Izrada platformske računalne 2D igre u Unity okruženju

Gazić, Dino

Undergraduate thesis / Završni rad

2020

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **University of Pula / Sveučilište Jurja Dobrile u Puli**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:137:004785

Rights / Prava: In copyright / Zaštićeno autorskim pravom.

Download date / Datum preuzimanja: **2024-11-29**

Repository / Repozitorij:

Digital Repository Juraj Dobrila University of Pula

Sveučilište Jurja Dobrile u Puli
Fakultet Informatike u Puli

Dino Gazić

IZRADA PLATFORMSKE RAČUNALNE 2D IGRE U UNITY OKRUŽENJU

Završni rad

Pula, rujan, 2020.

Sveučilište Jurja Dobrile u Puli
Fakultet Informatike u Puli

Dino Gazić

IZRADA PLATFORMSKE RAČUNALNE 2D IGRE U UNITY OKRUŽENJU

Završni rad

JMBAG: 0303076099 6 , redoviti student

Studijski smjer: Informatika

Predmet: Napredne tehnike programiranja
Znanstveno područje: Društvene znanosti

Znanstveno polje: Informacijske i komunikacijske znanosti

Znanstvena grana: Informacijski sustavi i informatologija

Mentor: izv.prof.doc.dr.sc. Tihomir Orehovački

Pula, rujan, 2020.

IZJAVA O AKADEMSKOJ ČESTITOSTI

Ja, dolje potpisani Dino Gazić, 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

Gazić Dino

U Puli, rujan 2020. godine

IZJAVA
o korištenju autorskog djela

Ja, Dino Gazić dajem odobrenje Sveučilištu Jurja Dobrile u Puli, kao nositelju prava iskorištavanja, da moj završni rad pod nazivom 'Izrada platformske računalne 2D igre 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, 21. rujna. 2020.

Potpis
Gazić Dino

SAŽETAK

Računalna 2D platformska igra koja je ovdje prikazana napravljena je u Unity razvojnom okruženju uz pomoć C# objektno-orijentiranog programskog jezika.
Korištena verzija softvera je Unity 2019.4. U završnom radu je prikazan proces korištenja animacija, te animatora koji je ključan za omogućavanje interakcija vidljivim. Prikazan je i način izrade 2D razina putem paketa „Tile Palette“. Mehanike na igraču koje su prikazane jesu skakanje, kretnja igrača lijevo i desno i korištenje oružja pomoću kojeg lik može ispucavati projektile(objekte) u svim smjerovima s lijevom tipkom na mišu. Interakcije unutar igre koje su implementirane jesu različiti dijalozi, platforme koje propadaju ili se kreću i tako stvaraju određenu dozu napetosti, ali to sve ne bi bilo izvedivo bez mehanike koja se zove detekcija kolizije koja je objašnjena nastavku.

Ključne riječi: Unity, 2D računalna igra, C#, platformska avantura

ABSTRACT

The computer 2D platformer game shown here was created in the Unity development environment using a C#, object-oriented programming language. The software version used is Unity 2019.4. In final thesis is described the process of creating and using animations and animator which was the main key of creating interactions becoming visible. It is also shown a creating of 2D levels with package „Tile Palette“. The mechanics on the player shown here are jumping, moving the player left and right and using weapon that the character can shoot in the direction of 360° and with the left mouse button. Interactions in the game are different dialogues, parts of the platform that suddenly fall apart or move which creates a certain dose of tension ,but all of this would not be feasible without a mechanics called collision detection which is explained in extension.

Keywords: Unity, 2D computer game, C#, platformer adventure

Sadržaj

1. UVOD 1
2. UNITY 2
3. SLIČNE IGRE 3
3.1. Hollow Knight 3
3.2. Sličnost igre Hollow Knight sa igrom The story of a blue Heart 4
3.3. Celeste 5
3.4. Sličnost igre Celeste sa igrom The story of a blue Heart 5
4. PROCES RAZVOJA IGRE 7
4.1. Kamera 7
4.2. Likovi 8
4.2.1. Igrač 8
4.2.2. Kooperativni lik 13
4.2.3. Protivnici 15
4.3. Animator 32
4.4. Dizajn razina 34
4.5. Grafičko korisničko sučelje 37
4.5.1. Glavni izbornik (Main Menu) 37
4.5.2. Sučelje unutar razine (Player UI) 39
5. ZAKLJUČAK 42
LITERATURA 43
POPIS SLIKA 44

1. UVOD

Tema ovog završnog rada je bila razviti računalnu 2D platformska igru pomoću Unity razvojnog okruženja gdje se većinom koristi objektno-orijentirani programski jezik C#. Verzija programa koja se koristi za izradu ovog projekta jest Unity 2019.4.

Kroz ovaj rad prikazan je proces izrade igre „The story of a blue Heart“, igra koja se temelji na priči koju saznajemo kroz različite dijaloge sa pticom Po. Prikazani su procesi animiranja likova i dodavanje funkcionalnosti tim istim likovima putem kodiranja skripti u programskom jeziku C#. Igrač na početku svake razine dobije savjet od ptice Po kako prelaziti određenu razinu, te ga kroz cijeli njegov put povratka u džunglu čekaju neprijatelji koje mora uništiti na određene različite načine. U ovom 2D računalnom platformeru napravljeno je šest različitih neprijatelja koji su specifični svaki na svoj način s kojima se naš glavni lik „Blue Heart“ susreće, od šest vrsta neprijatelja tri imaju umjetnu inteligenciju. Uz raznovrsne neprijatelje dodane su još i pomične platforme, i platforme koje propadaju prilikom igračevog prelaska preko njih da igra bude još dinamičnija. Igrač na početku ima pet virtualnih života koji mu se smanjuju nakon svakog točnog napada neprijatelja, no s druge strane ako igrač propadne u područja kao što su lava i voda (ako više ne stoji na igračkoj platformi) automatski umire te se vraća na početak razine.

Igra sadržava i različite efekte tj. „partical effects“ koji daju određenu zanimljivost i predodžbu igraču što se zaista dogodilo u određenom trenutku. Uz efekte vrlo važna stvar jest zvuk i pozadinska glazba koji kao i efekti daju do znanje igraču što se zbiva u kojem trenutku. U radu se prikazuje proces izrade, odnosno kodiranja mehanika igre te implementacija istih. Igra se sastoji od 2 razine. Mehanike prikazane u radu su kretanje, detektiranje kolizije, udaljeni napadi, različite zamke unutar igre te dijalozi.

2. UNITY

Unity je razvojno okruženje za izradu video igara napravljen od tvrtke Unity Technologies [1]. Napravljen je kako bi se razvoj video igara proširio na veću populaciju ljudi i trenutno je uz Unreal najkorišteniji engine za izradu video igara.

Koristi se za izradu 2D i 3D igara koje mogu biti pokretane trenutno na 21 platformi u to spadaju računala, konzole i mobilni uređaji. Unity sučelje sastoji se od prozora koji služe za obavljanje određenih radnji i poslova u procesu izrade igre. Osnovni i prvi prozor kojeg vidimo nalazi se u centru, to je scena u koju se postavljaju objekti te se stvara igra. Scena zapravo prikazuje ono što prvo vidimo kada pokrenemo igru, igra se najčešće sastoji od više scena, recimo to ovako, svaka razina je zasebna scena za sebe. Sa lijeve strane vidimo poredane objekte koji su poredani u hijerarhiji, objekti su temeljni dio igre te se od njih sama igra i izgrađuje. Na sve objekte možemo postavljati različite zvukove [3], efekte te skripte koje govore objektu što da radi. Odabirom pojedinog objekta otvara nam se na lijevoj strani inspektor u kojem možemo modificirati objekt, postavljati mu određeni položaj, gravitaciju, dodavati različite animacije i slično. Na dnu imamo prozor projekt u kojem nam se nalaze sve umetnute datoteke (zvuk, animacije, određene skripte, sprite-ovi itd.). Dolazimo i do itekako važnog prozora koji se zove animator koji služi za upravljanje animacijama bez kojih igra ne bi imala smisla. Skripte u Unity-u se pišu u programskim jezicima C#, JavaScript i Boo. U ovom projektu je korišten C# objektno-orijentirani programski jezik. Skripte imaju dvije osnovne funkcije: Start() funkcija koja se izvodi prilikom pokretanja skripte te funkcija Update() koja se izvodi prilikom svakom frame-a.

3. SLIČNE IGRE

The story of a blue Heart je igra koji ima dosta sličnosti kao što su igre istog žanra, neke od njih su Hollow Knight i Celeste. U nastavku slijedi opis navedenih igara i pregleda igrivosti istih.

3.1. Hollow Knight

Hollow Knight je 2D akcijski platformer razvijen i objavljen od strane studija Team Cherry. Igra je izdana za Microsoft Windows, MacOS i Linux u 2017., a za Nintendo Switch, PlayStation 4, Xbox One 2018. godine [6].

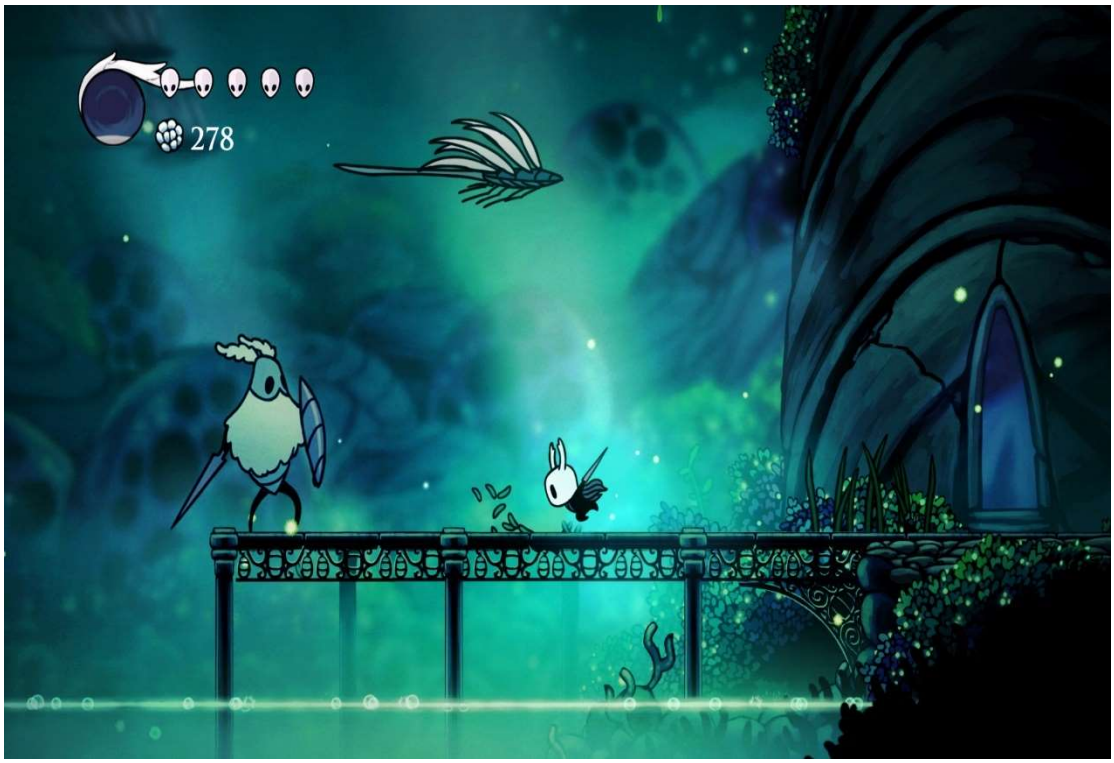
Igra priča priču o vitezu u potrazi za otkrivanjem tajni davno napuštenog kraljevstva kukaca Hallownest, čije zastrašujuće dubine privlače avanturiste i hrabre s obećanjima blaga i odgovorima na drevne tajne. U većini područja igre igrač susreće neprijateljske bube i druge vrste bića. Bliska borba uključuje korištenje mača da udari neprijatelje s kratke udaljenosti. Igrač također može naučiti magiju, dopuštajući napade na velike udaljenosti. Poraženi neprijatelji bacaju valutu pod nazivom Geo [6].

Tijekom igre, igrač susreće likove koji nisu igrači, odnosno NPC likove (eng. non-playable characters) s kojima može komunicirati. Ti likovi pružaju informacije o priči unutar igre, nude pomoć i prodaju predmete ili usluge. Igrač može nadograditi vitezov mač kako bi radio više štete ili pronašao spremnike duše za nošenje više duša. Tijekom igre igrač stječe predmete koji pružaju nove pokretne sposobnosti. Oni uključuju dodatni skok u zrak (predmet naziva Monarch Wings), prianjanje uz zidove i skakanje s njih (predmet naziva Mantis Claw), i brzo pomicanje [6].

3.2. Sličnost igre Hollow Knight sa igrom The story of a blue Heart

Igra Hollow Knight je sličan platformer kao i The story of a blue Heart. Sličnost dolazi do izražaja kod susreta sa NPC likovima. U igri Hollow Knight NPC likovi pružaju informacije o priči unutar igre, nude pomoć i prodaju predmete ili usluge, s druge strane u igri The story of a blue Heart imamo pticu Po koja nam pojašnjava detaljnije situaciju u kojoj se nalazimo u određenom trenutku te nudi različite savjete gdje bi smo morali krenuti.

Dakako da se može uz malo truda napraviti „inventory system“. Dakle sustav spremanja stvari te kupovanja različitih stvari od drugih NPC likova koji se mogu jednostavno dodati u nastavak igre te tako proširiti mogućnosti same igre. Izgled igre može se vidjeti na slici 1.



Slika 1. Snimak ekrana iz igre Hollow Knight

3.3. Celeste

Celeste je 2D stilizirana video igra od strane kanadskih programera Matta Thorsona i Noela Berryja, s umjetnošću brazilskog studija MiniBoss. Igra je originalno kreirana kao prototip u četiri dana tijekom Game Jam-a, te je kasnije proširena u potpuno izdanje. Celeste je izdana u Siječnju 2018. godine na platformama Microsoft Windows, Nintendo Switch, PlayStation 4, Xbox One, macOS i Linux [7].

Celeste je platformska igra u kojoj igrači kontroliraju djevojku po imenu Madeline kojoj je cilj proći kroz planinu dok izbjegava razne smrtonosne prepreke. Uz skakanje i penjanje po zidovima na limitirano vrijeme, Madeline ima sposobnost izvođenja lansiranja / polijetanja u zrak (eng. dash) u osam kardinalnih i interkardinalnih smjerova. Igrači također mogu pristupiti modu pomoći, gdje mogu promijeniti neke atribute o fizici igre. Neki od njih uključuju: beskonačna zračna lansiranja, nepobjedivost ili usporavanje brzine igre [7].

3.4. Sličnost igre Celeste sa igrom The story of a blue Heart

Slične mehanike u 2D platformer igri Celeste kao u igri The story of a blue Heart jesu smrtonosne zamke koje igrač treba sa opreznošću izbjegavati. Mehanika samog igrača u igri Celeste je malo više razvijenija jer lik uz kretnju ima i mogućost penjanja po zidovima. U igri Celeste još je implementiran način zračnog lansiranja u smjeru 360 stupnjeva, dok je u igri The story of a blue Heart ta mehanika napravljena za prikupljeno oružje koje ima mogućnost ispaljivanja projektila isto tako u smjeru 360 stupnjeva. Izgled igre može se vidjeti na slici 2.

Slika 2. Snimka ekrana iz igre Celeste

4. PROCES RAZVOJA IGRE

4.1. Kamera

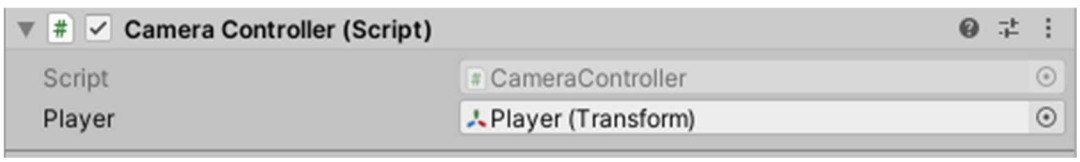
Započeti ćemo opis najvažnijeg dijela razvoja igre bez koje igra ne bi imala smisla. Kamera je objekt koji je automatski stvoren prilikom stvaranja nove scene. Kamera je zapravo pokazatelj stvorenog svijeta igraču koji igra igru. Kameri pomoću inspektora možemo namještati poziciji i veličinu, promjenom od ta dva atributa igraču smanjujemo ili povećavamo vidno polje.

Kamera u igri ima bitnu ulogu jer je zadužena za praćenje igrača u kojem god smjeru se on kretao, te mora samostalno zaustaviti kako ne bih izašla van granica napravljene mape, za tu funkcionalnost na objekt kamere je dodan atribut „CinemachineBrain“ koji ne dozvoljava kamera da izađe van granica izrađene mape. Na objekt kamere dodajemo skriptu „CameraController“ koja je vidljiva na slici 3. da gleda koordinate igrača te da ga prati. Ovdje vidimo samo Update() funkciju koja služi za praćenje igrača po koordinatnom sustavu putem x-osi, y-osi te z-osi.

```
Unity Script | 0 references
public class CameraController : MonoBehaviour
{
    public Transform player;

    Unity Message | 0 references
    private void Update()
    {
        transform.position = new Vector3(player.position.x, player.position.y, transform.position.z);
    }
}
```

Slika 3. Skripta CameraController

Zadnja stavka prilikom namještanja dobre kamere jest da u polje Player kojeg smo ranije napravili u skripti sa naredbom „public Transform player“ prebacimo objekt Player iz hijerarhije. Tada će skripta znati o kojem se objektu radi tj. koji objekt treba pratiti, slika 4.



Slika 4. Unity inspektor modifikacija kamere

4.2. Likovi

U ovom 2D platformeru postoji sedam vrsta likova, od tih sedam vrsta jedan je Blue Heart plavi majmun koji je ujedno i glavni lik u našoj igri kojega kontroliramo. Naš glavni lik Blue Heart ima svoju zasebnu skriptu „PlayerControllers“, dok ostalih šest vrsta likova koji su neprijatelji imaju zajedničku skriptu pod nazivom „Enemy“.

4.2.1. Igrač

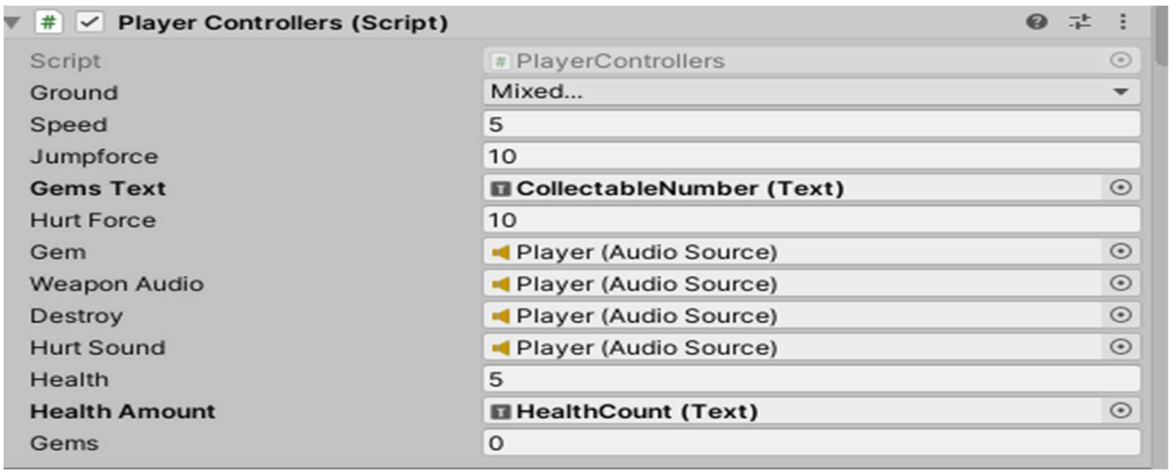
Player je lik kojim upravlja igrač a on je zapravo plavi majmun, slika 5. Naš igrač ima mogućnost kretanja lijevo i desno, može skakati te mu se otvara dodatna mogućnost nakon što pokupi oružje da mu se aktivira sposobnost udaljenog napada šurikenom koji ima mogućnost ispaljivanja u smjeru 360 °, sama akcija ispaljivanja je omogućena prilikom pritiska lijeve tipke na mišu.

Slika 5. Sprite igrača (BlueHeart)

Na početku skripte prikazane su varijable koje služe igraču da bi imao svoju stabilnost odnosno gravitaciju, stvaranje animatora kojeg ćemo detaljnije poslije objasniti no služi za izmjenu stanja animacija npr. (napad, trčanje, početna pozicija), slika 6. Nadalje, definiran je i Collider2D koji služi za detekciju samog igrača što se može odnositi na različite platforme na kojima stoji ili recimo primanje štete koje mu neprijatelj zada kada ga dodirne ili ispali nešto na njega, s druge strane u mom primjeru isto služi za prikupljanje različitih dijamanata iliti oružja. Dalje u primjeru imamo inspektor varijable vidljive na slici 7, to su varijable koje kad ih definiramo u skripti jednostavno postanu dio skripte koja se nalazi u Unity programu.

```
public class PlayerControllers : MonoBehaviour
{

    //Start variables
    private Rigidbody2D rb;
    private Animator anim;
    private enum State { idle, running, jumping, falling, hurt, attacking}
    private State state = State.idle;
    private Collider2D coll;




    //Inspector variables
    [SerializeField] private LayerMask Ground;
    [SerializeField] private float speed = 1f;
    [SerializeField] private float jumpforce = 10f;
    [SerializeField] private Text gemsText;
    [SerializeField] private float HurtForce = 10f;
    [SerializeField] private AudioSource gem;
    [SerializeField] private AudioSource weaponAudio;
    [SerializeField] private AudioSource destroy;
    [SerializeField] private AudioSource hurtSound;
    [SerializeField] private int health;
    [SerializeField] private Text healthAmount;

    public int gems = 0;
```

Slika 6. Inicijalne ulazne varijable u PlayerControllers skripti



Slika 7. Inspektor varijabli u PlayerControllers skripti

U ovoj funkciji koja se nalazi na slici 8 kada igrač dođe u područje na određenom djelu razine u kojem je objekt koji ima tag „Collectable“ automatski se funkcija izvršava te se odrađuju radnje prilikom sakupljanja dijamanata a to su pokreni zvuk [3], uništi pokupljeni dijamant, pribroji taj pokupljeni na zbroj. Drugi primjer je bio da tag odnosno oznaka mora biti „PickUp“ odnosno postavili smo objekt oružja na određenom djelu razine koji je ima tag „PickUp“ te kad igrač dođe blizu oružja automatski se okine zvuk [3] koji daje igraču do znanja da je pokupio nekakvu novu stvar.

```
private void OnTriggerEnter2D(Collider2D collision)
{
    if (collision.tag == "Collectable" )
    {

        gem.Play();
        Destroy(collision.gameObject);
        gems += 1;
        gemsText.text = gems.ToString();
        Debug.Log(gems);
    }
    if(collision.tag == "PickUp")
    {
        weaponAudio.Play();
    }
}
```

Slika 8. Funkcija za prikupljanje u PlayerControllers skripti

U funkciji koju vidimo na slici 9 imamo dva moguća slučaja ako igrač bude u bliskom susretu sa „colliderom“ na objektu sa tagom „Enemy“. Prvi slučaj je da igrač bude u state-u „falling“ tj. da skoči na neprijatelja, onda neprijatelj biva uništen, drugi slučaj je ako ga dotaknemo u bilo kojem drugom položaju osim skoka, aktivirat će se drugi dio funkcije koji će nam dati zvučni signal da smo ozlijeđeni, pozvat će se „state.hurt“ koji nas obavještava za ozljedu te se poziva funkcija HolderHealth() koju možemo vidjeti na slici 10 koja nam smanjuje život za jedan od mogućih pet. Ako nam je zdravlje jednako nuli, automatski nas vrati na početak razine. I ako se aktivira taj drugi dio funkcije aktivira se i odbacivanje igrača na suprotnu stranu gdje se neprijatelj nalazi (efekt udarca).

```
private void OnCollisionEnter2D(Collision2D other)
{
    if(other.gameObject.tag == "Enemy")
    {
        Enemy enemy = other.gameObject.GetComponent<Enemy>();
        if(state == State.falling)
        {
            destroy.Play();
            enemy.JumpedOn();
            Jump();
        }
        else
        {
            hurtSound.Play();
            state = State.hurt;
            HolderHealth(); //Deal with health, updating ui, and will reset level when hp is 0
            hurtSound.Play();
            if (other.gameObject.transform.position.x > transform.position.x)
            {
                //enemy is to my right therefore i should be damaged and go left

                rb.velocity = new Vector2(-HurtForce, rb.velocity.y);
            }
            else
            {
                //Enemy is to my left therefore i should be damaged and move right
                rb.velocity = new Vector2(HurtForce, rb.velocity.y);

            }
        }
    }
}
```

Slika 9. Funkcija za detekciju prilikom susreta sa neprijateljem

```
3 references
public void HolderHealth()
{
    hurtSound.Play();
    health -= 1;
    healthAmount.text = health.ToString();
    if (health <= 0)
    {
        SceneManager.LoadScene(SceneManager.GetActiveScene().name);
    }
}
```

Slika 10. Prikaz funkcije HolderHealth()

U funkciji Movement() koju vidimo na slici 11 inicijalizirali smo varijablu „hdirection“ odnosno horizontalni smjer, i ako je horizontalni smjer manji od nule krećemo se u lijevo i automatski nam se „spritesheet“ rotira u smjeru gdje smo se okrenuli, ako je horizontalni smjer veći od nule, tada idemo desno i „spritesheet“ se okreće u smjeru u kojem smo krenuli a to je desna strana. Imamo za kretanje lika još jednu mogućnost, ako pritisnemo tipku space odnosno razmak i ako lik dodiruje „LayerMask“ ground tada dozvoli igraču skok.

```
1 reference
private void Movement()
{
    float hdirection = Input.GetAxis("Horizontal");


    //moving left
    if (hdirection < 0)
    {
        rb.velocity = new Vector2(-speed, rb.velocity.y);
        transform.localScale = new Vector2(-1, 1);

    }
    //moving right
    else if (hdirection > 0)
    {
        rb.velocity = new Vector2(speed, rb.velocity.y);
        transform.localScale = new Vector2(1, 1);

    }
    //jumping
    if (Input.GetButtonDown("Jump") && coll.IsTouchingLayers(Ground))
    {
        Jump();
    }
}
```

Slika 11. Funkcija Movement() kretnje igrača

Ovdje ispod imamo posljednju funkciju u skripti „PlayerControllers“ zove se AnimationState() te je vidljiva na slici 12. Ona provjerava u kojoj se trenutno animaciji nalazimo, i ako ćemo brzo izaći iz nje da nas prebaci u drugu npr. (faza kada skočimo nakon kratkog vremena nas prebacuje u animaciju padanja odnosno state.falling).

```
private void AnimationState()
{
    if (state == State.jumping)
    {
        if (rb.velocity.y < .1f)
        {
            state = State.falling;
        }

    }
    else if (state == State.falling)
    {
        if (coll.IsTouchingLayers(Ground))
        {
            state = State.idle;
        }
    }


    else if (Mathf.Abs(rb.velocity.x) > .5f)
    {
        //moving

        state = State.running;

    }
    else if (state == State.hurt)
    {
        if (Mathf.Abs(rb.velocity.x) < .1f)
        {
            state = State.idle;
        }
    }
    else
    {
        state = State.idle;
    }

}
```

Slika 12. Funkcija AnimationState() u skripti PlayerControllers

4.2.2. Kooperativni lik

U ovoj igri kooperativi lik je ptica Po, sprite lika vidljiv na slici 13. Ptica prati našeg glavnog lika kroz dvije razine, te mu u određenim trenutcima kroz dijalog objašnjava što učiniti u kojem trenutku, te igrača polako upoznaje sa pričom.

Slika 13. Kooperativni lik ptica Po

Sada ćemo vidjeti kroz skriptu „BirdFollowing“ kako to sve skupa funkcionira. Na slici 14 koja je početak skripte „BirdFollowing“ vidimo stvorene varijable za brzinu i za distancu pri kojoj se ptica mora zaustaviti, te privatnu varijablu „Transform target“ koja će postati naš igrač. Kao što je gore navedeno postati će naš igrač baš Start() funkcijom kojoj kažemo da je varijabla „target“ baš „GameObject Player“, te baš u tom „GameObject-u“ uzmi komponentu Transform odnosno poziciju.

```
Unity Script | 0 references
public class BirdFollowing : MonoBehaviour
{



    public float speed;
    public float stoppingDistance;

    private Transform target;


    // Start is called before the first frame update
    Unity Message | 0 references
    void Start()
    {
        target = GameObject.FindGameObjectWithTag("Player").GetComponent<Transform>();
    }
```

Slika 14. Ulazne varijable te funkcija Start() u skripti BirdFollowing

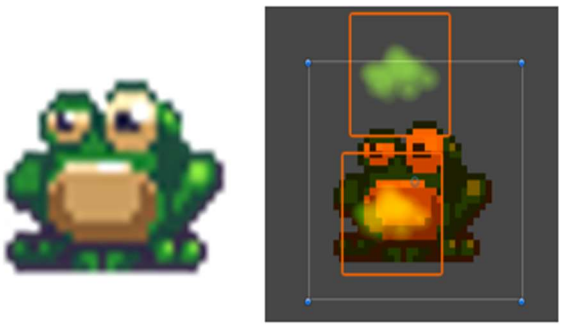
Na slici 15 vidimo upit ako je distanca ptice i distanca našeg igrača veća od varijable „stoppingDistance“, tada pomakni pticu na poziciju igrača. Pri kretanju u lijevo promjeni sprite da gleda u lijevo, a prilikom odlaska desno, postavi vektor da je okrenut desno.

```
void Update()
{
    if (Vector2.Distance(transform.position, target.position) > stoppingDistance)
    {
        transform.position = Vector2.MoveTowards(transform.position, target.position, speed * Time.deltaTime);

    }
    float hdirection = Input.GetAxis("Horizontal");
    //moving left
    if (hdirection < 0)
    {

        transform.localScale = new Vector2(-1, 1);

    }
    //moving right
    else if (hdirection > 0)
    {

        transform.localScale = new Vector2(1, 1);

    }

}
```

Slika 15. Funkcija Update() u skripti BirdFollowing

4.2.3. Protivnici

U igri uz našeg glavnog lika postoje još i šest vrsta neprijatelja na prvoj razini to su: žaba i otrovna žaba koje se nalaze na slici 16, statični orao i orao koji se nalaze na slici 17 dva orla se razlikuju po izgledu (crveni efekt na sprite-u) ima i umjetnu inteligenciju napada na igrača kada igrač uđe u zadano područje. Dok na drugoj razini imamo vojnika koji ima željezni oklop kojeg je nemoguće uništiti napadima na daljinu nego isključivo skokom, te top koji se aktivira i počne ispaljivati zelene otrovne kugle kada igrač uđe u zadano područje, sprite-ovi od neprijatelja sa druge razine sa nalaze na slici 18. Na sve neprijatelje dodana je skripta „Enemy“ koja služi isključivo za nasljeđivanje života tako da možemo postavljati dinamički koliko će koji neprijatelj imati zdravlja



Slika 16. Razlika obične žabe i otrovne žabe

Slika 17. Razlika orla i AI orla

Slika 18. Željezni vojnik i Cannon

Na slici 19 vidimo inspektor varijable koje će služiti žabi da zna od koje do koje točke na koordinatnom sustavu smije ići, hard kodirali smo duljinu i visinu skoka žabe, to inače nije dobra praksa ali u ovom slučaju može proći jer ne igra bitnu ulogu.
Dodali smo opet „LayerMask ground“ da žaba ne može prolaziti kroz određene objekte na mapi. Nadalje standardni animator koji služi za prebacivanje animacija u određenom trenutku sa određenim uvjetima, Rigidbody2D za unos gravitacije, detekcije kolizije itd.

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

public class Frog : MonoBehaviour
{
    [SerializeField] private float leftCap;
    [SerializeField] private float rightCap;
    [SerializeField] private float jumpLenght = 10f;
    [SerializeField] private float jumpHeight = 15f;
    [SerializeField] private LayerMask ground;
    [SerializeField] Animator anim;
    [SerializeField] Rigidbody2D rb;

    private Collider2D coll;
```

Slika 19. Početne inspektor varijable u skripti Frog

Na slici 20 se nalazi funkcija Update() koja nam prebacuje tranziciju iz skoka u pad, sa jednostavnim upitima ako je trenutno uključena animacija „Jumping“ i ako je „velocity“ od RigidBody2D po y-osi manji od .1 tada uključi animaciju padanja, a isključi animaciju skakanja. Dalje u funkciji provjeri ako je igračeva kolizija dotaknula „LayerMask ground“ i ako je animacija „Falling“ postavljena na „true“, tada postavi kod uvjeta na animatoru na „Falling“ dio da je logička varijabla jednaka „false“ tj. ugasi ju.

```
private void Update()
{

    //Transition from Jump to Fall
    if (anim.GetBool("Jumping"))
    {
        if(rb.velocity.y < .1)
        {
            anim.SetBool("Falling", true);
            anim.SetBool("Jumping", false);
        }
    }

    if (coll.IsTouchingLayers(ground) && anim.GetBool("Falling"))
    {
        anim.SetBool("Falling", false);
    }


    //Transition from Fall to Idle

}
```

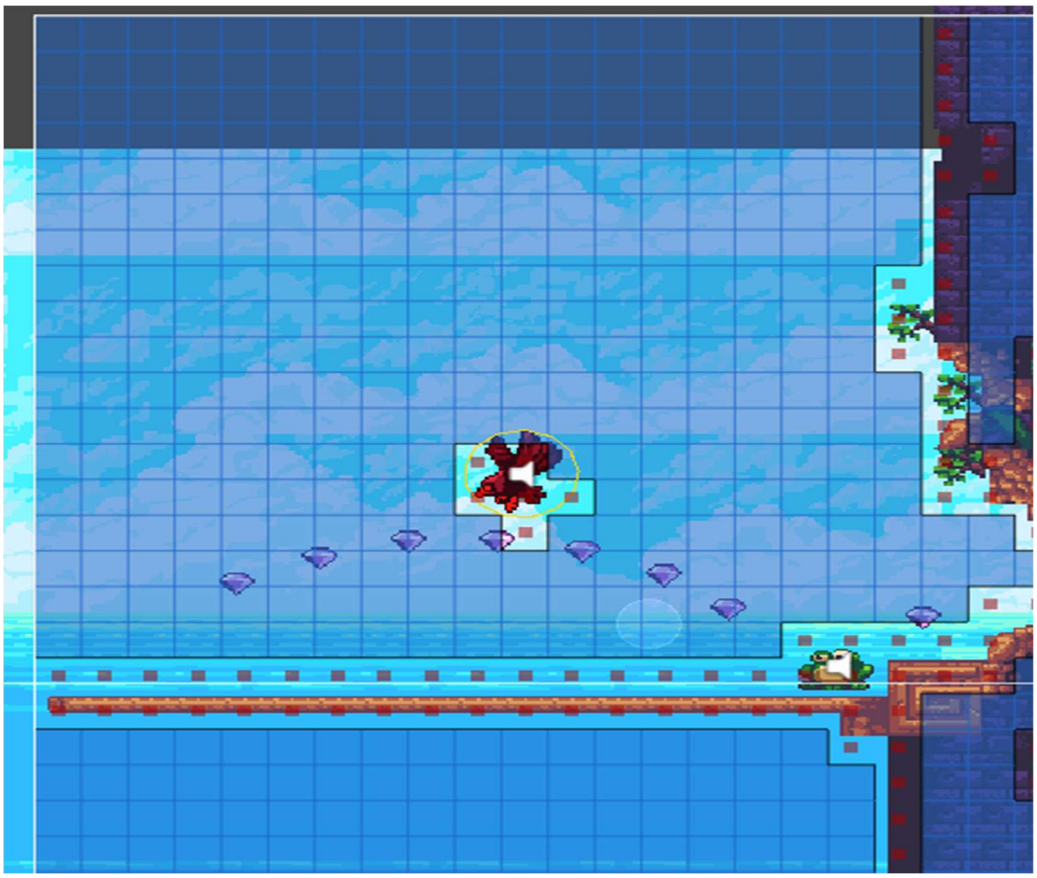
Slika 20. Update() funkcija koja vrši transakcije iz skoka u pad

Zadnja funkcija u „Frog“ skripti govori žabi gdje i kako se kretati, funkcija vidljiva na slici 21. Varijabla „facingLeft„ je na početku skripte postavljena logički izraz „true“ te tako ako je okrenuta prema lijevo događaju se akcije kao što su transformacija sprite-a u lijevu stranu, te ako je kolizijska detekcija sa groundom logički istinita, postavi animaciju „Jumping“ na „true“ te skoči. Druga situacija isto prikazuje samo za drugu stranu.

```
private void Move()
{
    if (facingLeft)
    {
        //test to see if we are beid the leftCap
        if(transform.position.x > leftCap)
        {
            if(transform.localScale.x != 1)
            {
                transform.localScale = new Vector3(1, 1);
            }
            //test to see if i am on the ground, if i am jump
            if (coll.IsTouchingLayers(ground))
            {
                rb.velocity = new Vector2(-jumpLenght, jumpHeight);
                anim.SetBool("Jumping", true);
            }
        }
        else
        {
            facingLeft = false;
        }
            //if it is not we are going to face right
    }
    else
    {
        //test to see if we are beid the leftCap
        if (transform.position.x < rightCap)
        {
            if (transform.localScale.x != -1)
            {
                transform.localScale = new Vector3(-1, 1);
            }
            //test to see if i am on the ground, if i am jump
            if (coll.IsTouchingLayers(ground))
            {
                rb.velocity = new Vector2(jumpLenght, jumpHeight);
                anim.SetBool("Jumping", true);
            }
        }
        else
        {
            facingLeft = true ;
        }
        //if it is not we are going to face right
    }
}
```

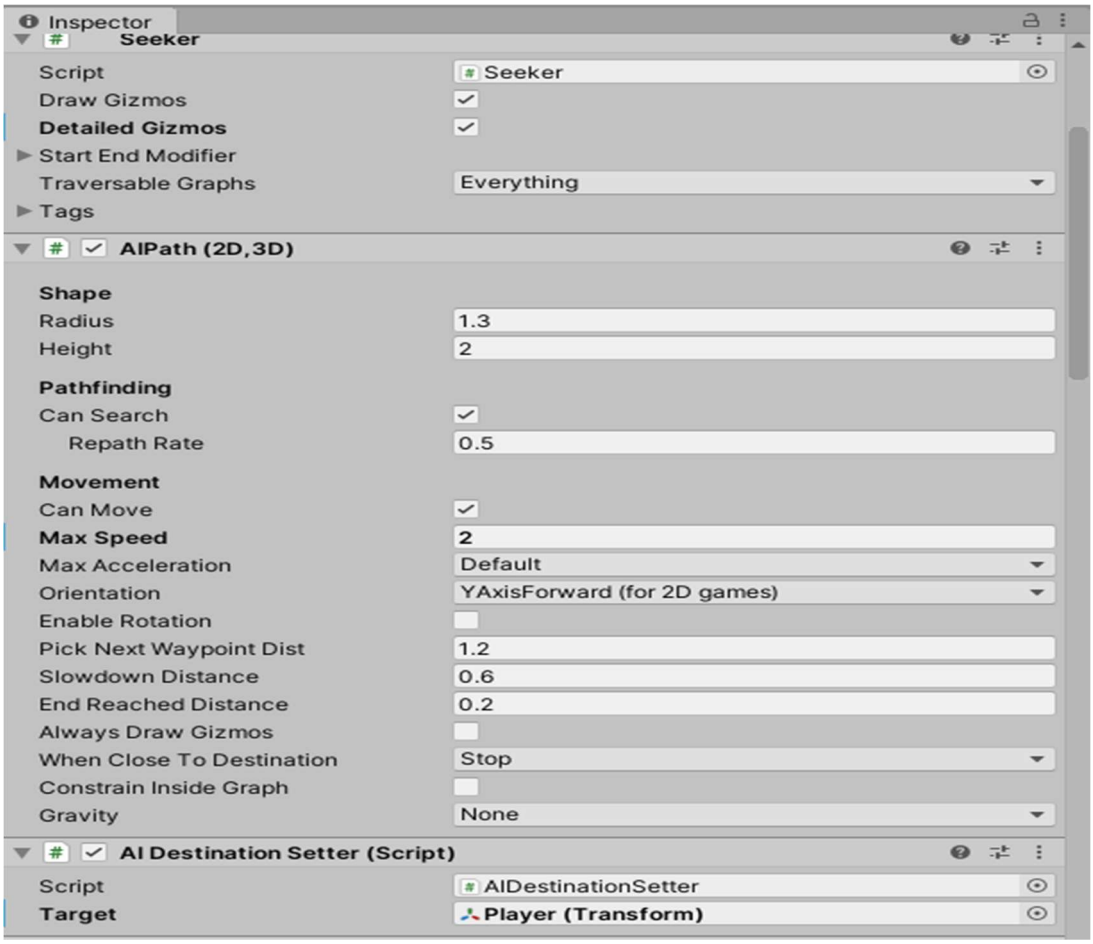
Slika 21. Funkcija Move() u skripti Frog koja služi za kretnju žabe

Drugi i treći neprijatelj je orao, prva varijanta orla je statična u koju je dodan animator za animaciju letenja, te dodana skripta „Enemy“ kako bi prilikom dodira igrača ako to ne bude skok, orao mogao nanijeti štetu igraču. Dok je druga varijanta orla malo zanimljivija spritesheet ovog neprijatelja je crveni da se mogu razlikovati te ima sustav umjetne inteligencije (prati lika unutar zadanog područja, primjer na slici 22).



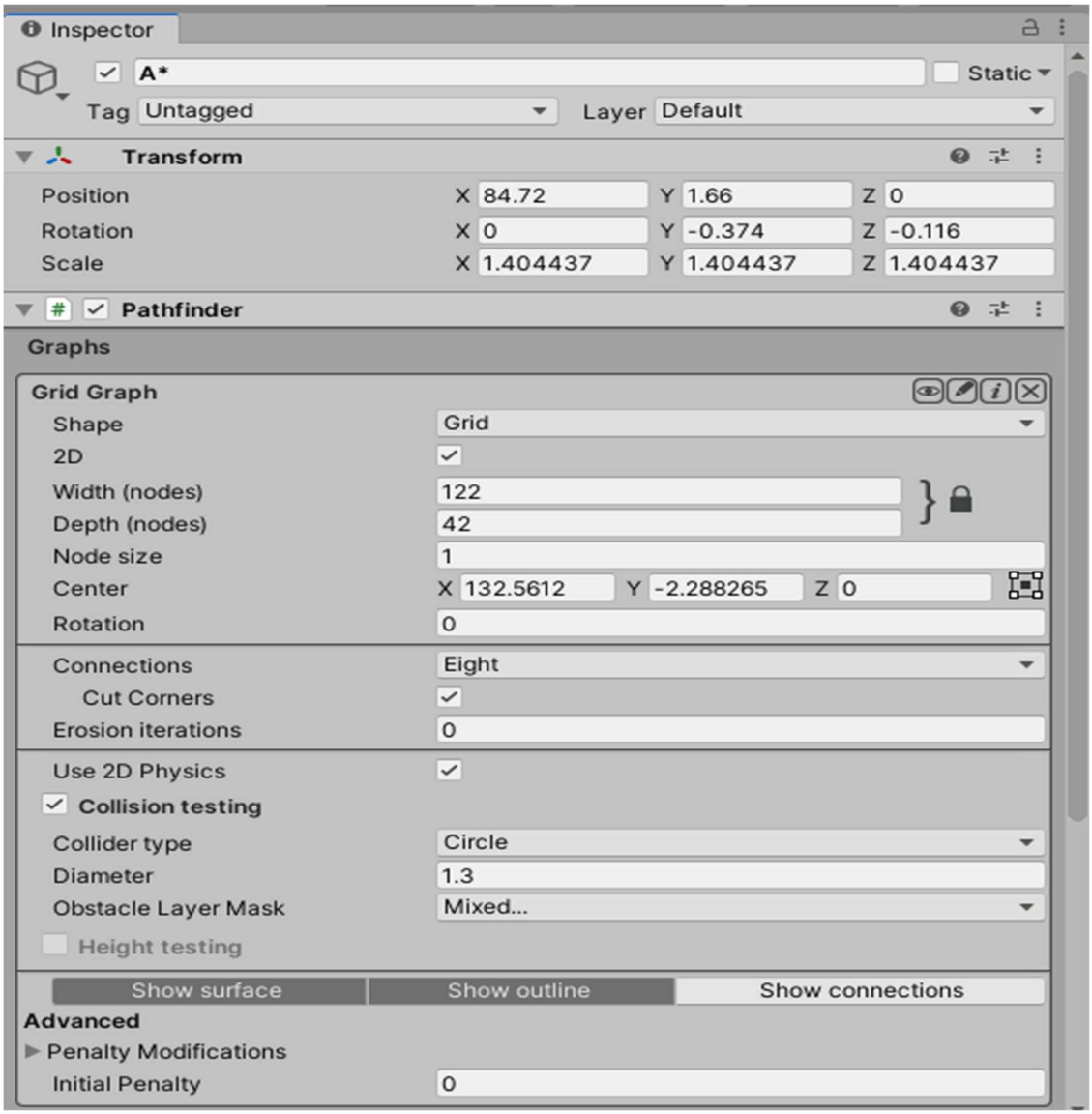
Slika 22. Primjer detekcije i smjer letenja AI orla na prvoj razini

Za ovaj primjer umjetne inteligencije u projekt je importan paket Pathfinding koji je dobro poslužio za ovaj mali dio umjetne inteligencije. Njega smo dodali na objekt lika orla čije su postavke u inspektoru vidljive na slici 23.



Slika 23. . Inspektor AI orla sa modificiranim svojstvima

Ovaj prikazani dio inspektora se odnosi na umjetnu inteligenciju lika orla, pri kojem su postavke napravljene da orao traži svoju metu, a ta meta je zapravo sam igrač. Nadalje dodan je i sam prostor kao objekt prikazan na slici 24. područje gdje AI orao smije letjeti.



Slika 24. Postavke Inspektora A područja AI orla*

Četvrta vrsta neprijatelja je željezni vojnik koji ima skriptu „Enemy_behavior“ koja opisuje njegovo ponašanje i kretnje, te dodatnu skriptu „Enemy“ koja služi kao i svakom neprijatelju da se može manualno u sučelju Unity-a definirati zdravlje lika. Početak skripte vidljiv na slici 25.

```
public class Enemy_behavior : MonoBehaviour
{


    #region Public Variables
    public Transform rayCast;
    public LayerMask raycastMask;
    public float rayCastLength;
    public float attackDistance;
    public float moveSpeed;
    public float timer;
    public Transform leftLimit;
    public Transform rightLimit;
    #endregion

    #region Private Variables

    private RaycastHit2D hit;
    private Transform target;
    private Animator anim;
    private float distance;
    private bool attackMode;
    private bool inRange;
    private bool cooling;
    private float intTimer;
    #endregion
```

Slika 25. Privatne i javne varijable kod neprijatelja "željezni vojnik"

Ovdje su varijable smještene u regije radi bolje preglednosti dijelimo ih na privatne i javne (public). Što se tiče javnih varijabli imamo opet vrstu kretanja kao što smo imali i kod žabe (kretanje od točke A, do točke B). Samo što u ovom primjeru kada se naš glavni lik „Blue Heart“ približi na određenu udaljenost od neprijatelja. Neprijatelj željezni vojnik automatski prestaje sa svim radnjama i kreće u napad na našeg glavnog lika. Kao što možemo vidjeti definirane su varijable za distancu napada, brzina kretanja lika, a od privatnih varijabli vidimo određeni logički izraz je li je naš igrač u njegovoj blizini, i proces hlađenja „cooling“ tj. vremenski „timeout“ između napada. Također, napravljen je i dio u grafičkom sučelju Unity-a možemo dodati objekt kojeg će naš neprijatelj napadati.

Funkcija Awake() vidljiva na slici 26 se poziva kada instanca skripte bude učitana tj. učitava se dodatna funkcija SelectTarget() koja označuje našeg lika, očitava se varijabla „timer“ koja je napominjem opet vremenski „timeout“ između napada. Sadržava još i animator za provođenje različitih animacija kroz određeno vrijeme i u određenim uvjetima.

```
void Awake()
{
    SelectTarget();
    intTimer = timer;
    anim = GetComponent<Animator>();
}
```

Slika 26. Funkcija Awake() u skripti Enemy_behaviour

U funkciji Update() koju vidimo na slici 27 se sve izvršava u svakom frame-u, imamo određene uvjetne naredbe ako neprijatelj ne napada, tada mora izvršavati naredbu kretanja od točke A do točke B. Ako se igrač nalazi unutar granica polja, te ako trenutna animacije nije pod nazivom „enemy_attack“ tada uoči metu i kreni prema njoj. Kada je igrač u određenoj udaljenosti za napasti igrača naređuje se napad. Nadalje, imamo uvjet da ako „collider“ koji je zadužen za detekciju udara nije „null“, upotrijebi funkciju EnemyLogic() koju ćemo dolje detaljnije opisati.

```
void Update()
{
    if (!attackMode)
    {
        Move();
    }

    if(!InsideOfLimits() && !inRange && !anim.GetCurrentAnimatorStateInfo(0).IsName("enemy_attack"))
    {
        SelectTarget();
    }


    if (inRange)
    {
        Debug.Log("PROSLOSUPER!!!!");
        hit = Physics2D.Raycast(rayCast.position, transform.right, rayCastLength, raycastMask);
        RaycastDebugger();
    }

    if(hit.collider != null)
    {
        EnemyLogic();
    }
    else if(hit.collider == null)
    {
        inRange = false;
    }

    if(inRange == false)
    {

        StopAttack();
    }

}
```

Slika 27. Funkcija Update() u skripti Enemy_behaviour

No ako je detekcija udara null, tj. ako je nema, to znači da igrač nije u blizini te postavi logičku varijablu inRange na false, te ako je varijabla inRange na false, pozovi funkciju za obustavu napada. Na slici 28 vidimo OnTriggerEnter2D funkciju koja kad igrač uđe u zadano područje neprijateljskog željeznog vojnika logička varijabla „inRange“ automatski se prebacuje na „true“, što znači da kreće proces napada na našeg igrača, svejedno na kojoj se strani igrač nalazio neprijatelj će okrenuti točno prema njemu pomoću funkcije Flip() koju ćemo kasnije u nastavku spomenuti.

```
void OnTriggerEnter2D(Collider2D trig)
{
    if(trig.gameObject.tag == "Player")
    {
        Debug.Log("TRIGGERAKTIVIRAN");
        target = trig.transform;
        inRange = true;
        Flip();
    }
}
```

Slika 28. Kada će se trigger aktivirati funkcija u skripti Enemy_behaviour

Funkcija EnemyLogic() koju možemo vidjeti na slici 29 ima varijablu distance kojom gleda udaljenost mete tj. našeg „Blue Heart-a“, te ako je distanca veća od napadačke moći neprijatelja prestani napadati, ali ako je napadačka distanca veća ili jednaka od distance i ako je neprijatelj spreman za napad tj. ako mu varijabla „cooling“ postavljena na „true“ počni sa napadom. Na kraju provjeri može li neprijatelj napasti, tu se provjerava da li je „cooling“ postavljen na „true“ ili „false“, ako je postavljen na „true“ odradi funkciju Cooldown() koja gleda ako su varijabla „cooling“ i varijabla „attackMode“ istiniti i ako je varijabla „timer“ manja ili jednaka nuli završi sa „hlađenjem“. Funkciju Cooldown() možemo pogledati na slici 30.

```
void EnemyLogic()
{
    distance = Vector2.Distance(transform.position, target.position);

    if(distance > attackDistance)
    {

        StopAttack();
    }
    else if(attackDistance >= distance && cooling == false)
    {
        Attack();
    }
    if (cooling)
    {
        Cooldown();
        anim.SetBool("Attack", false);
    }
}
```

Slika 29. Funkcija EnemyLogic() u skripti Enemy_behaviour

```
void Cooldown()
{
    timer -= Time.deltaTime;

    if(timer <= 0 && cooling && attackMode)
    {
        cooling = false;
        timer = intTimer;
    }
}
```

Slika 30. Funkcija Cooldown() u skripti Enemy_behaviour

U funkciji Attack() vidljiva na slici 31 neprijatelj dobiva naputke za napad ako je logička varijabla „attackMode“ u istinitom stanju, te mu se dodaje „timer“, te se postavljaju animacije za napad, a isključuju za šetnju od točke A do točke B. U funkciji StopAttack() koja je vidljiva na slici 32 napravljena je obrnuta logika, postavljanje

logičke varijable „cooling“ na „false“, točnije pri sljedećem napadu neprijatelj će odmah krenuti sa napadom. Dok su dalje u funkciji varijabla „attackMode“ i animacija za napad stavljeni na logičku vrijednost „false“.

```
void Attack()
{
    timer = intTimer;
    attackMode = true;

    anim.SetBool("canWalk", false);
    anim.SetBool("Attack", true);
}
```

Slika 31. Funkcija Attack()skripta Enemy_behaviour

```
void StopAttack()
{
    cooling = false;
    attackMode = false;
    anim.SetBool("Attack", false);
}
```

Slika 32. Funkcija StopAttack() skripta Enemy_behaviour

U funkciji SelectTarget() koju možemo pogledati na slici 33 definirali smo distance od našeg igrača prema lijevo i prema desno. Dalje radimo usporedbu ako je distanca s lijeve strane veća nego s desne znači da je naš lik „Blue Heart“ koga neprijatelj napada na lijevoj strani.

U suprotnom je na desnoj strani, te nakon toga izvrši funkciju Flip() koja okreće „spritesheet“ odnosno sliku i same animacije na stranu gdje se igrač nalazi. Funkciju Flip() možemo pogledati na slici 34 koja je na sljedećoj stranici.

```
2 references
private void SelectTarget()
{
    float distanceToLeft = Vector2.Distance(transform.position, leftLimit.position);
    float distanceToRight = Vector2.Distance(transform.position, rightLimit.position);

    if(distanceToLeft > distanceToRight)
    {
        target = leftLimit;
    }
    else
    {
        target = rightLimit;
    }
    Flip();

}
```

Slika 33. Funkcija SelectTarget() u skripti Enemy_behaviour

```
private void Flip()
{
    Vector3 rotation = transform.eulerAngles;
    if(transform.position.x > target.position.x)
    {
        rotation.y = 180f;
    }
    else
    {
        rotation.y = 0f;
    }
    transform.eulerAngles = rotation;
}
```

Slika 34. Funkcija Flip() u skripti Enemy_behaviour

I zadnji neprijatelj u igri je top koji ispaljuje otrovne kugle na poziciju gdje se igrač nalazi i to svake dvije sekunde. Top također ima „Enemy“ skriptu koja mu određuje trenutno zdravlje, ali ima još i skriptu „Cannon“ čiji početak možemo vidjeti na slici 35.

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

public class Cannon : MonoBehaviour
{

    [SerializeField]
     GameObject bullet;

    float fireRate;
    float nextFire;
```

Slika 35. Globalne varijable u skripti Cannon

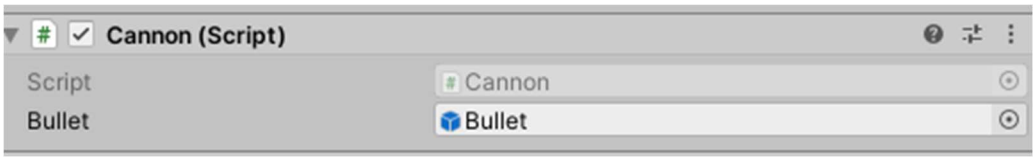
Ovdje smo definirali SerializeField koji nam služi i govori Unity engine-u da spremi ili vrati taj podatak iz diska, tu smo napravili „gameObject bullet“ u kojeg smo spremili „prefab“ odnosno otrovnu kuglu. Dalje smo još definirali varijable brzine ispaljivanja otrovne kugle. Ovdje vidimo dvije funkcije koje možemo pogledati na slici 36. Start() koja se aktivira na početku i tu smo stavili brzinu ispaljivanja otrovne kugle, te sljedeći vremenski rok ispaljivanja sljedeće. Dok se u funkciji Update() poziva druga funkcija CheckifTimeToFire(),funkcija vidljiva na slici 37, koju ćemo detaljnije pogledati ispod, a ona nam provjerava je li vrijeme koje je prošlo veće od varijable „nextFire“ ako je uvjet točan automatski stvori „gameObject bullet“ i dodaj na varijablu „nextFire“ trenutno vrijeme i vrijeme od varijable „fireRate“. U „gameObject“ „Cannon“ odnosno karakter topa dodali smo „prefab“ tj. objekt koji je otrovna kugla te i ona ima svoju skriptu, koju možemo vidjeti na slici 38.

```
// Start is called before the first frame update
void Start()
{


    fireRate = 3f;
    nextFire = Time.time+1;
}

// Update is called once per frame
void Update()
{
    CheckIfTimeToFire();

}
```

Slika 36. Start() i Update() funkcija u Cannon skripti

```
void CheckIfTimeToFire()
{
    if(Time.time > nextFire)
    {

        Instantiate(bullet, transform.position, Quaternion.identity);
        nextFire = Time.time + fireRate;


    }
}
```

Slika 37. Funkcija CheckifTimeToFire() u skripti Cannon



Slika 38. Inspektor game objekta Cannon

Na slici 39 deklarirane su varijable brzine određene otrovne kugle, deklarirana je i varijabla „LayerMask“ koja govori metku gdje može proći a da se odmah ne uništi npr. (kada otrovna kugla dodirne našeg lika odmah se uništava te ako dodirne bilo „foregrounda“). Dodan je još i „GameObject“ „destroyEffect“ gdje su korišteni „Partical Effects-i“ za bolji doživljaj pri zabijanju kugle gore napisani dio. Dakle, vidimo još i varijablu trajanja otrovne kugle u sekundama, te smo pozvali skriptu „PlayerControllers“ koju će otrovna kugla smatrati neprijateljem te mu svakim udarcem smanjivati po jedno zdravlje. „PlayerControllers“ je još bitan zbog raspoznavanja trenutne lokacije našeg lika.

```
public class EnemyBullet : MonoBehaviour
{


    float moveSpeed = 7f;
    public LayerMask whatIsSolid;
    public GameObject destroyEffect;
    public float lifeTime;


    Rigidbody2D rb;


    public PlayerControllers otherscript;
    PlayerControllers target;

    Vector2 moveDirection;
```

Slika 39. Globalne varijable u skripti EnemyBullet

U Start() funkciji koju vidimo na slici 40 imamo funkciju Invoke() koja djeluje na vremenski period, dalje smo definirali metu, našeg lika te zadali da se pronađe skripta pod nazivom „PlayerControllers“ zbog označavanja trenutne lokacije. Daljnja varijabla „moveDirection“ određuje putanju otrovne kugle prema izračunatoj poziciji gdje se naš lik nalazio, te za kraj definiran je „velocity“ preko RigidBody2D, odnosno smjer po x i y osi.

```
// Start is called before the first frame update
void Start()
{

    Invoke("DestroyProjectile", lifeTime);
    rb = GetComponent<Rigidbody2D>();
    target = GameObject.FindObjectOfType<PlayerControllers>();
    moveDirection = (target.transform.position - transform.position).normalized * moveSpeed;
    rb.velocity = new Vector2(moveDirection.x, moveDirection.y);


}
```

Slika 40. Start() funkcija u Enemy_bullet skripti

U navedenim funkcijama koje vidimo na slici 41, prilikom dolaska u određeno područje top se aktivira i počne ispaljivati otrovne kugle, ako metak dodirne našeg igrača automatski se poziva funkcija HolderHealth() u kojoj smo ranije vidjeli da se igraču smanjiva zdravlje za jedan dolje. Druga funkcija je DestroyProjectile() koja stvara „destroyEffect“ postavljen na njega te ga uništava.

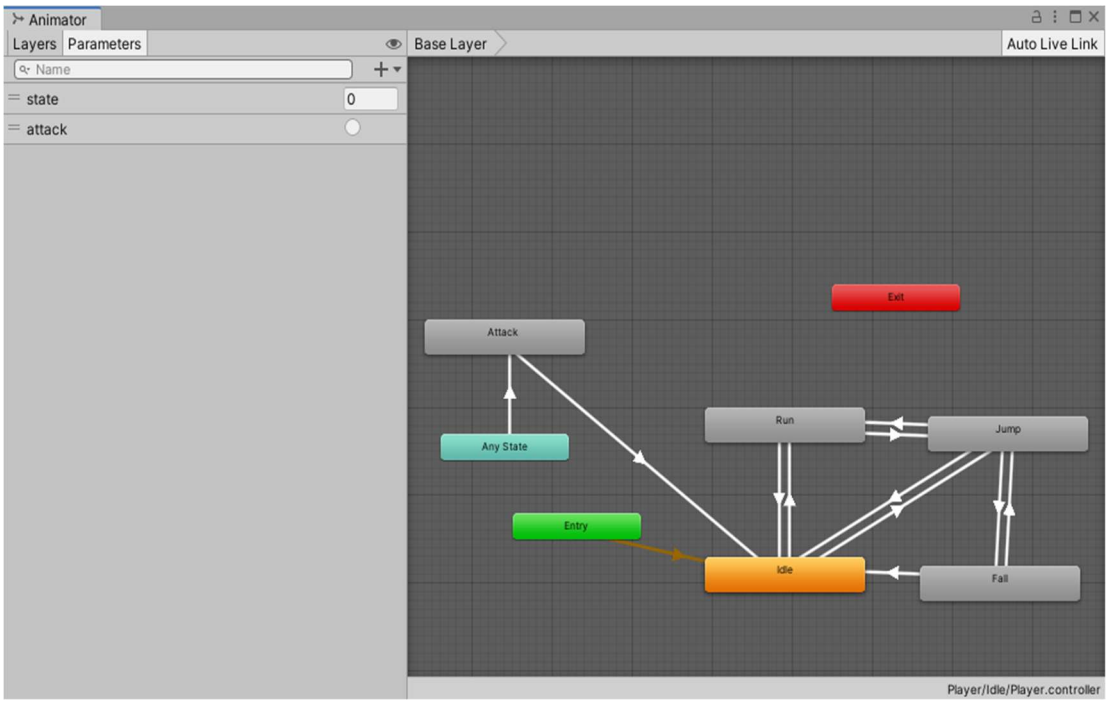
```
public void OnTriggerEnter2D(Collider2D other)
{
    if (other.tag == "Player")
    {

        Debug.Log("Hit");
        target.GetComponent<PlayerControllers>().HolderHealth();
        DestroyProjectile();
    }
}
void DestroyProjectile()
{
    Instantiate(destroyEffect, transform.position, Quaternion.identity);
    Destroy(gameObject);


}
```

Slika 41. Prikaz funkcija OnTriggerEnter2D i DestroyProjectile() u skripti Enemy_bullet

4.3. Animator

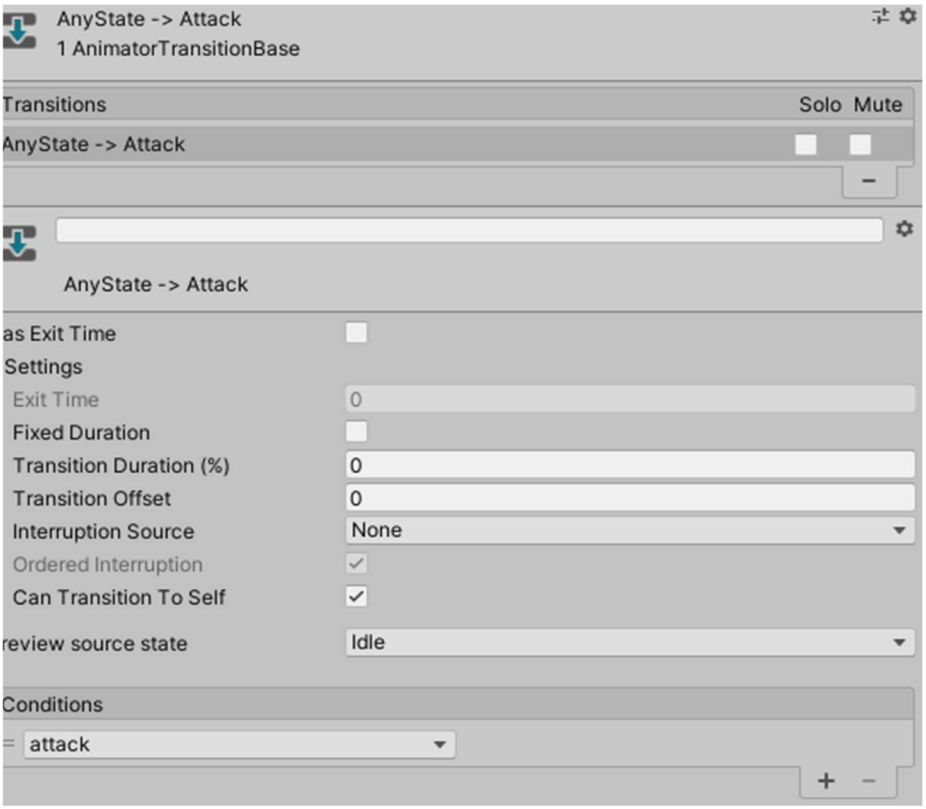
Animator je „stroj stanja“ koji obrađuje koje animacije se trenutno odvijaju te stvara blagi prijelaz između animacija [2]. U desnom djelu sučelja vidimo stablo koje se sastoji od različitih animacija objekta čiji je kontroler, u ovom slučaju to je kontroler od objekta igrača (Player) [5].

Narančasti pravokutnik predstavlja početno stanje u kojemu će objekt biti, to je stanje mirovanja. Ostala stanja su povezana strelicama koje naznačuju koja animacija se može pokrenuti poslije trenutnog stanja. Recimo, iz stanja mirovanja može elegantno ući u stanje trčanja i stanje skakanja i obrnuto, dok recimo u stanje padanja jedino može lik doći kada skoči. Imamo i plavi pravokutnik (Any State) koji nam prikazuje u koja stanja objekt može ući iz bilo kojeg drugog stanja, npr. Ako igrač trenutno trči, skače ili pada on će moći napasti napadom u daljinu svojim šurikenom iz bilo kojeg stanja. Pojedini objekti u Unity-u koji imaju animaciju imaju i kontroler koji upravlja tom animacijom te ako ima više animacija upravlja prijelazom iz animacije u animaciju. Na slici 42 prikazano je sučelje animatora.



Slika 42. Kontroler animator lika Player

Kako bi se prijelaz iz stanja u neko druge stanje dogodio moraju se zadovoljiti neki kriteriji. Na slici 42 vidimo na lijevoj strani parametre, koje na gumb + možemo s lakoćom dodati, parametri mogu biti tipa: bool, trigger, float, i int. Parametre koristimo i ubacujemo na strelice između stanja prikazano na slici ispod. Kao što možemo vidjeti u dijelu „Conditions“ mijenjamo različite parametre, i postavljamo ono što nam najviše odgovara, sliku opcija u animator inspektoru može se pogledati na slici 43.

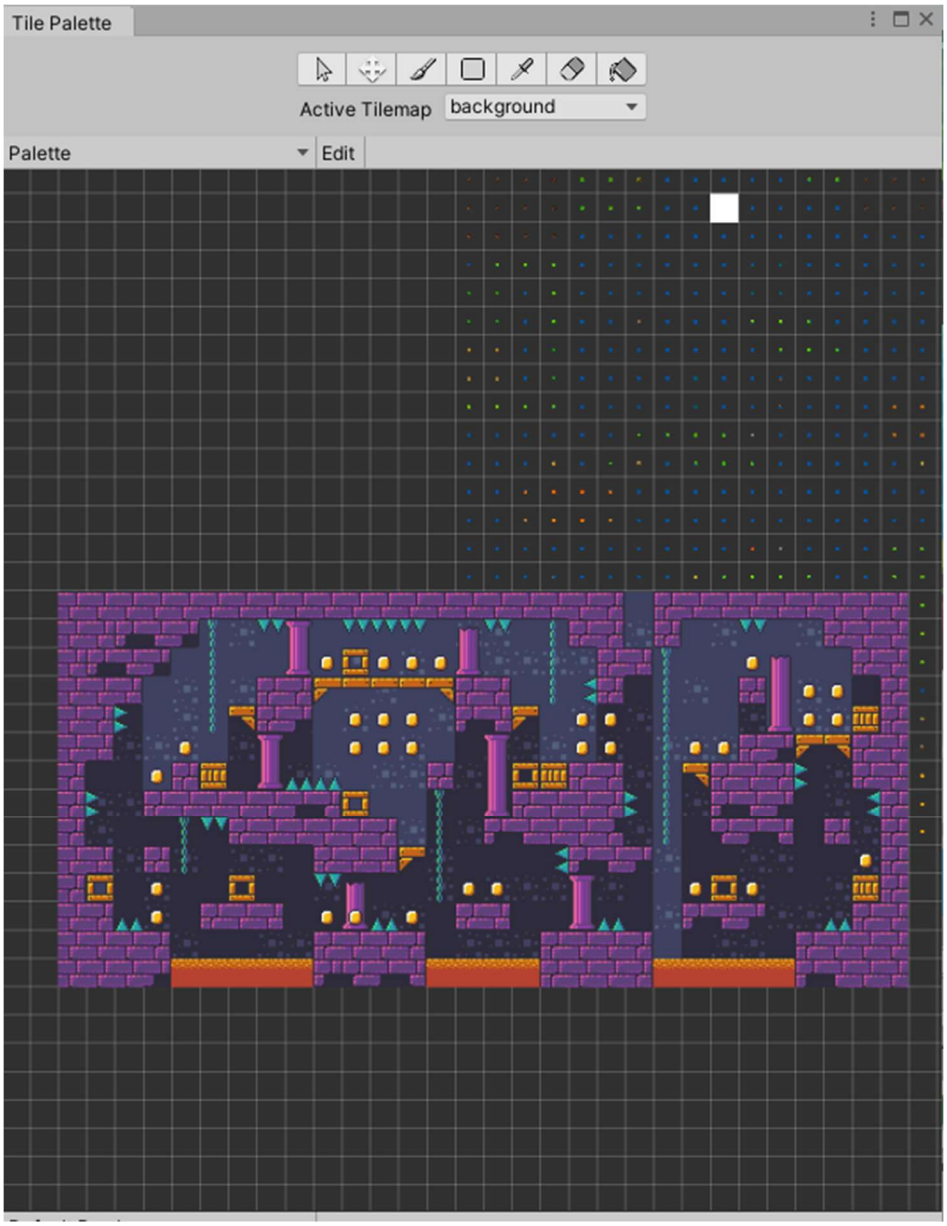


Slika 43. Inspektor u dijelu animatora

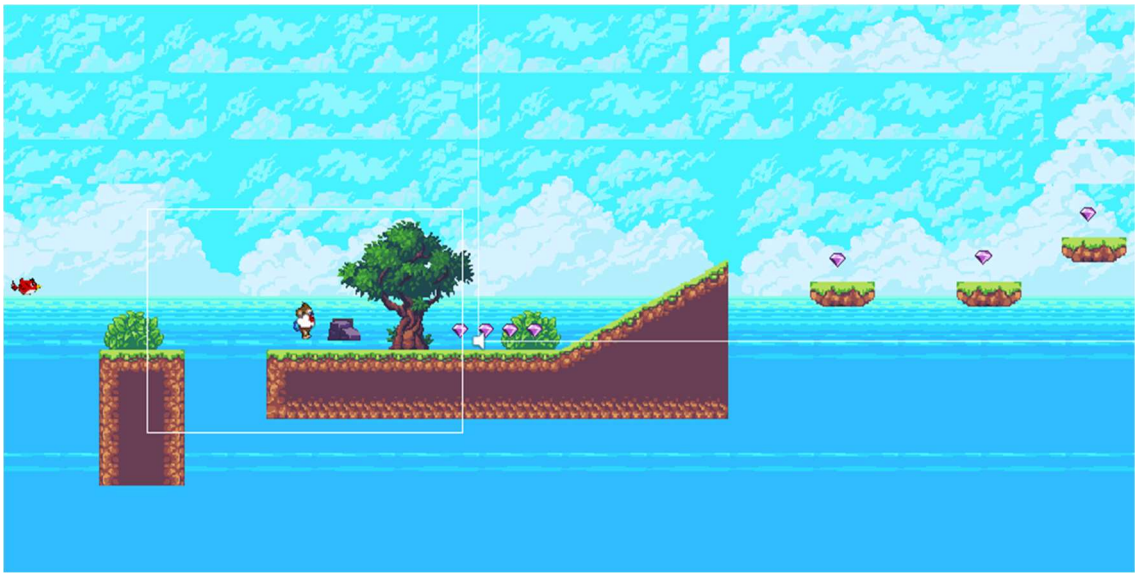
4.4. Dizajn razina

Kod stvaranja razina na igri importan je 2D paket Tile Pallete. U Tile Palleti napravljene su dvije vrste „background“ i „foreground“. „Background“ bi prikazivao onaj dio na mapi sa kojim se ne bi moglo doći u doticaj nego je samo služio kao vizualni detalj. Dok s druge strane imamo „foreground“ koji je zapravo aktivno tlo na kojem igrač stoji. Dizajniranje razine kreće sa „importanjem“ odnosno ubacivanjem sprite-ova unutar same Tile Palette, te ovisno koji dio nam treba da li je to „background“ ili „foreground“ postavljamo na našu scenu. Alat za uređivanje razina vidljiv je na slici 44. Primjer prve i druge razine možemo pogledati na slici 45 i slici 46.

U projektne datoteke postavljeni su sprite-ovi, ponekad sprite tekstura sadrži samo jedan grafički element, ali je često prikladnije kombinirati nekoliko povezanih grafika u jednu sliku. Na primjer, slika može sadržavati dijelove jednog znaka, kao kod automobila čiji se kotači pomiču neovisno o tijelu. Unity olakšava izdvajanje elemenata iz složene slike dodavanjem uređivača sprite-ova [4].



Slika 44. Izgled paketa za dizajniranje druge razine



Slika 45. Izgled početka prve razine

Slika 46. Izgled početka druge razine

4.5. Grafičko korisničko sučelje

Zapravo je način interakcije čovjeka s računalom kroz manipulaciju grafičkim elementima i dodacima uz pomoć tekstualnih poruka i obavijesti.
Korisničko sučelje je vrlo bitno igračev doživljaj u igri, sučelje igraču ukazuje na stanje igre te mu omogućuje neke opcije unutar igre.

4.5.1. Glavni izbornik (Main Menu)

Glavni izbornik je početna scena prilikom ulaska u igru koju možemo vidjeti na slici 47. Sastoji se od pozadine, imena igre, i četiri gumba: Start, Options, Controls, Quit. Na slici vidimo sve nabrojane sastavnice glavnog izbornika. Gumbovi i tekst su posebno razdvojeni objekti koji se postavljaju kao djeca objekta „canvas“ koji služi kao prikaz korisničkog sučelja. Gumbovi su zapravo tekst kojima je pridodan atribut Button te tako imaju funkcionalnost gumba. Dolje ćemo vidjeti funkcionalnost gumba Play i Quit iz skripte MainMenu.

Slika 47. Prva scena u igri, glavni izbornik

Na slici 48 vidimo funkcije PlayGame() i QuitGame(), prva funkcija je vrlo jednostavna učitali smo „SceneManager“ te preko njega pozvali sljedeću scenu, a prva sljedeća scena je prva razina. S druge strane korištena je samo naredba za izlaz. Na slici 49 ispod možemo pogledati redoslijed scena po redoslijedu.

```
Unity Script | 0 references
public class MainMenu : MonoBehaviour
{
    0 references
    public void PlayGame()
    {
        SceneManager.LoadScene(SceneManager.GetActiveScene().buildIndex +1);
    }

    0 references
    public void QuitGame()
    {
        Debug.Log("QUIT!");
        Application.Quit();
    }
}
```

Slika 48. Skripta MainMenu

Slika 49. Build settings redoslijed scena u igri

4.5.2. Sučelje unutar razine (Player UI)

Na prvoj razini na sučelju imamo prikazano trenutno stanje životnih bodova u gornjem lijevom kutu, sa trenutnim stanjem prikupljenih dijamanata koji su potrebni da bi se otišlo na drugu razinu. Primjer prikaza životnih bodova te trenutno prikupljenih dijamanata prikazan je na slici 50.

Slika 50. Prikaz životnih bodova te trenutno prikupljenih dijamanata

Na drugoj razini imamo malo drugačiji UI dakle naš glavni lik se nalazi na početku druge razine i vrata su zaključana. Igrač mora pronaći ključ i tek kada pronađe ključ, ključ sa zvučnim signalom i UI promjenom se pojavi na zaslonu, to možemo vidjeti na slici 51 ispod.

Slika 51. Prikaz UI druge razine kada se pronađe ključ

Bitna stavka u igri je i dijalog koji je isto dio sučelja unutar razine, dijalog se pojavljuje svaki put kada igrač dođe na određeno područje koje ima funkciju da aktivira dijaloški okvir. Dijaloški okviri su vrlo bitna stavka igra jer polako otkrivaju priču. Primjer dijaloga vidljiv na slici 52.

Slika 52. Prikaz dijaloga u igri

Unutar svake razine imamo opciju pauze preko „Esc“ tipke, aktivira nam se UI izbornik te se pauzira igra, pauza vidljiva na slici 53, izbornik nam daje mogućnosti da nastavimo igrati tamo gdje smo stali ili da otiđemo na glavni izbornik ili jednostavno da ugasimo cijelu igru. Na slici 54 ćemo vidjeti koja funkcija je izvedena kada smo stisnuli tipku „Esc“.

Slika 53. Izbornik pauze

```
public class PauseMenu : MonoBehaviour
{

    public static bool GameIsPaused = false;

    public GameObject pauseMenuUI;

    // Update is called once per frame
    void Update()
    {
        if (Input.GetKeyDown(KeyCode.Escape))
        {
            if (GameIsPaused)
            {
                Resume();
            }
            else
            {
                Pause();
            }
        }
    }

    public void Resume()
    {
        pauseMenuUI.SetActive(false);
        Time.timeScale = 1f;
        GameIsPaused = false;
    }

    void Pause()
    {
        pauseMenuUI.SetActive(true);
        Time.timeScale = 0f;
        GameIsPaused = true;
    }
```

Slika 54. Skripta PauseMenu

5. ZAKLJUČAK

U ovom radu je izrađen 2D računalni platformer u Unity Engine-u uz pomoć C# objektno-orijentiranog programskog jezika. U Unity-u imamo objekte koji su temeljni dio igre te na njih dodajemo C# skripte, sprite-ove te različita druga svojstva kao što su animacije, zvuk, različiti efekti itd. Unity Engine je vrlo pristupačan alat za izradu video igara. Unity je među-platformsko razvojno okruženje za razvoj igara koje je razvijeno od strane Unity Technologies. Verzija programa koja se koristi za izradu projekta i rada jest Unity 2019.4. Programski jezik u projektu je C#.

Igra sama po sebi se temelji na priči koju ptica Po priča našem glavnom liku, priča o samoj igri je napravljena putem dijaloških okvira te bez problema možemo napraviti različite nadogradnje koje bi donosile određene zaplete u igri. „The story of a blue Heart“ prikazuje glavne elemente klasične platformska igre u kojoj pratimo glavnog lika koji savladava prepreke, uništava neprijatelje te obavlja određene zadatke da bi došao u svoju džunglu koju treba obraniti od neprijatelja. Cilj ove igre je potaknuti igrača na efektivno i logičko razmišljanje. Igra je namijenjena prvenstveno mlađem uzrastu.

Kako bi se mogla izraditi jedna takva igra potrebno je posjedovati osnovno znanje za programiranje i korištenje programskog jezika i razumijevanja koncepta kako se objekti mogu kontrolirati pomoću skripti. Potrebno je i poznavati osnove fizike kretanja i odnosa među objektima koja je nužna kao element i daje smislenost igri kao takva.

Moguće su još brojne dorade na igri poput dodavanja glavnog neprijatelja tj. „Boss-a“ koji ima različite napade u intervalima i ima enormno više virtualnih života od ostalih neprijatelja te je još moguća nadogradnja više različitih razina ili dodavanje nekakve bonus razine npr. „puzzle“ (dodatna razina). Što se tiče glavnog izbornika moguće je dodavanje opcije za odabir različitih jezika s kojim bi igra dobila veći publicitet, izmjena kontroli jer moramo razmišljati da će možda i ljudi koji su ljevoruki igrati igru pa će im zbog toga odgovarati drugačiji poredak kontrola.

LITERATURA

[1] Unity Editor 2020.1

https://unity.com/learn

[2] The Animator Controler

https://learn.unity.com/tutorial/controlling-animation#

[3] Freesound

https://freesound.org/people/manu_rochon/downloaded_sounds/

[4] Unity documentation, 2019.1

https://docs.unity3d.com/Manual/SpriteEditor.html

[5] Unity documentation

https://docs.unity3d.com/Manual/StateMachineBasics.html

[6] „Hollow Knight“, Wikipedia

https://en.wikipedia.org/wiki/Hollow_Knight.html

[7] „Celeste“, Wikipedia

https://en.wikipedia.org/wiki/Celeste_(video_game).html

POPIS SLIKA

Slika 1. Snimak ekrana iz igre Hollow Knight 4
Slika 2. Snimka ekrana iz igre Celeste 6
Slika 3. Skripta CameraController 7
Slika 4. Unity inspektor modifikacija kamere 7
Slika 5. Sprite igrača (BlueHeart) 8
Slika 6. Inicijalne ulazne varijable u PlayerControllers skripti 9
Slika 7. Inspektor varijabli u PlayerControllers skripti 9
Slika 8. Funkcija za prikupljanje u PlayerControllers skripti 10
Slika 9. Funkcija za detekciju prilikom susreta sa neprijateljem 11
Slika 10. Prikaz funkcije HolderHealth() 11
Slika 11. Funkcija Movement() kretnje igrača 12
Slika 12. Funkcija AnimationState() u skripti PlayerControllers 13
Slika 13. Kooperativni lik ptica Po 13
Slika 14. Ulazne varijable te funkcija Start() u skripti BirdFollowing 14
Slika 15. Funkcija Update() u skripti BirdFollowing 14
Slika 16. Razlika obične žabe i otrovne žabe 15
Slika 17. Razlika orla i AI orla 15
Slika 18. Željezni vojnik i Cannon 16
Slika 19. Početne inspektor varijable u skripti Frog 16
Slika 20. Update() funkcija koja vrši transakcije iz skoka u pad 17
Slika 21. Funkcija Move() u skripti Frog koja služi za kretnju žabe 18
Slika 22. Primjer detekcije i smjer letenja AI orla na prvoj razini 19
Slika 23. . Inspektor AI orla sa modificiranim svojstvima 20
Slika 24. Postavke Inspektora A* područja AI orla 21
Slika 25. Privatne i javne varijable kod neprijatelja "željezni vojnik" 22
Slika 26. Funkcija Awake() u skripti Enemy_behaviour 23
Slika 27. Funkcija Update() u skripti Enemy_behaviour 24
Slika 28. Kada će se trigger aktivirati funkcija u skripti Enemy_behaviour 24
Slika 29. Funkcija EnemyLogic() u skripti Enemy_behaviour 25
Slika 30. Funkcija Cooldown() u skripti Enemy_behaviour 25
Slika 31. Funkcija Attack()skripta Enemy_behaviour 26
Slika 32. Funkcija StopAttack() skripta Enemy_behaviour 26

Slika 33. Funkcija SelectTarget() u skripti Enemy_behaviour 27
Slika 34. Funkcija Flip() u skripti Enemy_behaviour 27
Slika 35. Globalne varijable u skripti Cannon 27
Slika 36. Start() i Update() funkcija u Cannon skripti 28
Slika 37. Funkcija CheckifTimeToFire() u skripti Cannon 28
Slika 38. Inspektor game objekta Cannon 29
Slika 39. Globalne varijable u skripti EnemyBullet 29
Slika 40. Start() funkcija u Enemy_bullet skripti 30
Slika 41. Prikaz funkcija OnTriggerEnter2D i DestroyProjectile() u skripti Enemy_bullet 31
Slika 42. Kontroler animator lika Player 32
Slika 43. Inspektor u dijelu animatora 33
Slika 44. Izgled paketa za dizajniranje druge razine 35
Slika 45. Izgled početka prve razine 36
Slika 46. Izgled početka druge razine 36
Slika 47. Prva scena u igri, glavni izbornik 37
Slika 48. Skripta MainMenu 38
Slika 49. Build settings redoslijed scena u igri 38
Slika 50. Prikaz životnih bodova te trenutno prikupljenih dijamanata 39
Slika 51. Prikaz UI druge razine kada se pronađe ključ 39
Slika 52. Prikaz dijaloga u igri 40
Slika 53. Izbornik pauze 41
Slika 54. Skripta PauseMenu 41