Izrada videoigre žanra puzzle u programu Unity

Jandras, Matija

SVEUČILIŠTE U ZAGREBU
GRAFIČKI FAKULTET ZAGREB

MATIJA JANDRAS

IZRADA VIDEOIGRE ŽANRA PUZZLE U PROGRAMU UNITY

DIPLOMSKI RAD

Zagreb, 2022.

SVEUČILIŠTE U ZAGREBU

GRAFIČKI FAKULTET ZAGREB

Smjer: tehničko-tehnološki

MATIJA JANDRAS

IZRADA VIDEOIGRE ŽANRA PUZZLE U PROGRAMU UNITY

DIPLOMSKI RAD

Mentor:

Prof. dr. sc. Lidija Mandić

Student:

Matija Jandras

Zagreb, 2022.

SAŽETAK

Tema diplomskog rada je izrada videoigre žanra puzzle. U teorijskom dijelu rada prikazan je razvoj videoigara kroz povijest, kao i povijest uređaja na kojima se pokreću videoigre, te je opisan program u kojem se igra izrađivala. Videoigra se sastoji od nekoliko razina u kojima se koriste razne zagonetke koje su potrebne za prolaz i završetak razine. Za izradu je korišten program Unity, skripte su pisane u Visual Studio 19, a za potrebe izrade modela i tekstura korišteni su Blender i Photoshop. Eksperimentalni dio sastoji se od opisa postupaka izrade razina, funkcija interaktivnosti s elementima igre, izrada glavnog izbornika i prijelaza između razina.

Ključne riječi:

videoigra, Unity, skripta, animacija

ABSTRACT

This paper describes the development of a video game in the puzzle genre. The theoretical part of the paper shows the evolution of video games through history, as well as the development of devices used to play video games, and describes the program the game was made in. The video game consists of several levels where puzzles are used to pass a level and complete it. The game was created using the program Unity, the scripts were written in Visual Studio 19, whereas Blender and Photoshop were used to design models and textures. The experimental part of the paper consists of the description of steps in creation of levels, functions of interactivity with game elements, design of the main menu and transitions between levels.

Keywords: video game, Unity, animation, script

SADRŽAJ

1. UVOD

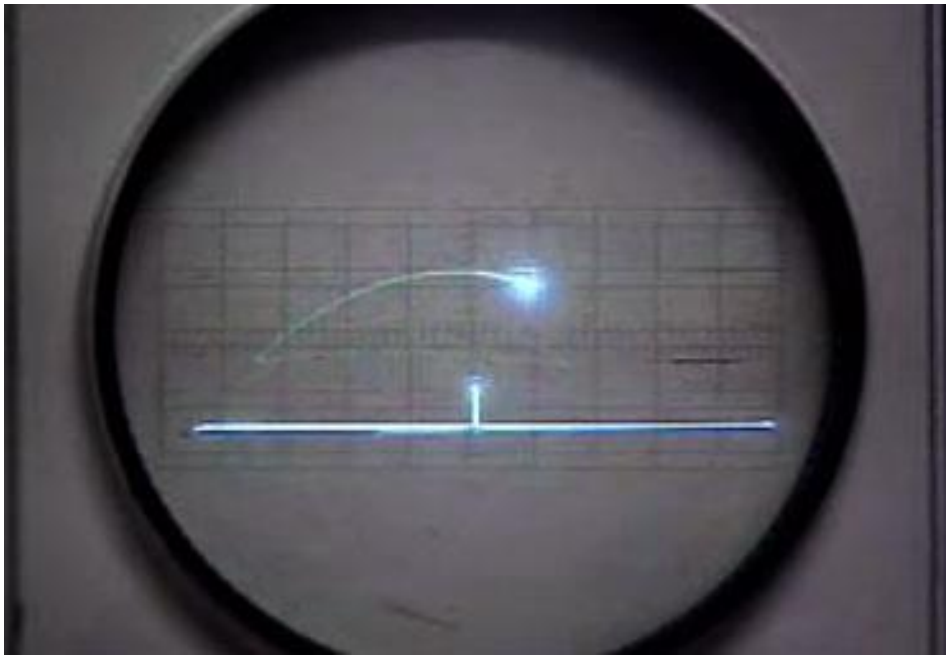
Kroz godine videoigre su se iz oblika zabave napravljenog na analognom računalu razvile u jedan od najpopularnijih načina zabave i interakcije s drugim korisnicima. Iako su videoigre u početku bile veoma jednostavne i koristile se za zabavu, programeri su prepoznali potencijal i postupno razvijali sve kompleksnije i realnije igre te je industrija videoigara postala jedna od danas najunosnijih.

Cilj ovog rada je prikazati mogućnosti programa Unity na primjeru izrade videoigre. Unity je jedan od najpopularnijih *game enginea* koji se koriste za izradu videoigara, ali može se koristiti i za izradu animacija u filmskoj industriji. Jedan od glavnih razloga takve popularnosti jest jednostavnost korištenja. Privlačan je početnicima, ali koristi ga i velik broj profesionalnih studija koji se bave izradom videoigara.

Izrada videoigre, korištene skripte i animacije razrađene su korak po korak u tekstualnom obliku i popraćene slikama.

2. POVIJEST VIDEOIGARA

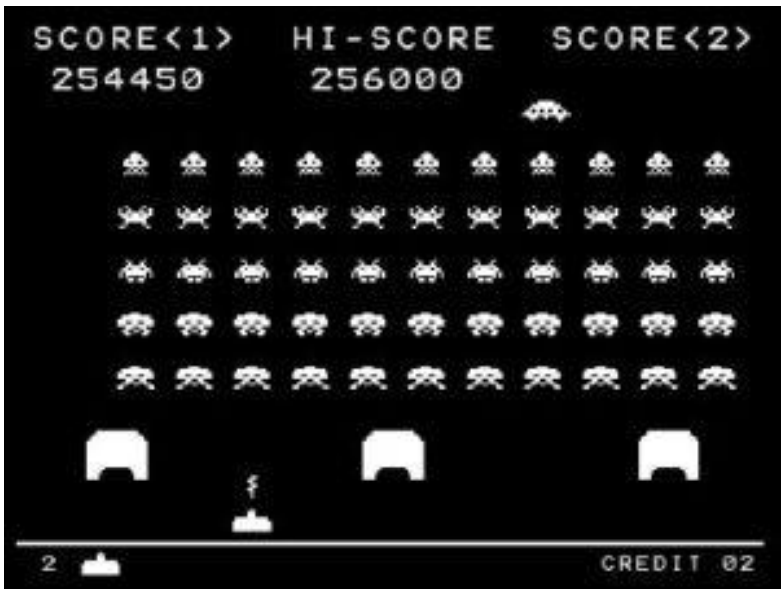
Povijest videoigara usko je povezana s razvojem računala. Od najranijih dana računala, programeri nisu samo tražili nove načine zabave nego su pronašli praktične razloge kako predstaviti potencijalne mogućnosti nove tehnologije. Tennis for Two (1958.), prva videoigra koje je napravljena isključivo u svrhu zabave, napravljena je na analognom računalu povezanom na ekran s osciloskopom. Predstavljena je na trodnevnoj izložbi u New Yorku i izazvala je veliku znatiželju među posjetiteljima. Iako je nakon toga zaboravljena smatra se prvom napravljenom videoigrom [1].



Slika 1. Prikaz videoigre Tennis for Two

(*https://videogamehistory.fandom.com/wiki/Tennis_for_Two*)

1972. godine osnovana je tvrtka Atari koja je dominirala industrijom videoigara iduće desetljeće te je razvila igru Pong koja je postala hit. Pong je načinom izvođenja igre podsjećao na svog prethodnika (Tennis for Two). Cilj igre bio je da se ne promašiti loptica kako bi se ostvario najbolji rezultat, a igra je bila predviđena za dvije osobe. Za razliku od svog prethodnika, Atari je za Pong integrirao računalo sa zaslonom u kutiju koja je imala utor za novčiće te je tako nastao prvi automat za videoigre koji je bio dostupan široj javnosti. Zlatno doba arkada nagoviješta izlazak igre Space Invaders (1976.) koja omogućava igračima da troše svoj džeparac na automatima za videoigre [4].



Slika 2. Space Invaders

(https://www.bespoke-arcades.co.uk/blog/space-invaders)

Razvojem računalne tehnologije i mikroprocesora rastu mogućnosti industrije videoigara te je Atari još jednom uspio lansirati konzolu Atari 2600, koju je kupilo više od 30 milijuna ljudi, a omogućavala je korisnicima pristup širokom spektru videoigara jer nije bila ograničena na samo jednu igru. Dostupne videoigre nalazile su se na kazetama, imale su jednostavnu grafiku i naraciju, a različite težine igre i sistem bodovanja postale su specifične karakteristike koje su poticale igrače da pokušaju nadmašiti najbolji rezultat [2].

Slika 3. Konzola Atari 2600

(https://spectrum.ieee.org/the-consumer-electronics-hall-of-fame-atari-2600)

2.1. Pad i ponovni uspon

Godine 1983. sjevernoamerička industrija videoigara doživjela je pad zbog brojnih čimbenika, uključujući prezasićeno tržište igraćih konzola, konkurenciju računalnih igara i mnoštvo razvikanih igara niske kvalitete poput igre E.T., koju je Atari utemeljio na istoimenom filmu, a mnogi ju i dan danas smatraju najgorom igrom ikad stvorenom. Pad koji je trajao nekoliko godina doveo je do bankrota nekolicine tvrtki u industriji videoigara i računala [2].

1985. započinje lagani oporavak dolaskom japanskog Nintendo Entertainment Systema (NES) u Sjedinjene Američke Države. NES je u odnosu na prethodnike imao napredniju 8-bitnu grafiku, boje i zvuk.

Slika 4. Nintendo Entertainment System

(https://en.wikipedia.org/wiki/Nintendo_Entertainment_System)

Nintendo je osnovan 1889. godine kao tvrtka koja proizvodi igraće karte. Od tog vremena izdali su niz važnih franšiza videoigara koje postoje i danas poput Super Mario Bros., Legend of Zelda i Metroid.

Slika 5. Legend of Zelda na NES konzoli

(https://www.theverge.com/2018/10/10/17959040/nintendo-switch-online-legend-of-zelda-nes-special-edition)

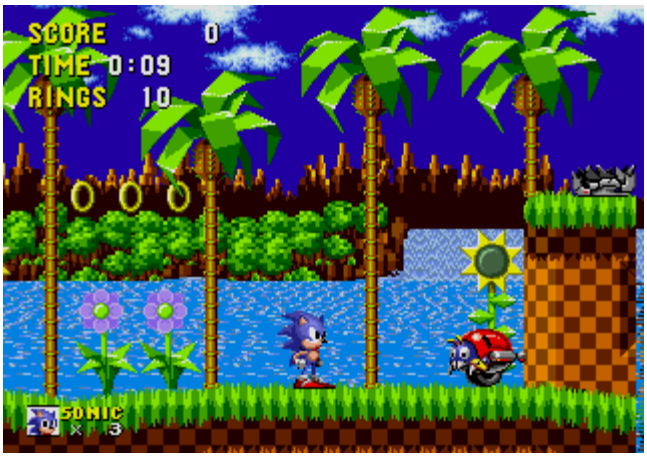
Osim toga Nintendo je uveo niz propisa o igrama trećih strana koje se razvijaju za njihov sustav, a neke od franšiza koje su izdali vanjski suradnici su Mega Man (CAPCOM), Final Fantasy (Square) i Dragon Quest (Enix). Godine 1989. Nintendo je izdavanjem 8-bitnog uređaja, Game Boya, dodatno popularizirao ručno igranje, a igra koja je često dolazila u paketu s konzolom bio je Tetris. Nintendo je do danas izdao nekoliko uspješnih nasljednika Game Boya poput Nintendo DS (2004.) i Nintendo 3DS (2011.) [2].

Slika 6. GameBoy

(www.engadget.com/2019-04-21-nintendo-game-boy-30th-anniversary.html)

2.2. Konkurencija među konzolama

Sega je kao odgovor na NES konzolu 1989. godine u Americi izdala svoju 16-bitnu konzolu Genesis koja je nasljednik ne tako uspješnog Sega Master Systema iz 1986. godine. Genesis je u odnosu na NES bio nadmoćniji sustav, a pametnim marketingom i izdavanjem igre Sonic the Headgehog (1991.), Genesis je napravio značajan napredak u odnosu na svog starijeg rivala.



Slika 7. Sonic the Hedgehog (1991.)

(https://en.wikipedia.org/wiki/Sonic_the_Hedgehog_%281991_video_game%29)

Kao odgovor na to, Nintendo 1991. godine izdaje svoju 16-bitnu Super NES konzolu i tako pokreće takozvani rat konzola. Od sredine 1990-ih objavljen je velik broj igara na obje konzole, uključujući nove franšize poput Street Fightera II i Mortal Kombata, borilačkih igara koje su prikazivale nasilje i krv na verziji za Genesis [2].

Slika 8. Mortal Kombat (1993.)

(https://sagamer.co.za/blast-from-the-past-mortal-kombat-sega-mega-drive/)

Zbog prikaza nasilnih scena, Sega je kao odgovor na nasilnu igru sastavila vijeće za ocjenjivanje videoigara kako bi svaka igra dobila dobno dopuštenje prije izdavanja na konzole. Vijeće je kasnije preoblikovano u odbor za ocjenjivanje zabavnog softvera u industriji videoigara koji i danas ocjenjuju videoigre na temelju sadržaja. Sredinom 1990-ih počele su se pojavljivati prve adaptacije videoigara na velikom platnu izdavanjem igranog filma baziranog na Super Mario Bros (1993.), a u sljedeće dvije godine objavljeni su i Street Fighter i Mortal Kombat. Od tada su objavljeni brojni filmovi temeljeni na videoigrama, ali je uglavnom riječ o manje kvalitetnim ostvarenjima. Iako je Genesis zbog veće zbirke igara i niže cijene u Sjevernoj Americi preskočio SNES, u Japanu nije postigao sličan uspjeh.

2.3. Uspon 3D igara

Daljnjim razvojem računalne tehnologije započinje peta generacija videoigara, a time dolazi do pojave prvih trodimenzionalnih igara. Sega je 1995. godine izdala Saturn, 32-

bitnu konzolu, koja je koristila CD umjesto kazeta i trebala je konkurirati Sonyjevom Playstationu. Sljedeće godine Nintendo je izbacio svoj 64-bitni sustav, Nintendo 64, temeljen na kazetama [2].

Slika 9. Konzola Nintendo 64

(https://en.wikipedia.org/wiki/List_of_Nintendo_64_games)

Iako su Sega i Nintendo izdali poveći broj visoko ocijenjenih 3D naslova poput Virtual Fightera (Sega) i Super Mario 64 (Nintendo), nisu uspjeli parirati Sonyju koji je imao jaku podršku trećih strana što im je osiguralo mnogobrojne ekskluzivne naslove. Sony je svoju dominaciju nastavio i u idućoj generaciji objavljivanjem konzole Playstation 2 (2000.) koja je podržavala igre s originalnog Playstationa, a postala je najprodavanija konzola svih vremena.

Slika 10. Sony PlayStation 2

(https://playstation.fandom.com/wiki/PlayStation_2)

Playstation 2 je prva konzola koja je koristila DVD-ove, a nadmašila je Segin Dreamcast (1999.), Nintendov Gamecube (2001.) i Microsoftov Xbox (2001.). Iako se za Dreamcast smatra da je jedna od najbolje napravljenih konzola i da je ispred svoga vremena zbog mogućnosti online igranja, bio je komercijalni neuspjeh i 2001. Sega je odustala od daljnje proizvodnje konzola te postala softverska tvrtka. Studiji za dizajn igara izbacili su inovativnije ideje, a povijest je često služila kao inspiracija za igru: u igri Age of Empires (1997.) igrači su izgradili čitave civilizacije, u igri Command & Conquer (1995.) vodili su rat. U Tomb Raideru (1996.) tragali su za povijesnim artefaktima zajedno s Larom Croft; u Monkey Islandu (1990.), tražili su gusarsko blago [1].

Slika 11. Age of Empires (1997.)

(https://gamerinfo.net/game/age-of-empire/)

2.4. Moderno doba

Moderno doba igranja u visokoj rezolucije započinje 2005. godine kad su objavljeni Xbox 360 (Microsoft), Sony Playstationa 3 i Nintendo Wii.

Slika 12. Konzole PlayStation 3 i xBox 360

(https://www.engadget.com/2008-11-12-ask-engadget-hd-best-gaming-movie-console-xbox-360-or-playst.html)

Playstation 3, jedini sustav koji je u to vrijeme reproducirao Blu-ray-ove bio je sam po sebi uspješan, međutim Sony se prvi puta suočio s oštrom konkurencijom. Xbox 360 imao je slične grafičke mogućnosti kao Playstation 3, a posebno je hvaljen zbog svog ekosustava online igranja. Osim toga, osvojio je mnogo više Game Critics nagrada u odnosu na svoju konkurenciju. 2007. godine izlazi Microsoft Kinect, sustav snimanja pokreta koji nudi drugačiji način igranja videoigara, no nikada nije bio previše popularan među najvećim ljubiteljima videoigara. U usporedbi s Playstationom 3 i Xboxom, Nintendo Wii bio je tehnološki inferiorniji, ali je svejedno pobijedio svoju konkurenciju u prodaji. Drugačijim pristupom igranju zbog kontrolera osjetljivih na pokrete, postao je popularniji među širom javnosti jer je igranje bilo aktivnije nego prije [3].

Po prvi put igre, kao što je World of Warcraft (2004), igrane su uglavnom na internetu.

Slika 13. World of Warcraft (2004.)

(https://www.nyfa.edu/student-resources/5-classic-video-games-that-changed-everything-from-mario-to-wow/)

Sa svakom godinom desetljeća, tehnologija se također razvijala skokovitim koracima. Snažne grafičke tehnologije učinile su svjetove videoigara još realističnijima. Uvedene su igre otvorenog svijeta u kojima igrači sami istražuju izmišljene svjetove i mogu slobodno određivati tijek igre. Ili sami stvaraju virtualni svijet, kao u pješčaniku: ovaj koncept igranja naziva se 'sandbox'. Desetljeće je proizvelo brojne bestselere, među kojima su najvažniji The Sims (2000), Grand Theft Auto: San Andreas (2004), Super Mario Galaxy (2007) i Minecraft (2010) [1].

Slika 14. Minecraft (2010.)

(https://www.rockpapershotgun.com/a-long-lost-minecraft-alpha-has-been-found)

U prvih deset godina 21. stoljeća videoigre su se proširile na platforme društvenih mreža poput Facebooka i mobilnih uređaja poput iPhonea te postale dostupnije nego ikada. U vlaku, čekajući autobus ili prije spavanja: u svakom slobodnom trenutku dostupna je videoigra. Iako kompulzivno igranje nije novost, broj ovisnika o igricama raste širenjem pametnih telefona. Međutim, za većinu onih koji igraju igre su bezopasna zabava. Igre kao što je Red Dead Redemption 2 (2019) koje, zahvaljujući sofisticiranim dijalozima i emotivnim pričama, ne samo da pružaju sate zabave u igricama, već također imaju drugačiji tijek u skladu s odlukama igrača, pa se mogu igrati uvijek iznova, stavljaju se na osobnim računalima, PlayStationu i Xboxu [1].

Slika 15. Red Dead Redemption (2019.)

(https://press-start.com.au/news/xbox/2018/04/11/red-dead-redemption-now-runs-4k-xbox-one-x-looks-beautiful/)

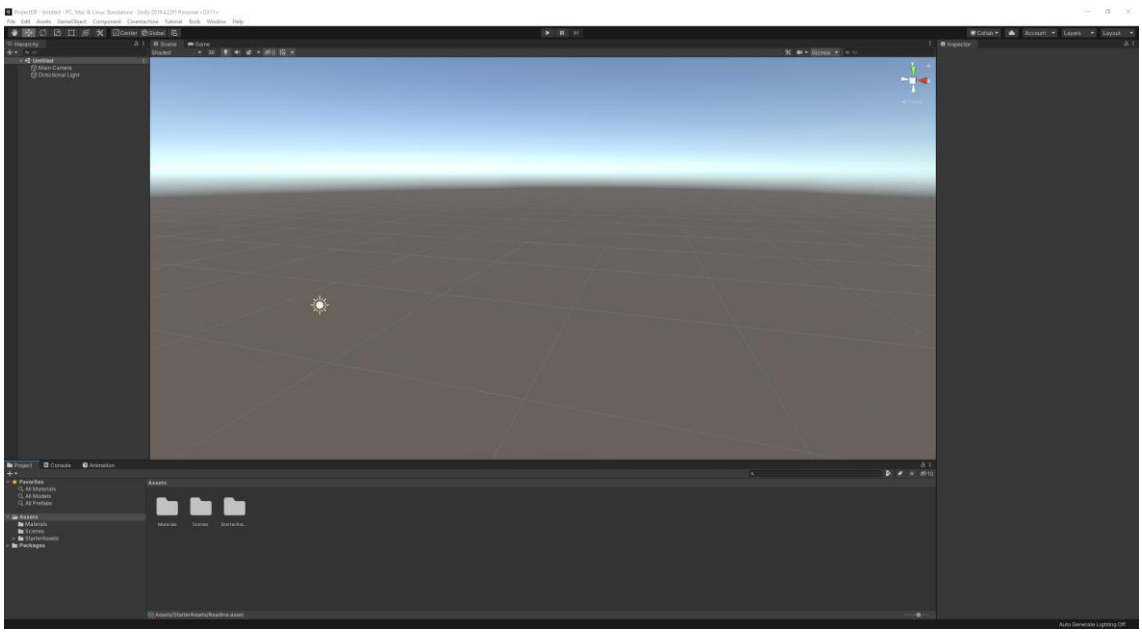
Osma generacija videoigara započela je izdavanjem Nintendo Wii U 2012. godine, nakon čega su uslijedili Playstation 4 i Xbox One 2013. godine. Iako je Wii U sadržavao daljinski upravljač sa zaslonom na dodir koji je omogućio igranje bez TV-a, doživio je komercijalni neuspjeh u odnosu na konkurenciju te je 2017. godine ukinut. Sony je 2016. godine izdao snažniju verziju svoje konzole, Playstation 4 Pro, prvu konzolu sposobnu za igranje u 4K rezoluciji, a Microsoft je spremno uzvratio svojom Xbox One X konzolom. Početkom 2017. godine Nintendo je izdao nasljednika Wii U, Nintendo Switch, koji omogućuje igranje i na televiziji i u ruci [2].

3. UNITY SOFTWARE

Unity je *game engine* koji omogućuje razvoj videoigara za više platforma. Koristiti za izradu 2D ili 3D igara, a može se primijeniti i za izradu interaktivnih simulacija. Osim upotrebe u industriji videoigara koristi se i u filmskoj i automobilskoj industriji, građevini i arhitekturu. Osim što je *game engine*, Unity nudi integrirano okruženje za razvoj (IDE), odnosno najvažnije značajke koje bi trebale omogućiti videoigrama da rade ugrađene su u softver. Neke od tih značajki uključuju fiziku, 3D prikaz i detekciju sudara. To uvelike smanjuje opseg posla i prepušta softveru teži dio posla. Još jedna od dodatnih stvari koje Unity čine privlačnim je opcija Asset Store, koja korisnicima omogućuje da postave svoja ostvarenja i na taj način ih učine dostupnima ostalim korisnicima. Unity softver sadrži jedan uređivač koji omogućuje korisnicima da povlačenjem dodaju elemente u scenu i manipuliraju njihovim karakteristikama. Za kodiranje skripti koristi se Microsoft Visual Studio, a kao programski jezik koristi se C# koji je jedan od najboljih programskih jezika za početnike [5].

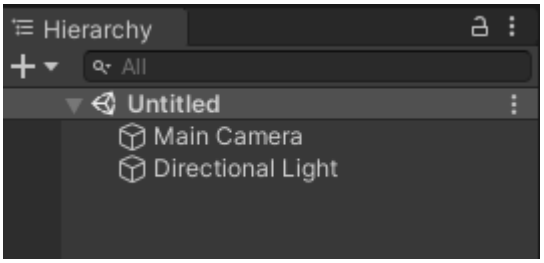
3.1. Sučelje

Pokretanjem softvera Unity otvori se nekoliko prozora koji su potrebni za rad na videoigri (slika 16).



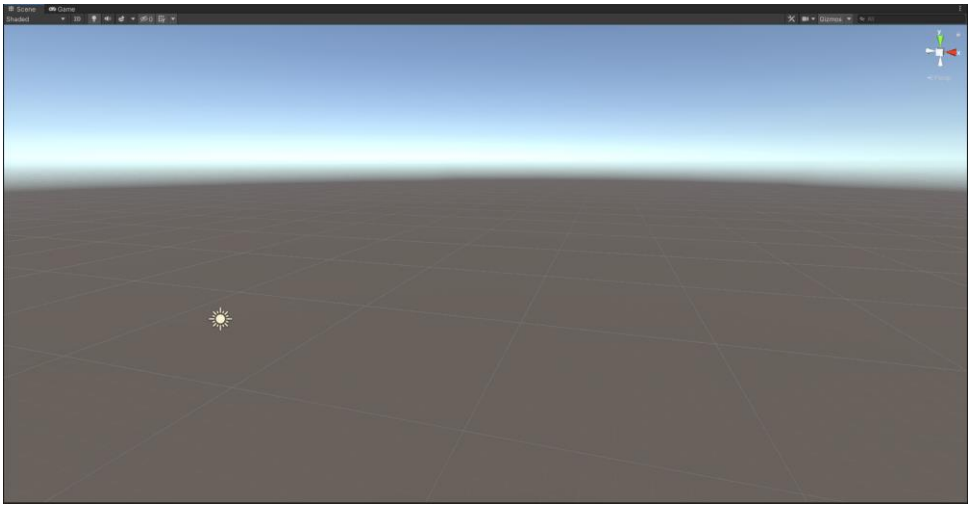
Slika 16. Sučelje softvera Unity

Hijerarhija pokazuje popis svih elemenata (*Game Objects*) koji se nalaze na sceni. To olakšava brzo lociranje i odabir bilo kojeg aspekta igre kako bi se mijenjale njegove karakteristike. Nalazi se na lijevoj strani (slika 17).



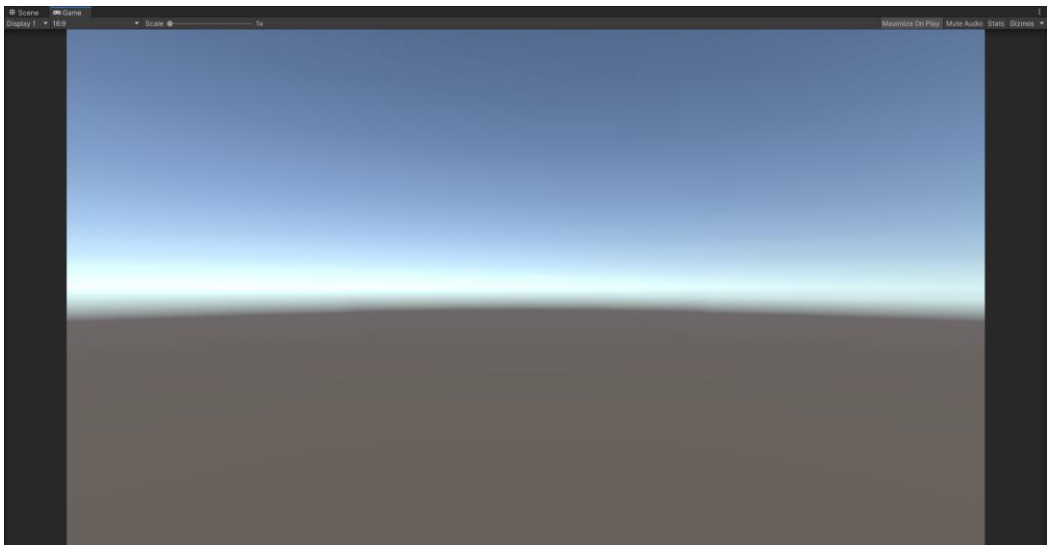
Slika 17. Hierarchy prozor

Scena je najveći prozor koji se nalazi u sredini softvera, a tamo se prikazuje ono na čemu se trenutno radi npr. razina, početni izbornik itd. Mjesto u kojem se dodaju objekti te se na njima mogu mijenjati postavke poput dimenzija, rotacije i položaja (slika 18). Za pristup postavkama objekata na sceni koriste se ikone iznad prozora scene.



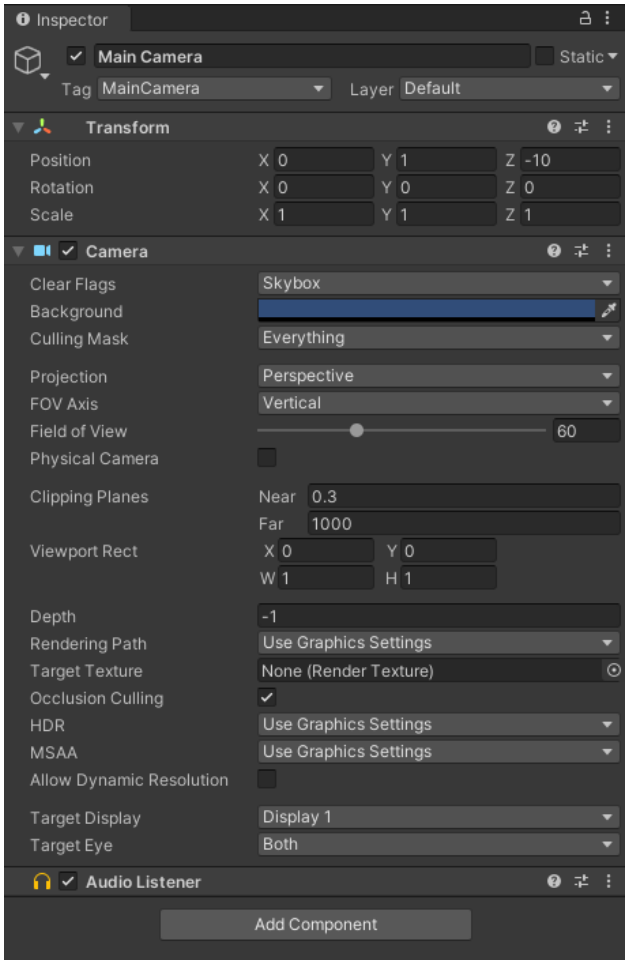
Slika 18. Scene prozor

Game je prozor koji je skriven iza scene, a do njega se dolazi klikom na gumb Game. Prikazuje pogled na scenu kroz kameru i koristi se za testiranje igre (slika 19). Prilikom korištenja prozora nije moguće pomicati objekte i mijenjati njihova svojstva, ali je omogućeno pomicanje lika po sceni ili korištenje elemenata ako se radi o izborniku.



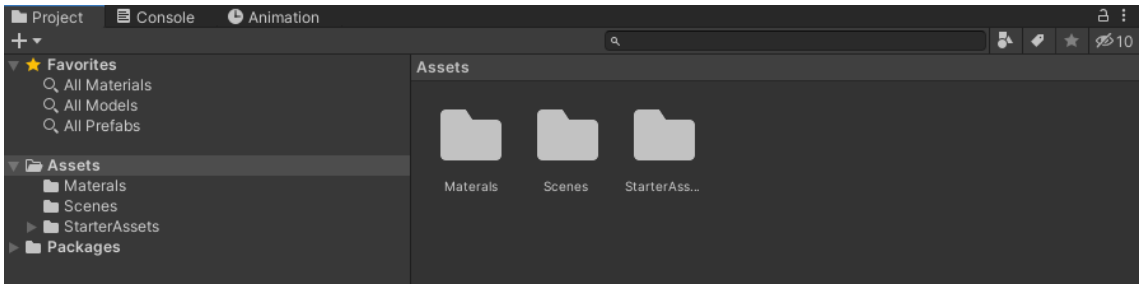
Slika 19. Game prozor

Inspector daje pristup svojstvima odabranog elementa i omogućuje njihovo mijenjanje ili dodavanje komponenti koje služe za interakciju sa ostalim objektima na sceni poput C# skripti ili *collidera*. Nalazi se na desnoj strani ekrana (slika 20).



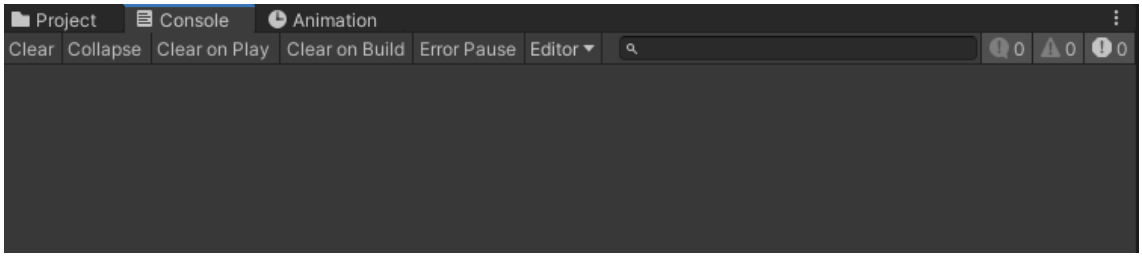
Slika 20. Inspector prozor

Project prikazuje sve datoteke koje se stvaraju igru i prostor gdje se kreiraju C# skripte ili dodaju gotovi 3D modeli (slika 21).



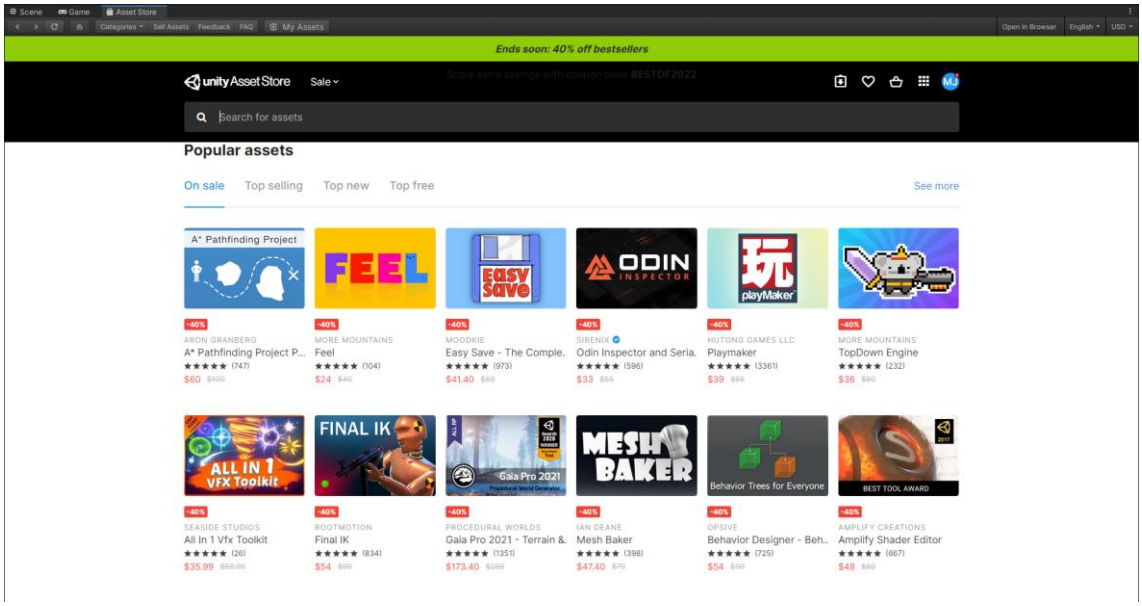
Slika 21. Project prozor

Console je prozor u kojem Unity šalje povratne informacije poput pogrešaka koje su nastale u kodu ili pogrešaka koje se pojavljuju u postavkama objekata. Console prozor se poput Project prozora nalazi na dnu ekrana (slika 22).



Slika 22. Console prozor

Asset Store je prozor u kojem se pristupa elementima ili modelima koje su kreirali drugi korisnici (slika 23).



Slika 23. Asset Store prozor

4. IZRADA 3D MODELA I ANIMACIJA

4.1. Igrač

Element kojim se pomiče po sceni naziva se igrač, a sastoji se od kapsule koja je nevidljiva. Kapsula na sebi ima kameru koja je namještena tako da izgleda kao da se igra iz prvog lica.

4.1.1. Orijentacija

Za početak potrebno je napraviti skriptu za orijentaciju igrača po sceni pomicanjem miša (slika 24).

```
public class PlayerCam : MonoBehaviour
{
    public float sensX;
    public float sensY;

    public Transform Orientation;

    float xRotation;
    float yRotation;

    private void Start()
    {
        Cursor.lockState = CursorLockMode.Locked;

    }

    private void Update()
    {
        float mouseX = Input.GetAxisRaw("Mouse X") * Time.deltaTime * sensX;
        float mouseY = Input.GetAxisRaw("Mouse Y") * Time.deltaTime * sensY;

        yRotation += mouseX;

        xRotation -= mouseY;
        xRotation = Mathf.Clamp(xRotation, -90f, 90f);

        transform.rotation = Quaternion.Euler(xRotation, yRotation, 0);
        Orientation.rotation = Quaternion.Euler(0, yRotation, 0);
    }
}
```

Slika 24. Skripta za orijentiranje igrača

Funkcija *Start* pokretanjem scene zaključava kursor na sredinu ekrana. Varijable xRotation i yRotation dobivaju ulazne informacije kursora te omogućavaju rotaciju po x i y osi. xRotation je rotacija prema gore i dolje pa je potrebno postaviti limit od -90f i 90f koje onemogućava rotaciju od 360 stupnjeva, za razliku od yRotation za koju je takva rotacija potrebna. Klasa Quaternion postavlja kut rotacije kamere na određene osi. Kada je skripta gotova, postavlja se na *Empty Object* koji se postavlja na objekt igrača, a predstavlja orijentaciju (slika 25).

Slika 25. Model igrača i kamera u hijerarhiji

Osim *Orientation* objekta na igraču se nalazi *CameraPos* koji određuje poziciju kamere i predstavlja ono što igrač vidi kada je igra pokrenuta. Da bi kamera mogla pratit igrača, potrebno je napraviti skriptu koja pretvara koordinate kamere u koordinate igrača (slika 26).

```
public class MoveCamera : MonoBehaviour
{
    public Transform cameraPosition;

    private void Update()
    {
        transform.position = cameraPosition.position;
    }
}
```

Slika 26. Skripta kamere kojom prati igrača

4.1.2. Kretanje

Funkcijom *MovePlayer* pomoću klase *Orientation* postavlja se kretanje lika u pravcu u kojem igrač gleda, a za to je još potrebno dodati *AddForce* na *Rigidbody* (slika 27).

```
private void MovePlayer()
{
    moveDirection = orientation.forward * verticalInput + orientation.right * horizontalInput;

    if (grounded)
        rb.AddForce(moveDirection.normalized * moveSpeed * 10f, ForceMode.Force);

    else if (!grounded)
        rb.AddForce(moveDirection.normalized * moveSpeed * 10f * airMultiplier, ForceMode.Force);
}
```

Slika 27. Funkcija MovePlayer

Da igrač ne bi klizio po površini, potrebno je dodati *Drag* koji varijablom *grounded* dodaje trenje na podlogu (slika 28).

```
if (grounded)
    rb.drag = groundDrag;
else
    rb.drag = 0;
```

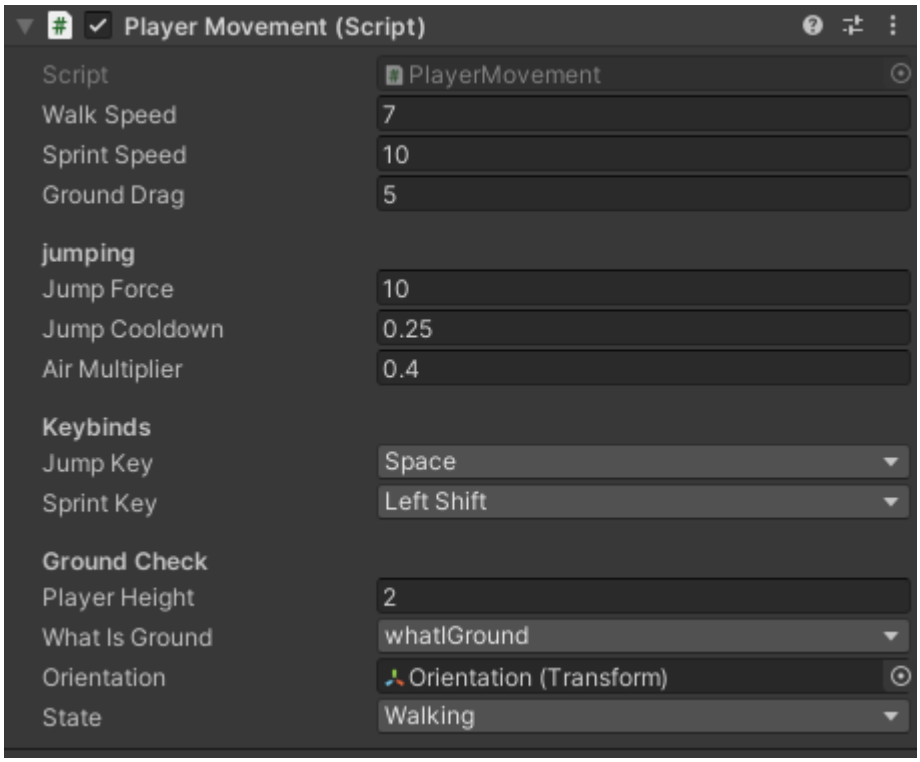
Slika 28. Postavljanje Drag na igrača

Kako bi trenje radilo kako treba, potrebno kreirati *layer* naziva *whatIsGround* koji se stavlja na površine po kojima se igrač kreće. Skakanje je jedna od vrsti kretanja za koju je u skriptu potrebno dodati funkciju *Jump* koja dodaje varijablu *velocity* na *rigidbody* i koja omogućuje pomicanje po y-osi pritiskom na tipku „Space“. Osim skakanja dodaje se mogućnost trčanja, a za to je potrebno napraviti tri stanja koja se mijenjaju pritiskom na tipku. Ukoliko je igrač prizemljen i pritisnuta je tipka za trčanje „Left Shift“, povećava mu se brzina kretanja i odlazi u stanje *sprinting*. Ako je igrač samo prizemljen, onda je u stanju *walking*, a u trećem stanju je samo kada je u zraku prilikom skoka (slika 29).

```
private void StateHandler()
{
    if (grounded && Input.GetKey(sprintKey))
    {
        state = MovementState.sprinting;
        moveSpeed = sprintSpeed;
    }

    else if (grounded)
    {
        state = MovementState.walking;
        moveSpeed = walkSpeed;
    }

    else
    {
        state = MovementState.air;
    }
}
```

Slika 29. StateHandler funkcija

Gotovu skriptu *PlayerMovement* potrebno je dodati na objekt *Player* i podesiti postavke u *inspector* prozoru (slika 30).



Slika 30. Inspector prozor sa PlayerMovement skriptom

4.1.3. Podizanje i spuštanje objekata

Određene razine zahtijevaju da igrač koristi kocku za aktiviranje *triggera*. Da bi se kocka mogla uhvatiti, potrebno je napraviti skriptu za dizanje i spuštanje objekata. Funkcije *PickupObject* i *DropObject* omogućuju podizanje i spuštanje objekta. Da bi se objekt mogao podići, na njega je potrebno postaviti *Rigidbody*. Ukoliko se na objektu nalazi *Rigidbody*, funkcijom *PickupObject* igrač postavlja objekt na *holdArea* što predstavlja *Empty Object* koji označava mjesto na kojem će se nalaziti objekt kada ga igrač podigne, te mu isključuje postavke poput gravitacije i trenja. Funkcijom *DropObject* igrač ispušta objekt te mu vraća sve postavke koje je izgubio (slika 31).

```
void PickupObject(GameObject pickObject)
{
    if (pickObject.GetComponent<Rigidbody>())
    {
        heldObjectRB = pickObject.GetComponent<Rigidbody>();
        heldObjectRB.useGravity = false;
        heldObjectRB.drag = 10;
        heldObjectRB.constraints = RigidbodyConstraints.FreezeRotation;

        heldObjectRB.transform.parent = holdArea;
        heldObject = pickObject;
    }
}

void DropObject()
{

        heldObjectRB.useGravity = true;
        heldObjectRB.drag = 1;
        heldObjectRB.constraints = RigidbodyConstraints.None;

        heldObject.transform.parent = null;
        heldObject = null;
}
```

Slika 31. Funkcije PickupObject i DropObject

Ukoliko je udaljenost od objekta veća od 0.1f, funkcijom MoveObject igrač omogućava pomicanje objekta koji drži (slika 32).

```
void MoveObject()
{
    if(Vector3.Distance(heldObject.transform.position, holdArea.position) > 0.1f)
    {
        Vector3 moveDirection = (holdArea.position - heldObject.transform.position);
        heldObjectRB.AddForce(moveDirection * pickupForce);
    }
}
```

Slika 32. Funkcija MoveObject

Da bi ta skripta bila ispravna, potrebno je u funkciji *Update* dodati da pritiskom na desnu tipku miša (RMB) igrač podigne objekt ili u slučaju da nešto nosi, spušta taj objekt i može podići drugi (slika 33).

```
private void Update()
{
    if (Input.GetMouseButtonDown(0))
    {
        if (heldObject == null)
        {
            RaycastHit hit;
            if (Physics.Raycast(transform.position, transform.TransformDirection(Vector3.forward), out hit, pickupRange))
            {
                PickupObject(hit.transform.gameObject);
            }

        }
        else
        {
             DropObject();
        }
    }
    if (heldObject != null)
    {
        MoveObject();
    }
}
```

Slika 33. Funkcija Update

4.1.4. Respawn

Zadnja mogućnost koju igrač posjeduje je *Respawn* na mjestu na kojem se nalazio na početku razine ukoliko padne s platforme prilikom prelaska. Skripta *Respawn* u funkciji *Update* provjerava nalazi li se igračeva pozicija ispod pozicije mjesta na kojem se stvorio, te funkcijom *RespawnPoint* vraća igrača na *spawnPoint* koji je određen *Empty Objectom* koji se nalazi na sceni (slika 34).

```
public class Respawn : MonoBehaviour
{

    [SerializeField] GameObject player;
    [SerializeField] Transform spawnPoint;
    [SerializeField] float spawnValue;

    void Update()
    {
        if(player.transform.position.y < -spawnValue)
        {
            RespawnPoint();
        }
    }

    void RespawnPoint()
    {
        transform.position = spawnPoint.position;
    }
}
```

Slika 34. Skripta za Respawn igrača

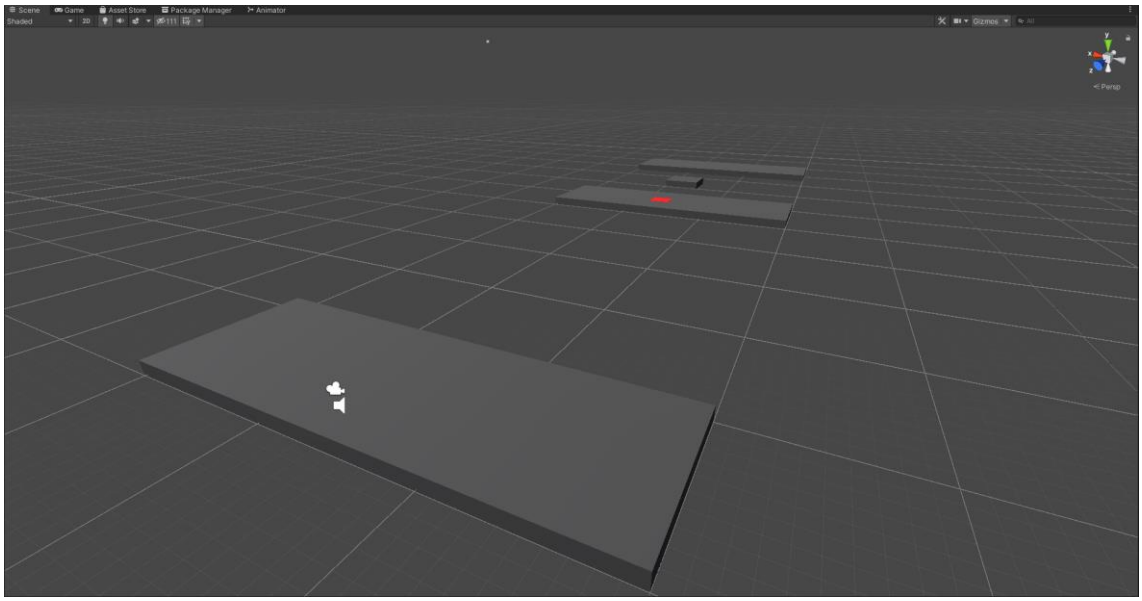
4.2.Izrada razina

Videoigra se sastoji od dvije vrste razina, jedna sa bazira na pokretnim platformama, a druga na vratima i kockama. Razina s platformama je otvorenog tipa pa sadržava Skybox slike svemira i dinamične ili statične platforme po kojima je potrebno doći do kraja razine. Druga vrsta razine je zatvorenog tipa i sadrži nekoliko prostorija u kojima se nalaze kocke za otvaranje vrata i prelazak u druge prostorije.

Za izradu razina uglavnom se koriste modeli izraženi s pomoću *add-ona* Pro Builder, a poneki elementi su napravljeni u programu Blender, poput vrata, kocke i pokretnih platforma.

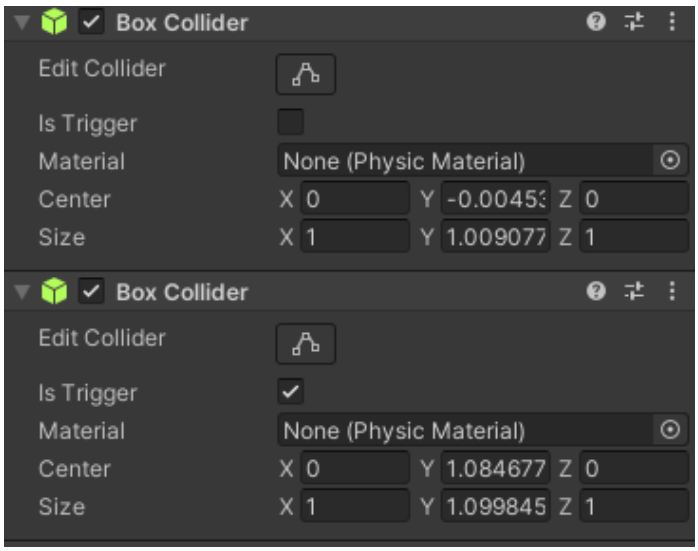
4.2.1. Otvorena razina

Izrada otvorenih razina započinje dodavanjem *Cube* u hijerarhiju, čije se dimenzije potom mijenjaju, te se taj postupak ponavlja nekoliko puta (slika 35).



Slika 35. Postavljanje objekata cube na scenu

Platforme koje se koriste u ovoj razini nisu uvijek stacionarne stoga je potrebno napraviti dodatne platforme koje se vizualno razlikuju. Model pokretne platforme izrađen je u Blenderu pa ga je potrebno dodati u *Project* prozor iz kojega se odvlači u hijerarhiju. Pokretnim platformama potrebno je dodati dva *Box Collidera*: jedan da se kroz nju ne propada i drugi koji igrača povezuje s platformom kada je na njoj (slika 36).



Slika 36. Dva collidera na pokretnim platformama

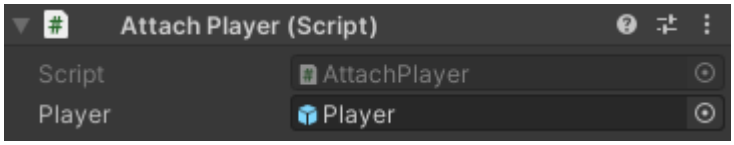
Da bi se igrač povezao s platformom potrebno je napraviti skriptu koja to omogućava (slika 37).

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

public class AttachPlayer : MonoBehaviour
{

    public GameObject Player;

    private void OnTriggerEnter(Collider other)
    {
        if(other.gameObject == Player)
        {
            Player.transform.parent = transform;
        }

    }

    private void OnTriggerExit(Collider other)
    {
        if (other.gameObject == Player)
        {
            Player.transform.parent = null;
        }
    }
}
```

Slika 37. AttachPlayer skripta

Metoda *OnTriggerEnter* provjerava je li *GameObject* koji je ušao u *Collider*, onaj koji predstavlja *Player* te ako je, platformu postavlja za njegova *Parenta* te se *Player* pomiče kako se pomiče i platforma na kojoj stoji. Metoda *OnTriggerExit* radi na izlasku iz *Collidera* i ukoliko se radi o *Playeru*, platforma više nije *Parent* te se *Player* može nastaviti kretati sam.

Navedena skripta se u *Inspector* prozoru stavlja na pokretnu platformu, a potom se igrač treba postaviti na *Player* da bi skripta znala o kojem se objektu radi kada dolazi do *Collidera* (slika 38).



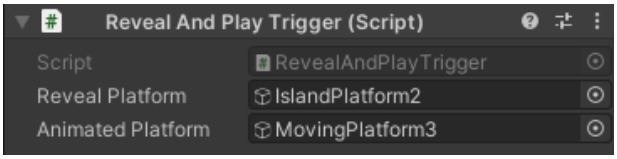
Slika 38. Attach Player skripta na platformi

Scena sadrži jedan *Pressure Plate* koji pokreće animaciju pokretne platforme i otkriva jednu sakrivenu platformu koja je potrebna za prelazak, a na toj platformi se nalazi *Trigger* koji otkriva vrata koja predstavljaju kraj. *Pressure Plate* je napravljena od *Cube* koja se doda u hijerarhiju, te joj se dimenzije namjeste u *Inspector* prozoru, te na sebi sadrži *Box Collider* koji služi kao *Trigger* zatim se pomoću skripte aktivira animacija i otkrije platforma (slika 39).

```
public class RevealAndPlayTrigger : MonoBehaviour
{
    public GameObject RevealPlatform;
    public GameObject AnimatedPlatform;

    void OnTriggerEnter(Collider other)
    {
        if (other.tag == "Player")
        {
            AnimatedPlatform.GetComponent<Animator>().Play("Platform3");
            RevealPlatform.SetActive(true);
        }
    }
}
```

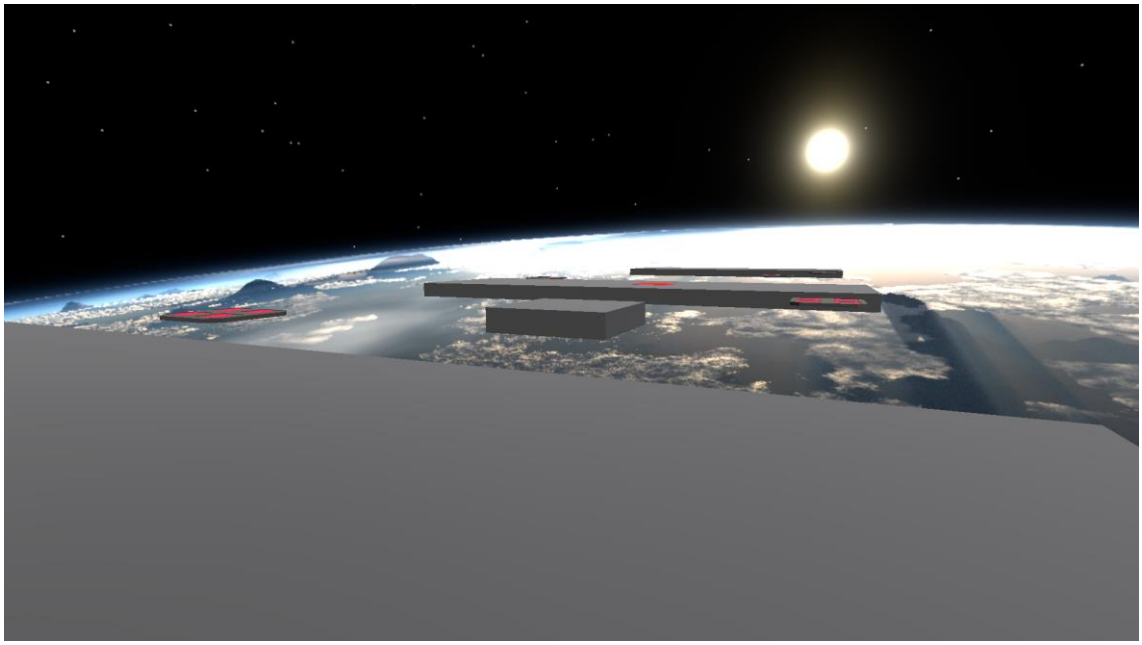
Slika 39. Skripta koja se nalazi na *Pressure Plateu*

Metoda *OnTriggerEnter* funkcionira na isti način kao i u prethodnoj samo sada pokreće animaciju na prvom objektu tako što pokrene animaciju koja se nalazi na komponenti *Animator*, a aktivira drugi objekt koji je skriven i otkrije. Potom je potrebno dodati objekte u *Inspector* prozoru (slika 40).



Slika 40. Skripta koja pokreće animaciju i otkriva platformu

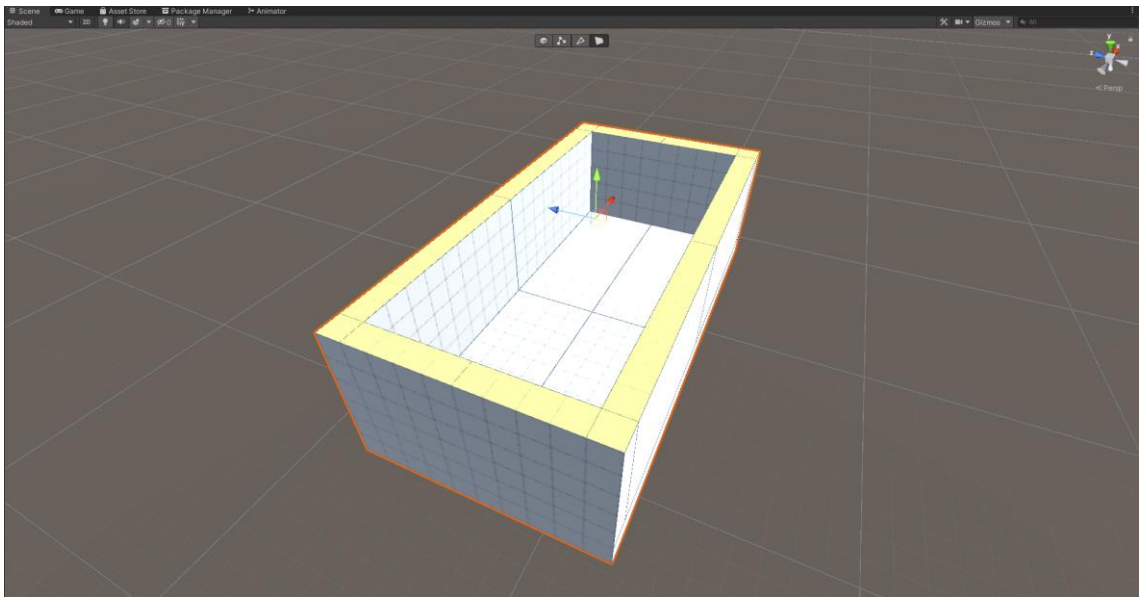
Otkrivena platforma na sebi sadrži *Collider* koji na isti način otkrije vrata koja je potrebno otvoriti tipkom „E“ te se pokreće animacija otvaranja vrata i igrač može proći kroz njih te se završava razina (slika 41).



Slika 41. Razina otvorenog tipa

4.2.2. Zatvorena razina

Izrada u Pro Builderu započinje otvaranjem prozora s alatima te se klikom na prvu ikonu dodaje kocka dimenzija koje se postave. Dodanoj kocki se označi gornja ploha te se opcijom *Subdivide Object* podijeli na 16 jednakih dijelova. Klikom na *Edge Selection* mijenja se mogućnost odabira rubova ploha te se njih pozicionira uz rub objekta. Alatom *Extrude Faces* označene plohe se podignu te se dobiju zidovi (slika 42).



Slika 42. Prostorija napravljena u Pro Builderu

Pošto se na razini nalazi nekoliko prostorija, potrebno je ovaj proces ponoviti nekoliko puta za svaku prostoriju, te se na kraju s *Merge Objects* svi kreirani objekti povežu u jedan. Zatim se scena popunjava dodavanjem interaktivnih objekata koji su prethodno napravljeni u Blenderu. Prije nego se objekti mogu dodati u hijerarhiju potrebno ih je ubaciti u Project prozor gdje se nalaze pod Assets. Potom se iz Assetsa mogu samo potegnuti u hijerarhiju i podešavati postavke u Inspector prozoru. Prvi objekt koji se dodaje su vrata koja će služiti za prolaz u druge prostorije i jedna vrata koja će predstavljati kraj razine. Na vrata se dodaje *Box Collider* i *Animator* koji će pokretati animaciju kada to bude potrebno. Animacija otvaranja vrata započinje tek nakon što kocka aktivira *Trigger* koji se nalazi na *Pressure Plateu*, a za to je potrebno napraviti skriptu (slika 43).

```
public class DoorTriggerRed : MonoBehaviour
{
    public GameObject Door;

    void OnTriggerEnter(Collider other)
    {
        if (other.tag == "CubeRed")
        {
            Door.GetComponent<Animator>().Play("DoorUp");
        }
    }
}
```

Slika 43. Skripta za otvaranje vrata

Kao što je već prije spomenuto, metoda *OnTriggerEnter* pokreće animaciju, ali ovaj put će se animacija aktivirati samo ako se u *Collideru* nalazi kocka koji ima oznaku „CubeRed“. Poput razina otvorenog tipa, i ova razina koristi sličnu mehaniku otkrivanja sakrivenih objekata tako da je princip isti, samo se ovaj put radi o kockama koje su potrebne za prelazak razine. Još jedna od mehanika koje se pojavljuju je upotreba dvije *Pressure Plate* koje se koriste za stvaranje nove kocke. Svaki *Pressure Plate* na sebi ima skriptu „Detector“ koja provjerava imaju li obje Pressure Plate na sebi kocku oznake „CubeBlue“ ili „CubeRed“. Varijabla „entered“ može imati samo dva stanja i stoga je odlično rješenje za ovu skriptu (Slika 44).

```
public class Detector : MonoBehaviour
{
    public bool entered = false;

    public void OnTriggerEnter (Collider _collider)
    {
        if (_collider.gameObject.name == "BlueCube" | _collider.gameObject.name == "RedCube")
        {
            entered = true;
        }
    }

    public void OnTriggerExit (Collider _collider)
    {
        if (_collider.gameObject.name == "BlueCube" | _collider.gameObject.name == "RedCube")
        {
            entered = false;
        }
    }
}
```

Slika 44. Detector skripta

Ukoliko se oba objekta nalaze unutar *Collidera* vrijednost „entered“ se mijenja u *true* i omogućava detektoru koji je u hijerarhiji napravljen od *Empty Game Objecta* da s pomoću druge skripte aktivira kocku koja je potrebna za otvaranje vrata. U slučaju da jedna od kocki nije u *Collideru*, stanje ostaje isto i kocka se ne aktivira. Skripta koja se nalazi na detektoru sastoji se od dva objekta koji predstavljaju Pressure Plate i jednog

koji označava kocku koja nastaje ako je uvjet da su oba detektora u stanju *True* (slika 45).

```
public class GameLogic1 : MonoBehaviour
{
    public Detector detector1;
    public Detector detector2;
    public GameObject Cube;

    public void Update()
    {
        if (detector1.entered && detector2.entered)
        {
            Cube.SetActive(true);
        }
    }
}
```

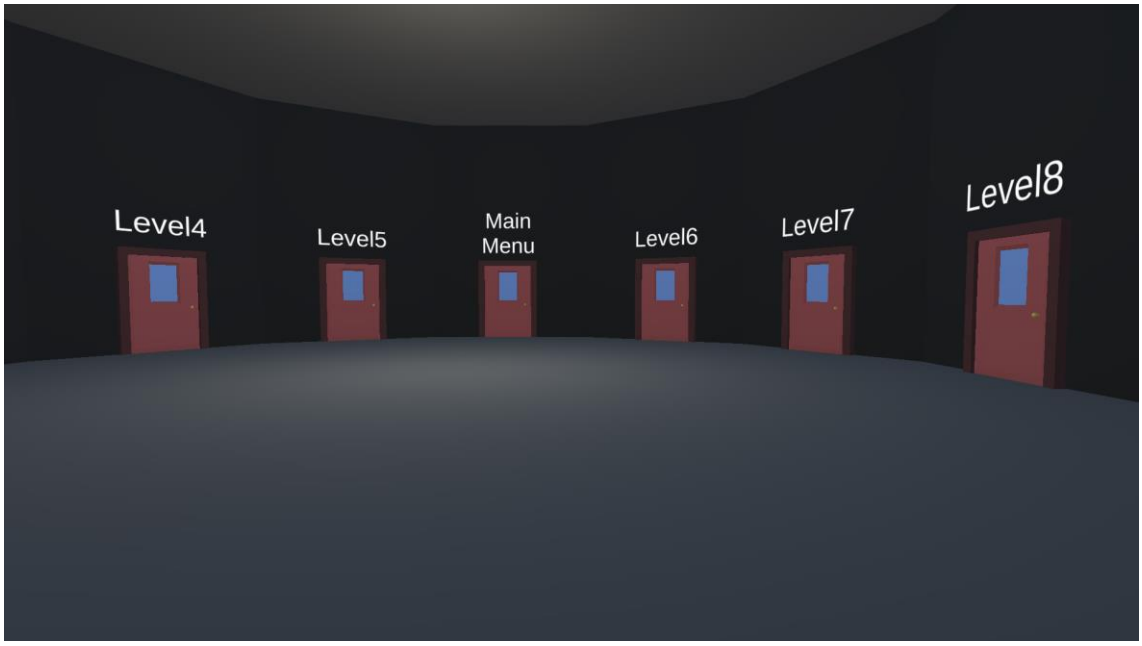
Slika 45. Skripta kojom detektor provjerava stanje Collidera

Sa svim trenutno dodanim objektima scena izgleda „prazno“ pa je potrebno dodati još nekoliko objekata kako bi malo popunili prostor, a za to se koriste kocke s kojima se ne može interaktirati ili objekti koji su prethodno izrađeni u Blenderu. Pošto se radi o zatvorenoj prostoriji, potrebno je još u hijerarhiju dodati *Light* za osvjetljenje i postaviti krov koji je napravljen isto kao i prostorija, ali se ne koristi *Subdivide i Extrude* kao kod izrade prostorije. Postavljanjem krova i jačine svjetlosti scena je gotova (slika 46).

Slika 46. Završena scena zatvorene razine

4.2.3. Level Select

Razini Level Select može se pristupiti samo iz glavnog izbornika. Ulaskom u razinu koja je zatvorenog tipa, igrač može odabrati koju razinu želi igrati te dolazi ispred vrata. Pritiskom tipke „E“ otvara vrata i prolaskom kroz njih započinje učitavanje željene razine. Razina se sastoji od vrata za svaku razinu i jednih vrata koja vraćaju na glavni izbornik (slika 47).



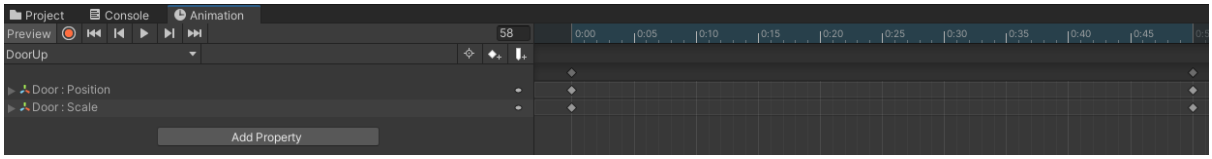
Slika 47. Level Select razina

4.3. Animacije

Animacije su napravljene za pokretne platforme ili za otvaranje vrata. Izrada animacije započinje kreiranjem nove animacije u Animation prozoru. Animation prozor sadržava vremensku traku i gumbe za pokretanje animacija i postavljanje *keyframeova* (slika 48).

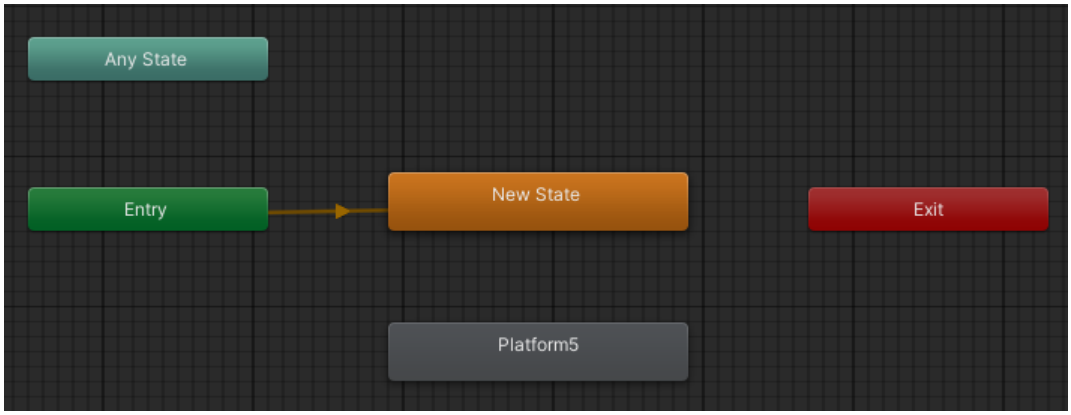
Slika 48. Animation prozor i vremenska traka

Gumb „Enable/Disable keyframe recording mode“ služi za snimanje svih promjena dimenzija ili pozicije objekta te se na vremenskoj traci postavlja *keyframe*. Pomicanjem po vremenskoj traci postavljaju se *keyframeovi* te se to ponavlja za svaku sekundu vremenske trake na kojoj je potrebna promjena. Kada se postave svi *keyframeovi* koji su potrebni, klikom na isti gumb kao i na početku prestaje snimanje svih promjena i dobije se gotova animacija (slika 49).



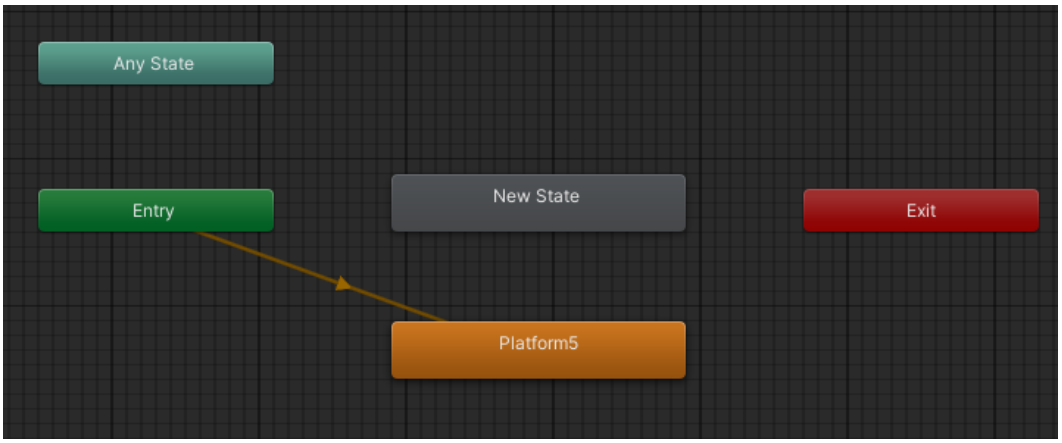
Slika 49. Promjena pozicije i veličine vrata prilikom otvaranja

Animacije koje se nalaze na vratima pokreću se tek kada se zadovolji uvjet postavljen na *Triggeru*. Da bi to realiziralo, potrebno je u Animator prozoru napraviti novo stanje objekta u kojem objekt miruje i postaviti to kao početno stanje (slika 50).



Slika 50. Prikaz početnog stanja objekta u animatoru

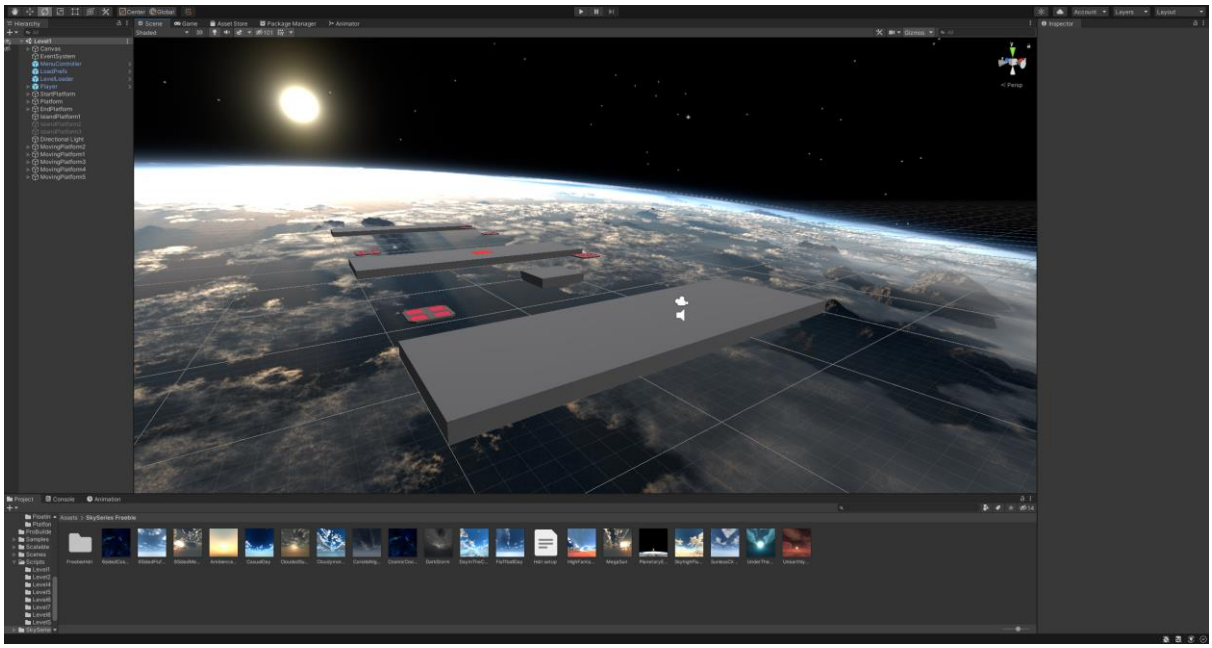
Tek kada se uvjet na Triggeru ispuni, objekt prelazi u stanje na kojem je animacija te se ona aktivira (slika 51).



Slika 51. Prikaz nakon zadovoljavanja uvjeta

4.4. Pozadina

Otvorene razine koriste skybox koji predstavlja pozadinu. Skybox slike su uvezene s Unity Asset Storea te se samo željena pozadina iz Project prozora povuče na scenu (slika 52).



Slika 52. Pozadina na sceni

4.5. Sučelje

4.5.1. Main Menu

Pokretanjem igre otvara se glavni izbornik koji se sastoji od pozadine i četiri gumba (slika 53).

Slika 53. Prikaz glavnog izbornika

Za izradu glavnog izbornika u hijerarhiju se dodaje *Canvas* na koji se dodaju UI elementi poput *Image* i *Button*. Prvi gumb je Play na koji je postavljena skripta *LevelLoader* koja pritiskom na gumb započinje učitavanje prve razine (slika 54).

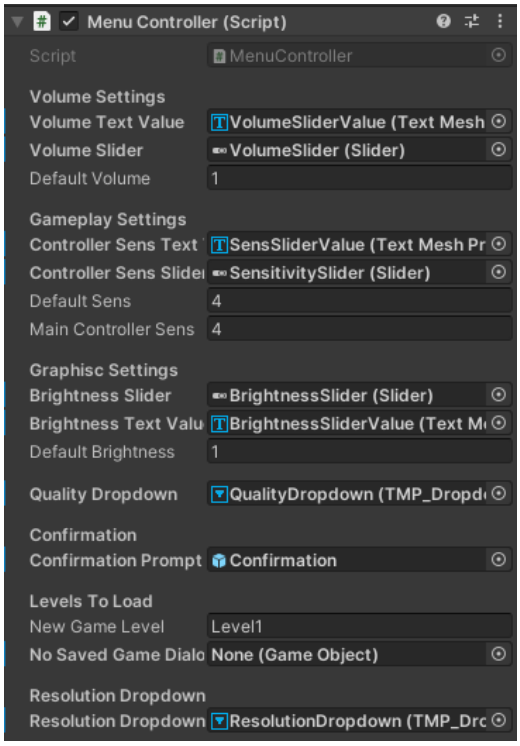
```
public class LevelLoader : MonoBehaviour
{
    public GameObject loadingScreen;
    public Slider slider;

    public void LoadLevel (int sceneIndex)
    {
        StartCoroutine(LoadAsynchronusly(sceneIndex));
    }

    IEnumerator LoadAsynchronusly (int sceneIndex)
    {
        AsyncOperation operation = SceneManager.LoadSceneAsync(sceneIndex);

        loadingScreen.SetActive(true);

        while (!operation.isDone)
        {
            float progress = Mathf.Clamp01(operation.progress / .9f);

            slider.value = progress;

            yield return null;
        }
    }
}
```

Slika 54. Skripta koja učitava iduću razinu

Level Select otvara razinu u kojoj se kretanjem po prostoriji dođe do vrata i odabere željena razina. Gumb Settings otvara novi izbornik u kojem se nude postavke poput rezolucije, jačine zvuka i popis kontrola (slika 55).

Slika 55. Izbornik sa postavkama

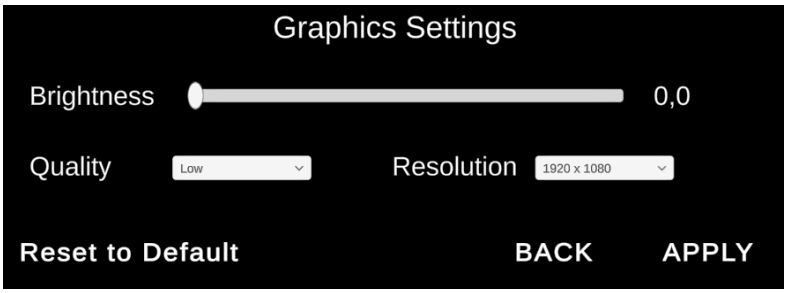
Kako bi se svi gumbi međusobno povezali, potrebno je u hijerarhiji dodati *Empty Object* koji će služiti kao upravljač, a na njega se dodaje skripta s funkcijama koje služe za mijenjanje postavki (slika 56).



Slika 56. Inspect prozor elementa na koji je postavljena skripta

4.5.1.1. Graphics settings

Klikom na gumb „Graphics“ otvara se prozorčić u kojem se mogu mijenjati grafičke postavke (slika 57).



Slika 57. Prozorčić sa grafičkim postavkama

Da bi se vrijednosti mogle mijenjati, potrebno je napraviti skriptu (slika 58).

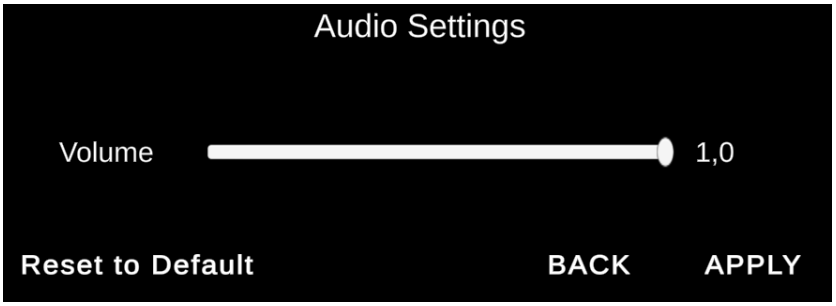
```
public void SetBrightness(float brightness)
{
    _brightnessLevel = brightness;
    brightnessTextValue.text = brightness.ToString("0.0");
}

public void SetQuality(int qualityIndex)
{
    _qualityLevel = qualityIndex;
}

public void GraphicsApply()
{
    PlayerPrefs.SetFloat("masterBrightness", _brightnessLevel);

    PlayerPrefs.SetInt("masterQuality", _qualityLevel);
    QualitySettings.SetQualityLevel(_qualityLevel);

    StartCoroutine(ConfirmationBox());
}
```

Slika 58. Skripta za mijenjanje vrijednosti

Funkcija *SetBrightness* određuje vrijednost svjetline koja se postavlja klizačem. *SetQuality* u padajućem izborniku nudi četiri opcije kvalitete slike koje se postave prilikom izrade videoigre. *GraphicsApply* sprema vrijednosti koje su prethodno namještene te ih postavlja za početne vrijednosti prilikom idućeg pokretanja videoigre.

4.5.1.2. Sound Settings

Gumb za postavke zvuka otvara prozor koji se sastoji od klizača za promjenu glasnoće te tri gumba za spremanje postavki, vraćanje na izbornik i vraćanje na originalne postavke (slika 59).



Slika 59. Prozor sa postavkama za zvuk

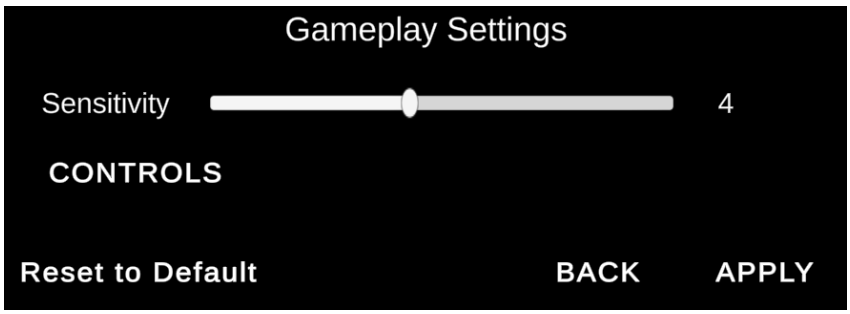
Za promjenu glasnoće potrebno je napraviti skriptu u kojoj funkcija *SetVolume* postavlja glasnoću zvuka na vrijednost koja se odredi na klizaču, te funkciju *VolumeApply* koja sprema promjene (slika 60).

```
public void SetVolume(float volume)
{
    AudioListener.volume = volume;
    volumeTextValue.text = volume.ToString("0.0");
}

public void VolumeApply()
{
    PlayerPrefs.SetFloat("masterVolume", AudioListener.volume);
    StartCoroutine(ConfirmationBox());
}
```

Slika 60. Skripta za postavke zvuka

4.5.1.3. Gameplay Settings

Gumb „Gameplay“ otvara prozor za promjenu osjetljivosti kontrolera i popis kontrola za igranje (slika 61).



Slika 61. Prozor sa gameplay postavkama

Skripta funkcijom *SetControllerSens* postavlja osjetljivost kontrolera na jačinu koja se postavlja klizačem te ju sprema pritiskom na gumb *Apply* (slika 62).

```
public void SetControllerSens(float sensitivity)
{
    mainControllerSens = Mathf.RoundToInt(sensitivity);
    controllerSensTextValue.text = sensitivity.ToString("0");
}

public void GameplayApply()
{
    PlayerPrefs.SetFloat("masterSens", mainControllerSens);
    StartCoroutine(ConfirmationBox());
}
```

Slika 62. Gameplay settings skripta

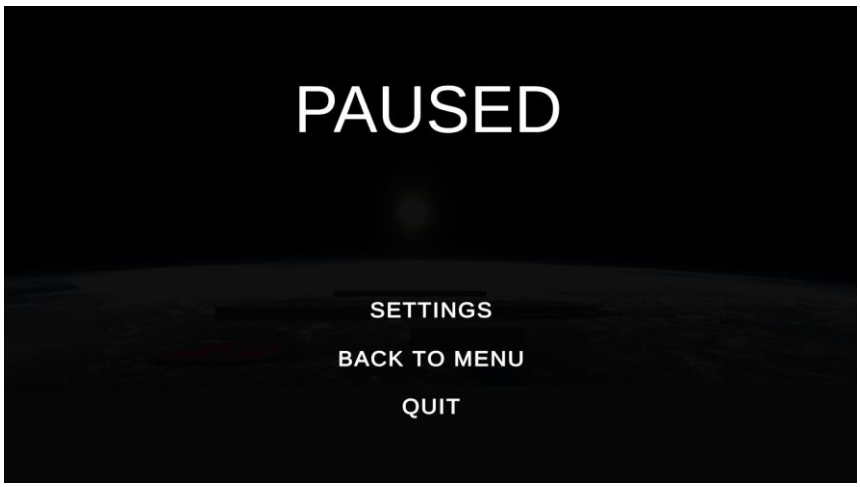
Zadnji gumb koji se nalazi na glavnom izborniku je Exit. Njime se igra gasi funkcijom *ExitButton* koja pomoću klase *Application* funkcijom *Quit* gasi igru (slika 63).

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

Slika 63. Funkcija ExitButton

4.5.2. Pause Menu

Prilikom igranja moguće je pritiskom na tipku „Escape“ pauzirati igru te se otvara izbornik sličan glavnom u kojem se mogu mijenjati postavke ili vratiti na glavni izbornik (slika 64).



Slika 64. Izbornik koji se otvara kada je igra pauzirana

Da bi opcija za pauziranje radila, potrebno je na element igrača dodati skriptu koja provjerava trenutno stanje igre (slika 65).

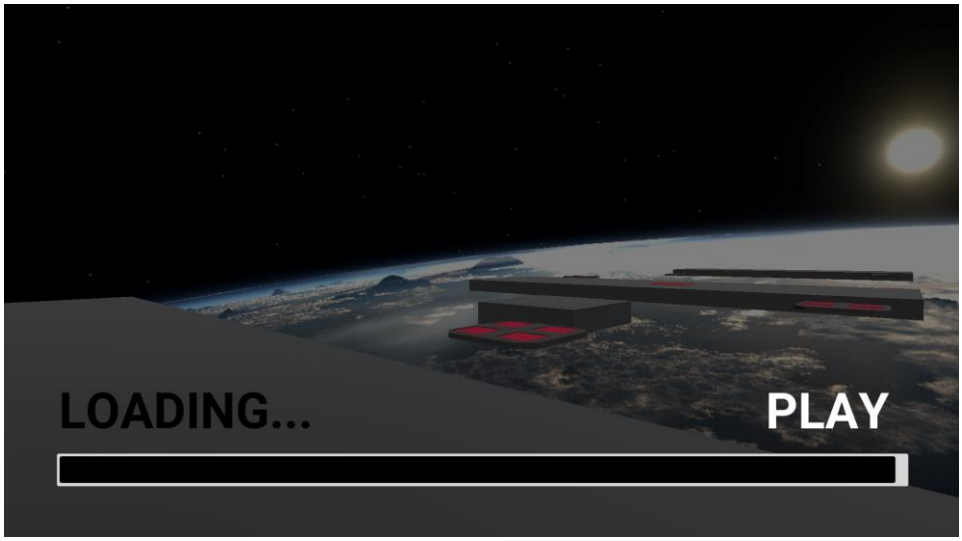
```
void Update()
{
    if(Input.GetKeyDown(KeyCode.Escape))
    {
        if(isPaused)
        {
            ResumeGame();
        }
        else
        {
            PauseGame();
        }
    }
}

public void PauseGame()
{
    pauseMenu.SetActive(true);
    Time.timeScale = 0f;
    isPaused = true;
}

public void ResumeGame()
{
    pauseMenu.SetActive(false);
    Time.timeScale = 1f;
    isPaused = false;
}
```

Slika 65. Skripta koja provjerava trenutno stanje igre

Ukoliko je pritiskom na tipku „Escape“ vrijednost *isPause* = *false* igra se pauzira te se vrijednost Time.timeScale postavlja na 0f, u suprotnom se vrijednost Time.timeScale postavlja na 1f i igra se nastavlja.

4.5.3. Loading Screen

Pokretanjem igre u glavnom izborniku ili odabirom razine u Level Selectu najprije je potrebno učitati razinu kako bi se moglo igrati. Loading Screen je sastavljen od slike, teksta, klizača i gumba za pokretanje razine (slika 66).



Slika 66. Loading Screen

Klikom na gumb Play, Loading Screen se deaktivira i može se igrati.

4.5.4. Ending Screen

Otvaranjem vrata na kraju razine aktivira se Collider koji otvara ending screen na kojem se može nastaviti igra na idućoj razini ili se može vratiti na početni izbornik (slika 67).

Slika 67. Ekran na završetku razine

5. ZAKLJUČAK

Videoigre su danas jedan od najpopularnijih medija zabave. Zahvaljujući razvoju tehnologije današnje videoigre naprednije su odnosu na rane, kompleksnije su i omogućavaju granje s većim brojem drugih korisnika istovremeno. Veliki broj videoigara danas nudi opciju online igranja s drugim igračima, što je stvorilo veliku zajednicu igrača koji se vole natjecati, pa se tako održavaju i velika natjecanja s novčanim nagradama koja bi se mogle usporediti s nekim sportovima.

Unity je kao *game engine* poprilično jednostavan i za one s veoma ograničenim predznanjem iz programiranja, a neće ga teško savladati ni netko bez ikakva iskustva. Sučelje možda zbunjuje na prvi pogled, ali ustvari je jednostavno snaći se u njemu. Jedan od velikih bonusa koje Unity nudi je Asset Store koji nudi širok spektar kreacija drugih korisnika, što omogućuje lakšu izradu željenih projekata, a isto tako ima nekolicinu *pluginova* koji omogućuju korištenje bez upotrebe drugih programa.

Videoigra se sastoji od nekoliko razina te glavnog izbornika i razine na kojoj je moguće odabrati određenu razinu. Izrada videoigre je poprilično kompleksna pa je za jednu osobu veoma dug proces zbog velikog broja dijelova koje je potrebno napraviti da bi igra bila funkcionalna. Kreiranje različitih razina nije suviše zahtjevno, ali se svaka razina sastoji od elemenata koji se povezuju s drugim elementima pa je izrada skripti malo kompliciranija i oduzima više vremena.

6. LITERATURA

1. https://blog.nationalmuseum.ch/en/2020/01/the-history-of-video-games/ - The history of video games, 01.08.2022.
2. https://www.history.com/topics/inventions/history-of-video-games - Video Game History, 01.08.2022.
3. https://www.museumofplay.org/video_games/ - Video Game History Timeline, 01.08.2022.
4. https://www.gamedesigning.org/gaming/history/ - Video Games History: From Magnavox Odyssey to The Wii, 01.08.2022.
5. https://www.androidauthority.com/what-is-unity-1131558/ - What is Unity? Everithing you need to know, 02.08.2022.