Razvoj ritmičke platformske igre

Prga, Josip

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Josip Prga

RAZVOJ RITMIČKE PLATFORMERSKE IGRE

DIPLOMSKI RAD

Varaždin, 2024.

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Josip Prga

Matični broj: 0016140108

Studij: Informacijsko i programsko inžinjerstvo

RAZVOJ RITMIČKE PLATFORMERSKE IGRE

DIPLOMSKI RAD

Mentor/Mentorica:

Doc. dr. sc. Robert Kudelić

Varaždin, kolovoz 2024.

Josip Prga

Izjava o izvornosti

Izjavljujem da je moj završni/diplomski rad izvorni rezultat mojeg rada te da se u izradi istoga nisam koristio drugim izvorima osim onima koji su u njemu navedeni. Za izradu rada su korištene etički prikladne i prihvatljive metode i tehnike rada.

Autor/Autorica potvrdio/potvrdila prihvaćanjem odredbi u sustavu FOI-radovi

Sažetak

„RythmUp“ je dvo-dimezionalna ritmička igra. Inspirirana je dijelom poznatom igrom zvanom „Geometry Dash“. Izrađena je u alatu Unity koristeći C# skripte. Korišteno je Unity2D sučelje. Igra se sastoji od tri razine. Tematika igre je da igrač ulazi u špilju punu neprijatelja. Prva razina jos sadrži građevine i stubove dok druga i treća razine istražuju dublje dijelove špilje. Igrač se sam kreće prema desno te je potrebno preskakati neprijatelje u ritmu glazbe. Neprijatelji su kosturi, šišmiši, bića napravljena od sluzi i oči s krilima. Pri dodiru neprijatelja razina prestaje. Igrač ima mogućnost skakanja i kliženja te kada pokupi kuglu umjesto skakanja ima mogućnost mjenjanja gravitacije.

Ključne riječi: Unity, C#, Unity2D, video igre, platformeri, ritam

Sadržaj

1. Uvod

„RythmUp“ je 2D ritmični platformer izrađen u Unity alatu. Alat koristi C# skripte kako bi upravljao objektima igre. Inspiracija za ovu igru je bila poznati ritmični platformer zvani „Geometry Dash“. „Geomerty Dash“ je stekao veliku popularnost zahvaljujući mogućnosti kreacije vlastitih razina te djeljenjem istih. Time je broj razina postao ogroman te sama kreativnost razina nije bila limitirana na timu developera zaslužnih za igru nego na čitavoj bazi igrača.

Projekt „RythmUp“ se sastoji od tri razine te nekoliko izbornika. Tematika igre je putovanje u dubinu špilje te izbjegavanje neprijatelja u istoj. Paljenjem igrice prezentirani smo početnim izbornikom na kojem imamo opcije „Start Game“, „Settings“ i „Quit Game“. Biranjem opcije „Quit Game“ igra se gasi. Biranjem opcije „Settings“ imamo mogućnost mjenjanja kontrola igrača. Biranjem opcije „Start Game“ otvara se izbornik razina u kojem možemo izabrati jedan od tri razine. Biranjem razina počinje dana razina. Završavanjem razine ili dodirom neprijatelja pali se „Game Over“ izbornik u kojem imamo mogućnost ponavljanja razine ili vraćanja na izbornik levela ili glavni izbornik.

2. Korišteni alati

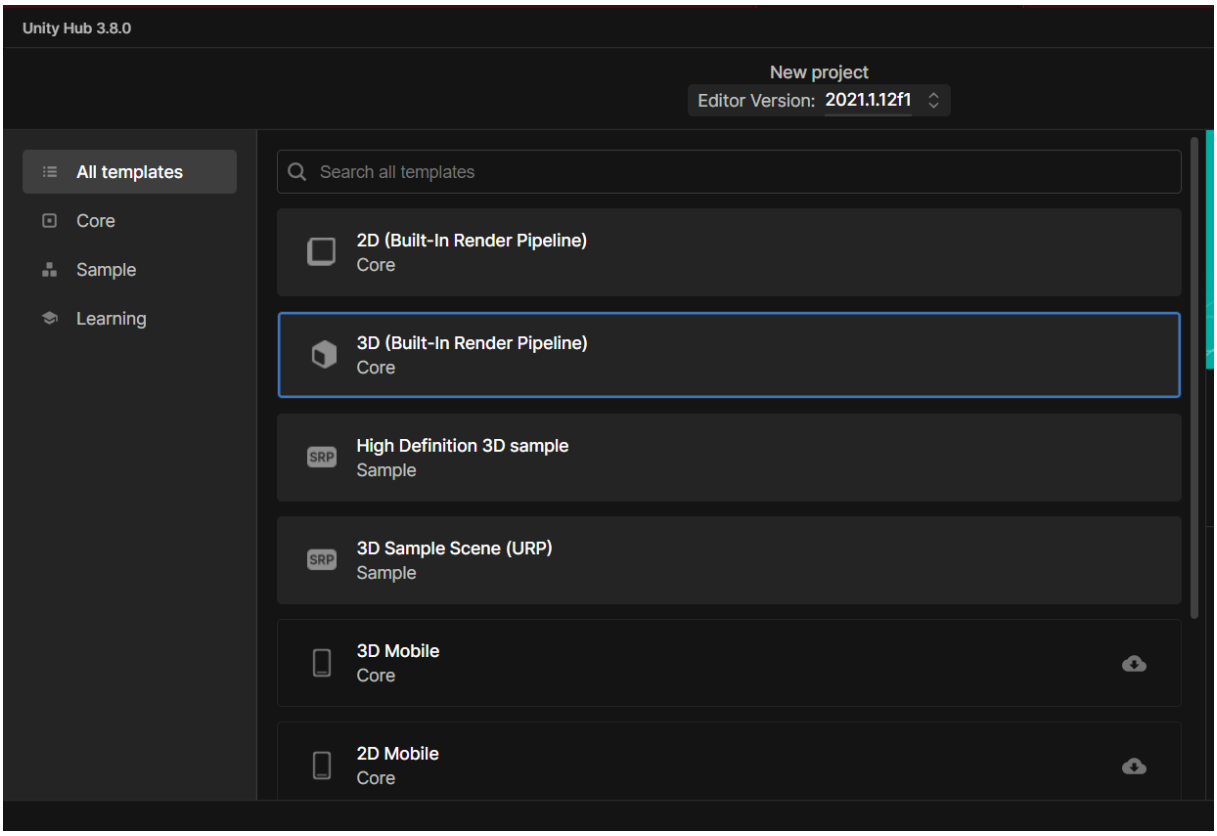
2.1. Unity

Unity je game engine napisan u C++-u. Međutim za skripte koristi C#. Napravljen je od strane istoimenog poduzeća. Unity je cross-platform te je danas jedan od najkorištenijih game engine-a osobito za indie igre. Neke poznatije igre napravljene u Unity-u su : „Ori and the Blind Forest“, „Hollow Knight“, „Fall Guys“ i mnoge druge.

Slika 1. Unity logotip

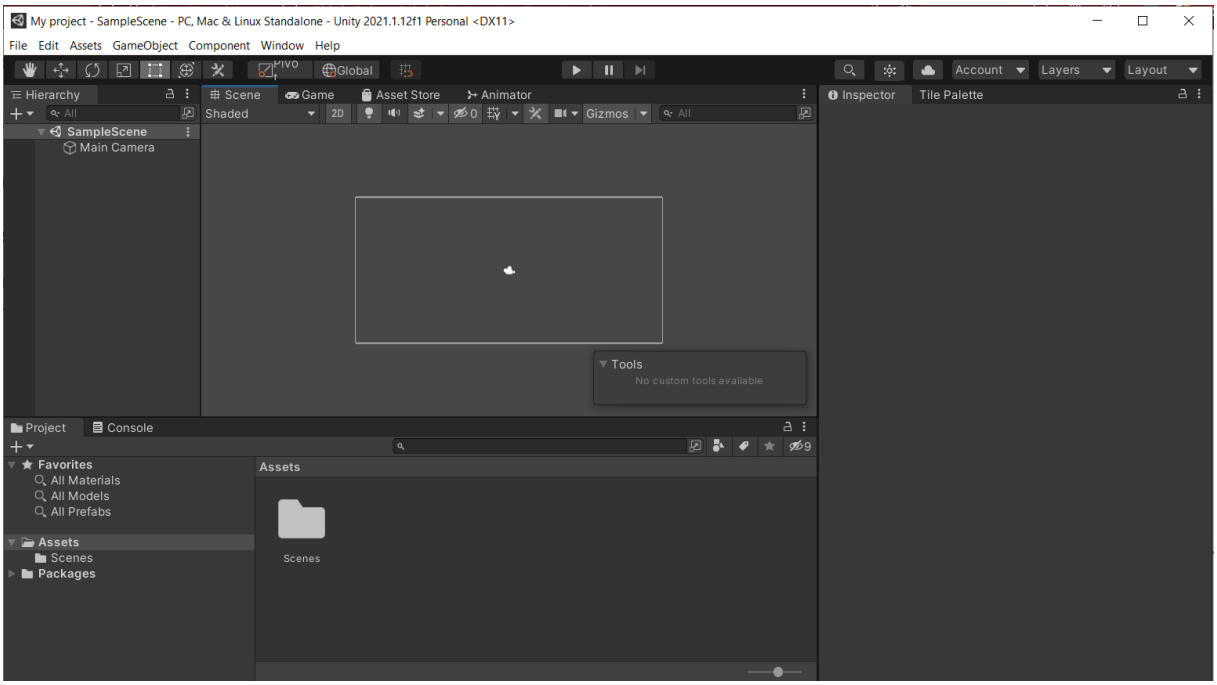
2.2. Unity alat

Sami alat Unity sadrži mnoge opcije kako bi olakšalo samu kreaciju video igre. Prije kreacije samog projekta imamo mogućnost birati neke već postojeće šablone. Možemo unaprijed odrediti koju vrstu video igre ćemo izrađivati kako bi Unity prilagodio svoje sučelje tom tipu video igre. Ponuđene opcije su: 2D, 3D, 2D Mobile, 3D Mobile, VR, AR i dva uzorka za 3D.



Slika 2. Unity šabloni

Za izradu projekta „RythmUp“ smo birali 2D šablon. Biranjem istog otvara se glavno sučelje za izradu video igre. Unutar glavnog sučelja Unity kreira početnu scenu zvanu „Sample Scene“ te u nju postavi kameru. Za početak izrade projekta potrebno je ubaciti objekte u scenu. Objekti mogu biti igrač, neprijatelj, okruženje, tipke i bilo što drugi s čim igra ima interakciju. U desnom dijelu sučelja imamo prozor zvani „Inspector“ koji prikazuje detalje izabranog objekta. Na drugom dijelu tog prozora imamo „Tile Palette“ koji služi za ubacivanje tekstura u 2D okruženje. U dnu sučelja imamo prozor sa dva dijela. Prvi dio je „Project“ koji služi za prolaženje kroz dijelove i datoteke projekta. Unutar toga ćemo vidjeti spremljene scene, teksture, slike, glazbu i sve ostalo vezano za projekt. Drugi dio je zvani „Console“ te služi kao razvojna konzola u koju možemo upistivati i iščitavati poruke projekta. Lijevo u sučelju se nalazi prozor zvan „Hierarchy“ u kojem vidimo scene i objekte scene ako je ta scena učitana. Odabirom objekta na „Hierarchy“ prozoru se njegovi detalji prikažu na „Inspector“ prozoru. U izbornicima iznad imamo opcije alata i tipke za pokretanje projekta. Izbor alata koje imamo na raspolaganju su alat za micanje scene, alat za micanje objekata u sceni, alat za rotiranje objekata u 2D prostoru, alat za promjenu veličine objekata, alat za selekciju vise objekata, alat za rotaciju u 3D prostoru te mogućnost dodavanja vlastitog prilagođenog alata. Sredina sučelja predstavlja sama aktivna scena s objektima.



Slika 3. Unity sučelje

2.3. C# skripte

Za interakciju između objekata igre koriste se C# skripte. Pri kreaciji skripte automatski dobijemo dvije metode „Start()“ i „Update()“. Metoda „Start()“ se okida jednom pri pokretanju skripte. Većinom se koristi za inicijalizaciju objekata ili postavljanje početnih vrijednosti varijabla. Metoda „Update()“ se poziva jednom po sličici (*frame*). Koristi se za većinu akcija igrača kao kretnja i skokovi. U projektu „RythmUp“ koristimo i metode „FixedUpdate()“ i „OnTriggerEnter2D(Collider2D collider)“. Za razliku od „Update()“ metode koja uvijek radi jednom po sličici metoda „FixedUpdate()“ se koristi za kalkulacije fizickih svojstava objekata te se po zadanom okida jednom svako 0.02 sekunde što je zadana frekvencija sustava za fiziku. Ta opcija se može promjeniti u opcijama Unity-a. „OnTriggerEnter2D()“ metoda služi kako bi odradili interakciju dva objekta kada dođu u dodir jedan s drugim.

3. Razrada teme

Ova igra je uvelike inspirirana video igrom „Geometry Dash“. Igra „Geometry Dash“ je izašla 2013. na Androidu i iOS-u te godinu nakon na Windowsu i Mac-u. U igri je cilj na ritam glazbe preskakati preko različitih prepreka. U slučaju igre „RythmUp“ prepreke su neprijatelji s različitim karakteristikama.

3.1. Tematika

„RythmUp“ razine su zamišljene kao različite dubine špilje u koju igrač ulazi. Neprijatelji prve razine su kosturi i šišmiši. Druga razina je zamisljena kao dublji dio špilje te su neprijatelji bića napravljena od sluzi i živuće oči s krilima. Posljednja razina posjeduje sve vrste neprijatelja. Igrač je obučen u žuto hazmat odlijelo.

3.2. Paketi

S tim da je cilj ovog rada implementacija logike video igre korišteni su već postojeći paketi. Paket koji je korišten je:

- „Super Grotto Escape Pack“ [3]

Paket je besplatan te sadrži teksture i animacije za igraca i sve neprijatelje. Također sadrzi i set tekstura za pozadine i okruženje.

Što se tiče glazbe video igre također smo posudili postojeću glazbu. Svaka razina ima svoju pjesmu i time imamo tri pjesme:

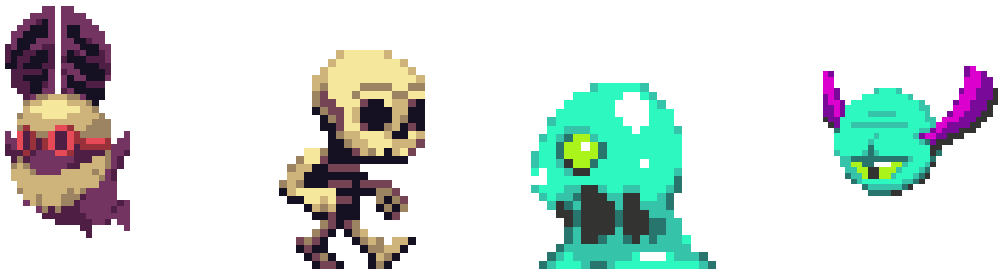
- Loader – Chris Huelsbeck (Turrican Soundtrack Vol.2) [4]
- Base After Base – DJVI (Geometry Dash Songs) [5]
- Samurai Techno – fizzd (Rythm Doctor Soundtrack) [6]

3.3. Dizajn likova

Od aktivnih likova unutar igre imamo glavnog igrača, kosture, šišmiše, bića napravljena od sluzi i leteće oči. Glavni igrač je lik obučen u hazmat odijelo žute boje. Za lika imamo 3 različtite animacije. Prva animacija i početna animacija je animacija trčanja. S tim da lik uvijek trči nemamo potrebe za neaktivnu animaciju. Uz animaciju za trčanje imamo i animacije za skok i za klizanje.

Slika 4. Izgled glavnog lika

Uz glavnog lika imamo i animacije za neprijatelje. Svaki neprijatelj ima samo jednu animaciju. U slučaju da je neprijatelj koji se kreće animacija je za kretanje dok u slučaju da je neprijatelj koji je stacioniran imamo neaktivnu animaciju. S tim da su kretajući likovi šišmis i kostur te s tim da šišmiš izgleda isto kretajući se i stojeći u mjestu animaciju za kretanje ima kostur.



Slika 5. Izgledi neprijatelja

3.4. Skripte

3.4.1. Skripta glavnog igrača

Većina koda video igre je sadržano u ovoj skripti. Skripta se sastoji od metoda „Start()“, „Update()“, „FixedUpdate()“, „OnTriggerEnter2D(Collider2D collision)“ i „DeathMenu()“. S tim da je skripta prilično velika objašnjavat ćemo metodu po metodu.

3.4.1.1. Start metoda

Start metoda služi kako bi inicijalizirali svih potrebnih objekata i varijabla. Metoda izgleda:

```
    public float speed;
    public static float xPosition;
    public float jumpHeight;
    public float gravityScale;
    public Animator animator;

    float colliderOffsetY;
    float colliderSizeY;
    float colliderSizeX;
    float direction = 1;
    bool isGrounded = false;
    Rigidbody2D r2d;
    CapsuleCollider2D collider;
    Transform t;
    bool hasOrb = false;

    void Start()
    {
        StaticData.sceneIndex = SceneManager.GetActiveScene().name;
        t = transform;
        r2d = GetComponent<Rigidbody2D>();
        collider = GetComponent<CapsuleCollider2D>();
        r2d.freezeRotation = true;
        r2d.collisionDetectionMode = CollisionDetectionMode2D.Continuous;
        gravityScale = 9.5f;
        r2d.gravityScale = gravityScale;
        colliderSizeY = collider.size.y;
        colliderSizeX = collider.size.x;
        colliderOffsetY = collider.offset.y;
        speed = 6.7f;
        jumpHeight = 20.5f;
    }
```

U kodu iznad vidimo mnoge varijable i objekte koje su potrebne kako bi upravljali našim glavnim likom. Varijable su:

- “speed” – varijabla koja određuje brzinu igrača. Postavlja se na 6.7 u “Start()” metodi
- “xPosition” – pozicija x osi igrača koju koristimo u drugoj skripti.
- “jumpHeight” – broj koji određuje koliko naš igrač može skočiti. Postavlja se na 25.5 unutar “Start()” metode
- “gravityScale” – broj koji određuje snagu gravitacije u igri. Zajedno s varijablom „jumpHeight“ određuje mogućnost skoka glavnog igrača
- „animator“ – objekt koji služi za upravljanjem animacija glavnog igrača
- „colliderOffsetY“ – predstavlja udaljenost „collider“-a od središnje točke. U „Start()“ metodi se postavlja kao zadani u Unity-u te služi da bi rotirali igrača pri klizanju.
- „colliderSizeY“ – predstavlja veličinu igračevog „collider“-a na y osi. Također služi za akciju klizanja. Postavlja se na zadanu u „Start()“ metodi.
- „colliderSizeX“ – prestavlja veličinu igračevog „collider“-a na x osi. Također služi za akciju klizanja. Postavlja se na zadanu u „Start()“ metodi.

- „moveDirection“ – smjer u kojem se igrač kreće. S tim da u igri „RythmUp“ igrač se uvijek kreće sam ovom varijablom definiramo smjer. Broj jedan predstavlja smjer kretanja u desno.
- „isGrounded“ – predstavlja je li igrač prizemljen ili ne. Ako igrač nije prizemljen nema mogućnost skakanja.
- „r2d“ – objekt tipa „Rigidbody2D“ predstavlja fizički objekt samog igrača. U metodi „Start()“ ovaj objekt se inicijalizira te se postavljaju neke osnovne postavke. Postavka „freezeRotation“ označava to da će sami objekt lika uvijek biti uspravan. „collisionDetectionMode“ označava konstantnu provjeru kolizije između objekata. Ta postavka je bitna kako bi igra u svakom trenutku mogla znati je li igrač dira neprijatelja ili završnu traku. „gravityScale“ predstavlja gravitaciju objekta.
- „mainCollider“ – objekt tipa „CapsuleCollider2D“ predstavlja „collider“ oko samog igrača. Taj objekt služi kako bi mogli detektirati kolizije s drugim objektima u igri.
- „t“ – objekt tipa „Transform“ predstavlja samu lokaciju objekta u igri
- „hasOrb“ – predstavlja drži li glavni lik kuglu. Kugla je tip objekta u igri koja mjenja način kretanja igrača.

Također u početku „Start()“ metode u statičnu skriptu zvanu „StaticData“ spremamo naziv trenutne scene. To nam je potrebno za izbornik „GameOver“.

3.4.1.2. Update metoda

„Update()“ metoda se okida jednom po okidanju sličice („once per frame“). Služi za generalno upravljanje likom.

```
    void Update()
    {
        xPosition = r2d.position.x;

        if (!hasOrb)
        {
            t.localScale = new Vector3(Mathf.Abs(t.localScale.x), t.localScale.y,
transform.localScale.z);

            if (Input.GetKeyDown(StaticData.jumpButton) && isGrounded &&
!Input.GetKey(StaticData.slideButton))
            {
                animator.SetBool("isJumping", true);
                r2d.velocity = new Vector2(r2d.velocity.x, jumpHeight);
            }

            if (Input.GetKey(StaticData.slideButton) && isGrounded)
            {
                animator.SetBool("isCrouching", true);
                mainCollider.direction = CapsuleDirection2D.Horizontal;
                mainCollider.size = new Vector2(colliderOffsetY, colliderSizeX);
                mainCollider.offset = new Vector2(mainCollider.offset.x, -0.09F);
            }

            if(Input.GetKeyUp(StaticData.slideButton))
            {
                animator.SetBool("isCrouching", false);
                mainCollider.direction = CapsuleDirection2D.Vertical;
                mainCollider.size = new Vector2(colliderSizeX, colliderSizeY);
                mainCollider.offset = new Vector2(mainCollider.offset.x,
colliderOffsetY);
            }
        }
        else
        {
            if (Input.GetKeyUp(StaticData.jumpButton))
            {
                r2d.gravityScale = gravityScale * -1;
                gravityScale = gravityScale * -1;
                transform.Rotate(0f, 180f, 180f);
            }
        }

    }
```

U skripti za početak postavljamo poziciju igrača u statičnu varijablu „xPosition“. Ta varijabla nam je potrebna u drugoj skripti. Nakon toga pregledavamo je li igrač pokupio kuglu. Po varijabli „hasOrb“ određujemo način kretanja igrača. Ako igrač nema kuglu, što je ujedno i zadana postavka svake razine, pregledavamo unose tipki. Tipke su kofigurabile te se statične varijable za skok i klizanje nazivaju „jumpButton“ i „slideButton“.

Prvoj provjeri gledamo je li kliknuta tipka za skok, je li igrač prizemljen te provjeravamo da tipka za klizanje nije kliknuta. Provjera za prizemljenje je napravljena kako igrač ne bi mogao skakati dok je već u skoku. Provjera za tipku za klizanje je napravljena kako igrač ne bi mogao skočiti

dok kliže. Ako su svi uvjeti zadovoljeni postavljamo animatoru animaciju za skakanje te objektu r2d postavljamo akciju za skok tako da mu promjenimo vertikalno ubrzanje.

U drugoj selekciji imamo dva uvjeta. Prvi uvjet je da je kliknuta tipka za klizanje. Drugi uvjet je da je igrač prizemljen. Ispunjivanjem ovih uvjeta igrač uđe u akciju klizanja. Prvo animator postavljamo u akciju klizanja. Nakon toga radimo adaptacije na igračevom „collider“-u. Prvo mjenjamo smjer kretanja „collider“-a. Potom varijable koje smo spremili u „Start()“ metodi zamjenjujemo kako bi promjenili veličinu samog „collider“-a. Nakon zamjene visine s širinom „collider“ pobjegne od centra objekta te ga s „offset“ postavkom postavimo na sredinu.

Treća selekcija je vraćanje igrača u stojeći položaj. Uvjet je puštanje tipke za klizanje. Nakon što je uvjet zadovoljen gasi se animacija za klizanje. „Collider“ se vraća u vertikalni smjer te mu vraćamo unaprijed zadane postavke pomoću varijablih definiranih u „Start()“ metodi.

Dalje imamo opciju kretanja ako igrač ima kuglu. Kada igrač posjeduje kuglu klikom na tipku za skok umjesto skoka mjenja smjer gravitacije. To postižemo tako što mnozimo varijablu gravitacije s negativnim jedan te radimo okretaj na poziciju igrača.

3.4.1.3. FixedUpdate metoda

„FixedUpdate()“ metoda je metoda koja se okida svako 0.02 sekunde te se u njoj većinom upravlja stvarima vezanim za samu fiziku objekta.

```
    void FixedUpdate()
    {
        float colliderCircle = collider.size.x * 0.4f * Mathf.Abs(transform.lo-
calScale.x);
        Bounds colliderBounds = collider.bounds;
        Vector3 groundChecker = colliderBounds.min + new Vector3(collid-
erBounds.size.x * 0.5f, colliderCircle * 0.9f, 0);
        isGrounded = false;

        Collider2D[] colliders = Physics2D.OverlapCircleAll(groundChecker, col-
liderCircle);

        if (colliders.Length > 0)
        {
            for (int i = 0; i < colliders.Length; i++)
            {
                if (colliders[i] != collider)
                {
                    animator.SetBool("isJumping", false);
                    isGrounded = true;
                    break;
                }
            }
        }

        r2d.velocity = new Vector2((direction) * speed, r2d.velocity.y);

    }
```

Na početku metode u varijablu spremamo limite i radijus glavnog „collider“-a. Nakon toga postavljamo poziciju dna. Provjeravamo postoje li kolizije između „collider“-a. Vraćamo varijablu „isGrounded“ na negativno kako bi je resetirali te nakon toga ako se igrač dodiruje s tlom postavljamo animaciju skoka na negativno te varijablu „isGrounded“ na istinito. U ovoj metodi također postavljamo neprestanu kretnju igrača prema desno postavljajući njegovu „r2d.velocity“ x os na potrebnu brzinu i smjer.

3.4.1.4. OnTriggerEnter2D i DeathMenu metode

„OnTriggerEnter2D“ je metoda koja se aktivira jednom kada „collider“ dođe u dodir sa drugim „collider“-om koji je definiran kao „trigger“. S tim da se metoda „DeathMenu“ poziva samo u ovoj metodi prikazat ćemo odmah i nju.

```
void OnTriggerEnter2D(Collider2D collision)
    {
        if(collision.gameObject.tag == "FinishLine" ||
            collision.gameObject.tag == "Skely" ||
            collision.gameObject.tag == "DeathMap")
        {
            DeathMenu();
        }
        if (collision.gameObject.tag == "Orb")
        {
            Destroy(collision.gameObject);
            if (hasOrb)
            {
                hasOrb = false;
            }
            else
            {
                hasOrb = true;
            }
        }
        if(collision.gameObject.tag == "SpeedBoost")
        {
            speed *= 1.49F;
            animator.speed = 1.49F;
        }
    }

    public void DeathMenu()
    {
        SceneManager.LoadScene("GameOver");
    }
```

Ova metoda je glavna metoda za provjeravanje svih interakcija između objekata. U prvoj selekciji imamo provjeru je li igrač dotakao jedno od entiteta koje završavaju igru. Entiteti su „FinishLine“ što predstavlja kraj igrice, „Skely“ što je univerzalni tag za sve neprijatelje i „DeathMap“ što su dijelovi mape koji su smrtonosni za igrača. Ako se ijedan od uvjeta ispuni okida se metoda „DeathMenu()“ koja otvara scenu za kraj igre zvanu „GameOver“.

Druga selekcija služi za akciju kupljenja kugle. Prvo se provjerava je li objekt koji ima koliziju ima tag „Orb“. Ako ima taj objekt se briše kako bi simulirali kupljenje kugle. Nakon toga provjeravamo je li igrač već jednom skupio kuglu ili ne i po tom definiramo varijablu „hasOrb“.

Zadnja selekcija metode je za funkcionalnost igre zvanu „SpeedBoost“. Ako igrač dođe u koliziju s objektom koji ima tag „SpeedBoost“ njegova brzina se poveća za 1.49.

3.4.2. Skripta za kameru

Potrebno nam je da kamera prati igrača. Zbog toga je napravljena skripta za kameru.

```
public class CameraScript : MonoBehaviour
{
    Rigidbody2D r2d;
    void Start()
    {
        r2d = GetComponent<Rigidbody2D>();
    }

    void Update()
    {
        r2d.MovePosition(new Vector2(MainPlayerScript.xPosition+15, 0));
    }
}
```

U ovoj skripti koristimo statičnu varijablu pozicije igrača na x osi spomenutu iznad. Jednostavniji način da kamera prati igrača je da jednostavno zakačimo objekt kamere na objekt igrača. Međutim kada napravimo to kamera prati i y os igrača što prouzrokuje skakanje kamere s igračem. Kako bi kamera pratila igrača samo na x osi napravljena je ova skripta. U „Start()“ metodi inicijaliziramo r2d kamere. „Update()“ metoda za svaku okinutu sličicu („once per frame“) ažurira lokaciju kamere na lokaciju igrača.

3.4.3. Skripta za kretajuće neprijatelje

Kretajući neprijatelji u igri su kosturi i šišmiši. Kako bi se kretali potrebna im je jednostavna skripta za kretanje.

```
public class SkelyScript : MonoBehaviour
{
    public float speed = 3.4f;

    Rigidbody2D r2d;

    void Start()
    {
        r2d = GetComponent<Rigidbody2D>();
    }

    void Update()
    {
        r2d.velocity = new Vector2((-1) * speed, r2d.velocity.y);
    }
}
```

U početku skripte definiramo brzinu kretanja. Brzinu postavljamo na 3.4. U „Start()“ metodu inicijaliziramo r2d objekt. U ovoj metodi smo mogli također postaviti brzinu kretanja no kako bi

mogli mjenjati samu brzinu u Unity sučelju definira se iznad. U „Update()“ metodi radimo akciju kretanja tako što objektu r2d mjenjamo ubrzanje na x osi.

3.4.4. Skripta za glavni izbornik

Za sve izbornike u projektu potrebne su nam skripte. Skripte služe za davanje komandi na klik tipki.

```
public class MainMenu : MonoBehaviour
{
    public void StartGame()
    {
        SceneManager.LoadScene("LevelSelect");
    }

    public void Settings()
    {
        SceneManager.LoadScene("Settings");
    }

    public void ExitGame()
    {
        Application.Quit();
    }
}
```

Metode se u Unity sučelju postavljaju na klik tipke. Imamo metode „StartGame()“ koja otvara izbornik za razine igre. Metodu „Settings()“ koja otvara izbornik za opcije. I metodu „ExitGame()“ koja gasi aplikaciju.

3.4.5. Skripta za izbornik razina

Unutar scene za izbornik razina imamo četiri tipke. Te tipke upravljamo skriptom za izbornik razina.

```
public class LevelSelectScript : MonoBehaviour
{
    public void LevelOne()
    {
        SceneManager.LoadScene("SampleScene");
    }

    public void LevelTwo()
    {
        SceneManager.LoadScene("SecondLevel");
    }

    public void LevelThree()
    {
        SceneManager.LoadScene("FinalLevel");
    }

    public void Back()
    {
        SceneManager.LoadScene("MainMenu");
    }
}
```

Prva metoda otvara prvu razinu zvanu „SampleScene“. Druga metoda otvara drugu razinu zvanu „SecondLevel“. Te metoda zvana „LevelThree()“ otvara zadnju razinu zvanu „FinalLevel“. Također imamo i tipku za povratak natrag na scenu „MainMenu“.

3.4.6. Skripta za kraj igre

Ako igrač dodirne neprijatelja, dio razine koji ga uništava ili kraj razine otvara se izbornik za kraj razine.

```
public class GameOverScript : MonoBehaviour
{
    public void MainMenu()
    {
        SceneManager.LoadScene("MainMenu");
    }

    public void LevelSelect()
    {
        SceneManager.LoadScene("LevelSelect");
    }

    public void Retry()
    {
        SceneManager.LoadScene(StaticData.sceneIndex);
    }
}
```

Kao i u izbornicima iznad imamo dvije tipke koje otvaraju drugu scenu. To su tipke „MainMenu“ koja vraća na početni izbornik te tipka „LevelSelect“ koja vraća na izbornik za selekciju razina. Također imamo tipku „Retry“ koja koristi podatke spremljene u statičnoj skripti koji se spremaju

u skripti za glavnog igrača. U skripti za glavnog igrača se upiše ime trenutne razine kako bi mogli u skripti za kraj igre ponoviti istu tu razinu klikom na tipku.

3.4.7. Skripta za opcije

Skipta za opcije služi kako bi mjenjali statične podatke tipki za skok i za klizanje.

```
public class SettingsScript : MonoBehaviour
{

    private void Start()
    {
        TMP_InputField jumpInputField =
GameObject.FindGameObjectWithTag("JumpInput").GetComponent<TMP_InputField>();
        TMP_InputField slideInputField =
GameObject.FindGameObjectWithTag("SlideInput").GetComponent<TMP_InputField>();


        jumpInputField.text = StaticData.jumpButton.ToString();
        slideInputField.text = StaticData.slideButton.ToString();
    }

    public void UpdateJump(string jumpButton)
    {
        System.Enum.TryParse(jumpButton, true, out KeyCode keyCode);
        StaticData.jumpButton = keyCode;
    }

    public void UpdateSlide(string slideButton)
    {
        System.Enum.TryParse(slideButton, true, out KeyCode keyCode);
        StaticData.slideButton = keyCode;
    }

    public void MainMenu()
    {
        SceneManager.LoadScene("MainMenu");
    }
}
```

U „Start()“ metodi lociramo dva ulazna polja za tipke. Nakon što smo ih locirali u njih upisujemo postojeće već unešene tipke za skok i za klizanje. Nakon toga imamo tri metode koje se okidaju iz same scene. Metoda „UpdateJump()“ i „UpdateSlide()“ se okidaju pri kraju unosa u njihova unosna polja. Nakon što smo unijeli tipku koju želimo te kliknemo negdje izvan unosnog polja okida se metoda. Metoda pokušava iz tipke izvuc enumarator odgovarajućeg ključa tipke. Ako uspije pronaći odgovarajući ključ taj isti se sprema u statičnu skriptu. Također imamo i metodu „MainMenu()“ koja na klik tipke nas vraća natrag u početni izbornik.

3.4.8. Skripta sa statičnim podacima

Ovo je skripta koja drži sve statične podatke kako bi se mogli djeliti između drugih skripti.

```
public class StaticData : MonoBehaviour
{

    public static KeyCode jumpButton = KeyCode.W;

    public static KeyCode slideButton = KeyCode.S;

    public static string sceneIndex = "";

}
```

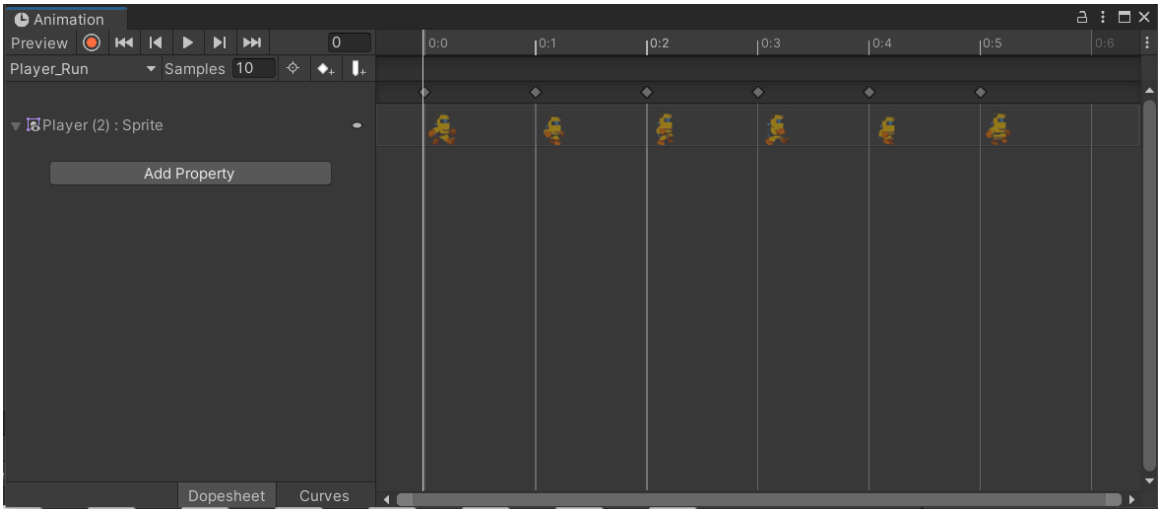
Kao što vidimo iznad postavljene su tipke W i S kao početne tipke za kretnju igrača. Osim te dvije tipke imamo i statičnu varijablu zvanu „sceneIndex“ u koju se upisuje trenutni naziv scene. S tim da to upisivanje pozivamo samo u skripti glavnog igrača tu se spremaju nazivi scena razina. Koristi se za ponavljanje iste razine u sceni za kraj igre.

3.5. Animacije

Svaki lik igre ima određenu animaciju. Svi neprijatelji imaju po jednu animaciju dok sami igrač ima tri animacije.

3.5.1. Izrada animacije

S tim da koristimo već postojeće pakete sličice za animacije su nam već izrađene. Kako bi napravili samu animaciju potrebno je otvoriti prozor za animacije. To možemo učiniti prečacem „Ctrl + 6“. Nakon sto otvorimo sami prozor za animacije potrebno je povući sličice u prozor.

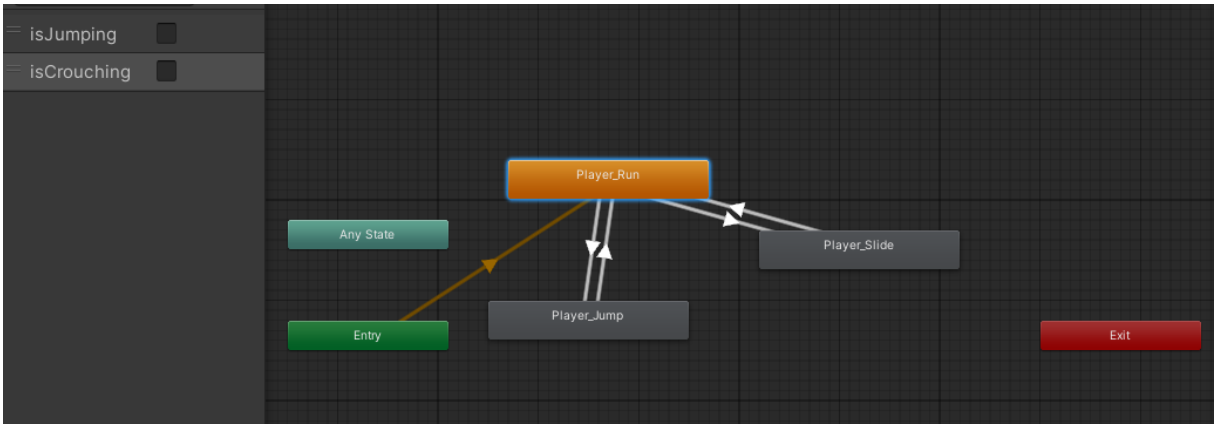


Slika 6. Prozor za izradu animacije

Na slici iznad vidimo alat za izradu sa šest povučenih sličica. Nakon što podesimo postavku „Samples“ na željenu, što je u našem slučaju 10, animacija je napravljena. Možemo je pregledati tipkom za start animacije u gornjem lijevom kutu. Nakon što kreiramo sve željene animacije potrebno ih je podesiti u animatoru.

3.5.2. Animator

Animator je alat kojim upravljamo animacijama. U animatoru, zajedno sa skriptama, postavljamo u kojem trenutku se aktivira koja animacija. U skriptama iznad smo vidjeli u trenucima skoka i klizanja promjene varijabla animatora. Varijable su „isJumping“ i „isCrouching".



Slika 7. Animator

U slici iznad vidimo sami alat. Animatoru moramo odrediti njegovu početnu animaciju. U našem slučaju početna animacija je animacija za trčanje. Za sve ostale likove osim glavnog igrača jedina animacija je početna animacija. Međutim igrač ima i animaciju za skok i animaciju za klizanje. U lijevom gornjem kutu vidimo varijable kojima određujemo samo ponašanje animatora. Mjenjanjem varijabli u skriptama mjenja se sama animacije. Vraćanjem varijabla na početna vraćamo se na početnu animaciju, animaciju za trčanje.

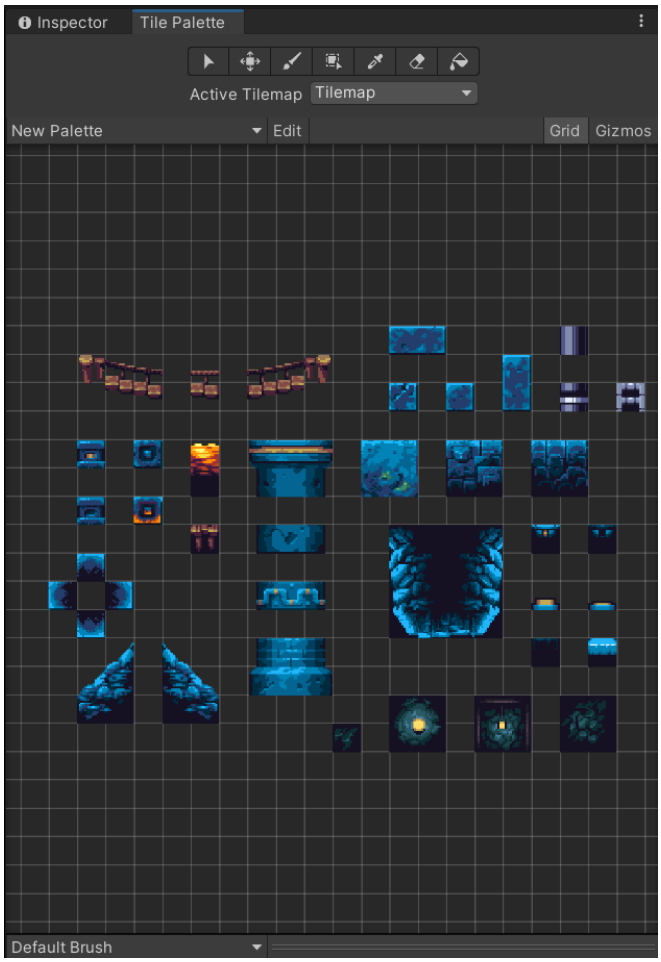
3.6. Dizajn razina

Za dizajn razina koristili smo većinom mrežu („Grid“) preko kojeg smo stavili teksture iz preuzetog paketa. U projektu imamo četiri različite mreže koje ponavljamo kroz razine. Te mreže su:

- „Tilemap“ – glavna mreža. Kreira razinu po kojoj se igrač kreće. Na sebi ima „collider“ koji nije okidač („trigger“). Pod i plafon svake razine je napravljen na ovoj mreži.
- „Background“ – pozadinska mreža. Sastoji se samo od tekstura koje koristimo kako bi uredili samu razinu. S tim da se nalazi u pozadini na sebi nema „collider“ te se ne koristi za akcije prema igraču.
- „DeathMap“ – u posljednjoj razini igre postoje dijelovi mape koji su prepreka igraču. U slučaju da igrač dođe u kontakt s ovim dijelom razine dogodit će se kraj igre te će se otvoriti scena za kraj igre. Sastoji se od „collider“-a koji služi kao okidač te je označen s imenom „DeathMap“ kako bi ga mogli razlikovati u skripti.
- „FinishLine“ – na kraju svake razine nalazi se tekstura koja označava kraj igre. Kada igrač dodirne istu okida se kraj igre. Sastoji se od „collider“-a koji služi kao okidač te ima naziv „FinishLine“ kako bi ga razlikovali u skripti.

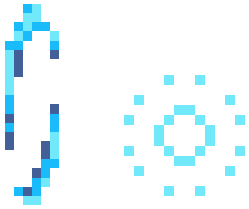
Većina tekstura je izvučeno iz „Tile Palette“ kojeg smo dobili iz paketa. U tom alatu možemo birati između sedam različitih alata.

- „Select tool“ – zadani alat s kojim biramo dio tekstura koje ćemo koristiti za uređivanje razina
- „Move tool“ – alat za pomicanje već postavljene teksture
- „Brush tool“ – alat koji se koristi za postavljanje teksture na razinu. Koristi se kao kist sa selekcijom odabranom prvim alatom
- „Box tool“ – alat koji postavlja izabranu teksturu tako što odabiremo površinu na razini koju želimo popuniti odabranom teksturom
- „Pick tool“ – alat kojim možemo odabrati teksturu na razini kao izabranu kako bi je ponovno koristili.
- „Eraser tool“ – alat za brisanje već postavljenih tekstura
- „Fill tool“ – popunjava sav prazni prostor sa odabranom teksturom



Slika 8. Tile Pallete

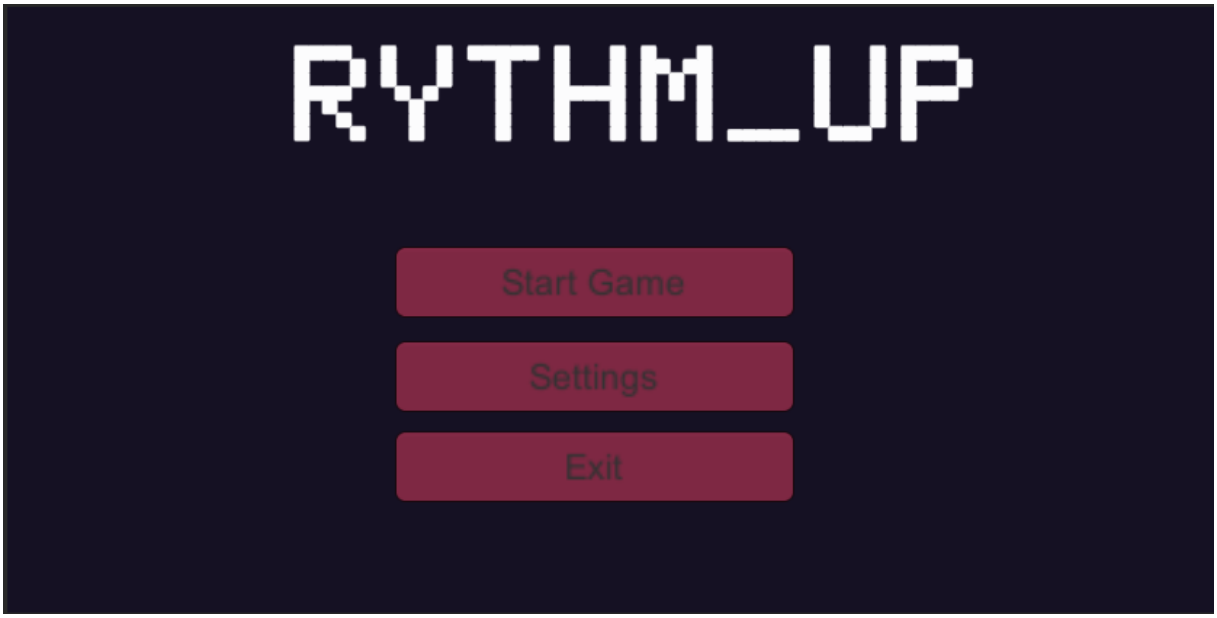
Na slici iznad vidimo sve teksture korištene za razine. Pored tekstura iz „tileset“-a imamo i teksture ubačene kao objekti. Imamo teksture koje su ubačene kao pozadine. Pored njih imamo i teksture koje smo koristili za „Speed Boost“ i „Orb“ akcije koje mjenjaju način ponašanja igre. Kada igrač pokupi „Orb“ mjenja se način kretnje te umjesto skoka dobije mogućnost mjenjanja gravitacije. „Speed Boost“ pobrzava kretnju igrača za 1.49.



Slika 9. „Speed Boost“(lijevo) i „Orb“(desno)

3.6.1. Dizajn početnog izbornika

Početni izbornik se sastoji od tri tipke i naslova. Prva tipka nas vodi na izbornik razina, druga tipka nas vodi na opcije. Treća tipka gasi samu aplikaciju. Ovo je ujedno i prva scena koja se otvori pri paljenju igre.



Slika 10. Dizajn početnog izbornika

3.6.2. Dizajn opcija

U sceni za opcije imamo dva polja za unos, jednu tipku i naslov. Unos se sprema čim prestanemo tipkati u polje. Tipka služi za povratak u glavni izbornik.

Slika 11. Dizajn scene za opcije

3.6.3. Dizajn izbornika za razine

Izbornik se sastoji od četiri tipke i naslova. Tri velike tipke predstavljaju različite razine dok četvrta tipka vodi natrag u glavni izbornik.

Slika 12. Dizajn izbornika razina

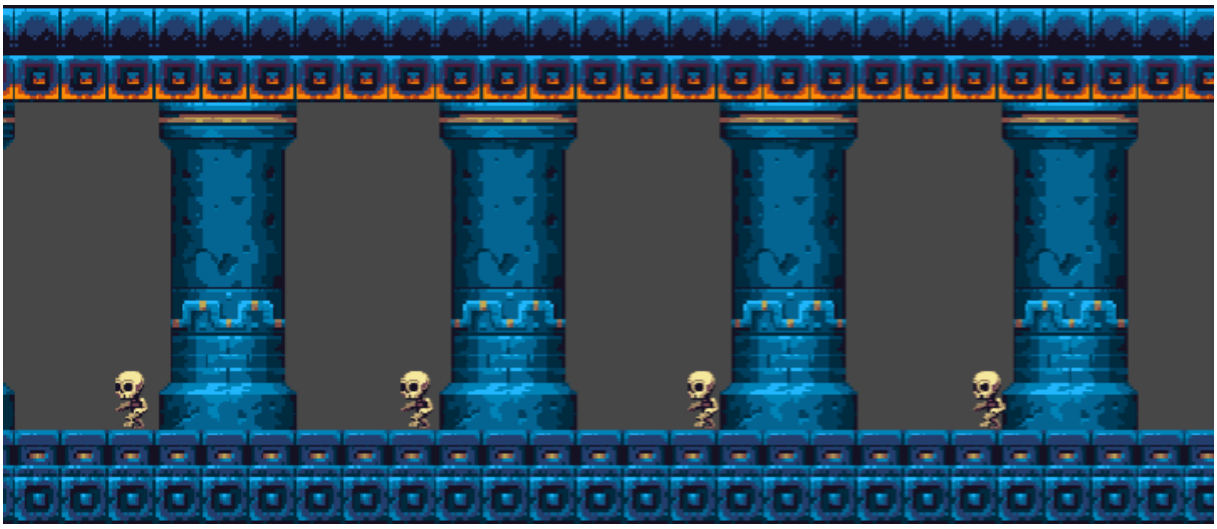
3.6.4. Dizajn za završnu scenu

Izbornik se sastoji od tri tipke i naslova. Prva tipka ponavlja razinu koja je upravo završila. Druga tipka vodi na izbornik razina i zadnja tipka vodi na početni izbornik.

Slika 13. Dizajn završne scene

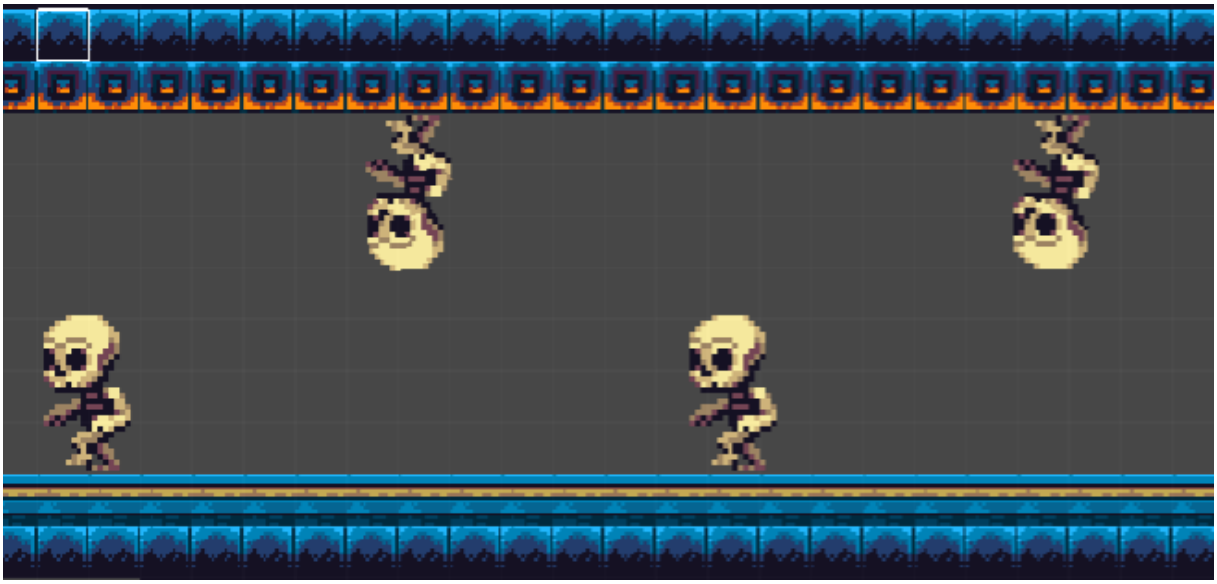
3.6.5. Dizajn prve razine

Prva razina predstavlja ulaz u samu špilju. Pri ulazu špilja je uređena i ima mnogo stubova. Ulaz podsjeća na ulaz u hram. Sastoji je od tri dijela. Prvi dio razine se sastoji samo od kostura te je jedina potrebna akcija skakanje. Kosturi se nalaze ispred gomilu stubova. Svi neprijatelji u ovoj razini su pokretni.



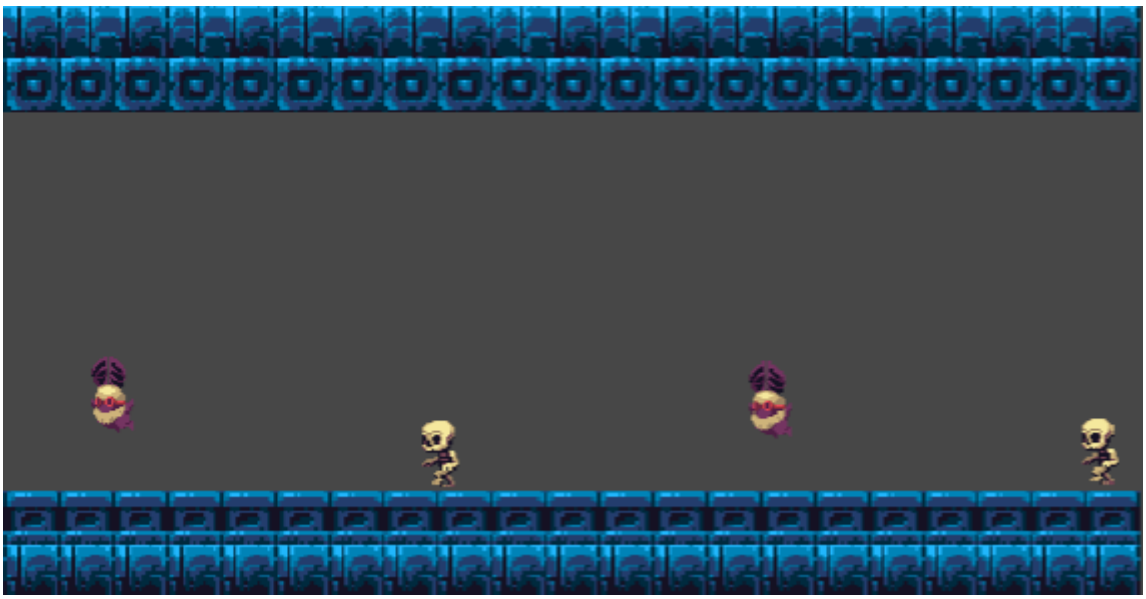
Slika 14. Prvi dio prve razine

Prije drugog dijela razine igrač kupi kuglu te mu se mjenja način igre. Sastoji se od velikih kosturova koji se nalaze nasuprot jedni drugim, plafon i pod. Potrebno je mjenjati gravitaciju kako bi prošli između istih.



Slika 15. Drugi dio prve razine

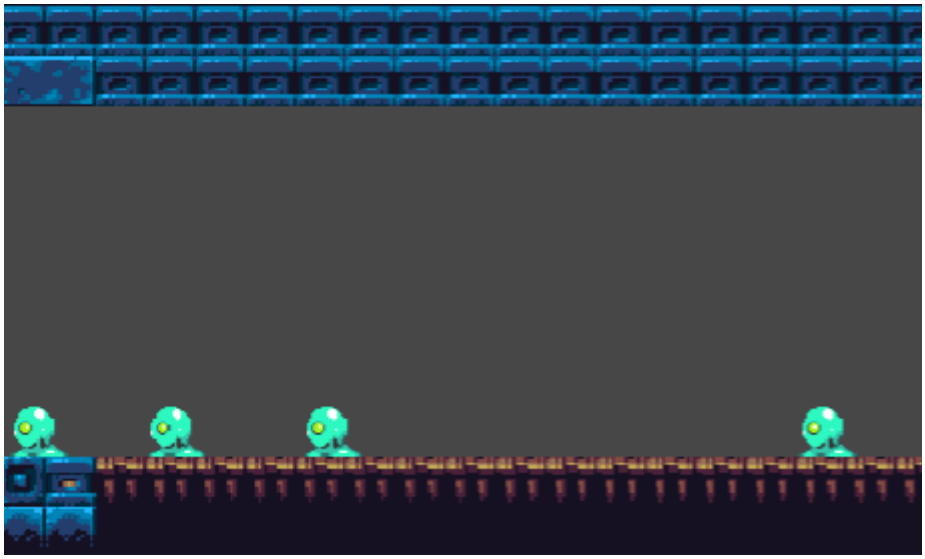
Na završetku drugog dijela igrač ponovno kupi kuglu te mu se vraća zadani način kretnje. Međutim pojavljuju se leteći neprijatelju, u ovom slučaju šišmiši, ispod kojih igrač mora klizati.



Slika 16. Treći dio prve razine

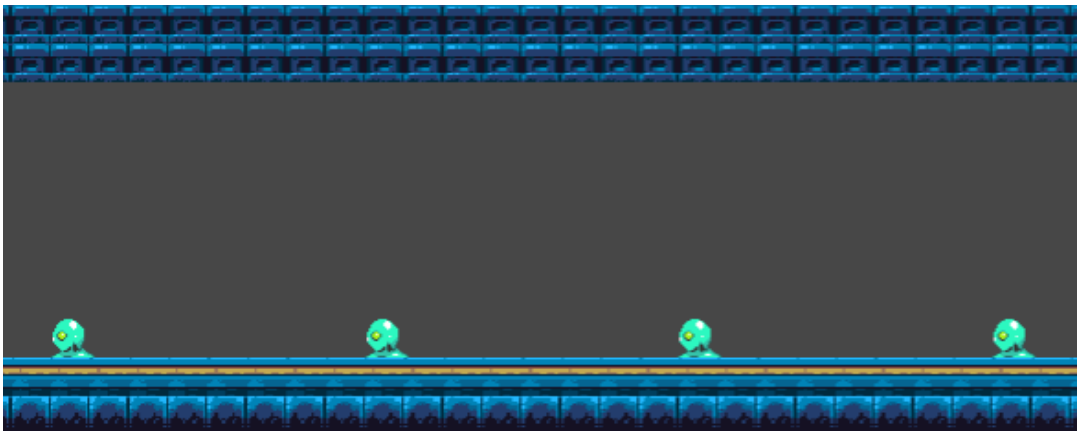
3.6.6. Dizajn druge razine

Druga razina predstavlja prelazak iz dijela špilje koji je dizajna hrama i ulazak više u dubinu špilje. Sastoji se od tri dijela. U prvom dijelu imamo nesimetrično postavljene neprijatelje napravljene od sluzi. Svi neprijatelji ove razine su stacionarni.



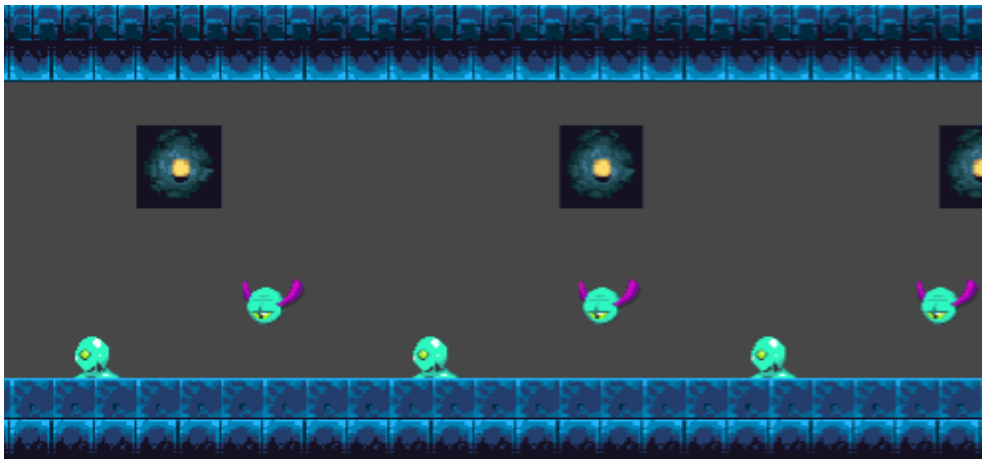
Slika 17. Prvi dio druge razine

Nakon prvog dijela igrač prolazi kroz ubrzanje („Speed Boost“) te glazba razine postane više simetrična. Nakon ubrzanja imamo simetrično postavljene neprijatelje od sluzi.



Slika 18. Drugi dio druge razine

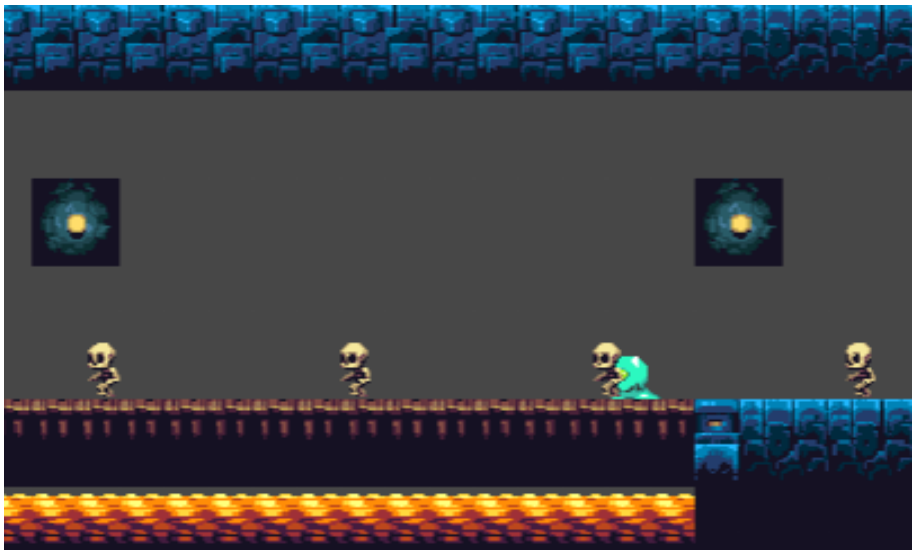
Zadnji dio razine dodaje i leteće neprijatelje s ubrzanjem glazbe. Uz već postojeće ubrzanje igrača dio postaje veoma izazovan.



Slika 19. Treći dio druge razine

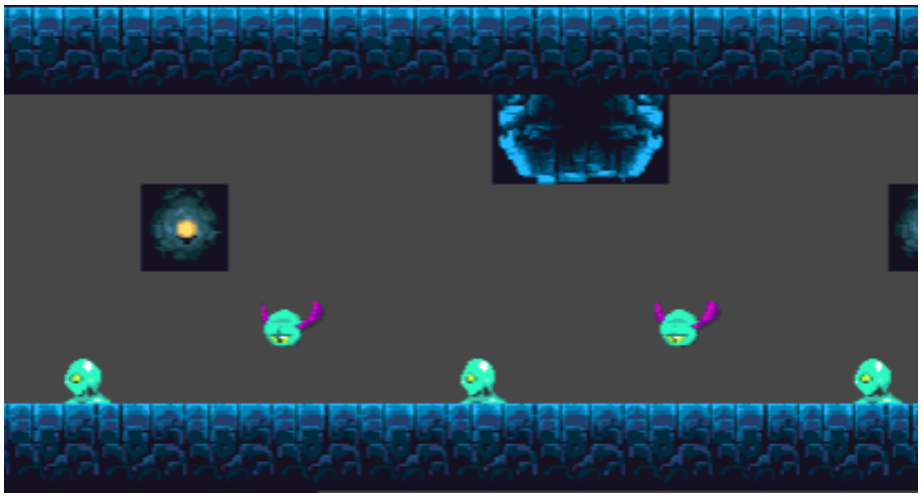
3.6.7. Dizajn treće razine

Treća razina je najduža te je zamišljena kao same dubine špilje. Prvi dio se sastoji od kretajućih kosturova i stacionarnih neprijatelja od sluzi. Kosturi su simetrični dok neprijatelji od sluzi nisu. Na slici ispod izgleda kao da su neprijatelj od sluzi i kostur na istom mjestu ali s tim da kostur nije stacionaran neće biti na istom mjestu kada igrač dođe do njih.



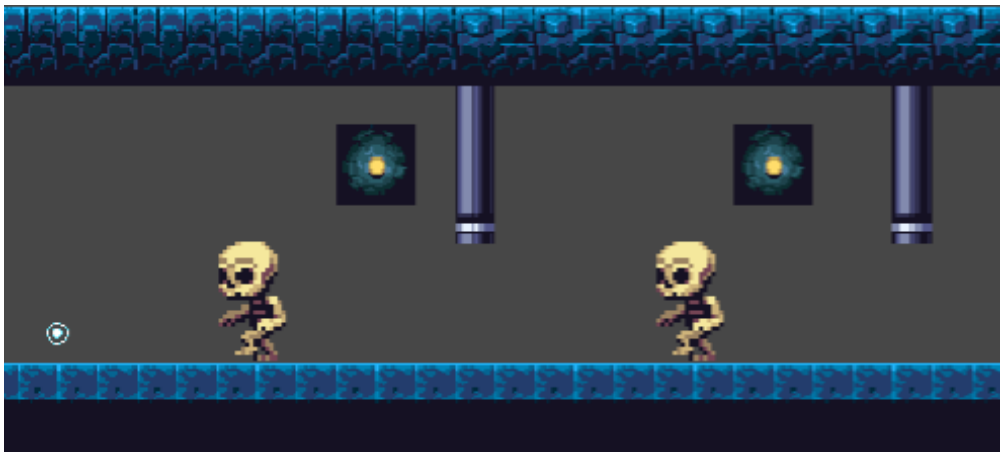
Slika 20. Prvi dio treće razine

U drugom dijelu igrač kupi ubrzanje te dolazi do stacionarnih letećih očiju i neprijatelja od sluzi. Potrebno je preskakati neprijatelje od sluzi i klizati ispod letećih očiju.



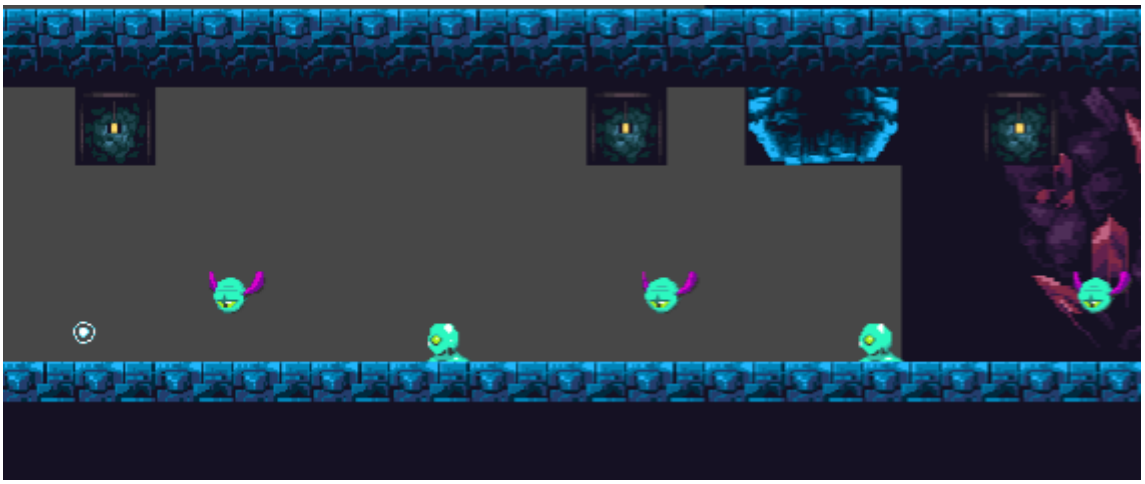
Slika 21. Drugi dio treće razine

Na trećem dijelu igrač kupi kuglu te mu se mjenja način kretnje. U ovom dijelu razine igrač mora izbjegavati velike kosturove i stubove na plafonu.



Slika 22. Treći dio treće razine

U zadnjem dijelu igrač ponovno kupi kuglu te je potrebno izbjegavati leteće neprijatelje i neprijatelje napravljene od sluzi.



Slika 23. Četvrti dio treće razine

4. Zaključak

Video igra „RythmUp“ je izrađena u Unity alatu koristeći C# skripte. Korišteno je 2D okruženje za izradu. Cilj projekta je bio komponiranje C# skripti u interakciji sa razinom i igračem. Iz tog razloga su se koristili već postojeći paketi za teksture.

Sama video igra je ritmični platformer sa tri razine. Cilj igre je izbjegavati neprijatelje i prepreke u ritmu glazbe. Postoje različiti objekti u igri koji mjenjanu način kretanja. Prvi objekt je kugla koja mjenja igračev stil igre tako da umjesto skokova tipkom za skok mjenja smjer gravitacije. Drugi objekt je objekt za ubrzanje kojim se pobrzava sama kretnja igrača.

Cilj samog projekta je bio uvod u izradu 2D platformera u Unity alatu. Smatram da sam završetkom ovog projekta uspješno naučio uvodnu razinu izrade platformera u Unityu.

Popis literature

[1] *Unity (game engine)* (kolovoz 2024.) U Wikipedia. Preuzeto 8.8.2024. s https://en.wikipedia.org/wiki/Unity_(game_engine)

[2] *Unity Techologies* (kolovoz 2024.) U Wikipedia. Preuzeto 8.8.2024. s https://en.wikipedia.org/wiki/Unity_Technologies

[3] *Super Grotto Escape Pack* (kolovoz 2024.) U Unity Asset Store, Preuzeto 1.8.2024 s https://assetstore.unity.com/packages/2d/environments/super-grotto-escape-pack-238393

[4] *Loader – Chris Huelsbeck (Turrican Soundtrack Vol.2)* U Soundcloud, Preuzeto 1.8.2024. s https://soundcloud.com/chris_huelsbeck/loader-turrican-1?in=chris_huelsbeck/sets/turrican-soundtrack-5

[5] *Base After Base – DJVI (Geometry Dash Songs)* U Soundcloud, Preuzeto 1.8.2024. s https://soundcloud.com/djviofficial/dj-v-i-base-after-base?in=jet-engine-56195977%2Fsets%2Fgeometry-dash-soundtrack-all

[6] *Samurai Techno – fizzd (Rythm Doctor Soundtrack)* U Soundcloud, Preuzeto 1.8.2024. s https://soundcloud.com/fizzd/samurai-techno?in=ethan-noobgt/sets/rhythm-doctor

Popis slika