Razvoj 2D avanture u Unity okruženju

Salopek, Danko

Sveučilište Jurja Dobrile u Puli
Fakultet informatike

DANKO SALOPEK

RAZVOJ 2D AVANTURE U UNITY OKRUŽENJU

Završni rad

Rujan, 2019.

Sveučilište Jurja Dobrile u Puli
Fakultet informatike

DANKO SALOPEK

RAZVOJ 2D AVANTURE U UNITY OKRUŽENJU

Završni rad

JMBAG: 0303069638, redoviti student
Studijski smjer: Informatika

Predmet: Programiranje
Znanstveno područje: Društvene znanosti
Znanstveno polje: Informacijske i komunikacijske znanosti
Znanstvena grana: Informacijski sustavi i informatologija
Mentor: doc. dr. sc. Tihomir Orehovački

Rujan, 2019.

IZJAVA O AKADEMSKOJ ČESTITOSTI

Ja, dolje potpisani Danko Salopek, kandidat za prvostupnika informatike ovime izjavljujem da je ovaj Završni rad rezultat isključivo mojega vlastitog rada, da se temelji na mojim istraživanjima te da se oslanja na objavljenu literaturu kao što to pokazuju korištene bilješke i bibliografija. Izjavljujem da niti jedan dio Završnog rada nije napisan na nedozvoljen način, odnosno da je prepisan iz kojega necitiranog rada, te da ikoji dio rada krši bilo čija autorska prava. Izjavljujem, također, da nijedan dio rada nije iskorišten za koji drugi rad pri bilo kojoj drugoj visokoškolskoj, znanstvenoj ili radnoj ustanovi.

Student

_____

U Puli, _____, _____ godine

IZJAVA

o korištenju autorskog djela

Ja, Danko Salopek dajem odobrenje Fakultetu informatike u Puli, kao nositelju prava iskorištavanja, da moj završni rad pod nazivom Razvoj 2d avanture u Unity okruženju koristi na način da gore navedeno autorsko djelo, kao cjeloviti tekst trajno objavi u javnoj internetskoj bazi Sveučilišne knjižnice Sveučilišta Jurja Dobrile u Puli te kopira u javnu internetsku bazu završnih radova Nacionalne i sveučilišne knjižnice (stavljanje na raspolaganje javnosti), sve u skladu s Zakonom o autorskom pravu i drugim srodnim pravima i dobrom akademskom praksom, a radi promicanja otvorenoga, slobodnoga pristupa znanstvenim informacijama.

Za korištenje autorskog djela na gore navedeni način ne potražujem naknadu.

U Puli, _____ (datum)

Potpis

_____

Sažetak

Ovaj rad opisuje koncept razvoja 2D igre u unity okruženju. Igra se sastoji od dvije scene. Prva scena sastoji se od glavnog izbornika. Druga scena sastoji se od levela kojeg igrač mora proći. Kamera prikazuje bočni pogled na igrača koji mora izbjeći sve prepreke i doći do kraja levela uz pomoć alata koji su mu dani (uže i baklje). U radu se kratko opisuje unity editor prozori i neke od komponenata koje se koriste u njemu. Opisuju se neke od glavnih funkcija koje se skoro uvijek koriste prilikom programiranja u Unityju. Nakon toga je opisan postupak izrade objekata koji se nalaze u igri i opis rada skripti koji se nalaze na njima.

Ključne riječi: Unity Engine, Through the Dark, C#, objekt

Abstract

This paper describes the concept of developing a 2D game in a unity environment. The game consists of two scenes. The first scene consists of the main menu. The second scene consists of a level that the player must go through. The camera shows a side view of a player who must avoid all obstacles and reach the end of the level with the tools given to him (ropes and torches). The paper briefly describes the unity editor windows and some of the components used in it. Paper describes some of the main functions that are almost always used when coding in unity. After that, the process of creating the objects that are in the game is described and the operation of the codes located on them are described as well.

Keywords: Unity Engine, Through the Dark, C #, Object

Sadržaj

1. Uvod

Tema ovog završnog rada je izrada 2D avanture igre u Unity Engine-u. Zadatak je bio izraditi igru u unity okruženju, koristeći sve alate koje sam unity pruža i dodatne alate kao što su Visual studio code i Photoshop. Igra je primarno rađena za desktop računala tj. za Windows i Linux operacijske sustave. Kreiranje ove igre inspirirano je igrom Spelunky koja je na tržište izašla 2008. godine koju je napravio Mossmouth, LCC studio. Through The Dark napravljena je slično kao Spelunky samo više se bazira na istraživanje uz sporiji tempo. Što igrač ide dublje kroz mapu težina prolaženja se povećava i igrač susreće nove neprijatelje i nove zamke koje mora izbjeći. Igrač ima jedan život i ako umre mora cijelu mapu prolaziti iz početka.

Video igre su svakodnevica u današnjem svijetu. Područje video igara raste i svake godine objavljuje se sve više i više igara od kojih većina dobrih i inovativnih igara koje su napravljene od indie developera prođe ispod radara ljudi. Dok AAA kompanije izbacuju nedovršene igre po cijeni pune igre koja inače kod AAA developera iznosi 60 eura. Uz to izbacuju dijelove igara koje se mogu smatrati kao dijelovi koji bi se već trebali nalaziti u igri kao dodatak koji se može kupiti za dodatne novce.

Slijede 3 poglavlja čija je svrha upoznavanje sa sučeljem Engine-a te uvod u klase i funkcije koje se koriste prilikom implementacije. Naglasak rada je na programiranju u C#-u i implementaciji raznih funkcionalnosti igre. Temu sam odabrao jer me osobno zanima razvoj video igara i to je nešto čime bi se htio baviti u budućnosti.

2. Unity Engine

2.1 Općenito

Unity Engine je softver koji tvorcima igara pruža potreban skup alata za brzu i učinkovitu izgradnju igara. Pruža okvir za razvoj igara koji podržava i okuplja nekoliko osnovnih područja. Pruža mogućnost unošenja slika ili asseta , s drugog softvera, poput Maya, 3ds Max ili Photoshopa. Sastavljanje tih asseta u scene. Dodavanje svjetla, zvuka, posebnih efekata, fizike i animacije, interaktivnost i logiku igranja; i uređivanje, uklanjanje pogrešaka i optimiziranje sadržaja za ciljne platforme.

Koristi se za stvaranje trodimenzionalnih, dvodimenzionalnih, virtualnih igara, kao i simulacija. Iako je prvobitno razvijen za video igrice koristi se i izvan industrije video igrica poput filma, automobila, arhitekture, inženjerstva i građevine [1].

Unity je popularan kod hobi programera i u AAA studijima. Koristi se za stvaranje igara poput Pokemon Go, Rimworld, Cuphead i još mnogo toga. Programeri ga vole zbog API-ja za skripte C# i ugrađene Visual Studio integracije. Unity također nudi JavaScript kao skriptni jezik i MonoDevelop kao IDE onima koji žele alternativu Visual Studio.

Umjetnici ga vole jer dolazi s moćnim alatima za animaciju koji pojednostavljuju izradu vlastitih 3D kreacija ili izradu 2D animacija od nule. U njemu se jednostavno može animirati gotovo sve.

2.2 Povijest

Pokrenut je 2005. godine s ciljem da "demokratizira" razvoj igara pružajući dostupnosti izrade većem broju programera

Unity 2.0 pokrenut je 2007. s oko 50 novih značajki. Izdanje je uključivalo optimizirani engine za detaljna 3D okruženja, dinamičke sjene u stvarnom vremenu, usmjerenih svjetala i reflektora, reprodukciju videa i druge

značajke. U izdanju su dodane i značajke pomoću kojih programeri mogu lakše surađivati. Uključio je mrežni sloj za programere za izradu igara za više igrača [2].

Unity 3.0 pokrenut je u rujnu 2010. godine sa značajkama koje proširuju grafički engine za stolna računala i konzole. Osim Android podrške, Unity 3 sadržavao je integraciju alata Beast Lightmap Illuminate Labs ', odloženo prikazivanje, ugrađeni uređivač stabala, prikazivanje izvornih fontova, automatsko UV preslikavanje i audio filtre, između ostalog [3].

Unity 4.0 u studenom 2012 . donosi DirectX 11 i Abobe Flash support, novi animacijski alat zvan Mecanim, i pristup Linux pregledu [4].

Unity 5 je izašao u 2015. s poboljšanjem svjetlosti i zvuka. Dodaje i WebGL koji developerima daje mogućnost da svoje igre stavljaju na web preglednike bez potrebnih dodataka za igrače [23]. Pruža real-time globalnu iluminaciju, pregled svjetlosnih mapa, Unity Cloud, novi audio system i Nvidia PhysX3.3. Ova generacija donosi Cinematic Image Effects kako bi igre izgledale bolje. Unity 5.6 dodaje nove svjetlosne i particle sisteme i support za Nintendo Switch, Facebook, Gameroom, Google Daydream VR i Vulkan [5].

Unity 2017 donosi renderiranje grafike u realnom vremenu, kreiranje svijeta ili levela, analitičke operacije uživo i izvještavanje o performansama. Unity 2017.2 naglasio je planove izvan videoigara. To je uključivalo nove alate kao što su Timeline koji je omogućio programerima da povuku i ispuste animacije u igre i Cinemachine, pametni fotoaparat unutar igara. Unity 2017.2 je također integrirao Autodesk-ove 3DS Max i Maye alate u mehanizam Unity-a za pojednostavljenje dijeljenja imovine u procesu iteracije u igri [6].

Unity 2018.1 unosi Scriptable Render Pipeline dizajniran da programerima olakša stvaranje vrhunske grafike. Dolazi u dva oblika: High-Definition Pipeline Rendering za konzole i PC projekte i Lightweight Render Pipeline koji je optimiziran za mobilne i razne uređaje „stvarnosti“ (virtualni, prošireni, miješani i tako dalje) [7].

Unity 2019. donosi ProBuilder 4.0 (jedinstveni hibrid 3D modeliranja i alata za oblikovanje nivoa), 2D animacije sa zamjenjivim dijelovima slika, 2D s

Shader Graph, Ostala poboljšanja u Visual Effect Graph & Shader Graph, DSPGraph Audio mixing/rendering sistem i još puno toga [8].

3. Izrada Through the Dark

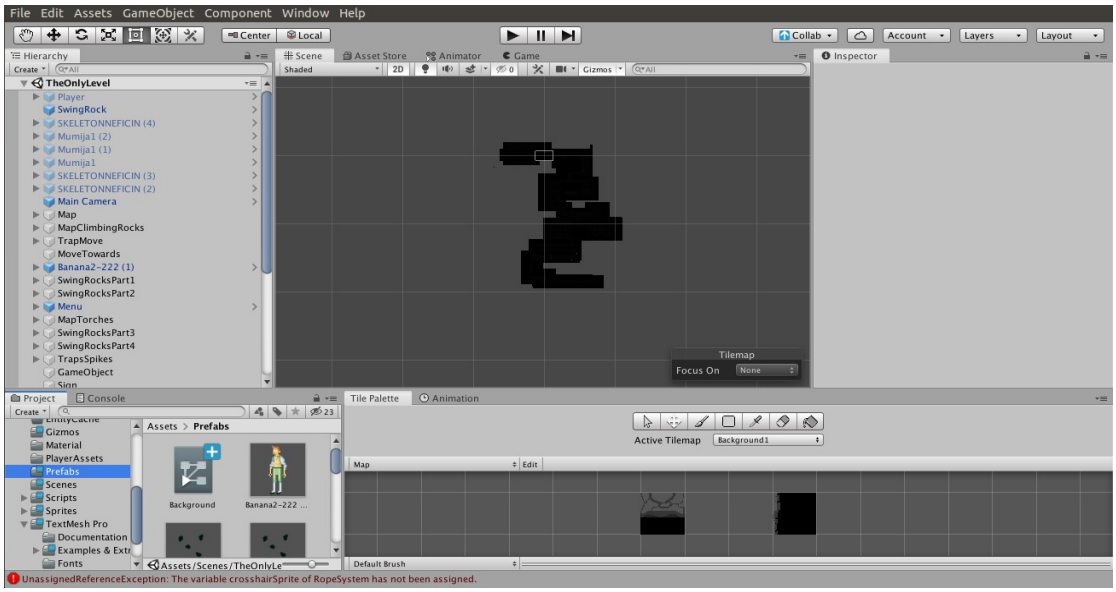
Izrada igre u Unity-u bit će prikazana pomoću igre „Through the Dark”. Through the Dark se sastoji od jednog levela koji postepeno postaje sve teži i teži. Cilj igre je da glavni lik „John” dođe do kraja i uzme zlato. Koristeći uže kako bi preskakao područja koja ne može normalno preskočiti i baklje s kojima si osvjetljuje put jer je cijela mapa mračna.

Igra je pokrenuta 2019.2 verzijom Unityja i 2d extras dodatkom s GitHuba koji omogućuje izrađivanje mape u samom Unityju. Vanjski alati korišteni su Visual Studio, Photoshop.

Glavni objekt igre je igrač (eng. player) koji se kreće pomoću skripte i komponenata. Skripte gledaju kada igrač pritisne određenu tipku na tipkovnici i prema tome se objekt koji reprezentira igrača miče ili ima interakciju s dijelovima mape. Kada igrač dotakne neke od zamka ili neprijatelja on umire i nakon određenog vremena se mapa resetira.

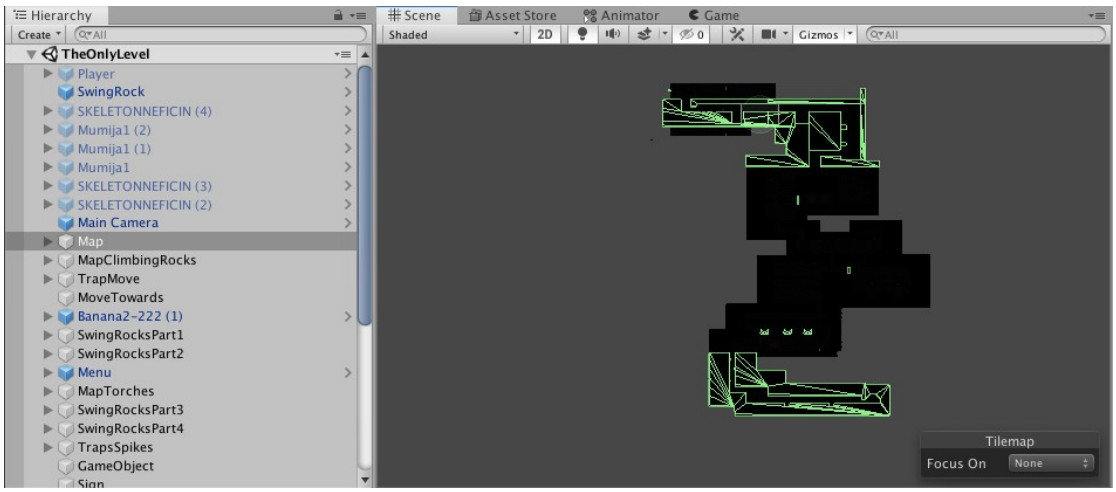
4. Unity editor

Slika 1 prikazuje dio unity-a u kojemu se vrši izrada igre . Sadrži prozore koji se mogu aktivirati ili deaktivirati. Neki od tih prozora su potrebni (Hierarchy, Scene, Game, Project, Console, Inspector), bez kojih se ne može izrađivati mapa jer oni sadrže objekte koji se nalaze na mapi ili objekte u cijelom projektu.



Slika 1: Prikaz izgleda unity editora

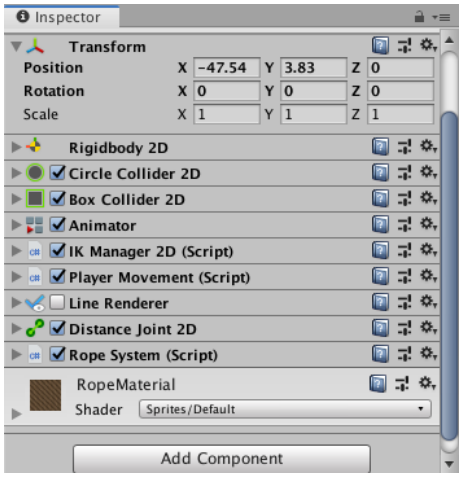
Na slici 2 prikazana je Hijerarhija (eng. Hierarchy) i Scena (eng. Scene). Hijerarhija prikazuje sve objekte koji se nalaze u trenutno aktivnoj Sceni (scena je trenutni level). Scena je prozor koji developerima prikazuje gdje se objekti iz Hijerarhije nalaze u trodimenzionalnom prostoru.



Slika 2: Prozori hijerarhije i scene

Na slici 3 prikazan je Inspektor koji prikazuje detaljan prikaz odabranog objekta tj. prikazuje sve elemente koji se nalaze na odabranom objektu i gdje se odabrani objekt nalazi u sceni. Pozicija objekta se može mijenjati u sceni ili

u elementu inspektora pod nazivom Transform. Pomoću inspektora objektu se mogu dodavati ili oduzimati elementi.



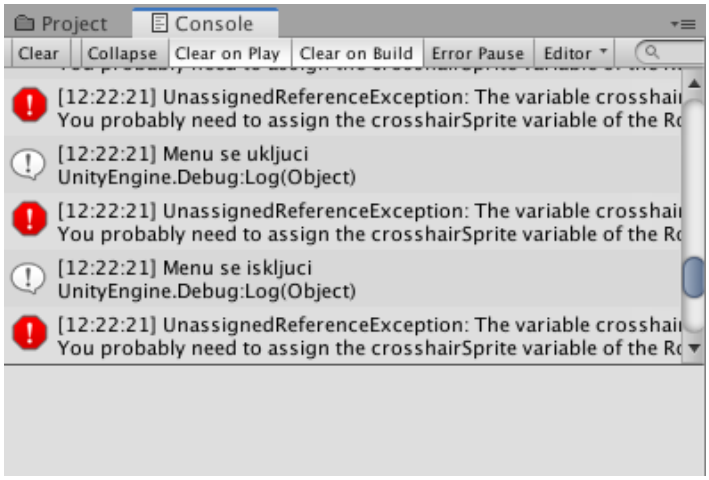
Slika 3: inspektor

Slika 4 prikazuje prozor projekta. Projekt prikazuje sve komponente koje se nalaze u projektu. Kao što su Prefab (objekti koji su napravljeni jednom i mogu se opet koristiti), druge mape koje se mogu pokrenuti i s njima se mijenjaju hijerarhija i scena. Iz projekta se objekti mogu povući u hijerarhiju ili scenu.

Slika 4: Izgled prozora Projekt

Console je prikazana na slici 5 i služi za pronalaženje problema u dijelovima skripti ili dijelovima scene. Postoje dvije vrste prikazivanja grešaka. Crveni uskličnik prikazuje greške koje se moraju popraviti prije pokretanja scene i najčešće onemogućuju developeru da pokrene scenu. Druga vrsta greške je

žuti uskličnik koji govori da nešto nedostaje, ali neće zaustaviti pokretanje scene.



Slika 5: Prozor konzole

4.1 Komponente

Objekti (eng. Game Objects) su temeljni dijelovi Unityja koji predstavljaju likove, rekvizite i mape. Sami po sebi su beskorisni jer su prazni spremnici koji dobivaju vrijednost samo ako sadrže neke komponente. Svaki objekt sadrži Transform komponentu koja prikazuje točnu poziciju samog objekta u sceni. Komponente se dodaju i brišu u inspektoru [21].

Jedna od važnih komponenata za bilo koju igru je **Collider** koja objektu dodaje fizičku prisutnost na mapi kako bi dobili interakciju između objekata. Collideri se dijele na 2D i 3D. Jedan objekt može imati više različitih ili istih Collidera od kojih svi ne moraju biti za fiziku. Neki od njih mogu biti okidači (eng. Trigger). Collideri koje koristi „Through the Dark" su **BoxCollider2D** koji ima pravokutan oblik i najčešće se koristi za označavanje tla levela. **PolygonCollider2D** koji može primijeniti bilo koji oblik i koristi se kada je potrebna preciznost kod sudaranja objekata, **CircleCollider2D** je uvijek krug koji ima svoj radius i tako se povećava ili smanjuje. Colliderima koji nisu okidači može se dodati materijal koji određuje kakvo će biti trenje prema drugim objektima. Okidači su vrsta Collidera koji nemaju fizičku prisutnost, nego djeluju kao zona koja reagira samo ako neki drugi object uđe u nju.

Ovisno o tome koji je naziv objekta ili koji mu je tag dodijeljen radi neke funkcije koje su mu dodane u kodu [11].

RigidBody2D omogućuje da na objekt utječe fizika. Na objekt koji sadrži rigidbody2D utječe gravitacija. Ako se objekt koji sadrži rigidbody2D sudari s nekim drugim objektom njegova sila se može prebaciti na objekt koji ima rigidbody2D. Rigidbody2D utječe na sve Collidere objekta koji nisu okidači. Kada se doda na objekt ne stvara sa sobom Collider, već se Collider mora dodati posebno, ako zaboravimo dodati Collider objekt će se konstantno micati u smjeru gravitacije koja utječe na njega. Postoji nekoliko različitih verzija tj. opcija koje mogu biti odabrane svaka sa svojim posebnim svojstvima. Simulated opcija određuje hoće li tijelo biti u interakciji s drugim fizičkim objektima u sceni. Kada je ova opcija uključena, prilikom simulacije objekt se može sudarati s drugima Colliderima. Ako je opcija isključena na tijelo neće djelovati niti jedna sila niti će ono samo moći djelovati na druge objekte. **Dynamic** opcija dizajnirana je za kretanje u simulaciji. Na raspolaganju ima čitav niz svojstava kao što su ograničena masa i povlačenje, a na njega utječe gravitacija i ostale sile. Dinamičko tijelo će se sudariti sa svim drugim tjelesnim tipovima i najinteraktivnije je za tjelesne tipove. Zbog dinamične prirode i interaktivnosti sa svime oko sebe zahtjeva najviše resursa. Sva svojstva rigidbody2D dostupna su s ovim tipom tijela. **Kinematic** je tip osmišljen za kretanje pod simulacijom, ali samo pod vrlo eksplicitnim korisničkim nadzorom. Dok na dynamic utječe gravitacija i sile, na kinematsko ne utječu. Iz tog razloga, to je brzo i ima manju potražnju za resursima sustava. Ne sudara se s drugim objektima osim s objektima koji imaju dinamički rigidbody2D [12].

Kamera je komponenta koja prikazuje što igrač vidi . Broj kamera i njihovi položaji su neograničeni i mogu se manipulirati koliko je god potrebno. Neke igre s više igrača zahtijevaju kameru za svakog igrača, druge igre imaju samo jednu kameru koja prikazuje sve igrači. Sastoji se od dvije perspektive: ortografska i perspektivna. Ortografska kamera nema nikakvu perspektivu i najčešće se koristi za 2D ili izometrijske igre. Perspektivna kamera se ponaša

sličnije ljudskom oku i ima svoju perspektivu. Obično se koristi za 3D igre, ali moguće ju je iskoristiti i za 2D igre ako su dizajnirane na poseban način [22].

Animatorski sustav zasnovan je na konceptu animacijskih isječaka, koji sadrže podatke o tome kako bi određeni objekti tijekom vremena trebali mijenjati položaj, rotaciju ili druga svojstva. Svaki isječak može se zamisliti kao pojedinačni linearni snimak. Animator sadrži sve animacije koje ima neki objekt i pomoću njega se mogu određivati uvjeti kada će se pozivati koja animacija ovisno o tome kada je potrebna i je li njezin uvjet istinit [24].

Prefab je objekt koji se sprema u projekt kako bi se taj objekt mogao koristiti kasnije na nekom drugom levelu bez da taj objekt od početka kreira. Ako se prefab promjeni u projektu sve njegove kopije na svim mapama se mijenjaju u objekt koji je u projektu, ali ako se promjeni kopija koja se nalazi na nekoj mapi ta kopija je samo promijenjena za sebe samu, naravno postoji drugi koje omogućuje da prefab uzme karakteristike kopije i s time se promjene i ostale kopije [14].

Layer služi za izradu 2D igre. Objekti u 2D igri se klasificiraju u ovisnosti o tome na kojoj se razini nalaze. Objekti koji imaju veći broj razine će biti vidljivi ispred objekata koji imaju manju razinu layera [15].

5. Osnovne funkcije Unity okruženja

U nastavku bit će opisane funkcije koje su gotovo uvijek potrebne prilikom kodiranja igre u Unityju. Sljedeće funkcije su standard i ne moraju se kreirati.

Svaka igra ima „frame rate” (broj koliko puta se izmjenjuju sličice u sekundi) za neku igru. Taj broj ovisi najviše o hardveru na kojemu se igra pokreće. Najbolje je imati 60 fps.

Update funkcija se izvršava svaki frame i zbog toga ju je najbolje koristiti za detektiranje inputa playera ili pokretanja nekog objekta koji se ne pomiče pomoću fizike i ne sadrži rigidbody2D [17].

FiexdUpdate je funkcija slična Update-u jedina razlika je što se poziva svakih nekoliko milisekundi. Update i FixedUpdate su jednako dobri, kada će

se koristiti jedan ili drugi ovisi o situaciji. Ali problem se javlja kada hardver na kojemu se igra izvodi slab. Što bi značilo da će broj slika po sekundi biti malen. Isto to se dešava i s hardverom koji je puno jači od potrebnog samo što se tada javlja problem prevelikog broja sličica. Tako da na kraju za pomicanje objekata s fizikom bolji FixedUpdate jer se ponavlja svakih nekoliko milisekundi [27].

Start se poziva prije bilo koje update funkcije. Zove se samo jednom kada se objekt inicijalizira. Najčešće se koristi za postavljanje varijabli potrebnih za daljnji rad koda [16].

OnTriggerEnter2D se aktivira samo jednom kada se aktivira zona [20]. **OnTriggerStay2D** poziva se svaki kadar dokle god je objekt koji je aktivirao okidač nalazi u zoni okidanja [18]. **OnTriggerExit2D** poziva se jednom nakon što objekt koji se nalazi u zoni izađe iz zone [19].

6. Implementacija

U ovom poglavlju biti će objašnjeno kako su sve funkcionalnosti igre implementirane pomoću C#-a i Unity-a.

6.1 Kamera

Kamera je objekt koji prikazuje igraču igru. Kamera omogućuje igraču pogled na dio igre koji je potreban. Zbog koda kamera nikad ne gleda direktno igrača, već gleda neku točku između igrača i pozicije miša. Kod koji pomiče kameru se naziva **CameraFolow** i sadrži sljedeće varijable:

```
public GameObject player;
public Vector3 offset;
public float mouse=2f;
```

Sljedeća linija koda pronalazi objekt s tagom „Player” u sceni i postavlja varijablu player na taj objekt. Tag se stavlja na objekte u sceni kako bi im

stavili oznaku. Pomoću te oznake u kodu lakše pristupimo tome objektu ili skupini objekata.

```
player = GameObject.FindGameObjectWithTag("Player");
```

Offset se postavlja na (0,0,-10) koji će kasnije biti korišten za udaljivanje kamere prilikom izvođenja koda.

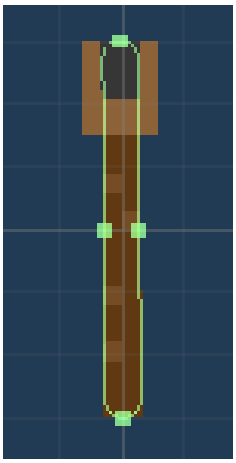
```
transform.localPosition = new Vector3 (player.transform.position.x,
                                       player.transform.position.y,-10);
offset = transform.position - player.transform.position;
```

Nastavak se nalazi u LateUpdate koji se izvršava nakon svih Update() funkcija. DesiredPositon je zbroj dvije Vector3 varijable. MousePosition je pozicija miša. Varijabla mousePosition se prosljeđuje u CameraMain funkciju koja vraća Vector3 pozicije miša u odnosu na poziciju kamere. Izračunava se desiredPosition i transform kamere se postavlja na desiredPosition. Što postavlja kameru na određenu udaljenost.

```
void LateUpdate()
{
    Vector3 desiredPosition= player.transform.position + offset;
    Vector3 mousePosition = Input.mousePosition;
    mousePosition = Camera.main.ScreenToViewportPoint(mousePosition);
    desiredPosition = desiredPosition + (mousePosition / mouse);
    transform.position = desiredPosition;
}
```

6.2 Baklja

Baklja (eng. torch) je prikazana na slici 6 i ona je jedini izvor svjetla u igri. Igrač mora koristiti baklju kako bi osvijetlio put ispred sebe, izbjegao sve zamke i prešao sve zapreke.



Slika 6: Prefab Torch

Sam prefab ima jednu skriptu na sebi koji služi za promjenu intenziteta svjetla. Objekt baklja se koristi u drugim skriptama.

U **FireFlicker** skripti se većina varijabli postavlja u inspektoru dok se jedino pointLight i time postavljaju u Start funkciji.

```
Light pointLight;
float lightInterval;
public float minInterval, maxInterval;
public float firstTime, time;
public int torchCount=1;
```

PointLight pronalazi komponentu „Light” koja se nalazi na objektu na kojemu je skripta. Komponenta Light osvjetljuje područje oko baklje ovisno o intenzitetu svjetla. Sadrži radijus koji odlučuje do koje će udaljenost svjetlo imati utjecaj. Time se postavlja na vrijednost firstTime kako bi se sačuvala vrijednost firstTime za kasnije.

```
void Start()
    {
        pointLight = GetComponent<Light>();
        time = firstTime;
    }
```

Time.deltaTime smanjuje vrijednost time za 1 svaku sekundu to u ovom kontekstu služi kao tajmer. Ako time prođe ispod 0 interval svjetlosti se promjeni i zatim se time postavlja natrag na firstTime kako bi proces počeo iz početka. S time se dobiva izgled treperenja svjetla. Širina u kojoj se odabire nasumični broj ovisi o broju baklji koje su blizu jedna drugoj.

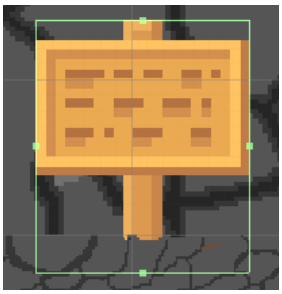
```
private void FixedUpdate()
{
    time -= Time.deltaTime;
    if (time <= 0)
    {
        lightInterval = Random.Range(minInterval/torchCount,
                                     maxInterval/torchCount);
        pointLight.intensity = lightInterval;
        time = firstTime;
    }
```

Kada baklja dođe u zonu koja služi kao okidač torchCount se poveća za 1, ako baklju koja je u zoni pokupi igrač ili sama izađe iz zone torchCount će se smanjiti za 1.

```
private void OnTriggerEnter2D(Collider2D collision)
{
    if (collision.tag=="Torch")
    {
        Debug.Log("torch++");
        torchCount++;
    }
}
```

6.3 Sign

Znak (eng. sign) prikazan na slici 7 je objekt koji nije napravljen u prefab jer se koristi samo u mapi gdje je tutorijal kako bi pomogao igraču da upozna kontrole.



Slika 7: Izgled znaka

Sadrži skriptu **Tutorijal** koji ima jednu varijablu canvas koja služi kao referenca na drugu skriptu ActivateCanvas kako bi pristupio varijabli secretText.

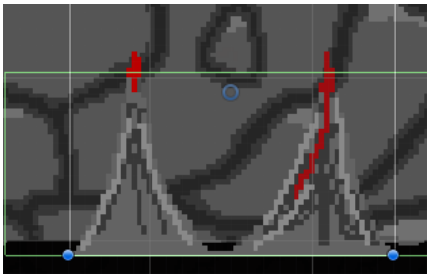
```
public ActivateCanvas canvas;
void Start()
{
    canvas = GameObject.FindGameObjectWithTag("SettingCanvas").
                                        GetComponent<ActivateCanvas>();
    canvas.secretText.text = ("");
}
```

Ako igrač uđe u zonu okidanja pokreće se OnTriggerEnder2D funkcija koja izmjenjuje secretText. I zatim se skriveni tekst pojavljuje na ekranu. Ako se igrač udalji iz zone okidanja, skriveni tekst se vraća na prazno

```
void OnTriggerEnter2D(Collider2D col)
{
    if (col.CompareTag("Player"))
    {
        canvas.secretText.text = "Press Q to pick up close torches.
                                Press LEFT CLICK to throw torches";
    }
}
```

6.4 Bodlje

Bodlje (eng. Spikes) prikazane na slici 8 su objekti koji se nalaze posvuda na mapi i ubijaju igrača ako ih on dotakne.



Slika 8: Spikes

Kod **DeathZone** ima samo jednu varijablu „wait" koja je vrijeme čekanja reseta levela.

```
public float wait;
```

Cijeli DeathZone se sastoji od okidača i jedne **IEnumerator** funkcije. Okidač se aktivira i objekt s kojim igrač igra nestaje s ekrana tj. skripta ga postavlja da je nevidljiv i da se ne može koristit. Zatim se poziva StartCoroutine funkcija koja je potrebna ako radimo s IEnumeratorima jer ih inače ne možemo drugačije pokrenuti. Poziva se funkcija WaitForRestart. IEnumeratori su drugačiji od običnih linija koda jer se izvršavaju normalno sve do yield return new waitForSeconds koji predstavlja tajmer na kojemu kod čeka. Kada taj tajmer dođe na 0 nastavlja se izvršavanje ostalih linija koda [19]. U ovom slučaju jedina linija koja je ostala je resetiranje trenutno aktivne scene.

```
private void OnTriggerEnter2D(Collider2D collision)
{
    if (collision.CompareTag("Player"))
    {
        collision.gameObject.SetActive(false);
        StartCoroutine(WaitForRestart(wait));
    }
}
```

```
private IEnumerator WaitForRestart(float wait)
{
    yield return new WaitForSeconds(wait);
    SceneManager.LoadScene(SceneManager.GetActiveScene().name);
}
```

Postoji druga varijanta bodlji koja se može kretati. Koristi **TrapMove** skripta. TrapMove sadrži 3 varijable moveTowards koja reprezentira objekt u sceni prema kojemu se kreće zamka, speed koja predstavlja brzinu kojom će se zamka kretati prema moveTowards objektu. OnTriggerEnter2D postavlja trapActivate na true kako bi se pokrenuo proces u Update.

```
public GameObject moveTowards;
public float speed;
public bool trapActivate=false;

private void OnTriggerEnter2D(Collider2D collision){
    if(collision.tag =="Player"){
        trapActivate=true;
    }
}
```

Update funkcija ima dva if-a gdje prvi if gleda da li se zamka aktivirala , dok drugi gleda udaljenost zamke od objekta prema kojemu se kreće. Ako su obadva if-a točna bodlje će se kretati prema prije određenom objektu uz određenu brzinu.

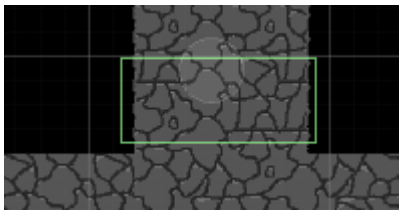
```
void Update()
{
    if(trapActivate){
        if(Vector2.Distance(new Vector2(moveTowards.transform.position.x,
                                        moveTowards.transform.position.y),
                            new Vector2(gameObject.transform.position.x,
                                        gameObject.transform.position.y))
                                                >1f){
```

```
        float step =  speed * Time.deltaTime; // calculate distance to move
        transform.position = Vector3.MoveTowards(transform.position,
                                    moveTowards.transform.position, step);
    }
```

Postoji i drugačija varijanta trenutno prikazane skripte. Gdje umjesto jednog objekta prema kojemu se bodlje miču imamo 2 objekta. Jedan objekt se nalazi na poziciji gdje se bodlje postavljene u sceni i drugi objekt se nalazi negdje drugdje. Stvar kod ove verzije je da se bodlje nikada ne prestaju kretati tj. kada dođu do jednog objekta okrenu se i vraćaju se do prvog objekta i tako dalje do beskonačnosti.

6.5 Sljedeća zona

S obzirom da „Through the Dark" ima samo jednu ogromnu mapu, zbog pokušaja optimizacije svi dijelovi mape, osim dijela u kojemu je igrač trenutno, su isključeni. Postoje sljedeće zone koje su okidači za aktiviranje sljedećeg dijela mape i isključivanje prošlog dijela mape. S obzirom da bi bilo čudno da igrač ne vidi mapu kada dođe blizu jedne od tih zona isključeni su samo Collideri svih objekata.



Slika 9: Izgled sljedeće zone

Aktivacija i deaktivacije se postiže pomoću skripte **NextStage.** Varijable u NextStage su liste tipa GameObject što znači da u njih možemo stavljati objekte iz mape. Test u sebi sadržava sve objekte iz prijašnjeg dijela mape, dok nextStage sadrži objekte koji se moraju aktivirati.

```
public List <GameObject> test, nextStage;
```

Kada player aktivira okidač pokreće se funkcija SetStage().

```
private void OnTriggerEnter2D(Collider2D player)
{
    if(player.tag == "Player")
    {
        SetStage();
    }
}
```

SetStage u sebi sadrži dvije foreach petlje. Prva petlja prolazi kroz listu nextStage , za svaku komponentu provjerava postoji li collider, ako postoji uključi ga. Zatim dolazi do provjere oznake objekta gdje gleda da li oznaka „Mumija” ili „Skeleton” jedan od neprijatelja u mapi, ako je aktivira ih. Za razliku od ostalih objekata neprijatelji su nevidljivi kako im se ne bi pokrenule animacije i kako zbog nepostojanja poda ispod njih ne bi zauvijek padali u beskonačnost. Ako objekt nije jedan od neprijatelja poziva se druga petlja koja užima sve Collidere djece objekta i aktivira ih sve.

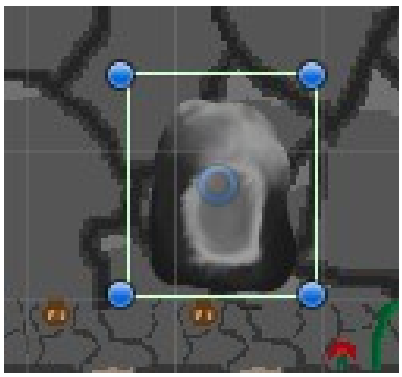
```
void SetStage(){
    foreach (GameObject next in nextStage){
        if(next.GetComponent<Collider2D>() != null){
            next.GetComponent<Collider2D>().enabled=true;
        }
        if(next.tag == "Mumija" || next.tag=="Skeleton"){
            next.SetActive(true);
        }else{
            foreach(Collider2D c in next.
                                    GetComponentsInChildren<Collider2D>()){
                if(c !=null){
                    c.enabled = true;
                    Debug.Log("Coliders Enabled");
                }
```

Drugi dio SetStage funkcije je deaktiviranje svih Collidera, mumija i skeletona. Jedina razlika ovog dijela i prošlog je to što su svi true promijenjeni u false.

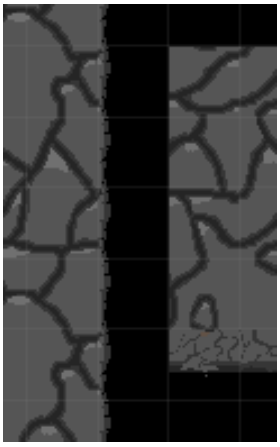
```
foreach (GameObject tt in test){
    if(tt.GetComponent<Collider2D>() !=null){
        tt.GetComponent<Collider2D>().enabled=false;
    }
    if(tt.tag == "Mumija" || tt.tag=="Skeleton"){
        tt.SetActive(false);
    }
    foreach (Collider2D c in tt.GetComponentsInChildren<Collider2D>()){
        if(c!=null){
            c.enabled = false;
        }
```

6.7 Skrivena zona

Slika 11 prikazuje oblik okidača kod kojega player može otvoriti skrivenu sobu, ulaz u skrivenu sobu je zid prikazan na Slici 10.



Slika 11: Kamen koji gasi zid



Slika 10: Izgled zida

HiddenRoom skripta izgleda skoro isto kao i skripta Tutorial jedina razlika je što postoje dvije nove varijable. Varijable lever i hiddenWall se postavljaju u inspektoru. U start funkciji canvas se postavlja na referencu

skripte ActivateCanvas kako bi se promijenio skriveni tekst kao i u znaku. Zatim se zid aktivira ako je slučajno neaktiviran.

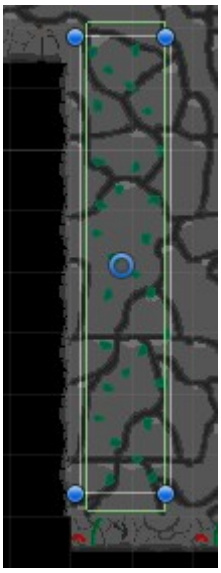
```
public GameObject lever;
public GameObject hiddenWall;
public ActivateCanvas canvas;

void Start()
{
    canvas = GameObject.FindGameObjectWithTag("SettingCanvas")
                                        .GetComponent<ActivateCanvas>();
    canvas.secretText.text = ("");
    hiddenWall.SetActive(true);
}
```

Okidač omogućuje igraču da pritisne dugme i s tim otključa skrivenu sobu.

```
void OnTriggerEnter2D(Collider2D col)
{
    if (col.CompareTag("Player"))
    {
        canvas.secretText.text = "Press X to open a secret room!";
        if ( Input.GetKey("x") || Input.GetKeyDown(KeyCode.X))
        {
            hiddenWall.SetActive(false);
            gameObject.GetComponent<BoxCollider2D>().enabled=false;
            canvas.secretText.text="";
        }
    }
}
```

6.8 Zona penjanja

Predstavlja dio mape gdje igrač mora popeti kako bi napredovao dalje. Prikazana je slikom 12.



Slika 12: Zona penjanja

Sadrži istoimeni skriptu samo na Engleskom jeziku **ClimbZone**. Skripta se aktivira kada igrač aktivira okidač. Cijela se skripta sastoji od OnTriggerEnter2D, OnTriggerStay2D i OnTriggerExit2D. Na ulazak i dokle god je igrač u zoni omogućuje mu se penjanje s postavljanjem varijable climb iz PlayerMovment skripte na istinito. Kada igrač izađe iz zone za climb varijabla se postavlja na netočno i igrač se dalje kreće normalno.

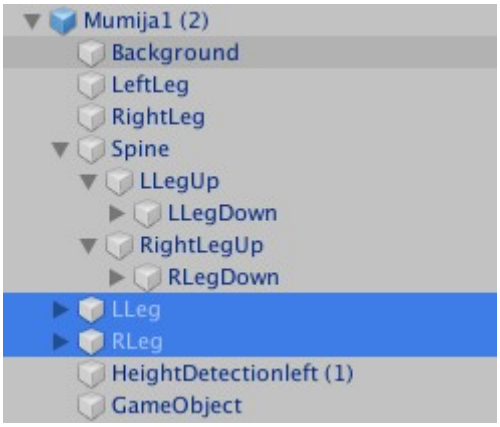
```
private void OnTriggerEnter2D(Collider2D collision)
{
    if (collision.tag=="Player")
    {
        // Debug.Log("clibm=true");
         PlayerMovement.climb = true;
    }
}
```

6.9 Mumija

Prikazan na slici 13 je jedan od dva neprijateljakoje je moguće samo ubiti s bakljom. Na sebi ima 2 Collidera jedan Collider je okidač koji koristi

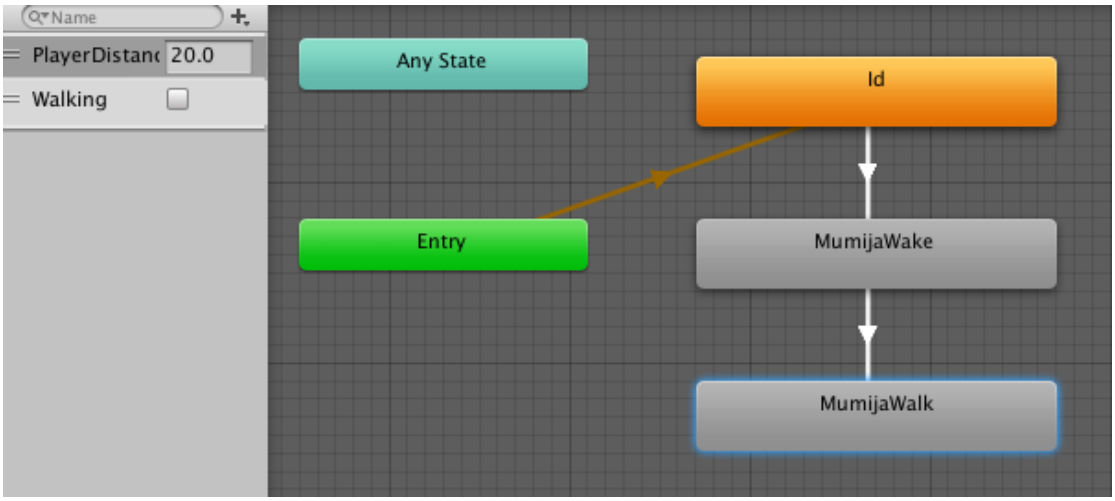
DeathZone skriptu koja ubije igrača ako dođe pre blizu, dok je drugi običan Box Collider s kojim vrši interakciju s ostatkom mape.

Slika 13: Mumija



Slika 14: Djeca mumije

Na slici 14 kost Spine i njegova djeca prikazuju kosti modela s kojima su napravljene animacije. Te kosti koriste **2D Inverse Kinematics (IK)** koji automatski izračunava položaje i rotacije lanaca kostiju koji se kreću prema ciljanom položaju. To olakšava pozicioniranje i animiranje udova u stvarnom vremenu. Tj koriste **LimbSolver2D** skripte koji omogućuje kretanje krajnje određene lokacije na lancu kostiju [29]. Uz pomicanje te određene lokacije pomiču se i ostale kosti modela u odnosu prema toj lokaciji. Pomoću tih IK solvera animiranje je lakše. Takva animacija se zove Bone Animation (koštana animacija) i za razliku od normalnih 2D animacija koje moraju imati mnogo drugačijih slika kako bi se napravila animacija sve se može napraviti s jednom slikom. Mumija ima dvije animacije prikazane na slici 15.



Slika 15: Izgled animatora

Prva animacija MumijaWake se aktivira kada Raycast (nevidljiva linija koja može biti ispucana u bilo kojem smjeru i bilo koje duljine, vraća pogođeni objekt) pogodi igrača i provjeri je li udaljenost manja od nekog broja. Mumija walk se aktivira pomoću MumijaWake animacije koja sadrži event pri kraju animacije koji omogućuje uvijte za izvođenje MumijaWalk animacije.

Mumija sadrži skriptu **EnemyMumija** koji ima varijable.

```
private int direction = -1;
public GameObject detectionOnLeft;
private float PlayerDistance=20;
private Animator anim;
public int destroy=0;
public GameObject player;
public LayerMask playerLayer, groundLayer;
private Rigidbody2D rb;
public GameObject dmgZone;
public float movmentSpeed;
```

Od kojih se detectionOnLeft, playerLayer, groundLayer i movementSpeed postavljaju u inspektoru. Dok se anim, player, rb postavljaju u start funkciji.

Destroy je na početku 0 jer se objekt uništi kada je 2 puta pogođen s bakljom tako da se taj broj povećava u skripti.

```
void Start()
{
    anim = gameObject.GetComponent<Animator>();
    player= GameObject.FindGameObjectWithTag("Player");
    rb = gameObject.GetComponent<Rigidbody2D>();
    gameObject.layer=16;
}
```

Mumija može biti pogođena s bakljom 2 puta nakon toga se uništava.

```
    if(destroy ==2){
        Destroy(gameObject);
    }
```

Ako je walking u animatoru lažan izvodi se sljedeći dio skripte gdje se kreira nevidljiva linija koja detektira kada igrač dođe u određenu udaljenost i tu udaljenost šalje u animator kako bi se aktivirala animacija MumijaWake ako je igrač dovoljno blizu.

```
    if(anim.GetBool("Walking") == false){
        RaycastHit2D playerHit=Physics2D.Raycast(detectionOnLeft.
                                transform.position,  Vector2.left,10f,playerLayer);
        Debug.DrawRay(detectionOnLeft.transform.position, Vector2.left *10);
        if(playerHit.collider != null){
            PlayerDistance = Vector2.Distance(gameObject.transform.position,
                        player.transform.position);
            anim.SetFloat("PlayerDistance", PlayerDistance);
        }
    }
```

Kada je Walking istinit mumija će se kretati. Mumija se kreće u jednom smjeru dok ne dođe do prepreke ili dijela gdje prestaje dio mape, za identificiranje

toga koristi se groundTest koji pomoću RaycastHit2D ispucava liniju ispod sebe i provjerava je li što pogođeno. Ako ništa ne pogodi to znači da je na kraju mape. Ako pogodi nešto, ali je u tom objektu udaljenost groundTest i tog objekta je 0. U obadva slučaja mijenja se localScale objekta s čime se objekt okrene za 180 stupnjeva na y osi i nastavi dalje s kretanjem.

```
else{
        if(transform.localScale.x== 1){
            RaycastHit2D groundTest=Physics2D.Raycast(detectionOnLeft.
                transform.position,
                Vector2.down,2f,groundLayer);
            if(groundTest.collider == null){
                direction=1;
                transform.localScale= new Vector3(-1,1,1);}
            else if(groundTest.distance==0){
                direction=1;
                transform.localScale=new Vector3(-1,1,1);
            }
        }
        // ako se krece desno
        if(transform.localScale.x== -1){
            RaycastHit2D groundTest=Physics2D.Raycast(detectionOnLeft.
                transform.position,
                Vector2.down,2f,groundLayer);
            if(groundTest.collider == null){
                direction=-1;
                transform.localScale= new Vector3(1,1,1);}
            else if(groundTest.distance==0){
                direction=-1;
                transform.localScale=new Vector3(1,1,1);
            }
```

FixedUpdate pomiče mumiju kada je walking. Mimija se kreće po x osi, ovisno o tome je li direction 1 ili -1 mumija će se kretati lijevo ili desno.

```
void FixedUpdate()
{
    if(anim.GetBool("Walking")){
        rb.velocity= new Vector2(movmentSpeed * direction,0);
    }
}
```

WakeToRun je funkcija koja se poziva kada završi animacija WakeUp. Tj animacija ima event na kraju koji aktivira ovu funkciju

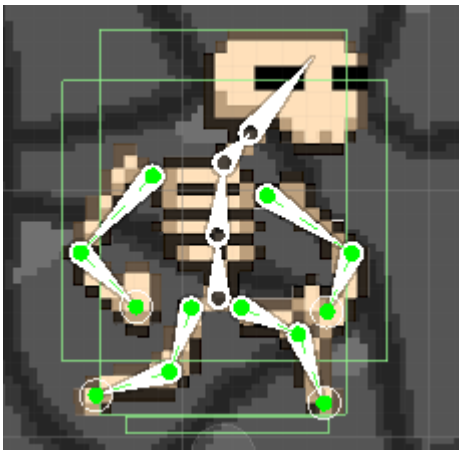
```
private void WakeToRun(){
    anim.SetBool("Walking",true);
    gameObject.layer=15;
    dmgZone.GetComponent<BoxCollider2D>().enabled=true;
}
```

OnCollisionEnter2D se aktivira kada se dva objekta sudare u ovom slučaju kada objekt baklja udari mumiju povećava destroy za 1 i baklja se uništava.

```
void OnCollisionEnter2D(Collision2D torch){
    if(torch.gameObject.tag == "Torch"){
        destroy++;
        Destroy(torch.gameObject);
    }
}
```

6.10 Kostur

Slika 16 prikazuje Kostura (eng. Skeleton). Isto koristi IK limb solvere da se može animirati. Jedina razlika je što mumija koristi solvere za noge dok kostur koristi solvere za noge i ruke. Sadrži dva više collider od mumije jedan od njih provjerava je li na zemlji ili nije. Ako je na zemlji odmah se pokreće. Drugi collider je nevidljiv i aktivira se samo kada kostur napravi animaciju napada. Taj collider na sebi ima DeathZone skriptu kako bi ubio igrača ako stoji pre blizu.



Slika 16: Skeleton [25]

Kostur na sebi ima **EnemyMovment** skriptu. Koji je veoma sličan skripti od mumije. Varijable su.

```
public GameObject player;
public GameObject detectionHightLeft, detectionHightRight;
public LayerMask layersToHit;
public float movmentSpeed;
private Rigidbody2D rb;
private int direction = -1;
private bool onGround =false;
public float wait = 2f;
private Animator anim;
private float PlayerDistance;
public int destroy=0;
public GameObject dmgZone;
public bool patrol = false;
```

U startu se dodaju vrijednosti varijablama player ,anim i rb. Isto kao i kod EnemyMumija.

```
void Start()
{
    anim= GetComponent<Animator>();
    rb=GetComponent<Rigidbody2D>();
```

```
        player = GameObject.FindGameObjectWithTag("Player");
    }
```

Kostur može izdržati 2 udarca s bakljom.

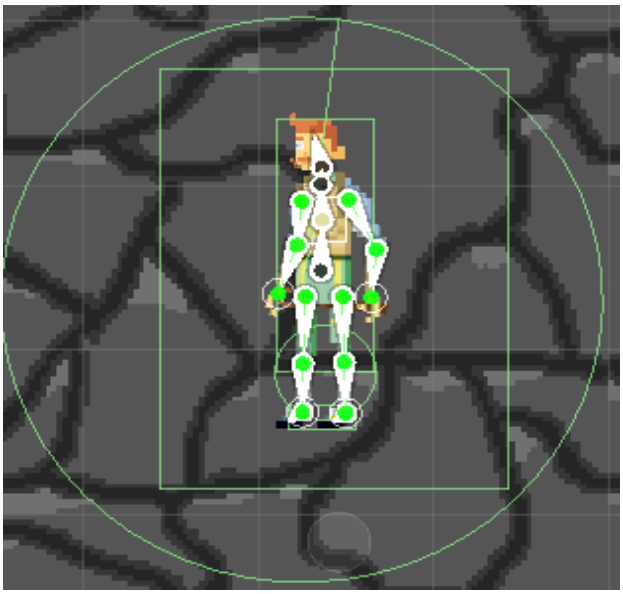
```
        if(destroy==2){
            Destroy(gameObject);
        }
```

Kod je skoro isti skripti koja se nalazi na mumiji. Jedina razlika su sljedeće dvije funkcije. Attack i AttackStop. Pozivaju se u animacijama. Prilikom animacije napada se aktivira okidač koji sadrži DeathZone skriptu na sebi kako bi kostur pokušao ubiti igrača na daljinu. Obadvije funkcije se nalaze na istoj animaciji, ali u različitim vremenima. Attack se nalazi prije jer aktivira collider, AttackStop se nalazi kasnije jer deaktivira collider.

```
public void Attack(){
        dmgZone.GetComponent<BoxCollider2D>().enabled=true;
    }
    public void AttackStop(){
        dmgZone.GetComponent<BoxCollider2D>().enabled=false;
    }
```

6.11 Igrač

Slika 17 prikazuje igrača (Player) koji je prefab i u sebi drži drugi prefab (baklju) isto kao i skeleton koristi IK solvere za noge i ruke. Igrač na sebi ima dva Collidera jedan kockasti collider i jedan kružni collider. Ispod kružnog Collidera je još jedan manji kockasti collider koji se nalazi na jednom objektu koji je dijete igrača. Pomoću tog Collidera detektira se je li igrač na podu ili ne i ta detekcija daje mogućnost igraču da skače. Kockasti Collider veći od igrača je zona okidanja koja služi za detekciju baklji koje uđu u nju, baklje koje se nalaze u zoni igrač može pokupiti s pritiskom određene tipke. Najveći kružni Collider vidljiv na slici nije od igrača, već je od baklje koju igrač drži.



Slika 17: Player

Jedan od skripti koji se nalaze na igraču je PlayerMovment. Koji omogućuje igraču da se pokreće u ovisnosti koja je tipka pritisnuta.

```
    public float movementSpeed ,climbSpeed;
    public float jumpForce;
    public float sprintMultyplyer;
    public bool sprint=false;
    public bool isGrounded = true;
    public bool jump = false;
    public static int torchCount;
    public int torchCount2;  //variabla u inspektoru da vidimo koliko je jos ostalo
    private Rigidbody2D rb;
    public static bool climb = false, climbing=false;
    public bool wPressed = false, sPressed = false, aPressed=false, dPressed
=false;
    public float currentGravityScale;
    public bool isSwinging;

    Vector3 horizontalMove ;
    float horizontalMoveRB;
```

```
public Animator anim;

//za bacanje baklje
public GameObject torchThrowPoint;

// Grappling hook
public Vector2 ropeHook;
public float swingForce = 4f;
private RopeSystem ropeSystem;
```

Movment speed je varijabla koja određuje kojom će se brzinom objekt igrač kretati lijevo i desno, climbSpeed predstavlja brzina penjanja po ClimbZone, jump force je snaga kojom će igrač biti lansiran prema gore. SprintMultiplayer se množi s movmentSpeedom kako bi dobili brzinu kretanja pod uvjetom da je tipka shift pritisnuta, sprint se aktivira kada igrač pritisne shift. IsGrounded prikazuje da li se igrač nalazi na tlu ili ne i s time limitira skakanje samo na jedan put. Jump prikazuje kada igrač skoči, torchCount je static varijabla koja može biti mijenjana iz bilo koje druge skripte u sceni i označava broj baklji koje igrač može koristiti. TorchCount2 prikazuje broj baklji u inspektoru, rb je rigidbody2D igrača, climb provjerava je li igrač u zoni za penjanje a climbing će biti istinit ako se igrač penje po toj zoni za penjanje. Varijable wPressed, sPressed, aPressed i dPressed točni su samo tijekom penjanja kada igrač pritisne određenu tipku. CurrentGravityScale se isto koristi prilikom penjanja gdje se gravitacija postavlja na 0 kako igrač ne bi padao, horizontalMove prikazuje x os -1 ili 1 tj. kada se pritisne tipka a ili tipka d, horizontalMoveRB koristi se kod penjanja za pokretanje tijela objekta, anim je animator od igrača, torchThrowPoint je prazan objekt u kojemu se instanciraju nove baklje, ropeHook lokacija gdje je hook pogodio, swingForce je snaga kojom igrač ide prema sredini na užetu i ropeSystem je referenca na skriptu RopeSystem.
U Start funkciji postavljaju se neke od varijabli, sve ostale varijable se postavljaju u inspektoru.

```
void Start()
```

```
{
    torchCount = torchCount2;
    rb = GetComponent<Rigidbody2D>();
    currentGravityScale = rb.gravityScale;
    Debug.Log(currentGravityScale);
    anim = GetComponent<Animator>();
    ropeSystem = this.GetComponent<RopeSystem>();
}
```

Kada god se izvede Update funkcija torchCount2 se postavlja na torchCount kako bi se u inspektoru prikazao trenutni broj baklji. Koristi se torchCount2 zbog toga što je torchCount static varijable i kao takva se ne prikazuje u inspektoru. Ako je if točan svi unosi igrača će biti blokirani, označuje da je izbornik otvoren.

```
torchCount2 = torchCount;
    if (ActivateCanvas.esc == false)
    {
```

U puno dijelova skripte se nalaze anim.SetBool i anim.SetFloat funkcije koje stavljaju variable u animatoru na određene vrijednosti kako bi se pokrenule ili zaustavile određene animacije. Kada igrač pritisne tipku a ili tipku d mijenja se x os u horizontalMove. Prema tome se zna da li se igrač kreće lijevo x=1 ili desno x=-1, horizontalMoveRB uzima taj 1 ili -1 za pokretanje igrača u tom smjeru.

```
        horizontalMove = new Vector3(Input.GetAxis("Horizontal"), 0f, 0f);
        horizontalMoveRB = Input.GetAxis("Horizontal");

        anim.SetFloat("Speed", Mathf.Abs(horizontalMoveRB));
        if (climb == false)
        {
            anim.SetFloat("vSpeed", rb.velocity.y);
        }
```

```
        anim.SetBool("Grounded", isGrounded);
```

S obzirom je li shift pritisnut ili ne mijenjaju se animacije i sprint varijabla koja u FixedUpdate određuje kojom brzinom će se igrač kretati

```
    //Update
        if (Input.GetKeyDown(KeyCode.LeftShift)){
            anim.SetBool("Run", true);
            sprint = true;
        }
        else if (Input.GetKeyUp(KeyCode.LeftShift))
        {
            anim.SetBool("Run", false);
            sprint = false;
        }

       //FixedUpdate
    if (sprint)
    {
        rb.velocity = new Vector2(horizontalMoveRB * (movementSpeed*
                                                    sprintMultyplyer), rb.velocity.y);
    }
    else
    {
        rb.velocity = new Vector2(horizontalMoveRB * movementSpeed,
                                                        rb.velocity.y);
    }
```

Sama varijabla jump ne omogućuje igraču da skoči, već i varijabla isGrounded mora biti točna kako bi igrač mogao skočiti.

```
    //Update
        if (Input.GetKeyDown(KeyCode.Space))
        {
            jump = true;
        }
```

```
        else if (Input.GetKeyUp(KeyCode.Space))
        {
            jump = false;
        }

    //FixedUpdate
     if(jump && isGrounded)
     {
        rb.velocity = Vector2.up * jumpForce;
        jump = false;
     }
```

Kada igrač pritisne lijevu tipku miša iz objekta torchThrowPoint se izvršava funkcija ThrowTorch koja izrađuje klon baklje i baca ga u smjeru miša određenom snagom.

```
    //Update
        if (Input.GetKeyDown(KeyCode.Mouse0))
        {
            if (torchCount > 0)
            {
                torchThrowPoint.GetComponent<TorchThrow>().ThrowTorch();
                torchCount--;
            }
        }
```

Pritiskom desne tipke miša izbacuje se uže koje sadrži baklju. Igrač može koristit to uže kako bi vidio što se nalazi u nekoj rupi.

```
    //Update
        if (Input.GetKeyDown(KeyCode.Mouse1))
        {
            torchThrowPoint.GetComponent<TorchThrow>().ThrowRope();
        }
```

Kada igrač pritisne Q zove se funkcija TorchPickUp s jednog objekta koji je dijete igrača i sadrži skriptu ItemPickup. TorchPickUp povećava broj baklji koje player može baciti.

```
//Update
    if (Input.GetKeyDown(KeyCode.Q))
    {
        GetComponentInChildren<ItemPickup>().TorchPickUp();
    }
```

Dolje prikazan kod pokazuje provjere za stisnute tipke prilikom penjanja igrača.

```
/Update
    if (climb)
    {
        if (Input.GetKeyDown(KeyCode.W))
        {
            wPressed = true;
        }
        else if (Input.GetKeyUp(KeyCode.W))
        {
            wPressed = false;
        }
        if (Input.GetKeyDown(KeyCode.S))
        {
            sPressed = true;
        }
        else if (Input.GetKeyUp(KeyCode.S))
        {
            sPressed = false;
        }
            if (Input.GetKeyDown(KeyCode.A))
```

```
        {
            aPressed = true;
        }
        else if (Input.GetKeyUp(KeyCode.A))
        {
            aPressed = false;
        }
        if (Input.GetKeyDown(KeyCode.D))
        {
            dPressed = true;
        }
        else if (Input.GetKeyUp(KeyCode.D))
        {
            dPressed = false;
        }
    }
```

U FixedUpdate dijelu koda za penjanje igrač se pokreće samo dok se nalazi u zoni za penjanje. Ako igrač izađe iz zone sve varijable se postavljaju na netočno i igrač se dalje kreće normalno

```
        //FixedUpdate
      if (climb)
    {
       if(aPressed && wPressed)
       {
           rb.gravityScale = 0f;
           climbing = true;
           rb.velocity = (Vector2.left + Vector2.up) * climbSpeed;
       }else if(aPressed && sPressed)
       {
           rb.gravityScale = 0f;
           climbing = true;
```

```
        rb.velocity = (Vector2.left + Vector2.down) * climbSpeed;
    }
    else if(dPressed && wPressed)
    {
        rb.gravityScale = 0f;
        climbing = true;
        rb.velocity = (Vector2.right + Vector2.up) * climbSpeed;
    }else if(dPressed && sPressed)
    {
        rb.gravityScale = 0f;
        climbing = true;
        rb.velocity = (Vector2.right + Vector2.down) * climbSpeed;
    }
    else if (wPressed)
    {
        rb.gravityScale = 0f;
        climbing = true;
        rb.velocity = Vector2.up * climbSpeed;
    }
    else if (sPressed)
    {
        climbing = true;
        rb.gravityScale = 0f;
        rb.velocity = Vector2.down * climbSpeed;
    }
    else if (aPressed )
    {
        rb.velocity = Vector2.left * climbSpeed;
    }
    else if (dPressed )
    {
        rb.velocity = Vector2.right * climbSpeed;
```

```
        }
    }
    else if (!climb)
    {
        rb.gravityScale = currentGravityScale;
        climbing = false;
        dPressed = false;
        aPressed = false;
        wPressed = false;
        sPressed = false;
    }
    }
```

Grappling hook funkcionira tako da igrač ispali kuku koja se može zakačiti za određene plohe u igri. Nakon što se kuka zakači igrač se uzdigne u zrak i može se ljuljati lijevo desno na užetu. Uže se ispali tamo gdje je igrač držao miš. **LineRenderer** komponenta se koristi da bi se iscrtalo uže na pravom mjestu a **DistanceJoint2D** je taj koji pretvara igrača u njihalo. Za provjeru je li se išta nalazi na putanji od igrača prema mišu se koristi **Raycast**. Ako Raycast pogodi ispravan objekt stvorit će se uže. To je princip rada RopeSystem skripte koja je napravljena prema [10]. Joint2D povezuje dva objekta koji sadrže rigidbody2D. Primjenjuju sile kako bi održali objekte koje povezuju na određenoj udaljenosti [13].

```
    if(horizontalMove.x < 0)
    {
        if (isSwinging && this.transform.position.y <= ropeSystem.
                                                    hook_coordinates.y)
        {
            var playerToHookDirection = (ropeHook - (Vector2)transform.
                                                        position.normalized;
            Vector2 perpendicularDirection;
            perpendicularDirection = new Vector2(-playerToHookDirection.y,
```

```
                                                    playerToHookDirection.x);
            var leftPerpPos = (Vector2)transform.position - p
                                                    erpendicularDirection * -2f;
            var force = perpendicularDirection * swingForce;
            rb.AddForce(force, ForceMode2D.Force);
        }
        transform.localScale = new Vector3(1, 1, 1);
    }else if(horizontalMove.x > 0)
    {
        if (isSwinging && this.transform.position.y <= ropeSystem.
                                                    hook_coordinates.y)
        {
            var playerToHookDirection = (ropeHook - (Vector2)transform.
                                                    position).normalized;
            Vector2 perpendicularDirection;
            perpendicularDirection = new Vector2(playerToHookDirection.y, -
                                                    playerToHookDirection.x);
            var rightPerpPos = (Vector2)transform.position +
                                                    perpendicularDirection * 2f;
            var force = perpendicularDirection * swingForce;
            rb.AddForce(force, ForceMode2D.Force);
        }
            transform.localScale = new Vector3(-1, 1, 1);
    }
```

GroundCheck skripta služi za provjeru da li se igrač nalazi za tlu mape. Koristi zonu okidanja koja se nalazi ispod nogu samog igrača. Varijabla playerMovment je referenca na skriptu PlayerMovment. OnTriggerEnter2D i OnTriggerStay2D funkcije provjeravaju da li je collider na kojemu igrač stoji tlo, ako je postavljaju varijablu isGrounded u skripti PlayerMovment na istinu. Što omogućuje skakanje igrača. Na OnTriggerExit2D se ta varijabla postavlja na laž jer igrač više nije na tlu.

```
    private PlayerMovement playerMovement;
    void Start() {
        playerMovement = this.GetComponentInParent<PlayerMovement>();
    }
    private void OnTriggerEnter2D(Collider2D ground)
    {
        if(ground.tag == "Ground")
        {
            playerMovement.isGrounded = true;
        }
    }
```

ItemPickUp omogućuje igraču da pokupi baklje koje su mu u blizini. Sadrži listu svih baklji koje se nalaze u zoni okidača. Kada se pozove TorchPickUp funkcija u PlayerMovment skripti prvo se provjerava postoji li koja baklja u okruženju igrača. Ako postoji baklja ona se uništava i broj baklji koje igrač može imati se povećava za jedan. Ako se u zoni nalazi više baklji uništava se prva baklja u listi i baklje koje igrač može koristiti se povećava za jedan.

```
    public List<GameObject> torches;
    public GameObject torchToDestroy;
    private int listCounter=0;
    private string i;
    public string name;

    public void TorchPickUp()
    {
        if (torches.Count>0)
        {
            PlayerMovement.torchCount++;
            torchToDestroy = torches[0];
            Destroy(torchToDestroy);
        }
    }
```

OnTriggerEnter2D provjerava nalazi se baklja u zoni. Ako se nalazi brojač se povećava za jedan i listi baklji se dodaje novi objekt baklja koji je ušao u zonu.

```
private void OnTriggerEnter2D(Collider2D collision){
    if (collision.CompareTag("Torch"))
    {
        listCounter++;
        torches.Add(collision.gameObject);
    }
}
```

OnTriggerExit2D provjerava ako baklja izađe iz zone. Zatim iz liste baklji izbacuje taj element.

```
private void OnTriggerExit2D(Collider2D collision)
{
    if (collision.CompareTag("Torch"))
    {
        torches.Remove(collision.gameObject);
    }
}
```

Skripta **TorchThrow** se nalazi na jednom djetetu igrača i cilj joj je kreirati kopije baklji i kreirati kopiju užeta. Varijable torch i rope su prefabi koji će se klonirati pomoću torchClone i ropeClone kasnije, throwForceTorch i throwForceRope predstavljaju snaga kojima se bacaju užad i baklje

```
private GameObject player;
//bacanje baklje
public GameObject torch;
public GameObject torchClone;
public float throwForceTorch,throwForceRope;
//bacanje užeta
public GameObject rope;
```

```
    public GameObject ropeClone;
```

TorchThrow sadrži samo dvije funkcije ThrowTorch i ThrowRope koje se pozivaju u PlayerMovment skripti. ThrowTorch uzima poziciju miša u svijetu u odnosno na kameru, directionToMouse vraća vektor 2 koji pokazuje razliku miša i throwPoint pozicije. Transform.up postavlja y os objekta na kojem je TorchThrow skripta. Klonira se torch prefab i tijelu od baklje dodajemo snagu u smjeru y od trenutnog objekta. Roditelj klona se postavlja da ne postoji.

```
    public void ThrowTorch()
    {
        Vector3 mousePosition = Input.mousePosition;
        mousePosition = Camera.main.ScreenToWorldPoint(mousePosition);
        Vector2 directionToMouse = new Vector2(mousePosition.x - transform.
                                    position.x, mousePosition.y - transform.position.y);
        transform.up = directionToMouse;
        torchClone = Instantiate(torch, transform.position ,Quaternion.
                                                        Euler(0,0,0));
        torchClone.GetComponent<Rigidbody2D>().AddForce(transform.up *
                                                        throwForceTorch);
        torchClone.transform.parent = null;
    }
```

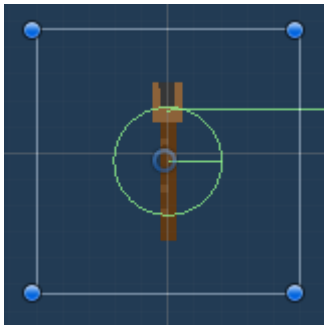
ThrowRope funkcija je identična ThrowTorch funkciji jednina razlika je što se klonira rope prefab.

```
public void ThrowRope()
    {
        Vector3 mousePosition = Input.mousePosition;
        mousePosition = Camera.main.ScreenToWorldPoint(mousePosition);
        Vector2 directionToMouse = new Vector2(mousePosition.x -
                transform.position.x, mousePosition.y - transform.position.y);
        transform.up = directionToMouse;
        ropeClone = Instantiate(rope, transform.position,
                                                    Quaternion.Euler(0, 0, 0));
```

```
        ropeClone.GetComponent<Rigidbody2D>().
                                        AddForce(transform.up * throwForceRope);
        ropeClone.transform.parent = null;
}
```

6.12 Početak užeta

Slika 18 prikazuje početak užeta (eng. Rope start). Na slici je prikazana baklja jer početak užeta u sebi sadrži objekt baklja. Uže se koristi za provjeravanje mape gdje igrač mora preskakati, kako bi igrač mogao vidjeti što se nalazi ispod njega. Sastoji se od jednog kružnog Collidera koji služi za interakciju s ostatkom mape.



Slika 18: Rope start

Početak užeta sadrži skriptu **Rope2 .** Varijable prikazane biti će objašnjene kroz kod.

```
    public GameObject player;
    public GameObject rope;
    public GameObject cloneRope;
    public GameObject oldRope;
    public int curentCount;
    public float distance = 2f;
    public int minimumNodes;
    public bool  minimumDone=false;
    public bool triggerd = false;
    public List<GameObject> listOfNodes = new List<GameObject>();
    public LineRenderer lineRender;
```

U Start funkciji postavljaju se varijable player, oldRope, listi listOfNodes se doda trenutni objekt jer je početak užeta i lineRenderer koji se nalazi za prefabu.

```
void Start()
{
    player = GameObject.FindGameObjectWithTag("ThrowPoint");
    oldRope = this.gameObject;
    listOfNodes.Add(this.gameObject);
    lineRender = this.gameObject.GetComponent<LineRenderer>();
}
```

Ako je minimumDone netočan onda će se instancirati nodovi sve dok se ne izjednači s minimumom i zatim se minimumDone postavlja true.

```
private void Update()
{
        if (!minimumDone)
        {
            if (curentCount < minimumNodes)
            {
                CreateNode();
            }
            else
            {
                minimumDone = true;
            }
```

Kada se u listi nalaze manje od 4 elementa uništavaju se svi elementi liste.

```
        }else if (minimumDone && curentCount<4){
            foreach (GameObject littleMan in listOfNodes){
                Destroy(littleMan);
            }
        }
```

Igrač može koristiti scroll kako bi produžio ili smanjio većinu užeta. LineRender() se poziva stalno kako bi stalno imali liniju od užeta

```
        if(Input.mouseScrollDelta.y > 0)
        {
            CreateNode();
        }
        if(Input.mouseScrollDelta.y < 0)
        {
            DeleteNode();
        }
        LineRender();
    }
```

CreateNode funkcija se poziva u Update funkciji klonira rope u ovom slučaju rope je prefab koji sadrži komponente rigidbody2d, DistanceJoint2D, SpringJoint2D i CircleColider2D. DistanceJoint2D od klona se povezuje prijašnjom kopiranom objektu u užetu, i postavlja se dužinu koju on pokušava održati. Aktivira se element SpringJoint2D i spoji se na rigidbody2D od igrača zbog toga što je trenutna kopija zadnji dio užeta, dodaje mu se udaljenost koju on pokušava održati. Ako prijašnja kopija užeta sadrži SpringJoint2D rigidbody2D za nju postavlja se na trenutno instanciranu komponentu. Trenutni klon postavlja za stari staro uže i dodaje se jedan currentCount. CurentCount služi kao ograničenje kada dođemo do maksimuma kopija.

```
    private void CreateNode()
    {
        cloneRope = Instantiate(rope, player.transform);
        cloneRope.transform.SetParent(null);
        cloneRope.GetComponent<DistanceJoint2D>().connectedBody =
                                    oldRope.GetComponent<Rigidbody2D>();
        cloneRope.GetComponent<DistanceJoint2D>().distance = distance;
        listOfNodes.Add(cloneRope);
```

```
        cloneRope.GetComponent<SpringJoint2D>().enabled = true;
        cloneRope.GetComponent<SpringJoint2D>().connectedBody =
                                    player.GetComponent<Rigidbody2D>();
        cloneRope.GetComponent<SpringJoint2D>().distance = distance;
        if (oldRope.GetComponent<SpringJoint2D>() != null)
            oldRope.GetComponent<SpringJoint2D>().connectedBody=
                                    cloneRope.GetComponent<Rigidbody2D>();
        }
        oldRope = cloneRope;
        curentCount++;
    }
```

DeleteNode je funkcija koja se poziva kada player koristi scroll. Objekt na kojemu se nalazi baklja teleportira se na mjesto sljedećeg objekta i zatim se treći objekt u listi spoji s prvim i izbriše se drugi objekt tj. Objekt koji se nalazi na indexu 1 i smanji se currentCount za jedan. I s time dobijemo smanjivanje užeta.

```
    private void DeleteNode(){
        gameObject.transform.position = listOfNodes[1].transform.position;
        listOfNodes[2].GetComponent<DistanceJoint2D>().connectedBody =
                                    gameObject. GetComponent<Rigidbody2D>();
        GameObject deleteNode = listOfNodes[1];
        listOfNodes.RemoveAt(1);
        Destroy(deleteNode);
        curentCount--;
    }
```

LineRenderer funkcija crta liniju u ovisnosti o svim pozicijama nodova u listi.

```
    private void LineRender() {
        lineRender.positionCount =curentCount;
        int i;
        for (i=0; i< curentCount; i++){
            lineRender.SetPosition(i, listOfNodes[i].transform.position);
          }
```

7. Zaključak

Unity je jedan od najraširenijih alata za izradu video igrica. Njegovo korištenje vrlo je jednostavno, intuitivno i ispunjava sve osnovne funkcije koje su potrebne za razvoj igara, ali omogućuje i napredno programiranje i razvoj. Unity Engine ima ogromne količine materijala i dokumentacije iz kojih se može učiti te postoje mnogobrojni online tutorijali i za to je odličan alat za početnike.

Photoshop je jedan od alata koji se mogu koristiti za izradu slika potrebnih za izradu ove igre. Problem s photoshopom je što se mora plaćati na mjesečnoj razini. Ali to nije problem jer postoje i druge besplatne varijante editora slika koje omogućuju slabiju ili istu kvalitetu izrade kao npr. Gimp, Krita i Pixlr editor. Oni su programi koji se moraju skinuti i instalirati kako bi se koristili. Postoje i alternative na webu kao što je Photopea koji je djelomično korišten u izradi ovog rada zbog problema s hardverom.

Bone animacije koje su korištene za izradu koriste Inverse Kinematic 2D sistem koji omogućuje da se udovi kreću kao i u stvarnosti. Problem bone animacije nasprema animiranja slike po slike je to što na dijelovima gdje su kosti se dobije produživanje slika što izgleda čudno. Dok je prednost to što se ne mora toliko posla uložiti u crtanje slike po slike.

Kroz izradu igre istraženi su procesi izrade igre kao što su animiranje, programiranje i kreiranje slika potrebnih za izradu same igre. Prilikom programiranja bilo je potrebno primijeniti znanja iz objektno orijentiranog programiranja, ali i znanja iz fizike i matematike. Kod animacije je korišten koštani sustav koji olakšava animiranje objekata u ovom slučaju slika, uz pomoć 2D Inverse Kinematics sistema koji omogućuje realistično kretanje udova objekata. Kod kreiranja slika bilo je potrebno biti precizan i odrediti koja će biti razlučivost komponenta, kako bi izgledalo realistično prema ostalima grafikama igre.

Premda je ova igra predana kao završni rad planiram ju i dalje nadograditi kako bi još više proširio svoje znanje o izradi igara.

8. Literatura

1. https://unity3d.com/what-is-a-game-engine
2. https://www.macworld.com/article/1060484/unity.html
3. https://arstechnica.com/information-technology/2010/09/unity-3-brings-very-expensive-dev-tools-at-a-very-low-price/?comments=1
4.https://www.polygon.com/2012/11/14/3645122/unity-4-0-available-download
5. https://techcrunch.com/2014/03/18/unity-5-announced-with-early-support-for-plugin-free-browser-games/?guccounter=1&guce_referrer_us=aHR0cHM6Ly9lbi53aWtpcGVkaWEub3JnLw&guce_referrer_cs=GUgaM71y4bNqC5ZWuCtryQ
6.https://www.gamasutra.com/view/news/307050/Unity_20172_brings_Autodesk_integration_into_the_fold.php
7. https://www.gamesindustry.biz/articles/2018-03-20-unity-2018-detailed-in-gdc-keynote
8. https://blogs.unity3d.com/2019/07/30/heres-whats-in-the-brand-new-unity-2019-2/
9. https://github.com/Unity-Technologies/2d-extras
10. https://www.raywenderlich.com/348-make-a-2d-grappling-hook-game-in-unity-part-1
11. https://docs.unity3d.com/Manual/CollidersOverview.html
12. https://docs.unity3d.com/Manual/class-Rigidbody2D.html
13. https://docs.unity3d.com/Manual/Joints2D.html
14.https://docs.unity3d.com/Manual/Prefabs.html
15. https://docs.unity3d.com/Manual/Layers.html
16. https://docs.unity3d.com/ScriptReference/MonoBehaviour.Start.html
17. https://docs.unity3d.com/ScriptReference/MonoBehaviour.Update.html
18.https://docs.unity3d.com/ScriptReference/MonoBehaviour.OnTriggerStay2D.html

19.https://docs.unity3d.com/ScriptReference/MonoBehaviour.OnTriggerExit2D.html
20.https://docs.unity3d.com/ScriptReference/MonoBehaviour.OnTriggerEnter2D.html
21.https://docs.unity3d.com/ScriptReference/GameObject.html
22. https://docs.unity3d.com/Manual/class-Camera.html
23. https://www.theverge.com/2015/3/3/8142099/unity-5-engine-release
24. https://docs.unity3d.com/Manual/AnimationOverview.html
25.https://www.deviantart.com/shkvapper/art/sprite-skeleton-2D-game-character-607247016
26.https://docs.unity3d.com/Manual/class-LineRenderer.html
27.https://docs.unity3d.com/ScriptReference/MonoBehaviour.FixedUpdate.html
28. https://docs.unity3d.com/Manual/Coroutines.html
29.https://blogs.unity3d.com/2018/12/04/getting-started-with-2d-inverse-kinematics/

Slike