Izrada 2D platformera s bočnim pomicanjem u programskom alatu Unity

Kolar, Teodor

Undergraduate thesis / Završni rad

2023

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **University of Zagreb, Faculty of Organization and Informatics / Sveučilište u Zagrebu, Fakultet organizacije i informatike**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:211:430638

SVEUČILIŠTE U ZAGREBU
FAKULTET ORGANIZACIJE I INFORMATIKE
VARAŽDIN

Teodor Kolar

IZRADA 2D PLATFORMERA S BOČNIM POMICANJEM U PROGRAMSKOM ALATU UNITY

ZAVRNI RAD

Varaždin, 2023.

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Teodor Kolar

Matični broj: 0016150608

Studij: Informacijski sustavi

IZRADA 2D PLATFORMERA S BOČNIM POMICANJEM U PROGRAMSKOM ALATU UNITY

ZAVRNI RAD

Mentor :

Doc. dr. sc. Mladen Konecki

Varaždin, lipanj 2023.

Teodor Kolar

Izjava o izvornosti

Izjavljujem da je moj završni 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 potvrdio prihvaćanjem odredbi u sustavu FOI-radovi

Sažetak

U ovom radu obradit će se izrada dvodimenzionalnog platformera u stroju igara Unity. Cilj ovog rada je obraditi osnovne stvari koje ulaze u kreiranje jednog 2D platformera s bočnim pomicanjem u stroju igara Unity. Posebnost igre je da igrač ima mogućnost da promijeni smjer djelovanja gravitacije, prema dolje ili prema gore. Glavna ideja da velika većina funkcionalnosti radi preko Unity sustava fizike korištenjem sila. U radu će se obraditi osnovne mehanike pomicanja po svijetu i kreiranje razine i zagonetke. Objasnit će se kako način implementacije tih mehanika utječe na iskustvo igrača. Obradit će se kreiranje glavnog izbornika i pauziranja. U radu će još biti prikazano kako kreirati razine i zagonetke koje funkcioniraju kao komponente koje se lagano i efikasno mogu koristiti u više navrata kroz proces razvoja videoigre.

Ključne riječi: videoigra, platformer, bočno pomicanje, Unity, mehanike, kontrole

Sadržaj

1. Uvod

Danas je teško reći da postoji osoba koja nikad nije igrala ili vidjela neku videoigru. Ljudska bića od davnih vremena se bave aktivnošću igranja iz različitih razloga. Igranje pomaže da se ljudi povežu jedni s drugima, da budu sretni ili da se nose s nedaćama života. Videoigre donose novu razinu gdje ne postoje granice stvarnog fizičkog svijeta i moguće je modelirati bilo kakvi svijet u virtualnom okruženju.

Kreiranje videoigre je posebno iskustvo razvoja softvera gdje najbolje dolazi na vidjelo interdisciplinarnost u umjetnosti. Videoigre zahtijevaju da vizualna i auditivna komponenta zajedno s samim programskim kodom igre dođu da stvore posebno iskustvo koje ne može niti jedan drugi oblik medija ponuditi.

Ideja za ovu igrala nastala iz željom da se testira funkcioniranje Unity sustava fizike i da se što je potrebno da se kreiraju zagonetke za jednu igru. Zagonetke i prepreke su nešto puno apstraktnije od kreiranja klasičnih protivnika i više su podložne subjektivnom mišljenju iz pogleda težine i pristupačnosti.

Tijekom kreiranja igre korišteni su slijedeći alati: **Unity**, **Visual Studio 2022** i **GIMP**.

Unity je stroj igre koji omogućuje bilo kome da se okuša u izradi videoigre. Stroj igre je skup programa koji omogućuju izradu videoigara. Sve komponente stroja igre su apstraktirane na način da olakšaju korištenje i implementaciju tijekom izrade videoigre. Svaki stroj igre se sastoji od nekoliko ključnih podsustava: podsustav za unos, grafički podsustav, podsustav za fiziku, mrežni podsustav, podsustav za zvuk. Unity omogućuje izradu igara za Windows, Linux, Mac, iOS, Android, PlayStation i Xbox platforme. Unity Weta Tools koriste u filmskoj industriji za izradu računalno generirane grafike i specijalne efekte. Unity Industry se koristi za rad s CAD i 3D modelima u područjima industrijske proizvodnje, arhitekture i inženjerstva.

Visual Studio je integrirano razvojno okruženje kreirano od strane Microsoft-a. Omogućuje rad u svim modernim programskim jezicima kroz razne dodatke. Dolazi s posebnom Intellisense tehnologijom koja omogućuje automatsko dovršavanje koda, prikaz mogućih metoda, prikaz informacija i predviđanje sljedećih koraka programera kroz korištenje strojnog učenja. Visual Studio se ovdje koristio za kreiranje skripti C# programskom jeziku kojeg koristi Unity.

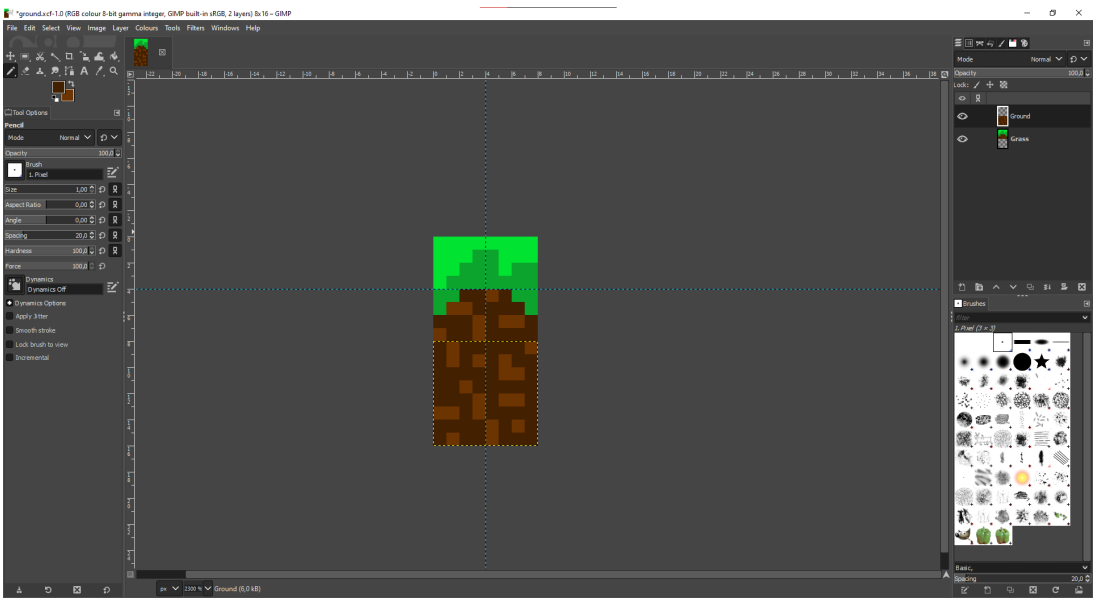
GIMP je besplatan program otvorenoga koda za rad s rasterskom grafikom. Omogućuje izmjenu i manipulaciju slika, a može se koristiti i za crtanje. Njegov rad se može automatizirati kreiranjem i korištenjem skripti u Python-u. GIMP se može nadograditi izmjenom izvornog koda ili kreiranjem dodataka. GIMP se ovdje koristio za izradu svih resursa potrebnih za vizuale igre.

2. Izrada igre

2.1. Stvaranje svijeta igre

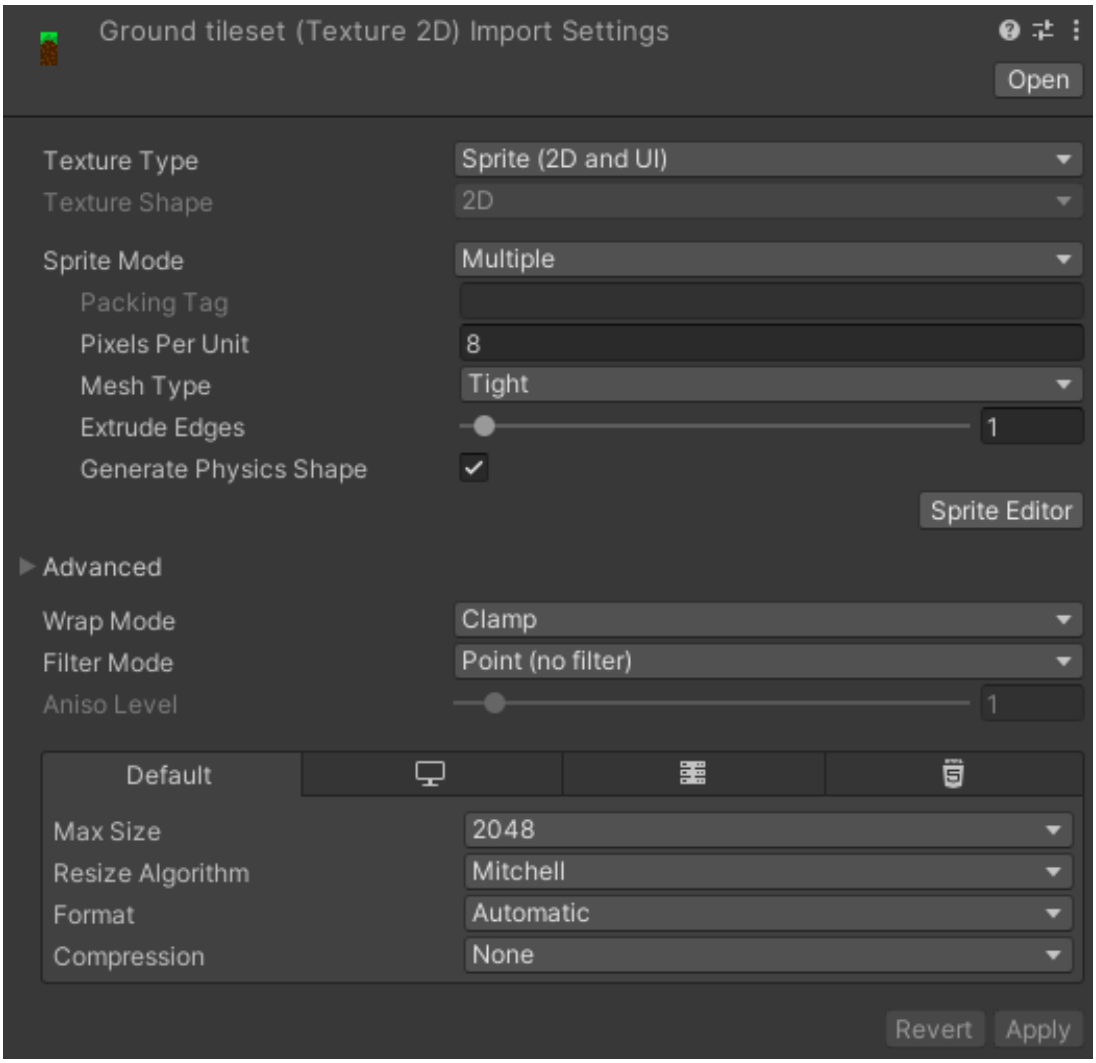
Za izradu prostora igre korišten je Unity-ov **TileMap** sustav. Sustav se sastoji od pločica (engl. *Tiles*) koje se onda postavljaju na mrežu (engl. *Grid*). Pločice su samo slike (engl. *Sprite* fiksne veličine koje se mogu po želji postaviti. Glavna ideja je da postoji nekolicina pločica koje čine paletu pločica (engl. *Tile Palette*) koje se samo ponavljaju, kombiniraju da se stvori najveći dio vizualnog dizajna i izgleda neke razine igre. Kada se jedanput kreira paleta pločica one se postavljaju tako da se uz pomoć kista samo slika s njima. Na taj sustav se nadovezuju posebne komponente poput posebnog TileMap sudarača koji je napravljen da automatski kreira optimalni i odgovarajući oblik kolizije za bilo koji objekt kreiran uz pomoć pločica. Sustav podržava i slojevito kombiniranje pločica gdje je moguće posebno odvojiti pozadinske pločice od glavnih koji čine razinu. Sustav je jako nalik starim dvodimenzionalnim igrama koje su koristile jednu sliku koja je imala mnogo manjih slika koje su se onda koristile kao pločice.

Za korištenje ovog sustava bitno je unaprijed odrediti stvari poput veličine pločica, broja piksela pojedinih pločica zbog kreiranja konzistentnog i usklađenog stila dizajna. Odlučeno je koristiti slike dimenzija 8x8 za pločice. Mjesto radnje ove razine igre je neko mjesto u prirodi uz neku špilju u dubini pa je potrebno kreirati jednu travnatu pločicu i jednu isključivo zemljastu pločicu koja se dobro nadovezuje na samu sebe. Prvo je u GIMP-u kreirano platno veličine 8x16 i koristeći običan kist podešen na veličinu jednog piksela je kreirano sljedeće:



Slika 1: Pločice u GIMP-u (Izvor: vlastita izrada)

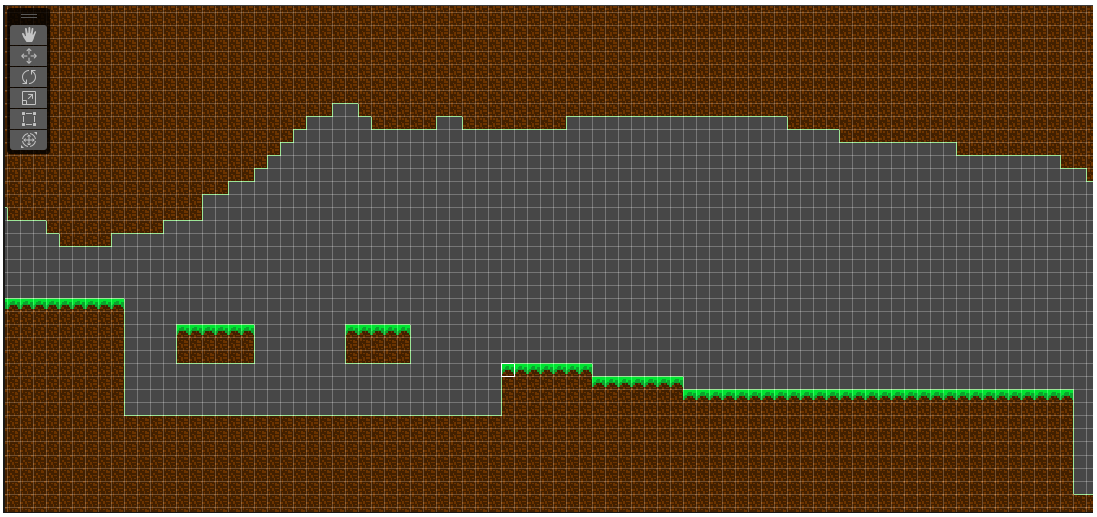
Slike su spremljene u PNG formatu i uvezene u Unity projekt. Kada se koristi ovakva vrsta resursa bitno je prilagoditi kako ih Unity tretira. Unity-ove zadane postavke automatski kompresiraju slike i filtriraju ih. Kod ovakvih slika niske razine detalja to uzrokuje mutni i nejasan izgled stoga je potrebno to isključiti.



Slika 2: Postavke slike (Izvor: vlastita izrada)

Ovdje je moguće uočiti da je **Compression** stavljen na **None**, **Filter Mode** stavljen na **Point (no filter)**. Ova slika se sastoji od dvije buduće pločice, jedne travnate i jedne zemljaste pa je potrebno staviti i **Sprite Mode** na **Multiple**. Budući da će pločice biti dimenzija 8x8 potrebno je još staviti **Pixels Per Unit** na vrijednost **8**. Sliku je još trebalo otvoriti u uređivaču slika u Unity-u i podijeliti ju na komade veličine 8x8. Sljedeće je kreirana nova paleta pločica imena "ground tileset". Sada je potrebno dodati objekte koji će sadržati igrivi prostor razine.

Prvo je kreiran prazan objekt imena "Level" i na njega dodana komponenta mreže koja će biti mreža na koju će se stavljati sve pločice. Zatim je tom objektu dodan dijete objekt imena "Ground" i na izborniku je odabrano da se kreira kao dvodimenzionalni objekt s kartom kvadratnih pločica (engl. *Tilemap Rectangular* što automatski dodalo i sve komponente sudarača koji su bitni je moguće ostvariti kontakt s podom razine i da objekti poput lika igrača, platformi ne padaju kroz pod.



Slika 3: Odsječak razine (Izvor: vlastita izrada)

2.2. Stvaranje lika igrača

Lik igrača je izrađen u alatu GIMP na isti način kao i ostale sličice u igri. Odabran je jednostavan dizajn u obliku kvadrata dimenzija 8x8 piksela:

Slika 4: Dizajn lika igrača (Izvor: vlastita izrada)

Ovaj dizajn je izabran zbog laganog korištenja i jednostavnosti. Kod micanja, skakanja i okretanja nisu potrebne nikakve posebne animacije nego je dovoljno samo manipulirati direktno slikom lika igrača.

Sljedeće je bilo potrebno uvesti ga i implementirati u igru. Kreiran je objekt "Player" koji se sastojao od gore navedene sličice i na njega su dodane komponente **Box Collider 2D** i **Rigidbody 2D** koje su bitne da lik igrača ima mogućnost sudaranja i da na njega mogu djelovati sile. Sljedeće je bilo potrebno kreirati skriptu za kontrolu lika igrača.

2.2.1. Kretanje lika igrača

Budući da je ideja da se koriste sile što je više moguće ovaj dio koda se nalazi u funkciji **FixedUpdate** koja se poziva točno 60 puta u sekundi jer istom brzinom radi i Unity sustav fizike. Ovdje se prvo dohvaća vrijednost horizontalnog unosa čija vrijednost može biti od -1 do 1 ovisno da li je igrač stisnuo tipku za micanje lijevo ili desno. Ta vrijednost se se koristi za izračunavanje sile koja će se dodati. Varijabla **speed** ovdje služi da se dodatno modulira vijednost sile koja se dodaje na lika igrača tako da se može točnije definirati željena brzina lika igrača. **ForceMode2D.Impulse** ovdje označava da će se sila dodati u cijelosti instantno. Tijekom testiranja **ForceMode2D.Force** je uzrokovao sporije dodavanje sile na lika igrača što

je činilo kretanje nezadovoljavajuće i nepogodno za ideju igre.

```
void FixedUpdate()
{
    float moveX = Input.GetAxisRaw("Horizontal");
    rb.AddForce(new Vector2(moveX * speed, 0), ForceMode2D.Impulse);
```

Slika 5: Kod za micanje lika igrača horizontalno (Izvor: vlastita izrada)

Sljedeće je potrebno implementirati mogućnost skakanja lika igrača. Skakanje se isto mora odvijati u funkciji **FixedUpdate**. Ovdje nastaje problem jer je skakanje samo pritisak gumba za skok, a ako se provjerava je li gumb za skok pritisnut u funkciji FixedUpdate onda se može dogoditi da baš u tom trenutku se ne izvršava FixedUpdate jer se ta funkcija izvršava samo 60 puta u sekundi. To nije problem kod implementacije kretanja jer se radi o situaciji gdje igrač drži gumb neki dulji period vremena.

Za skok je još bilo potrebno implementirati dodatne stvari koje su bile primijećene na jednoj stranici za bolje kontrole u platformerima. Na [1] se navode **Vrijeme kojota** (engl. *Coyote Time*) i **Međuspremnik skoka** (engl. Jump buffering). **Coyote Time** se odnosi na stil animacije lika kojota iz Looney Tunes serijala crtića gdje se često kojot nađe u situaciji da pretrči rub litice i onda ostane visjeti u zraku neko vrijeme. U razvoju igara taj pojam se odnosi na mogućnost lika igrača da skoči neko vrijeme nakon što je već prošao preko ruba poda ili platforme. To se implementira jer često igrač ne zna točno u piksele oblik sudarača svojega lika, a potrebno je napraviti skok baš na rubu neke platforme i kada igrač misli da može napraviti skok, a njegov lik ne skoči, pojavljuje se frustracija. Takvo ponašanje je van kontrole igrača te je dobro ako se izbjegne. **Jump Buffering** predstavlja mogućnost da se unos za skok uzme u obzir čak i ako lik igrača u tom trenutku ne zadovoljava uvijete za skok, npr. nije u kontaktu s podom. Može postojati situacija u kojoj igrač želi raditi uzastopne skokove, ali igrač opet ne zna točan oblik sudarača njegovog lika i ne zna kada je njegov lik u kontaktu s podom i kada pritisne gumb za skok lik igrača ne skoči jer je on bio golim okom nevidljivo udaljen od poda. Implementacijom gore navedene metode opet se eliminira nešto što je van kontrole igrača i omogućuje se bolje iskustvo koje je usklađeno s očekivanjima igrača.

Osnovna stvar kod skoka je kreiranje provjere da li je lik igrača prizemljen ili bilo kakva provjera da li lik igrača smije u tom trenutku skočiti. Korištena je funkcija **BoxCast**. Ta funkcija omogućuje kreiranje "kutije" čija se kolizija ili preklapanje s nekim sudaračem može detektirati. U ovoj situaciji se kreira kutija jednaka po poziciji, obliku i veličini komponenti sudarača lika igrača. Posebnost je to što je ona pomaknuta za vrijednost od 0.1 jedinica u smjeru djelovanja gravitacije i definirano je da se detektiraju samo kolizije s podom koji je označen varijablom **jumpableGround**. Podu, zemlji je ovdje pridružen posebno definiran sloj koji se zove **Ground**. Ako postoji preklapanje ili sudar između te kutije i poda funkcija **GroundedCheck** će vratiti istina, a ako nema onda laž.

Ovdje su vidljive varijable **lastJumpTime** i **lastGroundedTime**. One su inicijalizirane s vrijednošću 0. Prva služi za praćenje koliko je vremena prošlo od posljednjeg skoka lika igrača, a druga koliko je vremena prošlo od trenutka kada je posljednji put lik igrača bio prizemljen. Vrijeme se prati na način da se od tih varijabli oduzme vrijeme između zadnje dvije sličice i onda se

```
bool GroundedCheck()
{
    return Physics2D.BoxCast(coll.bounds.center, coll.bounds.size, 0f, Physics2D.gravity.normalized, .1f, jumpableGround);
}
```

Slika 6: Kod za detektiranje prizemljenosti lika igrača (Izvor: vlastita izrada)

provjeri ako su te varijable veće od nule jer sve dok jesu znači da još nije isteklo vrijeme i igraču je dopušten skok. Ovdje su i varijable **maxJumpKeyPrePressTime** i **maxNotGroudedTime**. Prva se koristi da se odredi koliko prije nego lik igrača bude prizemljen će se uzeti u obzir da je igrač pritisnuo gumb za skok, a druga koliko vremena nakon što lik igrača prođe preko ruba neke litice mu je još omogućen skok. Obje su postavljene na vrijednost od 0.2 sekunde. Kada lik igrača pritisne tipku za skok **lastJumpTime** se postavlja na vrijednost **maxJumpKeyPrePressTime**. Kada je lik igrača prizemljen tada se cijelo vrijeme vrijednost **lastGroundedTime** postavlja na **maxNotGroudedTime**. Ako je igraču dopušten skok globalna varijabla **justJumped** se postavlja na istina. Ta varijabla je globalna jer se ovaj dio koda nalazi u **Update** funkciji, a sam skok je implementiran u **FixedUpdate**.

```
lastJumpTime -= Time.deltaTime;
lastGroundedTime -= Time.deltaTime;

if (Input.GetButtonDown("Jump"))
{
    lastJumpTime = maxJumpKeyPrePressTime;
}

if (GroundedCheck())
{
    lastGroundedTime = maxNotGroundedTime;
}

if ((lastGroundedTime > 0) && (lastJumpTime > 0))
{
    justJumped = true;
}
```

Slika 7: Kod za bolje iskustvo kod skakanja (Izvor: vlastita izrada)

Ovdje se provjerava ako je varijabla **justJumped** istinita. Ako je onda se postavlja na laž tako da lik igrača ne skoči više puta uzastopno, poziva se funkcija **Jump** i odsvira se zvučni efekt za skok.

```
if (justJumped)
{
    justJumped = false;
    Jump();
    jumpSoundEffect.Play();
}
```

Slika 8: Kod za provjeru skoka lika igrača (Izvor: vlastita izrada)

U funkciji za skok se varijable **lastGroundedTime** i **lastJumpTime** postavljaju na 0 tako da se izbjegne nepoželjno i nepredvidivo ponašanje i na lik igrača se dodaje sila čija je suprotna smjeru djelovanja gravitacije i koja je modulirana varijablom **jumpHeight**.

```
void Jump()
{
    lastGroundedTime = 0;
    lastJumpTime = 0;
    rb.AddForce(jumpHeight * Physics2D.gravity.normalized * -1, ForceMode2D.Impulse);
}
```

Slika 9: Kod za skok lika igrača (Izvor: vlastita izrada)

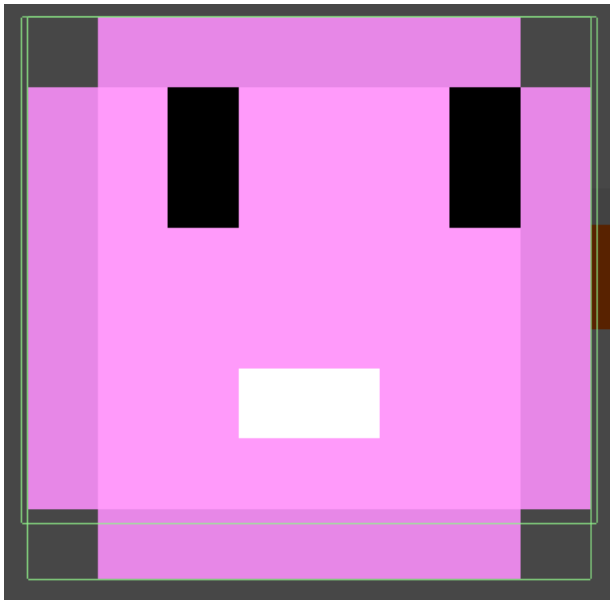
Testiranjem ovog skoka otkriveno je da se sam skok osjeća loše i da lik igrača prebrzo počne padati, a sam po sebi jako sporo pada. To može utjecati na preciznost skoka i samo iskustvo igrača jer time nastaje jako strma i nepredvidiva putanja, krivulja skoka. Dodatno su još implementirane manipulacije utjecaja gravitacije na lika igrača ovisno o uvjetima. Prvo se provjerava ako je lik igrača prizemljen, ako nije nema potrebe da se utječe na djelovanje gravitacije na njega. Ako lik igrača nije prizemljen, prvo se njegova brzina na Y-osi uspoređuje s varijablom **peakAirTimeSpeed**. Ta varijabla označava interval u kojem se utjecaj gravitacije na lika igrača prepolovi, tako da na samom vrhuncu skoka lik igrača malo "visi" u zraku. Varijabla je postavljena na vrijednost od 0.5. Uspoređivanje se radi s apsolutnom vrijednošću brzine lika igrača na Y-osi jer kada lik igrača ide prema gore, njegova brzina je pozitivna, a kada pada je negativna i onda se uvjet zadovoljava samo u tom intervalu od -0.5 do 0.5. Kada lik igrača već ubrza u svojem padu utjecaj gravitacije na njega se poveća za 2.8 puta tako da onda brže pada. Ovdje su dodane provjere jer kod promijene smjera djelovanja gravitacije se sve mora provjeriti dodatno u drugom smjeru. Na kraju, ako je lik igrača prizemljen utjecaj gravitacije na njega se postavlja na normalnu vrijednost. Varijabla **gravityScale** sadrži početnu vrijednost utjecaja gravitacije na lika igrača i služi kao baza za ostale manipulacije djelovanja gravitacije na lika igrača. Ovaj kod se nalazi u **Update** funkciji.

```
if (!GroundedCheck())
{
    if (MathF.Abs(rb.velocity.y) < peakAirTimeSpeed)
    {
        rb.gravityScale = gravityScale * 0.5f;
    }
    else if ((Physics2D.gravity.y < 0 && rb.velocity.y < 0) || (Physics2D.gravity.y > 0 && rb.velocity.y > 0))
    {
        rb.gravityScale = gravityScale * 2.8f;
    }
}
else
{
    rb.gravityScale = gravityScale;
}
```

Slika 10: Kod za bolji osjećaj skoka(Izvor: vlastita izrada)

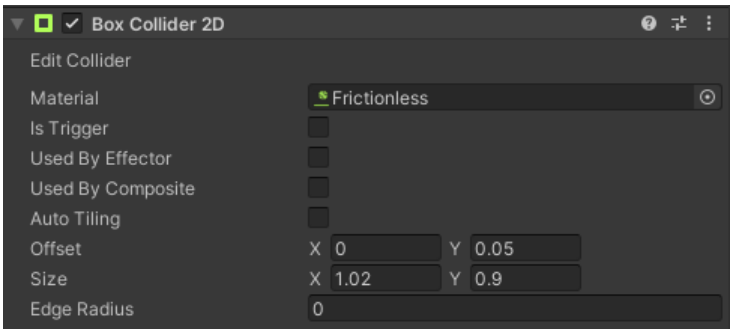
Na lika igrača je dodan još jedan sudarač. Kroz testiranje se pokazalo da se lik igrača prilijepi za zidove ako igrač skoči i drži unos za kretanje u smjeru zida. To je zato jer početni materijal na komponenti sudarča ima određeno trenje koje u ovom trenutku zajedno s silom koja se dodaje da se lik igrača kreće nedopušta da lik igrača pada i klizi uz zidove. Taj novi sudarač je napravljen da bude malo širi od običnog i da nema kontakta s podom preko njega.

Ideja je da se na njega doda novi materijal koji nebi imao trenje i tako dopustio liku igrača da se ne zalijepi za zidove. Ako bi se taj materijal dodao na glavni sudarač onda bi lik igrača tijekom kretanja klizio po podu.

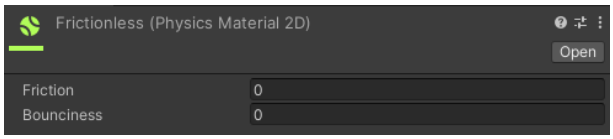


Slika 11: Lik igrača s dva sudarača(Izvor: vlastita izrada)

Ovdje je vidljivo da je taj sudarač pomaknut malo prema gore i da vertikalno bude manji, ali veći horizontalno tako da kontakt sa zidovima bude ostvaren samo preko njega.



Slika 12: Svojstva komponente drugog sudarača(Izvor: vlastita izrada)

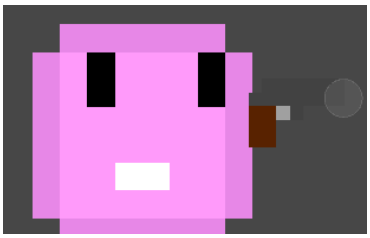


Slika 13: Svojstva komponente drugog sudarača(Izvor: vlastita izrada)

2.2.2. Pucanje projektila lika igrača

Za mehaniku pucanja projektila lika igrača bilo je potrebno kreirati sličicu oružja. Oružje je običan objekt sa slikom. Oružje je dijete objekt objekta lika igrača tako da objekt oružja cijelo

vrijeme prati kretanje lika igrača i relativno njemu bude na istoj poziciji. Na objekt oružja je još dodan i prazan objekt dijete koji predstavlja mjesto kreiranja i ispucavanja projektila.



Slika 14: Lik igrača s pridruženim oružjem(Izvor: vlastita izrada)

Skripta za pucanje se mora pobrinuti za više stvari. Prva je da oružje cijelo vrijeme prati pokazivač igrača. To je postignuto tako da se prvo odredi smjer gledanja oružja na način da se oduzmu pozicije oružja i pokazivača. Smjer je određen vektorom kao točka u svijetu igre pomoću funkcije **ScreenToWorldPoint** koja pripada glavnoj kameri. Sljedeće se odredi kut koji zatvaraju vektori koji definiraju tu točku u svijetu igre pomoću funkcije **Atan2**. Ta funkcija vraća kut u radijanima pa je potrebno pretvoriti u stupnjeve. Još je potrebno dobiti rotaciju i staviti ju na objekt oružja. To je postignuto funkcijom **AngleAxis** koja omogućuje stvaranje rotacije s željenim kutom oko neke osi. Ovdje to je Z-os, koja se može zamisliti kao linija koja ide u ekran ili van njega jer se radi o 2D igri. Ovaj kod se nalazi u **FixedUpdate** funkciji jer bi inače došlo do titranja oružja. To nastaje zbog načina na koji je implementirano kretanje igrača, isto u **FixedUpdate** funkciji te je bitno da se kut gledanja oružja i lokacija igrača ažuriraju jedan za drugim.

```
void FixedUpdate()
{
    Vector2 direction = mainCamera.ScreenToWorldPoint(Input.mousePosition)
                                                    - transform.position;
    float angle = Mathf.Atan2(direction.y, direction.x) * Mathf.Rad2Deg;
    Quaternion rotation = Quaternion.AngleAxis(angle, Vector3.forward);
    transform.rotation = rotation;
}
```

Slika 15: Kod za praćenje pozicije pokazivača(Izvor: vlastita izrada)

Sljedeće je implementirano ispaljivanje projektila. Ispaljivanje projektila je ograničeno vremenom čekanja između ispaljivanja tako da se poveća potreba za preciznošću igrača. Ovdje se varijabla **cooldown** koristi da se prati to vrijeme. Varijabla **cooldown** ima početnu vrijednost istu kao i varijabla **fireRate** koja označava koliko često igrač može ispucavati projektile i ovdje je postavljena na vrijednost od 1. Varijabla **cooldown** se kontinurano poveća za vrijeme između zadnje dvije sličice tako se može pratiti protok vremena. Kada igrač pritisne tipku za ispucavnaje i kada je varijabla **cooldown** veća od dozovljene brzine ispucavanja zadovoljava se uvjet za ispucavanje. Prvo se svira zvučni efekt i onda se kreira projektil koji je već prije kreiran predložak referenciran u varijabli **projectilePreFab**. On se kreira na poziciji objekta koji predstavlja mjesto kreiranja i ispucavanja projektila u varijabli **projectileFirePoint** i daje mu se vrijednost rotacije oružja. Na kraju se **cooldown** postavlja na 0 tako da igrač može pucati ponovno tek nakon vremena definiranog u **fireRate**. Ovaj kod se nalazi u **Update** funkciji jer je

bitno da se provjera za pritisak gumba za pucanje bude pouzdana.

```
void Update()
{
    cooldown += Time.deltaTime;
    if (Input.GetButtonDown("Fire1") && cooldown > fireRate)
    {
        gunshotSound.Play();
        GameObject newProjectile = Instantiate(projectilePrefab, projectileFirePoint.transform.position, transform.rotation);
        cooldown = 0f;
    }
}
```

Slika 16: Kod za praćenje pozicije pokazivača(Izvor: vlastita izrada)

Projektil je obična slika žute boje malih dimenzija. Na projektil je dodana skripta koja osigurava da se projektil ispravno ispali kad se kreira.

Slika 17: Slika projektila(Izvor: vlastita izrada)

U skriptu za projektil već pri samom stvaranju projektila u **Start** funkciji se prvo dohvaća **RigidBody2D** komponenta objekta i na nju se dodaje sila. Sila se dodaje u smjeru X-osi i modulirana je varijablom **force** koja je postavljena na vrijednost 4. Ispod je još vidljiva i funkcija **OnCollisionEnter2D** koje se brine da kad se projektil sudari s bilo čime što pripada "Ground" sloju da se projektil uništi. Projektile kada više nisu bitni potrebno je uništiti tako da se nezauzimaju nepotrebno resursi računala.

```
void Start()
{
    rb = GetComponent<Rigidbody2D>();
    rb.AddForce(transform.right * force, ForceMode2D.Impulse);
}

Unity Message | 0 references
private void OnCollisionEnter2D(Collision2D collision)
{
    if (collision.gameObject.layer == 6) //6 is the "Ground" layer
    {
        Object.Destroy(this.gameObject);
    }
}
```

Slika 18: Kod projektila(Izvor: vlastita izrada)

Tijekom testiranja je otkriveno da je potrebno posebno se pobrinuti da slika lika igrača bude usmjerena ovisno o smjeru kretanja zajedno s oružjem. Prvo se u varijablu **currentDirection** sprema trenutna orijentacija tako da se oduzme trenutna pozicija lika igrača i njegova pozicija u prošloj sličici. X-os predstavlja smjerove lijevo i desno, ako je X vrijednost vektora koji

sadrži smjer veća od 0 znači da se lik igrača kreće prema desno što je i početno očekivanje te se rotacija lika igrača i razmjer veličina postavljaju na početne vrijednosti. Ako je X vrijednost manja od 0 znači da se lik igrača kreće prema lijevo i potrebno ga je rotirati. Prvo se lik igrača rotira za 180 stupnjeva po Y osi tako da se njegova slika zrcali na drugu stranu zajedno s oružjem. Još je potrebno utjecati na razmjer veličina oružja tako da se i oružje ispravno zrcali jer će zbog skripte koja se brine da oružje bude usmjereno u smjeru pokazivača igrača biti naopako okrenuto. Ako se na Y vrijednost razmjera stavi negativna Y vrijednost od početne dobiva se efekt zrcaljenja po X-osi i onda i oružje i lik igrača izgledaju ispravno okrenuto prema lijevo.

```
Vector2 currentDirection = (Vector2)transform.position - previousPosition;
previousPosition = transform.position;
if (currentDirection.x > 0)
{
    transform.rotation = Quaternion.Euler(0, 0, 0);
    weapon.transform.localScale = weaponScale;
}
else if (currentDirection.x < 0)
{
    transform.rotation = Quaternion.Euler(0, 180, 0);
    weapon.transform.localScale = new Vector2(weapon.transform.localScale.x, -weaponScale.y);
}
```

Slika 19: Kod za ispravno orijentiranje lika igrača(Izvor: vlastita izrada)

2.3. Implementacija moći promjene gravitacije

2.3.1. Korištenje moći promjene gravitacije

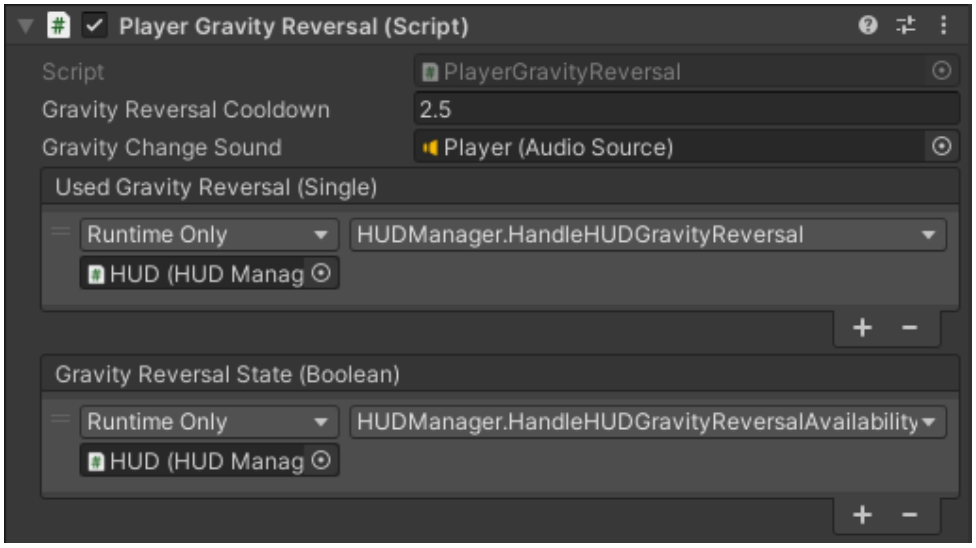
Ideja moći promjene gravitacije je da igrač može utjecati na smjer djelovanja gravitacije tako da djeluje prema dolje ili prema gore. To bi se koristilo u raznim preprekama, zagonetkama i slično. Ta moć bi se aktivirala pritiskom na gumb i igrač bi morao čekati kada ju jedanput iskoristi. Prva stvar koja je potrebna je kreirati posebnu skriptu koja će brinuti samo za tu funkcionalnost. Varijabla **lastGravityReversal** se koristi da se prati vrijeme od posljednjeg korištenja moći. Ona se kontinuirano povećava da se prati protok vremena. Njezina početna vrijednost je isto kao i varijabla **gravityReversalCooldown** koja označava koliko često igrač može iskoristiti moć i ovdje je postavljena na vrijednost od 2.5 što označava 2.5 sekundi. Kada je ta moć omogućena, igrač pritisne gumb za korištenje moći i kada je varijabla **lastGravityReversal** veća od **gravityReversalCooldown** zadovoljavaju se uvjeti za korištenje moći. Posebni uvjet za provjeru je li moć omogućena je bitan za kasnije jer je ideja da postoje mjesta u prostoru igre u kojima kada se nalazi lik igrača on ne može koristiti tu moć. Prva stvar koja se dogodi je da se promijeni gravitacija u cijeloj igri. Preko **Physics2D** objekta moguće je utjecati na cijeli sustav fizike u Unity-u. Gravitacija se postavlja na svoju suprotnu vrijednost. Varijabla **lastGravityReversal** se postavlja na 0 da se označi da je sada bila iskorištena. Vertikalna brzina igrača se postavlja na 0 tako da u slučaju ako igrač iskoristi moć kad je u zraku da lik igrača odmah počne padati u smjeru djelovanja gravitacije. To je napravljeno tako da se korištenje moći osjeća responzivnije. Sljedeće se svira zvučni efekt i doziva se događaj **usedGravityReversal** s argumentom **gravityReversalCooldown**. Ovdje je **usedGravityReversal** Unity događaj.

Unity događaji su poseban način implementacije komunikacije između objekata koji se

```
void Update()
{
    lastGravityReversal += Time.deltaTime;

    if (enableGravityReversal && Input.GetButtonDown("ReverseGravity") && lastGravityReversal > gravityReversalCooldown)
    {
        Physics2D.gravity = Physics2D.gravity * -1;
        lastGravityReversal = 0;
        rb.velocity = new Vector2(rb.velocity.x, 0);
        gravityChangeSound.Play();
        usedGravityReversal.Invoke(gravityReversalCooldown);
    }
}
```

Slika 20: Kod za moć promjene gravitacije(Izvor: vlastita izrada)

može lagano konfigurirati preko Unity uređivača [2]. Umjesto da se u skripti ručno dohvati referenca na objekt korisničkog sučelja, ovdje je dovoljno samo događaju pridružiti skriptu i funkciju iz skripte koja se nalazi na objektu s kojim se želi komunicirati. Ovdje je u uređivaču vidljivo pod **Used Gravity Reversal** da mu je pridružena skripta **HUD Manager** i funckija **HandleHUDGravityReversal** iz te skripte.

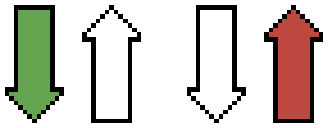


Slika 21: Konfiguracija Unity događaja(Izvor: vlastita izrada)

2.3.2. Prikaz moći na korisničkom sučelju

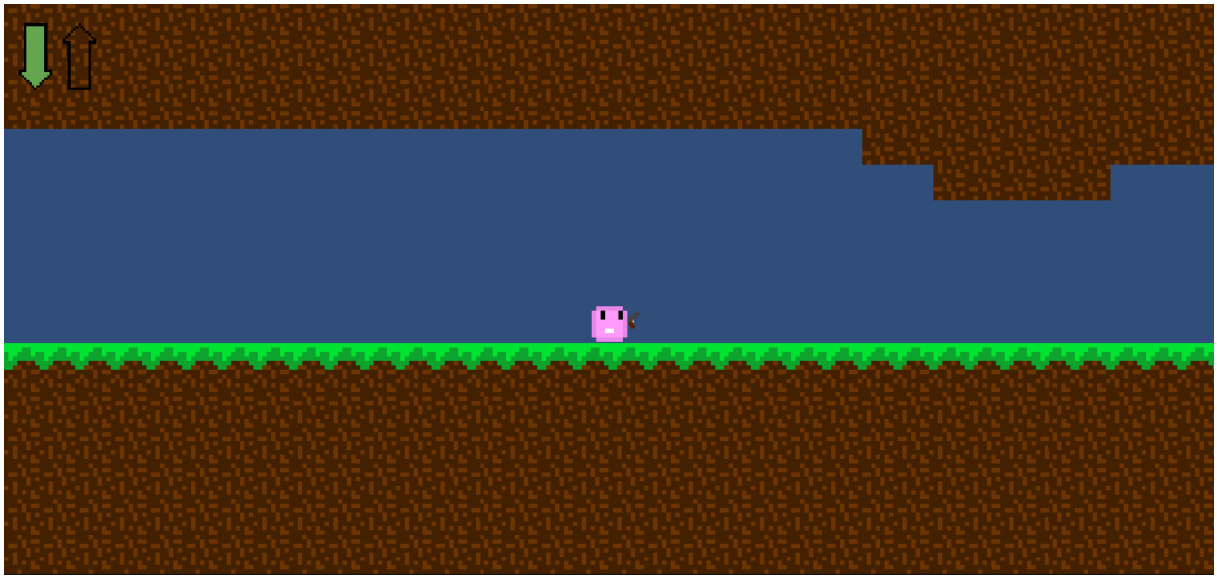
Za prikaz moći na korisničkom sučelju igre kreirane su dvije ikone dimenzija 32x32. Obojana strelica predstavlja u kojem smjeru djeluje gravitacija. Za bolju jasnoću strelice su drukčijih boja tako da bude igraču odmah na pogled jasno. Ideja je da kada igrač iskoristi moć da se ikona promijeni i da joj se kompletno makne ispuna jer u tom trenutku više igrač ne može iskoristiti moć. Kako prolazi onih 2.5 sekundi između svake uporabe moći ikona se ponovno ispunjava kružno suprotno smjeru kazaljke na satu i kada je kompletno puna igrač može ponovno koristiti moć. Kada se lik igrača nalazi u prostoru koji ne dozvoljava korištenje moći ikona se sakrije sve dok lik igrača ne izađe van iz tog prostora.

U igri korisničko sučelje je postavljeno tako da ikona za prikaz moći bude uvijek u gornjem lijevom rubu ekrana igrača. Za implementaciju je korišteno Unity Platno (engl. *Canvas*



Slika 22: Ikone prikaza moći korisničkom sučelju(Izvor: vlastita izrada)

koje je podešeno da se skalira s veličinom ekrana i da očekivana rezolucija bude 1920x1080. Ta rezolucija je odabrana jer prema Steam korisničkim istraživanjima to je najčešće korištena rezolucija, točnije koristi ju 60.75% svih korisnika [3].



Slika 23: Korisničko sučelje u igri(Izvor: vlastita izrada)

Za funkcionalnost korisničkog sučelja se brinu dvije skripte. Jedna se nalazi na glavnom "HUD" objektu koji sadrži sve elemente i objekte korisničkog sučelja. Skripta na tom objektu je zaslužna za prenošenje stvari u svoje dijete objekte i upravljanje njima. Trenutno je to samo element za prikaz moći, ali bitno je unaprijed razmišljati i planirati ako se kasnije želi nadograditi. Tom objektu je pridružen dijete objekt koji sadrži ikonu koja prikazuje trenutno stanje moći. Na njemu se nalazi skripta koja je zaslužna za animaciju. Ovdje događaj i posao putuju kroz hijerarhiju objekata što povećava apstrakciju i modularnost koda.

U prvoj skripti se nalazi prije spomenuta funkcija **HandleHUDGravityReversal** koja prima argument **cooldown**. Ova funkcija pokreće Unity korutinu. Korutine su posebna vrsta funkcije koje omogućuju da obave neki posao u nekom trenutku izvođenja igre, točnije sličici i iduću sličicu se odmah vratiti na to isto mjesto u kodu [4]. To je korisno za animiranje jer klasične funkcije bi u potpunosti blokirale glavnu petlju igre. Time se stvara osjećaj paralelizma, ali bitno je napomenuti da se korutine i dalje izvode na glavnoj niti. **abilityIconScript** je objekt koji sadrži sliku moći i sadrži funckiju **IndicateAbilityUse** koja prima vrijeme čekanja između korištenja kao argument jer animacija ovisi o vremenu. Prenošenjem te vrijednost omogućuje se dinamično trajanje animacije koje direktno ovisi o vremenu koje mora proći između korištenja moći bez da se mora mijenjati ručno na svakom mjestu gdje se koristi.

```
public void HandleHUDGravityReversal(float cooldown)
{
    StartCoroutine(abilityIconScript.IndicateAbilityUse(cooldown));
}
```

Slika 24: Funkcija za pokretanje animacije(Izvor: vlastita izrada)

Ovdje je vidljiv kod za animaciju ikone moći. Prvo se provjerava koja je ikona jer se ova funkcija poziva samo kada igrač iskoristi moć pa je potrebno samo staviti drugu ikonu. Varijabla **timerToEnd** prati vrijeme od početka do kraja animacije. Onda se ulazi u petlju koja je zaslužna za animaciju. Petlja traje sve do kraja animacije. Ispuna ikone se izračunava ovisno o vremenu koje je prošlo tako da se podijeli trenutno vrijeme od početka animacije s vremenom trajanja animacije. Ispuna slike je vrijednost od 0 do 1.

```
public IEnumerator IndicateAbilityUse(float cooldown)
{
    if(icon.sprite == downIcon)
    {
        icon.sprite = upIcon;
    }
    else
    {
        icon.sprite = downIcon;
    }

    float timerToEnd = 0f;
    while (timerToEnd <= cooldown)
    {
        icon.fillAmount = CalculateFill(cooldown, timerToEnd);
        timerToEnd += Time.deltaTime;
        yield return null;
    }
}
1 reference
private float CalculateFill(float cd, float timer)
{
    float fillAmount;
    fillAmount = timer / cd;
    return fillAmount;
}
```

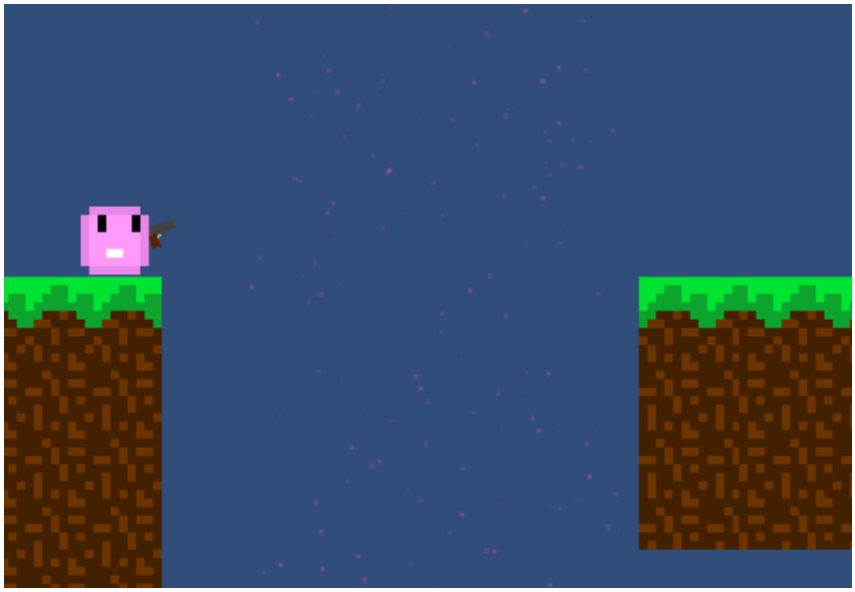
Slika 25: Funkcija za animiranje slike(Izvor: vlastita izrada)

2.3.3. Prostor isključenja moći

Prostor isključenja moći je namijenjen tome da onemogući igraču korištenje moći promjene gravitacije. Ulaskom u taj prostor ako gravitacija djeluje prema gore ona se postavlja da djeluje prema dolje. Prostor se sastoji od dva elementa: objekt s okidačem i njegov dijete objekt sustav čestica. Objekt s okidačem je bitan jer ovaj objekt ne smije blokirati put igraču tako da se sudari s njime nego samo da se detektira kad je igrač u njemu. Objekt sa sustavom čestica je kreiran tako da igraču vizualno označi veličinu tog prostora.

Sustav čestica se sastoji od petlje koja traje 5 sekundi. Svaka pojedina čestica male

je veličine i polako se kreće prema gore. Roze su boje tako da odstupaju od pozadine i da budu dobro vidljive. Varijabilne su duljine trajanja od 10 do 15 sekundi. Njihova veličina se mijenja kroz životni vijek, prvo kad se stvori čestica je mala pa se polako povećava do sredine životnog vijeka i onda se ponovno smanjuje sve dok ne nestane. Na taj način se stvara dojam glatkog stvaranja i nestajanja čestica. Inače bi one naglo nastale i naglo nestale što bi bilo distraktirajuće i neprirodno.



Slika 26: Izgled prostora isključivanja moći u igri(Izvor: vlastita izrada)

U skripti za prostor isključenja moći prvo je implementirano dinamičko određivanje broja čestica. To se radi kad se objekt kreira. Prvo je potrebno dohvatiti sve potrebne komponente sustava čestica da se može modificirati količina čestica koja se kreira. Uzmu se vrijednosti skaliranja okidača objekta za prostor te se stavi na vrijednosti skaliranja sustava čestica tako da oboje budu iste veličine. Broj čestica koji se kreira se odredi tako da se pomnože visina i širina okidača i podijele s dva. Na taj način gustoća čestica ostaje konzistentna kroz veličine okidača i dobiva se objekt koji se lagano može ponovno iskoristiti.

```
void Start()
{
    ParticleSystem particleSystemComponent = particleSystemObject.GetComponent<ParticleSystem>();
    ParticleSystem.ShapeModule shape = particleSystemComponent.shape;
    ParticleSystem.EmissionModule emission = particleSystemComponent.emission;
    shape.scale = new Vector3(transform.localScale.x, 1, transform.localScale.y);
    emission.rateOverTime = Mathf.Round(transform.localScale.x * transform.localScale.y / 2);
}
```

Slika 27: Kod za dinamičko određivanje količine čestica(Izvor: vlastita izrada)

Kada nešto uđe u prostor isključivanja prvo se provjeri što je ušlo. Ako se radi o projektilu koji je igrač ispucao taj projektil se ispali u smjeru suprotnom kojim je ušao. Projektil se ispaljuje van silom koja je jednaka 30% njegove brzine. Projektil se van lansira brže nego ga igrač može ispucati i to će se kasnije iskoristiti u dizajnu prepreke. Ako se radi o igraču poziva se njegova funckija **DisableGravityReversal** koja gasi moć promjene gravitacije. Za ostale objekte se samo promjeni smjer djelovanja gravitacije na njih, npr. neka platforma.

```
private void OnTriggerEnter2D(Collider2D collision)
{
    if(collision.CompareTag("Player"))
    {
        collision.GetComponent<PlayerGravityReversal>().DisableGravityReversal();
    }
    else if(collision.CompareTag("PlayerProjectile"))
    {
        collision.attachedRigidbody.AddForce(collision.attachedRigidbody.velocity * -0.3f, ForceMode2D.Impulse);
    }
    else
    {
        collision.attachedRigidbody.gravityScale = -1 * collision.attachedRigidbody.gravityScale;
    }
}
```

Slika 28: Kod za razrješavanje ulaska objekta u prostor(Izvor: vlastita izrada)

Kada lik igrača izađe iz tog prostora njemu se vraća moć utjecaja na gravitaciju. Ovdje se samo provjerava da li se radi o igraču jer je samo za lika igrača bitno da mu se vrati moć. Ovdje je vidljiva još jedna funkcija skripte igrača imena **EnableGravityReversal**.

```
private void OnTriggerExit2D(Collider2D collision)
{
    if (collision.CompareTag("Player"))
    {
        collision.GetComponent<PlayerGravityReversal>().EnableGravityReversal();
    }
}
```

Slika 29: Kod za razrješavanje izlaska objekta iz prostora(Izvor: vlastita izrada)

U skripti igrača ulaskom u područje isključenja moći varijabla **enableGravityReversal** se postavlja na laž i gravitacija se vraća na normalno stanje. Svira se zvučni efekt i poziva se Unity događaj **gravityReversalState** s argumentom koji prati da li je dozvoljeno korištenje moći.

Izlaskom lika igrača iz područja isključenja moći varijabla **enableGravityReversal** se postavlja na istinu i poziva se Unity događaj **gravityReversalState** s argumentom koji prati da li je dozvoljeno korištenje moći. Taj argument se prenosi tako da objekt korisničkog sučelja zna ako igrač može koristiti moć.

```
public void EnableGravityReversal()
{
    enableGravityReversal = true;
    gravityReversalState.Invoke(enableGravityReversal);
}
1 reference
public void DisableGravityReversal()
{
    enableGravityReversal = false;
    Physics2D.gravity = new Vector2(0, -9.81f);
    gravityChangeSound.Play();
    gravityReversalState.Invoke(enableGravityReversal);
}
```

Slika 30: Kod za razrješenje interakcije s područjem na igraču(Izvor: vlastita izrada)

U skripti za upravljanje korisničke sučeljem se poziva funkcija ikone **ChangeAbilityIconVisibility** s arugmentom **available** koji predstavlja ako igrač može ili ne može koristiti moć.

```
public void HandleHUDGravityReversalAvailabilityChange(bool available)
{
    abilityIconScript.ChangeAbilityIconVisibility(available);
}
```

Slika 31: Kod za pozivanje funkcije za promjenu vidljivosti ikone(Izvor: vlastita izrada)

U skripti ikone funkcija se prvo postavlja ikona na onu koja označava da gravitacija djeluje prema dolje, a stanje komponente ikone se postavlja vrijednost varijable **visible** tako da ako igrač može koristiti moć da bude ikona vidljiva, a ako ne može da bude nevidljiva.

```
public void ChangeAbilityIconVisibility(bool visible)
{
    icon.sprite = downIcon;
    icon.enabled = visible;
}
```

Slika 32: Kod mijenjanja ikone moći(Izvor: vlastita izrada)

2.4. Stvaranje prepreka

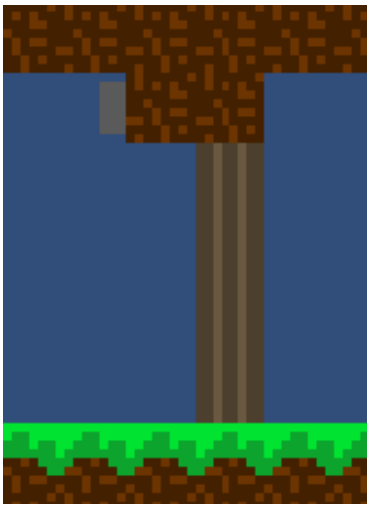
2.4.1. Vrata na gumb

Prepreke u ovoj igri se baziraju na dvije stvari: mehaničke sposobnosti i domišljatost. Prepeke koje zahtijevaju mehaničke sposobnosti zahtjevaju od igrača da pokaže sposobnosti ispravnog i preciznog kretanja, skakanja i pucanja. Domišljatost je potrebna kada igrač mora smisliti kako da iskoristi moć promjene gravitacije da savlada prepreku.

Najosnovnija prepreka u igri su vrata na gumb. Ova prepreka se bazira na tome da igrač ispali projektil i pogodi sivi gumb da se otvore vrata. Ova vrata se koriste kroz razne dijelove igre i mogu se kombinirati s ostalim preprekama da se oteža pogađanje gumba. U igri ova prepreka se sastoji od glavnog objekta koji služi kao objekt koji sadržava dijete objekte gumba i vrata. Na tom objektu se nalazi skripta koja se zaslužna za upravljanje tom preprekom. Na gumbu se nalazi skripta za provjeru ako nešto dotakne gumb, a na objektu vrata je skripta za pomicanje vrata kada projektil igrača pogodi gumb. Vrata su ponovno iskorišten objekt platforme koja se može pomicati.

Kada nešto dotakne gumb prvo se provjerava ako se radi o projektilu igrača. Ako je projektil igrača pogodio gumb tada se poziva Unity događaj **ButtonTriggered** i svira se zvučni efekt. Ovdje je korišten Unity događaj tako da objekt gumba bude lako iskoristiv u drugim preprekama jer se njemu lako može preko Unity uređivača pridružiti funkcija razrješavanja događaja.

Ovome gumbu je pridružena funkcija **ButtonTriggered** u skripti za upravljanje preprekom koja poziva funkciju vrata za njihovo pomicanje.



Slika 33: Izgled vrata na gumb u igri(Izvor: vlastita izrada)

```
private void OnTriggerEnter2D(Collider2D collision)
{
    if (collision.gameObject.tag == "PlayerProjectile")
    {
        buttonActivated.Play();
        ButtonTriggered.Invoke();
    }
}
```

Slika 34: Kod za pogađanje gumba(Izvor: vlastita izrada)

```
public void ButtonTriggered()
{
    door.GetComponent<SimpleDoor>().MovePlatform();
}
```

Slika 35: Kod za pozivanje funkcije za pomicanje vrata(Izvor: vlastita izrada)

Pomicanje vrata je implementirano na način da postoje dva objekta koji označavaju lijevu i desnu točku između kojih se vrata mogu micati. Funkcija **MovePlatform** postavlja globalnu varijablu **triggered** na istina što označava da je gumb pritisnut. Kada je vrijednost **triggered** istinita zadovoljava se uvjet za pomicanje vrata. Prvo se provjerava udaljenost između vrata i točke prema kojoj se treba pomaknuti sadržano u varijabli **movingTowards**. Ako je udaljenost veća od 0.01 jedinica tada se vrata pomiču prema toj točki konstantnom brzinom. Kada vrata dođu na točku prema kojoj su se micala, mijenja se točka prema kojoj će se micati i varijabla **triggered** se postavlja na vrijednost laž tako da se vrata ne počnu ponovno micati.

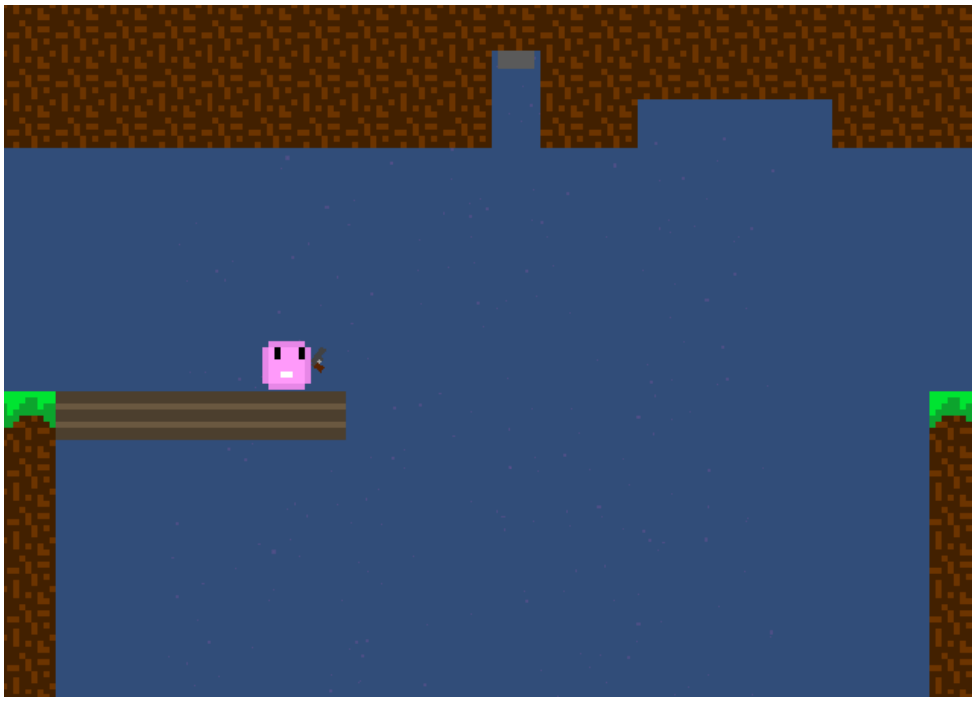
```
void Update()
{
    if (triggered)
    {
        if (Vector2.Distance(platform.transform.position, movingTowards.transform.position) > 0.01f)
        {
            platform.transform.position = Vector2.MoveTowards(platform.transform.position,
                movingTowards.transform.position, speed * Time.deltaTime);
        }
        else
        {
            triggered = false;
            if (movingTowards == leftMostPoint)
            {
                movingTowards = rightMostPoint;
            }
            else
            {
                movingTowards = leftMostPoint;
            }
        }
    }
}

1 reference
public void MovePlatform()
{
    triggered = true;
}
```

Slika 36: Kod za pomicanje vrata(Izvor: vlastita izrada)

2.4.2. Izmjenjujuće platforme

Ova prepreka se sastoji od gumba i dvije platforme. Kada igrač pogodi projektilom gumb, jedna platforma se povlači, a druga izlazi van. Ovdje je još dodano i područje isključenja moći promjene gravitacije tako da igrač ne može samo po stropu zaobići prepreku. Za uspješan prolazak igrač mora skočiti s jedne platforme i usred skoka pogoditi gumb tako da ispod njega dođe druga platforma. Prepreka je dizajnirana tako da igrač ne može pogoditi gumb bez da skoči s platforme. Uz to još je potrebno uzeti u obzir da se nalazi u području isključenja moći pa i projektili će biti ispaljeni u suprotnome smjeru.

U kodu ove prepreke sve je isto osim koda u skripti za upravljanje preprekom gdje se umjesto pozivanja funkcije za pomicanje platforme poziva kod za pomicanje dvije platforme, jedne prema lijevo i jedne prema desno.



Slika 37: Izgled prepreke izmjenjujućih platformi u igri(Izvor: vlastita izrada)

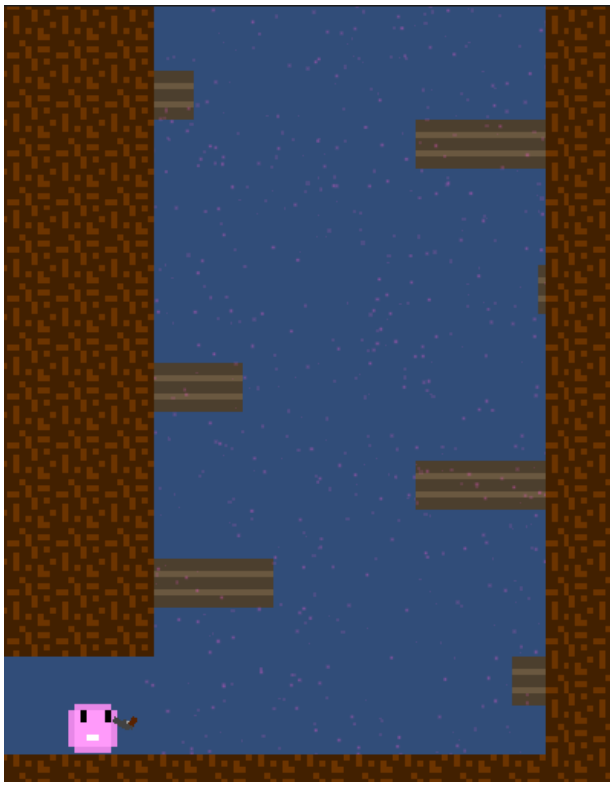
```
public void ButtonTriggered()
{
    leftPlatform.GetComponent<ExtendingPlatform>().MovePlatform();
    rightPlatform.GetComponent<ExtendingPlatform>().MovePlatform();
}
```

Slika 38: Kod za pozivanje funkcije za pomicanje vrata(Izvor: vlastita izrada)

2.4.3. Pomične platforme

Ova prepreka se sastoji od velikog broja platformi koje se cijelo vrijeme pomiču lijevo-desno. Ovdje igrač mora doći do vrha skakajući s platforme na platformu kako bi mogao doći do vrha. Platforme su složene tako da igrač nema mnogo vremena da razmišlja kako i kada da skoči. Tu se nalazi i područje isključenja moći tako da igrač ne može samo okrenuti smjer gravitacije i samo "pasti" prema gore da zaobiđe sve platforme.

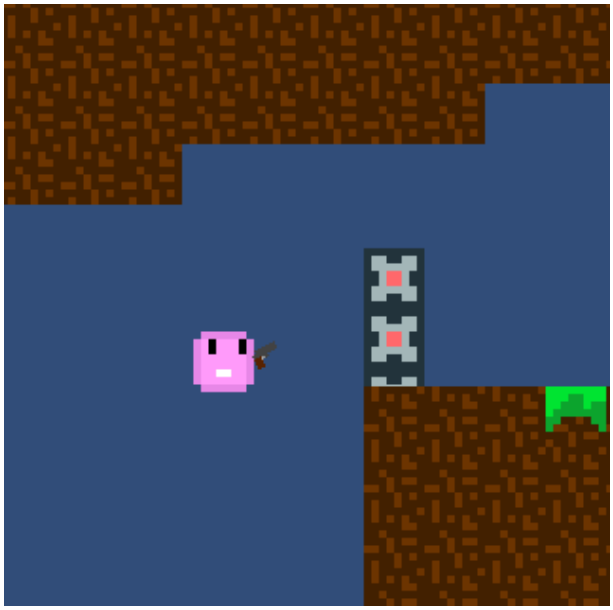
Ova prepreka je implementirana na način da su uzete prije korištene platforme i samo su pomaknute tako da imaju drukčije startne pozicije. U skripti je promijenjeno da nema nikakvog uvjeta za pomak platforme nego da se cijelo vrijeme kroz izvođenje igre pomiču.



Slika 39: Izgled prepreke pomičnih platformi u igri(Izvor: vlastita izrada)

2.4.4. Vrata u zidu

Ova prepreka se sastoji od vrata koja se nalaze u zidu. Iznad igrača na vrhu se nalazi prolaz koji ga vodi dalje i jedini način da dođe do njega je da iskoristi moć promjene gravitacije. Kada igrač dođe do vrha vidi da se odjedanput pojavljuju vrata koja zatvaraju prolaz prije nego što igrač može proći kroz njega.



Slika 40: Izgled prepreke "vrata u zidu" u igri(Izvor: vlastita izrada)

Ovdje igrač mora iskoristiti posebno kreiran dio terena razine da se pozicionira bliže prolazu tako da stigne proći kroz njega prije nego mu vrata zatvore put.

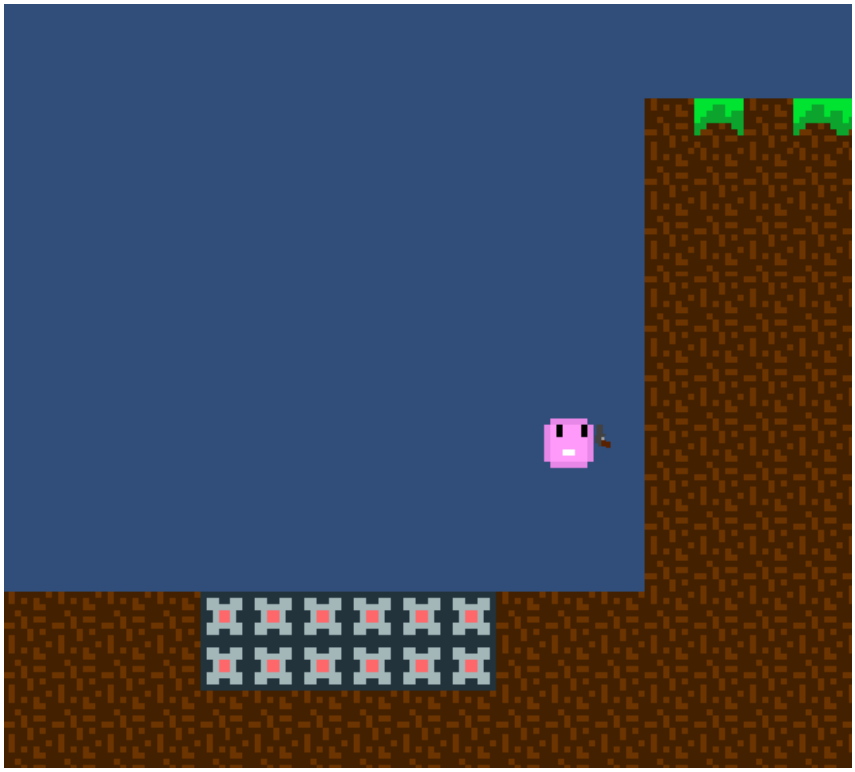
Za ispravnu funkcionalnost ove prepreke je bilo potrebno promijeniti neke elemente. Prvo je trebalo složiti da se pločice od kojih se sastoji razina prikazuju iznad svih ostalih elemenata. Inače, vrata bi se vidjela gdje se nalaze u zidu, a ovdje je bitno da se igrača iznenadi kako se vrata pojave iz zida. Vrata također moraju moći proći kroz zid stoga je bilo potrebno složiti da Unity ignorira neke kolizije. Ova skripta je dodana na posebni objekt postavljanje igre koji se brine da se postave sva "pravila" po kojima igra treba raditi. Ovdje se to sastoji od koda kojim se Unity podsustav za fiziku konfigurira da ignorira sudare između objekata s određenim pridruženim slojevima. Prva linija govori da se ignoriraju sudari između sloja **door** i sloja **ground** što je bitno za ovu prepreku. Iduća linija se brine da lik igrača ne ometa ispucavanje projketila lika igrača. Zadnja linija je da lik igrača ignorira sudare s posebnim sudaračima za zaustavljanje. To su posebni sudarači čija je svrha da zaustave razne objekte poput vrata u ovoj prepreci da ne padnu kroz pod u beskonačnost.

```
Physics2D.IgnoreLayerCollision(6, 9); //make door and ground ignore collision
Physics2D.IgnoreLayerCollision(7, 8); //make player and projectile ignore collision
Physics2D.IgnoreLayerCollision(8, 10); //make player and stop collider ignore collision
```

Slika 41: Kod za postavljanje pravila igre(Izvor: vlastita izrada)

2.4.5. Vrata u zidu s platformom

Ova prepreka je slična prošloj, ali sada ima dodanu platformu na sredini. Ta platforma je pod utjecajem gravitacije. Ovdje igrač mora proći kroz prolaz koji se nalazi iznad njega, ali kada igrač iskoristi moć promjene gravitacija ponovno ga blokiraju vrata koja dođu iz zida. Igrač mora iskoristiti platformu na sredini da uspije skočiti do prolaza. Platforma je složena da kada igrač iskoristi moć promjene gravitacije da ona stane u zraku. Kada igrač ponovno iskoristi moć one počinje padati, ali igrač pada brže od nje i tako ju može stići da ju iskoristi kao odskočnu dasku da prođe dalje. To je ujedno i kraj prve razine.



Slika 42: Izgled prepreke "vrata u zidu s platformom" u igri (Izvor: vlastita izrada)

2.4.6. Velika prostorija

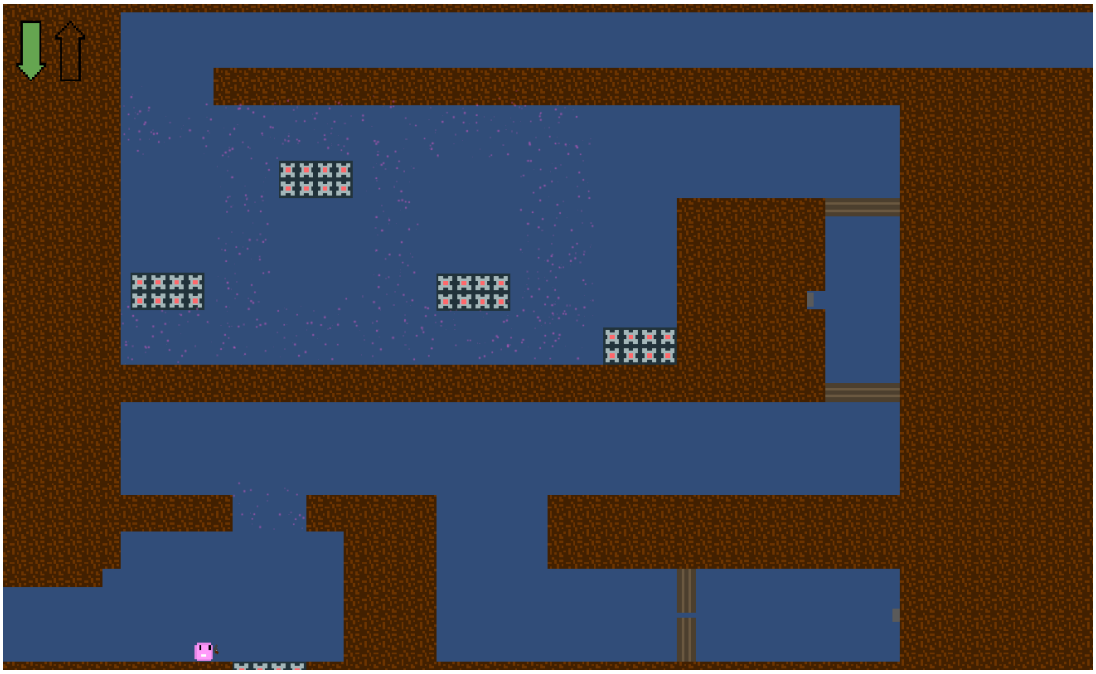
2.4.6.1. Prepreke

Na drugoj razini se nalazi velika prostorija u koju kada lik igrača uđe se udalji kamera tako da igrač ju može vidjeti u cijelosti. Kamera je na toj poziciji sve dok igrač ne dođe do kraja. Velika prostorija se sastoji od raznih prepreka koje igrač treba savladati. Prva prepreka se sastoji od jedne platforme koju igrač treba iskoristiti da može doći u unutrašnjost prostorije. Ako se lik igrača postavi na platformu i iskoristi moć promjene gravitacije, oboje će krenuti padati prema gore, ali čim lik igrača dotakne područje isključenja moći on će pasti na pod. Ovdje igrač treba iskoristiti moć bez da se nalazi na platformi tako da ona dotakne područje i padne sama na pod. Tada igrač treba ponovno vratiti gravitaciju na normalno i iskoristiti platformu kojoj je sad promijenjen smjer djelovanja gravitacije da dođe do vrha i skoči kroz ulaz prije nego

platforma ponovno padne dolje.

Sljedeće igrač treba pogoditi gumb koji će otvoriti vrata da ide prema gore. Tu su postavljene dvije "platforme" koje se miču na način da kontinuirano zatvaraju i otvaraju prostor gdje igrač može ispaliti projektil da pogodi gumb. Kada to igrač uspije, dolazi do još jednih vrata. Za otvaraje ovih vrata mora pogoditi gumb tijekom pada ili prema dolje ili prema gore.

Sljedeće igrač dolazi do najveće prepreke koja se sastoji od platformi koje se nalaze između dva područja isključenja moći tako da se kreću cijelo vrijeme gore-dolje. Ovdje igrač mora preciznim korištenjem svoje moći ili doticajem područja isključenja moći utjecati na platforme tako da se poravnanju u dobar ciklus u kojem može doći do zadnje platforme. Na zadnjoj platformi igrač mora skočiti kroz područje isključenja moći tako da na trenutak izađe van da može iskoristiti moć da padne na vrh.



Slika 43: Izgled velik prostorije u igri(Izvor: vlastita izrada)

2.4.6.2. Kamera

Kreiranje samih prepreka se sastojalo od kombiniranja postojećih elemenata i mehanika igre. Najveći izazov je nastao kod kreiranja glatke animacije pomicanja kamere. Cijela prostorija se nalazi u velikom objektu okidaču koji kada igrač uđe udalji kameru tako da se vidi cijela prostorija, a kada izađe se kamera vrati da ponovno prati lika igrača.

Pri ulasku igrača u prostoriju, prvo se provjerava ako se radi o liku igrača koji je ušao i provjerava se ako je već ta ista animacija aktivirana pomoću varijable **hasTriggered**. Animacija je implementirana pomoću korutina pa bi višestruko pozivanje korutine uzrokovalo probleme. Ako su uvjeti zadovoljeni, prvo se spremaju pozicija i zum kamere u varijable **preTriggerCameraPosition** i **preTriggerCameraDistance**. Te vrijednosti se koriste kasnije kao izvorišne točke od kojih započinju animacije kamere. Sljedeće se dohvaća i isključuje skripta za praćenje lika

igrača. Još se provjerava ako već postoji u izvođenju neka animacije kamere. Reference na korutine se mogu spremiti u varijable, ovdje je to **moveCamera**. Ako je neka animacije kamere već u tijeku ona se zaustavlja. Na kraju se pokreće korutina pomoću funkcije **ZoomOutCameraOverTime**.

```
private void OnTriggerEnter2D(Collider2D collision)
{
    if (collision.tag == "Player" && hasTriggered == false)
    {
        hasTriggered = true;

        preTriggerCameraPosition = cameraComp.transform.position;
        preTriggerCameraDistance = cameraComp.orthographicSize;

        mainCamera.GetComponent<CameraFollow>().enabled = false;

        if(moveCamera != null)
        {
            StopCoroutine(moveCamera);
        }

        moveCamera = StartCoroutine(ZoomOutCameraOverTime());
    }
}
```

Slika 44: Kod za pripremu prije pokretanja animacije kamere(Izvor: vlastita izrada)

U funkciji **ZoomOutCameraOverTime** se nalazi kod za animaciju kamere. Prvo se definiraju varijable **timerToEnd** i **progress**. Prva varijabla prati vrijeme koje je prošlo od početka animacije, a druga koliko je napredovala animacija. Varijabla **duration** sadrži vrijednost koliko dugo treba trajati animacija u sekundama. Sada se ulazi u petlju koja je zaslužna animaciju i ona traje sve do kraja animacije. Napredak animacije se izračuna tako da se podijeli vrijeme koje je prošlo od početka animacije s ukupnim trajanjem animacije. Ta ista vrijednost se koristi za izračun pomaka u funkciji **EaseInAndOut**. Ideja je da animacija ima spori početak, s vremenom ubrza i kad je pri kraju ponovno uspori. Za to je korištena trigonometrijska funkcija kosinus. Ovdje je funkcija kosinus oblikovana na način da za vrijednosti od 0 do 1 prođe jedan "uspon" od 0 do 1. To je vidljivo i na grafu funkcije. Vrijednost koja se dobi kosinusom se koristi u funkciji **Lerp** da se izračuna gdje se treba nalaziti kamera u tom trenutku izvođenja animacije. **Lerp** uzima kao argumente izvorišnu točku, odredišnu točku i neki argument **t** [5]. Ako bi se između izvorišne i odredišne točke potegnula crta, argument **t** predstavlja za koji postotak te crte da se izračunaju koordinate točke na toj crti. Vrijednost **t** od 0.5 bi se nalazila na pola te crte. Bitna stvar kod funkcije **Lerp** je da ona osigurava da bilo koji argument **t** se automatski ograniči na vrijednost 0 ako je manji od 0 ili na vrijednost 1 ako je veći od 1.

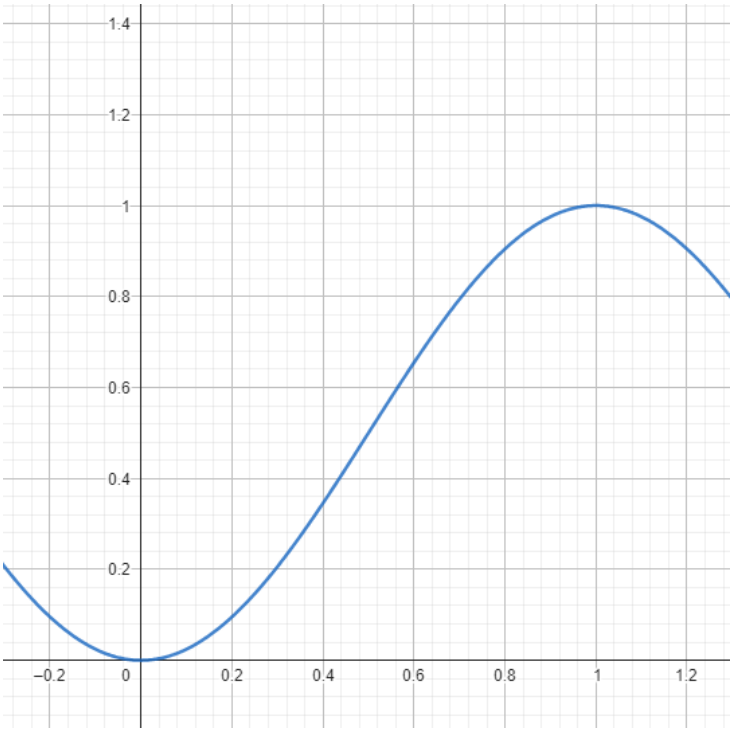
Kada je trenutna pozicija kamere neprimjetno udaljena od odredišta, kamera se postavi na odredište. Zbog različitih mogućih efekata kod brzine animacije kamere i modularnosti ove animacije moguće je da se dogodi da argument **t** funkcije **Lerp** nikad ne bude 1 i onda kamera nikad neće biti točno na odredišnoj poziciji. Ovdje nema opasnosti od petlje koja nikad neće završiti jer je ona ograničena vremenom, a ne udaljenošću. Na isti način je implementirana i promjena zuma kamere, koristeći svojstvo **orthographicSize** kamere.

```
private IEnumerator ZoomOutCameraOverTime()
{
    float timerToEnd = 0f;
    float progress = 0f;

    while (timerToEnd < duration)
    {
        progress = timerToEnd / duration;
        progress = EaseInAndOut(progress);

        cameraComp.transform.position = Vector3.Lerp(preTriggerCameraPosition, newCameraPosition, progress);
        if (Vector3.Distance(cameraComp.transform.position, newCameraPosition) < 0.02)
        {
            cameraComp.transform.position = newCameraPosition;
        }

        cameraComp.orthographicSize = Mathf.Lerp(preTriggerCameraDistance, newCameraDistance, progress);
        if (Mathf.Abs(cameraComp.orthographicSize - newCameraDistance) < 0.02)
        {
            cameraComp.orthographicSize = newCameraDistance;
        }

        timerToEnd += Time.deltaTime;
        yield return null;
    }
}
```

Slika 45: Kod za animaciju kamere(Izvor: vlastita izrada)

```
private float EaseInAndOut(float t)
{
    return -0.5f * (Mathf.Cos(Mathf.PI * t) - 1.0f);
}
```

Slika 46: Kod za izračun pomaka pomoću trigonometrijske funkcije kosinus(Izvor: vlastita izrada)



Slika 47: Graf funkcije korištene za udaljavanje kamere(Izvor: Geogebra)

Kada lik igrača izađe iz prostorije kod za pripremu ostaje isti, ali se poziva druga funkcija imena **MoveCameraToPlayer**. Ovdje je implementirano da se kamera vrati na lika igrača. Pozicija igrača se dobiva iz prenesenog objekta lika igrača koji se dobiva preko njegovog sudarača u trenutku izlaska iz velike prostorije. Petlja animacije se izvršava tako dugo dok pozicija kamere nije jednaka poziciji lika igrača. Ova animacija se može ograničiti vremenskim trajanjem, ali zbog robusnosti koda odlučeno je da traje sve dok se kamera ne nalazi na poziciji lika igrača. Varijabla **timer** se koristi za praćenje vremena koje je prošlo od početka animacije. Ideja je animacija počne sporo i da s vremenom postane sve brža. Za izračunavanje pomaka koristi se funkcija **EaseIn** s argumentom koliko vremena je prošlo od početka animacije. Matematička funkcija glasi x^2. Na grafu funkcije vidljiv spori početak rasta vrijednosti s ubrzavanjem. Na kraju kada se kamera nalazi na poziciji lika igrača, ponovno se upali skripta koja je zaslužna da kamera prati lika igrača.

```
private IEnumerator MoveCameraToPlayer(GameObject player)
{
    float timer = 0f;
    Vector3 playerPosition = player.transform.position;

    while(cameraComp.transform.position != playerPosition)
    {
        playerPosition = player.transform.position;
        cameraComp.transform.position = Vector3.Lerp(preTriggerCameraPosition, playerPosition, EaseIn(timer));
        if (Vector3.Distance(cameraComp.transform.position, playerPosition) < 0.02)
        {
            cameraComp.transform.position = player.transform.position;
        }

        cameraComp.orthographicSize = Mathf.Lerp(preTriggerCameraDistance, defaultCameraSize, EaseIn(timer));
        if (Mathf.Abs(cameraComp.orthographicSize - defaultCameraSize) < 0.02)
        {
            cameraComp.orthographicSize = defaultCameraSize;
        }

        timer += Time.deltaTime;

        yield return null;
    }

    mainCamera.GetComponent<CameraFollow>().enabled = true;
}
```

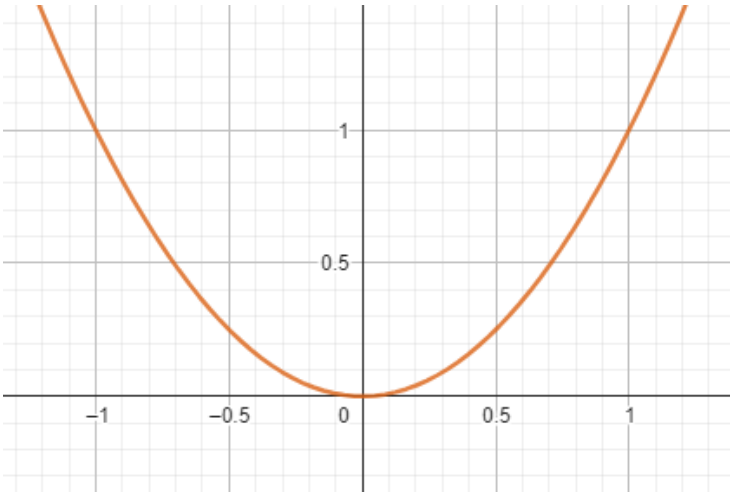
Slika 48: Kod za vraćanje kamere na lika igrača(Izvor: vlastita izrada)

```
private float EaseIn(float t)
{
    return t * t;
}
```

Slika 49: Kod za izračun pomaka pomoću kvadratne funkcije(Izvor: vlastita izrada)

2.4.6.3. Popravak gravitacije

U dijelovima igre gdje se kombiniraju platforme i područje isključenja moći nastaje jedan problem kada platforma dotakne to područje i igrač u tom trenutku iskoristi moć promjene gravitacije. Kada platforma uđe u to područje ona malo ostane unutra i ako u tom trenutku igrač



Slika 50: Graf funkcije korištene za vraćanje kamere na lika igrača(Izvor: Geogebra)

iskoristi moć, platforma će imati krivi smjer djelovanja gravitacije i ostati će u tom području. Zato je kreirana posebna skripta koja se brine da se to ne dogodi.

Ovdje se provjerava ako u područje isključenja moći uđe bilo što osim lika igrača. Ako se ne radi o liku igrača onda započinje korutina **PreventStuck**. Ona prvo čeka vrijeme definirano u sekundama u varijabli **allowedTimer**. Postavljeno je na 0.3 sekunde. U tu korutinu se prenosi referenca na objekt koji je sudjelovao u koliziji. Nakon tih 0.3 sekunde smjer djelovanja gravitacije se mijenja u suprotan. Ako u tih 0.3 sekunde objekt izađe van iz područja isključenja moći, korutina se zaustavlja jer nema potrebe mijenjati smjer djelovanja gravitacije ako objekt nije zapeo unutra.

```
private void OnTriggerEnter2D(Collider2D collision)
{
    if (!collision.CompareTag("Player"))
    {
        fixer = StartCoroutine(PreventStuck(collision));
    }
}

Unity Message | 0 references
private void OnTriggerExit2D(Collider2D collision)
{
    if(!collision.CompareTag("Player"))
    {
        if(fixer != null)
        {
            StopCoroutine(fixer);
        }
    }
}

1 reference
private IEnumerator PreventStuck(Collider2D collision)
{
    yield return new WaitForSeconds(allowedTimer);
    collision.attachedRigidbody.gravityScale = -1 * collision.attachedRigidbody.gravityScale;
}
```

Slika 51: Kod za ispravljanje smjera djelovanja gravitacije(Izvor: vlastita izrada)

2.5. Izbornici

2.5.1. Početni izbornik

Početni izbornik se prikaže kada se pokrene igra. Implementiran je pomoću Canvas, tj. platno komponente i taj objket je roditelj objekt cijelog izbornika. Sastoji se od dijete objekta pozadinske slike i 3 dijete objekata gumba za ulazak u prvu razinu, prikaz kontrola i izlazak iz igre. Korišteni su gumbi iz TextMeshPro paketa iz Unity trgovine za resurse. Taj paket ima više mogućnosti prikaza i prilagodbe nego što izvorno nudi Unity. Gumbi dolaze zajedno s TextMeshPro tekstom. Gumbi su kompletno transparetni, bez rubova i dobe malo bijelu pozadinu kada se prijeđe pokazivačem preko njih.

Slika 52: Izgled početnog izbornika(Izvor: vlastita izrada)

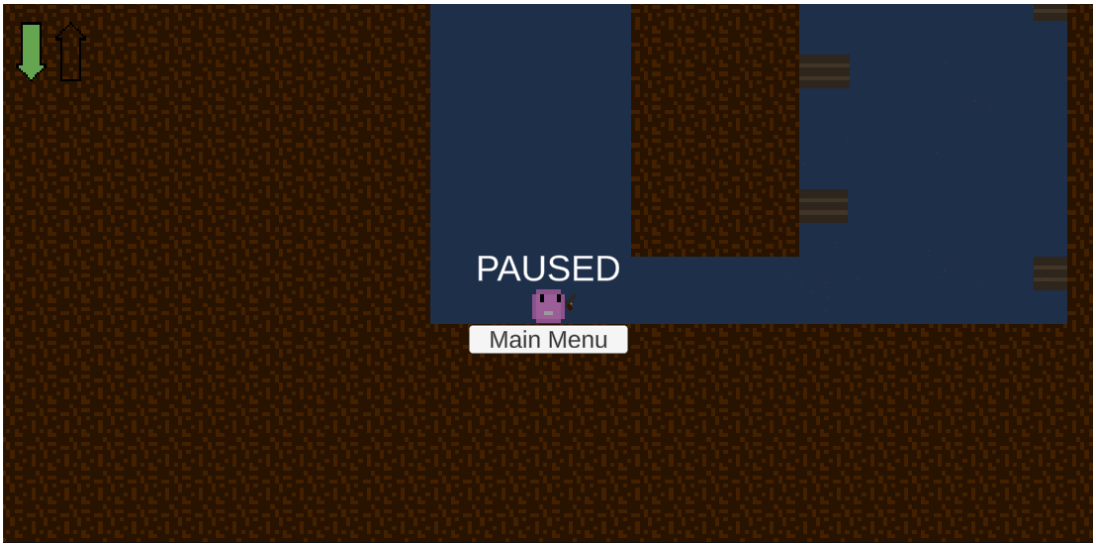
Što se događa kada igrač klikne na gumb je implementirano u skripti koja se nalazi na roditelj objektu izbornika. Gumbi koriste sustav Unity događaja pa se **OnClick** događaju mogu preko Unity uređivača lako pridružiti funkcije iz skripte na objektu izbornika. Sve razine su implementirane kao različite scene i njima se pristupa preko imena. Ekran za prikaz kontrola je isto implementiran kao posebna scena. Za zatvaranje igre potrebno je samo pozvati **Application.Quit**.

```
public void Play()
{
    SceneManager.LoadScene("Level_1");
}

0 references
public void HowToPlay()
{
    SceneManager.LoadScene("HowToPlay");
}

0 references
public void Quit()
{
    Application.Quit();
}
```

Slika 53: Kod početnog izbornika(Izvor: vlastita izrada)

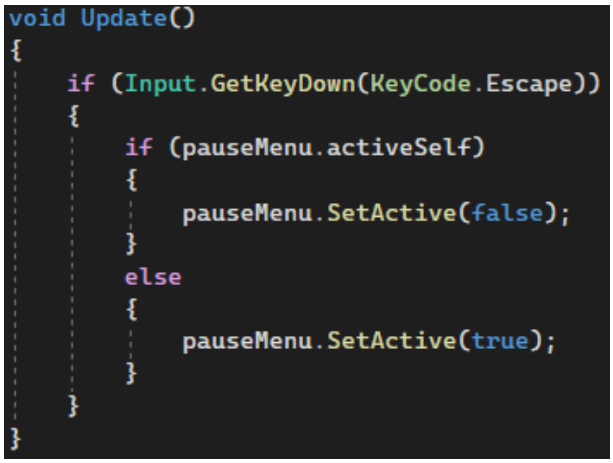
2.5.2. Pauziranje

Pauziranje je implementirano kao dio korisničkog sučelja igre. Sastoji se od roditelj objekta koji je tamnosiva slika koja se prikazuje kada igrač pauzira igru kao najgornji element sučelja tako da sve u pozadini izgleda tamnije jer je van fokusa.



Slika 54: Izgled izbornika pauze(Izvor: vlastita izrada)

Igra se pauzira pritiskom **Esc** gumba na tipkovnici. Kada igrač pritisne tipku za pauziranje, provjerava se ako je već aktivan izbornik pauze. Ako je već aktivan tada se gasi, a ako nije onda se upali. Varijabla **pauseMenu** sadrži referencu na objekt izbornika pauze.



Slika 55: Kod za provjeru ako je pritisnuta tipka za pauziranje(Izvor: vlastita izrada)

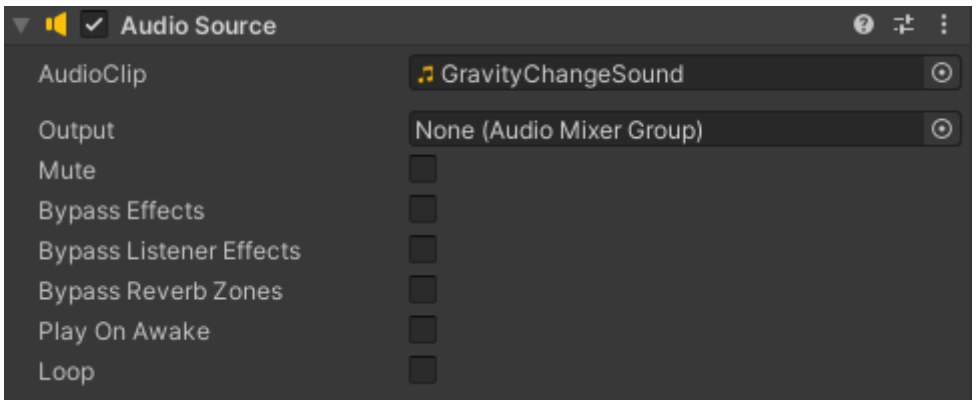
U skripti za pauziranje igre prvo kada igra započne se isključuju izbornik pauze i dohvaćaju se skripte iz objekata igrača i oružja. Definirane su funkcije **OnEnable** i **OnDisable**. One se izvršavaju kada se izbornik pauze uključi i isključi. Osnovni način da se pauzira igra je da se postavi protok vremena na 0 pomoću **Time.timeScale** [6]. To zaustavlja izvršavanje **FixedUpdate** funckije, ali **Update** će se i dalje izvršavati. Pomicanje lika igrača se nalazi u **FixedUpdate** no sve ostalo se nalazi u **Update** funkciji kao što su ispaljivanje projektila, korištenje moći promjene gravitacije. Zato je još dodatno potrebno kompletno isključiti skripte koje se brinu za te stvari. Inače bi igrač i kad je igra pauzirana mogao ispaljivati projektile i koristiti moć. Za nastavak igre potrebno je ponovno upaliti isključene skripte i vratiti protok vremena na normalnu vrijednost. Na izborniku pauze se još nalazi i gumb za povratak na početni izbornik i za njega je definirana funkcija **GoToMainMenu** koja kada igrač pritisne gumb učita scenu početnog izbornika.

```
void Start()
{
    this.gameObject.SetActive(false);
    playerScript = player.GetComponent<Player>();
    weaponScript = weapon.GetComponent<Shooting>();
    playerGravityReversalScript = player.GetComponent<PlayerGravityReversal>();
}

Unity Message | 0 references
private void OnEnable()
{
    Time.timeScale = 0f;
    playerScript.enabled = false;
    weaponScript.enabled = false;
    playerGravityReversalScript.enabled = false;
}

Unity Message | 0 references
private void OnDisable()
{
    Time.timeScale = 1f;
    playerScript.enabled = true;
    weaponScript.enabled = true;
    playerGravityReversalScript.enabled = true;
}

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

Slika 56: Kod za pauziranje igre(Izvor: vlastita izrada)

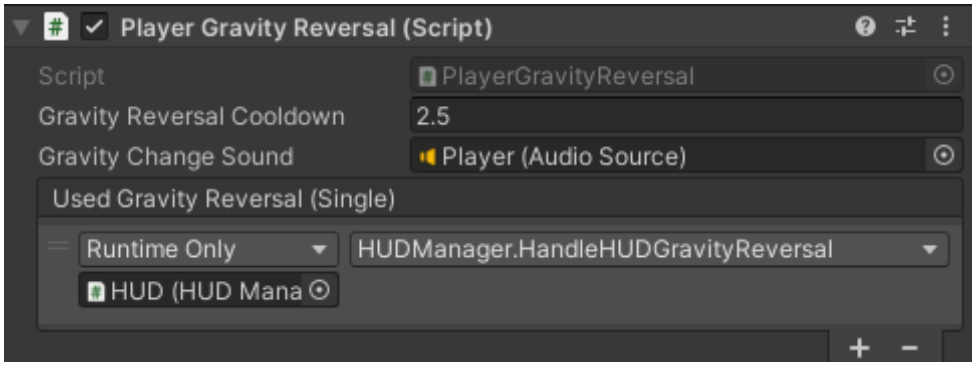
2.6. Zvučni efekti

Za zvučne efekte korišten je "Casual Game Sounds" paket iz Unity dućana resursima. Zvučni efekti su implementirani tako da je na objekte dodana komponenta **Audio Source** i njoj pridružen neki zvučni efekt.



Slika 57: Komponenta za izvor zvuka(Izvor: vlastita izrada)

U Unity uređivaču je moguće pridružiti reference na komponente tako da im se može direktno pristupiti bez dohvaćanja komponente programski i određivanja o kojoj se radi. S referencom na komponentu se tako mogu i svirati zvučni efekti.



Slika 58: Referenca na komponentu izvora zvuka na skripti(Izvor: vlastita izrada)

3. Zaključak

Izrada ove igre je bio zanimljiv i zahtjevan posao. Unity se pokazao kao jednostavan sustav s mnogo dubine. Implementacija bilo čega što je koristilo Unity sustav fizike pokazalo se jako napornim jer je zahtjevalo puno testiranja i namještanja brojki da se dobro osjeća. Modificiranje mase objekata, djelovanje trenja, međusobnog djelovanja objekata tijekom sudaranja je uzelo mnogo vremena. Puno toga se može otkriti eksperimentiranjem, ali za neke specifičnosti je potrebno istražiti dobre prakse. Implementacija dodatnih stvari oko skoka da se bolje osjeća je definitivno stvar koja se uzima zdravo za gotovo i velik broj igrača očekuje da bude implementirano.

Unity se sam po sebi pokazao kao robusan stroj igara koji je zadovoljio sve potrebe ovog projekta. Igra je uspješno kreirana i postignuti su željeni ciljevi. Implementirane su sve željene mehanike i testiranjem se pokazalo da su dobro implementirane.

Unity sustav pločica se pokazao malo sklon greškama. U Unity uređivaču znale su se dogoditi greške gdje nije bilo moguće promijeniti pločicu koja se postavlja ili promjeniti način rada kista te jedino što je preostalo je bilo ponovno pokrenuti Unity. Sustav sam po sebi je jednostavan za korištenje kad se jedanput kreiraju resursi pločica. Jedini nedostatak je što nema podrške za pomicanje individualnih pločica ili grupe pločica. Jedini način da se to postigne da se kreira posebni objekt samo za to što može učiniti izbornik sa slojevima pločica nepreglednim.

Popis literature

[1] G. link, Facebook, Twitter, Pinterest, Email i O. Apps, *Improve annoying jump controls with coyote time and jump buffering - unit game development tutorial*, en-GB, ožujak 2021. adresa: https://www.ketra-games.com/2021/08/coyote-time-and-jump-buffering.html (pogledano 28. 8. 2023.).

[2] U. Technologies, *Unity - Manual: UnityEvents*, en. adresa: https://docs.unity3d.com/Manual/UnityEvents.html (pogledano 1. 9. 2023.).

[3] *Steam Hardware & Software Survey*. adresa: https://store.steampowered.com/hwsurvey (pogledano 3. 9. 2023.).

[4] U. Technologies, *Unity - Manual: Coroutines*, en. adresa: https://docs.unity3d.com/Manual/Coroutines.html (pogledano 7. 9. 2023.).

[5] U. Technologies, *Unity - Scripting API: Vector3.Lerp*, en. adresa: https://docs.unity3d.com/ScriptReference/Vector3.Lerp.html (pogledano 10. 9. 2023.).

[6] J. French, *The right way to pause a game in Unity*, en-GB, veljača 2020. adresa: https://gamedevbeginner.com/the-right-way-to-pause-the-game-in-unity/ (pogledano 11. 9. 2023.).

Popis slika