Izrada 2D platformske računalne igre u Unity Engine-u

Rabar, Marin

Undergraduate thesis / Završni rad

2016

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:200:493552

Rights / Prava: In copyright / Zaštićeno autorskim pravom.

Download date / Datum preuzimanja: **2024-04-24**

Repository / Repozitorij:

Faculty of Electrical Engineering, Computer Science and Information Technology Osijek

SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA

Preddiplomski studiji računarstva

Izrada 2D platformske računalne igre u Unity Engine-u

Završni rad

Marin Rabar

Osijek, 2016.

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA **OSIJEK**

Obrazac Z1P - Obrazac za ocjenu završnog rada na preddiplomskom sveučilišnom studiju

Osijek, 21.09.2016.

Odboru za završne i diplomske ispite

Prijedlog ocjene završnog rada

Ime i prezime studenta:	Marin Rabar
Studij, smjer:	Preddiplomski sveučilišni studij Računarstvo
Mat. br. studenta, godina upisa:	R3525, 02.08.2013.
OIB studenta:	08795412668
Mentor:	Doc.dr.sc. Časlav Livada
Sumentor:	
Naslov završnog rada:	Izrada 2D platformske računalne igre u Unity Engine-u
Znanstvena grana rada:	**Programsko inženjerstvo (zn. polje računarstvo)**
Predložena ocjena završnog rada:	Izvrstan (5)
Kratko obrazloženje ocjene prema Kriterijima za ocjenjivanje završnih i diplomskih radova:	Primjena znanja stečenih na fakultetu: 3 Postignuti rezultati u odnosu na složenost zadatka: 3 Jasnoća pismenog izražavanja: 2 Razina samostalnosti: 2
Datum prijedloga ocjene mentora:	21.09.2016.
Datum potvrde ocjene Odbora:	28.09.2016.

Potpis mentora za predaju konačne verzije rada u Studentsku službu pri završetku studija:	Potpis:
	Datum:

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA **OSIJEK**

IZJAVA O ORIGINALNOSTI RADA

Osijek, 29.09.2016.

Ime i prezime studenta:	Marin Rabar
Studij:	Preddiplomski sveučilišni studij Računarstvo
Mat. br. studenta, godina upisa:	R3525, 02.08.2013.
Ephorus podudaranje [%]:	0

Ovom izjavom izjavljujem da je rad pod nazivom**: Izrada 2D platformske računalne igre u Unity Engine-u**

izrađen pod vodstvom mentora Doc.dr.sc. Časlav Livada

i sumentora

moj vlastiti rad i prema mom najboljem znanju ne sadrži prethodno objavljene ili neobjavljene pisane materijale drugih osoba, osim onih koji su izričito priznati navođenjem literature i drugih izvora informacija.

Izjavljujem da je intelektualni sadržaj navedenog rada proizvod mog vlastitog rada, osim u onom dijelu za koji mi je bila potrebna pomoć mentora, sumentora i drugih osoba, a što je izričito navedeno u radu.

Potpis studenta:

SADRŽAJ

1. UVOD 1

1.1. Zadatak završnog rada 1

2. UNITY 2

3. RAZVOJ IGRE 3

3.1. Ideja 3

3.2. Kamera 3

3.3. Likovi 5

3.3.1. Igrač (Player) 6

3.3.2. Protivnici (Enemy) 9

3.4. Grafika 14

3.5. Animator 16

3.6. Grafičko korisničko sučelje (GUI) 19

3.6.1. Glavni izbornik 20

3.6.2. Nivo 21

3.6.3. Sučelje unutar nivoa 23

4. ZAKLJUČAK 26

LITERATURA .....

SAŽETAK .....

ABSTRACT .....

ŽIVOTOPIS .....

PRILOZI .....

1. UVOD

U ovom završnom radu izrađen je računalni 2D platformer za jednu osobu pod nazivom Blu Bear. Glavni lik igre je plavi medvjed koji nije prihvaćen od ostatka šumske populacije zbog njegove boje krzna te se kroz svoju tugu bori protiv drugih životinja kako bi pobjegao što je i sam cilj igre. Igra je podržana na Windows, Linux i Mac OS operacijskim sustavima. Izrada igre je ostvarena u Unity *game engineu* te je sav kod pisan u C# programskom jeziku. Kodovi upravljaju svim mehanikama igre. Većina grafike je izrađena u web aplikaciji Piskel dok su zvukovi[1], fontovi[2] i ostali materijali preuzeti sa stranica slobodnog sadržaja.

1.1. Zadatak završnog rada

U završnom radu bilo je potrebno napraviti 2D platformsku igru s funkcijama i mehanikama koje bi jedna takva igra sadržavala. Za izradu rada bilo je potrebno koristiti Unity Engineu te C# programski jezik.

2. UNITY

Unity je razvojno okruženje za izradu video igara izrađeno od tvrtke Unity Technologies.[3] Nastao je kako bi se razvoj video igara proširio na veću populaciju i trenutno je najkorišteniji engine za izradu video igara. Može se koristiti za izradu 2D i 3D igara za trenutno 21 platformu : računala, konzole i mobilni uređaji. Unity sučelje sastoji se od prozora koji služe za obavljanje određenih poslova u izradi igre. Osnovni prozor je scena u kojemu se postavljaju i namještaju objekti te se stvara igra. Scena prikazuje ono što vidimo kada pokrenemo igru te se igra uglavnom sastoji od više scena (npr. za svaki nivo jedna scena). Objekti su osnovne cjeline od kojih se izrađuje igra te ih možemo vidjeti u prozoru hijerarhija. Na objekte možemo pridružiti različite grafike, zvukove, skripte te ponašanja. Odabirom objekta se otvara prozor inspektor u kojemu možemo manipulirati određenim veličinama objekta što se odražava na samu igru. Npr.: položaj, masa, svjetlost, odnos s ostalim objektima i sl. U prozoru projekt možemo uvoziti vlastite grafike, zvukove, skripte i druge materijale od kojih kasnije stvaramo objekte. Posljednji prozor koji je korišten u ovom projektu je prozor animator, a služi za upravljanje animacijama. Kodovi koji služe za upravljanje i mehaniku igre pišu se u skripte koje se onda pridružuju objektima ili im se pristupa pomoću drugih skripti. Skripte u Unityu se pišu u programskim jezicima C#, JavaScript i Boo. U ovom projektu korišten je C# programski jezik. Skripte imaju dvije osnovne funkcije: Start() koja se izvodi pri pokretanju skripte i Update() koja se izvodi jednom po frame-u.

3. RAZVOJ IGRE

3.1. Ideja

Ideja projekta je bila napraviti 2D računalnu igru sa elementima klasičnih 2D platformera. Kao inspiracija su uzeti Super Mario Bros. te Volgarr The Viking. Glavni lik je bezimeni plavi medvjed koji se našao u šumi gdje nije prihvaćen od ostatka šumske populacije ponajviše od strane „normalnih“ smeđih medvjeda koji ga se žele riješiti. Plavog medvjeda proganja tuga dok u jednom trenutku ne preuzima stvar u svoje ruke i kreće u borbu protiv ostatka šume u pokušaju da napusti negativno nastrojenu okolinu. U igri je omogućeno kretanje lijevo desno te gore dolje pomoću opcije skakanja po platformama različitih veličina koje se nalaze na različitim visinama i područjima nivoa. Igrač je u mogućnosti koristiti opcije za kretanje lijevo-desno, opciju za skok i udarac kandžom kojima šteti protivnicima te opciju saginjanja radi izbjegavanja napada u kojoj može i puzati. U igri postoje dvije vrste neprijatelja: smeđi medvjed te vjeverica. Smeđi medvjed se kreće lijevo-desno dok ne ugleda igrača te također može napasti kandžom kako bi oštetio igrača. Vjeverica je stacionaran lik koji napada na udaljenost, kada ugleda igrača gađa ga žirevima te mu nanosi štetu. Igrač i neprijatelji imaju određen broj životnih bodova koji kada dosegnu nulu igrač odnosno neprijatelj umiru. Igračevi životi su prikazani u gornjem lijevom kutu te ako dosegnu nulu igra kreće ispočetka. Protivnici imaju mogućnosti bacanja borovica koje igrač može pokupiti te mu vraćaju po jedan životni bod. Svi napadi.: igračeva kandža, kandža smeđeg medvjeda i vjeveričin žir su jednako snažni te oduzimaju jedan životni bod. Igrač i protivnički medvjed imaju po 3 životna boda dok vjeverica ima 2. U igri je velik dio vremena bio usmjeren na izradu grafike i animacija te tako za sve likove postoji broj različitih animacija za različite funkcije koje se odvijaju. Igra se sastoji od dvije scene. Prva scena je glavni izbornik iz koje se može u nivo. Sam nivo je druga scena. Pomoću izrađenih objekata u igri možemo slagati nove scene i raditi nove nivoe što ne donosi nove funkcionalnosti te je zbog toga u prvoj verziji rada izrađen samo jedan.

3.2. Kamera

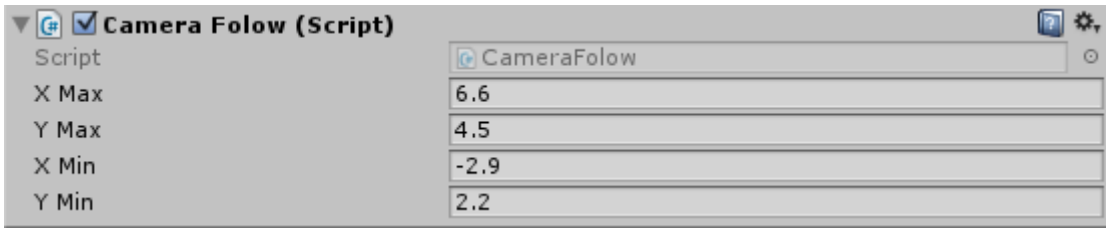
Kamera je objekt koji je automatski stvoren pravljenjem nove scene igre. Služi kao uređaj kojime igrač vidi svijet. Kameri možemo pomoću inspektora namještati poziciju i veličinu odnosno promjenom tih atributa možemo povećavati i smanjivati

vidno polje igraća. Kamera u ovoj igri ima bitnu ulogu jer mora pratiti igrača u kojem god smjeru se on kretao te se u određenim položajima treba samostalno zaustaviti kako ne bi izašla van granica scene. Zbog toga dodajemo na kameru skriptu CameraFollow.

```
public class CameraFolow : MonoBehaviour {


    [SerializeField]
    private float xMax;
    [SerializeField]
    private float yMax;
    [SerializeField]
    private float xMin;
    [SerializeField]
    private float yMin;

    private Transform target;

    // Use this for initialization
    0 references
    void Start () {

        Camera.main.aspect = 16f / 9f;

        target = GameObject.Find("Player").transform;


    }

    // Update is called once per frame
    0 references
    void LateUpdate ()
    {
        transform.position = new Vector3(Mathf.Clamp(target.position.x,xMin,xMax),
                    Mathf.Clamp(target.position.y,yMin,yMax), transform.position.z);

    }

}
```

Sl. 3.1. Skripta CameraFollow

U funkciji Start() kamera traži objekt pod nazivom „Player“ te ako postoji počinje ga pratiti. Polja xMax, xMin, yMax i yMin služe kao granice do kojih kamera može ići prije zaustavljanja. Pošto su privatna polja dodaje im se opcija SerializeField kako bi se te granice lakše mogli postaviti u inspektoru. Osim tih funkcija imamo i liniju koda koja nam ograničava kameru na aspekt 16:9 kako pri pokretanu igre na monitorima različitih rezolucija ne bi došlo do ne željenog izlaženja kamere iz granica scene i mijenjanja veličine UI elemenata.



Sl. 3.2. Prikaz namještanja granica kamere u inspektoru

CameraFollow skripta se koristi samo u sceni nivoa.

3.3. Likovi

U igri postoje tri vrste likova: igrač te dva protivnika. Igrač i protivnici imaju neke različite funkcije ali i dosta sličnih te zato imamo jednu glavnu skriptu *character* koja sadrži zajedničke funkcije igrača i protivnika i koju nasljeđuju skripte *player* i *enemy* u kojima su definirane zasebne funkcije pojedinog lika. U *character* neka od zadanih varijabli su brzina kretanja, smjer gledanja, životni bodovi, izvori nanošenja štete i sl. Također su određene neke funkcije koje su zajedničke svim likovima. Dio funkcija likova je pokretan pomoću animatora o kojem ćemo pričati u nastavku.

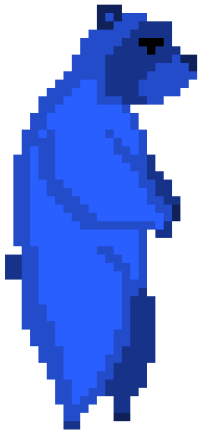
```
public void ChangeDirection()
{
    facingRight = !facingRight;
    transform.localScale = new Vector3
                (transform.localScale.x * -1, transform.localScale.y* 1, 1);
}

0 references
public void MeleeAttack()
{
    ClawCollider.enabled = true;
}


2 references
public virtual void OnTriggerEnter2D(Collider2D other)
{
    if (damageSources.Contains(other.tag))
    {
        StartCoroutine(TakeDamage());
    }
}
```

Sl. 3.3. Funkcije ChangeDirection, MeleeAttack i funkcija za primanje štete u skripti character

Funkcija ChangeDirection provjerava u koju stranu je lik okrenut te ga po potrebi okreče u drugu stranu.

3.3.1. Igrač (Player)

Player je lik kojim upravlja igrač i predstavljen je plavim medvjedom. Medvjed ima mogućnost kretanja lijevo desno te skoka i udarca kandžom.



Sl. 3.4. Sprite Playera (plavog medvjeda)

Igračev lik ima više različitih animacija. Sve animacije nalaze se u prilozima. Na objekt Player dodana je skripta *player* koja nasljeđuje skriptu *character*.

```
private void HandleMovement(float horizontal)
{
    if (MyRigidbody.velocity.y <0)
    {
      MyAnimator.SetBool("land", true);
    }
    if (!Attack && (OnGround || airControl) )
    {
      MyRigidbody.velocity = new Vector2 (horizontal*movementSpeed, MyRigidbody.velocity.y);
    }
    if (Jump && MyRigidbody.velocity.y == 0)
    {
     MyRigidbody.AddForce(new Vector2(0, jumpForce));
    }


    MyAnimator.SetFloat("speed",Mathf.Abs(horizontal));
}
```

Sl. 3.5. Funkcija HandleMovement kretanja Playera

Funkcija HandleMovement se bavi kretanjem lika. Provjerava napada li igrač i nalazi li se na zemlji te mu ako ne napada pridodaje brzinom kojom se kreće u strane. Također ako želi skočiti pridodaje mu silu skoka da odredi koliko će skočiti.

```
private bool IsGrounded()
{
    if (MyRigidbody.velocity.y <= 0)
    {
        foreach (Transform point in groundPoints)
        {
            Collider2D[] colliders = Physics2D.OverlapCircleAll(point.position, groundRadius, whatIsGround);

            for (int i = 0; i < colliders.Length; i++)
            {
                if (colliders[i].gameObject != gameObject)
                {
                    return true;
                }
            }
        }

    }
        return false;
```

Sl. 3.6. Funkcija IsGrounded

Objektu Player su pridodani prazni objekti groundPoints u njegovo podnožje te funkcija IsGrounded provjerava koliko su groudPointovi udaljeni od zemlje odnosno površine na kojoj se igrač kreće. Ta funkcija nam omogućava da provjerimo kada je lik na zemlji kako ne bi smo mogli u zraku nastaviti beskonačno skakati.

```
private IEnumerator IndicateImmortal()
{
    while (immortal)
    {
        spriteRenderer.enabled = false;
        yield return new WaitForSeconds(.1f);
        spriteRenderer.enabled = true;
        yield return new WaitForSeconds(.1f);
    }
}
3 references
public override IEnumerator TakeDamage()
{
    if (!immortal)
    {
        healthStat.CurrValue -= 1;

        if (!IsDead)
        {
            MyAnimator.SetTrigger("dmg");

            immortal = true;

            StartCoroutine(IndicateImmortal());

            yield return new WaitForSeconds(immortalTime);

            immortal = false;
        }
        else
        {
            MyAnimator.SetLayerWeight(1, 0);
            MyAnimator.SetTrigger("death");
        }
    }
}
```

Sl. 3.7. Funkcije IndicateImmortal i TakeDamage

Funkcija TakeDamage() služi kako bi lik primio štetu od protivnika te mu se životni bodovi smanjili za 1. Nakon što primi štetu lik postaje besmrtan te poziva funkciju IndicateImmortal() kako bi se ukazalo na to da je oštećen i da se spriječi daljnje nanošenje štete u zadanom vremenu. Kako bi omogućili da se te funkcije izvode usporedno s drugim funkcijama te kako bi mogli koristiti opciju WaitForSeconds, moramo koristiti interface IEnumerator te funkcije pozivati kao korutine. TakeDamage() funkcija također ukazuje na smrt lika. Sličnu funkciju kao TakeDamage() koristi i protivnik.

3.3.2. Protivnici (Enemy)

U igri postoje dvije vrste protivnika: smeđi medvjed (enemy1) i vjeverica (enemy2). Oni se razlikuju, osim po izgledu, po tome što je medvjed pokretan i udara na blisku udaljenost, a vjeverica je stacionarna i gađa na daljinu. Kako bi ih u skripti razlikovali pridodali smo bool melee te medvjedu stavili „1“, a vjeverici „0“.

Sl. 3.8. Sprite za enemy1 i enemy2

Kao i igračev lik protivnici imaju više različitih animacija. Na objekte Enemy1 i Enemy2 dodana je skripta *player* koja nasljeđuje skriptu *character*.

```
public bool InMeleeRange
{

    get
    {
        if (Target != null)
        {
            return Vector2.Distance(transform.position, Target.transform.position) <= meleeRange;
        }

        return false;
    }
}

1 reference
public bool InThrowRange
{

    get
    {
        if (Target != null)
        {
            return Vector2.Distance(transform.position, Target.transform.position) <= throwRange;
        }

        return false;
    }
}
```

Sl. 3.9. Funkcije InMeleeRange i InThrowRange

Funkcije InMeleeRange i InThrowRange služe kako bi provjerile dali je igrač na dovoljnoj udaljenosti kako bi ga protivnici mogli napasti. Vjeverica koristi funkciju InThrowRange a medvjed InMeleeRange.

```
1 reference
private void LookAtTarget()
{
    if (Target != null)
    {
        float xDir = Target.transform.position.x - transform.position.x;

        if (xDir < 0 && facingRight || xDir > 0 && !facingRight)
        {
            ChangeDirection();
        }
    }
}
```

Sl. 3.10. Funkcija LookAtTarget

Funkcija LookAtTarget služi kako bi protivnik uvijek pratio gdje se nalazi igrač ako mu je u vidokrugu odnosno provjerava da li je igrač prošao iza protivnika te na temelju pozicije tih dvaju likova protivnik se po potrebi okreće. Ova funkcija sprječava da protivnik izađe iz borbe ako mu igrač ode iza leđa.

```
public void ThrowNut(int value)
{
    if (facingRight)
    {
        GameObject tmp = (GameObject)Instantiate(nutPrefab,
            nutPos.position, Quaternion.Euler(new Vector3(0,0,-90)));
        tmp.GetComponent<Nut>().Initialize(Vector2.right);
    }
    else
    {
        GameObject tmp = (GameObject)Instantiate(nutPrefab,
            nutPos.position, Quaternion.Euler(new Vector3(0, 0, 90)));
        tmp.GetComponent<Nut>().Initialize(Vector2.left);
    }
}
```

Sl. 3.11. Funkcija ThrowNut

Funkcija ThrowNut koristi enemy2 (vjeverica) a služi kako bi se žir koji baca kretao u onom smjeru u kojem vjeverica gleda te okretao na tu stranu. Osim skripte *enemy* protivnici za protivnike je napravljen interface odnosno sučelje IenemyState koje se bavi stanjima u kojima se oni nalaze.[4] Ta stanja su: IdleState, PatrolState, MeleeState i RangedState odnosno stanje mirovanja, patrole, bliskog sukoba i sukoba na daljinu. Interfaceu se pristupa sa skripte *enemy* te se ove četiri skripte stanja izmjenjuju po potrebi.

```
public void Execute()
{
    Idle();

    if (enemy.Target != null && enemy.melee)
    {
        enemy.ChangeState(new PatrolState());
    }
    else if (enemy.Target != null && !enemy.melee)
    {
        enemy.ChangeState(new RangedState());
    }

}

5 references
public void Enter(enemy enemy)
{
    idleDuration = UnityEngine.Random.Range(2, 5);
    this.enemy = enemy;
}
1 reference
private void Idle()
{
    enemy.MyAnimator.SetFloat("speed", 0);

    idleTimer += Time.deltaTime;

    if (idleTimer >= idleDuration && enemy.melee)
    {
        enemy.ChangeState(new PatrolState());
    }
    else if (idleTimer >= idleDuration && !enemy.melee)
    {
        enemy.ChangeDirection();
        enemy.ChangeState(new PatrolState());
    }
}
```

Sl. 3.12. IdleState stanje

U stanju mirovanja protivnici se ne kreću i igra njihova animacija mirovanja. Pri ulasku u ovo stanje pridodaje se varijabli idleDuration nasumičan broj u zadanom intervalu te taj broj određuje koliko dugo protivnik se zadržava u ovom stanju. Nakon završetka tog vremena protivnik se prebacuje u stanje patroliranja ukoliko ne ugleda igrača. U tom slučaju protivnik ako je medvjed (melee bool mu je postavljen u „1“) i dalje odlazi u stanje patrole i potjeri za igračem, a ako je vjeverica (melee = „0“) ulazi u stanje sukoba na daljinu (RangedState).

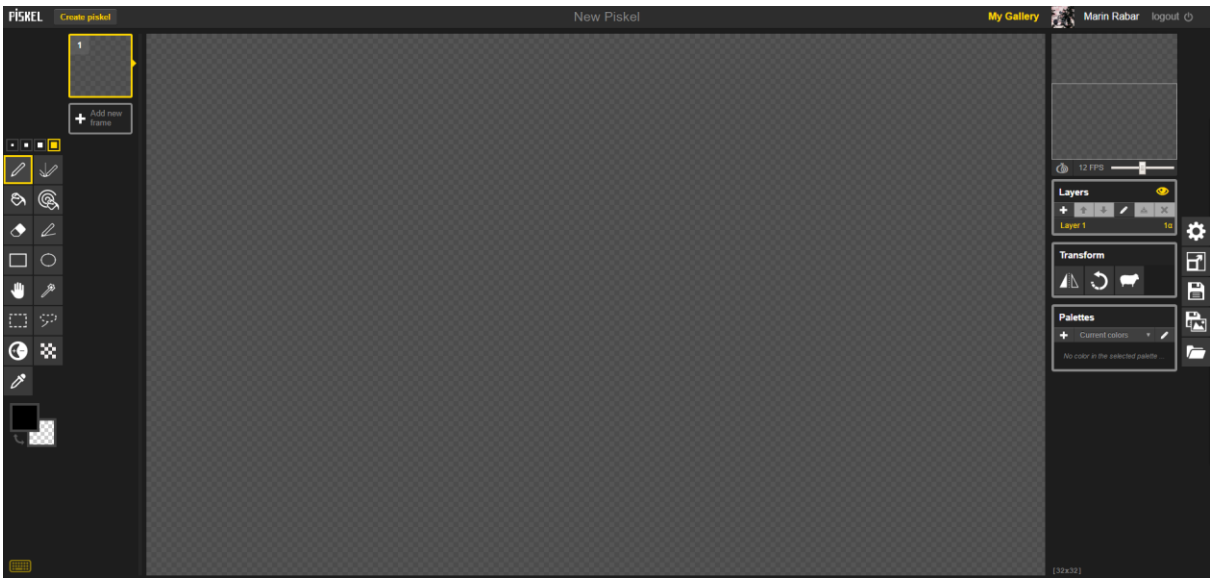
```
public void Execute()
{
    Patrol();

    enemy.Move();

    if (enemy.Target != null && enemy.InMeleeRange && enemy.melee)
    {
        enemy.ChangeState(new MeleeState());
    }

    else if (enemy.Target != null && enemy.InThrowRange && !enemy.melee)
    {
        enemy.ChangeState(new RangedState());
    }
}

5 references
public void Enter(enemy enemy)
{
    patrolDuration = UnityEngine.Random.Range(5,10);
    this.enemy = enemy;
}
1 reference
private void Patrol()
{

    patrolTimer += Time.deltaTime;

    if (patrolTimer >= patrolDuration)
    {
        enemy.ChangeState(new IdleState());
    }
}
```

Sl. 3.13. PatrolState stanje

U stanju patrole protivnik dobiva također nasumičan broj koji određuje koliko će se zadržati u tom stanju. Ovo stanje govori protivniku da se kreće do svojih zadanih granica, a slučaju vjeverice samo okreće lika. Nakon vremenskog intervala vraća se u stanje mirovanja ako ne ugleda u međuvremenu igrača. U tom slučaju enemy1 (medvjed) ulazi u stanje bliske borbe (MeleeState) a enemy2 (vjeverica) u stanje borbe na daljinu (RangedState). MeleeState i RangedState pokreću napade protivnika te postavljaju protivnicima vremensko ograničenje na napade kako bi morali pričekati zadani interval prije novog napada.

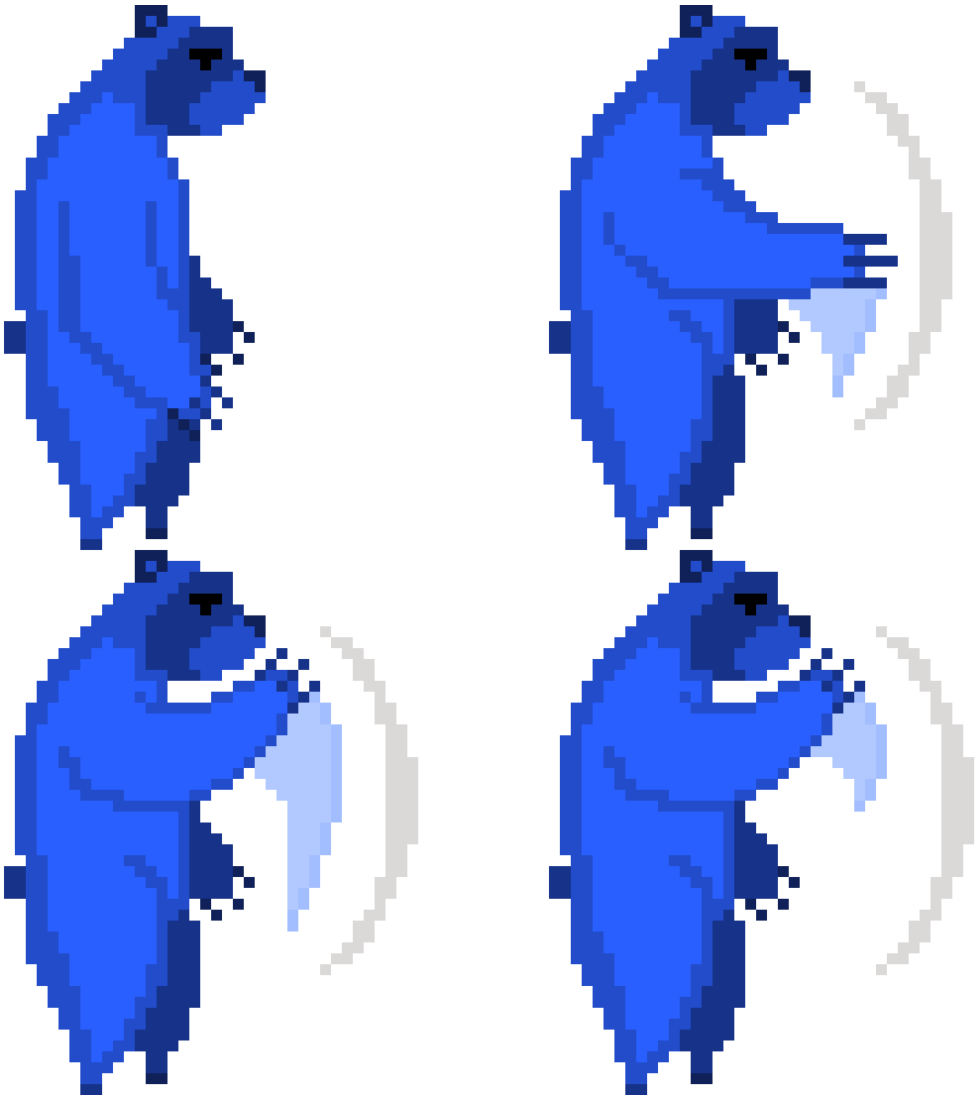
3.4. Grafika

Sva grafika igre je vlastoručno rađena u web aplikaciji Piskel (http://www.piskelapp.com). Piskel je aplikacija za izradu 2D pikselizirane grafike. Zbog jednostavnosti korištenja odličan je alat za jednostavan *pixelart* te za manje projekte poput ovog završnog rada. Aplikacija nam omogućuje određivanje dimenzija na kojima želimo raditi što se odražava na veličinu „piksela“, rad na različitim layerima te razne druge alate. Također nam dozvoljana izvoženje spriteova u obliku spritesheeta bez potrebe za nekim drugim alatima.



Sl. 3.14. Sučelje Piskel aplikacije

Aplikacija je korištena za izradu spriteova (2D grafički objekt) koji naizmjeničnim prikazivanjem čine animaciju.



Sl. 3.15. Spritesheet za animaciju napada plavog medvjeda

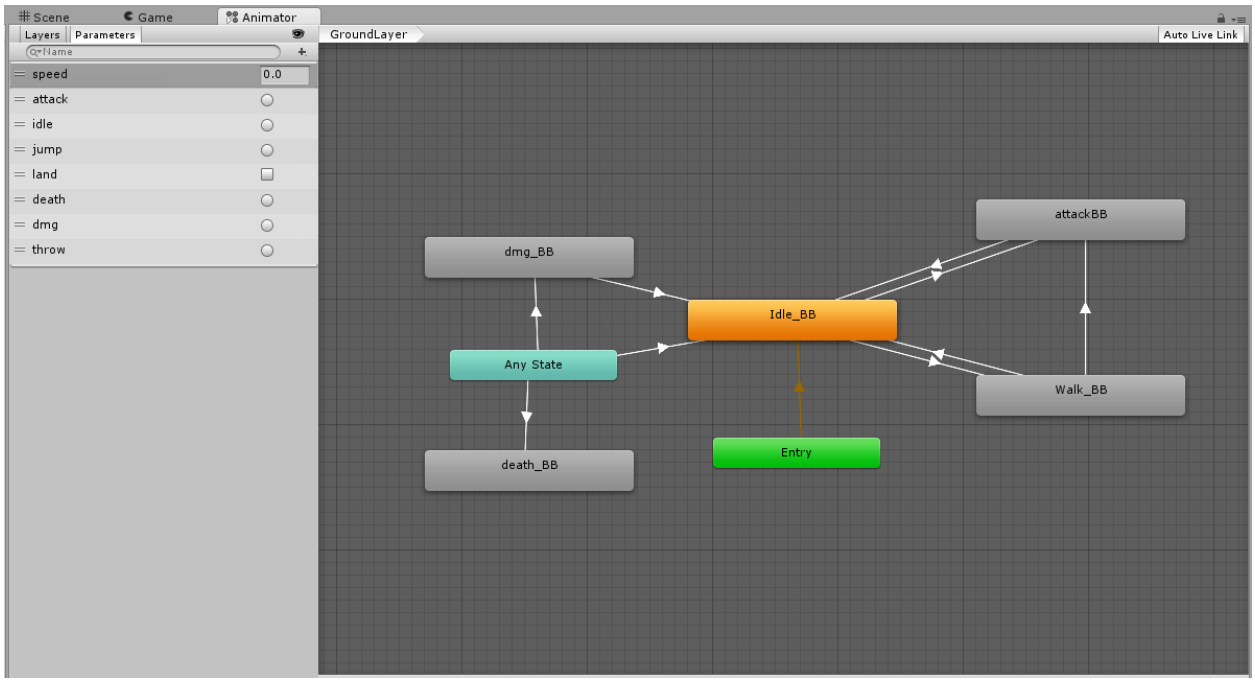
Svaka animacija se sastoji od određenog broja spriteova koji se po željenoj brzini izmjenjuju. U ovom projektu sve animacije se sastoje od 4 do 12 spriteova. Nakon uvoženja spriteova u Unity, Unity nam omogućava da spritesheet pretvorimo u animaciju te stvaranjem animacije se stvara i kontroler za upravljanje animacijama. Postavke kontrolera animacija podešavamo u prozoru animator.

3.5. Animator

Animator je „stroj stanja“ koji određuje koje animacije se trenutno odvijaju te stvara blag prijelaz između animacija.[5]

Svaki objekt koji ima animaciju ima kontroler koji upravlja tom animacijom te ako ima više animacija upravlja prijelazom iz animacije u animaciju.

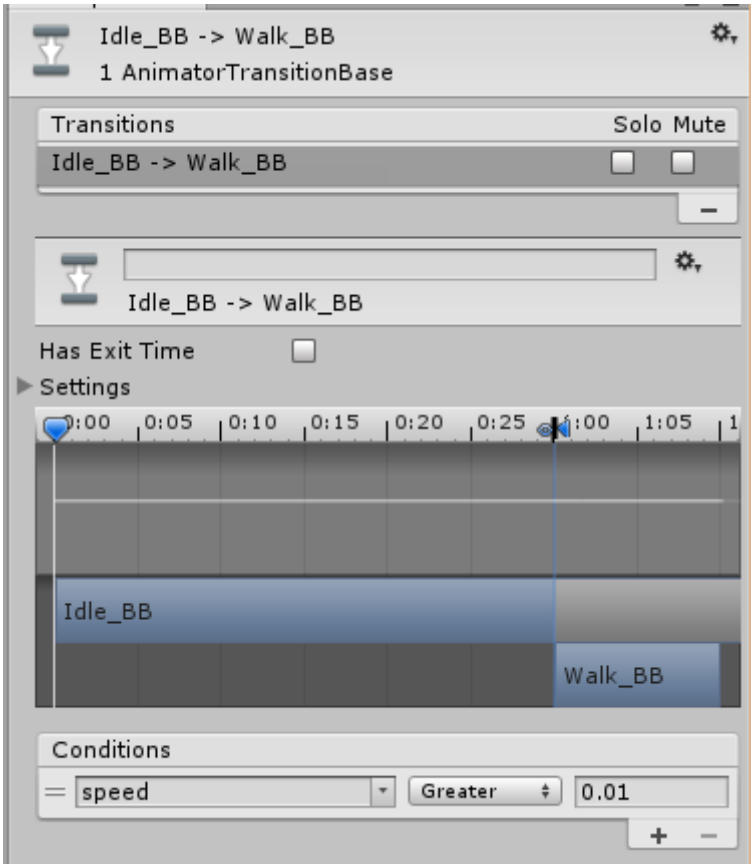
U ovom projektu velik broj funkcionalnosti je učinjen pomoću animatora.



Sl. 3.16. Kontroler lika Player

Na slici 3.16. je prikazano sučelje prozora animator. U glavnom dijelu sučelja vidimo stablo koje se sastoji od različitih animacija objekta čiji je kontroler, u slučaju na slici to je kontroler od objekta Player. Narančasta kućica predstavlja početno stanje u kojemu će se objekt naći što je ovdje stanje mirovanja. Sva stanja su povezana strelicama koje naznačuju koja animacija se može pokrenuti poslije trenutnog stanja. Iz stanja mirovanja možemo ući u stanje hodanja i stanje napadanja te također iz njih nazad u stanje mirovanja. Plava kućica (Any State) prikazuje u koja stanja objekt može ući iz bilo kojeg drugog stanja. Npr.: ako Player trenutno hoda i izvodi se animacija *Walk_BB*, ako ga neko udari i nanese mu štetu kućica Any State prekida animaciju hodanja i prebacuje objekt u animaciju primanja štete (*dmg_BB*). Kako bi se prijelaz iz stanja u stanje odvio moraju se zadovoljiti neki kriteriji. Na lijevoj strani prozora nalazi se odjeljak Parameters

(parametri). Tu dodajemo prametre koji mogu biti tipova: int, float, bool i trigger te ih pridodajemo na strelice između stanja.



Sl. 3.17. Inspektor prijelaza iz mirovanja u hodanje

Na slici 3.17. vidimo da možemo dodavati uvjete da dođe do neke promjene između stanja. U našem primjeru to je uvjet da varijabla speed tipa float bude veća od 0.01. U tom slučaju pokreče se animacija hodanja. Na slici možemo vidjeti i neke druge opcije vezane za prijelaz između stanja. Upute o realizaciji uvjeta definirani su u skriptama.

```
private void HandleInput()
{
    if (Input.GetKeyDown(KeyCode.W))
    {
        MyAnimator.SetTrigger("jump");
    }
    if(Input.GetKeyDown(KeyCode.Space))
    {
        MyAnimator.SetTrigger("attack");
    }
}
```

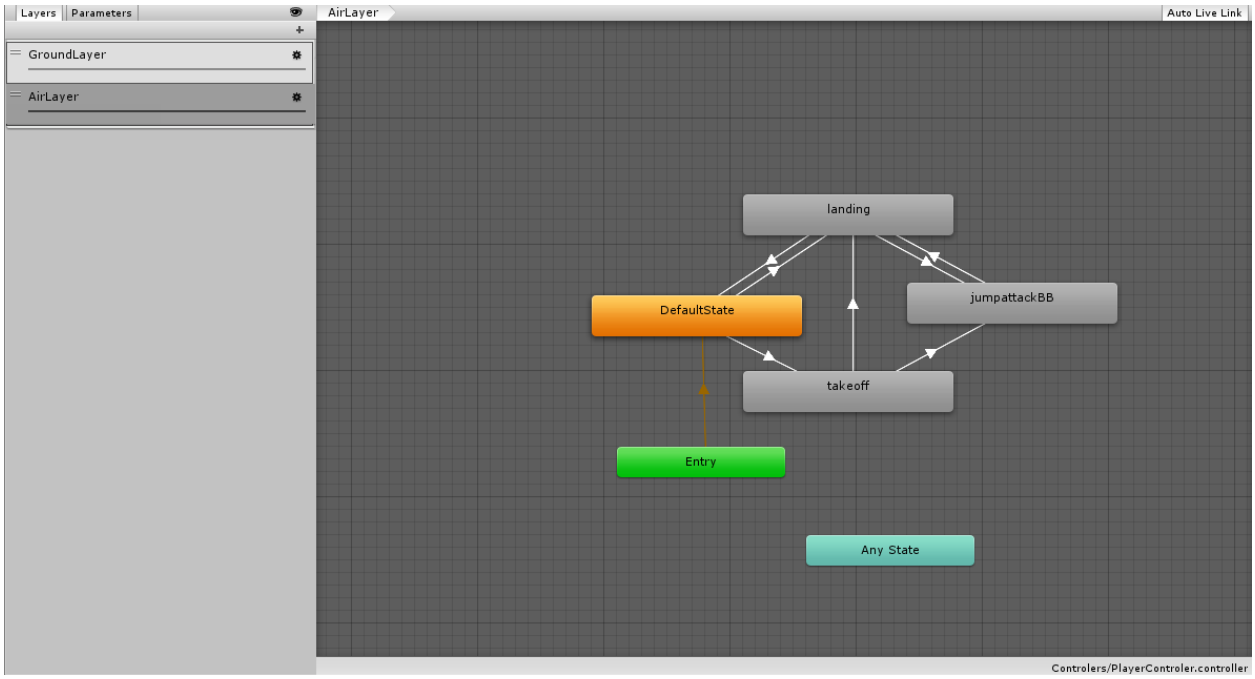
Sl. 3.18. Funkcija HandleInput skripte player

Funkcija HandleInput nalazi se u skripti *player* koju smo ranije spominjali. Ovdje vidimo kako funkcija upravlja uvjetima za promjenu stanja animatora. Pritiskom tipke *W* trigger „jump“ u animatoru se postavlja na „1“ te se pokreče animacija skakanja. Osim regularnih skripti na stanja animatora postavljamo skripte „ponašanja“ koje također upravljaju uvjetima kontrolera i cjelokupnim mehanikama igre.

```
public class AttackBehaviour : StateMachineBehaviour {

    // OnStateEnter is called when a transition starts and the state machine starts to evaluate this state
    0 references
    override public void OnStateEnter(Animator animator, AnimatorStateInfo stateInfo, int layerIndex)
    {
        animator.GetComponent<character>().Attack = true;

        animator.SetFloat("speed", 0);

        if (animator.tag == "Player")
        {
            if (player.Instance.OnGround)
            {
                player.Instance.MyRigidbody.velocity = Vector2.zero;
            }

        }
    }
    // OnStateExit is called when a transition ends and the state machine finishes evaluating this state
    0 references
    override public void OnStateExit(Animator animator, AnimatorStateInfo stateInfo, int layerIndex)
    {
        animator.GetComponent<character>().Attack = false;
        animator.GetComponent<character>().ClawCollider.enabled = false;
        animator.ResetTrigger("attack");
        animator.ResetTrigger("throw");
    }
```

Sl. 3.19. Skripta AttackBehaviour

Skripta *AttackBehaviour* opisuje ponašanje animacije, a time i objekta dok napadaju. Ona je pridružena svim kontrolerima na stanje *attack*. Funkcija OnStateEnter daje uputstva ponašanja pri ulazu u stanje *attack*. Govori objektu da udara te pokreće time funkcije za nanošenje štete ako je potrebno i sl. Postavlja uvjet „speed“ na 0 te time sprječava likove da se kreću dok napadaju. Funkcija OnStateExit daje uputstva za ponašanje pri izlasku iz stanja *attack*. Ona prekida napadanje objekta te resetira uvijete „attack“ i „throw“ kako bi se mogli reaktivirati pri potrebi ponovnog ulaženja u ovo stanje. Na slici 3.16 još možemo vidjeti karticu Layers. Nju koristimo ako želimo da nam animator ima više slojeva, primjerice sloj za stanje kada je objekt na zemlji i sloj za kada je objekt u zraku kao u primjeru Playera.



Sl. 3.20. Air Layer kontrolera Player

Na slici vidimo prikaz AirLayer Playerovog animatora. Ono se bavi kontroliranjem animacija kada se objekt nalazi u zraku.

```
private void HandleLayers()
{
    if (!OnGround)
    {
        MyAnimator.SetLayerWeight(1, 1);
    }
    else
    {
        MyAnimator.SetLayerWeight(1, 0);
    }
}
```

Sl. 3.21. Funkcija HandleLayers skripte player

Za izmjenu layera na animatoru potrebna nam je funkcija HandleLayers koja postavlja težinu layera te time odlučuje koji se layer koristi. Ako objekt nije na zemlji težina layera se postavlja na (1,1) te se koristi AirLayer. U suprotnom se težina postavlja na (1,0) te se koristi GroundLayer.

3.6. Grafičko korisničko sučelje (GUI)

Korisničko sučelje je premosnica između korisnika i programa. Ono igraču ukazuje na stanje igre te mu omogućuje neke opcije unutar igre.

3.6.1. Glavni izbornik

Glavni izbornik je početna scena koju igrač vidi ulaskom u igru. Sastoji se od pozadine, imena igre, animiranog logoa te dva gumba: Start i Exit. Pritiskom na tipku exit otvara se novi prozor za potvrdu, s dodatna dva gumba, koji je također dio scene. Na glavnom izborniku se nalazi i pozadinska glazba.

Sl. 3.22. Glavni izbornik

Na slici vidimo sve nabrojane sastavnice glavnog izbornika. Gumbi i tekst su posebni objekti koji se postavljaju kao potomci objekta canvas koji služi za prikaz korisničkog sučelja. Gumbi su text kojima je pridodan atribut Button te time imaju tu funkcionalnost. Može im se mijenjati veličina izgled. boja i sl.

```
public void StartLevel()
{
    source.Stop();
    source.clip = clipone;
    source.Play();
    animator.SetTrigger("start");
    StartCoroutine("Wait");
}

0 references
public void ExitGame()
{

    Application.Quit();
}

0 references
IEnumerator Wait()
{
    yield return new WaitForSeconds(1.5f);

    SceneManager.LoadScene("Scene1");
}
```

Sl. 3.23. Dio skripte MenuScript

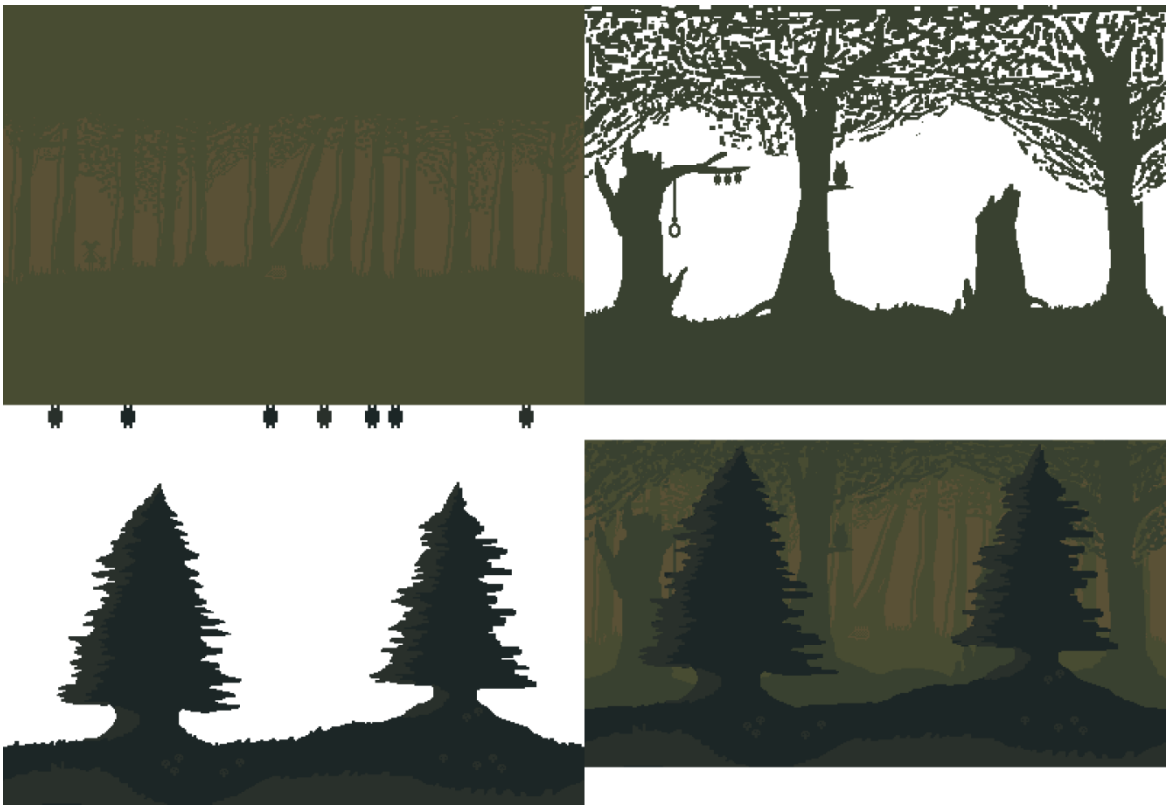
MenuScript skripta dodana je na canvas početne scene i služi za upravljanje nekim od objekata na sceni. Na slici vidimo funkcije StartLevel i ExitGame koje upravljaju gumbima Start i Exit. Pritiskom gumba Exit i potvrdom, aplikacija se gasi dok pritiskom gumba Start se zaustavlja glazba u pozadini, mijenja se audio clip te pušta novi zvuk, pokreče se animacija logoa te se pokreće korutina koja čeka nekoliko sekundi da se animacija dovrši te nakon toga pokreće igru.

3.6.2. Nivo

Nivo je druga scena igre kojoj igrač pristupa pomoću glavnog izbornika. U ovoj je sceni postavljen glavni lik, svi protivnici i duge prepreke te se ovdje odvija radnja.

Sl. 3.24. Nivo

Nivo je napravljen tako da su postavljene platforme kojima je dodana opcija *collider* kako bi likovi mogli stajati na njima. Neke platforme služe kao tlo dok neke lebde u zraku i treba na njih skočiti. Na platforme su postavljeni glavni lik i protivnici. Iza platformi i likova se nalazi pozadina koja služi za stvaranje ambijenta. Pozadina se sastoji od tri *layera* koji se pri kretanju lika kreću različitim brzinama te time stvaranju dojam dubine.



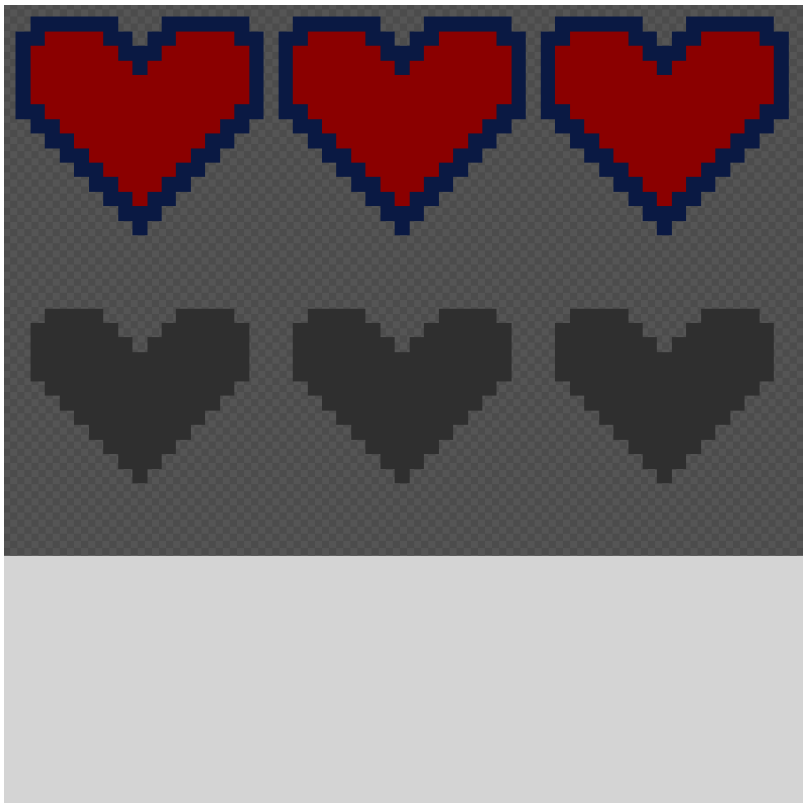
Sl. 3.25. Pozadina nivoa po layerima

3.6.3. Sučelje unutar nivoa

Unutar scene nivoa na sučelju imamo prikazano stanje trenutnih životnih bodova u obliku srca u gornjem lijevom kutu.

Sl. 3.26. Prikaz životnih bodova

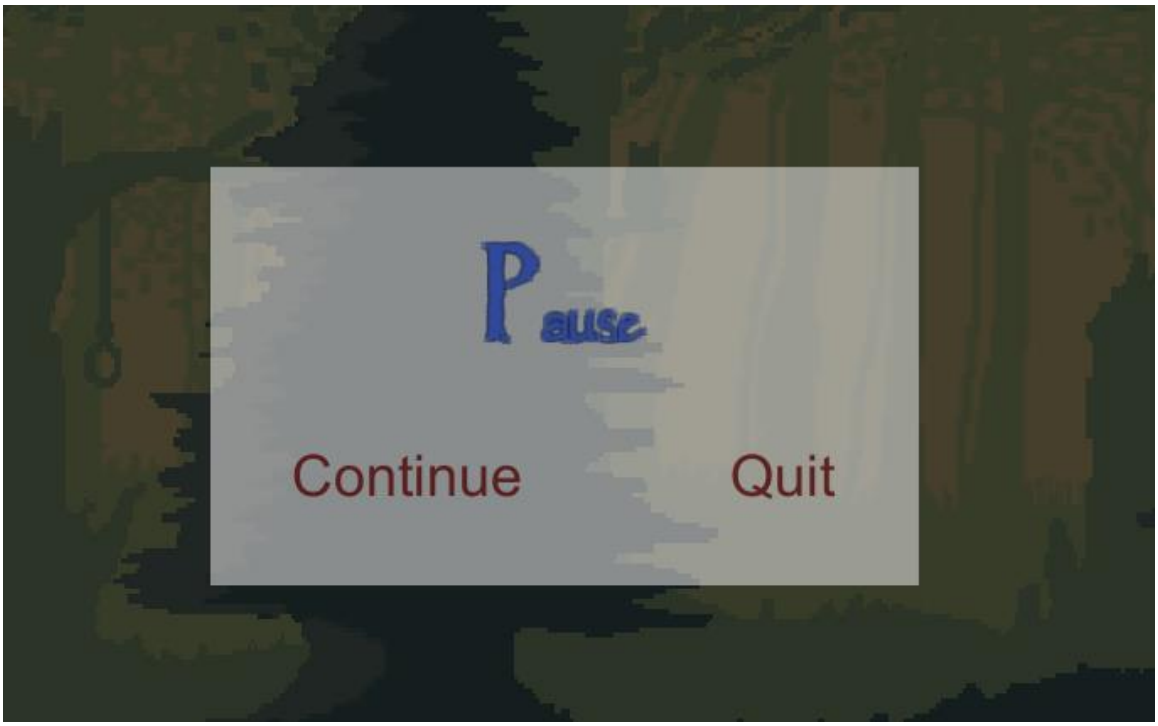
Plava srca prikazuju koliko nam je životnih bodova preostalo.



Sl. 3.27. Spritesheet životnih bodova

Kako igrač ima 3 životna boda u početku počinje sa sva 3. Svaki put kad mu protivnik nanese štetu jedno srce nestane. Na slici 3.25. imamo tri reda. Prvi red (crvena srca) je pozadina srca i služi kao temelj za grafički element. Kao njegovog potomka postavljamo siva srca iz drugog reda i postavljamo ih da se poklapaju s crvenim srcima. Sivim srcima dodajemo atribut maska koji onemogućuje svim potomcima tog objekta da se vide izvan granica objekta. Kao potomka tom objektu stavljamo bijeli pravokutnik kojega u Unity mijenjamo u plavu boju te su stoga srca u igri plava. Pomoću funkcije upravljamo da za svaku nanesenu štetu na Playeru se pravokutnik pomiče za 1/3 u lijevo te tako dobivamo efekt gubljenja životnih bodova.

Unutar nivoa također imamo opciju pauze stiskanjem tipke „esc“ pri čemu se igra zaustavlja te se otvara izbornik koji nam omogućuje da nastavimo igru ili izađemo iz nje.



Sl. 3.28. Izbornik pauze

Igra se može također nastaviti ponovnim pritiskom tipke „esc“ što je izvedeno funkcijom pause.

```
public void pause()
{
    if (Input.GetKeyDown(KeyCode.Escape))
    {
        if (canvas.gameObject.activeInHierarchy == false)
        {
            canvas.gameObject.SetActive(true);
            Time.timeScale = 0;
            sound1.Pause();


        }
        else
        {
            canvas.gameObject.SetActive(false);
            Time.timeScale = 1;
            sound1.UnPause();

        }
    }
}
```

Sl. 3.29. Funkcija pause

4. ZAKLJUČAK

U ovom radu izrađen je 2D platformer u Unity Engineu uz pomoć C# programskog jezika. Kodovi pisani u C# jeziku se nalaze u skriptama. Svi spriteovi su načinjeni pomoću web aplikacije Piskel te nakon toga pretvoreni u animacije. Skripte, spriteovi i animacije se dodaju na objekte unutar igre, kao i neki drugi elementi primjerice zvuk. Cilj igre je tuga plavog medvjeda zbog neprihvaćenosti od ostatka šumske populacije koja se pretvara u bijes i navodi ga na borbu i bijeg iz te okoline. Igra se bavi problemom diskriminacije u životinjskom svijetu na temelju fizičkog izgleda. Igrač upravlja plavog medvjeda koji ima mogućnost kretanja lijevo desno, skakanja, napada kandžom, saginjanja i puzanja. Kao prepreke mu nailaze protivnici: smeđi medvjedi i vjeverice. Smeđi medvjed je pokretljiv i ulazi u blisku borbu dok je vjeverica nepokretna i gađa na daljinu. Za postavljanje animacija u igru i ponašanje vezani uz animacije korišten je animator koji omogučava kontroliranje animacija u ovisnosti o zadovoljenim uvjetima. Igra se sastoji od dvije scene. Prva scena je glavni izbornik koja služi kao uvod igrača u igru na kojoj možemo ući u nivo ili izaći iz igre. Pomoću nje su prikazane neke mogućnosti Unity Engine-a. Druga scena je sam nivo u kojemu igrač upravlja lika i te pokušava ostvariti zadani cilj. Temelj igre je uspostavljen te se u igru mogu bez većih poteškoća dodavati novi nivoi, neprijatelji i sl.

LITERATURA

[1] Royalty free music - 20. 6. 2016.

http://www.newgrounds.com/audio

[2] Royalty free fonts - 20. 6. 2016.

http://www.1001fonts.com/free-fonts-for-commercial-use.html

[3] Unity Editor - 16.06.2016.

https://unity3d.com/unity/editor

[4] Interfaces - 16.06.2016.

https://unity3d.com/learn/tutorials/topics/scripting/interfaces

[5] The Animator Controler - 18.06.2016.

https://unity3d.com/learn/tutorials/topics/animation/animator-controller.

SAŽETAK

Ovim završnim radom prikazana je izrada 2D platformske računalne igre za u Unity Engine-u. Bit igre je borba plavog medvjeda protiv negativno nastrojenih šumskih stanovnika te bijeg iz te okoline. Igra je rađena u Unity okruženju, a kodovi su pisani pomoću C# programskog jezika. Sva grafika je napravljena u Piskel web aplikaciji. U igri igrač kontrolira lika plavog medvjeda koji ima funkcije kretanja lijevo desno skoka i napada kandžom. Protivnici su mu pokretni smeđi medvjed koji ulazi u blisku borbu te stacionarna vjeverica koja gađa žirevima. Mehanike igre su pokretane skriptama i animatorom. Igra ima dvije scene. Prva scena je glavni izbornik koji je prvo što igrač vidi kada uđe u igru. Iz njega možemo pristupiti drugoj sceni koja je sam nivo igre. Ovime je načinjen temelj igre koji se može nadopunjavati.

Ključne riječi: 2D platformerska računalna igra, Unity, razvojno okruženje, C#, Piskel, skripta, animator, scena, plavi medvjed

ABSTRACT

Development of a 2D platformer video game

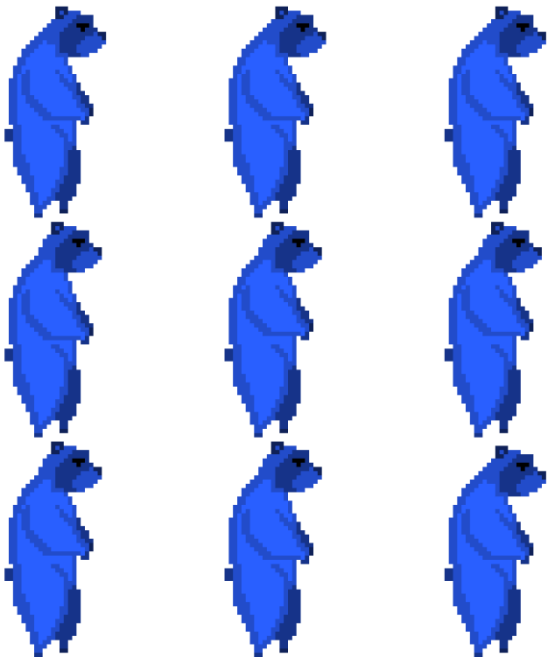
With this final paper the making of a 2D platformer video game is shown. The essence of the game is the fight of a blue bear against other, hostile forest residents and his escape from such an environment. The game is developed in Unity game engine and the codes are written in C# programming language. All graphics are made in the Piskel web application. In the game the player controls the blue bear who has the functions of moving left and right, jumping and attacking with the claw. His enemies are a mobile brown bear who engages in melee combat and a squirrel who attacks from range with nuts. All mechanics of the game are driven by scripts and the animator. The game consists of two scenes. The first one is the main menu which is the first thing the player sees. Through him the player can access the second scene which is the level itself. With this the core of the game is made which can be build upon.

Keywords: 2D platformer video game, Unity, game engine, C#, Piskel, script, animator, scene, blue bear

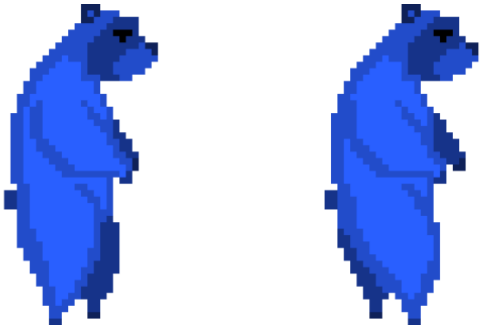
ŽIVOTOPIS

Marin Rabar rođen je 27.04.1994 u Osijeku. Od 2001. do 2009. pohađao je OŠ Frana Krste Frankopana. Nakon toga je upisao Gaudeamus, prvu privatnu srednju školu u Osijeku s pravom javnosti koju je završio 2013. godine. Iste je godine redovno upisao preddiplomski studij računarstva na Elektrotehničkom fakultetu u Osijeku koji i sada pohađa. Od malena je razvio ljubav prema video igrama te mu interesi leže u izradi video igara od programskog do grafičkog dijela. Poznaje rad na računalu, koristi se C# programskim jezikom te je upoznat s radom u Unity okruženju. Ima iskustva u Photoshopu i Piskelu. Od sportova se bavi skijanjem i vožnjom bicikla. Dugi je niz godina trenirao košarku i plivanje. Za osnovnu i srednju školu je sudjelovao na natjecanjima iz matematike, fizike i informatike te trčao kros. Govori engleski i njemački jezik. Od drugih se aktivnosti bavi volontiranjem. Do sada je volontirao na šest Pannonian Challenge-a na poziciji infrastrukture te u području IT sektora, na dva Grand prixa Osijek, Frame festivalu, Slami i sličnim događanjima. U ljeto 2012. je bio suorganizator prve biciklijade u Osijeku koja je promovirala broj biciklističkih staza i zdrav duh. Prisustvovao je na The Geek Gathering konferencijama u Osijeku 2013. i 2014. godine.

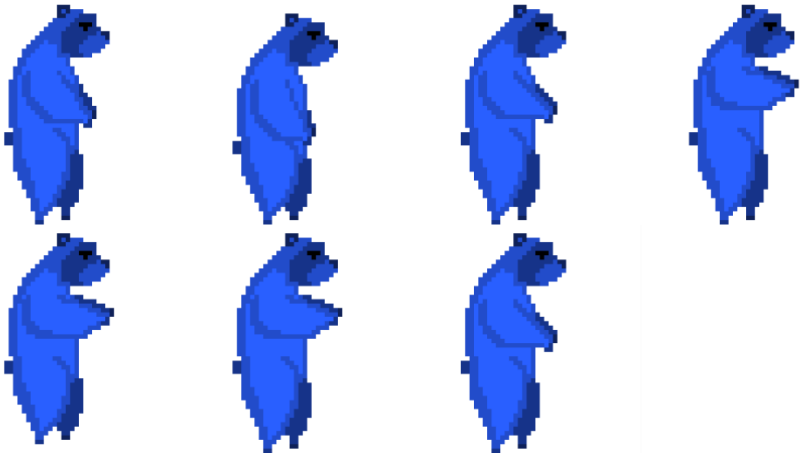
PRILOZI



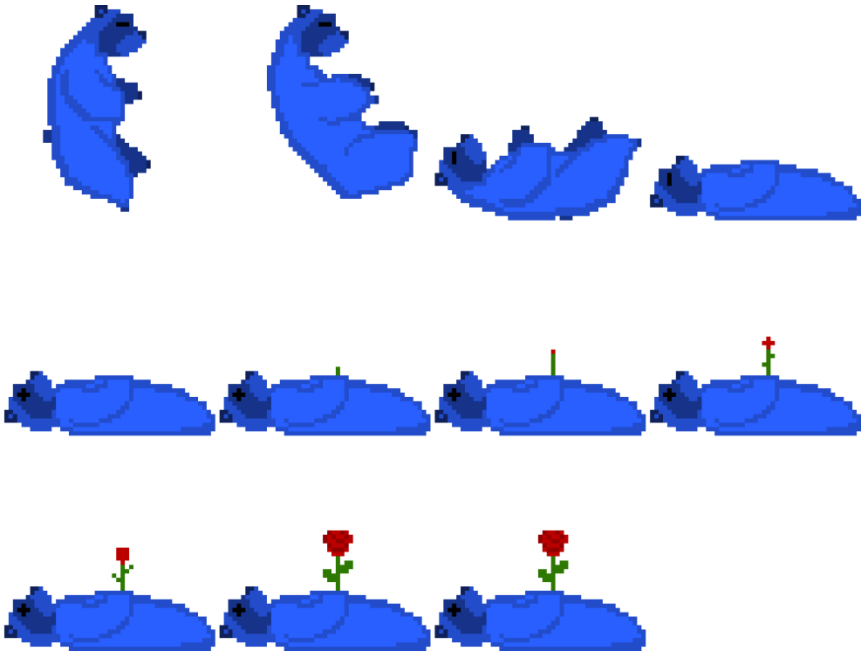
P 1 Blubear idle



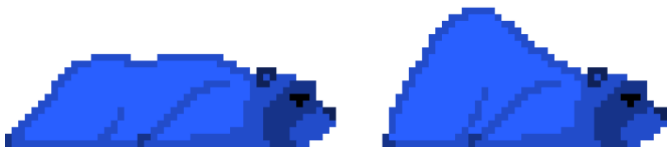
P 2 Blubear walk



P 3 Blubear jump



P 4 Blubear death



P 5 Blubear crawl

P 6 Enemy1 death

P 7 Enemy2 idle

P 8 Enemy2 throw

P 9 Enemy2 death