2D Platformer

Duvnjak, Predrag

Undergraduate thesis / Završni rad

2019

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **Josip Juraj Strossmayer University of Osijek, Faculty of Electrical Engineering, Computer Science and Information Technology Osijek / Sveučilište Josipa Jurja Strossmayera u Osijeku, Fakultet elektrotehnike, računarstva i informacijskih tehnologija Osijek**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:200:377971

Rights / Prava: In copyright / Zaštićeno autorskim pravom.

Download date / Datum preuzimanja: **2024-04-19**

Repository / Repozitorij:

Faculty of Electrical Engineering, Computer Science and Information Technology Osijek

SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU

FAKULTET ELEKTROTEHNIKE,RAČUNARSTVA I

INFORMACIJSKIH TEHNOLOGIJA

Sveučilišni studij

2D Platformer

Završni rad

Predrag Duvnjak

Osijek,2019.

SADRŽAJ

1. UVOD 1

1.1. Zadatak završnog rada 1

2. KORIŠTENI ALATI 2

2.1. Unity 2

2.2. MonoDevelop 3

2.3. Aseprite 4

3. RAZVOJ IGRE 5

3.1. Likovi 5

3.1.1. Prase 5

3.1.2. Mesar 10

3.2. Prepreke 15

3.2.1. Toranj 15

3.2.2. Šiljci 16

3.2.3. Ljepljiva sluz 17

3.3. Platforme 19

3.4. Zvukovi i glazba 21

3.5. Izrada razina 24

3.6. Animacije 26

3.7. Kamera 28

3.8. Izbornici i upravitelj igrom 29

3.9. Igračevo sučelje 35

4. ZAKLJUČAK 36

LITERATURA 37

SAŽETAK 39

ABSTRACT 40

ŽIVOTOPIS 41

1. UVOD

Tema završnoga rada je izrada 2D platformerske igre. Platformer [1] tip igara odnosi se na igre u kojima lik kojega kontrolira igrač mora skakati između platformi, uz izbjegavanje zapreka. U okolišu se često nalazi neravan teren kojega igrač mora prijeći. Igrač najčešće ima neku kontrolu nad visinom i udaljenošću skoka kako bi omogućio liku izbjegavanje smrti. Najpoznatija karakteristika ovakvog oblika igara je gumb za skakanje, uz to da postoje alternative ako se radi o drugim okolnostima kao što mogu biti ekrani na dodir. Rad koristi Unity game engine kao podršku za razvoj igre, te za pisanje skripti korišten je objektno orijentirani programski jezik C#. Tematika igre odnosi se na prase koje pokušava pobjeći iz neobične klaonice u kojoj se je našlo. Kako bi uspjelo u tome, igrač mora zaobići zamke postavljene po klaonici, te se boriti s neprijateljima čiji je cilj zaustaviti prase pod svaku cijenu.

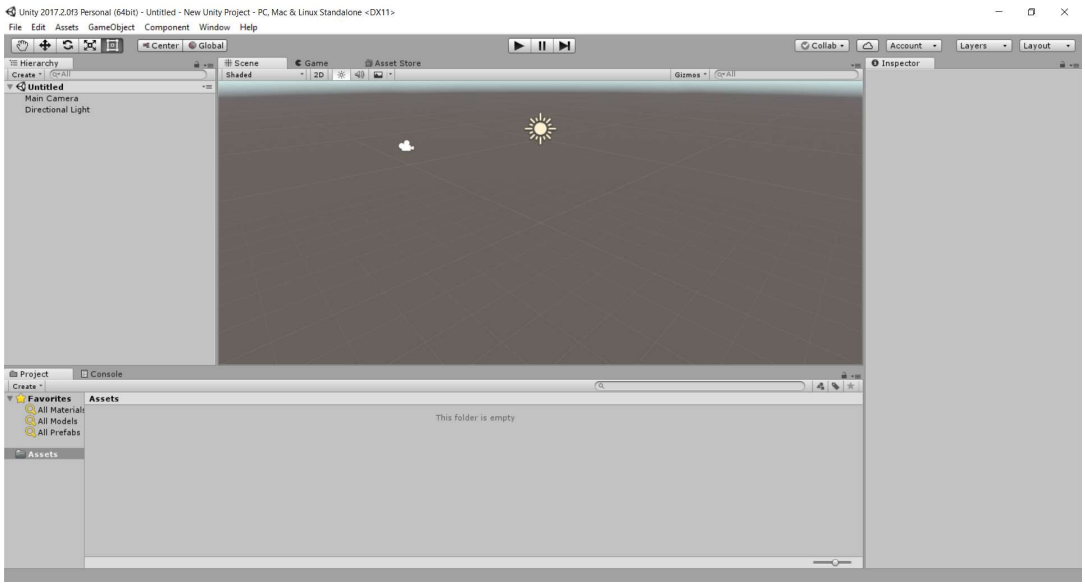
1.1. Zadatak završnog rada

Zadatak rada odnosi se na izradu 2D Platformer igre u Unity-u , uz pomoć objektno orijentiranog jezika C#.

2. KORIŠTENI ALATI

2.1.Unity

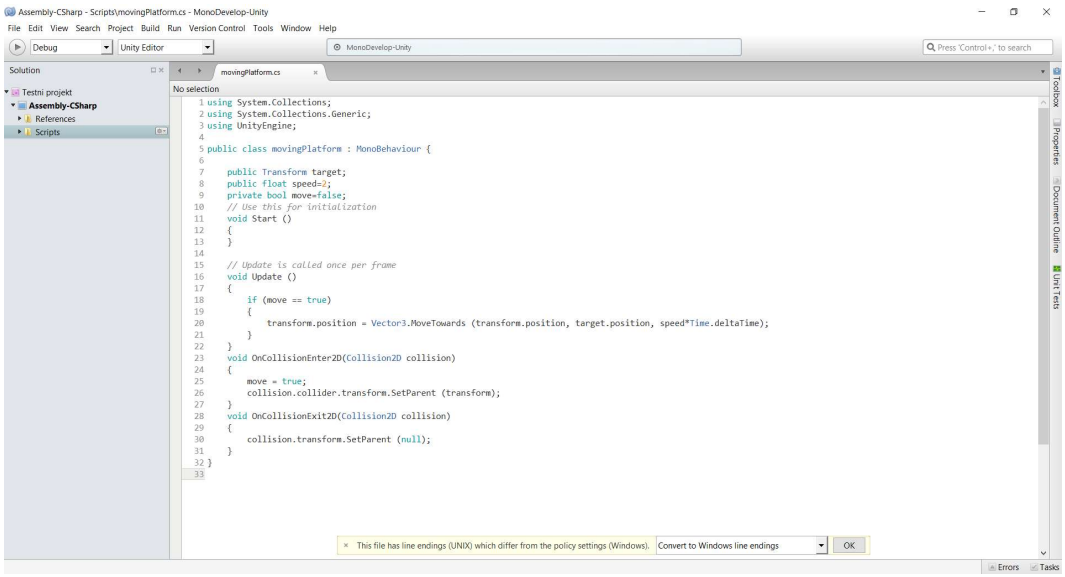
Unity [2] je višeplatformski game engine razvijen od strane Unity Technologies kompanije. Prvo izdanje Unity-a objavljeno je 2005, te mu je originalna namjena bila služiti kao ekskluzivni game engine za Mac OS X. Od 2018, broj platformi koje Unity podržava veći je od 25 što znatno pridonosi razvoju igara. Unity može biti korišten za razvoj različitih tipova igara, kao što su dvodimenzionalne i trodimenzionalne igre, no također može biti korišten i van industrije gdje kao primjer možemo uzeti arhitekturu. Pri stvaranju skripti koje tvore funkcionalnosti igre, Unity koristi objektno orijentirani jezik C#.



Sl. 2.1. Sučelje Unity-a

2.2.MonoDevelop

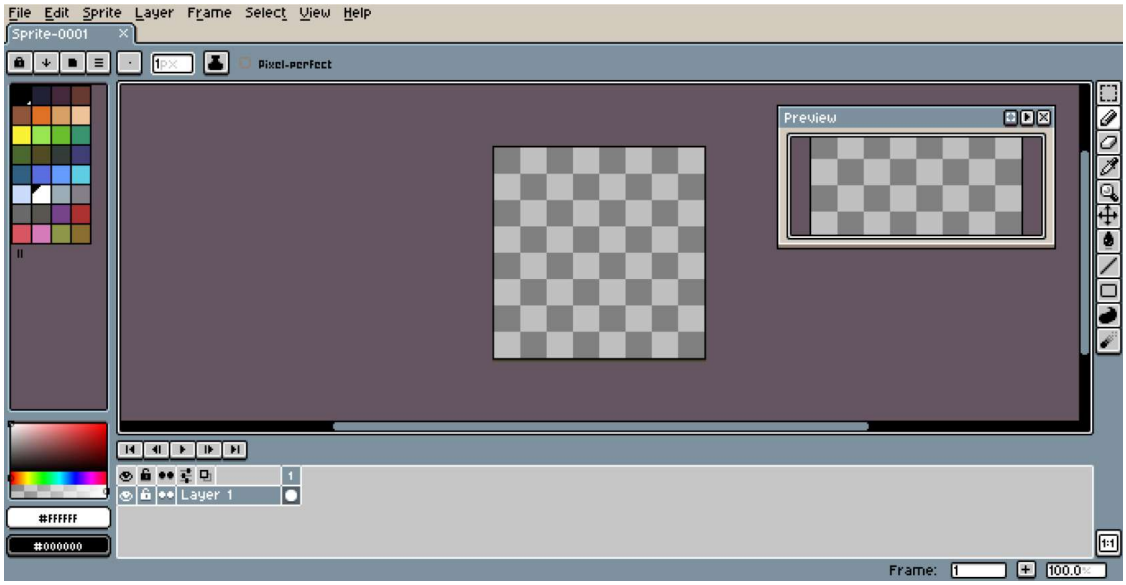
MonoDevelop [3] je open source razvojno okruženje za Linux, macOS i Windows operacijske sustave. MonoDevelop pruža značajke slične onima Microsoft Visual Studia, kao što su automatsko dovršavanje koda, korisničko sučelje te web dizajner. Podržava jezike : Boo, C, C++, C#, CIL, D, F#, Java, Oxygene, Vala, JavaScript, TypeScript i Visual Basic.NET. Od verzije 2018.1 pa nadalje, Unity je prekinuo podršku za MonoDevelop te koristi Visual Studio.



Sl. 2.2. Sučelje MonoDevelop-a

2.3.Aseprite

Aseprite [4] je popularni alat koji služi za stvaranja pixel arta. Jedna od najvećih prednosti Aseprite-a je njegova jednostavnost korištenja , koju omogućuje intuitivno korisničko sučelje. Uz crtanje moguće je i animiranje likova stvaranjem i izmjenom frameova korak po korak, te exportanje u obliku sprite sheeta što omogućuje njihovu laku implementaciju u Unity-u .Također, Aseprite nudi opciju stvaranja i učitavanja paleta boja, koje služe kako bi se dočarao distinktan izgled likovima korištenjem boja koje idu zajedno.



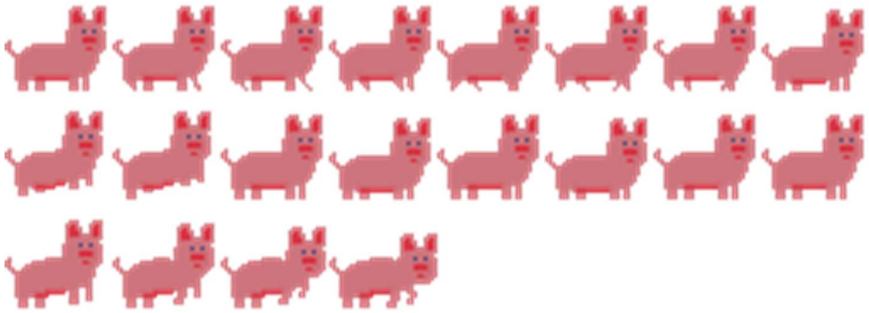
Sl. 2.3. Sučelje Aseprite-a

3. RAZVOJ IGRE

3.1.Likovi

3.1.1. Prase

Prase u ovoj igri predstavlja protagonista čiji je glavni cilj bijeg iz klaonice pod svaku cijenu. Kako bi ostvarilo svoj cilj, mora se suočiti sa zamkama postavljenima unutar klaonice te neprijateljima koji ga žele zaustaviti.



Sl. 3.1. Sprite sheet glavnoga lika

```
void Update()
{
    if (Input.GetKey (KeyCode.D))
    {
        rigidbody.velocity = new Vector2 (pigSpeed * Time.deltaTime, rigidbody.velocity.y);
    }
    else if (Input.GetKey (KeyCode.A))
    {
        rigidbody.velocity = new Vector2 (-1 * pigSpeed * Time.deltaTime, rigidbody.velocity.y);
    }
    else
    {
        rigidbody.velocity = new Vector2 (0, rigidbody.velocity.y);
    }
    float x_movement = rigidbody.velocity.x;
    animator.SetFloat ("horizontal_movement", Mathf.Abs (x_movement));
    if (x_movement > 0 && !moveRight || x_movement < 0 && moveRight)
        ChangeDirection ();

    if (Input.GetKeyDown (KeyCode.F))
    {
        Heal ();
    }
```

Programski kod 3.1. Funkcionalnosti igrača

Za horizontalno kretanje, unutar *if* i *else if* uvjeta provjerava se jesu li pritisnute tipke za kretanje lijevo i desno, te ako jesu mijenja se brzina Rigidbody2D [5] komponente prikazane pomoću rigidbody reference. Brzina kretanja koja je postavljena unutar rigidbody reference jednaka je umnošku protoka vremena i pigSpeed varijable koja se odnosi na brzinu praseta. Uvjet *else* zaustavlja igračevo horizontalno kretanje ako tipke nisu pritisnute. Unutar varijable x_movement spremljena je brzina kretanja po x osi koja se prosljeđuje animatoru radi korištenja pri tranziciji stanja, te se koristi kod *if* uvjeta gdje se zajedno sa varijablom moveRight koja sadrži smjer prema kojemu je igrač okrenut određuje je li potrebno okrenuti igrača pozivanjem metode ChangeDirection(). Također, postavljen je i *if* uvjet koji služi za očitavanje pritiska tipke određene za pozivanje metode Heal() koja vraća zdravlje igraču.

```
if (Input.GetKeyDown (KeyCode.Space))
{
    if (grounded)
    {
        animator.SetBool ("Ground", false);
        animator.SetBool ("Jump", true);
        rigidbody.AddForce (new Vector2 (0, jumpStrength));
        doubleJump = true;
        audioManager.Play ("jump");
    }
    else
    {
        if (doubleJump)
        {
            doubleJump = false;
            rigidbody.velocity = new Vector2 (0, 0);
            rigidbody.AddForce (new Vector2 (0, jumpStrength));
            animator.SetBool ("Jump", true);
            audioManager.Play ("jump");
        }
    }
}
animator.SetFloat ("vertical_speed", rigidbody.velocity.y);
checkGround ();
```

Programski kod 3.2. Funkcionalnosti igrača

Kao što je vidljivo iz programskog koda 3.2., nakon pritiska na tipku dodijeljenu za skok potrebno je provjeriti nalazi li se igrač na tlu što je predstavljeno *bool* varijablom grounded. Ukoliko je grounded u istinitom stanju, parametar animatora Ground se postavlja u neistinito stanje a Jump se postavlja u istinito. Komponenti Rigidbody2D se dodaje sila za odskakanje predstavljena kao Vector2 koja sadrži jumpStrength vrijednost koja se odnosi na jačinu skoka. Također doubleJump *bool* varijabla postavlja se u istinito stanje kako bi igrač mogao skočiti još jednom u zraku, te se preko audio upravitelja pušta zvuk skakanja. Ukoliko je grounded varijabla bila neistinita, unutar *if* uvjeta provjerava se doubleJump varijabla koja određuje može li igrač ponovno skočiti. Ako je

istinita procedura je ista kao i za običan skok, osim što se prvo uklanja prethodna brzina koju je igrač imao.

```
private void Heal()
{
    if (pig_formula >= 2 && health < maxHealth-1)
    {

                health += 2;
                pig_formula -= 2;
                audioManager.Play ("heal");

    }
}
public void TakeDamage(int damage)
{
    health -= damage;
    gameObject.GetComponent<Animation> ().Play ("pig_damage_flash");
    audioManager.Play ("pig_hit");
}
void ChangeDirection()
{
    moveRight = !moveRight;
    Vector3 scale = transform.localScale;
    scale.x *= -1;
    transform.localScale = scale;

}
```

Programski kod 3.3. Funkcionalnosti igrača

Metoda Heal() služi kako bi se igraču vratio izgubljeni život ukoliko ima dovoljan broj svinjskih formula. Ukoliko je broj svinjskih formula veći ili jednak 2 te igraču nedostaju barem 2 jedinice zdravlja, moguće je povećati razinu zdravlja. Kako bi bilo moguće primiti štetu, metoda TakeDamage() prima kao argument količinu štete, te se preko Animation komponente pokreće animacija primitka štete popraćena odgovarajućim zvukom. Pri promjeni smjera gledanja metodom ChangeDirection(), moveRight varijabla mijenja stanje te se izmjenjuje localScale vrijednost transform komponente tako da se x os invertira.

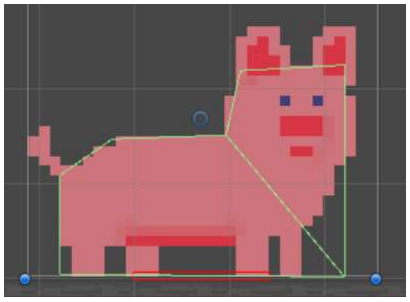
```
public void playFootsteps()
{
    if(grounded)
        audioManager.Play ("walk");

}
public void checkGround()
{
    grounded = Physics2D.OverlapBox (GroundChecker.position, new Vector2 (CheckRangeX, CheckRangeY), 0,WhatIsGround);
    animator.SetBool ("Ground", grounded);
}
void OnDrawGizmosSelected()
{
    Gizmos.color = Color.red;
    Gizmos.DrawWireCube (GroundChecker.position, new Vector3(CheckRangeX,CheckRangeY,1));
}
```

Programski kod 3.4. Funkcionalnosti igrača

Tijekom izvođenja animacije trčanja poziva se metoda playFootsteps(), te pod uvjetom da je varijabla grounded istinita audio upravitelj proizvodi zvuk trčanja. Metoda checkGround() provjerava nalazi li se igrač na tlu pomoću korištenja OverlapBox metode Physics2D komponente. Kao argumenti koriste se pozicija objekta koji se na sceni služi za provjeru tla, što je u ovome slučaju predstavljeno GroundChecker referencom, Vector2 koji u sebi sadrži raspon koji će *collider* koji provjerava doticanje tla zahvaćati , kut i sve podloge koje se smatraju tlom.

Kako bi oblik collider-a koji provjerava tlo bio vidljiv, koristi se funkcija OnDrawGizmosSelected() koja ga unutar inspektora iscrtava crvenim linijama.



Sl. 3.2. Prikaz collider-a praseta i provjere tla

```
    if (Input.GetKeyDown (KeyCode.Mouse0) && !attacking && grounded)
    {
        attackCollider.enabled = true;
        attackTimer = 0;
        attacking=true;
    }
    if (attacking == true)
    {
        attack ();
    }
        animator.SetBool("attack",attacking);
}
private void attack()
{
    if (attackTimer < attackTime)
    {
        attackTimer += Time.deltaTime;
    }
    else
    {
        attacking = false;
        attackCollider.enabled = false;
    }
```

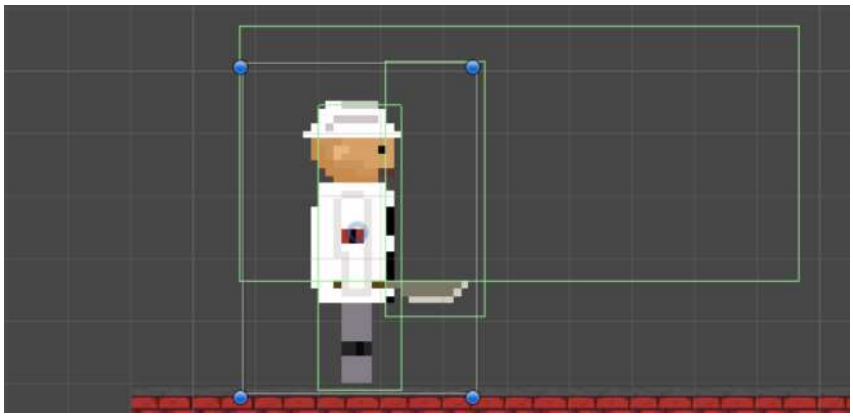
Programski kod 3.5. Kod za napad igrača

Kako bi igrač mogao napadati neprijatelje, korišten je programski kod 3.5.. Ukoliko je igrač pritisnuo lijevi klik, *bool* varijabla attacking koja predstavlja napad je neistinita i igrač se nalazi na tlu dolazi pokreće se napad. Dolazi do aktiviranja *collider*-a zaslužnog za napad neprijatelja, te se attackTimer zaslužan za odbrojavanje trajanja napada postavlja na nula i varijabla attacking se postavlja u istinito stanje. Ukoliko je unutar drugog uvjeta attacking istinit, poziva se metoda attack() koja upravlja pojedinim napadom. Unutar attack() metode odbrojava se vrijeme trajanja napada, nakon čega dolazi do deaktiviranja *collider*-a korištenog za napad i postavljanja varijable attacking u *false*.

3.1.2. Mesar

Mesar je neprijatelj na kojega igrač nailazi na prvoj razini. Naoružan satarom, mesar patrolira platformom na kojoj se nalazi, te ukoliko mu igrač ušeta u *collider* koji se koristi za provjeru blizine neprijatelja počinje se približavati igraču te ga napadati. Ukoliko se igrač ne nalazi u blizini, mesar patrolira platformom određeno vrijeme nakon kojega uzima kratki predah.

Sl. 3.3. Sprite sheet mesara



Sl. 3.4. Collideri mesara

Na slici 3.4. vidljiva su 3 BoxCollider2D-a [6] koje mesar koristi : *collider* mesara koji onemogućuje prolaz igraču kroz mesara, *collider* koji se koristi za napade satarom i najveći *collider* koji se koristi kod provjere nalazi li se igrač u blizini.

```
void Update ()
{
    checkGround ();
    if (range == false||(range==true&&grounded==false)) {
        if (passiveState == "idle") {
            animator.SetBool ("Idle", true);
            animator.SetBool ("Walk", false);
            idleTimer += Time.deltaTime;
            if (idleTimer > idleTime) {
                passiveState = "walk";
                idleTimer = 0;
            }
        }
        if (passiveState == "walk") {
            animator.SetBool ("Walk", true);
            animator.SetBool ("Idle", false);
            walkTimer += Time.deltaTime;
            move ();
            if (walkTimer > walkTime) {
                passiveState = "idle";
                walkTimer = 0;
            }
        }
    }
    else if(range==true&&grounded)
    {
```

Programski kod 3.6. Kod za pasivno ponašanje mesara

Mesar, u ovisnosti o tome nalazi li mu se igrač u blizini, može imati 2 oblika ponašanja koja se odnose na čekanje i patroliranje te proganjanje i napadanje. Kao što je vidljivo prema programskom kodu 3.6., prvo se poziva metoda checkGround() koja provjerava može li mesar nastaviti hodati u istome smjeru. Unutar *if* uvjeta , ako je range varijabla koja se odnosi na prisutnost igrača neistinita ili je range istinit no grounded varijabla koja predstavlja mogućnost hoda u istome smjeru neistinita , tada će mesar odrađivati pasivne radnje patroliranja i odmaranja. Ukoliko mesar odrađuje pasivne radnje, prvo se unutar *if* uvjeta provjerava varijabla passiveState koja se odnosi na trenutnu pasivnu radnju. Ako se radi o idle pasivnoj radnji dolazi do postavljanja parametara Idle u istinito i Walk u neistinito unutar animatora mesara kako bi se prikazala animacija odmaranja. Varijabla idleTimer nam služi radi toga da odbrojava koliko dugo je mesar odmarao, te ako je idleTimer veći od idleTime varijable koja predstavlja očekivano vrijeme odmaranja tada se passiveState postavlja u walk te se idleTimer resetira. Kod drugog *if* uvjeta pasivnih radnji vrijede ista pravila, osim toga da se također poziva metoda move() koja služi za kretanje mesara kako bi on mogao patrolirati.

```
else if(range==true&&grounded)
{
    animator.SetBool ("Idle", false);
    animator.SetBool ("Attack", attack);
    if ((playerPosition.position.x - transform.position.x) > 2.6f)
    {
        attack = false;
        setLookingSide (true);
        animator.SetBool ("Walk", true);
        move ();
    }
    else if ((playerPosition.position.x - transform.position.x) < -2.6f)
    {
        attack = false;
        setLookingSide (false);
        animator.SetBool ("Walk", true);
        move ();
    }
    else
    {
        animator.SetBool ("Walk", false);
        if (!attack) {
            attackTimer = 0;
            attackCollider.active = true;
            attack = true;
        }
        else
        {
            if (attackTimer < attackTime)
            {
                attackTimer += Time.deltaTime;
            }
            else
            {
                attack = false;
                attackCollider.active = false;
            }
        }
    }
```

Programski kod 3.7. Kod za proganjanje igrača

Ako se igrač nalazi u blizini mesara te je varijabla grounded također istinita, tada mesar proganja i napada igrača. Prvo, kao što je na programskom kodu 3.7. vidljivo, parametar Idle animatora mesara se postavlja u false te se parametar Attack postavlja u ovisnosti o varijabli attack. U naredna dva *if* uvjeta provjerava se nalazi li se igrač lijevo ili desno od mesara te je li od mesara udaljen za određenu vrijednost. Oduzimanjem pozicije x koordinate igrača i mesara uspoređujemo je li rezultat veći od određene vrijednosti. Ukoliko je vrijednost veća, attack varijabla se postavlja u neistinito stanje, te se poziva i setLookingSide() metoda kojoj kada se proslijedi *true* vrijednost mesar će se okrenuti desno kako bi gledao desno prema igraču. Također parametar Walk animatora se postavlja u istinito stanje te se poziva metoda move(). Unutar *else if* provjere vrijedi ista logika , samo što će tu u setLookingSide() metodu biti proslijeđena false vrijednost te će mesar gledati prema lijevo. Kada u kodu dođe do *else* slučaja, tada je igrač dovoljno blizu mesaru kako bi ga on

mogao napasti. Prvo se parametar animatora Walk postavlja u false te se provjerava je li varijabla attack neistinita, to jest napada li mesar trenutno igrača. Pod uvijetom da ga ne napada, attackTimer koji služi za odbrojavanje svakoga pojedinog napada se resetira te se attack varijabla postavlja u istinito stanje i uključuje se attackCollider kako bi došlo do kolizije s igračem i registriranjem napada. Ukoliko je attack varijabla bila istinita, tada se attackTimer varijabla uspoređuje sa attackTime koja predstavlja ukupno trajanje napada te ako napad završava attack se postavlja u false i attackCollider postaje neaktivan.

```
private void move()
{
    transform.Translate (Vector2.right * butcherSpeed*Time.deltaTime);
    if (!grounded)
    {
        if (moveRight == true)
        {
            setLookingSide (false);
        }
        else
        {
            setLookingSide (true);
        }
    }
}
public void setRange(bool inRange)
{
    range = inRange;
}
```

Programski kod 3.8. Pomoćne metode ButcherAI skripte

Na programskom kodu 3.8. prikazane su metode move() i setRange() koje se koriste u mesarevom kodu. Metoda move() vrši pomicanje mesara tako da se pri translaciji kao argument zadaje umnožak brzine mesara, vrijeme od izvršavanja zadnjega framea i Vector2 vrijednosti. Tada se provjerava ukoliko varijabla grounded nije istinita, nakon čega se provjerava kreće li se igrač desno te se poziva setLookingSide() kako bi se promjenio smjer. Metoda setRange() poziva se iz pomoćne skripte Butcher_range koja postavlja vrijednost varijable range ovisno o tome nalazi li se igrač u blizini.

```
private void setLookingSide(bool side)
{
    if (side == true)
    {
        transform.eulerAngles = new Vector3 (0, 0, 0);
        moveRight = true;
    }
    else
    {
        transform.eulerAngles = new Vector3 (0, -180, 0);
        moveRight = false;
    }
}
public void TakeDamage(int ammount)
{
    gameObject.GetComponent<Animation> ().Play ("Butcher_damage");
    health -= ammount;
    audioManager.Play ("butcher_hit");
}
public void checkGround()
{
    RaycastHit2D ground = Physics2D.Raycast (groundDetect.position, Vector2.down, 0.3f,walkable);
    if (ground.collider == false)
    {
        grounded = false;
    }
    else
    {
        grounded = true;
    }
}
```

Programski kod 3.9. Pomoćne metode BucherAI skripte

Kako bi se mesar mogao okrenuti, korištena je metoda setLookingSide() koja kao argument prima stranu prema kojoj gleda. Unutar *if* uvjeta provjerava se je li varijabla side istinita, te ukoliko je istinita eulerAngles svojstvo transform komponente se postavlja u vektor koji predstavlja rotaciju igrača. Kod else slučaja vidljivo je da ukoliko igrač gleda prema lijevo dolazi do rotacije od -180 stupnjeva oko y osi. Metoda TakeDamage() prima kao argument količinu štete koja se nanosi mesaru, koja se tada oduzima od zdravlja mesara predstavljenog varijablom health. Također dohvaća se Animation komponenta te se pušta animacija koja naznačuje da je mesar udaren popraćeno i sa zvukom proizvedenim pomoću upravitelja zvukom.

3.2.Prepreke

3.2.1. Toranj

Toranj služi kao automatizirana zaštita protiv izbjeglih svinja u klaonici. Ovisno o udaljenosti igrača, toranj može biti isključen ili u stanju pripravnosti.

Sl 3.5. Sprite sheet tornja

```
void Update () {
    animator.SetBool ("Awake", awake);
    CheckRange ();
    if (player.transform.position.x > transform.position.x)
    {
        lookingRight = true;
    }
    else
    {
        lookingRight = false;
    }
}
void CheckRange()
{
    playerDistance = Vector3.Distance (transform.position, player.position);
    if (playerDistance < awakeRange)
    {
        awake = true;
    }
    else
    {
        awake = false;
    }
```

Programski kod 3.10. Kod za aktiviranje tornja

Toranj pomoću metode CheckRange() provjerava nalazi li se igrač u neposrednoj blizini tornja kako bi se varijabla awake mogla postaviti u *true*, što rezultira time da se awake prosljeđuje animatoru koji je u ovome slučaju predstavljen pod imenom animator. Također kako bi toranj znao s koje strane je potrebno stvoriti metak, uspoređuje se pozicija tornja naspram pozicije igrača.

```
public void Attack(bool attackRight)
{

    bulletTimer += Time.deltaTime;
    if (bulletTimer >= reloadTime)
    {
        Vector3 direction = player.transform.position - transform.position;
        direction.Normalize ();
        if (attackRight) {
            GameObject bulletClone;
            bulletClone = Instantiate (bullet, BulletSpawnRight.transform.position, BulletSpawnRight.transform.rotation)as GameObject;
            bulletClone.GetComponent<Rigidbody2D> ().velocity = direction * bulletSpeed*Time.deltaTime;
            audioManager.Play ("shoot");
            bulletTimer = 0f;
        }
        else
        {
            GameObject bulletClone;
            bulletClone = Instantiate (bullet, BulletSpawnLeft.transform.position, BulletSpawnLeft.transform.rotation)as GameObject;
            bulletClone.GetComponent<Rigidbody2D> ().velocity = direction * bulletSpeed*Time.deltaTime;
            audioManager.Play ("shoot");
            bulletTimer = 0f;
        }
    }
}
```

Programski kod 3.11. Kod za ispaljivanje metaka iz tornja

Pozivanje Attack metode koja je opisana u skripti Turret vrši se iz skripte Turret_attack u kojoj se određuje strana sa koje igrač dolazi, što se prosljeđuje metodi Attack kao *bool* varijabla.

Unutar Attack metode nalazi se varijabla bulletTimer koja služi za praćenje vremena od ispaljivanja metaka. Nakon što bulletTimer ima veću vrijednost od reloadTime, koji se odnosi na vrijeme potrebno tornju kako bi pripremio drugi metak, određuje se smjer u kojemu se igrač nalazi naspram tornja. Nakon toga, provjerom nalazi li se igrač lijevo ili desno od tornja dolazi do instanciranja metka na mjestu objekta BulletSpawnRight ili BulletSpawnLeft, te se bulletTimer postavlja na 0 kako bi započelo odbrojavanje do novoga ispaljivanja metka.

3.2.2. Šiljci

Šiljci predstavljaju prepreku igraču koja oduzima zdravlje ukoliko se preko nje prijeđe.

Osim oduzimanja zdravlja, prelaskom preko šiljaka glavni lik refleksivno odskakuje prema gore u svrhu izbjegavanja šiljaka.

Sl. 3.6. Izgled šiljaka

```
public void LaunchVertical()
{
    float timer = 0f;
    rigidbody.velocity = new Vector2 (rigidbody.velocity.x, 0);
    if (timer < 0.15f)
    {
        rigidbody.AddForce (new Vector2 (0, launchStrength),ForceMode2D.Impulse);
        timer += Time.deltaTime;
    }
}
```

Programski kod 3.12. Kod za odbacivanje igrača u zrak

Pri koliziji sa šiljcima, dolazi do pozivanja metode LaunchVertical() sa programskog koda 3.12. koja služi za odbacivanje igrača u zrak i metode TakeDamage() koja smanjuje igračevo zdravlje. Unutar LaunchVertical() metode kako bi igrač svaki puta, neovisno o visini sa koje je padao, odskočio uvijek na istu visinu brzina po y osi postavljena je na 0 dok brzina po x osi ostaje ista. Sve dok je timer varijabla manja od unaprijed određenog vremena, zbraja joj se protok vremena u sekundama te se tijelu igrača dodaje sila koja ga lansira prema gore pomoću AddForce() metode koja kao argumente prima vektor sile i način na koji sila djeluje.

3.2.3. Ljepljiva sluz

Ljepljiva sluz je postavljena od strane mesara kako bi usporila prase, te olakšala posao tornjevima.

Sl. 3.7. Izgled ljepljive sluzi

```
void OnTriggerEnter2D(Collider2D collider)
{
    if (collider.CompareTag ("Player"))
    {
        player.topspeed *= 0.5f;
        player.JumpForce *= 0.8f;
        audioManager.Play ("slime");
    }
}
void OnTriggerExit2D(Collider2D collider)
{
    if (collider.CompareTag ("Player"))
    {
        player.topspeed *= 2f;
        player.JumpForce *= 1.25f;
    }
}
```

Programski kod 3.13. Kod za usporavanje igrača ulaskom u sluz

Na programskom kodu 3.13. vidljiv je kod za usporavanje igrača i smanjenje jačine skoka ulaskom u sluz.

Prolaskom kroz collider sluzi , koji je označen kao *trigger*, poziva se OnTriggerEnter2D() koji prvo provjerava radi li se o igraču pomoću CompareTag() metode. Ukoliko se radi o igraču, dolazi do smanjenja brzine hoda i jačine skoka. Izlaskom iz sluzi poziva se OnTriggerExit2D() koji igraču vraća brzinu i jačinu skoka.

3.3.Platforme

Platforme u igri predstavljaju objekte po kojima igrač može hodati, te služe za dosezanje određenih lokacija. U igri postoji 3 različita tipa platformi : statične platforme, platforme koje padaju nakon određenog vremena nakon što igrač skoči na njih te platforme koje se miču iz jednoga mjesta u drugo.

Sl. 3.8. Izgled platforme

```
void Update ()
{
    if (move == true)
    {
        transform.position = Vector3.MoveTowards (transform.position, target.position, speed*Time.deltaTime);
    }
}
void OnCollisionEnter2D(Collision2D collision)
{
    move = true;
    collision.collider.transform.SetParent (transform);
}
void OnCollisionExit2D(Collision2D collision)
{
    collision.transform.SetParent (null);
}
```

Programski kod 3.14. Kod pokretne platforme

Prema programskom kodu 3.14. vidimo da se pri pozivanju OnCollisionEnter2D() kod kolizije varijabla move postavlja u istinito stanje, te se platforma postavlja kao roditeljski objekt igraču. Time, pri pokretanju platforme igrač će se kretati zajedno sa platformom. Pri izlasku iz kolizije i pozivanju OnCollisionExit2D(), roditeljski objekt igraču je postavljen kao *null* kako bi se igrač ponovno samostalno kretao.

Kretanje platforme definirano je unutar Update() metode, tako da nakon što je bool varijabla move postavljena u istinito stanje dolazi do izmjene pozicije platforme pomoću MoveTowards() metode koja kao argumente prima trenutnu poziciju, poziciju mete i brzinu kretanja prema meti koja je pomnožena sa tokom vremena.

```
    void OnCollisionEnter2D(Collision2D collision)
    {
        if (collision.collider.CompareTag("Player"))
            {

            StartCoroutine (dropPlatform());
            }
    }
            private IEnumerator dropPlatform()
            {
                yield return new WaitForSeconds(waitDuration);
                rigidBody.isKinematic = false;
                yield break;
            }

}
```

Programski kod 3.15. Kod padajuće platforme

Padajuća platforma pri koliziji prvo provjerava radi li se o koliziji s igračem što je vidljivo kod poziva OnCollisionEnter2D(), gdje se prvo sa CompareTag() provjerava radi li se o igraču. Ukoliko se radi o igraču, poziva se korutina dropPlatform() koja služi za odbrojavanje do pada platforme. Unutar dropPlatform() korutine, prvi se poziva WaitForSeconds() koji kao argument uzima duljinu trajanja odbrojavanja do pada platforme nakon čega se isKinematic svojstvo RigidBody2D komponente postavlja na *false* te platforma započinje propadanje prema dolje.

3.4.Zvukovi i glazba

Glazba i zvukovi [7][8] unutar igre reproducirani su pomoću AudioSource [9] komponente. AudioSource omogućuje upravljanje nad audio klipovima kao što je upravljanje nad glasnoćom, reproduciranje, zaustavljanje i pauziranje. Kako bi igrač mogao čuti zvukove nastale iz AudioSource-a, u sceni je potrebno imati AudioListener [10] komponentu. Uloga AudioListener komponente je takva da osluškuje zvukove koji se događaju oko igrača te ih usmjerava na igračeve zvučnike. Radi lakoće upravljanja nad zvukovima, stvoren je audio upravitelj koji koristi zvučne klipove predstavljene pomoću Sound klase.

```
[System.Serializable]
public class Sound
{
    public string clipName;
    public AudioClip audioClip;
    public bool loop;
    [Range(0f,1f)]
    public float volume;
    public AudioSource source;

}
```

Programski kod 3.16. Sound klasa

Prema programskom kodu 3.16. vidljivi su elementi kojima je predstavljen pojedini zvuk. Varijabla clipName predstavlja naziv kojime je audio klip predstavljen, audioClip predstavlja referencu za audio klip, loop varijabla predstavlja treba li se zvuk ponavljati, volume predstavlja glasnoću klipa te je iznad njega prikazan i raspon (eng. *Range*) što olakšava modificiranje u inspektoru. Varijabla source predstavlja referencu na AudioSource komponentu koja reproducira pojedini zvuk. Također, korištenjem System.Serializable povrh klase nam omogućuje uređivanje pojedinog Sound objekta koji se nalazi unutar niza iz inspektora.

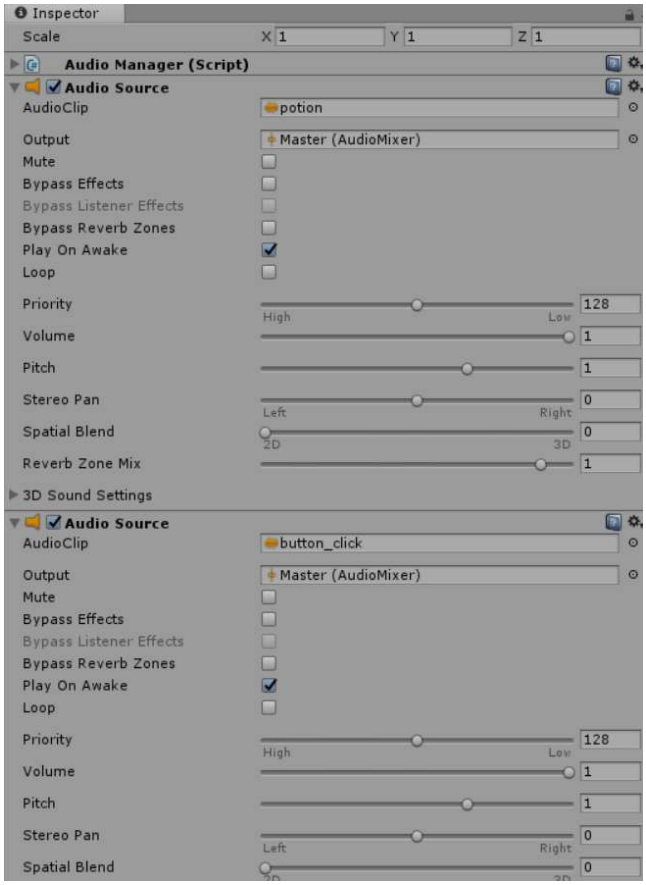
```
public class AudioManager : MonoBehaviour {
    public Sound[] sounds;
    public AudioMixerGroup audioMixer;
    void Awake ()
    {
        foreach (Sound sound in sounds)
        {
            sound.source = gameObject.AddComponent<AudioSource> ();
            sound.source.outputAudioMixerGroup = audioMixer;
            sound.source.clip = sound.audioClip;
            sound.source.volume = sound.volume;
            sound.source.loop = sound.loop;
        }

    }

    public void Play(string name)
    {
        foreach (Sound sound in sounds)
        {
            if (sound.clipName == name)
            {
                sound.source.Play ();
            }
        }
    }
    public void Stop(string name)
    {
        foreach (Sound sound in sounds)
        {
            if (sound.clipName == name)
            {
                sound.source.Stop ();
            }
        }
    }
}
```

Programski kod 3.17. Upravitelj zvukovima

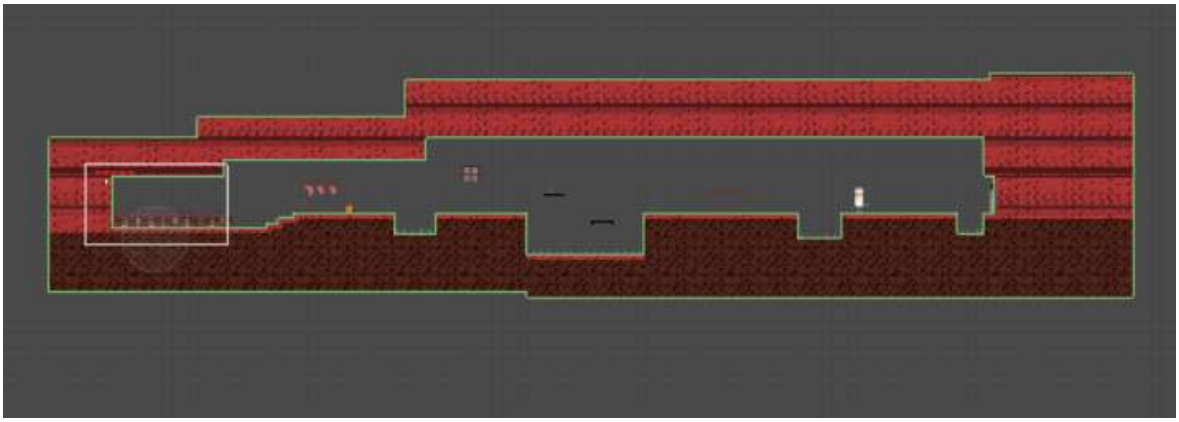
Upravitelj zvukovima sa programskog koda 3.17. služi radi bolje kontrole i lakoće nad reproduciranjem zvukova. Sastoji se od Sound niza koji sadrži zvukove i audioMixer varijable koja se koristi kako bismo svakome zvuku pridijelili isti Audio Mixer kako bismo iz postavki mogli utjecati nad glasnoćom svih zvukova odjednom. Ovakav pristup nam omogućuje dodavanje zvukova u sounds listu iz inspektora. Pri pokretanju igre, pozivanjem Awake() metode se unutar *foreach* petlje za svaki zvuk stvara AudioSource komponenta na objektu na kojemu se nalazi AudioManager skripta. Također, pri stvaranju AudioSource komponente postavljaju se preferencije koje se povlače iz svakog pojedinog Sound objekta kao što su AudioMixerGroup, naziv klipa , glasnoća te treba li se zvuk ponavljati više puta.



Sl. 3.9. Prikaz par stvorenih AudioSource komponenti

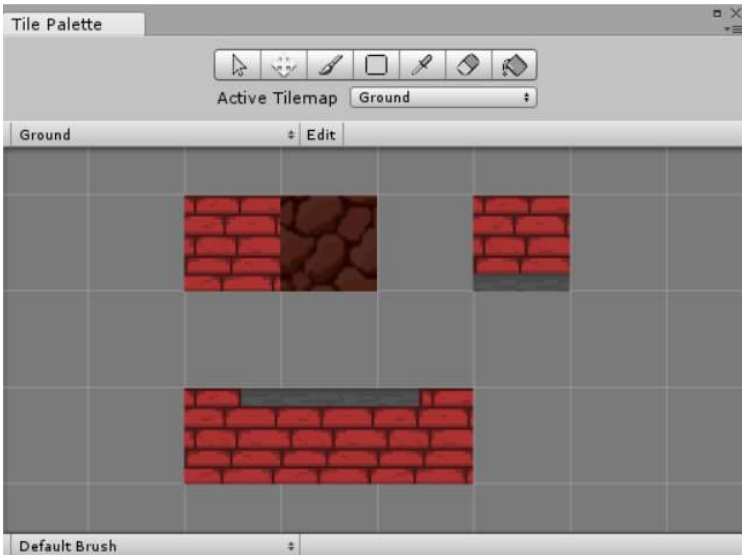
3.5.Izrada razina

Za izradu razina igre korišten je Tilemap [11] sustav Unity-a koji omogućuje lagano stvaranje razina na osnovu slaganja blokova koji se nalazi unutar palete blokova.



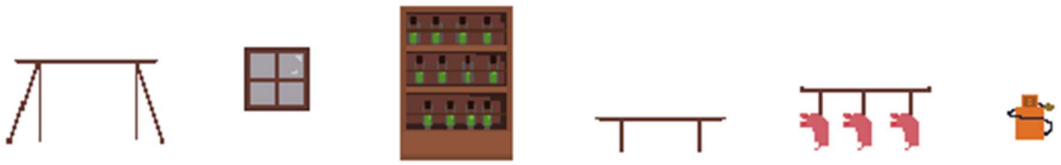
Sl. 3.10. Prikaz prve razine igre

Na slici 3.10. prikazan je izgled prve razine igre, te je važno za napomenuti da Tilemap koristi Grid [12] komponentu prilikom slaganja blokova razine. Također, Tilemap sustav ima mogućnost korištenja Tilemap Collider 2D komponente koja omogućuje koliziju između igrača i dodanih blokova. Blokovi su uokvireni zelenom bojom što označuje da se na njima nalazi *collider*, te je nužno napomenuti da je takav oblik *collider*-a postignut pomoću Composite Collider 2D komponente koja grupira *collider*-e svakoga pojedinoga bloka.



Sl. 3.11. Paleta korištena pri stvaranju prve razine

Na slici 3.11. nalazi se sučelje koje sadrži alate i opcije korištene pri slaganju blokova. Vidljivo je da su ponuđeni alati slični alatima koji se nalaze u većini programa korištenih za crtanje. Ispod alata mogće je odabrati koji je trenutno aktivan Tilemap kako bismo odredili gdje želimo dodavati blokove. Također, prikazani su i blokovi korišteni prilikom stvaranja razine.

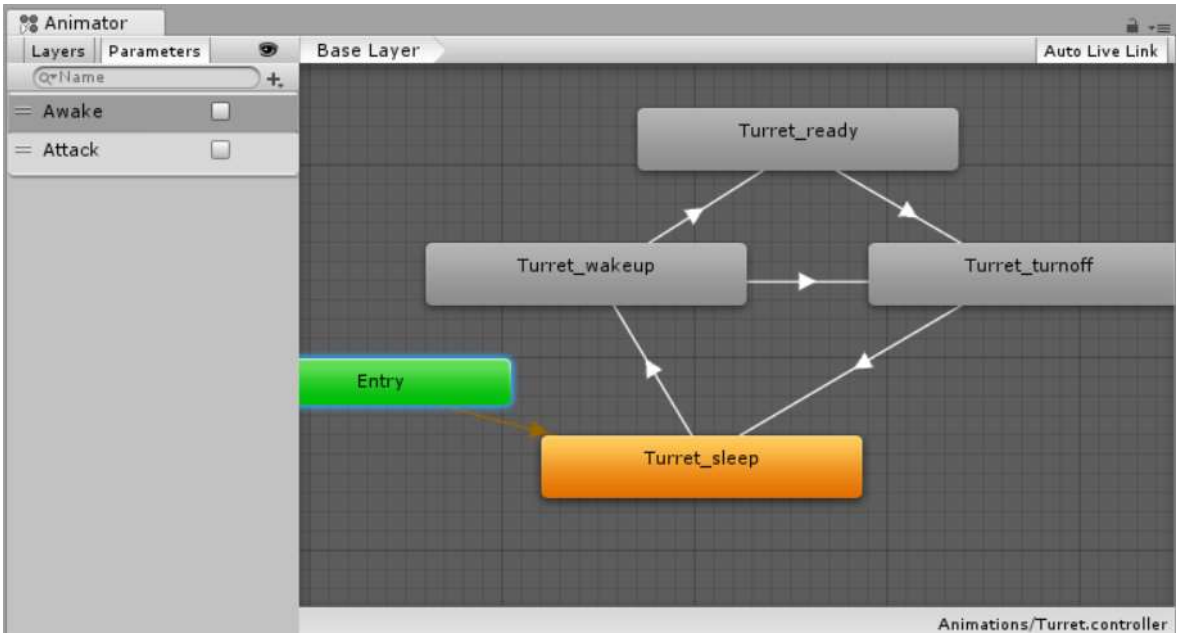


Sl. 3.12. Neke od stvari postavljenih u pozadinu

Kako bi se dočarala atmosfera u igri, na slici 3.12. vidljivi su objekti koji su postavljeni u pozadinu. Radi se o objektima sa kojima igrač ne može imati interakciju te se nalaze u igri samo radi izgleda.

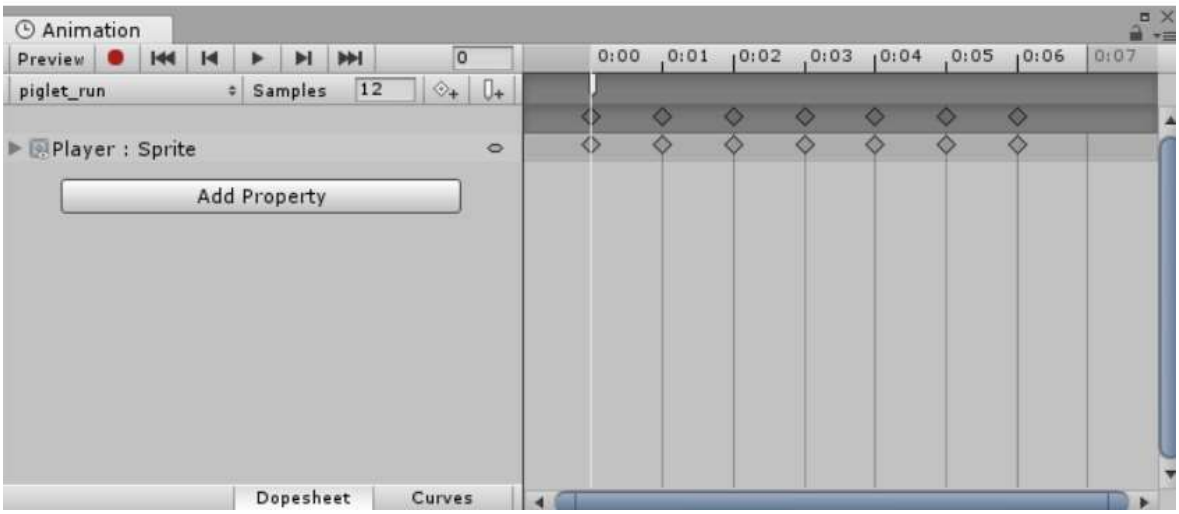
3.6.Animacije

Kod animiranja likova unutar igre, dolazi do stvaranja Animator Controller [13] komponente koja u Unity-u pruža mnoge mogućnosti vezane uz animiranje. Pružen je sustav na kojemu je prikazan stroj stanja što pripomaže pri translaciji kod ponašanja likova.



Sl. 3.13. Prikaz stanja likova

Na slici 3.13. vidljiv je primjer kontrolera koji je korišten kod tornja prikazan unutar Animator prozora. Prvo stanje u kojemu se toranj nalazi odnosi se na stanje spavanja, što je vidljivo prema tome da sa zeleno označenog ulaznog bloka *Entry* dolazi do tranzicije prema stanju spavanja. Postoji više različitih stanja koji na sebi mogu sadržavati animacijski klip, te je važno za znati da se tranzicije odvijaju u ovisnosti o uvjetima o kojima ovise. Sa lijeve strane vidljvi su parametri Awake i Attack koji se koriste kod uvjeta pri tranzicijama. Primjer tomu bilo bio bi slučaj kada se toranj nalazi u stanju spavanja, što je prekinuto nakon što se parametar Awake postavlja u istinito stanje te dolazi u prijelaza na stanje buđenja što je popraćeno animacijom. Nakon što se toranj aktivira ulazi u stanje spremnosti što postavlja parametar Attack u istinito stanje te omogućuje ispaljivanje projektila prema igraču. Još jedna od mogućnosti koje su u ovome sustavu pružene su skripte u kojima je mogući opisati ponašanje lika u određenom trenutku kao što može biti ulazak u stanje, izlazak iz stanja te akcije tijekom trajanja stanja.

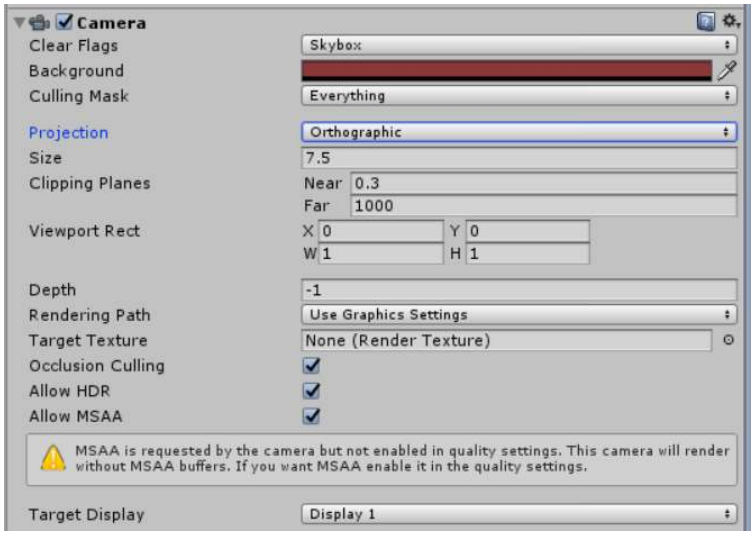


Sl. 3.14. Postupak stvaranja animacijskih klipova

Animation prozor omogućuje nam uređivanje animacijskih klipova. Na vremenskoj traci vidljiva je pozicija svake sličice animacije u vremenu, te je također moguće puštanje animacije te uređivanje brzine izvođena animacije. Na slici 3.14. vidljiva je bijela oznaka pri trenutku pokretanja animacije, što predstavlja događaj kojega će pokrenuti izvođenje animacije kao što je na slici prikazano pokretanje zvuka kretanja.

3.7.Kamera

Camera [14] predstavlja komponentu koja služi za prikazivanje okoliša u kojemu se igrač nalazi.



Sl. 3.15. Prikaz kamere unutar inspektora

Sadrži brojne postavke koje utječu nad načinom kojim je svijet prikazan igraču kao što su boja pozadine, način projekcije koji može biti ortografski ili perspektivni, veličina kamere te udaljenost nakon koje kamera ne prikazuje objekte.

```
void FixedUpdate()
{
    Vector3 targetPos = target.position;
    targetPos.z = transform.position.z;
    targetPos.y += verticalShift;
    transform.position = Vector3.SmoothDamp (transform.position, targetPos, ref velocity, smoothTime);
}
```

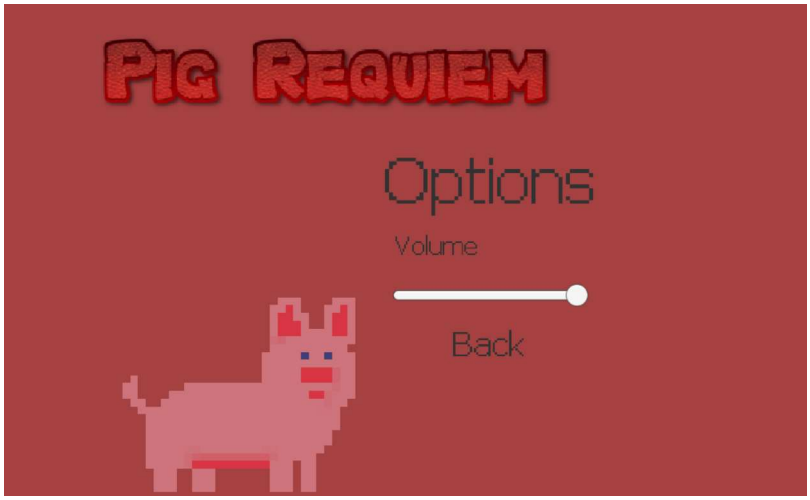
Programski kod 3.18. Kod za kretanje kamere

Prema programskom kodu 3.18. vidljiva je skripta zaslužna za kretanje kamere prema igraču. Prvo se dobavlja trenutna pozicija igrača koji je predstavljen kao target, te se to skladišti u Vector3 varijablu imena targetPos. Nakon toga, potrebno je postaviti i z koordinatu targetPos-a na istu poziciju kao što je i igrač. Vidljivo je da je nakon toga koordinata y targetPos-a pomaknuta za verticalShift vrijednost , što se odnosi na proizvoljnu vrijednost dodanu kako bi kameru postavili na vertikalnu poziciju po vlastitoj želji. Nakon toga mijenja se pozicija kamere uz pomoću SmoothDamp() metode koja nam služi za blaži prijelaz kamere prema igraču. Ta metoda kao argumente prima poziciju kamere, poziciju mete, brzinu te vrijeme prijelaza.

3.8.Izbornici i upravitelj igrom

Ulaskom u igru igraču se prikazuje glavni izbornik , iz kojih je moguće pristupti postavkama i odabiru razina.

Sl. 3.16. Glavni izbornik



Sl. 3.17. Izbornik postavki

Sl. 3.18. Odabir razine

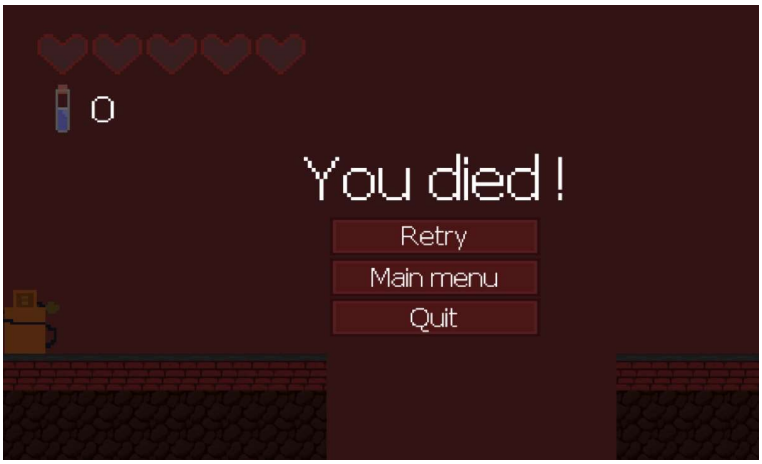
Na slikama 3.16., 3.17. i 3.18. vidljivi su izbornici kojima je moguće pristupiti pri ulasku u igru. Na njima su vidljive klikabilne opcije koje su predstavljene pomoću Button [15] komponente koja služi za pokretanje određenih akcija. Pri kliku dolazi do blage promjene boje i proizvodnje zvuka kako bi bilo naznačeno da je došlo do odabira. Funkcionalnost Button-a izrađena je uz pomoću On Click () postavki unutar Unity inspektora, te korištenjem jednostavnih skripti. Izbornici se izmjenjuju tako da unutar Canvas [16] komponente pri promjeni s jednog izbornika na drugi jedan sadržaj postavi kao neaktivan a drugi kao aktivan.

```
void Start ()
{
    levelsUnlocked = PlayerPrefs.GetInt ("levelsAvailable");
    for (int i = 0; i<=2; i++)
    {
        if (i<=levelsUnlocked)
        {
            levels [i].interactable = true;
        }
        else
        {
            levels [i].interactable = false;
        }
    }
}
public void loadLevel(int level)
{
    SceneManager.LoadScene (level);
}
```

Programski kod 3.19. Skripta za odabir razine

Kako sve razine ne bi bile dostupne pri prvome pokretanju igre te bi bilo potrebno slijedno preći razine korišten je programski kod 3.19.. Varijabla levelsUnlocked skladišti broj otključanih razina koji se dohvaća pomoću PlayerPrefs-a [17] koji se unutar Untiy-a koristi za skladištenje podataka koji nadilaze trenutnu scenu unutar koje se igrač nalazi. Unutar for petlje prolazi se kroz svaki

button koji se nalazi unutar levels niza, te se omogućuje interakcija sa njime ukoliko je razina otključana. Metoda loadLevel() koristi se unutar On Click () postavki unutar inspektora, te se pojedinom gumbu određuje koju će razinu učitati.



Sl. 3.19. Izbornik nakon smrti

Sl. 3.20. Izbornik nakon prelaska razine

Sl. 3.21. Izbornik tijekon pauziranja igre

Izbornici sa slika 3.19., 3.20. i 3.21. se pojavljuju tijekom igre u slučaju smrti, pauze ili prijeđene razine. Funkcionalnost navedenih izbornika opisana je unutar skripte gameManager, koja se unutar igre nalazi na gameManager objektu koji služi kao upravitelj igre.

```
public void loadNextLevel()
{
    SceneManager.LoadScene (SceneManager.GetActiveScene ().buildIndex + 1);

}
public void Resume()
{
    pause = false;
    Time.timeScale = 1;
}
public void Restart()
{
    SceneManager.LoadScene (SceneManager.GetActiveScene ().name);
}
public void loadMainMenu()
{
    Application.LoadLevel(0);
}
public void Quit()
{
    Application.Quit ();
}
```

Programski kod 3.20. Kod za funkcionalnost tipki

Unutar gameManager skripte sa programskog koda 3.20. vidljive su metode korištene pri kliku na tipke sa navedenih izbornika. Metoda loadNextLevel() se poziva nakon prelaženja razine i odabira opcije „*New level*“ tako da se pomoću SceneManager-a dohvaća trenutna aktivna scena, te se indeksu broja scene nadodaje 1 kako bi naznačili učitavanje iduće razine. Unutar Resume() metode pause varijabla je postavljena na *false*, te se tok vremena postavlja na 1 što predstavlja normalan

tok vremena. Metode Restart() i loadMainMenu() funkcioniraju slično kao i loadNextLevel(), te su zadužene za učitavanje specifičnih razina. Kako bi bilo moguće i izaći iz igre, nadodana je Quit() metoda koja gasi aplikaciju.

```
void Update ()
{
    if (playerController.health <= 0)
    {

        Time.timeScale = 0f;
        DestroyObject (player);
        gameOver.active = true;
    }
    if (levelDoor.win == true)
    {
        Time.timeScale = 0f;
        audioManager.Stop ("game_music");
        levelCompleted.active = true;
        if (PlayerPrefs.GetInt ("levelsAvailable") < currentLevel)
        {
            PlayerPrefs.SetInt ("levelsAvailable", currentLevel );
        }
        time.text = Time.timeSinceLevelLoad.ToString ();
        if (highScore == 0 || Time.timeSinceLevelLoad < highScore)
        {
            PlayerPrefs.SetFloat ("highScore_" + currentLevel, Time.timeSinceLevelLoad);
            newHighScore.active = true;
            recordTime.text = Time.timeSinceLevelLoad.ToString();
        }
        else
        {
            recordTime.text = highScore.ToString();
        }
```

Programski kod 3.21. Upravitelj igre

Također, gameManager skripta sadrži i kod koji služi za određivanje je li igrač završio trenutnu razinu. Na programskom kodu 3.21. unutar prvoga *if* uvjeta provjeravamo je li trenutno zdravlje igrača manje ili jednako 0, što bi naznačilo da je igra gotova. Ukoliko je uvjet istinit, protok vremena se postavlja na 0 što zaustavlja igru te se uništava igrač. Time, gameOver objekt se postavlja u aktivno stanje što će na Canvas komponenti prikazati izbornik sa slike 3.19.. Unutar drugoga *if* uvjeta se provjerava je li win varijabla unutar levelDoor reference istinita, što bi naznačilo da je igrač završio razinu. Ukoliko je uvjet istinit, vrijeme i glazba u pozadini se zaustavljaju te se postavljanjem levelCompleted objekta pojavljuje izbornik sa slike 3.20.. Nakon toga, provjerava se je li zadnja otključana razina manja od trenutne, te ako je postavlja se nova vrijednost unutar PlayerPrefs *integer* varijable levelsAvailable. Na slici 3.20. prikazano je i vrijeme koje je bilo potrebno za prelazak razine, što je postignuto tako da je referenci na Text komponentu pod imenom time text svojstvo postavljeno na vrijeme od kada je razina učitana. Nakon toga, unutar *if* uvjeta provjerava se je li varijabla highScore koja predstavlja najveći rezultat

postignut na trenutnoj razini nula ili veći od trenutnog postignutog što dovodi do postavljanja novog rekorda.

Postavljanje rekorda razine se izvodi tako da unutar PlayerPrefs-a postoji posebna varijabla za svaku razinu čije ime završava sa brojom razine. Pri novome rekordu pojavljuje se i tekst u kojemu je to naglašeno, tako da se referenca na Text komponentu newHighScore postavi u aktivno stanje.

```
if (Input.GetButtonDown ("Pause") && !levelDoor.win)
{
    pause = !pause;
}
if (pause)
{
    pausemenu.active = true;
    Time.timeScale = 0f;
}
else if (!pause && !levelDoor.win && playerController.health>0)
{
    pausemenu.active = false;
    Time.timeScale = 1f;

}
```

Programski kod 3.22. Kod za pauziranje igre unutar gameManager-a

Pri pritisku gumba naznačenog za pauzu, pod uvjetom da igra nije završila, pause varijabla se postavlja u istinito ili neistinito stanje ovisno je li igra već pauzirana. Ukoliko je igra pauzirana, pausemenu referenca nad izbornikom sa slike 3.21. se postavlja u istinito stanje te se tok vremena postavlja na 0. Ukoliko igra nije pauzirana, nije završena i zdravlje igrača je veće od 0, naznačeno je da protok vremena treba biti 1 i izbornik sa slike 3.21. ugašen.

3.9.Igračevo sučelje

Unutar igre, kako bi igrač bio svjestan trenutnoga stanja svinje prikazana je količina preostalih života svinje i broj skupljenih svinjskih formula.

Sl.3.22. Tri izgleda srca

Sl. 3.23. Svinjska formula

```
void Update () {
    health = player.health;
    pig_formula = player.pig_formula;
    for(int i=1;i<=hearts.Length;i++)
    {
        if ((health / 2) >= i) {
            hearts [i - 1].sprite = fullHeart;
        }
        else if ((health / 2) + 1 == i && health%2==1)
        {
            hearts [i - 1].sprite = halfHeart;
        }
        else
        {
            hearts[i-1].sprite=emptyHeart;
        }
    }
    pig_formula_text.text = pig_formula.ToString ();
```

Programski kod 3.23. Kod za prikazivanje statusa igrača

Programski kod 3.23. služi za prikaz trenutnog zdravlja i broja prikupljenih svinjskih formula igrača. Prilagođen je tako da svako srce može prikazivati 2 jedinice zdravlja, tako što srce može biti punoga oblika i polupraznoga oblika. Maksimalan iznos zdravlja praseta je 10, što je na sceni predstavljeno sa 5 srca prikazanih na UI komponenti Image koja služi za prikazivanje *sprite*-ova. Prije ulaska u *for* petlju, unutar *integer* varijabli health i pig_formula dohvaćamo trenutno zdravlje praseta i broj pokupljenih svinjskih formula. Nakon toga, ulaskom u *for* petlju prvo se provjerava da li trenutna image komponenta iz niza(eng. *Array*) hearts treba imati sliku punoga srca. To se provjerava tako da se trenutno zdravlje igrača podijeli sa 2, te se gleda da li trenutni element i u for petlji veći od rezultata dijeljenja. Ukoliko rezultat dijeljenja nije veći, iduća provjera je da li se radi o slučaju kada je trenutno zdravlje neparnoga broja te je *sprite* te Image komponenete postavljen na poluprazno srce. Ukoliko se ne radi ni o tome slučaju, image komponenti se pridjeljuje *sprite* praznoga srca.

4. ZAKLJUČAK

Dovršetkom završnog rada, izrađena je 2D platformer igra koja omogućuje interakciju igrača sa svijetom definiranim unutar igre. Pri razvoju igre kao glavni alat korišten je Unity koji služi kao game engine, dok je za stvaranje skripti pisanih u obliku C# koda korišten MonoDevelop. Likovi, prepreke kao i izgled razina stvoreni su unutar Aseprite-a koji je pružio lagano rješenje za stvaranje pixel arta. Svi zvukovi i glazba korišteni u igri preuzeti su sa interneta. Unutar rada detaljno su pojašnjeni ključni dijelovi stvaranja igre, prikazani su sprite-ovi koji tvore likove te pojašnjene su glavne skripte koje tvore njihovu funkcionalnost. Svrha igre bijeg je iz klaonice u kojoj se prase nalazi , te kako bi se to postignulo potrebno je izbjeći postavljene zamke te poraziti neprijatelje koji preprečuju put. Kako bi igra sadržavala više dodatnih mehanika, dodani su i svinjski eliksiri koji vraćaju zdravlje igrači, no ne osiguravaju prijelaz razine.

LITERATURA

[1] Platformer game

https://en.wikipedia.org/wiki/Platform_game ,pristupljeno: lipanj 2019.

[2] Unity

https://en.wikipedia.org/wiki/Unity_(game_engine) ,pristupljeno: kolovoz 2019.

[3] MonoDevelop

https://en.wikipedia.org/wiki/MonoDevelop ,pristupljeno: lipanj 2019.

[4] Aseprite

https://www.aseprite.org/ ,pristupljeno: lipanj 2019.

[5] Rigidbody2D

https://docs.unity3d.com/ScriptReference/Rigidbody2D.html ,pristupljeno: kolovoz 2019.

[6] BoxCollider2D

https://docs.unity3d.com/ScriptReference/BoxCollider2D.html ,pristupljeno: kolovoz 2019.

[7] Abundant Music procedural generator

https://pernyblom.github.io/abundant-music/index.html ,pristupljeno: srpanj 2019.

[8] Freesound

https://freesound.org/ ,pristupljeno: kolovoz 2019.

[9] AudioSource

https://docs.unity3d.com/ScriptReference/AudioSource.html ,pristupljeno: kolovoz 2019.

[10] AudioListener

https://docs.unity3d.com/Manual/class-AudioListener.html ,pristupljeno: kolovoz 2019.

[11] Tilemap

https://docs.unity3d.com/Manual/class-Tilemap.html ,pristupljeno: lipanj 2019.

[12] Grid

https://docs.unity3d.com/Manual/class-Grid.html ,pristupljeno: lipanj 2019.

[13] Animator Controller

https://docs.unity3d.com/Manual/class-AnimatorController.html ,pristupljeno: kolovoz 2019.

[14] Camera

https://docs.unity3d.com/Manual/class-Camera.html ,pristupljeno: kolovoz 2019.

[15] Button

https://docs.unity3d.com/Manual/script-Button.html ,pristupljeno: kolovoz 2019.

[16] Canvas

https://docs.unity3d.com/Manual/UICanvas.html ,pristupljeno: kolovoz 2019.

[17] PlayerPrefs

https://docs.unity3d.com/ScriptReference/PlayerPrefs.html ,pristupljeno: kolovoz 2019.

[18] 1001 Fonts (free fonts)

https://www.1001fonts.com/ ,pristupljeno: kolovoz 2019.

[19] Cool Text Graphics Generator

https://cooltext.com/ ,pristupljeno: srpanj 2019.

SAŽETAK

Završni rad temelji se na igri u kojoj prase kao glavni lik bježi iz klaonice kako bi izbjeglo okrutnu sudbinu. Igrač ima kontrolu nad horizontalnim kretanjem praseta, skakanjem, skupljanjem svinjskih formula koje se koriste u svrhu vraćanja izgubljenog zdravlja te napadima. Prelazak razine zahtijeva izbjegavanje zamki postavljenih unutar klaonice te borba s mesarima kojima je cilj zaustaviti prase pod svaku cijenu. Igra je razvijena unutar Unity game engine-a koji pruža podršku za velik broj platformi. MonoDevelop je korišten pri stvaranju C# skripti koje tvore funkcionalnosti objekata unutar igre. U završnome radu pojašnjene su glavne skripte koje omogućuju funkcionalnosti u igri kao što su kretanje kamere za igračem, reproduciranje zvukova preko upravitelja zvukova, pomicanje i pad platformi, prepreke te likovi unutar igre. Objekti i likovi vidljivi u igri stvoreni su u Aseprite-u, koji sadrži mnoge opcije koje pridonose lakoći stvaranja pixel arta kao što je spremanje sprite-ova u obliku sprite sheet-ova. Unity omogućuje rezanje sprite sheet-ova u zasebne sprite-ove što značajno pridonosi brzini stvaranja likova. Glazba i zvukovi preuzeti su sa interneta, te se unutar igre koristi upravitelj zvukovima kako bi se olakšao način stvaranja reproduciranja zvukova.

Ključne riječi: 2D platformer, klaonica, mesar, prase, prepreke, Unity

ABSTRACT

Title: 2D Platformer

The final paper is based on a game in which the pig as the main character escapes the slaughterhouse so it may avoid it's cruel fate. The player has the control over the horizontal movement of the pig, jumping, collection of pig formula used to regain lost health and attacks. Finishing the level requires avoiding the straps set up in the slaughterhouse and fighting butchers whose goal is to stop the pig at all costs. The game was developed within Unity game engine, which provides support for a large number of platforms. MonoDevelop was used to create C# scripts that form the functionality of in-game objects. The final paper clarifies the main scripts that provide in-game functionality such as moving the camera towards the player, playings sounds through the sound manager, moving and crashing platforms, obstacles and in-game characters. The objects and characters visible in-game were created in Aseprite, which contains many options that contribute to the ease of creating pixel art such as saving sprites in the form of sprite sheets. Unity allows you to cut sprite sheets into separate sprites, which significantly contributes to the speed of character creation. Music and sound effects were downloaded from the internet, and an audio manager is used within the game to make it easier to play sounds.

Key words: 2D platformer, butcher, obstacles, pig, slaughterhouse, Unity

ŽIVOTOPIS

Predrag Duvnjak rođen je 23.7.1997. u Našicama. Pohađao je Osnovnu školu Matije Gupca u Magadenovcu , gdje je pokupio interes za programiranjem nakon rada u Logo-u. Završetkom osnovne škole, upisuje prirodoslovnu matematičku gimnaziju u Srednjoj Školi Isidora Kršnjavoga u Našicama. Tijekom srednjoškolskog obrazovanja , započinje raditi na manjim igrama u Pythonu radi projekata na nastavi. Nakon toga, 2016. godine upisuje preddiplomski studij računarstva na Fakultetu elektrotehnike , računarstva i informacijskih tehnologija u Osijeku.

Vlastoručni potpis:

_____