Izrada platformer video igrice

Curavić, Mario

SVEUČILIŠTE U DUBROVNIKU
ODJEL ZA ELEKTROTEHNIKU I RAČUNARSTVO

IZRADA PLATFORMER VIDEO IGRE

ZAVRŠNI RAD

Dubrovnik, rujan 2022.

SVEUČILIŠTE U DUBROVNIKU
ODJEL ZA ELEKTROTEHNIKU I RAČUNARSTVO

IZRADA PLATFORMER VIDEO IGRE

ZAVRŠNI RAD

Studij: Primijenjeno/poslovno računarstvo
Kolegij: Objektno orijentirano programiranje
Mentor: izv. prof. dr. sc. Krunoslav Žubrinić
Komentorica: Ana Kešelj, mag. ing. comp.
Student: Mario Curavić
JMBAG: 0036518350

Dubrovnik, rujan 2022.

SAŽETAK

Ovaj završni rad opisuje pojam 2D platformerskog žanra video igre kao i neke od njegovih najstarijih podžanrova. S obzirom na osnovne karakteristike 2D platformera u ovom radu je izrađen prototip video igre koji je inspiriran igrama poput *Mega Man* i *Shovel Knight*.

Kroz rad su pojašnjene sve programerske skripte korištene pri izradi jedinstvenog prototipa 2D platformer video igre. Rad također opisuje i sve scene koje se nalaze unutar spomenute video igre napravljene u *pixel art* stilu unutar *Unity* programskog okruženja.

Ključne riječi: *2D platformer, video igra, Unity, pixel art*

ABSTRACT

This undergraduate thesis describes the concept of a 2D platformer video game genre as well as some of its oldest subgenres. Taking into account the basic characteristics of 2D platformers, in this paper a video game prototype was created that was inspired by games like Mega Man and Shovel Knight.

The paper explains all the programming scripts used in the creation of this unique prototype of a 2D platformer video game. It also describes all the scenes that are inside the mentioned video game made in pixel art style within the Unity game engine.

Keywords*: 2D platformer, video game, Unity, pixel art*

SADRŽAJ

1 UVOD

Cilj ovog završnog rada je izrada prototipa video igre koji se sastoji od glavnog izbornika i jedne razine. Video igra pripada žanru dvodimenzijskog (2D) platformera i uzima veliku inspiraciju iz igara *Mega Man* i *Shovel Knight* [1]. Prototip video igre će biti izrađen unutar Unity programskog okruženja koristeći C# programski jezik, a za izradu vizualnih elemenata prototipa koristi će se program PixilArt.

Završni rad je podijeljen na osam poglavlja:

- Uvod;
- Povijest platformera;
- Video igra King's Return;
- Lik igrača;
- Neprijatelji;
- Vizuali video igre;
- Audio video igre;
- Zaključak.

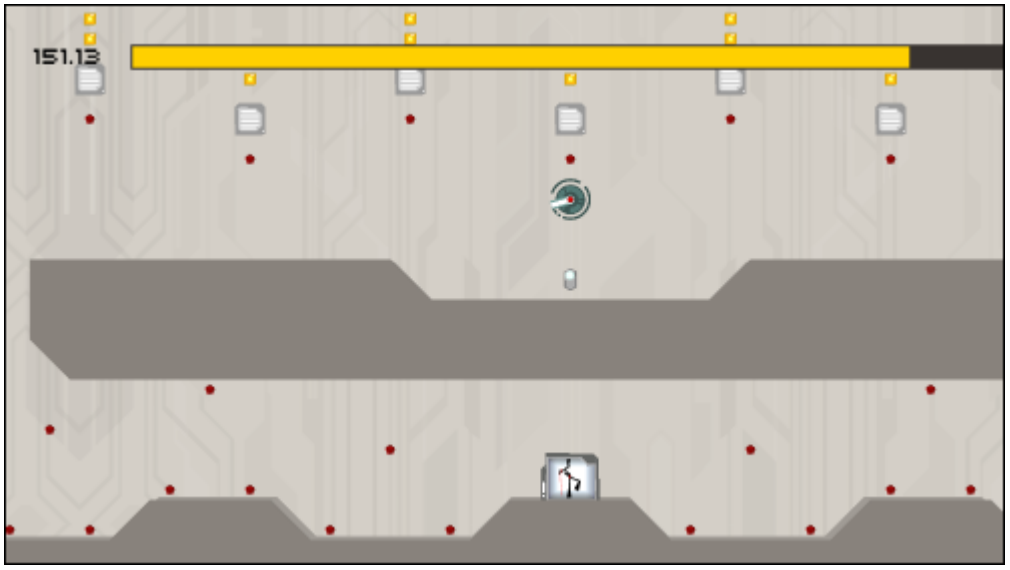
Uvod opisuje prototip video igre koji se izrađuje kroz ovaj rad. U poglavlju Povijest platformera opisane su osnovne značajke koje čine 2D platformerski žanra kao i opisi nekih od najpopularnijih podžanrova u ranim danima 2D platformera. Treće poglavlje objašnjava bitne elemente Unity programskog okruženja i scene koje koristimo unutar prototipa video igre. Sve potrebne programske skripte za rad lika igrača opisane i objašnjene su u istoimenom poglavlju. Poglavlje Neprijatelji opisuje ponašanje svih neprijatelja unutar video igre kao i skripte koje omogućuju to ponašanje. Poglavlje Vizuali video igre pokazuju koje su animacije korištene za lik igrača i neprijatelje. Poglavlje Audio video igre opisuje kako su muzika i zvučni efekti implementirani unutar video igre. U Zaključku je opisan finalni prototip video igre koji je napravljen kroz ovaj rad.

2 POVIJEST PLATFORMERA

Jedan od najstarijih žanrova video igara su platformerske video igre. U ovom stilu video igre igrač kontrolira lika koji prolazi kroz različite razine u kojima moraju skakati između raznih platforma ili preko raznih prepreka i mogu uključivati borbu protiv neprijatelja. Platformerske igre se najčešće klasificiraju kao dio žanra akcijskih igara [2].

Premda se tada nije koristio termin platformske video igre, prve video igre koje bi po današnjoj definiciji pripadale žanru su izašle početkom 1980-ih. Za titulu prve platformerske igre postoji debata između video igre *Space Panic* [1] koja je izašla 1980. godine i *Donkey Kong* koja je izašla 1981. godine. Iako je *Space Panic* izašao ranije i dijeli dosta elemenata s budućim platformerima, nedostatak mogućnosti skakanja glavnog lika dovodi u upit njezino uključivanje u žanr.

Platformere možemo generalno podijeliti u dva tipa: jednozaslonske igre i igre s pomičnom kamerom. Jednozaslonski platformeri se uobičajeno sastoje od više razina koje su svaka prikazana na vlastitom zaslonu. Za prelazak s jedne razine na drugu igrač mora doći do nekog određenog cilja ili mora pobijediti sve neprijatelje na trenutnoj razini. Ovaj tip platformera je bio vrlo popularan u doba arkadnih video igara, kao na primjer *Donkey Kong* i *Acid Trap*, a u novijim igrama, igre poput *N* i *N+* (Slika 1.) su dobar primjer ovog tipa platformera.



Slika 1 Razina iz N+ video igre

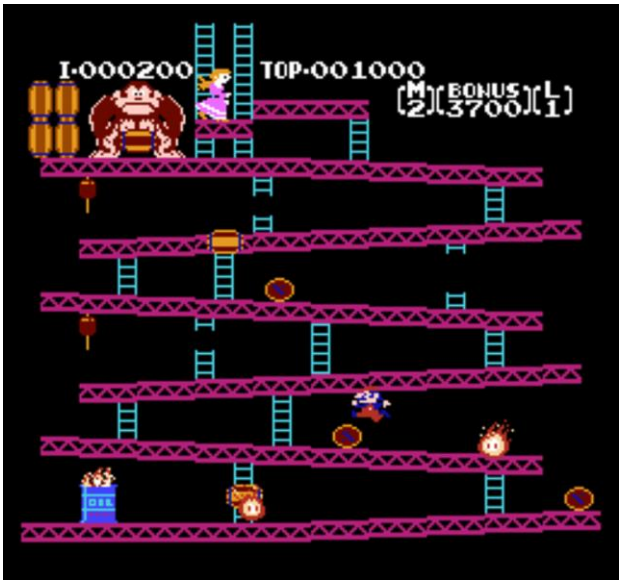
Platformeri s pomičnom kamerom imaju kameru koja horizontalno ili vertikalno slijedi igrača kroz razinu. Ovaj tip platformera se sastoji od više razina gdje je cilj doći do kraja. Veliki broj

igara ovog tipa sadrži borbu protiv glavnog neprijatelja (engl. *boss*) na kraju pojedinačnih razina. Primjeri ovog tipa platformskih video igara su *Super Mario Bros*, *Mega Man* i *Shovel Knight* (Slika 2.).

Slika 2 Isječak iz Shovel Knight video igre

2.1 Platformeri u 1980-ima

Među prvim podžanrovim koji su proizašli iz platformerskih video igara su *hop and bop* video igre. Ove igre su povezana sa svojim vizualima punim jarkih boja koji podsjećaju na crtane filmove i glavnim likovima koji su marketirani kao maskote, tj. likovi koji su jako prepoznatljivi i koji su često bili korišteni za prodaju igara mladoj djeci [3]. *Hop and bop* podžanr dobiva svoje ime od načina na koji igrač uništava neprijatelje, tako da skače na njih. Prva igra u ovom podžanru je *Donkey Kong* (Slika 3.), a neki od najpopularnijih primjera su *Super Mario Bros*, *Sonic the Hedgehog* i *Crash Bandicoot*.



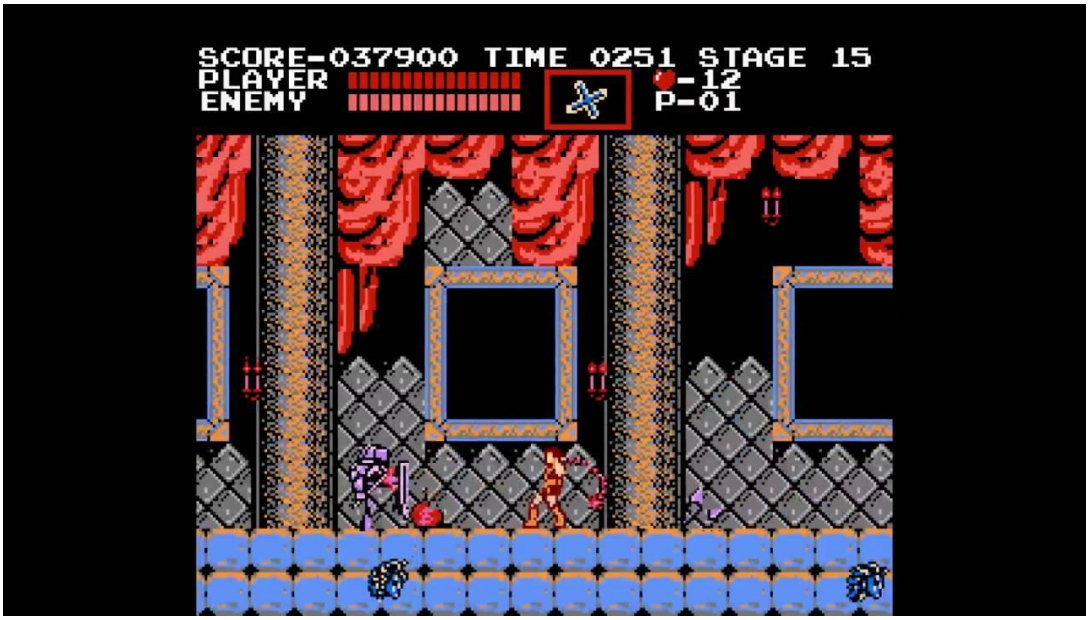
Slika 3 Isječak iz Donkey Kong igre

S izlaskom igre *Jump Bug* 1981. godine došlo je do stvaranja novog podžanra u platformerima *run and gun*. *Jump Bug* video igra je pripadala tipu platformera s pomičnom kamerom i sastojala se od većih razina s mogućnošću pucanja prema nadolazećim neprijateljima. Ovaj podžanr je kasnije popularizirani s video igrama *Contra* i *Metal Slug*. Zbog igara koje su popularizirale ovaj podžanr jedna od njegovih prepoznatljivijih značajki je postala velika težina prelaska video igre.

Još jedan od najstarijih podžanrova platformerskih video igara su igre koje spajaju elemente *puzzle* igara i platformera. Zbog spajanja tih elemenata ovaj podžanra je dobio naziv *puzzle platformer*. Izazov u ovom tipu platformera ne proizlazi iz neprijatelja ili prepreka nego iz rješavanja *puzzla* kako bi igrač mogao napredovati. Primjeri ovog podžanra su starije igre kao *Wario Land II*, *Oddworld: Abe's Oddysee*, te novije video igre kao *Inside* i *Fez* (Slika 4.).

Slika 4 Isječak iz Fez video igre

U 1985. i 1986. popularnost je stekao novi podžanr. Glavno obilježje ovog podžanra je njegov veliki otvoreni svijet koji je uobičajeno podijeljen na više dijelova kojima igrač ne može pristupiti dok ne dobije nove vještine ili alate. Za razliku od igara koje su podijeljene na razine, u igrama s otvorenim svijetom cijeli prostor koji bi igrač mogao istraživati je spojen i za prijelaz iz jednog mjesta u drugo ne treba prolaziti kroz ekrane za učitavanje. Ovaj podžanr je dobio ime *Metroidvania* po igrama *Metroid* i *Castlevania* (Slika 5.) koje su najpopularnije igre u ovom podžanru.



Slika 5 Isječak iz Castlevania video igre

Zadnji podžanr koji je nastao u 1980-ima su *cinematic* platformeri. Ovi platformeri koriste rotoskopiranje[4] za izradu animacija unutar video igre. Ova tehnika animacije zahtjeva precrtavanje preko snimke uživo kadar po kadar kako bi se dobila animacija što sličnija stvarnom životu. Prva igra koja je koristila rotoskopiranje i koja je ujedno i prva igra koje pripada ovom podžanru je *Prince of Persia* koja je izašla 1989. godine.

3 VIDEO IGRA KING'S RETURN

Video igra King's Return je platformerska video igra inspirirana video igrama *Mega man* i *Shovel Knight*. Cilj ovog završnog rada je bila izrada prototipa koji se sastoji od jedne razine i glavnog izbornika. Ova video igra pripada platformerima s pomičnom kamerom i dijeli elemente sa *run and gun* i *hop and bop* podžanrom.

Cilj igre King's Return je izazvati igrača s razinom naseljenom raznim neprijateljima te borbom protiv *boss* tipa neprijatelja na kraju razine. Prototip igre se sastoji od 3 normalna neprijatelja i jednog *boss* neprijatelja.

3.1 Programska izvedba video igre

Prototip video igre King's Return je razvijen u Unity[1] programskom okruženju za razvoj video igara. Prilikom razvoja prototipa korištena je 2020.3.25f1 verzija Unity programskog okruženja kao i dodatni Unity paketi *TextMeshPro* i *Cinemachine*. Unity ima mogućnost razvoja video igara za više platforma, ali ovaj prototip je razvijen samo za Windows osobna računala.

Ovaj prototip se sastoji od dvije scene:

- **MainMenu** – MeinMenu scena sadrži glavni izbornik video igre i izbornik razina;
- **CloverStage** – CloverStage scena sadrži jedinu razinu unutar ovog prototipa, za pristup ovoj sceni igrač je mora izabrati iz izbornika razina u MainMenu sceni.

Za dodavanje funkcionalnosti pojedinim elementima prototipa koristile su se C# skripte kao i drugi osnovni elementi Unity programskog okruženja. Najčešće korišteni osnovni elementi su *BoxCollider2D* i *RigidBody2D.*

BoxCollider2D[5] element služi za pojednostavljenja provjere prilikom dodira dvaju ili više objekata unutar video igre (npr. dodir glavnog lika s neprijateljima, sudar lika sa zidom ili tlom). Ovaj element u zajednici s Unity oznakama se može koristiti i za otkrivanje je li neki napad pogodio svoju namjenjenu metu. *RigidBody2D* element postavlja objekt pod kontrolom *Unity physics enginea* što znači da fiziku video igre ne treba pisati od nule.

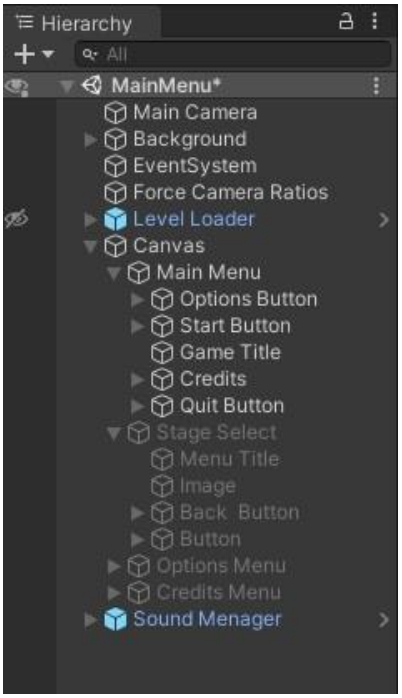
C# skripte koje su napravljene za ovaj prototip se mogu podijeliti na:

[1] Unity, https://store.unity.com

- skripte za kontrolu lika igrača;
- skripte za normalne i *boss* neprijatelje;
- skripte za kontrolu kamere;
- skripte za kontrolu promjene scena.

3.2 MainMenu scena

MainMenu scena u sebi sadrži *Canvas* i *Background* elemente. *Background* element služi da uz pomoć *Unity Tile Map*-a lako možemo napraviti pozadinu za naše izbornike. *Canvas* element je onaj koji sadrži glavno tijelo izbornika, unutar njega se nalaze još dva *canvas* elementa koji se zovu *Main Menu*, *Stage Select*, *Options menu* i *Credits* oni sadrže sve gumbe i tekst potrebne za glavni izbornik, izbornik razina, kontrolu volumena i izvore muzike respektivno (Slika 6.).

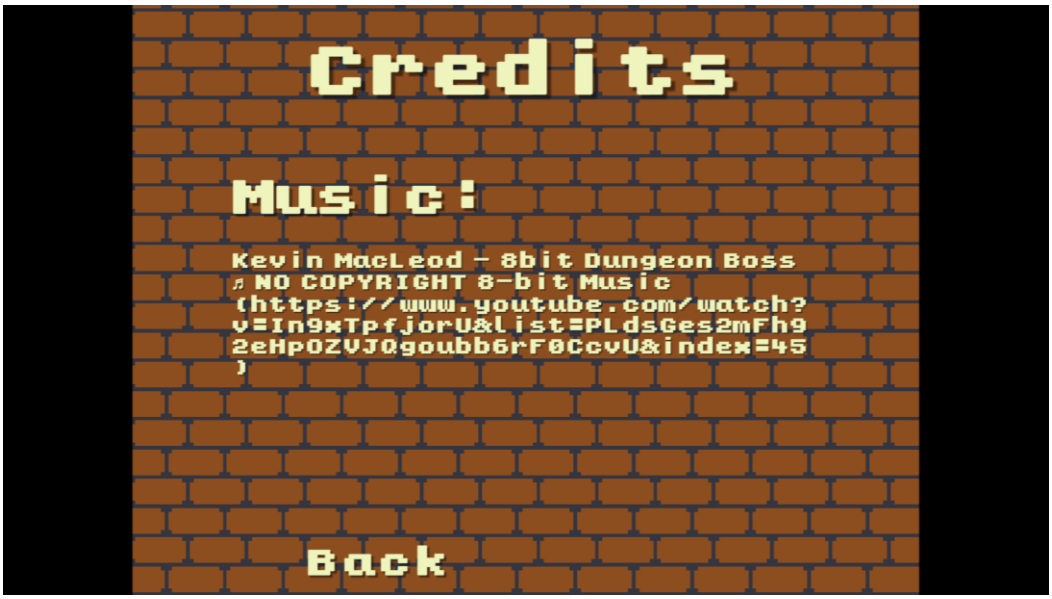


Slika 6 Hijerarhija MainMenu Scene

Glavni izbornik se sastoji od naslova video igre koji se nalazi na vrhu zaslona i ispod kojega se nalaze gumbi *Start*, *Options* i *Quit*, a na desnom donjem kutu zaslona se nalazi *Credits* gumb (Slika 7.). Prilikom pritiska na gumb Start zatvara se glavni izbornik, a otvara izbornik razine. Ova funkcionalnost se ostvarila uz pomoć Unity elementa *Button* gdje se na *Start* gumb dodao *OnClick* događaj u kojem se *Main Menu canvas* postavlja kao neaktivan a *Stage Select canvas* kao aktivan. Na isti način kao i *Start* gumb *Options* gumb otvara izbornik za kontrolu volumena Pritiskom na gumb *Quit* igrač izlazi iz video igre. Ova funkcionalnost se ostvarila uz pomoć

MainMenu C# skripte koja je napisan posebno za ovaj projekt. Zadnji gumb unutar *Main Menu canvas*-a je *Credits* gumb, a njegovim pritiskom korisnik može vidjeti izvor glazbe korištene u projektu (Slika 8.).

Slika 7 Glavni izbornik

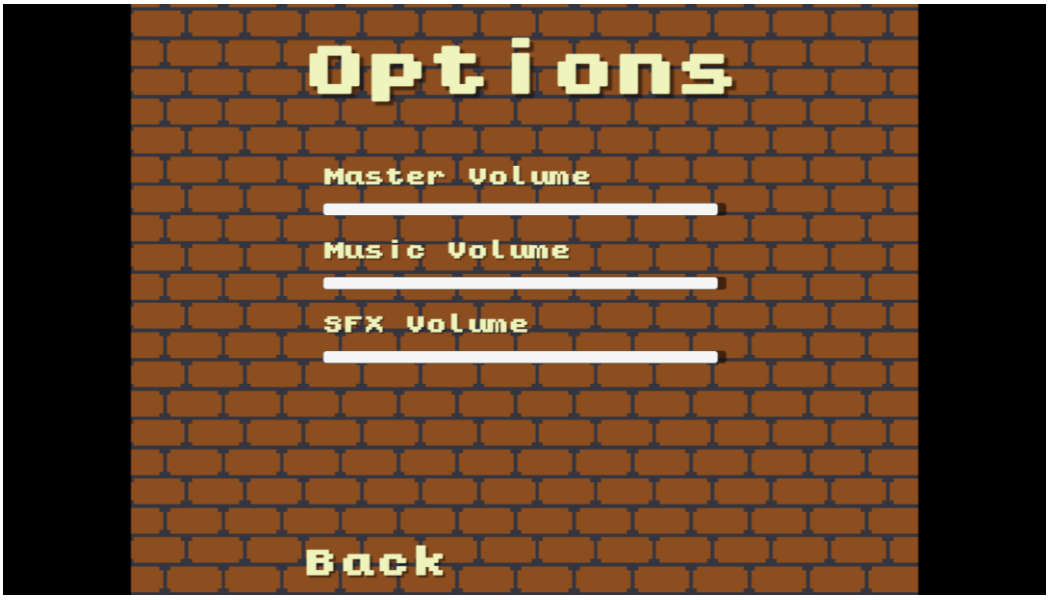


Slika 8 Izvor Glazbe

Izbornik razina se sastoji od naziva izbornika na vrhu zaslona. Ispod naziva se nalaze *sprite*ovi lika igrača i *boss*a prve razine, a ispod se nalazi njegov naziv (Slika 9.). Na lijevom dnu zaslona se nalazi *Back* gumb koji ima suprotnu funkcionalnost *Start* gumba, njegovim klikom *Stage Select canvas* postaje neaktivan a *Main Menu canvas* postaje aktivan. Klikom na *sprite boss* neprijatelja uz pomoć MainMenu C# skripte igra učitava prvu razinu prototipa.

Slika 9 Izbornik razina

Izbornik za kontrolu volumena se sastoji od tri klizača (Slika 10.). Prvi klizač kontrolira razinu zvuka u cijeloj igri. Drugi klizač kontrolira razinu pozadinske glazbe a zadnji klizač kontrolira razinu zvukova koji se puštaju tijekom kontrole lika ili prijelaza između izbornika.



Slika 10 Izbornik za kontrolu volumena

3.3 CloverStage scena

Unutar CloverStage scene se nalaze tri *Tile Map*a. Prvi *Tile Map Background* prikazuje pozadinu razine, drugi *Tile Map Platforms* prikazuje dio razine na kojem se igračev lik kreće dok zadnji *Tile Map Decorations* služi za prikaz dekoracija koje se nalaze u prvom planu.

Canvas elementi unutar CloverStage scene služi za prikaz broja napada koje igrač i *boss* mogu preživjeti i za prikaz izbornika pauze.

CloverStage scena također sadrži sve obične neprijatelje i glavnog, takozvanog *boss* neprijatelja, koji se javljaju unutar razine. U sceni se nalaze *Game Objecti* koji opisuje rute patrole za neprijatelje tipa vitez. Kako bi *boss* neprijatelj mogao ispravno funkcionirati ova scena također sadrži dodatne elemente koji *boss*u javljaju gdje se njegova arena nalazi te gdje se stvaraju projektili za određene napade. Zadnje elemente koje CloverStage scena sadrži su *collider*-i koji sprječavaju kameru da izađe izvan okvira razine (Slika 11.).

Slika 11 Prikaz CloverStage scene unutar Unity uređivača

4 LIK IGRAČA

Lik kojeg igrač kontrolira unutar video igre King's Return sastoji se od glavnog objekta koji predstavlja glavno tijelo lika te koji sadrži *BoxCollider2D*, *RigidBody2D* i sve osim jedne C# skripte koje se odnosu na lika igrača. Djeca glavnog objekta su objekti: *FeetPos*, *Particle System*, *Player Hands* i *Knockback*.

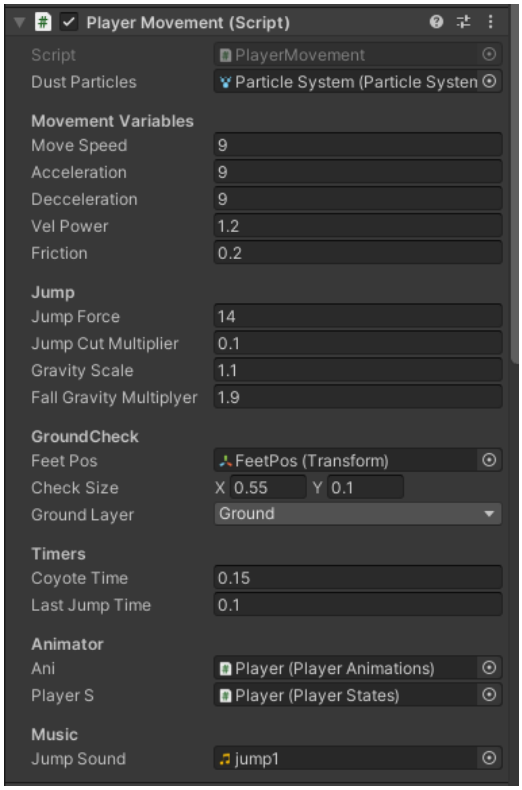
Player Hands objekt predstavlja ruke i/ili oružja koja igrač trenutno koristi. Ovaj objekt također sadrži i zadnju C# skriptu koja se odnosi na lik igrača.

Skripte koje se odnose na lik igrača su:

- PlayerMovement;
- PlayerAnimations;
- PlayerStates;
- PlayerDamage;
- PlayerHealthBar;
- PlayerMelee;
- ColisionWithEnemy;
- PlayerMeleeHitDetection.

4.1 PlayerMovement

PlayerMovement C# skripta je glavna skripta zadužena za omogućavanje kretanja lika igrača. Ova skripta omogućuje laku promjenu svih varijabli vezanih uz kretanje lika unutar 2D platformera kao što su: brzina kretanja, ubrzanje, usporenje, visina skoka, itd. (Slika 12.). U skripti se koriste metode *Start*, *Update* i *FixedUpdate* iz *UnityEngine* knjižnice. Metoda *Start* se poziva prije prvog poziva metode *Update*, a ona se poziva tijekom svakog okvira (engl. *frame*). Za razliku od metode *Update* metoda *FixedUpdate* se uvijek poziva 50 puta u sekundi bez obzira na brzinu osvježavanja (engl. *frame rate*) video igre. Unutar ove metode izvodimo sve operacije koje se dodiruju sa Unity physics engineom.



Slika 12 PlayerMovement skripta unutar Unity uređivača

```
void Start()
    {
        playerRB = GetComponent<Rigidbody2D>();
    }
```

Unutar *Start* metode varijabli *playerRB* pridružujemo *RigidBody2D* element koji se nalazi na istom objektu.

```
    moveInput = Input.GetAxisRaw(„Horizontal“);
        if ( moveInput  *  transform.localScale.x  <  0){
     transform.localScale = new Vector3(transform.localScale.x * -1,
transform.localScale.y,      transform.localScale.z);
        }
```

Unutar metode *Update* odvijamo više radnji prva od kojih je dobivanje smjera horizontalnog kretanja lika igrača pomoću metode *Input.GetAxisRaw()* koja vraća:

- 1 ako se igrač kreće desno;
- 0 ako nema unosa od strane igrača;
- -1 ako se igrač kreće lijevo.

Nakon što je smjer kretanja dobiven, skripta provjerava je li se igrač kreće u istom smjeru u kojem lik gleda. To provjeravamo tako što pomnožimo smjer željenog kretanja i trenutačni smjer u kojem lik gleda. Ako je rezultat manji od 0, to znači da moramo promijeniti orijentaciju lika unutar video igre.

```
If (Input.GetKeyDown(KeyCode.Space)){
            lastJump = lastJumpTime;
        }
```

Drugi dio *Update* metode se fokusira na vertikalno kretanje igrača, tj. skakanje. Skripta prvo provjerava je li igrač stisnuo gumb za skok, u ovom slučaju gumb za razmak, te ako je taj gumb stisnut postavlja brojač *lastJump* na vrijednost koju smo odredili unutar Unity uređivača.

```
If(lastGrounded > 0 && Mathf.Abs(playerRB.velocity.x) > .01f &&
playerS.currentState != PlayerStates.States.Hurt)
        {
            playerS.ChangeState(PlayerStates.States.Walk);
        }
        else if (playerS.currentState != PlayerStates.States.Hurt)
        {
            playerS.ChangeState(PlayerStates.States.Idle);
        }
```

Zadnji dio *Update* metode postavlja stanje igrača unutar *PlayerState* C# skripte. Ako se igrač kreće i nije već u *Hurt* stanju tada se postavlja u *Walk* stanje, inače se postavlja u *Idle* stanje.

Unutar *FixedUpdate* metode skripta prvo poziva *IsGrounded*() metodu.

```
Public void IsGrounded (){
        if (Physics2D.OverlapBox(feetPos.position, checkSize, 0,
groundLayer))
        {
       lastGrounded = coyoteTime;
            isJumping = false;
        }
    }
```

Ova metoda uz pomoć *Physics2D.OverlapBox* metode provjerava nalazi li se lik igrača na tlu.

Ako se lik nalazi na tlu metoda tada postavlja *lastGrounded* brojač vremena (engl. *timer*) na otprije zadanu vrijednost i ostatku skripte govori da glavni lik nije trenutno u sredini skoka.

```
Float targetSpeed = moveInput * moveSpeed;

float speedDif = targetSpeed – playerRB.velocity.x;

float accelRate = (Mathf.Abs(targetSpeed) > .01f) ? acceleration :
decceleration;

float movement = Mathf.Pow(Mathf.Abs(speedDif) * accelRate, velPower)
* Mathf.Sign(speedDif);

playerRB.AddForce(movement * Vector2.right);
```

Dio skripte koji je zadužen za izvođenje željenog kretanja igrača se izvodi samo ako lik već nije u *Hurt* stanju. Skripta prvo izračunava željenu brzinu i razliku između trenutne i željene brzine. Željena brzina je umnožak smjera kretanja i brzine kretanja koja je određena unutar Unity uređivač. Nakon toga ovisno o željenoj brzini varijabli *accelRate* pridružujemo vrijednost ubrzanja ako je apsolutna željena brzina veća od 0.1 ili vrijednost usporavanja ako je manja od tog broja. Na kraju ovog dijela skripte po gore navedenoj formuli se računa silu koja će biti primijenjena na lika unutar video igre.

```
If (lastGrounded > 0 && Mathf.Abs(moveInput) < .01f)
            {
                float amount =
Mathf.Min(Mathf.Abs(playerRB.velocity.x), Mathf.Abs(friction));

                amount *= Mathf.Sign(playerRB.velocity.x);

                playerRB.AddForce(Vector2.right * -amount,
ForceMode2D.Impulse);
            }
```

Ovaj dio skripte se samo aktivira ako se igrač ne kreće, a nije u zraku. Tada nad likom primjenjujemo silu koja u suprotnom smjeru kretanja lika i amplitude trenutne brzine ili vrijednosti trenja, koja je unesena unutar Unity uređivača, ovisno o tome koja je manja.

```
If (lastGrounded > 0 && lastJump > 0 && !isJumping)
        {
```

```
            Jump();
        }

if (!jumpInputRelesed && !Input.GetKey(KeyCode.Space))
        {
        SmallJump();
        }
```

Ovaj dio skripte provjerava jesu li brojači vremena *lastGrounded* i *lastJump* veći od nule te da igrač već ne skače i nije u *Hurt* stanju. Ako su svi ovi uvjeti zadovoljeni poziva se metoda *Jump*.

```
Private void Jump(){
        createDust();
        playerRB.AddForce(Vector2.up * jumpForce,
ForceMode2D.Impulse);
        lastGrounded = 0;
        lastJump = 0;
        isJumping = true;
        jumpInputRelesed = false;
    }
```

Metoda Jump prvo poziva metodu *createDust*() koja uz pomoć objekta *Particle System* stvara čestice koje prate lika u skoku. Nakon toga dodaje impulsnu silu u gornjem smjeru u vrijednosti koja je zadana unutar Unity uređivača. Na kraju metode postavljamo brojač *lastGrounded* i *lastJump* na 0 te ostatku skripte dajemo do znanja da igrač trenutno skače i da još uvijek drži pritisnutim gumb za skok.

Nakon izvedbe skoka ako igrač otpusti gumb za skok skripta poziva *SmallJump*() metodu.

```
Private void SmallJump(){
        if (playerRB.velocity.y > 0 && isJumping){
            playerRB.AddForce(Vector2.down * playerRB.velocity.y * (1
- jumpCutMultiplier), ForceMode2D.Impulse);
        }

        jumpInputRelesed = true;
        lastJump = 0;
    }
```

Ova metoda smanjuje visinu igračevog skoka tako da na lik primjenjuje silu u donjem smjeru

u unaprijed zadanom iznosu.

```
If (playerRB.velocity.y < 0)
        {
            playerRB.gravityScale = gravityScale *
fallGravityMultiplyer;
        }
        else
        {
            playerRB.gravityScale = gravityScale;
        }
```

Ovo je zadnji dio *FixedUpdate* metode i on povećava gravitaciju ako igrač pada ili je postavlja na normalnu ako skače.

4.2 PlayerAnimations

PlayerAnimations C# skripta omogućuje lakše povezivanje između Unity animatora i ostalih C# skripti vezanih uz lika igrača. To omogućava uz pomoć javnih metoda *StartAnimationPlayer*, *StartAnimationHands* i *ChangeHandsController*.

```
Public void StartAnimationPlayer (string aniName){
        playerA.Play(aniName);
    }

public void StartAnimationHands (string aniName){
        handsA.Play(aniName);
    }

public void ChangeHandsController(AnimatorOverrideController
newController)
    {
        handsA.runtimeAnimatorController = newController;
    }
```

4.3 PlayerStates

PlayerStates C# skripta prati u kojem se trenutno stanju nalazi lik igrača i s obzirom na trenutno stanje uz pomoć *PlayerAnimations* skripte pusti ispravnu animaciju.

```
Void Update()
    {
        switch (currentState)
        {
            case States.Idle:
                playerAni.StartAnimationPlayer(„Idle“);
                break;
            case States.Walk:
                playerAni.StartAnimationPlayer(„Walk“);
                break;
            case States.Hurt:
                playerAni.StartAnimationPlayer(„Hurt“);
                playerAni.StartAnimationHands(„Hands_Hurt“);
                break;
            default:
                break;

        }
    }
```

4.4 PlayerDamage

PlayerDamage C# skripta služi za kontroliranje broja udaraca koje igrač može primiti prije nego što umre, tj. HP-a (engl. *Health Power*) mu padne na nulu. Unutar nje se nalaze metode za smanjenje i povećanje trenutnog HP-a i za stvaranje povratnog udarca prilikom smanjenja HP-a.

```
public void TakeDamage (int dmg)
    {
        if (currentInvulnerableTime > 0)
        {
            return;
        }

        currentHealth = currentHealth  - dmg;
        currentInvulnerableTime = invulnerableTime;


        playerS.ChangeState(PlayerStates.States.Hurt);
        playerHB.UpdateHealthBar(currentHealth);

        if (currentHealth <= 0)
        {
            SceneManager.LoadScene("MainMenu");
```

```
            return;
        }

        Knockback(knockbackDir);

    }
```

TakeDamage metoda prvo provjerava vrijednost *currentInvulnerableTime* brojača i ako ona nije veća od nule, izlazi iz metoda. U nastavku metoda smanjuje igračev trenutni HP i postavlja brojač neranjivosti na predodređenu vrijednost te uz pomoć *PlayerState* i *PlayerHealthBar* skripti postavlja stanje igrača u Hurt i ažurira prikaz igračevog života na zaslonu respektivno. Nakon toga ako je HP igrača na nuli ili ispod Skripta ga vraća na *MainMenu* scenu. Na kraju metode poziva se knockback metoda u kojoj kao argument predajemo položaj *Knockback Game Objecta*.

```
public void Knockback(Transform hitPos)
    {
        Vector2 knockbackDir = new Vector2((transform.position.x -
hitPos.position.x), 0);
        rb.velocity = new Vector2(knockbackDir.x /
Mathf.Abs(knockbackDir.x) * knockback, knockup);
    }
```

Metoda *Knockback* odbacuje igrača unazad ovisno o vrijednosti koja je unaprijed odabrana.

```
public void Heal(int hp)
    {
        currentHealth += hp;
        if (currentHealth > maxHealth)
        {
            currentHealth = maxHealth;
        }
        playerHB.UpdateHealthBar(currentHealth);
    }
```

Metoda *Heal* povećava trenutni HP igrača i kontrolira njegov maksimalni iznos kako ne bi došlo do prekoračenja.

4.5 PlayerHealthBar

PlayerHealthBar C# skripta povezuje skriptu *PlayerDemage* i prikaz HP igrača na zaslonu. Ona sadrži samo jednu metodu *UpdateHealthBar* koja ažurira prikaz HP-a na zaslonu.

```
public void UpdateHealthBar (int health)
  {
      int nCrowns = health / 2;
      int hasHalf = health % 2;

      for (int i = 0; i < healthBar.Length; i++)
      {
          if ( i <= nCrowns - 1)
          {
              healthBar[i].sprite = healthSprites[0];
          }

          if (i > nCrowns - 1)
          {
              healthBar[i].sprite = healthSprites[1];
          }

          if ( i == nCrowns && hasHalf == 1)
          {
              healthBar[i].sprite = healthSprites[2];
          }

      }
  }
```

4.6 PlayerMelee

PlayerMelee C# skripta je zadužena za kontroliranje igračevog napada protiv neprijatelja.

```
if (lastAtk > 0){
          lastAtk -= Time.deltaTime;
      }

      if (Input.GetButtonDown("Fire1") && lastAtk <= 0){
          ani.StartAnimationHands("Hands_Punch");
          lastAtk = atkSpeed;    }
```

Unutar *Update* metode prvo se smanjuje *lastAtk* brojač ako je veći od nule, zatim skripta gleda je li je igrač pritisnuo gumb za napad, koji je zadan kao lijevi klik miša. Ako je gumb stisnut i brojač *lasAtk* ima vrijednost 0 ili manju, uz pomoć *PlayerAnimation* skripte započinje animaciju za napad u *Player Hands Game Object-u*.

4.7 ColisionWithEnemy

ColisionWithEnemy C# skripta provjerava je li lik igrača došao u direktan kontakt s neprijateljima. Ako se kontakt dogodio, tada pozivamo metodu *TakeDamage* iz *PlayerDamage* skripte.

```
 void OnCollisionEnter2D(Collision2D coll)
    {
        if (coll.gameObject.CompareTag("Enemy") ||
coll.gameObject.CompareTag("Boss"))
        {
            GetComponent<PlayerDamage>().TakeDamage(colisionDamage);
        }
    }
```

4.8 PlayerMeleeHitDetection

PlayerMeleeHitDetection C# skripta je zadužena za provjeru je li igračev napad pogodio neprijatelja i ako je neprijatelj pogođen poziva se metode za smanjenje HP iz C# skripta.

```
void OnTriggerEnter2D(Collider2D col)
    {

        if (col.CompareTag("Enemy") || col.CompareTag("Boss"))
        {
            col.GetComponent<IDamage>().TakeDamage(atkDmg);
            col.GetComponent<IDamage>().Knockback(transform);
        }
    }
```

Ova skripta sadrži metode za aktiviranje i deaktiviranje polja za registraciju napada. Ove metode se uz pomoć Unity animation događaja pozivaju unutar animacija za napad.

```
public void EnableHitBox (){
       hitBox.enabled = true;
    }

    public void DisableHitBox (){
        hitBox.enabled = false;
    }
```

5 NEPRIJATELJI

U video igri King's Return se nalaze tri vrste normalnih neprijatelja i *boss* neprijatelj. Tri vrste normalnih neprijatelja su: Vitez, Čarobnjak i Dvorska luda. Svi neprijatelji uključujući i *boss* neprijatelja koriste tri iste C# skripte:

- EnemyAnimator;
- EnemyDamage;
- EnemyStates.

EnemyAnimator skripta ima iste funkcionalnosti kao i *PlayerAnimator* skripta, tj. povezuje ostatak skripta s animatorom lika i njegovih ruku.
EnemyStates skripta je zadužena za čuvanje informacija unutar kojeg stanja se nalazi neprijatelj trenutno. Postoji pet mogućih stanja:

- Idle;
- Walk;
- Hurt;
- Jump.

Ovisno u kojem se stanju nalazi neprijatelj *EnemyStates* skripta uz pomoć *EnemyAnimator* skripte igra odgovarajuće animacija. Nazivi animacija za određene neprijatelje se nalaze unutar polja za svako stanje neprijatelja, dodatne animacije ili promjena njihovog naziva se mogu izvoditi unutar Unity uređivača.

```
[Header("Animations")]
    [SerializeField] public string[] idleAnimations = { "GK_Idle",
"GM_Idle", "Jester_Idle", "Boss_Idle" };
    [SerializeField] public string[] idleAnimationsHands = {
"GK_Sword_Idle", "", "JesterHands_Idle", "BossHands_Idle" };
    [SerializeField] public string[] walkAnimations = { "GK_Walk",
"", "", "" };
    [SerializeField] public string[] walkAnimationsHands = { "", "",
"", "" };
    [SerializeField] public string[] hurtAnimations = { "GK_Hurt",
"GM_Hurt", "Jester_Hurt", "Boss_Hurt" };
    [SerializeField] public string[] hurtAnimationsHands = {
"GK_Sword_Hurt", "", "JesterHands_Hurt", "" };
    [SerializeField] public string[] jumpAnimations = { "", "",
```

```
"Jester_Jump", "" };
    [SerializeField] public string[] jumpAnimationsHands = { "", "",
"JesterHands_Jump", "" };
```

Za izbor animacije koje odgovaraju željenom neprijatelju koristi se *EnemyType enum* koji kad ga pretvorimo u *int* odgovara ispravnim animacijama unutar svakog polja. Izbor animacija se izvodi unutar *Update* metode pomoću *switch* izjave.

EnemyDamage skripta je zadužena za praćenje HP-a pojedinih neprijatelja. Većina skripte ima istu funkcionalnost kao i *PlayerDamage* skripta ali se razlike nalaze unutar *Knockback* metode i u tome kako skripta tretira smrt lika. Za neprijateljev povratni udarac metoda se poziva zasebno i kao argument uzima trenutnu poziciju igračevog lika, dok je ostatak metode identičan. Ako HP neprijatelja ikad dosegne nula ili manje stvara se *GameObject* s animacijom eksplozije na njihovoj lokaciji smrti te ako je neprijatelj *boss* uz eksploziju stvara se i portal koji igrača vodi natrag na glavni izbornik.

5.1 Vitez

Vitez ima posebnu skriptu *EnemyAI* koja je zadužena za njegovo ponašanje unutar igre. Vitez će patrolirati unutar određenog područja dok mu se igrač ne približi, tada će postati agresivan i početi pratiti igrače te ako igrač dođe unutar njegovog područja napada vitez će ga napasti. Za određivanje područja patroliranja koriste se zidovi, krajevi platformi i posebni objekti unutar igre koji su klasificirani kao patrolne točke.

Za izbor načina ponašanja između agresivnog ili patrolnog koristimo *raycast*. *Raycast* pušta zraku koja se unutar igre sudara sa svim objektima koji imaju *collider*-e te ih javlja skripti. U *EnemyAI* skripti neprijatelj pušta dvije zrake jednu ispred, a jednu iza sebe te ako bilo koja zraka detektira lik igrača mijenja neprijateljevo ponašanje u agresivno. Ako niti jedna zraka ne detektira igrača, a neprijatelj se nalazi u agresivnom stanju početi će brojač koji kada dođe do kraja vraća neprijatelja u patrolno stanje.

```
 RaycastHit2D hitFront = Physics2D.Raycast(rayStartPoint.position,
new Vector2(transform.localScale.x, 0), aggroRange, playerMask);
        RaycastHit2D hitBack =
Physics2D.Raycast(rayStartPoint.position, new Vector2(-
transform.localScale.x, 0), aggroRange, playerMask);

if (hitFront.collider != null)
```

```
        {
            currentBehaviour = Behaviour.Aggro;
            currentDeAggroTimer = deAggroTimer;
            direction = transform.localScale.x;
        }

        if (hitBack.collider != null)
        {
            currentBehaviour = Behaviour.Aggro;
            currentDeAggroTimer = deAggroTimer;
            transform.localScale = new Vector3(transform.localScale.x
* -1, transform.localScale.y, transform.localScale.z);
            direction = -transform.localScale.x;
        }

        if (hitBack.collider == null && hitFront.collider == null)
        {
            if (currentDeAggroTimer <= 0)
            {
                currentBehaviour = Behaviour.Patrol;
            }
            else
            {
                currentDeAggroTimer -= Time.deltaTime;
            }
        }
```

Ako se neprijatelj nalazi u patrolnom stanju on će šetati u jednom smjeru dok ne naiđe na zid, kraj platforme ili objekt unutar igre koji je klasificiran kao patrolna točka. Tada neprijatelj mijena smjer i ponavlja ovo ponašanje. Za detekciju kraja patrole koristimo *Physics2D.OverlapCircle* koji provjerava je li se objekti određenih kvalifikacija nalaze unutar određenog područja. Ove provjere se odrađuju unutar *FixedUpdate* metode.

```
seesGround = Physics2D.OverlapCircle(groundCheck.position,
checkRadius, groundMask);
seesWall = Physics2D.OverlapCircle(wallCheck.position, checkRadius,
groundMask);
seesPatrol = Physics2D.OverlapCircle(wallCheck.position, checkRadius,
patrolMask);
```

Unutar posebne metode *PatrolBehaviour* se provjerava istinitost ovih provjera te po potrebi mijenja smjer neprijatelja.

```
private void PatrolBehaviour()
    {
        if (!seesGround || seesWall || seesPatrol)
        {
            direction *= -1;
            transform.localScale = new Vector3(transform.localScale.x
* -1, transform.localScale.y, transform.localScale.z);
        }
    }
```

Kad neprijatelj slijedi agresivno ponašanje tada pomoću *raycast*-a ispred neprijatelja skripta provjerava za lik igrača i ako je dovoljno blizu izvršava napad.

```
RaycastHit2D hitFront = Physics2D.Raycast(rayStartPoint.position, new
Vector2(transform.localScale.x, 0), atkRange, playerMask);

        if (hitFront.collider != null)
        {
            if (atkTimer <= 0)
            {
                enemyS.ChangeState(EnemyStates.States.Attack);
                enemyAni.StartAnimationHands("GK_Slash");
                atkTimer = atkSpeed;
            }

            direction *= 0;
        }
```

5.2 Čarobnjak

Za kontroliranje ponašanja neprijatelja tipa čarobnjak u projektu se koristi C# skripta *MageEnemyAI*. Čarobnjak će stajati nepokretno na jednom mjestu dok lik igrača ne dođe dovoljno blizu da ga neprijatelj napadne. Provjera je li igrač ušao u čarobnjakov prostor se izvršava pomoću *Physics2D.OverlapBox* metode, koja kao i *Physics2D.OverlapCircle* metoda obavještava skriptu ako se objekt određene kategorije nalazi unutar nekog područja. Područje u kojem skripta traži lik igrača se može podesiti unutar Unity uređivača.

```
  seesPlayer = Physics2D.OverlapBox(transform.position, atkRange, 0,
playerLayer);
```

Kada se igrač nalazi unutar prostora za napad neprijatelja skripta prvo provjerava je li je prošlo dovoljno vremena od zadnjeg napada ili ako je igrač tek ušao u prostor provjerava je li prošlo vrijeme kašnjenja prvog napada. Ako su oba uvjeta zadovoljena skripta stvara novi *Game Object* koji se kreće u ravnoj liniji kroz platforme prema lokaciji lika igrača u trenutku početka napada. Ako igrač izađe izvan područja napada tada se resetira vrijeme kašnjenja prvog napada i čarobnjak čeka ponovni ulazak igrača u područje napada.

```
   if (seesPlayer)
        {
            Vector3 playerPosition = new Vector2
(playerCollider.transform.position.x,
playerCollider.transform.position.y) +
playerCollider.GetComponent<BoxCollider2D>().offset;

            Vector3 projectileDirection = playerPosition -
projectileSpawnPoint.position;

            if (projectileDirection.x * transform.localScale.x < 0)
            {
                transform.localScale = new
Vector3(transform.localScale.x * -1, transform.localScale.y,
transform.localScale.z);
            }
            if (firstShotTimer < firstShotDelay)
            {
                firstShotTimer += Time.deltaTime;
            }


            if (currentTime <= 0 && firstShotTimer >= firstShotDelay)
            {
                GameObject tempProjectile = Instantiate(projectile,
projectileSpawnPoint.position, projectileSpawnPoint.rotation);
                float angle = Mathf.Atan2(playerPosition.y -
tempProjectile.transform.position.y, playerPosition.x -
tempProjectile.transform.position.x) * Mathf.Rad2Deg;
                tempProjectile.transform.rotation =
Quaternion.Euler(new Vector3(0, 0, angle));
                currentTime = atkDelay;
            }
        }
        else
        {
            if (firstShotTimer > 0)
            {
```

```
                firstShotTimer = 0;
            }
        }
```

5.3 Dvorska luda

C# skripta *JesterAI* služi za kontrolu ponašanja neprijatelja tipa dvorske lude. Ponašanje dvorske lude je slično neprijatelju čarobnjaka, samo umjesto lansiranja projektila protiv lika igrača dvorska luda pokušava skočiti na igrača ako on uđe unutar njegovog prostora za napad. Za razliku od Čarobnjaka provjera ovog prostora se izvršava pomoću *Physics2D.OverlapCircle* metode kojoj je radijus podesiv u Unity uređivač.

```
 seesPlayer = Physics2D.OverlapCircle(transform.position, atkRange,
0, playerMask);
```

Ako se igrač nalazi unutar područja za napad skripta provjerava nalazi li se dvorska luda u kontaktu sa tlom i je li prošlo dovoljno vremena od zadnjeg napada. Ako su oba uvjeta zadovoljena tada se izvršava napad. Za provjeru je li neprijatelj u kontaktu s tlom koristimo istu metodu kao i kod lika igrača. Za izvršavanje skoka koristimo *AddForce* metodu koja je dio *RigidBody* klase. Ova metoda izvršava silu nad objektom u zadanom smjeru.

```
  if (seesPlayer && isGrounded && atkTimer <= 0 &&
enemyS.currentState != EnemyStates.States.Hurt)
        {
            JumpAttack();
            atkTimer = atkSpeed;
        }
```

```
  private void JumpAttack()
    {
        float playerDistance = playerPos.position.x -
transform.position.x;

        enemyRB.AddForce(new Vector2(playerDistance, jumpHeight),
ForceMode2D.Impulse);
    }
```

5.4 Boss neprijatelj

Ponašanje *boss* neprijatelja se sastoji od pet napada koja može izvršiti ovisno o igračevoj udaljenosti od njega. Za odabir napada je zadužena C# skripta *BossAI*. Provjera položaja igrača u odnosu na *boss* neprijatelja se izvršava pomoću tri *Physics2D.OverlapCircle* metode, koje provjeravaju da li se igrač nalazi unutar daleko od *boss*a, unutar srednje udaljenosti ili vrlo blizu *boss*a. Ovisno o udaljenost, *boss* ima nasumičnu šansu da izabere napad. Odabir napada se izvršava u metodi *ChooseAttack*.

```
private void ChooseAttack()
   {
       if (inMaleeRange)
       {
           int randomNum = Random.Range(0, 100);
           if (randomNum <= 90)
           {
               bossAttacks[0].StartAttack();
               currentAtk = 0;
           }
           if (randomNum > 90 && randomNum <= 95)
           {
               bossAttacks[1].StartAttack();
               currentAtk = 1;
           }
           if (randomNum > 95)
           {
               bossAttacks[2].StartAttack();
               currentAtk = 2;
           }
       }
       if (!inMaleeRange && inMidRange)
       {
           int randomNum = Random.Range(0, 100);

           if (randomNum <= 45)
           {
               bossAttacks[1].StartAttack();
               currentAtk = 1;
           }
           if (randomNum > 45 && randomNum <= 90)
           {
               bossAttacks[2].StartAttack();
               currentAtk = 2;
           }
           if (randomNum > 90 && randomNum <= 95)
           {
```

```
                bossAttacks[3].StartAttack();
                currentAtk = 3;
            }
            if (randomNum > 95)
            {
                bossAttacks[4].StartAttack();
                currentAtk = 4;
            }
        }
        if (!inMaleeRange && !inMidRange && inLlongRange)
        {
            int randomNum = Random.Range(0, 100);

            if (randomNum <= 30)
            {
                bossAttacks[3].StartAttack();
                currentAtk = 3;
            }
            if (randomNum > 30)
            {
                bossAttacks[4].StartAttack();
                currentAtk = 4;
            }
        }
    }
```

Pojedini napadi *boss*a su posebne C# skripte koje imaju zajedničko sučelje *IBossAttack* koje definira metode *StartAttack*, *DamageStart* i *EndAttack*. Unutar Start metode u skripti BossAI sve napade koji su pridruženi *boss* neprijatelju stavljamo unutar *bossAttacks* polja iz kojeg kasnije biramo napade.

Najučestaliji napad ako se igrač nalazi u blizini bossa je *Ground Slam* napad. Ovaj napad se ponaša isto kao napad vitez neprijatelja ili lika igrača.

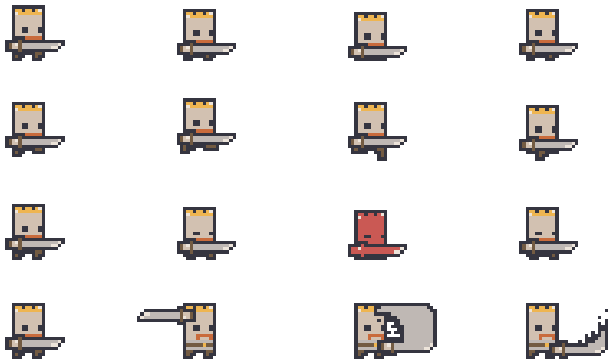
Ako se igrač nalazi na srednjoj udaljenosti od *boss*a, *boss* će najčešće izvršiti napad koji će ga prebaciti na drugi kraj borbene arene, to može biti *Jump* ili *Dash* napad. Logika *Jump* napada je ista kao i kod neprijatelja tipa dvorske lude. Koristeći *Dash* napad boss u vrlo kratkom vremenu pređe cijelu dužinu borbene arene. Ovo ponašanje se dobiva koristeći *Vector3.Lerp* metodu, ova metoda linearno interpolira između dvije točke u prostoru. Pošto ne koristimo *Unity Physics Engine* ova metoda se izvršava unutar *Update* metode.

```
 if (isDashing)
        {
            currentDashTime += Time.deltaTime;
            float perc = Mathf.Clamp01(currentDashTime / dashTime);
            transform.position = Vector3.Lerp(dashStart, dashTarget,
perc);
            if (currentDashTime >= dashTime)
            {
                bossBC.isTrigger = false;
                isDashing = false;
                transform.position = dashTarget;
                currentDashTime = 0;
                EndAttack();
             }
          }
```

Kad se igrač nalazi na velikoj udaljenosti od *boss*-a najčešći napadi su *Call Down* i *Projectile Slam*. Ovi napadi su slični u tome što oboje koriste dodatne *Game Object*e koji stvaraju projektile koji mogu našteti igraču. Napad *Call Down* poziva spore projektile koji sporo padaju s vrha arene i ograničavaju kretanje igrača tijekom sljedećih napada. *Projectile Slam* napad nasuprot stvara brze projektile koji izlaze iz tla i u ravnoj putanji idu prema gore.

6 VIZUALI VIDEO IGRE

Za izradu svih vizualnih elemenata video igre King's Return je korišten besplatni online alat PixilArt[2]. Animacije korištene za igru su animacije ključnih okvira (engl. *key frame*).



Slika 13 Animacije lika igrača

Slika 14 Animacije boss neprijatelja

Slika 15 Animacije običnih neprijatelja

[2] PixilArt, https://www.pixilart.com/draw

7 AUDIO VIDEO IGRE

Zvučni efekti tijekom igranja i u raznim izbornicima su napravljeni pomoću sfxr[3] programa, a pozadinska muzika je "Kevin MacLeod - 8bit Dungeon Boss ♫ NO COPYRIGHT 8-bit Music"[4] preuzeta s YouTube-a.

Uniutar video igre koristimo *SoundMenager Game Object* koji je stvoren samo jednom tijekom pokretanja igre i koji ne uništavamo tijekom prijelaza između scena.

```
private void Awake()
   {
       if (Instance == null)
       {
           Instance = this;
           DontDestroyOnLoad(gameObject);
       }
       else
       {
           Destroy(gameObject);
       }
   }
```

Za puštanje zvuka koristimo 4 izvora zvuka: *musicSource, playerEffectSource, enemyEffectSource, UIEffectSource*. Različite izvori zvuka se koriste kako se zvukovi koji se trebaju puštati u isto vrijeme ili jedan odmah nakon drugog ne bi prekinuli. Za kontrolu razine zvuka koristimo tri metode koje odgovaraju klizačima u izborniku.

[3] Sfxr, https://sfxr.me
[4] Youtube video, https://www.youtube.com/watch?v=In9xTpfjorU

```
public void SetMasterVolume(float value)
  {
      AudioListener.volume = value;
  }
  public void SetMusicVolume(float value)
  {
      musicSource.volume = value;
  }
  public void SetSFXVolume(float value)
  {
      playerEffectSource.volume = value;
      enemyEffectSource.volume = value;
      UIEffectSource.volume = value;
  }
```

S ovim metodama ako dođe do promjene u vrijednosti klizača mijenjamo jačinu određenih izvora zvuka.

8 ZAKLJUČAK

Rezultat ovog završnog rada je uspješno izrađen prototip video igre King's Return. Izrađen prototip je 2D platformer uvelike inspiriran video igrama *Mega Man* i *Shovel Knight*. Za izradu prototipa video igre King's Return korišteno je Unity programsko okruženje, te za izradu vizuala video igre korišten je program PixilArt. Izrađeni prototip čine glavni izbornik i jedna razina. Za nadogradnju prototipa mogu se dodati nove razine kao i novi tipovi neprijatelja. U prototip se mogu dodati elementi za poboljšanje igračkog iskustva kao što su više zvučnih efekata vezanih uz radnje i napade unutar igre kao i dodatne pjesme koje sviraju u pozadini.

9 LITERATURA

[1] Toni Minkkinen, „Basics of Platform Games“, 2016, pristupljeno 18.8.2022. [Online]. Preuzeto s: https://www.theseus.fi/bitstream/handle/10024/119612/Thesis%20-%20Toni%20Minkkinen.pdf?sequence=1

[2] Michael Klappenbach, “What is a Platform Game?” 19.9.2021, pristupljeno 18.8.2022. [Online]. Preuzeto s: https://www.lifewire.com/what-is-a-platform-game-812371

[3] Daniel Boutros, “A Detailed Cross-Examination of Yesterday and Today's Best-Selling Platform Games” 5.8.2006, pristupljeno 18.8.2022. [Online]. Preuzeto s: https://www.gamedeveloper.com/business/a-detailed-cross-examination-of-yesterday-and-today-s-best-selling-platform-games

[4] NFI, “Rotoscoping: Everything You Need To Know”, pristupljeno 18.8.2022. [Online]. Preuzeto s: https://www.nfi.edu/rotoscoping/

[5] Unity, „Unity User Manual 2021.3 (LTS)“, 2022, pristupljeno 18.8.2022. [Online]. Preuzeto s: https://docs.unity3d.com/Manual/index.html

[6] Magdalena Kačić-Barišić, „Primjena univerzalnog dizajna u edukacijskoj mobilnoj aplikaciji zasnovanoj na tehnologiji proširene stvarnosti“, siječanj 2021

10 PRILOZI

10.1 Popis slika

IZJAVA

Izjavljujem pod punom moralnom odgovornošću da sam završni rad izradio samostalno, isključivo znanjem stečenim na studijima Sveučilišta u Dubrovniku, služeći se navedenim izvorima podataka i uz stručno vodstvo mentora izv. prof. dr. sc. Krunoslav Žubrinić i komentorice Ane Kešelj mag. ing. comp. kojima se još jednom srdačno zahvaljujem.

Mario Curavić