objekata koje upotrebljavamo prilikom izrade igre. Najbitniji prozor je projekt (engl. *Project*) koji služi za navigaciju kroz sve elemente koje koristimo prilikom izrade igre (skripte, animacije, zvuk, *spriteove*).

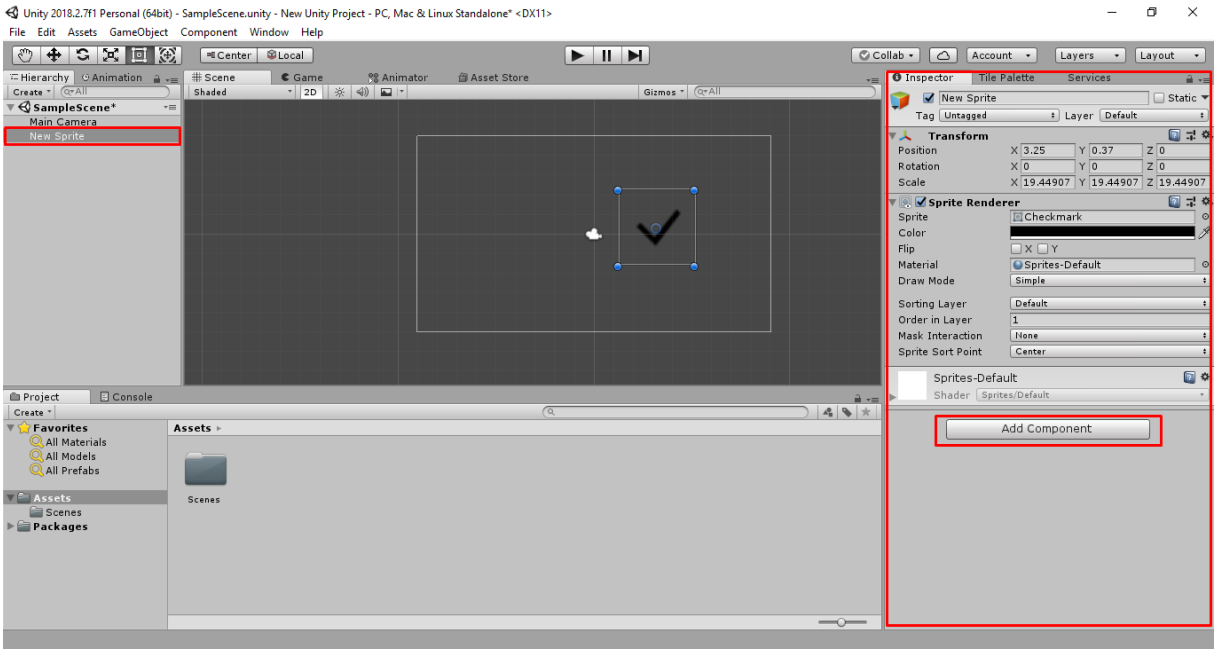


Slika 1. Unity sučelje

Na prozoru scena (engl. Scene) [10] stvaraju se i postavljaju objekti koji se koriste u igri. Svaka scena predstavlja jedan dio igre (glavni meni, igra, bitka) koje potom možemo sastaviti u jednu cjelinu. Objekti (engl GameObject) [11] čine elemente koje vidimo na sceni kao što su: Igrač, tlo,

kuće, neprijatelji itd., koje organiziramo u prozoru hijerarhija (eng. Hierarchy).

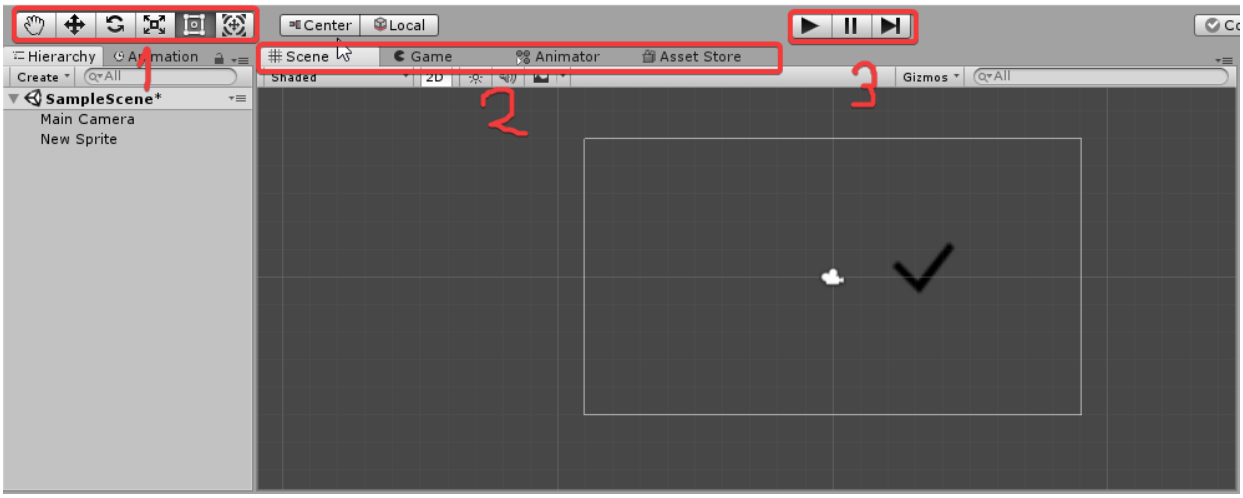
Pomoću inspektora (engl. Inspector) koji vidimo na Slici 2., možemo svaki objekt koji se nalazi na sceni ili unutar projekta izmjenjivati i dodavati mu komponente (eng. Add Component). U prozoru koji se otvori klikom na novu komponentu objekta postoje brojne mogućnosti koje možemo dodati na objekt.



Slika 2. Hijerarhija i objekti

Alatna traka (oznaka 1) označena na Slici 3, koja se nalazi iznad prozora za scenu omogućava prelazak na više bitnih prozora. *Game* [13] prozor prikazuje što će igrač vidjeti to jest što kamera koja se postavlja na igrača vidi. Kamere su ključan dio igre jer s njima manipuliramo izradu scena u igri.

Kako bismo pokrenuli igru koristimo gumbe koji se nalaze iznad alatne trake (oznaka 2). Osim toga za kretanje scenom i manipuliranjem objekata na sceni koristimo elemente alatne trake (oznaka 3).



Slika 3. Alatna traka

Pokraj prozora Game nalazi se još jedan bitni prozor kontrolor animacije (engl. Animator Controller) [14] koji omogućava animacije objekta ili lika. Kontrolor upravlja animacijama pomoću stanja i prijelaza među stanjima (prijelaz između animacije hoda i stajanja). Parametri animacije su vrijednosti to jest koordinate objekta kojima manipuliramo.

Kodovi kojima se služimo za manipulaciju objekata i mehaniku igre pišu se u skripte. Unity nudi 2 načina programiranja, JavaScript ili C#. Skripte se potom dodaju na objekte ili ih se poziva pomoću drugih skripti. U ovom radu korišten je C# programski jezik te alat Visual Studio [15].

Izrada igre

Ideja

Ideja završnog rada jest napraviti 2D TurnBased RPG računalnu igru. Kao pomoć pri izradi igre uzeo sam osnovne elemente koje bi takva igra sadržavala. Igra se sastoji od *playera* s kojim upravljamo i koji se bori s neprijateljima, skuplja iskustvo koje mu povećava štetu te život. *Player* također može pronaći škrinje s blagom u kojima se nalaze razne stvari koje mu mogu pomoći u borbi s neprijateljima. *Player* se po mapi kreće s tipkama W, A, S,D ili strelicama. U borbu s neprijateljima ulazi nasumično, jer su neprijatelji zapravo nevidljivi. Kada *player* naiđe na neprijatelja ulazi u novu scenu gdje se on nalazi na desnoj strani dok su neprijatelji na lijevoj. Potom odabire napad koji želi te ozljeđuje neprijatelja. Nakon što *player* završi potez, neprijatelj napada *playera*. Borba traje sve dok *player* ne umre ili dok ne porazi neprijatelje. Postoji više vrsta neprijatelja koji se mogu nasumično pojaviti na raznim dijelovima mape. Nakon što *player* porazi neprijatelja, povećava mu se iskustvo te nastavlja svoju avanturu. Cilj igre je skupiti sve škrinje s blagom te postati dovoljno jak da može poraziti sve neprijatelje u sve 4 zone mape.

Igra

Igra se sastoji od 3 bitne scene. Prva scena je glavni izbornik u kojemu su ponuđene 4 opcije. Ostale scene su *Game* u kojoj se odvija većina radnji te *Battle* scena gdje se *player* bori s neprijateljima. Sve opcije (*play*, *options*, *about*, *exit*) koje možemo izabrati su zapravo dugmadi. Ovdje ćemo opisati opciju *play*, jer je najvažnija.

Klikom na opciju *play* poziva se skripta *MainMenuControl* koja upravlja prijelazom iz jedne scene u drugu. Prilikom pritiska na dugme *play*, poziva se funkcija *NameEntry*, gdje igrač (osoba koja igra igru) upisuje svoje ime ili odabire opciju *load*. Pritiskom na *load* poziva se funkcija *LoadGame* koja učitava datoteku *SaveFiles* koja sadrži podatke o *playeru*, kao što su: pozicija, iskustvo, život. Datoteku potom pomoću funkcije *StartCoroutine* [16] čitamo te postavljamo podatke na vrijednosti koje su bile prije nego je igrač izašao iz igre.

```
public void LoadGame()
    {
        FileBrowser.SetFilters(true,new FileBrowser.Filter("Text Files", ".txt"));
        FileBrowser.SetDefaultFilter(".txt");
        FileBrowser.SetExcludedExtensions("");
        FileBrowser.AddQuickLink("SaveFiles", "Assets/SaveFiles/", null);
        StartCoroutine(ShowLoadDialogCoroutine());
    }
    IEnumerator ShowLoadDialogCoroutine()
```

```
    {
        yield return FileBrowser.WaitForLoadDialog(false, null, "Load File", "Load");
        try
        {
            string line;
            StreamReader theReader = new StreamReader(FileBrowser.Result, Encoding.Default);
            using (theReader)
            {
                do
                {
                    line = theReader.ReadLine();
                    file.Add(line);
                    if (line != null)
                    {

                        Initiate.Fade("Loading", Color.black, 3.0f);
                        file_loaded = true;
                    }
                }while (line != null);

                theReader.Close();
            }
        }
        catch { file_loaded = false; }
    }
```

Ako igrač prvi put igra igru ili želi krenuti ispočetka, upisuje svoje ime te pritiskom na *start* poziva se metoda *StartGame* koja pokreće scenu *Loading* te nakon 3 sekunde scena *Loading* pokreće scenu *Game*. Istovremeno se u datoteku *SaveFiles* upisuju osnovni statusi o *playeru*.

```
public void StartGame()
    {
        PlayerName = Name.text;
        Initiate.Fade("Loading", Color.black, 3.0f);
    }
```

Player

Ulaskom u scenu *Game* prvo što vidimo je *player* i njegovo sučelje. *Player* se pomiče pomoću skripte *PlayerMovement*, gdje se u funkciji *Update* [17] koja se pokreće određeni broj puta u svakom *frameu* [18] provjerava je li se promijenila vrijednost x ili y osi u bilo kojem smjeru. Kada se *player* pomakne, skripta detektira male promijene vrijednosti koordinata te se player pomiče zadanom brzinom.

```
void Update()
    {
        velocity.x = Input.GetAxisRaw("Horizontal");
        velocity.y = Input.GetAxisRaw("Vertical");
        body.MovePosition(body.position + velocity * speed * Time.deltaTime);

        Anim.SetFloat("MoveX", velocity.x);
        Anim.SetFloat("MoveY", velocity.y);
    }
```

Player ima još jednu skriptu *playerControl* koja ima više funkcija. Prva je bilježenje igračevih statusa kao što su: *health*, *level*, *armor*, *damage*. Ti

podaci se bilježe na način da se prvo provjerava je li odabrana opcija *Load*, ako je tada se podaci učitavaju iz datoteke. Kako bi smo izbjegli korištenje više datoteka za spremanje podataka o *playeru*, početni podaci o *playeru* definiraju se u glavnom izborniku u skripti *MainMenuControl*. Priloženi kod funkcije *Start* [19] nije potpun, ali potrebno je samo dio koda objasniti jer ostatak koda je vrlo sličan. Ovdje se koristimo funkcijama za parsiranje datoteka [20], to jest postavljamo vrijednosti za *playera* na vrijednosti koje se nalaze u *SaveFiles* datoteci.

```
void Start () {

if (MainMenuControl.file_loaded == true)

        {
            MainMenuControl.weapons.Clear();
            PlayerPosition.x = float.Parse(MainMenuControl.file[0]);
            PlayerPosition.y = float.Parse(MainMenuControl.file[1]);
            Player.transform.position = PlayerPosition;
            MainMenuControl.PlayerName = MainMenuControl.file[2];
            MainMenuControl.PlayerHealth = int.Parse(MainMenuControl.file[3]);
            MainMenuControl.PlayerMaxHealth = int.Parse(MainMenuControl.file[4]);
            MainMenuControl.PlayerExperience = int.Parse(MainMenuControl.file[5]);
            MainMenuControl.PlayerLevel = int.Parse(MainMenuControl.file[6]);
            MainMenuControl.potionCounter = int.Parse(MainMenuControl.file[7]);
            MainMenuControl.PlayerArmor = int.Parse(MainMenuControl.file[8]);
            MainMenuControl.PlayerAttack = int.Parse(MainMenuControl.file[9]);
            MainMenuControl.weaponmelee = int.Parse(MainMenuControl.file[10]);
            MainMenuControl.weaponbow = int.Parse(MainMenuControl.file[11]);
```

Player može otvarati škrinje pritiskom tipke e, ako se nalazi u blizini škrinje. Udaljenost igrača od škrinje provjeravamo uvjetom ako je udaljenost između *playera* i određene škrinje manja od 1.3 tada *player* otvara škrinju koja mu daje određenu nagradu. Ovih 8 *chest* pozicija služe kako bi smo saznali gdje se na mapi točno nalazi svaka pojedinačna škrinja, te za svaku škrinju moramo provjeravati udaljenosti između nje i igrača.

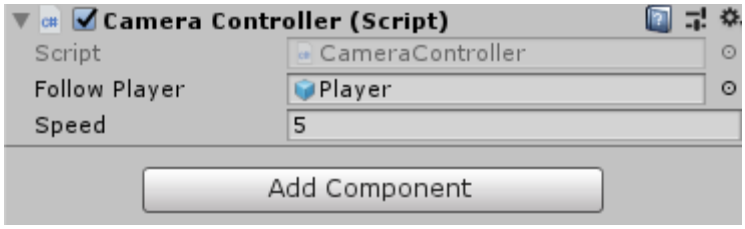
```
if (Input.GetKeyDown("e") && !MessageBox.activeSelf)
        {
            Vector2 Chest1Position = Chest1.transform.position;
            Vector2 Chest2Position = Chest2.transform.position;
            Vector2 Chest3Position = Chest3.transform.position;
            Vector2 Chest4Position = Chest4.transform.position;
            Vector2 Chest5Position = Chest5.transform.position;
            Vector2 Chest6Position = Chest6.transform.position;
            Vector2 Chest7Position = Chest7.transform.position;
            Vector2 Chest8Position = Chest8.transform.position;

            if ((Vector2.Distance(Chest1Position, PlayerPosition) <= 1.3) && chest1opened
== false)
            {
                MainMenuControl.potionCounter = MainMenuControl.potionCounter + 5;
```

```
                MessageBoxControl.messagetext = MainMenuControl.PlayerName + ", you found
5 health potions! You can use it in battle and your health will be refulled! But be carefull,
when you drink it you'll skip that turn.";
                MessageBox.SetActive(true);
                chest1opened = true;
            }
```

Kamera

Kamera (engl. Camera) [21] je objekt koji se automatski stvara na svakoj sceni. Kamera se koristi za prikaz svijeta te okoline koju *player* vidi. Kameru kao i svaki drugi objekt možemo izmjenjivati u inspektoru, možemo joj mijenjati poziciju te vertikalnost kako bi postigli drugačije perspektive *playera*. U sceni *Game* kameru koristimo na *playeru*, to jest kamera je igračevo vidno polje. To se postiže sa skriptom *CameraController* prikazano na Slici 4., kojoj pridružimo prefab [22] (objekt koji koristimo na više scena) *playera*. Kako bi kamera pratila *playera* u funkciji *Update* služimo se opet pozicijama *playera* te pozicijom kamere. Kada se *player* pomakne kamera prepozna da se pomaknuo te nam se čini da kamera „slijedi" *playera*.

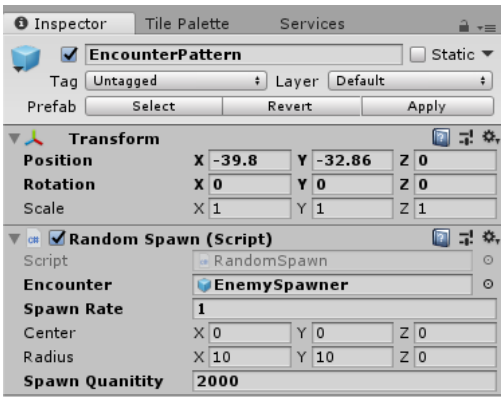
```
public GameObject followPlayer;
    private Vector3 PlayerPos;
    public float speed;

       void Update () {
        PlayerPos        =        new        Vector3(followPlayer.transform.position.x,
followPlayer.transform.position.y, transform.position.z);
        transform.position   =   Vector3.Lerp(transform.position,   PlayerPos,   speed   *
Time.deltaTime);
       }
```



Slika 4. CameraController skripta

Neprijatelji

Neprijatelji su napravljeni na način da se u *Game* sceni pojavljuju nasumično njihovi generatori odnosno *EncounterPattern prefabov koje vidimo na Slici 5*. *EncounterPattern* je objekt koji sadrži skriptu koja omogućava nasumično generiranje neprijatelja po mapi koji su sadržani u *EnemySpawner prefabu*.



Slika 5. EncounterPattern detalji skripte

Prvo ćemo pojasniti skriptu. U *Start* funkciji generiramo dva polja, jedno *GameObject* koje će sadžavati *EnemySpawner* klonove (engl. *Clone*) [23], te *Vector3* polje koje će pamtiti sve pozicije klonova. Potom u *Update* funkciji koja se poziva određeni broj puta po *frameu*, imamo dvije provjere. Prva provjera fokusira se na to je li broj *EnemySpawnera* veći ili jednak onome što smo odredili u polju *SpawnQuanitity*. Ako je uvjet petlje istinit tada se brišu svi postojeći *EnemySpawneri* te se stvaraju novi.

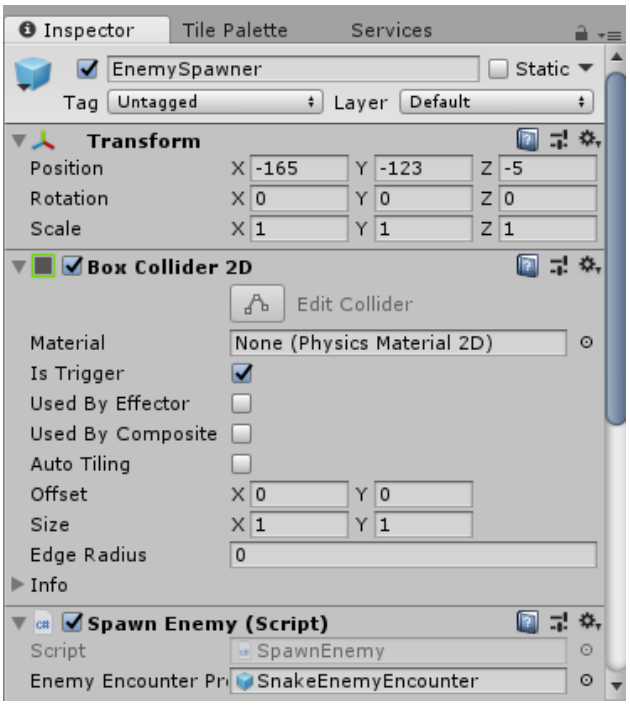
```
void Start()
    {
        EncounterArr = new GameObject[SpawnQuanitity];
        spacingArr = new Vector3[SpawnQuanitity];
    }
    void FixedUpdate()
    {
        if (Spawn >= SpawnQuanitity)
        {
            DeleteAllEncounters();
        }
```

Druga provjera fokusira se na to da je vrijeme između svakog novog *EnemySpawnera* koji stvorimo veće od vremena koje je zadano varijablom *SpawnRate*. Ako je uvjet istinit *nextSpawn* se određuje za sljedeći *EnemyEncounter*. Ulazimo u do while petlju koja za svaku novu *spawn* poziciju provjerava sa funkcijom *ComparePosition* [24] je li udaljenost nove pozicije manja od 2. To radimo da bi izbjegli preklapanje objekata na mapi te da bi postigli veću rasprostranjenost objekata. Petlja se izvršava sve dok se u varijablu *diff* ne vrati vrijednost *false* to jest da je udaljenost veća od 2. Tada dodajemo novu poziciju u *Vector3* polje za pozicije, te generiramo objekt. Skripta se izvršava sve dok ne izađemo iz igre.

```
        else if (Time.time > nextSpawn)
        {
            nextSpawn = Time.time + spawnRate;
            diff = true;
```

```
            do
            {
              SpawnPosition = center + new Vector3(Random.Range(-radius.x / 2, radius.x /
              2), Random.Range(-radius.x / 2, radius.x / 2), 0);
                diff = ComparePosition(spacingArr, Spawn, SpawnPosition);

            } while (diff != false);

            spacingArr[Spawn] = SpawnPosition;

            EncounterArr[Spawn]       =       Instantiate(Encounter,       SpawnPosition,
            Quaternion.identity) as GameObject;
            Spawn++;
        }
    }


private void DeleteAllEncounters()
    {
        foreach (GameObject go in EncounterArr)
        {
            Destroy(go);
        }

        spacingArr = new Vector3[SpawnQuanitity];
        Spawn = 0;
    }

    private bool ComparePosition(Vector3[] spacingArr, int Spawn, Vector3 Spawnpos)
    {
        for (int i = 0; i <= Spawn; i++)
        {
            if (Vector3.Distance(Spawnpos, spacingArr[i]) > 2)
            {
                return false;
            }
        }
        return true;
    }
```

Nakon što smo prošli kroz skriptu *RandomSpawn* te odredili poziciju sljedećeg *EnemySpawnera* i generirali novi *EnemySpawner* na mapi, trebamo vidjeti što taj *EnemySpawner* sadrži. Prva bitna stvar je *BoxCollider2D* [25], prikazano na Slici 6., koji nam omogućava prepoznavanje *playerovog Collidera* te pomoću njih možemo napraviti da kada *player* dodirne *EnemySpawner Collider* , pozove se *SpawnEnemy* skripta te igra prelazi u *Battle* scenu. *Battle* scenu ćemo objasniti kasnije, prvo pogledajmo *SpawnEnemy* skriptu.



Slika 6. EnemySpawner Collider i skripta

Na početku definiramo koju vrstu *EnemyEncountera* ćemo koristit tako što povučemo i stavimo *prefab* na poziciju *EnemyEncounterPrefab*, potom postavimo *spawning* na *false*, jer ne želimo da se neprijatelji stvaraju u bilo kojoj sceni nego samo u *Battle* sceni. Potom u *Start* funkciji postavljamo da se objekt ne može uništiti prilikom učitavanja *Battle* scene te učitamo *Battle* scenu. Potom u funkciji *OnSceneLoad* provjeravamo je li nova scena *Battle*, ako je stvaramo *enemyEncounterPrefab* te završavamo skriptu, inače ako nije uništavamo *enemyEncounterPrefab* te postavljamo scenu na game.

```
[SerializeField]
private GameObject enemyEncounterPrefab;

private bool spawning = false;

void Start() {
	DontDestroyOnLoad (this.gameObject);

	SceneManager.sceneLoaded += OnSceneLoaded;
}

private void OnSceneLoaded(Scene scene, LoadSceneMode mode) {
	if (scene.name == "Battle") {
		if (this.spawning) {
			Instantiate (enemyEncounterPrefab);
		}
		SceneManager.sceneLoaded -= OnSceneLoaded;
		Destroy (this.gameObject);
	}
}
```

```
void OnTriggerEnter2D(Collider2D other) {
	if (other.gameObject.tag == "Player") {
		this.spawning = true;
		SceneManager.LoadScene ("Battle");
	}
```

Mapa

Mapa je kreirana korištenjem *BrowserQuest*[1] *spritetova* [26] u Unity alatu za bojanje po sceni. Mapa je napravljena tako što se prvo odabere *Tileset* [27] koji će se koristiti, potom se od njega napravi paleta, prikazana na Slici 7. Potom u hijerarhiji odaberemo novi 2D objekt tipa *grid* [28], koji nam omogućava da svaka ikonica u paleti predstavlja jedan kvadratić na sceni.

Slika 7. Paleta za tilemap

Mapu koju vidimo na Slici 8., kreiramo u slojevima kako bi postigli dubinu te kasnije omogućili *playeru* interakciju s objektima na određenim slojevima. Slojeve generiramo tako što odaberemo novi 2D objekt tipa

Tilemap koji treba biti dijete od *grida*. Potom u paleti pomoću opcije active theme odaberemo sloj.

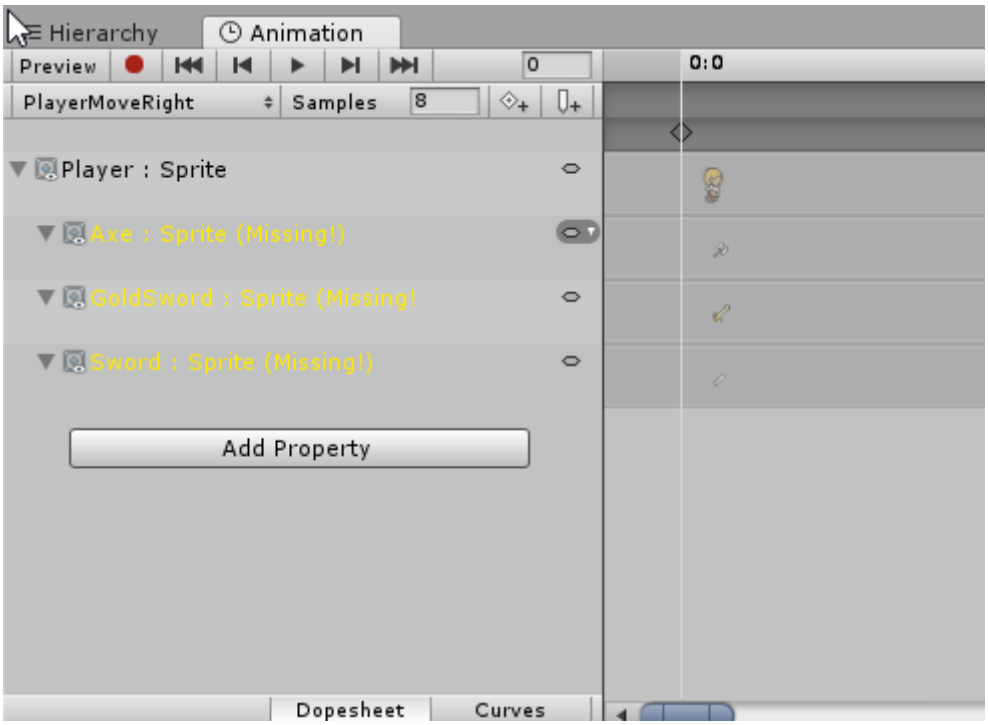
Slika 8. Mapa igre

Za mapu (Slika 8.) koristio sam više slojeva kako bi postigao željeni rezultat. Prvi sloj je zemlja na koji predstavlja podlogu na kojoj se nalaze svi ostali slojevi. Drugi važni sloj je put koji daju osjećaj raznolikosti te nasumičnosti mape. Od ostalih slojeva bitno je spomenuti sloj objekti te drveće, koji zapravo predstavljaju sve objekte s kojima *player* može ući u kontakt. Na tim slojevima postavljeni su *2DBoxCollideri*. Da bi se postigao rezultat kao što vidimo na Slici 9., potrebno je ta dva sloja razdvojiti u 4 sloja odnosno prepoloviti svaki objekt u dva djela, gornji i doljni. Prva dva sloja su postavljeni u istom sloju (ovdje pričamo o slojevima koji su pridruženi svakom objektu, ne o slojevima za palete) [29] kao što je i *player*, dok su druga dva (ObjektiIza, DrvećeIza) postavljena za jedan sloj više od *playera*. Rezultat toga je kada *player* prolazi kroz gornji dio objekta čini se kao da nestaje iza objekta.

Slika 9. Prikaz prolaska igrača iza objekta

Animacije

Animacije (engl. Animation) [30] u programu Unity moguće je raditi na više načina: stvaranje animacija u programu, uvozom gotovih animacija iz drugih programa te animacije pomoću skripti. Animacije za ovu igru napravljene su ručno. Dodavanjem *spriteova frame* po *frame*. Na slici Slici 10. prikazan je postupak dodavanja animacija za *playera*. Prvo odaberemo naziv animacije, potom dodajemo *clipove* [31], koji nam služe kao bismo jednu animaciju *playerovog* kretanja odvojili na više manjih dijelova. Slika 11. prikazuje nam podjelu *Tilesheeta*(slike koja se sastoji od više *spriteova*) na više manjih *spriteova* koje potom povlačimo na određene *frameove* kako bismo stvorili animaciju. Kada smo zadovoljni sa animacijom odaberemo broj *frameova* (*Samples na slici*) te odlazimo na sljedeći *clip*. Postupak se ponavlja sve dok ne animiramo sve pokrete koje smo predvidjeli.

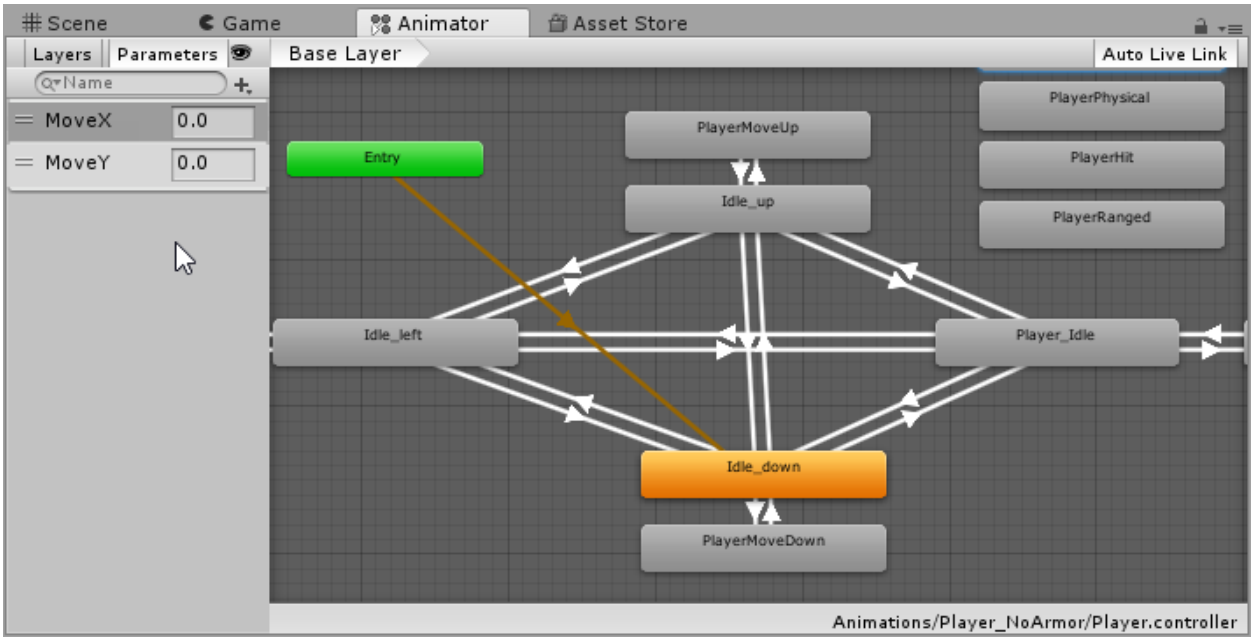


Slika 10. Animacija playera

Slika 11. Podjela Tilesheeta na spriteove

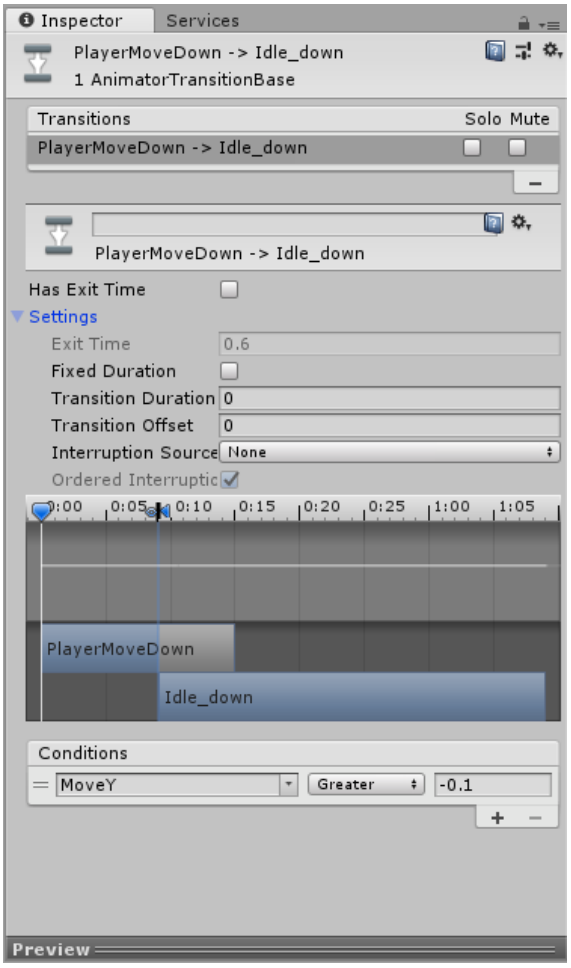
Animator

Animator ili *Animator Controller* [32] omogućuje organiziranje skupa animacija za objekt. Kontrolor upravlja stanjima animacija i prijelazima među njima pomoću stanja stroja (engl. *State Machine*) [33]. Kontrolor je koristan kod prebacivanja iz različitih animacija. Na primjer, želimo prebaciti animaciju igrača iz stanja mirovanja u stanje kretanja. Slika 12. prikazuje Kontrolora koji sam koristio za animaciju kretanja *playera*. U glavnom dijelu nalazi se stablo koje se sastoji od različitih klipova animacija za objekt *Player*. *Entry* stanje predstavlja stanje ulaza, to jest stanje iz kojeg krećemo. Potom povezujemo to stanje s stanjem iz kojeg želimo kretati, u ovom slučaju to je *Idle_down*. Strijelce predstavljaju koje sve animacije se mogu pokrenuti odnosno u koja sva stanja se može prijeći iz početnog stanja. Ovdje su sva stanja odnosno animacije za hod spojene međusobno tako da *player* u bilo kojem trenutku može ući u bilo koje stanje. Kako bi se aktivirala strijelaca između stanja potrebno je zadovoljiti određene kriterije.



Slika 12. Prikaz kontrolora za playera

S desne strane slike nalaze se parametri (eng. *Parameters*) koje koristimo u skripti. U ovom slučaju korištene su koordinate objekta za manipulaciju. U slici 13. vidimo razne parametre koje dodajemo na prijelaze, no najbitniji su uvijete (eng. *Conditions*).



Slika 13. Uvjeti za prijelaz između stanja

U ovom primjeru *player* će se pomaknuti iz stanja *PlayerMoveDown* u stanje *Idle_down* ako se koordinata y promijeni za 0.1. Stanje će se promijeniti zbog skripte *PlayerMovement* koju smo pojasnili prije. U funkciji *Update* dohvatimo promijenjene x i y koordinate, pomaknemo objekt te provjerimo uvjet za prijelaz među stanjima, ako je uvjet ispunjen promijeni se stanje.

```
void Start()
    {
        Anim = GetComponent<Animator>();
    }
    void Update()
    {
        velocity.x = Input.GetAxisRaw("Horizontal");
        velocity.y = Input.GetAxisRaw("Vertical");
        body.MovePosition(body.position + velocity * speed * Time.deltaTime);

        Anim.SetFloat("MoveX", velocity.x);
        Anim.SetFloat("MoveY", velocity.y);
    }
```

Borba

Borba između *playera* i neprijatelja odvija se u *Battle* sceni prikazanoj na Slici 14. Bitka se odvija tako da prvo *player* napada neprijatelje odabirom miša, potom neprijatelji uzvraćaju udarac. Borba se odvija sve dok jedna strana ne bude poražena. Kada igrač ubije neprijatelje završava *Battle* scena te *player* skuplja iskustvo i vraća se u *Game* scenu.

Slika 14. Prikaz Battle scene

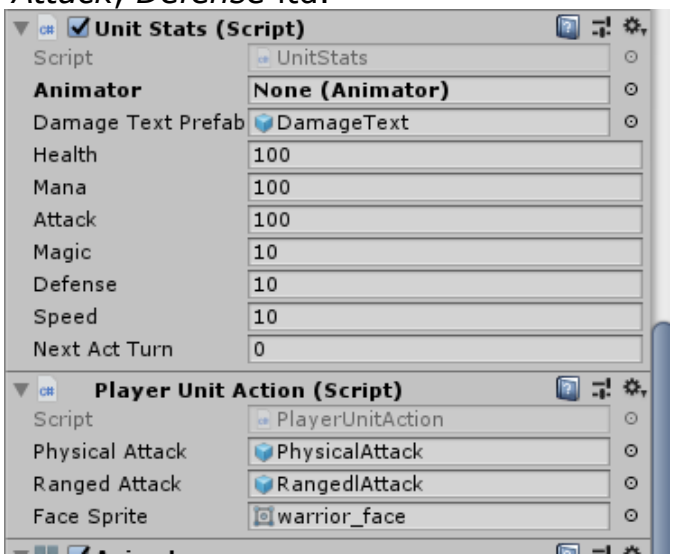
PlayerParty

Battle scena je vrlo složena stoga ću objasniti samo osnovne dijelove. Prvo se fokusirajmo na *playera*. *Playera* smo smjestili kao dijete od *PlayerParty*, jer u budućnosti će imati više članova družine. *PlayerParty* upravlja *playerom* i napadima *playera* putem skripte *SelectUnit*. Funkcija *selectCurrentUnit* odabire kojeg će od neprijatelja *player* napasti. Odabir se vrši tako što pomoću strelice miša prelazimo preko neprijatelja i kliknemo desnim klikom. Potom odabire napad pomoću funkcije *selectAttack*, gdje je *physical* zadana vrijednost, te napada neprijatelja funkcijom *attackEnemyTarget*.

```
public void selectCurrentUnit(GameObject unit)
    {
        this.currentUnit = unit;
```

```
        this.actionsMenu.SetActive(true);

        this.currentUnit.GetComponent<PlayerUnitAction>().updateHUD();
    }

    public void selectAttack(bool physical)
    {
        this.currentUnit.GetComponent<PlayerUnitAction>().selectAttack(physical);

        this.actionsMenu.SetActive(false);
        this.enemyUnitsMenu.SetActive(true);
    }

    public void attackEnemyTarget(GameObject target)
    {
        this.actionsMenu.SetActive(false);
        this.enemyUnitsMenu.SetActive(false);

        this.currentUnit.GetComponent<PlayerUnitAction>().act(target);
    }
}
```

Potom treba objasniti skripte *playera prikazane na Slici 15*. Prva skripta koja je bitna je *UnitStats*. Ova skripta sadržava sve podatke o *playeru* kao što su: *Health*, *Mana*, *Attack*, *Defense* itd.



Slika 15. Skripte za playera

U ovoj skripti najvažnija funkcija je receiveDamage. Ova funkcija izvršava više stvari. Prvo smanjuje život playera za vrijednost štete koju je napravio neprijatelj. Potom taj život prikazuje na platnu kako se smanjuje. Prvi if uvjet provjera ako player nije primio štetu, tada se zadržava početna vrijednost života, inače se život smanjuje za određenu vrijednost na platnu.

```
public void receiveDamage(float damage)
    {
        this.health -= damage;
```

```
        animator.Play("Hit");

        GameObject HUDCanvas = GameObject.Find("HUDCanvas");
        GameObject damageText = Instantiate(this.damageTextPrefab) as GameObject;
        Vector2 screenPos = Camera.main.WorldToScreenPoint(gameObject.transform.position +
new Vector3(0,1,0));
        damageText.transform.SetParent(HUDCanvas.transform, false);
        var textObj = damageText.GetComponent<TextMeshProUGUI>();
        if(damage==0)
            textObj.text = "0";
        else
         textObj.text= damage.ToString("F2");
        damageText.transform.position = screenPos;
        damageText.transform.localScale = new Vector3(1f, 1f);
```

Ako je život playera manji ili jednak 0 nakon napada, tada se pokreće funkcija isDead, te se objekt uništava pozivom Destroy(this.gameObject).

```
        if (this.health <= 0)
        {
            this.dead = true;
            this.gameObject.tag = "DeadUnit";
            Destroy(this.gameObject);
        }
    }

public void calculateNextActTurn(int currentTurn)
    {
        this.nextActTurn = currentTurn + (int)Math.Ceiling(100.0f / this.speed);
    }

    public int CompareTo(object otherStats)
    {
        return nextActTurn.CompareTo(((UnitStats)otherStats).nextActTurn);
    }

    public bool isDead()
    {
        return this.dead;
    }

    public void receiveExperience(float experience)
    {
        this.currentExperience += experience;
    }
```

Druga skripta je *PlayerUnitAction* u kojoj *player* odabire napad koji će koristiti na neprijatelju. Ima dvije mogućnosti. Prva je *physicalAttack* a druga *rangedAttack*. U funkciji Awake [34] *„instanciraju"* se objekti za ta dva napada. Potom u funkciji *selectAttack* se provjerava koji napad je izabran. Odabir neprijatelja se vrši kreiranjem liste u koju se spremaju svi neprijatelji s opcijom *clickable*, čiju skriptu ćemo objasniti kasnije. Potom se svi neprijatelji s opcijom *clickable* postavljaju na *true* vrijednosti *foreach* petljom. To koristimo jer želimo da svi objekti na koje vršimo napade budu *clickable*. Nakon toga vrši se udarac na željenog neprijatelja.

```
void Awake()
    {
        this.physicalAttack = Instantiate(this.physicalAttack, this.transform) as
GameObject;
        this.rangedAttack = Instantiate(this.rangedAttack, this.transform) as GameObject;

        this.physicalAttack.GetComponent<AttackTarget>().owner = this.gameObject;
        this.rangedAttack.GetComponent<AttackTarget>().owner = this.gameObject;

        this.currentAttack = this.physicalAttack;
    }

    public void selectAttack(bool physical)
    {
        this.currentAttack = (physical) ? this.physicalAttack : this.rangedAttack;
        List<ClickableObject> clickableEnemies = new List<ClickableObject>();
        var objs = GameObject.FindGameObjectsWithTag("EnemyUnit");
        foreach(var obj in objs)
        {
            clickableEnemies.Add(obj.GetComponent<ClickableObject>());
        }
        foreach(var ce in clickableEnemies)
        {
            ce.Enabled = true;
        }
    }

    public void act(GameObject target)
    {
        this.currentAttack.GetComponent<AttackTarget>().hit(target);
    }

    public void updateHUD()
    {
        MainMenuControl.PlayerHealth = (int)gameObject.GetComponent<UnitStats>().health;
    }
}
```

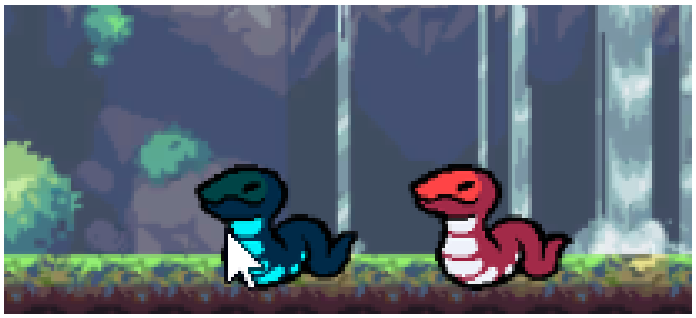
EnemyParty

EnemyParty se sastoji od neprijatelja na koje smo naišli pomoću *EnemyEncounter* skripte. Sada se neprijatelji sadržani u *EnemyEncounter prefabu* postavljaju na scenu. Svaki od neprijatelja sadrži skripte *UnitStats*, *EnemyUnitActions* i *ClickableObjects*. *UnitStats* skripta je ista kao i za *playera* pa ćemo ju preskočiti. *EnemyUnitActions* skripta slična je *PlayerUnitActions* skripti, no neprijatelji imaju samo jedan napad. Ovdje je samo važno objasniti u funkciji *Awake if* uvjet. Ovdje se provjerava je li postoji više *playera* koje treba napasti, ako postoje napad se „*randomizira*" između meta.

```
void Awake () {
            this.attack = Instantiate (this.attack);

            this.attack.GetComponent<AttackTarget> ().owner = this.gameObject;
      }
```

```
	GameObject findRandomTarget() {
		GameObject[] possibleTargets = GameObject.FindGameObjectsWithTag
(targetsTag);

		if (possibleTargets.Length > 0) {
			int targetIndex = Random.Range (0, possibleTargets.Length);
			GameObject target = possibleTargets [targetIndex];

			return target;
		}

		return null;
	}

	public void act() {
		GameObject target = findRandomTarget ();
		this.attack.GetComponent<AttackTarget> ().hit (target);
	}
}
```

Druga skripta *ClickableObject* bavi se davanjem boje neprijateljima, to jest vizualni prikaz kada *player* odluči kliknuti na neprijatelja da bi ga napao. Ovdje je bitno napomenuti da ova skripta radi uz pomoć još jedne skripte koja je sadržana u *ClickManager* objektu na *Battle* sceni. Ukratko, *ClickManager* pomoću kamere i pozicije miša određuje koji objekti će postati *clickable*. Potom *ClickableObject* skripta koristi te objekte u listi *ClickManager* te im mijenja boju kada pređemo mišem preko njih. Slika 16. prikazuje promjenu boje prelaskom miša.



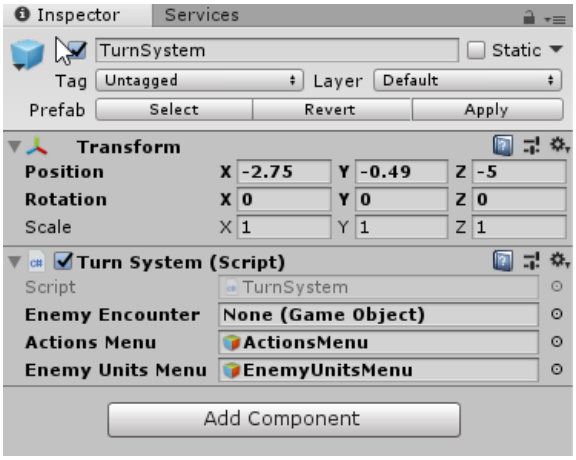
Slika 16. Clickable objekt

```
public Color HighlightColor = Color.cyan;
    private Color OriginalColor;
    private SpriteRenderer spriteRenderer;
    private ClickManager cm;
    public bool Enabled = false;

    public UnityEvent OnClick = new UnityEvent();

    void Start()
    {
        spriteRenderer = GetComponent<SpriteRenderer>();
        OriginalColor = spriteRenderer.color;
        if (!GetComponent<BoxCollider2D>())
```

```
            gameObject.AddComponent<BoxCollider2D>();
        cm = FindObjectOfType<ClickManager>();
    }
    void FixedUpdate()
    {
        if    (cm.hoverObject    !=    null    &&    cm.hoverObject.GetInstanceID()    ==
gameObject.GetInstanceID())
        {
            Highlight();
        }
        else
        {
            DeHighlight();
        }

    }

    public void Click()
    {
            OnClick.Invoke();
           Debug.Log("Nemoze xD");
    }

    void Highlight()
    {
        spriteRenderer.color *= HighlightColor;
    }

    void DeHighlight()
    {
        spriteRenderer.color = OriginalColor;
    }
}
```

Turn System

Sada kada smo bolje pojasnili *PlayerParty* i *EnemyEncounter*, možemo prijeći na *TurnSystem prikazan na Slici 17*. *TurnSystem* je onaj element koji spaja sve dosadašnje skripte i objekte u jedan element. To je naravno, borba.



Slika 17. Turn System

Prvo, u *Start* funkciji definiraju se *PlayerParty* i *EnemyUnit*, te se dodaju statusi kao što su *health*, *damage*, *defense*. To se postiže korištenjem foreach petlje koja svakom *unitu* pridružuje statuse. Potom se na kraju Start funkcije poziva *this.nextTurn(),* što poziva funkciju *nextTurn*. Funkcija *nextTurn* provjerava if uvjetima jesu li svi neprijatelji poraženi, ako jesu *player* uzima nagradu te nastavlja avanturu u *Game* sceni. Ako je *player* poražen, završava igra te se pojavljuje *Game Over* screen. Inače se nastavlja borba tako što prvo *player* napada te nakon njega neprijatelji.

```
void Start() {
		this.playerParty = GameObject.Find ("PlayerParty");

		unitsStats = new List<UnitStats> ();
		GameObject[] playerUnits = GameObject.FindGameObjectsWithTag("PlayerUnit");
		foreach (GameObject playerUnit in playerUnits) {
			UnitStats currentUnitStats = playerUnit.GetComponent<UnitStats> ();
			currentUnitStats.calculateNextActTurn (0);
			unitsStats.Add (currentUnitStats);
		}
		GameObject[] enemyUnits = GameObject.FindGameObjectsWithTag("EnemyUnit");
		foreach (GameObject enemyUnit in enemyUnits) {
			UnitStats currentUnitStats = enemyUnit.GetComponent<UnitStats> ();
			currentUnitStats.calculateNextActTurn (0);
			unitsStats.Add (currentUnitStats);
		}
		unitsStats.Sort ();

		this.actionsMenu.SetActive (false);
		this.enemyUnitsMenu.SetActive (false);

		this.nextTurn ();
	}

	public void nextTurn() {
		GameObject[] remainingEnemyUnits = GameObject.FindGameObjectsWithTag
("EnemyUnit");
		if (remainingEnemyUnits.Length == 0) {
```

```
                this.enemyEncounter.GetComponent<CollectReward> ().collectReward ();
            MainMenuControl.PlayerExperience += 50;
                Initiate.Fade("Game",Color.black, 2.0f);
              }

              GameObject[]   remainingPlayerUnits   =   GameObject.FindGameObjectsWithTag
("PlayerUnit");
              if (remainingPlayerUnits.Length == 0) {
            Initiate.Fade("Main Menu", Color.black, 2.0f);
        }

              UnitStats currentUnitStats = unitsStats [0];
              unitsStats.Remove (currentUnitStats);

              if (!currentUnitStats.isDead ()) {
                GameObject currentUnit = currentUnitStats.gameObject;

                currentUnitStats.calculateNextActTurn (currentUnitStats.nextActTurn);
                unitsStats.Add (currentUnitStats);
                unitsStats.Sort ();

                if (currentUnit.tag == "PlayerUnit") {
                  this.playerParty.GetComponent<SelectUnit> ().selectCurrentUnit
(currentUnit.gameObject);
                } else {
                  currentUnit.GetComponent<EnemyUnitAction> ().act ();
                }
              } else {
                this.nextTurn ();
              }
      }
}
```

Zaključak

U ovom radu predstavljen je razvoj 2D TurnBased RPG igre u Unity Engineu uz pomoć C# programskog jezika za izradu skripti. *Spriteovi* su preuzeti s interneta, te su potom animirani ili nacrtani na sceni pomoću Unitya. Zvuk je također preuzet s interneta.

Cilj izrade igre je bio naučiti proces izrade jednostavnih 2D igara korištenjem Unity enginea. Ovaj rad opisuje postupak izrade jednostavne igre i sve bitne elemente koji igru čine TurnBased RPGom .

Prvo je opisano Unity sučelje kako bi se alat mogao brže i lakše koristiti. Potom su objašnjeni svi elementi igre, od scena koje su ključne za izradu dobre igre do elemenata svake scene, kao što su: Player, Mapa, Kamera, *EnemyEncounter* i druge. Najzahtjevniji dio je bio pronalaženje funkcija u C# koje obavljaju specifične radnje pa su one detaljno opisane. Najbitniji dio je bio postavljanje dobrog temelja tako da se igra može vrlo jednostavno nadograditi i dodavati nove funkcije koje će ju učiniti boljom.

Bibliografski navod

[1] [Mrežno]. Available: https://bs.wikipedia.org/wiki/RPG. [Pokušaj pristupa 19 Rujan 2018].

[2] [Mrežno]. Available: https://unity3d.com/. [Pokušaj pristupa 19 Rujan 2018].

[3] [Mrežno]. Available: https://www.microsoft.com/hr-hr/windows. [Pokušaj pristupa 19 Rujan 2018].

[4] [Mrežno]. Available: https://www.linux.org/. [Pokušaj pristupa 19 Rujan 2018].

[5] [Mrežno]. Available: https://www.apple.com/mac/. [Pokušaj pristupa 19 Rujan 2018].

[6] [Mrežno]. Available: https://en.wikipedia.org/wiki/C_Sharp_(programming_language). [Pokušaj pristupa 19 Rujan 2018].

[7] [Mrežno]. Available: https://opengameart.org/content/browserquest-sprites-and-tiles. [Pokušaj pristupa 19 Rujan 2018].

[8] [Mrežno]. Available: https://freesound.org/. [Pokušaj pristupa 19 Rujan 2018].

[9] [Mrežno]. Available: https://en.wikipedia.org/wiki/Unity_Technologies. [Pokušaj pristupa 19 Rujan 2018].

[10] [Mrežno]. Available: https://en.wikipedia.org/wiki/Unity_Technologies. [Pokušaj pristupa 19 Rujan 2018].

[11] [Mrežno]. Available: https://docs.unity3d.com/ScriptReference/SceneManagement.Scene.html. [Pokušaj pristupa 19 Rujan 2018].

[12] [Mrežno]. Available: https://docs.unity3d.com/ScriptReference/GameObject.html. [Pokušaj pristupa 19 Rujan 2018].

[13] [Mrežno]. Available: https://unity3d.com/learn/tutorials/topics/interface-essentials/game-view. [Pokušaj pristupa 19 Rujan 2018].

[14] [Mrežno]. Available: https://docs.unity3d.com/ScriptReference/Animator.html. [Pokušaj pristupa 19 Rujan 2018].

[15] [Mrežno]. Available: https://visualstudio.microsoft.com/. [Pokušaj pristupa 19 Rujan 2018].

[16] [Mrežno]. Available: https://docs.unity3d.com/ScriptReference/MonoBehaviour.StartCoroutine.html. [Pokušaj pristupa 19 Rujan 2018].

[17] [Mrežno]. Available: https://docs.unity3d.com/ScriptReference/MonoBehaviour.Update.html. [Pokušaj pristupa 19 Rujan 2018].

[18] [Mrežno]. Available: https://docs.unity3d.com/Manual/TimeFrameManagement.html. [Pokušaj pristupa 19 Rujan 2018].

[19] [Mrežno]. Available: https://docs.unity3d.com/ScriptReference/MonoBehaviour.Start.html. [Pokušaj pristupa 19 Rujan 2018].

[20] [Mrežno]. Available: https://docs.unity3d.com/Manual/JSONSerialization.html. [Pokušaj pristupa 19 Rujan 2018].

[21] [Mrežno]. Available: https://docs.unity3d.com/ScriptReference/Camera.html. [Pokušaj pristupa 19 Rujan 2018].

[22] [Mrežno]. Available: https://docs.unity3d.com/Manual//Prefabs.html. [Pokušaj pristupa 19 Rujan 2018].

[23] [Mrežno]. Available: https://docs.unity3d.com/ScriptReference/Object.Instantiate.html. [Pokušaj pristupa 19 Rujan 2018].

[24] [Mrežno]. Available: https://docs.unity3d.com/ScriptReference/Vector3.Distance.html. [Pokušaj pristupa 19 Rujan 2018].

[25] [Mrežno]. Available: https://docs.unity3d.com/ScriptReference/BoxCollider2D.html. [Pokušaj pristupa 19 Rujan 2018].

[26] [Mrežno]. Available: https://docs.unity3d.com/ScriptReference/Sprite.html. [Pokušaj pristupa 19 Rujan 2018].

[27] [Mrežno]. Available: https://docs.unity3d.com/Manual/Tilemap-CreatingTiles.html. [Pokušaj pristupa 19 Rujan 2018].

[28] [Mrežno]. Available: https://docs.unity3d.com/ScriptReference/Grid.html. [Pokušaj pristupa 19 Rujan 2018].

[29] [Mrežno]. Available: https://docs.unity3d.com/Manual/Layers.html. [Pokušaj pristupa 19 Rujan 2018].

[30] [Mrežno]. Available: https://unity3d.com/learn/tutorials/s/animation. [Pokušaj pristupa 19 Rujan 2018].

[31] [Mrežno]. Available: https://docs.unity3d.com/Manual/AnimationClips.html. [Pokušaj pristupa 19 Rujan 2018].

[32] [Mrežno]. Available: https://unity3d.com/learn/tutorials/topics/animation/animator-controller. [Pokušaj pristupa 19 Rujan 2018].

[33] [Mrežno]. Available: https://docs.unity3d.com/Manual/AnimationStateMachines.html. [Pokušaj pristupa 19 Rujan 2018].

[34] [Mrežno]. Available: https://docs.unity3d.com/ScriptReference/MonoBehaviour.Awake.html. [Pokušaj pristupa 19 Rujan 2018].

[35] »Wikipedija,« [Mrežno]. Available: https://bs.wikipedia.org/wiki/RPG.

[36] [Mrežno]. Available: https://en.wikipedia.org/wiki/MacOS. [Pokušaj pristupa 19 Rujan 2018].

[37] [Mrežno]. Available: https://www.apple.com/hr/iphone/. [Pokušaj pristupa 19 Rujan 2018].

Popis Slika

Popis priloga

Uz ovaj rad priložen je CD na kojemu se nalazi ovaj rad, igra te sve komponente za izradu igre.