Izrada 2D akcijske videoigre u programskom alatu Unity

Vuljak, Dominik

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Dominik Vuljak

IZRADA 2D AKCIJSKE VIDEOIGRE U PROGRAMSKOM ALATU UNITY

ZAVRŠNI RAD

Varaždin, 2023.

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Dominik Vuljak

Matični broj: 0016141425

Studij: Informacijski sustavi

IZRADA 2D AKCIJSKE VIDEOIGRE U PROGRAMSKOM ALATU UNITY

ZAVRŠNI RAD

Mentor:

Doc. dr. sc. Mladen Konecki

Varaždin, kolovoz 2023.

Dominik Vuljak

Izjava o izvornosti

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

Autor potvrdio prihvaćanjem odredbi u sustavu FOI-radovi

Sažetak

Ovaj rad bavi se izradom 2D akcijske igre u programskom Alatu *Unity*. Ono što je u centru promatranja je implementiranje kretanja glavnog lika i neprijatelja, implementiranje sakupljanja životnih bodova i zlatnika te napad na neprijatelja, odnosno napad neprijatelja na glavnog lika. Također će biti prikazan način na koji se prikazuju životni bodovi te drugi podaci bitni za pobjedu u igri.

Ključne riječi: Unity, 2D, akcijski žanr, C#, skripta

Sadržaj

1. Uvod

Ovaj završni rad baziran je na izradi 2D akcijske igre u programskom alatu *Unity*. Motivacija za ovaj rad je osobni interes za učenjem razvoja videoigara. te željom za daljnjim napredovanjem u tom području.

Ono što je bila glavna ideja pri izradi videoigre je da se implementiraju sve glavne radnje koje su osnovne za sve 2D igre ovog žanra, a u to spadaju kretanje, napadanje te sakupljanje raznih predmeta, u ovom slučaju zlatnika. Neprijateljima je omogućeno kretanje između dvije koordinate X osi, a nekim vrstama neprijatelja je omogućeno ispaljivanje projektila u smjeru glavnog lika. Uz sve to, implementiran je prikaz životnih bodova te prikazi eliminiranih neprijatelja i sakupljenih zlatnika s obzirom da su to uvjeti za pobjedu u igri.

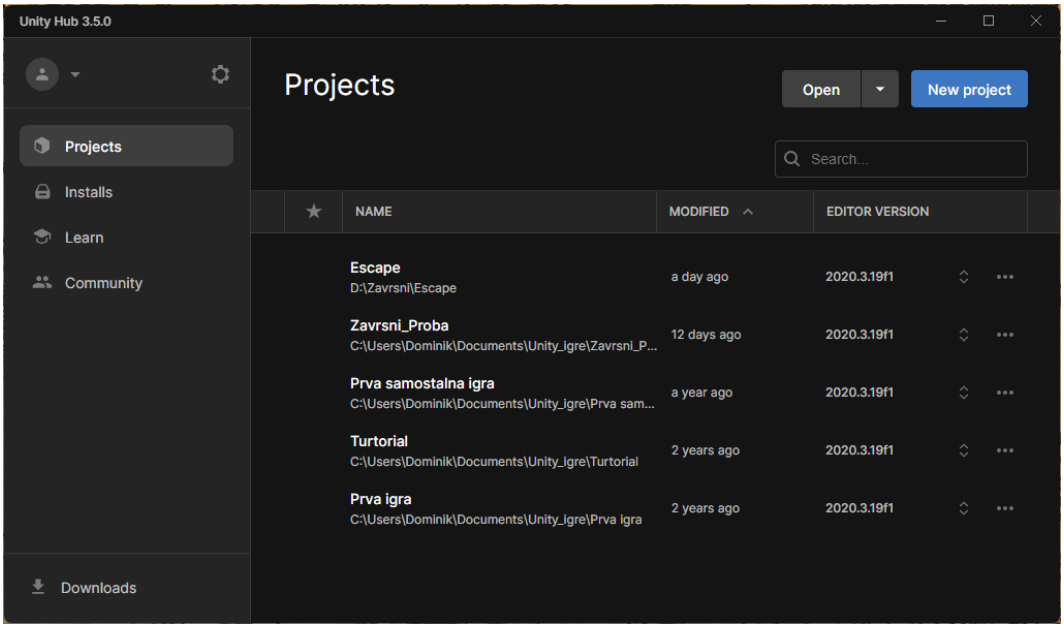
2. Metode i tehnike rada

Za potrebe ovog rada korišteno je nekoliko različitih tehnologija koje su korištene u radu te su one opisane u nastavku.

2.1. Unity

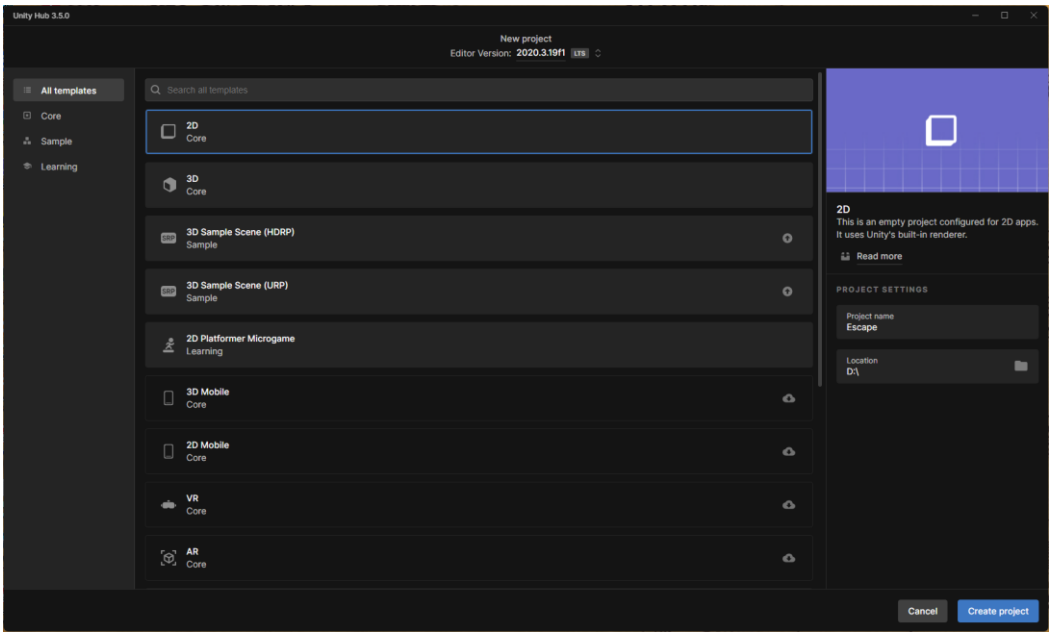
„Unity je real-time platforma za stvaranje 2D i 3D aplikacija, najčešće video igara, uz pomoć C# jezika i .NET razvojnog okvira“ [1]. Unutar Unitya je moguće stvarati aplikacije za više od 25 platformi koje su dostupne unutar same platforme, od mobilnih uređaja i osobnih računala pa sve od uređaja za virtualnu stvarnost. Korištenje Unitya je besplatno, a moguće je na Windows i macOS operacijskim sustavima dok je na Linuxu moguće samo na distribucijama Ubuntu 16.04 i Ubuntu 18.04 te na distribuciji CentOS7. [2]

Kako bismo pokrenuli alat Unity, prvo je potrebno pokrenuti program Unity Hub, prikazan na doljnjoj slici, koji će nam prikazati popis naših dosadašnjih projekata, ali će nam i ponuditi mogućnost stvaranja novog projekta te otvaranja projekta koji još do sada nismo otvarali.



Slika 1 Unity Hub

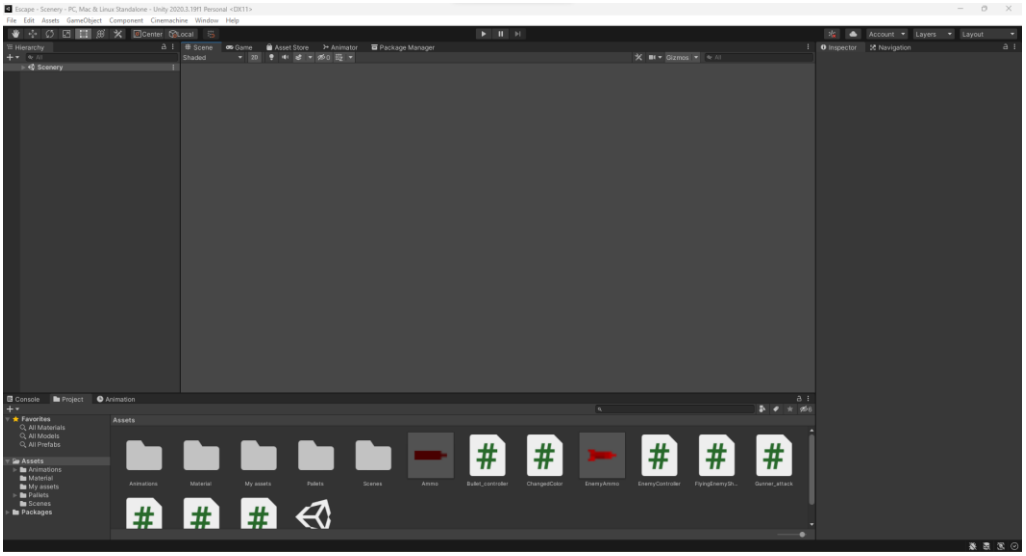
U slučaju da želimo stvoriti novi projekt, kliknuti ćemo na gumb *New project* u gornjem lijevom kutu te će se otvoriti prozor za stvaranje novog projekta, prikazan na slici ispod.



Slika 2 Mogućnosti za stvaranje novog projekta

Kao što je prikazano, Unity nam nudi različite mogućnosti za razvijanje videoigara. Za potrebe ovog rada, odabrana je opcija 2D igre.

Nakon što je stvoren novi projekt, prikazati će nam se ekran kao na slici ispod.



Slika 3 Prazni projekt

Glavni dio ekrana raspoređen je u 4 dijela. Lijevo je hijerarhija prikaza objekata na ekranu videoigre, desno su informacije o označenom objektu, na dnu konzola i pregled materijala, a u središnjem su dijelu prikaz videoigre, ekran za kreiranje scene, te još neke kartice koje će se usput koristiti. Sve se kartice mogu premještati prema potrebama korisnika kako bi se

stvorilo što bolje iskustvo korištenja. Na ovoj su slici već dodani neki materijali koji će se koristiti u ovom radu.

2.2. Visual Studio

„Visual Studio je moćan razvojni alat koji se koristi za sve korake razvojnog ciklusa na jednom mjestu.“[3] Unutar njegovog okruženja moguće je pisati kod, kompajlirati ga, ortkrivati pogreške u kodu, testirati kod te na kraju razviti cijelu aplikaciju. Osim toga, u Visual Studiu je moguće povezati se na git sustav za kontrolu verzija koda te na Gituhub kako bi se omogućilo zajedničko razvijanje aplikacija.

2.3. GIMP

„GIMP je akronim za GNU Image Manipulation Program. To je besplatano distribuiran program za zadatke poput retuširanja fotografije, kompozicije slike i autorstvo slike.“[4] Program nudi mogućnosti slikanja uz pomoć raznih alata, naprednu manipulaciju slikovnim datotekama, stvaranje animacija te upravljanje raznim formatima datoteka.

2.4. itch.io

„itch.io je otvorena trgovina za nezavisne digitalne autore s naglaskom na nezavisne video igre.“[5] Ova platforma omogućava svakome pordaju svojeg sadržaja. Autori imaju potpunu slobodu u postavljanju cijene svog sadržaja, upravljanju prodaje te dizajnu svojih stranica. Da bi se neki sadržaj odobrio potrebno je skupiti glasove, praćenja ili „likeova“, a promjene na sadržaju se mogu raditi prema autorovoj želji. Za potrebe ovog rada koristiti će se sadržaj preuzet s ove trgovine.

2.5. C#

„C# je moderan, objektno orijentirani i *type-safe* programski jezik.“[6] S obzirom da ima korijene u C programskom jeziku, jezik je vrlo sličan jezicima C, C++, Java i JavaScript. Trenutna verzija ovog jezika je C# 11. Skoro sve aplikacije pisane u programskom jeziku C# koriste .NET okruženje koje će biti korišteno i za potrebe ovog rada.

3. Razrada teme

U ovom dijelu rada biti će opisan žanr akcijske igre te će biti prikazan postupak izrade odabrane igre. Biti će prikazano kako se koriste skripte za upravljanje elementima unutar Unity-a te koje sve komponente možemo dodavati elementima u igri.

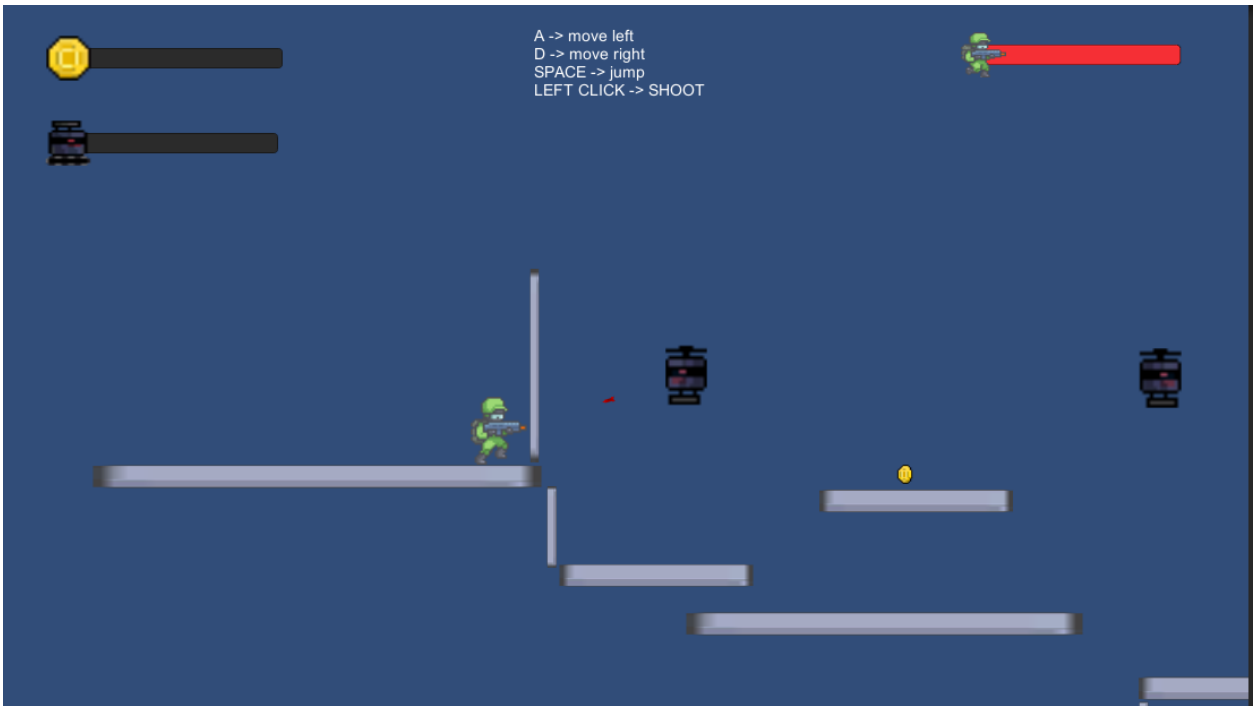
3.1. Žanr igre

„Akcijska igrica je svaka igrica koja zahtjeva od igrača da izvrši izazove koji često zahtjevaju dobru koordinaciju očiju i ruku i reakcije. Ovakvi tipovi videoigara su obično brze i u njima najčešće uživaju oni koji često igraju videoigre.“ [7]
Postoji nekoliko podžanrova akcijskih videoigara[7]:

- Battle royale igre
- FPS (first-person shooter) igre
- Borbene igre
- Hack and slash igre
- Igre labirinta
- Platformske igre
- Side-scrollers igre

3.2. O igri

Videoigra koja je stvorena za potrebe ovog rada nosi naziv *Escape* jer je cilj *pobjeći* svojim neprijateljima te ih sve eliminirati prije nego oni eliminiraju igrača, a uz to, potrebno je sakupiti sve zlatnike.

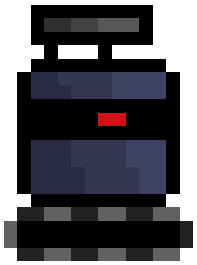


Slika 4 Izgled videoigre

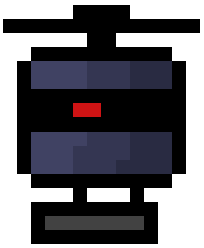
Glavni lik ove igre je zeleni vojnik koji je nazvan *Gunner*. Njegovo se kretanje vrši pritiskom na tipke A, za kretanje u lijevu stranu, D, za kretanje u desnu stranu, te SPACE za skakanje. Pucanje se vrši lijevim klikom, a ciljanje je implementirano da prati poziciju pokazivača na ekranu.


Slika 5 Gunner

Unutar ove videoigre postoje dvije glavne vrste neprijatelja: leteći i zemljani. Zemljani se dijele na obične i napredne neprijatelje. Razlika između običnih i naprednih je ta što se napredni brže mogu kretati te ispaljuju projektile u smjeru glavnog lika dok obični to ne mogu. Leteći neprijatelji također ispaljuju projektile u smjeru glavnog lika, ali ih ispaljuju u većim vremenskim razmacima od naprednih zemljanih neprijatelja.



Slika 6 Običan zemljani neprijatelj



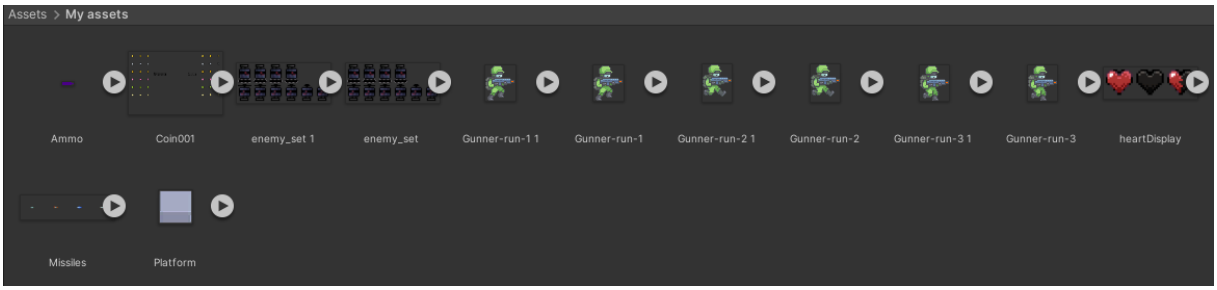
Slika 7 Leteći neprijatelj

Slika 8 Napredan zemljani neprijatelj

3.3. Proces izrade

3.3.1. Odabir materijala

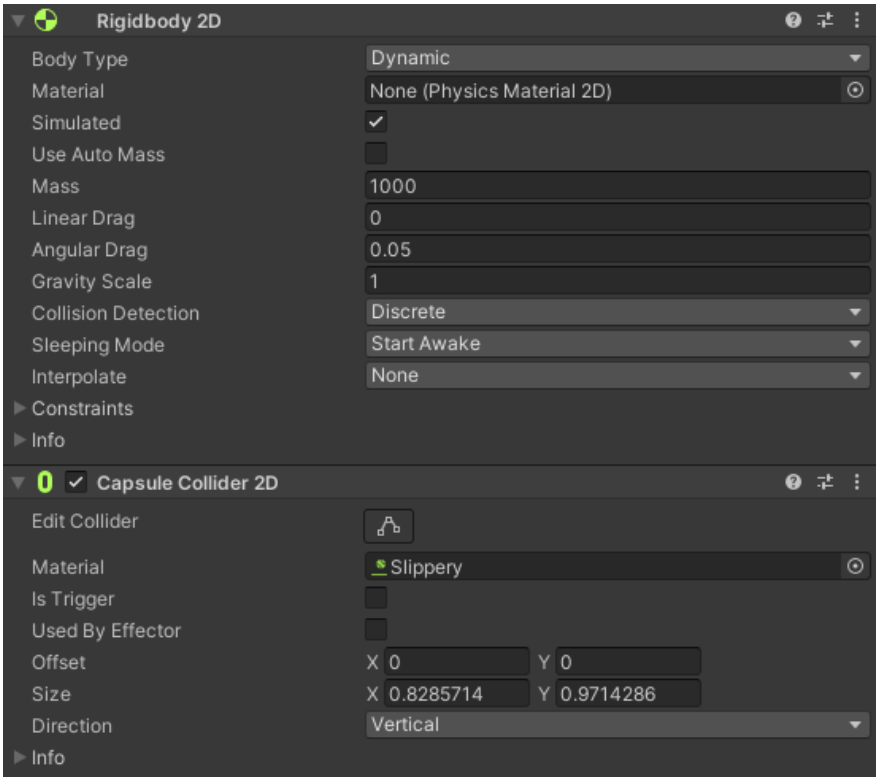
Kao što je već spomenuto, za potrebe ovog rada koristio se sadržaj preuzet sa stranice itch.io. Glavni lik ove videoigre te platforme koje su u njoj korištene preuzeti su sa stranice https://secrethideout.itch.io/team-wars-platformer-battle, projektili su preuzeti sa stranice https://mattwalkden.itch.io/lunar-battle-pack, neprijatelji sa stranice https://eragoth.itch.io/eragoths-tiny-platform-shooter, zlatnici sa stranice https://kevinshathotmailcom.itch.io/animated-coin-icon, a ikona srca koja se koristi za sakupljanje životnih bodova je preuzeta sa stranice https://rollinrock.itch.io/hearts-pixelart-assets. Preuzeti materijali dodani su u mapu My assets unutar mape Assets koja je sastavni dio projekta kako bi se njima lakše upravljalo.



Slika 9 Pregled mape My assets unutar programskog alata Unity

3.3.2. Dodavanje igrača i implementacija ponašanja igrača

Kako bi se u igru dodao glavni lik, dovoljno je bilo samo povući željenu sliku glavnog lika iz mape *My assets* u prostor za uređivanje scene. nakon što je glavni lik bio dodan u scenu, samom glavnom liku smo dodali dvije komponente: *Rigidbody 2D* i *Capsule Collider 2D*.



Slika 10 Rigidbody 2D i Capsule Colider 2D

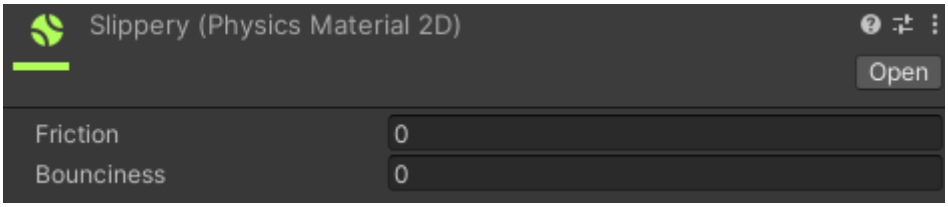
Rigidbody 2D definira težinu elementa, faktor gravitacije te još nekoliko varijabli koje u ovom radu nisu bile korištene. Ono što je za ovaj rad bilo najbitnije je faktor gravitacije koji omogućava da tijelo „pada“ sve dok se ne sudari s nekim drugim tijelom.

Ono što omogućava da se detektira sudar s drugim tijelom je komponenta *Capsule Collider 2D*. Capsule *Collider 2D* nam omogućava da definiramo *kapsulu* oko nekog tijela koja će definirati unutar kojeg područja će se registrirati sudar s nekim drugim tijelom ili elementom.

Da bi se sudar registrirao, potrebno je da i drugo tijelo na sebi ima implementiranu neku vrstu *Collidera* (*Box Collider 2D, Circle Collider 2D, ...*). Područje *Capsule Collider*-a je prikazano na sljedećoj slici.

Slika 11 Područje Capsule Collider-a označeno zelenom linijom

Unutar *Capsule Collidera* dodanje ručno kreiran materijal pod nazivom *Slippery* kako ne bi došlo do *zapinjanja* glavnog lika za platforme u slučaju pada s istih. U postavkama materijala postavljeno je da je koeficijent trenja jednak 0 kao bi se omogućila *skliskost*.



Slika 12 Postavke materijala Slippery

Sljedeće što treba postaviti za glavnog lika su kontrole kretanja. Da bi se kretanje moglo izvršiti, potrebno je glavnom liku dodati novu komponentu: skriptu *Gunner_controller*. Skripta je kod pisan u C# programskom jeziku. U skriptu je potrebno dodati metodu *Move*, koja omogućava kretanje lijevo i desno, i metodu *Jump*, koja omogućava skakanje. Obje metode pozivamo unutar *Update* metode. Unutar Update metode se nalazi još nekoliko metoda i dijelova koda koji će kasnije biti objašnjeni.

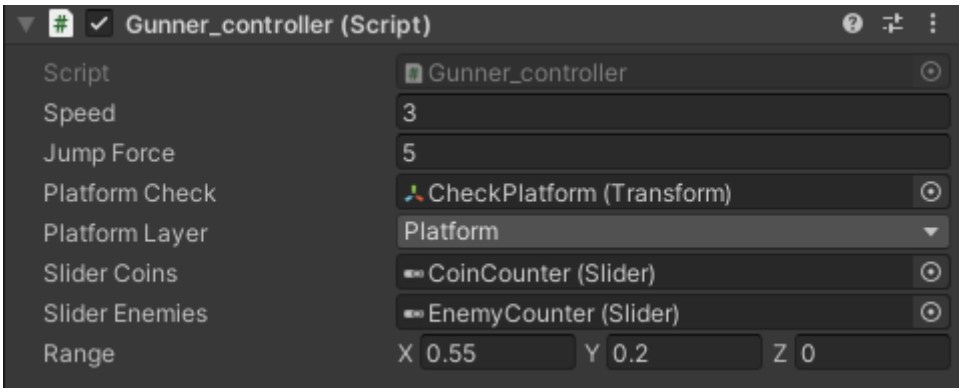
```
void Update()
{
    CountRemainingEnemies();
    coinPercentage = coinCount / maxCoins;
    enemiesEliminated = enemiesTotal - enemiesRemaining;
    enemiesPercentage = enemiesEliminated / enemiesTotal;
    sliderCoins.value = coinPercentage;
    sliderEnemies.value = enemiesPercentage;
    if (coinPercentage == 1 && enemiesPercentage == 1)
    {
        Time.timeScale = 0;
        foreach (GameObject restartBtn in GameObject.FindGameObjectsWithTag("restart"))
        {
            Vector3 newBtnPos = transform.position;
            newBtnPos.z = 100;
            newBtnPos.y = 0;
            newBtnPos.x = lastGunnerPosX;
            restartBtn.transform.position = newBtnPos;
        }
    }

    CollectCoins();

    Jump();

    Move();

    foreach (GameObject gunner in GameObject.FindGameObjectsWithTag("Player"))
    {
        lastGunnerPosX = gunner.transform.position.x;
    }
}
```

Slika 13 Metoda Update skripte Gunner_controller

```
void Move(){
    move = Input.GetAxisRaw("Horizontal") * speed;

    anim.SetFloat("Speed", Mathf.Abs(move));

    gunner.velocity = new Vector2(move, gunner.velocity.y);

    if (move > 0 && !directionRight || move < 0 && directionRight)
    {
        Flip();
        gameObject.GetComponent<Gunner_attack>().ChangeDirectionRight(directionRight);
    }
}
```

Slika 14 Metoda Move

Da bi ova metoda radila kako treba, morali odrediti vrijednost varijabli *speed*. To smo učinili tako da vrijednost nismo odredili unutar skripte već smo u skripti samo definirali da je varijabla tipa *float* i da je *public*. To nam omogućuje da unutar Unity-a unosimo vrijednost varijable radi lakšeg reguliranja brzine. Ovakav princip je korišten za još nekoliko varijabli tijekom razvoja ove videogire.



Slika 15 Varijable kojima je moguće mijenjati vrijednosti

Kao što vidimo, unutar metode *Move* provjeravamo u koju se stranu kreće igrač kako bi glavni lik bio okrenut prema stranu u koju se kreće. Metoda koja nam to omogućuje je *Flip*, a također pozivamo i metodu *ChangeDirectionRight* koja se nalazi u skripti *Gunner_attack*. Nju pozivamo kako bi se uz okretanje samog glavnog lika postiglo i okretanje smjera napada.

```
void Flip()
{
    directionRight = !directionRight;
    Vector3 transformScale = transform.localScale;
    transformScale.x *= -1;
    transform.localScale = transformScale;
}
```

Slika 16 Metoda Flip

```
public void ChangeDirectionRight(bool direction)
{
    directionRight = direction;
}
```

Slika 17 Metoda ChangeDirectionRight

Istu metodu kreirali smo i u skripti *Gunner_controller* kako bi se smjer kretanja promjenio u slučaju da napadamo u smjeru suprotnom od posljednjeg kretanja.

Također u metodi *Move* vidimo da je korištena varijabla anim koja je tipa *Animator*. Ta nam varijabla služi za upravljanje animacijama o kojima će kasnije biti riječi.

Uz sve ovo, potrebno je dodati već spomenutu metodu *Jump*.

```
void Jump()
{
    Collider2D platformHit = Physics2D.OverlapBox(platformCheck.position, range, 0, platformLayer);

    if(platformHit != null)
    {
        if (platformHit.gameObject.tag == "Platform" && Input.GetKeyDown(KeyCode.Space))
        {
            gunner.velocity = new Vector2(gunner.velocity.x, jumpForce);

        }
    }
}
```

Slika 18 Metoda Jump

Ova metoda provjerava je li lik u koliziji s platformom ispod sebe kako bi mogao napraviti skok. U slučaju da nije, skok se neće dogoditi.

Nakon kretanja, potrebno je isprogramirati ponašanje glavnog lika za napad. Za potrebe toga, kreirana je skripta *Gunner_attack*. U ovoj skripti unutar *Update* metode provjeravamo koja je trenutna pozicija pokazivača na ekranu kako bi se izračunao kut pod kojim će biti ispaljen projektil. Ako je pritisnuta lijeva tipka miša, tada će projektil biti ispaljen pod kutom koji je izračunat.

```
void Update()
{
    Vector3 mousePosition = Camera.main.ScreenToWorldPoint(Input.mousePosition);
    float xDiff = mousePosition.x - gunner.transform.position.x;
    float yDiff = mousePosition.y - gunner.transform.position.y;
    gunAngle = (float)Math.Atan2(yDiff, xDiff);
    gunAngle *= 180 / (float)Math.PI;
    fireFlag = Input.GetButton("Fire1");
    fireTimer -= Time.deltaTime;
    float move = Input.GetAxisRaw("Horizontal");
    if (move > 0 && !directionRight || move < 0 && directionRight) directionRight = !directionRight;
    if (fireFlag && (fireTimer < 0))
    {
        if ((gunAngle > 90 || gunAngle < -90) && directionRight || (gunAngle < 90 && gunAngle > -90) && !directionRight)
        {
            Flip();
            gameObject.GetComponent<Gunner_controller>().ChangeDirectionRight(directionRight);
        }
            ShootBullet();
        fireTimer = fireDelay;
    }
}
```

Slika 19 Metoda Update skripte Gunner_attack

Metoda *Flip* u ovom slučaju je ista kao u skripti *Gunner_controller*, a metoda *ShootBullet* ispaljuje projektil.

```
void ShootBullet()
{
    GameObject newBullet = Instantiate(ammoObj);
    newBullet.GetComponent<Bullet_controller>().FireShot(gunAngle, transform.position);
}
```

Slika 20 Metoda ShootBullet

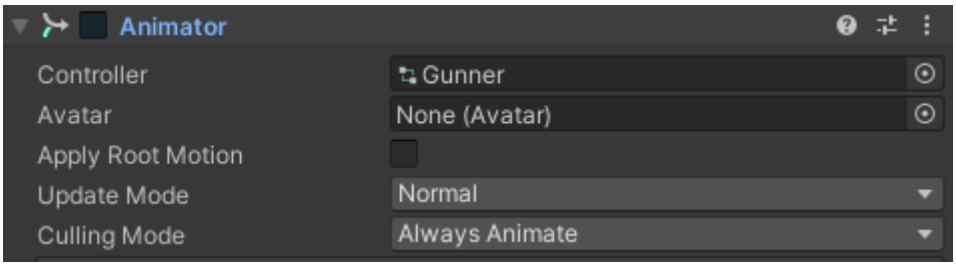
Za ispaljivanje projektila korištena je skripta *Bullet_controller* o kojoj će više riječi biti kasnije u radu.

U ovoj skripti postoje tri *public* varijable kojima možemo upravljati iz Unity-a, a prikazane su na idućoj slici.

Slika 21 Public varijable skripte Gunner_attack

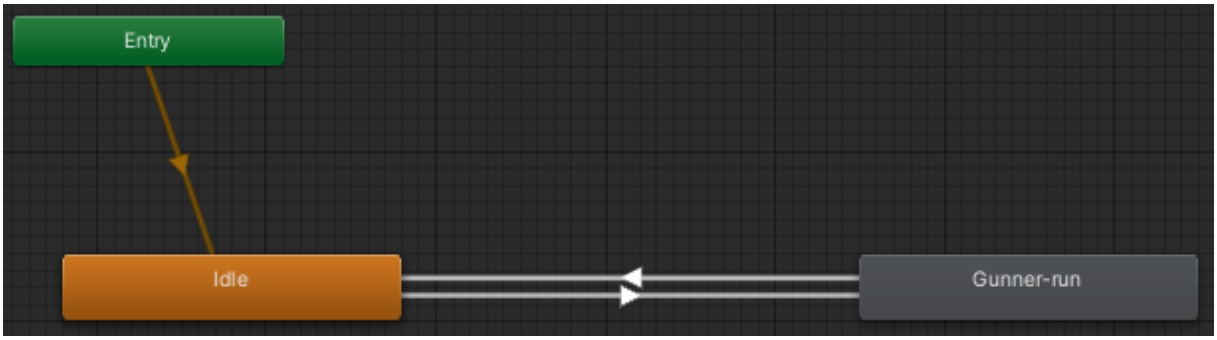
Varijabla *Ammo Obj* određuje koji će se projektil ispaliti, varijabla *Gunner* određuje koji će ju element ispaliti, a varijabla *Fire Delay* koji će biti razmak između dva ispaljivanja projektila.

Nakon što su osposobljene kontrole za kretanje i napad, potrebno je bilo dodati animaciju za kretanje. da bi bilo moguće korištenje animacija, potrebno je dodati komponentu *Animator*.



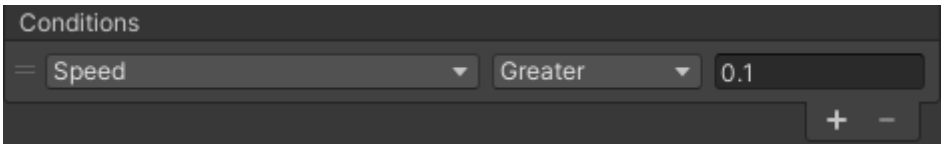
Slika 22 Komponenta Animator

Unutar *Animatora* smo kreirali novo moguće stanje našeg lika *Gunner-run*, uz stanja *Entry* i *Idle* koja su automatski generirana te smo definirali moguće prijelaze između stanja *Idle* i *Gunner-run*.



Slika 23 Mogući prijelazi između animacija

Kao što smo vidjeli u prikazu metode *Move*, uz pomoć varijable *anim* postavljamo vrijednost animatorskog paramtera pod nazivom *Speed* kako bismo videoigri dali do znanja koje stanje animacije se treba koristiti. Svaka veza između stanja ima definirano na koju vrijednost parametra *Speed* će se aktivirati. Tako će se veza od *Idle* prema *Gunner-run* aktivirati ako je *Speed* veći od 0.1, a veza od *Gunner-run* prema *Idle* ako je Speed manji od 0.1.

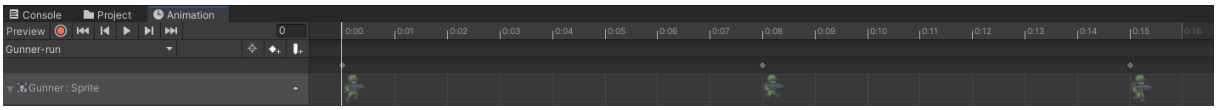


Slika 24 Uvjet prijelaza iz stanja mirovanja u stanje trčanja

Početno stanje parametra *Speed* postavljeno je na 0 u popisu parametra.

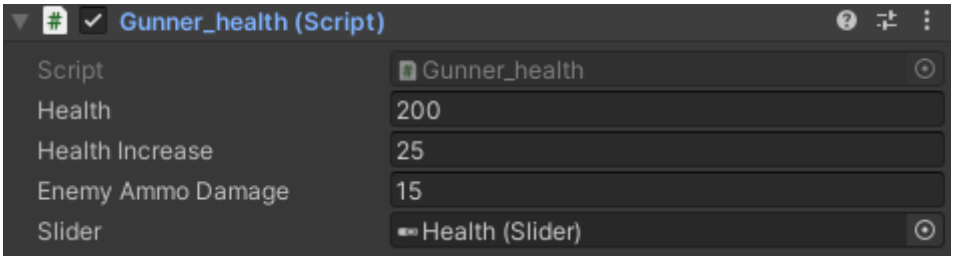
Slika 25 Početno stanje parametra Speed

Na kraju je potrebno odabrati sličice koje će se izmjenjivati prilikom animacije te odrediti njihovo trajanje.



Slika 26 Uređivanje animacije

Nakon animacija, na red dolazi upravljanje životnim bodovima. Za tu svrhu stvorena je skripta Gunner_health preko koje možemo definirati ukupan broj životnih bodova, količinu životnih bodova koji će se dodati ako igrač sakupi dodatne životne bodove tijekom igre te količinu životnih bodova koji će biti oduzeti u slučaju da igrača pogodi neprijateljski projektil.



Slika 27 Skripta Gunner_health u Unityu

Uz sve prethodno nabrojeno, određeno je i koji će s*lider* prikazivati postotak životnih bodova te će o *sliderima* biti riječi kasnije u radu.

```
void Update()
{
    healthPercentage = health/maxHealth;

    slider.value = healthPercentage;

    if (health <= 0)
    {
        Destroy(gameObject);
        Time.timeScale = 0;
        foreach (GameObject restartBtn in GameObject.FindGameObjectsWithTag("restart"))
        {
            Vector3 newBtnPos = transform.position;
            newBtnPos.z = 100;
            newBtnPos.y = 0;
            newBtnPos.x = lastGunnerPosX;
            restartBtn.transform.position = newBtnPos;

        }

    }
    if (CheckEnemyAmmoCollision())
    {
        health -= enemyAmmoDamage;
    }

    if (transform.position.y < -5) health = 0;

    AddHealth();

    foreach (GameObject gunner in GameObject.FindGameObjectsWithTag("Player"))
    {
        lastGunnerPosX = gunner.transform.position.x;
    }
}
```

Slika 28 Metoda Update skripte Gunner_health

Na početku Update metode postavljamo vrijednost *Slidera* da odgovara trenutnom postotku životnih bodova. Nakon toga vrši se provjera iznosa životnih bodova te, ukoliko su oni manji od 0, dolazi do uklanja glavnog lika iz igre te se prikazuje gumb za ponovni početak. Ukoliko nije došlo do uklanjanja glavnog lika, provjeriti će se je li došlo do kolizije s neprijateljskim projektilom preko metode *CheckEnemyAmmocollision* te će se oduzeti životni bodovi prema vrijednosti zadanu u samom Unityu. Uz oduzimanje životnih bodova preko projektila, moguće je i oduzimanje životnih bodova samom kolizijom s neprijateljima, ali to je implementirano u posebnoj skripti za neprijatelje koja će se kasnije spomenuti.

```
bool CheckEnemyAmmoCollision()
{
    SpriteRenderer mySR;
    mySR = gameObject.GetComponent<SpriteRenderer>();
    foreach (GameObject ammoObj in GameObject.FindGameObjectsWithTag("enemyAmmo"))
    {
        SpriteRenderer ammoSR = ammoObj.GetComponent<SpriteRenderer>();
        if (ammoSR.bounds.Intersects(mySR.bounds))
        {
            Destroy(ammoObj);
            return true;
        }
    }
    return false;
}
```

Slika 29 Metoda CheckEnemyAmmoCollision

Nakon ovoga, provjerava se je li došlo do *ispadanja* glavnog lika iz okvira igre te, ako je došlo do toga, životni bodovi se postavljaju na vrijednost 0. Posljednji bitan dio je onaj za dodavanje životnih bodova u slučaju da glavni lik *sakupi* srce. Tu funckiju izvršava metoda *AddHealth*.

Slika 30 Srce za dodatne životne bodove

```
public void AddHealth()
{
    SpriteRenderer mySR;
    mySR = gameObject.GetComponent<SpriteRenderer>();
    foreach (GameObject heartObj in GameObject.FindGameObjectsWithTag("heart"))
    {
        SpriteRenderer heartSR = heartObj.GetComponent<SpriteRenderer>();
        if (heartSR.bounds.Intersects(mySR.bounds))
        {
            health+=healthIncrease;
            if(health > maxHealth)
            {
                health = maxHealth;
            }
            Destroy(heartObj);
        }
    }
}
```

Slika 31 Metoda AddHealth

Metoda *AddHealth* provjerava je li došlo do kolizije s ikonom srca te, ukoliko se to i dogodilo, povećava životne bodove za unaprijed definiranu vrijednost te uklanja tu ikonu srca iz igre.

Posljednji dio koji je trebalo posložiti vezano uz glavnog lika je sakupljanje zlatnika.

Slika 32 Zlatnik

Metoda koja kontrolira sakupljanje novčića stvorena je u skripti *Gunner_controller*, a funkcionira na isti način kao metoda za sakupljanje srca za životne bodove.

```
void CollectCoins()
{
    SpriteRenderer mySR;
    mySR = gameObject.GetComponent<SpriteRenderer>();
    foreach (GameObject coinObj in GameObject.FindGameObjectsWithTag("coin"))
    {
        SpriteRenderer coinSR = coinObj.GetComponent<SpriteRenderer>();
        if (coinSR.bounds.Intersects(mySR.bounds))
        {
            coinCount++;
            Destroy(coinObj);
        }
    }
}
```

Slika 33 Metoda CollectCoins

Kao što je prikazano na slici 13 na kojoj se vidi metoda *Update* iz skripte *Gunner_controller* ako su sakupljeni svi zlatnici i eliminirani svi neprijatelji dolazi do pauziranja igre te prikazivanja gumba za ponovno pokretanje igre.

3.3.3. Dodavanje neprijatelja i implementacija ponašanja neprijatelja

Nakon što je implementirano sve potrebno za glavnog lika, potrebno je implementirati ponašanje neprijatelja. Za potrebe toga stvorena je skirpta *EnemyController*.

```
void Update()
{
    if (CheckCollisionAmmo())
    {
        Destroy(gameObject);
    }
    Move();
    Attack();
}
```

Slika 34 Update metoda skripte EnemyController

U *Update* metodi pozivaju se metoda *CheckCollisionAmmo* koja provjerava je li došlo do kolizije neprijatelja i projektila glavnog lika, metoda *Move* koja omogućuje kretanje između dvije X koordinate, te metoda *Attack* koja omogućava oduzimanje životnih bodova igraču u slučaju da dođe do kontakta.

```
bool CheckCollisionAmmo()
{
    SpriteRenderer mySR;
    mySR = gameObject.GetComponent<SpriteRenderer>();
    foreach (GameObject ammoObj in GameObject.FindGameObjectsWithTag("ammo"))
    {
        SpriteRenderer ammoSR = ammoObj.GetComponent<SpriteRenderer>();
        if (ammoSR.bounds.Intersects(mySR.bounds))
        {
            Destroy(ammoObj);
            return true;
        }
    }
    return false;
}
```

Slika 35 Metoda CheckCollisionAmmo

```
void Move()
{
    if (directionLeft)
    {
        if(transform.position.x > minX)
        {
            enemy.velocity = new Vector2(-speed, enemy.velocity.y);
        }
        else
        {
            Flip();
        }
    }
    else
    {
        if (transform.position.x < maxX)
        {
            enemy.velocity = new Vector2(speed, enemy.velocity.y);
        }
        else
        {
            Flip();
        }
    }
}
```

Slika 36 Metoda Move

Metoda *Flip* koja se koristi unutar metode *Move* implementirana je na isti način kao i prijašnje *Flip* metode.

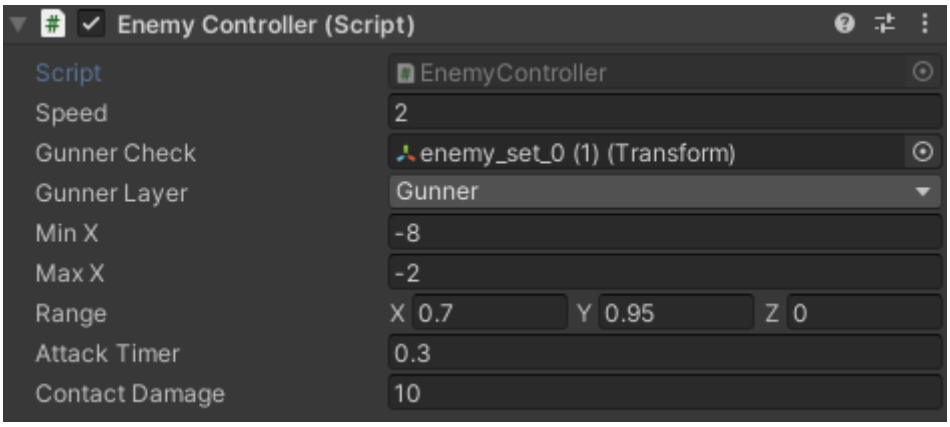
```
void Attack()
{
    Collider2D attackGunner = Physics2D.OverlapBox(gunnerCheck.position, range, 0, gunnerLayer);
    if(attackGunner != null)
    {
        sinceLastAttack -= Time.deltaTime;
        if (sinceLastAttack<0)
        {
            foreach(GameObject gunner in gunners)
            {
                gunner.GetComponent<Gunner_health>().LoseHealth(contactDamage);
            }
            sinceLastAttack = attackTimer;
        }

    }
}
```

Slika 37 Metoda Attack

Unutar *Attack* metoda poziva se metoda *LoseHealth* iz skripte *Gunner_health*.

```
public void LoseHealth(int damage)
{
    health -= damage;
}
```

Slika 38 Metoda LoseHealth



Slika 39 Skripta EnemyController u Unityu

Skripta *EnemyController* zajednička je svim neprijateljima, a za potrebe letećih i naprednih zemljanih neprijatelja stvorena je dodatna skripta pod nazivom *FlyingEnemyShootingAttack*.

```
void Update()
{
    Attack();
}
```

Slika 40 Update metoda skripte FlyingEnemyShootingAttack

Kao što vidimo, jedino što se događa u metodi *Update* je pozivanje metode *Attack*.

```
void Attack()
{
    bool canAttack = false;
    Vector3 currentPos = new Vector3();
    currentPos = transform.position;
    SpriteRenderer gunner = FindGunner();
    if (gunner != null)
    {
        Vector3 gunnerPosition = gunner.transform.position;
        float xDiff = gunnerPosition.x - currentPos.x;
        float yDiff = gunnerPosition.y - currentPos.y;
        if (xDiff <= range && yDiff <= range && xDiff >= -range && yDiff >= -range) canAttack = true;
        else canAttack = false;
        if (canAttack)
        {
            shootAngle = (float)Mathf.Atan2(yDiff, xDiff);
            shootAngle *= 180 / (float)Mathf.PI;
            fireTimer -= Time.deltaTime;
            if (fireTimer < 0)
            {
                ShootBullet();
                fireTimer = attackTime;
            }
        }
    }

}
```

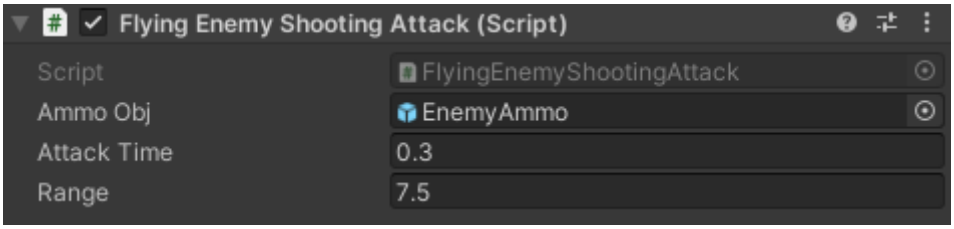
Slika 41 Metoda Attack

Metoda *Attack* najprije dohvaća trenutnu poziciju neprijatelja, a nakon toga i trenutnu poziciju glavnog lika kao bi se mogao izračunati kut podkojim će neprijatelj ispaliti projektil. Ispaljivanje projektila događa se uz vremensku odgodu koju je moguće regulirati u Unityu preko varijable *attackTime*. Posljednja metoda koja se poziva je metoda *ShootBullet* za ispaljivanje projektila.

```
void ShootBullet()
{
    GameObject newBullet = Instantiate(ammoObj);
    newBullet.GetComponent<Bullet_controller>().FireShot(shootAngle, transform.position);
}
```

Slika 42 Metoda ShootBullet

Slika 43 Postavke skripte za leteće neprijatelje



Slika 44 Postavke skripte za napredne zemljane neprijatelje

Kao što vidimo na prethodne dvije slike, napredni zemljani neprijatelji imaju kraću pauzu u ispaljivanju projektila, a istovremeno i veći radijus napada.

Kao i u skripti *Gunner_attack* i ovdje je korištena skripta *Bullet_controller* koja upravlja ispaljivanjem i letom projektila. Prije nego pogledamo *Update* metodu, pregledat ćemo metodu *FireShot* koja je već prije korištena u ovom radu.

```
public void FireShot(float angle, Vector3 position)
{
    shotAngle = angle * Mathf.Deg2Rad;
    transform.position = position;
    transform.Rotate(0, 0, angle, Space.Self);
}
```

Slika 45 Metoda FireShot

Ova metoda određuje kut pod kojim će se kretati projektil, nakon toga određuje početnu poziciju, a na kraju rotira projektil kako bi bio okrenut u smjeru kretanja.

```
void Update()
{
    Vector3 newPosition = new Vector3();
    newPosition = transform.position;
    newPosition.x += Mathf.Cos(shotAngle) * speed * Time.deltaTime;
    newPosition.y += Mathf.Sin(shotAngle) * speed * Time.deltaTime;

    transform.position = newPosition;

    CheckColisionWithPlatform();
}
```

Slika 46 Update metoda skripte Bullet_controller

U *Update* metodi mijenja se pozicija projektila i poziva se metoda *CheckColisionWithPlatform*.

```
public void CheckColisionWithPlatform()
{
    SpriteRenderer mySR;
    mySR = gameObject.GetComponent<SpriteRenderer>();
    foreach (GameObject platformObj in GameObject.FindGameObjectsWithTag("Platform")) {
        SpriteRenderer platformSR = platformObj.GetComponent<SpriteRenderer>();
        if (platformSR.bounds.Intersects(mySR.bounds))
        {
            Destroy(gameObject);
        }
    }
}
```

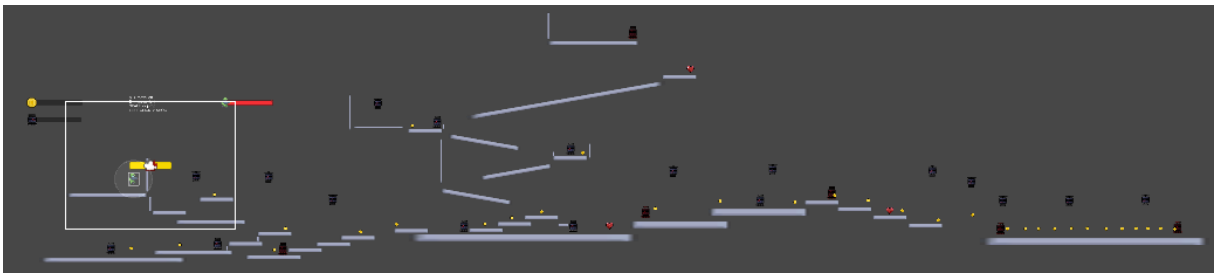
Slika 47 Metoda CheckColisionWithPlatform

U ovoj se metodi provjerava je li došlo do kolizije projektila s platformom. Ukoliko je došlo do kolizije, projektil se uništava.

Za potrebe animacija neprijatelja stvorene su animacije za kretanje i to na isti način kao što je stvorena animacija za kretanje glavnog lika.

3.3.4. Kreiranje mape videoigre

Za kreiranje mape videoigre korištena je ranije spomenuta platforma. Jedna platforma je korištena u različitim veličinama i rotirana pod raznim kutevima kako bi se stvorio put od početka do kraja videoigre.

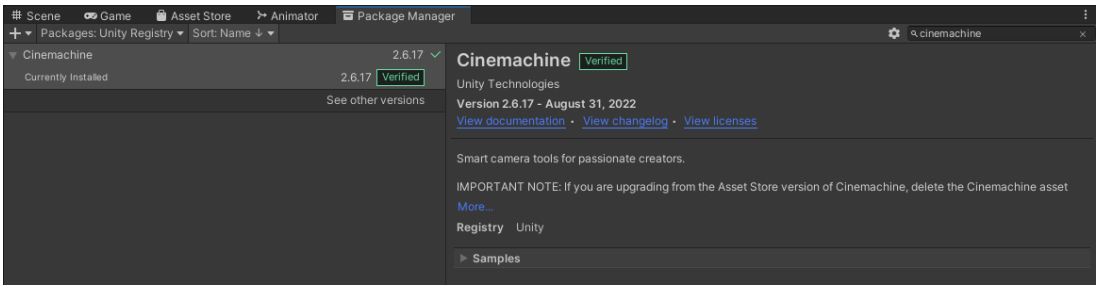


Slika 48 Cijela mapa videoigre

Ukupno je korišteno 44 različite platforme. Također, na mapu su dodani neprijatelji, ukupno njih 27 od čega je 11 letećih neprijatelja, 10 običnih zemljanih i 6 naprednih zemljanih neprijatelja te ih je potrebno sve eliminirati kako bi igraču bila omogućena pobjeda. Dodani su i već ranije spomenuti zlatnici koje je potrebno sakupiti sve kako bi bila moguća pobjeda. Posljednje što je dodano su srca za povećanje životnih bodova.

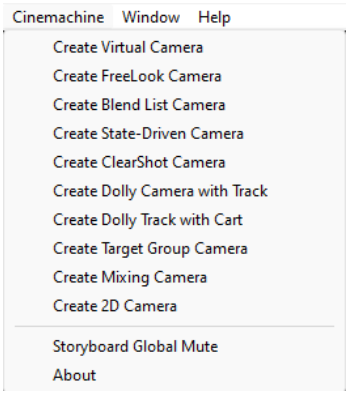
3.3.5. Postavljanje kamere

Kako bi bilo moguće postavljanje kamere koja će pratiti kretanje glavnog lika, prvo je bilo potrebno instalirati paket *Cinemachine* iz *Package Managera* unutar Unity-a.



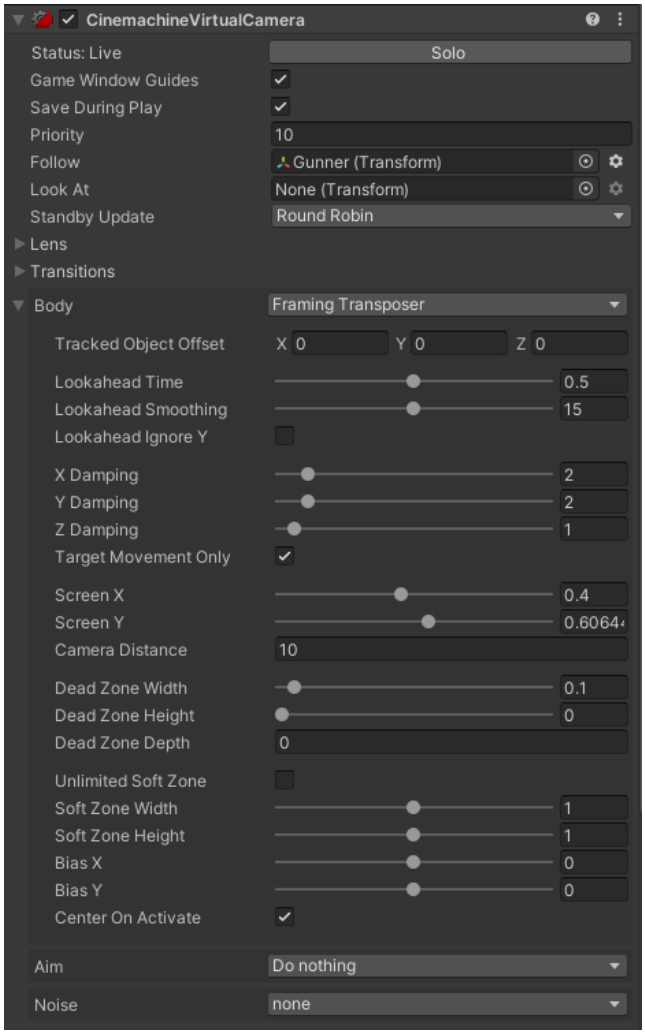
Slika 49 Dodavanje paketa Cinemachine

Nakon što je paket dodan, na vrhu alatne trake pojavila se kartica pod nazivom *Cinemachine* sa sadržajem koji je prikazan na sljedećoj slici.



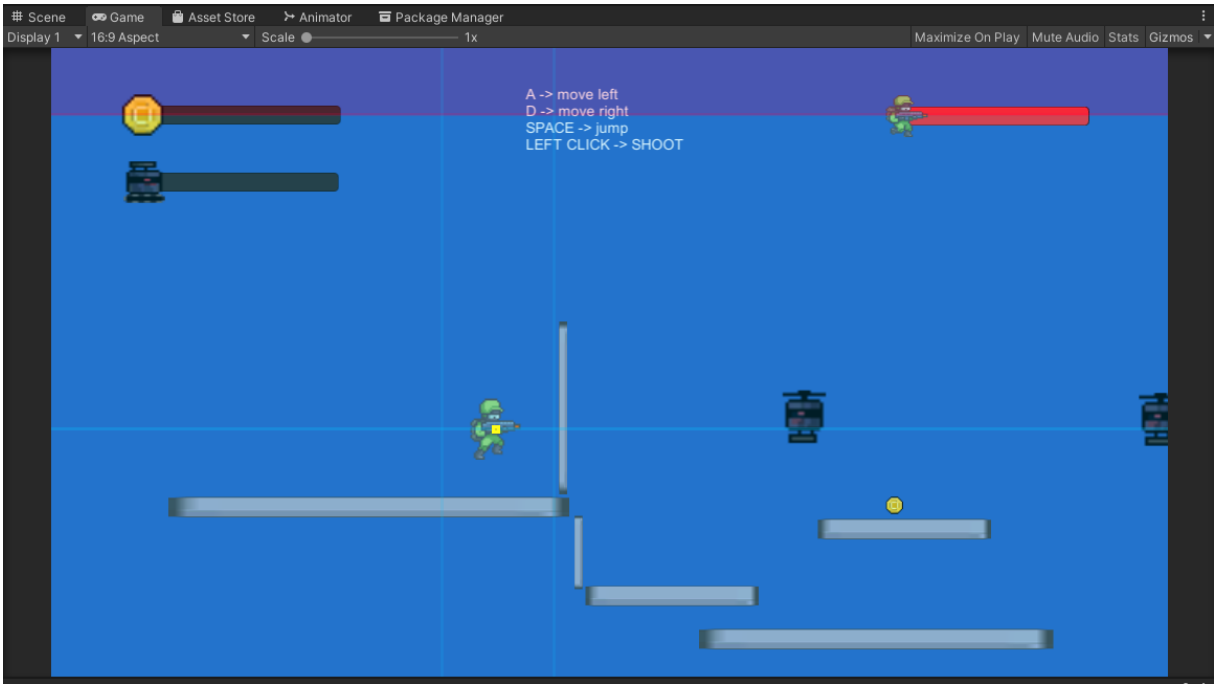
Slika 50 Kartica Cinemachine

Za potrebe ovog projekta odabrana je opcija *Create 2D Camera*. Nakon toga automatski je dodana kamera u videoigru. S obzirom da vrijednosti početnih postavki nisu u potpunosti odgovarale potrebama ove videoigre, postavke su na kraju postavljene na vrijednosti sa sljedeće slike.



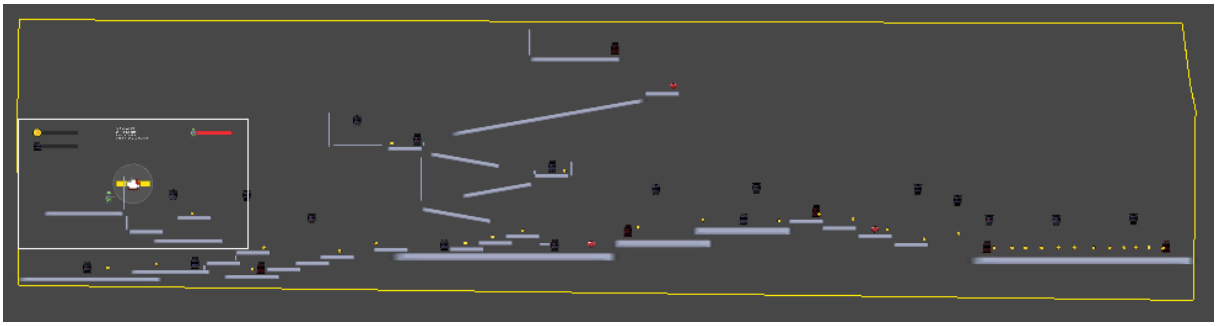
Slika 51 Postavke kamere

Ono što je najvažnije napomenuti je to da je za vrijednost varijable *Follow* označen glavni lik te je samim time omogućeno da kamera prati glavnog lika. Varijablu koju bi također trebalo istaknuti je i *Dead Zone Width* koja određuje koliko je dopušteno da se glavnik lik pomakne lijevo ili desno sa sredine ekrana prije nego ga kamera počne slijediti. Na isti način funkcionira i varijabla *Dead Zone Height* koja regulira kretanje kamere s obzirom na podizanje i spuštanje igrača u odnosu na centar ekrana. Prilikom promjene varijabli, u kartici *Game* će se prikazati pomoćne osi s kojima će se lakše regulirati iznosi pojedinih varijabli kao što je prikazano na sljedećoj slici.



Slika 52 Pomoćne osi

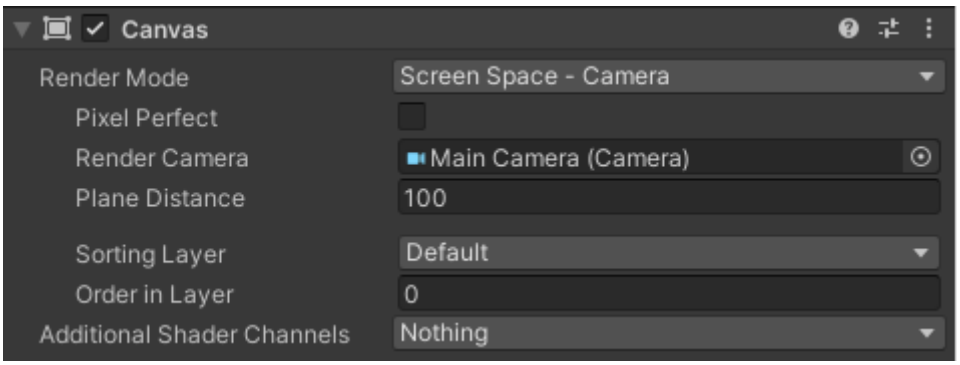
Posljednje što je bilo dodano vezano uz kameru je okvir koji ograničava kameru da napusti područje mape videoigre.



Slika 53 Okvir za kretanje kamere

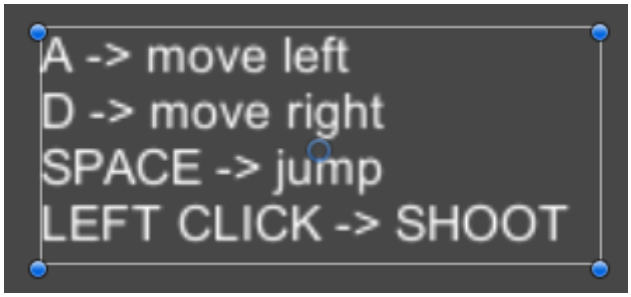
3.3.6. Dodavanje Canvasa

Ono što je posljednje dodano u ovu videoigru su sučelja za prikaz životnih bodova, eliminiranih neprijatelja i sakupljenih zlatnike te gumb za ponovno pokretanje. Kako bi ti elementi uopće mogli biti dodani, mora postojati *Canvas* unutar kojeg ćemo sve te elemente smjestiti. *Canvas* je moguće *prikačiti* za kameru tako da uvijek bude prikazan na ekranu igrača.



Slika 54 Postavke Canvasa

Nakon što je stvoren prazni Canvas, u njega možemo dodati elemente koje želimo da budu prikazani igraču. Prvo će, kao pomoć igraču u igri, biti dodan tekst u kojem će biti popis kontrola.



Slika 55 Kontrole

Nakon što su dodane kontrole, biti će dodano sučelje za prikaz životnih bodova.

Slika 56 Sučelje za prikaz životnih bodova

Ovim sučeljem se upravlja iz skripte Gunner_health. U toj skripti se najprije u metodi *Awake* odredi koliki je maksimalni broj životnih bodova kako bi se u svakom trenutku mogao izračunati trenutni postotak životnih bodova, te se vrijednost *Slidera* postavlja na 1.

```
void Awake()
{
    maxHealth = health;
    slider.value = 1;
}
```

Slika 57 Metoda Awake skripte Gunner_health

Nakon što su spomenute vrijednosti postavljene, u *Update* metodi se vrijednost *Slidera* postavlja na trenutni postotak ukupnih životnih bodova.

```
healthPercentage = health/maxHealth;

slider.value = healthPercentage;
```

Slika 58 Postavljanje vrijednosti slidera

Ovime se omogućava da prikaz životnih bodova bude uvijek usklađen s ukupnim životnim bodovima.

Još jedna situacija na koju je trebalo obratiti pozornost je slučaj u kojem bi sakupljanjem srca za životne bodove došlo do toga da je trenutno stanje životnih bodova veće nego maksimalno stanje životnih bodova, te je stoga postavljena dodatna provjera, u već ranije spomenutoj metodi *AddHealth*, u kojoj se trenutno stanje postavlja na maksimalno ukoliko dođe do prekoračenja broja trenutnih životnih bodova.

```
if(health > maxHealth)
{
    health = maxHealth;
}
```

Slika 59 Provjera životnih bodova

Nakon što je implementirano prikazivanje životnih bodova, implementirano je i prikazivanje prikupljenih zlatnika te prikazivanje eliminiranih neprijatelja.

Slika 60 Sučelje za prikaz sakupljenih zlatnika

Slika 61 Sučelje za prikaz eliminiranih neprijatelja

I ova sučelja implementirana su uz pomoć *Slidera* kako bi se prikazao postatak sakupljenih novčića, odnosno eliminiranih neprijatelja. Upravljanje ovim *Sliderima* je implementirano u skripti *Gunner_controller*.

Za početak, u *Awake* metodi se postavlja vrijednost oba *Slidera* na 0, a uz to se prebrojavaju zlatnici i neprijatelji kako bi se mogao izračunati postotak.

```
coinCount = 0;
maxCoins = GameObject.FindGameObjectsWithTag("coin").Length;
sliderCoins.value = 0;

enemiesTotal = GameObject.FindGameObjectsWithTag("groundEnemy").Length;
enemiesTotal += GameObject.FindGameObjectsWithTag("flyingEnemy").Length;
enemiesRemaining = enemiesTotal;
enemiesEliminated = 0;
sliderEnemies.value = 0;
```

Slika 62 Početno postavljanje broja zlatnika i neprijatelja

Nakon toga, u *Update* metodi prvo se dohvaća broj preostalih neprijatelja, a nakon toga se provjerava je li došlo do sakupljanja nekog od dostupnih novčića. Nakon tih provjera, postavljaju se *Slideri* za zlatnike i eliminirane neprijatelje na potrebne vrijednosti.

```
CountRemainingEnemies();
CollectCoins();
coinPercentage = coinCount / maxCoins;
enemiesEliminated = enemiesTotal - enemiesRemaining;
enemiesPercentage = enemiesEliminated / enemiesTotal;
sliderCoins.value = coinPercentage;
sliderEnemies.value = enemiesPercentage;
```

Slika 63 Upravljanje Sliderima u Update metodi

U metodi *CountRemainingEnemies* prebrojavaju se neprijatelji na isti način na koji su se prebrojavali u metodi *Awake*.

```
void CountRemainingEnemies()
{
    enemiesRemaining = GameObject.FindGameObjectsWithTag("groundEnemy").Length;
    enemiesRemaining += GameObject.FindGameObjectsWithTag("flyingEnemy").Length;
}
```

Slika 64 Metoda CountRemainingEnemies

Metoda *CollectCoins* provjerava je li došlo do kolizije s nekim zlatnikom te, ukoliko je došlo do toga, povećava broj sakupljenih novčića te uništava novčić koji je sakupljen.

```
void CollectCoins()
{
    SpriteRenderer mySR;
    mySR = gameObject.GetComponent<SpriteRenderer>();
    foreach (GameObject coinObj in GameObject.FindGameObjectsWithTag("coin"))
    {
        SpriteRenderer coinSR = coinObj.GetComponent<SpriteRenderer>();
        if (coinSR.bounds.Intersects(mySR.bounds))
        {
            coinCount++;
            Destroy(coinObj);
        }
    }
}
```

Slika 65 Metoda CollectCoins

Posljednje što je implementirano unutar *Canvasa* je gumb za ponovni početak. Gumb je na početku nevidljiv te se otkriva u slučaju pobjede ili u slučaju eliminacije glavnog lika.

Slika 66 Gumb za ponovni početak

Ukoliko dođe do spomenutih događaja, gumb će se tada prikazati uz pomoć koda na sljedećim slikama. Dio vezan uz sakupljanje novčića i eliminiranje neprijatelja implementiran je u skripti *Gunner_controller* dok je dio vezan uz gubljenje životnih bodova implementiran u skripti *Gunner_health*.

```
if (coinPercentage == 1 && enemiesPercentage == 1)
{
    Time.timeScale = 0;
    foreach (GameObject restartBtn in GameObject.FindGameObjectsWithTag("restart"))
    {
        Vector3 newBtnPos = transform.position;
        newBtnPos.z = 100;
        newBtnPos.y = 0;
        newBtnPos.x = lastGunnerPosX;
        restartBtn.transform.position = newBtnPos;
    }
}
```

Slika 67 Provjera uvjeta za pobjedu

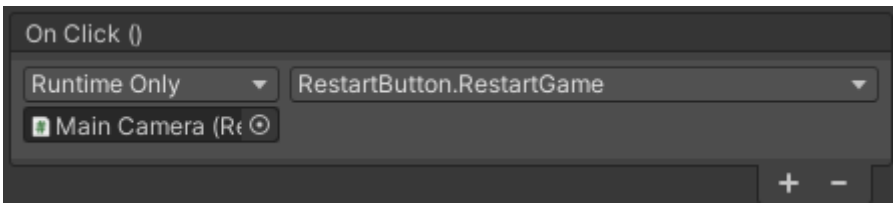
```
if (health <= 0)
{
    Destroy(gameObject);
    Time.timeScale = 0;
    foreach (GameObject restartBtn in GameObject.FindGameObjectsWithTag("restart"))
    {
        Vector3 newBtnPos = transform.position;
        newBtnPos.z = 100;
        newBtnPos.y = 0;
        newBtnPos.x = lastGunnerPosX;
        restartBtn.transform.position = newBtnPos;

    }

}
```

Slika 68 Provjera uvjeta životnih bodova

Kao što možemo vidjeti na prikazanim slikama, u oba slučaja igra se pauzira postavljanjem varijable *Time.timeScale* na vrijednost 0 te se nakon toga prikazuje gumb za ponovno pokretanje. Y i Z koordinata gumba će biti fiksne dok X koordinata ovisi o posljednjoj poziciji glavnog lika.

Slika 69 Prikaz gumba za ponovno pokretanje u slučaju pobjede

Ono što je također postavljeno za ovaj gumb je poziv metode *RestartGame* iz skripte *RestartButton*. Ovaj poziv će se dogoditi na klik gumba, a to je omogućeno uz već ugrađenu mogućnost gumba prikazanu na idućoj slici.



Slika 70 OnClick() mogućnost gumba

Nakon što je gumb kliknut, varijabla *Time.timeScale* biti će postavljena na vrijednost 1 te će se videoigra ponovno pokrenuti.

```
public void RestartGame()
{
    Time.timeScale = 1f;
    SceneManager.LoadScene(SceneManager.GetActiveScene().buildIndex);
}
```

Slika 71 Metoda RestartGame

4. Zaključak

Ovaj rad rezultirao je izradom jednostavne funkcionalne 2D akcijske videoigre koja implementira svažnije aspekte akcijske videoigre. Za potrebe ovog rada korišten je programski alat Unity uz pomoć još nekoliko programa koji su pomogli pri uređivanju slika (GIMP) i koda (Visual Studio, C# programski jezik), a za preuzimanje materijala korištena je stranica itch.io. Kod glavnog lika omogućeno je kretanje, napadanje te sakupljanje zlatnika, dok je neprijateljima omogućeno kretanje između dvije koordinate X osi te ispaljivanje projektila u smjeru glavnog lika. Na kraju je prikazano na koji način se koristi Canvas te što sve Canvas može sadržavati te na koji način upravljati prikazom nekih informacija unutar Canvasa.

Igra je testirana na više različitih računala te je utvrđeno da je funkcionalna na svakom od njih te je spremna za daljnje moguće nadogradnje.

Popis literature

[1] Microsoft, *Unity*, Dostupno: https://dotnet.microsoft.com/en-us/apps/games/unity [pristupano 02.08.2023.]

[2] Unity, *Installing the Unity Hub*, Dostupno: https://docs.unity3d.com/2020.1/Documentation/Manual/GettingStartedInstallingHub.html [pristupano 02.08.2023.]

[3] Microsoft, *What is Visual Studio*, Dostupno: https://learn.microsoft.com/en-us/visualstudio/get-started/visual-studio-ide?view=vs-2022 [pristupano 02.08.2023.]

[4] GIMP, *About GIMP*, Dostupno: https://www.gimp.org/about/introduction.html [pristupano 02.08.2023.]

[5] itch.io, *About itch.io*, Dostupno: https://itch.io/docs/general/about [pristupano 02.08.2023.]

[6] Microsoft, *A tour of the C# language*, Dostupno: https://learn.microsoft.com/en-us/dotnet/csharp/tour-of-csharp/ [pristupano 02.08.2023.]

[7] ComputerHope, *Action game*, Dostupno: https://www.computerhope.com/jargon/a/action-game.htm [pristupano [pristupano 03.08.2023.]

Popis slika