Izrada videoigre Jolly Roger u alatu Unity

Ban, Antonio

Undergraduate thesis / Završni rad

2018

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **University of Rijeka / Sveučilište u Rijeci**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:195:750842

Rights / Prava: In copyright / Zaštićeno autorskim pravom.

Download date / Datum preuzimanja: **2024-03-28**

Sveučilište u Rijeci
Fakultet informatike i digitalnih tehnologija

Repository / Repozitorij:

Repository of the University of Rijeka, Faculty of Informatics and Digital Technologies - INFORI Repository

zir.nsk.hr

DIGITALNA KNJIŽNICA

Image not found or type unknown

Sveučilište u Rijeci – Odjel za informatiku

Preddiplomski studij jednopredmetne informatike

Antonio Ban

Izrada videoigre Jolly Roger u alatu Unity

Završni rad

Mentor: doc. dr.sc. Marina Ivašić-Kos

Rijeka, srpanj 2018.

Rijeka, 22.07.2018.

Zadatak za završni rad

Pristupnik: Antonio Ban

Naziv završnog rada: Izrada video igre Jolly Roger u alatu Unity

Naziv završnog rada na eng. jeziku: Development of Jolly Roger game in Unity engine

Sadržaj zadatka:

Proučiti odgovarajuće alate za dizajn i razvoj računalnih igara. Proučiti vrste računalnih igara i kratko ih opisati s naglaskom na tip igre koji namjeravate razviti. Razviti igru Jolly Roger te opisati scenarij i cilj igre.

Objasniti implementaciju igre s ključnim dijelovima koda i elementima alata Unity koji su korišteni prilikom razvoja igre kako bi se ostvarila potrebna funkcionalnost objekata u igri kao što je upravljanje likom i kretanje na sceni, generiranje prepreka i neprijatelja, određivanje fizike objektima, računanje rezultata i praćenje napretka u igri.

Opisati dizajn igre, stvaranje grafičkih modela i objekata te njihovu integraciju s Unity-jem ukoliko su korištene vanjske aplikacije poput Blendera.

Mentor

doc. dr. sc. Marina Ivašić-Kos

_____

Voditelj za završne radove

dr. sc. Miran Pobar

_____

Zadatak preuzet: 15.02.2018.

_____

(potpis pristupnika)

Sažetak

U ovom radu opisuje se izrada videoigre Jolly Roger. Definirana je razlika između tipa i žanra videoigara. Jolly Roger je po tipu platformer, a po žanru akcijska igra. Cilj igre je dovesti glavnog lika Jolly Rogera do njegovog broda. U radu se također nalazi i opis alata Unity koji je korišten prilikom razvoja igre kako bi se ostvarila potrebna funkcionalnost objekata u igri, kao što je upravljanje likom i kretanje na sceni, određivanje fizike objektima, računanje rezultata i praćenje napretka. Istaknuta je implementacija navedenih funkcionalnosti objekata pomoću C# programskog jezika.

Ključne riječi

Videoigra Jolly Roger, Unity engine, 2D, platformer, C#, objekt

Sadržaj

Zadatak za završni rad 1
Sažetak 2
Ključne riječi 2
1. Uvod 4
2. Video igre 5
3. Unity 6
3.1. Unity Editor 8
4. Razvoj igre 10
4.1. Ideja 10
4.2. Kamera 11
4.3. Igrač 11
4.4. Neprijatelji 14
4.5. Nivo 17
4.5.1. Level Manager 17
4.5.2. Checkpoint 17
4.5.3. Platforme 18
4.5.4. Moving platform 18
4.5.5. Portal 19
4.5.6. Piratski brod 20
4.5.7. Pozadina 20
4.6. Animacije 21
4.7. Grafičko sučelje 22
4.7.1. Glavni izbornik 22
4.7.2. Pauza 24
4.7.3. Sučelje unutar nivoa 25
5. Zaključak 26
6. Literatura 27
7. Prilozi 28
7.1. Popis slika 28
7.2. Animacije 29

1. Uvod

Budući da su videoigre su postale sastavni dio svakodnevnog života zbog zabave i zanimljivih priča koje pružaju, ne iznenađuje rast industrije videoigara, kako u Hrvatskoj tako i u svijetu. Industrija videoigara zaradila je 108.4 biliona dolara u 2017. godini (Games industry 2018.).

Proces izrade video igre danas je, zbog dostupnosti informacija na internetu i mnogih razvojnih okruženja, moguć čak i nekome kome je game development nepoznat. U ovom radu je opisan proces razvoja videoigre Jolly Roger u jednom od najpoznatijih razvojnih okruženja Unity-u. Za izradu Jolly Rogera bilo je potrebno spojiti modele, animacije, zvuk i sve ostale multimedijske jedinice. Kako bi se ostvarila potrebna funkcionalnost upravljanja likom, kretanja po sceni, računanje rezultata, praćenje napretka u igri i intrakcije s neprijateljima, dodane su skripte pisane u C# programskom jeziku koje će biti objašnjene u sljedećim poglavljima.

2. Video igre

Video igre pokrivaju različite vrste igara, koje se razlikuju po tipu i žanru. Tip igre je opis načina igranja igre, dok žanr opisuje sadržaj igre i radnju (Game Type and Game Genre 2005.).

Tipovi igre su: akcija, avantura, puzle, role-playing, strategija i simulacija. Na primjeru igre Doom mogu se vidjeti glavne karakteristike akcijskog tipa igre koji podrazumijeva akciju kao glavni izvor zabave. U ovakvim igrama najčešće su potrebni dobri refleksi i primjećivanje neprijatelja. Tetris koji spada po tip puzli, pruža zabavu u rješavanju problema. Takve igre su pretežno zanimljive starijoj populaciji igrača. Simulacija je tip igre u kojem se pokušava imitirati stvarni svijet. Primjer takve igre je Tycoon franšiza u kojoj igrač gradi zabavni park te mora paziti na ekonomiju i estetiku parka.

Žanrovi igara su: drama, crime, fantasy, horor, science fiction, western, mystery i war. Na primjeru Resident Evil-a mogu se vidjeti karakteristike horor igara u kojima se glavni lik nalazi u zastrašujućoj (horor) situaciji. Call of Duty spada pod žanr war. Glavni lik se nalazi u atmosferi Prvog svjetskog rata. Portal je science fiction igra u kojoj se glavni lik nalazi u razrušenom laboratoriju te pokušava izaći pomoću portala.

Različiti žanrovi i tipovi mogu se kombinirati. Većina današnjih igara se ne mogu kategorizirati u samo jedan tip ili žanr. Takav primjer igre je Diablo koja po žanru spada u fantasy i horor, a po tipu je akcija, role-playing i avantura.

3. Unity

Unity-a game engine kojeg je 2005. izradila tvrtka Unity Technologies (Dice 2013.). Tvrtku su osnovali David Helgason, Nicolas Francis i Joachim Ante 2004. godine u Danskoj. Osnivanje tvrtke je slijedilo nakon relativnog neuspjeha igre koju su napravili i želje za izradom razvojnog okruženja koje bio bilo dostupno na više platforma, Slika 1. (Unity public relations 2018.).



Slika 1 - Unity platforme

Doživjeli su veliki financijski uspjeh, uspjeli su privući programere zbog portabilnosti i slabe konkurencije s obzirom da je alternativa bila skupa, a za izradu vlastitog razvojnog okruženja su potrebne velike količine resursa.

Nove inačice Unity-a dodavale su nove funkcionalnosti kao što su Unity Asset Server (Unity Collaborate 2018.) koji je omogućio developerima da međusobno dijele resurse, mogućnost izrade videoigre za Android sustave, podrška za Windows Phone/Store te DirectX 11. Nakon verzije Unity 5.0, koja je izašla 2015. godine i dodala podršku za nove platforme i poboljšanje funkcija animiranja i audio mixanja, 2016. godine promijenio se način numeriranja verzija na sistem godine dospijeća (Games industry 2018.). Zadnja verzija je Unity 2018.2.0. koja je izašla 10.07.2018. godine.

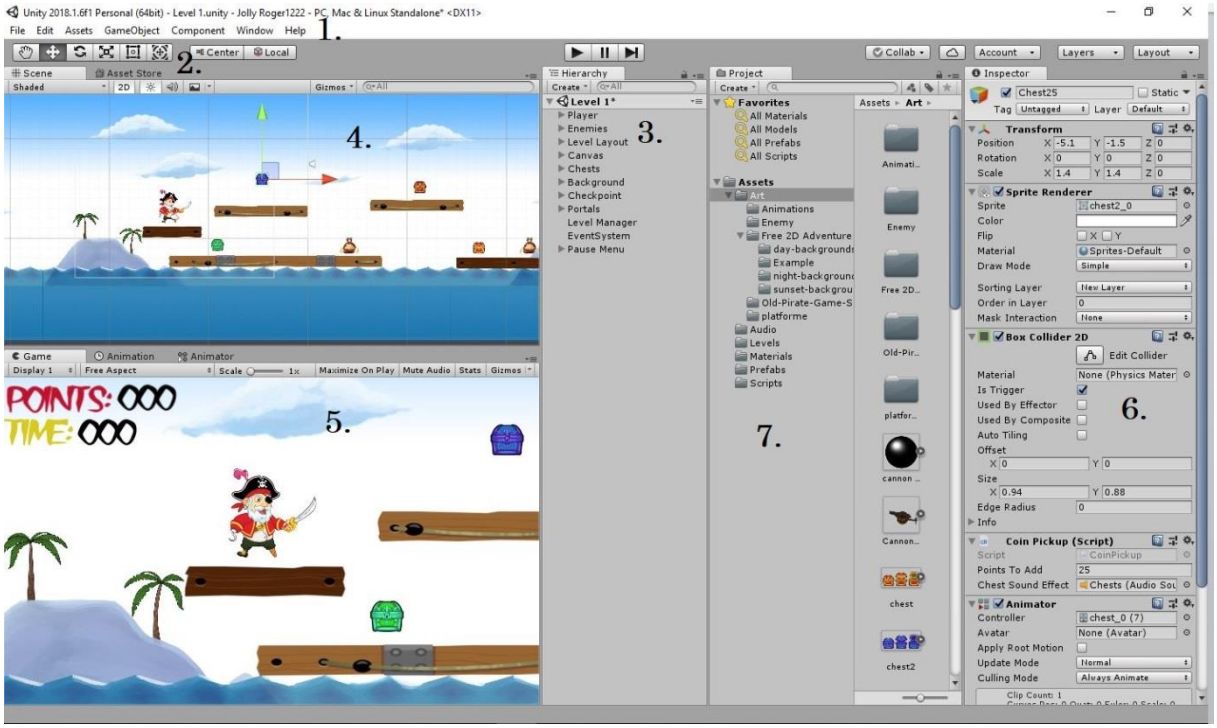
Unity se može besplatno preuzeti sa službenih stranica pri čemu se dobiva Personal licence. Personal licence omogućava izradu videoigre, ali ne i komercijalu prodaju ako ste zaradili više od $100 000. Zato postoje licence Plus, Pro i Enterprise (Unity Store 2018.).

Programiranje u Unity-u je olakšano s integriranim editorom za pisanje koda Monodevelop, a dostupni jezici za korištenje su C# i Javascript (Unity docs 2018.). Korisnik također može koristiti i neki drugi editor kao što je Microsoft Visual Studio koji je korišten tijekom izrade videoigre Jolly Roger.

Jedna od velikih prednosti Unity-a je Unity Asset Store u kojem se mogu naći resursi kao što su 2D modeli, efekti i muzika koji mogu biti besplatni ili se za njih plati onoliko koliko je vlasnik postavio cijenu.

3.1. Unity Editor

Unity Editor je sučelje u kojem korisnik izrađuje videoigru, ima pristup različitim prozorima i alatnim trakama. Korisnik može koristiti predefinirani razmještaj prozora ili može sam odrediti koje prozore želi prikazati.



Slika 2 - Unity Editor

Na Slici 2 vidimo glavne elemente predefiniranog sučelja, a to su :

1.Glavni meni

2. Osnovna alatna traka

3. Hijerarhija scene

4. Prozor pregleda scene

5.Prozor pregleda igre

6. Inspector

7. Preglednik projekta

Glavni meni sadrži glavne funkcije koje su povezane s projektom: spremanje projekta, rad s assetima, scenama, komponentama i već spomenutim prozorima. Osnovna alatna traka omogućuje odabir alata kojima se upravlja objektima na sceni kao što je Hand Tool koji omogućuje kretanje po sceni. Alatom za pomicanje objekata pomiču se odabrani objekti po sceni tako da se povlače strelice po koordinatnim osima ili klikom na pravokutnik i pomicanjem objekata u smjeru kretanja miša. Alat za rotaciju omogućuje rotiranje objekta, a alat za skaliranje objekta povećanje ili smanjivanje veličine objekta. Na alatnoj traci se nalaze i tri tipke koje omogućuju testiranje igre. Prva tipka pokreće igru, druga je pauzira, a treća pomiče igru „frame by frame“. Uz njih postoje i gumbi Collab koji omogućuju pristup servisu Unity Colaborate čija funkcionalnost daje višekorisnički pristup projektu. Sljedeći gumb je Unity Services kojim se pristupa integriranim uslugama Unity-a. Zadnjim gumbom se dolazi do korisničkog računa. Na hijerarhiji scene se vidi prikaz trenutne razine u kojoj se nalaze svi objekti. Posloženi su na principu „parent-child“ te im se može mijenjati raspored i imena bez utjecanja na samu igru. U prozoru pregleda scene mogu se mijenjati, pomoću gore navedenih alata, svi objekti koji se nalaze u hijerarhiji scene. Sve se promjene odmah vide na sceni i u pregledu igre. Prozor pregleda igre prikazuje konačni rezultat naše scene, to jest kako će naša igra izgledati korisniku. Inspector daje mogućnost mijenjanja svih postavki selektiranog objekta, od veličine i lokacije do dodavanja komponenti kao što su Collideri i skripte na sami objekt. U pregledniku projekta vide se svi asseti koje projekt sadrži, kao što su slike, animacije, audio fileovi, skripte i sve ostale komponente. Mogu se dodavati novi asseti drag and drop metodom ili uvesti iz Assset Stora.

4. Razvoj igre

Početak izrade video igre je odluka o vrsti igre kakvu želimo napraviti. Nakon toga se određuje cilj i priča igre, čime je određen njezin tip i žanr. U daljnjem tekstu ćemo objasniti kako implementirati najvažnije elemente igre. Prikazat ćemo dijelove koda koji su ključni za funkcioniranje igre.

4.1. Ideja

Jolly Roger je igra inspirirana starim 2D platformerima, kao što su Super Mario, Sonic, Metroid, kao i novim trendom „speed runova“, u kojem igrači pokušavaju prijeći igricu u što kraćem vremenu. Jolly Roger, igra čija se izrada opisuje u ovom radu, spada u podtip akcijskih igara, a to je platformer. U platformerima lik skače po platformama izbjegavajući prepreke i neprijatelje kako bi došao do nekog cilja. Po žanru je akcijska igra jer glavni lik mora izbjegavati različite neprijatelje na putu do cilja.

Jolly Roger je pirat koji se izgubio na moru te pokušava pronaći svoj brod. Kako bi došao do svog broda mora izbjeći neprijatelje, ali, pošto je pirat, ne može odoljeti skupljanju izgubljenog blaga. Igra Jolly Roger potiče igrače da razmisle isplati li im se uzeti duži put, gdje će dobiti više bodova, ali izgubiti više vremena. U igri se nalaze tri vrste škrinja koje se mogu skupiti. Svaka nosi određeni broj bodova. One koje nose više bodova su na teže dostupnim mjestima od onih koje nose manje. Kako bi igra bila zanimljivija, dodani su i neprijatelji od kojih postoje tri vrste: statični, pokretni i neprijatelji koji napadaju pirata s udaljenosti. Ako pirat padne u vodu ili se sudari s neprijateljem, umre, izgubi određeni broj bodova te se pojavi na zadnjem checkpointu. Kako bi završio igru, pirat mora doći do svog broda koji se nalazi na kraju razine.

4.2. Kamera

Kamera je element scene koji prikazuje sve ostale elemente. Kako bi se prikazala igra moramo sve elemente scene kao što su: likovi, platforme i sve ostalo postaviti ispred kamere. Kamera je definirana kao djete igrača, kako bi ga pratila dok se kreće.

4.3. Igrač

Igrač upravlja glavnim likom, piratom, (Slika 4) koji se može kretati lijevo, desno i skočiti. Lik na sebi ima collider koji nam omogućava interakciju s ostalim objektima, to jest kada collider lika dotakne collider od nekog dugog objekta moguće je pokrenuti određenu skriptu na temelju te interakcije. Na lika je dodana skripta PlayerController u kojoj je određeno kretanje, skok i dupli skok. Kako bi omogućili dupli skok varijablu doubleJumped postavljamo na false kada se lik nalazi na platformi. Nakon prvog stiskanja tipke „Space“ poziva se funkcija Jump(), a kada se ponovno stisne ponovno se poziva funkjcija Jump(), ali se i varijabla doubleJumped postavlja na true. Tijekom drugog skoka provjerava se je li vrijednost varijable doubleJumped false to jest, da bi igrač ponovno napravio dupli skok lik mora prvo pasti na platformu kako bi se varijabla doubleJumped ponovno postavila na false. To se može vidjeti na Slici 3.

```
if (grounded) doubleJumped = false;
anim.SetBool("Grounded", grounded);
MoveVelocity = 0f;
if (Input.GetKeyDown (KeyCode.Space) | Input.GetKeyDown(KeyCode.UpArrow) && grounded)
{
    Jump();
}
if (Input.GetKeyDown(KeyCode.Space) | Input.GetKeyDown(KeyCode.UpArrow) && !doubleJumped && !grounded)
{
    Jump();
    doubleJumped = true;
}
```

Slika 3 – Jumping PlayerController.cs



Slika 4 - Sprite Jolly Roger

Varijabla moveVelocity određuje trenutnu brzinu kretanja lika. Ako nema inputa vrijednost je postavljena na 0f. Ako je pritisnuta tipka „D“ ili strelica desno vrijednost se postavlja na vrijednost moveSpeed varijable koja je u editoru postavljena na 5. Ako je pritisnuta tipka „A“ ili lijeva tipka vrijednost moveVelocity se postavlja na –moveSpeed, Slika 5.

```
MoveVelocity = 0f;
if (Input.GetKeyDown (KeyCode.Space) | Input.GetKeyDown(KeyCode.UpArrow) && grounded)
{
    Jump();
}
if (Input.GetKeyDown(KeyCode.Space) | Input.GetKeyDown(KeyCode.UpArrow) && !doubleJumped && !grounded)
{
    Jump();
    doubleJumped = true;
}
if (Input.GetKey(KeyCode.D) | Input.GetKey(KeyCode.RightArrow))
{
    MoveVelocity = moveSpeed;
}
if (Input.GetKey(KeyCode.A) | Input.GetKey(KeyCode.LeftArrow))
{
    MoveVelocity = -moveSpeed;
}
GetComponent<Rigidbody2D>().velocity = new Vector2(MoveVelocity, GetComponent<Rigidbody2D>().velocity.y);
anim.SetFloat("Speed", Mathf.Abs(GetComponent<Rigidbody2D>().velocity.x));
```

Slika 5 - Movement PlayerController.cs

Kako bi sprječili da lik može beskonačno skakati definirana je varijabla grounded tipa bool. Varijabla grounded poprima vrijednost koju joj vraća funkcija OverlapCircle koja provjerava koliko je udaljena točka, koja je pozicionirana blizu nogu lika, udaljena od platforme koja je definirana kao „ground“, Slika 6.

```
grounded =Physics2D.OverlapCircle(groundCheck.position, groundCheckRadius, whatIsGround) ;
```

Slika 6 – Grounded PlayerController.cs

U PlayerController skripti imamo i funkciju Flip koja okreće lik s obzirom u kojem se smjeru kreće, to jest ako je lik okrenut prema desno vrijednost varijable facingRight će biti true. Vrijednost varijable x koordinate transform.localScale pomnožimo s -1 kako bi okrenuli lika , Slika 7. Ova se funkcija poziva u funkciji FixedUpdate() kako možemo vidjeti na Slici 8.

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

Slika 7 - Flip PlayerController.cs

```
float h = Input.GetAxis("Horizontal");
if (h > 0 && !facingRight)
    Flip();
else if (h < 0 && facingRight)
    Flip();
```

Slika 8 - FixedUpdate PlayerController.cs

4.4. Neprijatelji

Na svakom nivou se nalazi novi neprijatelj koji ima povećane funkcionalnosti u odnosu na prethodni nivo. Tako se na prvoj razini nalazi rak (Slika 10) koji je statični neprijatelj. Na njega je spojena skripta KillPlayer koja će ubiti pirata ako piratov collider dođe u kontakt s neprijateljem. Kako bi ubililika, poziva se funkcija RespawnPlayer iz klase levelManager. Ta funkcija instancira deathParticle koja je prefab (predefinirani game object koji ima svoje vrijednosti). Nakon što se funkcija WaitForSeconds() , koja uzima varijablu respawnDelay postavljenu na jednu sekundu, izvede, postavljamo lika na koordinate trenutnog checkpointa i instanciramo respawnParticle na toj poziciji. To možemo vidjeti na slici 9.

```
public class KillPlayer : MonoBehaviour {
    public LevelManager levelManager;
    void Start () {
        levelManager = FindObjectOfType<LevelManager>();
    }

    void Update () {    }
    void OnTriggerEnter2D(Collider2D other)
    { if(other.name == "Player")
        {
            levelManager.RespawnPlayer();
        }
    }
}
```

Slika 9 – KillPlayer.cs



Slika 10 – Sprite EnemyCrab

Na sljedećem nivou nalazi se kostur (Slika 11) koji šeće platformom te sadrži skriptu KillPlayer. Uz tu skriptu sadrži i EnemyPatrol koja pomoću točki WallCheck i EdgeCheck (koje su postavljene pokraj neprijatelja) pregledava hoće li se kostur sudariti sa zidom ili doći do kraja platforme i početi kretati u suprotnom smjeru. Na Slici 10 možemo vidjeti varijablu moveRight kojoj se vrijednost mijenja u False ako udari u zid ili dođe do kraja platforme te se zbog toga izvodi else uvjet koji postavlja vrijednost brzine na negativnu vrijednost pri kojoj se neprijatelj kreće prema lijevoj strani. Nakon što kostur ponovno dođe do zida ili kraja platforme vrijednost moveRight se postavlja na True pa ga okreće sa funkcijom transform.localScale i postavlja mu pozitivnu brzinu kako bi se kretao udesno.

```
public class EnemyPatrol : MonoBehaviour {
    public float moveSpeed;
    public bool moveRight;
    public Transform wallCheck;
    public float wallCheckRadius;
    public LayerMask whatIsWall;
    private bool hittingWall;
    private bool notAtEdge;
    public Transform edgeCheck;
    void Start () {
    }
    void Update () {
        hittingWall = Physics2D.OverlapCircle(wallCheck.position, wallCheckRadius, whatIsWall) ;
        notAtEdge = Physics2D.OverlapCircle(edgeCheck.position, wallCheckRadius, whatIsWall);
        if (hittingWall || !notAtEdge )
        {
            moveRight = !moveRight;
        }
        if (moveRight)
        {
            transform.localScale = new Vector3(1f, 1f, 1f);
            GetComponent<Rigidbody2D>().velocity = new Vector2(moveSpeed, GetComponent<Rigidbody2D>().velocity.y);
        }
        else
        {
            transform.localScale = new Vector3(-1f, 1f, 1f);
            GetComponent<Rigidbody2D>().velocity = new Vector2(-moveSpeed, GetComponent<Rigidbody2D>().velocity.y);
        }
    }
}
```

Slika 11 – EnemyPatrol.cs



Slika 12 –Sprite EnemySkeleton

Na trećem nivou nalazi se top (Slika 14) koji ispaljuje topovsku kuglu prema liku ukoliko se nalazi u njegovoj blizini. Na top je dodana skripta ShootAtPlayerInRange. Ona instancira prefab, koji smo nazvali CannonBall, s određenom silom na mjestu launchPoint koji je postavljen na ispred topa. Kako bi se instancirala topovska kugla prvo se mora provjeriti uvijet da se lik nalazi lijevo od topa, da se lik nalazi na udaljenosti playerRange koja je postavljena na 10 te da je prošlo dovoljno vremena od prošlog hitca. Kako bi odredili koliko vremena mora proći između hitaca, u funkciji Update shotCounter smanjujemo za Time.deltaTime koji ima vrijednost vremena koliko je prošlo od zadnjeg ispaljenog hitca. Nakon instanciranja topovske kugle, vrijednost postavljamo na waitBetweenShots koja ima vrijednost 3, Slika 13. Na taj CannonBall je dodana skripta DestroyObjectOverTime kako bi se riješio problem velikog broja instanciranih objekata, to jest da se nakon određenog vremena izbrišu, Slika 15.

```
public class ShootAtPlayerInRange : MonoBehaviour
{
    public float playerRange;
    public GameObject CannonBall;
    public PlayerController player;
    public Transform launchPoint;
    public float waitBetweenShots;
    private float shotCounter;
    void Start()
    {
        player = FindObjectOfType<PlayerController>();
        shotCounter = waitBetweenShots;
    }
    void Update()
    {
        shotCounter -= Time.deltaTime;
        if (player.transform.position.x < transform.position.x && player.transform.position.x > transform.position.x - playerRange &&  shotCounter < 0)
        {
            Instantiate(CannonBall, launchPoint.position, launchPoint.rotation);
            shotCounter = waitBetweenShots;
        }
    }
}
```

Slika 13 – ShootAtPlayerInRange.cs



Slika 15 –Sprite EnemyCannon

```
public class DestroyObjectOverTime : MonoBehaviour
{
    public float lifetime;
    void Start()
    {
    }
    void Update()
    {
        lifetime -= Time.deltaTime;
        if (lifetime < 0)
        {
            Destroy(gameObject);
        }
```

Slika 14 - DestroyObjectOverTime.cs

4.5. Nivo

4.5.1. Level Manager

U hijerarhiji scene nalazi se objekt nazvan Level Manager na koji je dodana istoimena skripta. Skripta određuje koji će se checkpoint koristiti ukoliko lik umre. Varijabla currentCheckpoint dobiva vrijednosti kada se aktivira skripta Checkpoint.cs. Kada lik umre instancira se pojavljivanje efekta smrti na toj poziciji, pozicija lika se mijenja na poziciju currentCheckpoint-a i na toj poziciji se instancira efekt respawna, kako se može vidjeti na Slici 16. Uz to određuje koliko igrač gubi poena sa svakom smrću s varijablom PointPenaltyOnDeath to jest, sa funkcijom addPoints iz klase ScoreManager pribrajamo negativne bodove. Sve ove varijable su public te se mogu mijenjati u Inspectoru.

```
public void RespawnPlayer() {
    StartCoroutine("RespawnPlayerCo");
    ScoreManager.addPoints(-PointPenaltyOnDeath);
}
public IEnumerator RespawnPlayerCo()
{
    Instantiate(deathParticle, player.transform.position, player.transform.rotation);
    player.enabled = false;
    player.GetComponent<Renderer>().enabled=false;
    Debug.Log("Player Respawn");
    yield return new WaitForSeconds(respawnDelay);
    player.transform.position = currentCheckpoint.transform.position;
    player.enabled = true;
    player.GetComponent<Renderer>().enabled = true;
    Instantiate(respawnParticle, currentCheckpoint.transform.position, currentCheckpoint.transform.rotation);

}
```

Slika 16 - LevelManager.cs

4.5.2. Checkpoint

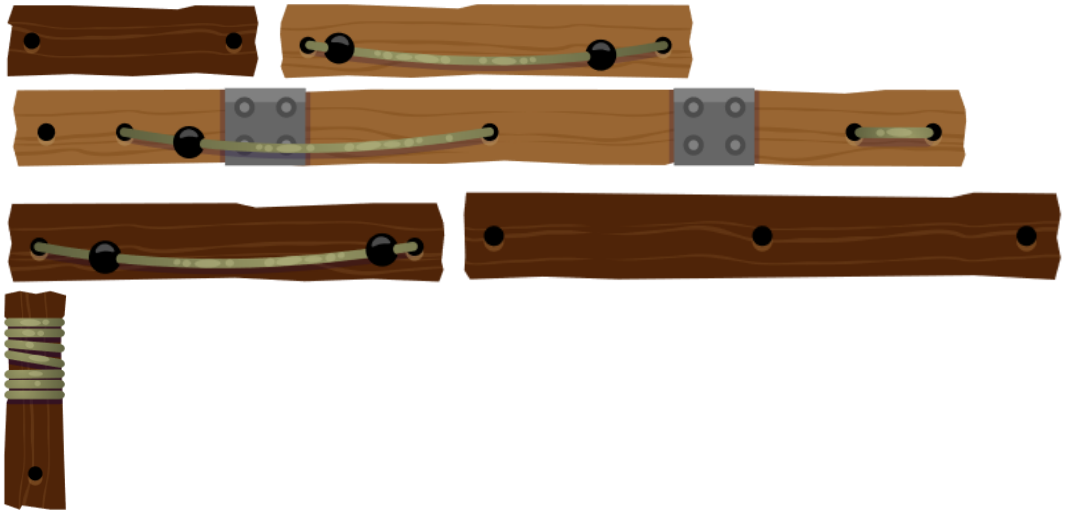
U skripti Checkpoint u funkciji Update određuje se checkpoint na koji se lik vraća ako umre. Taj chekpoint je zadnji checkpoint kroz koji je igrač prošao, to jest koji se aktivirao ulaskom lika u njegov collider. Postavlja se vrijednost trenutnog checkpointa kao currentCheckpoint u klasi levelManager, kako se može vidjeti na Slici 17.

```
void OnTriggerEnter2D(Collider2D other)
{
    if (other.name == "Player")
    {
        levelManager.currentCheckpoint = gameObject;
        Debug.Log("Activated Checkpoint " + transform.position);
    }
}
```

Slika 17 – Checkpoint.cs

4.5.3. Platforme

Platforme (Slika 18) su spriteovi kojima je dodan box collider kako bi lik mogao stajati na njima. Na njih je dodan tag „ground“ kako bi ih lik mogao razlikovati od ostalih objekata.



Slika 18 - Drvene Platforme

4.5.4. Moving platform

Kako bi se obična platforma kretala između određenih točaka, na nju dodajemo skriptu MovingPlatform u kojoj je određena brzina kretanja platforme. Varijabla currentPoint sadrži polje points[pointSelection] koje se popunjava točkama koje su određene u editoru. Platforma se kreće prema trenutno odabranoj točki currentPoint koju poziva funkcija MoveTowards. Zatim se provjerava uvijet da se pozicija platforme nalazi na poziciji trenutne odabrane točke te da se odabire sljedeća moguća točka, Slika 19. U primjeru igre Jolly Roger platforma se kreće između dvije točke (na drugoj razini) i između tri točke (na trećoj razini).

```
void Start () {
    currentPoint = points[pointSelection];
}
void Update () {
    platform.transform.position = Vector3.MoveTowards(platform.transform.position, currentPoint.position, Time.deltaTime * moveSpeed);
    if(platform.transform.position == currentPoint.position){
        pointSelection++;
        if(pointSelection == points.Length){
            pointSelection = 0;
        }
        currentPoint = points[pointSelection];
    }
```

Slika 19 - MovingPlatform.cs

Kako bi omogućili liku da ne padne s te platforme, koja se kreće, dodali smo u skripti PlayerController uvijet u kojem, ako se lik nalazi na objektu s tagom MovingPlatform, on postane „dijete“ te platforme te se zbog toga kreće relativno u njezinom smjeru i neće pasti. Ukoliko se lik nalazi na platformi koja nema tag MovingPlatform vraća se u početno stanje u kojem on nema roditelja, kako se može vidjeti na Slici 20.

```
private void OnCollisionEnter2D(Collision2D other){
    if(other.transform.tag == "MovingPlatform"){
        transform.parent = other.transform;}
}
private void OnCollisionExit2D(Collision2D other){
    if (other.transform.tag == "MovingPlatform"){
        transform.parent = null;
    }
}
```

Slika 20 - Parent&Child PlayerController.cs

4.5.5. Portal

Da bi igrač prešao s jedne razine na drugu, mora doći do zastavice. Na zastavicu je dodana skripta Teleport koja ima dvije varijable: jedna je Portal tipa GameObject, druga je Player, koja je istog tipa. Da bi dodali varijabli Portal game object, prvo ga moramo napraviti u sceni te ga onda drag and drop metodom odvući u inspector. Koordinate game objecta, u ovom slučaju zastavice, određuje na koju poziciju će se premjestiti lik. Zastavica ima collider kojemu je Is Trigger upaljen, to jest kada lik dotakne zastavicu, ona će pokrenuti skriptu i poslati ga na zastavicu koju smo prije napravili. To se može vidjeti na Slici 21.

```
public void OnTriggerEnter2D(Collider2D other)
{
    if (other.gameObject.tag == "Player")
    {
        StartCoroutine(Teleporter());
    }
}
IEnumerator Teleporter()
{
    yield return new WaitForSeconds(1);
    Player.transform.position = new Vector2(Portal.transform.position.x, Portal.transform.position.y);
}
}
```

Slika 21 –Teleport.cs

4.5.6. Piratski brod

Kako bi igrač završio nivo, lik mora doći do kraja platformi gdje se nalazi brod. Kada lik skoči na brod, poziva se skripta EndGame. Ona trenutno vrijeme i score šalje, pomoću funkcija sendToMain , koje se nalaze u klasama TimeManager i ScoreManager, u glavni izbornik. EndGame prikazuje isti pozivom funkcije LoadScene() iz klase SceneManager, kako možemo vidjeti na Slici 22.

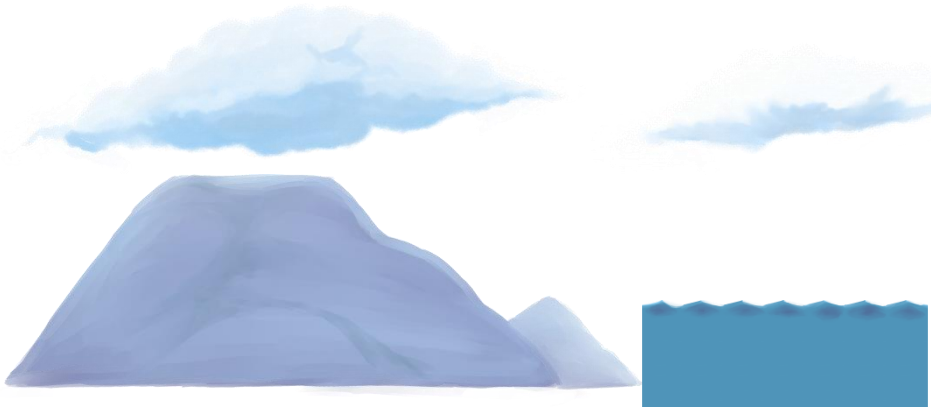
```
public class EndGame : MonoBehaviour
{
    private int score;
    private int time;
    void OnTriggerEnter2D(Collider2D other)
    {
        score = ScoreManager.getScore();
        ScoreManager.sendToMain(score);
        time = TimeManager.getTime();
        TimeManager.sendToMain(time);
         if (other.GetComponent<PlayerController>() == null)
             return;
         SceneManager.LoadScene(0);
    }

}
```

Slika 22 – EndGame.cs

4.5.7. Pozadina

Pozadina je uvezena s Unity Asset Store-a i postavljena iza platformi i likova. Pozadine su spajane su jedna do druge kako bi izgledalo da se nastavljaju. Svaki dio pozadine je zasebno postavljen (od oblaka do valova i palmi). Dijelovi pozadine vide se na Slici 23.



Slika 23 - Pozadinske slike

4.6. Animacije

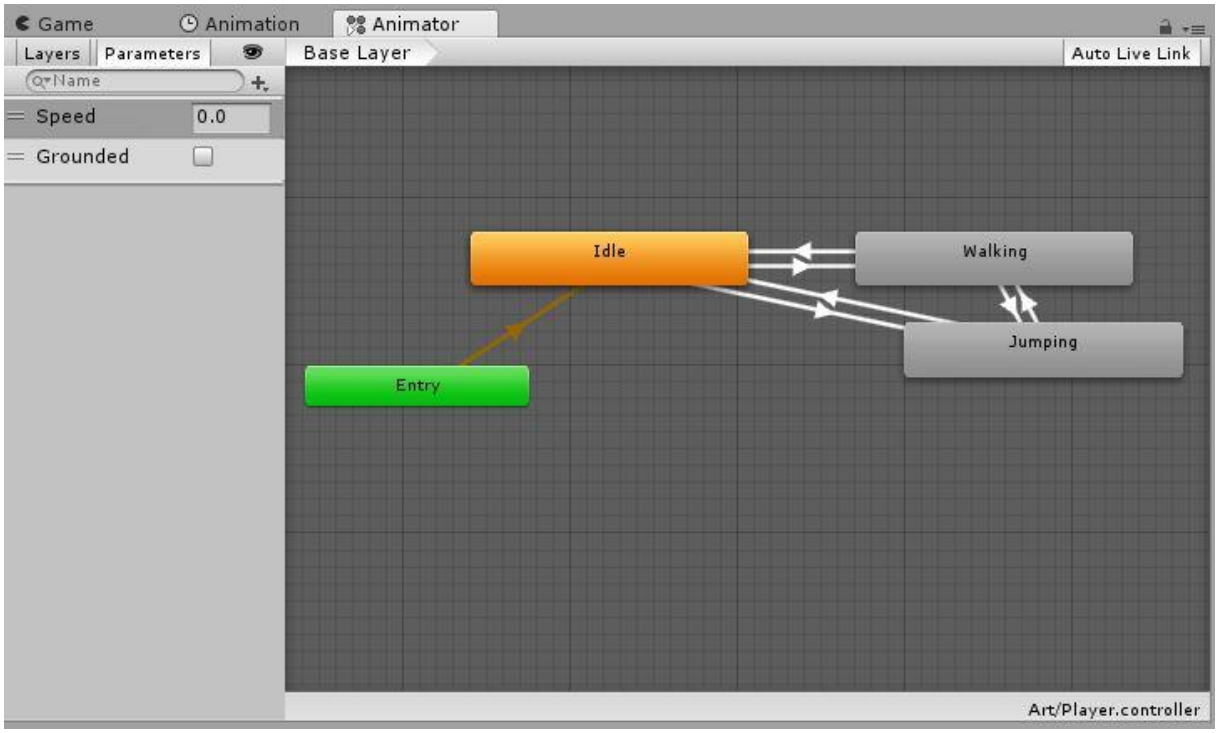
Animacija se sastoji od više spriteova koji se po određenoj brzini izmjenjuju. Nakon što se uveze spritesheet, Unity ima mogućnost „rezanja“ koje zatim posložimo jedan iza drugog praveći animaciju. Animacije u Unity-u kontrolira Animator (Slika 24) koji određuje u kojem se stanju nalazi game object kojem se radi animacija. Svako stanje predstavlja jednu animaciju. Player u igri Jolly Roger ima 3 stanja:

Idle – početno stanje u kojem se odvija animacija stajanja i iz kojeg može prijeći u bilo koje drugo stanje

Walking – stanje u koje se prelazi kada je varijabla brzine veća od zadane vrijednosti, u ovom slučaju 0.1

Jumping – stanje u koje se prelazi kada je varijabla grounded koja označava da se lik trenutno nalazi na platformi false

Lik se vraća u zadano stanje obrnuto od uvijeta koji su potrebni za ulazak u to stanje, to jest iz stanja Jumping lik se vraća u Idle ili Walking pod uvijetom da je varijabla grounded true ili Idle ako je brzina manja od 0.1 .



Slika 24 - Animator

4.7. Grafičko sučelje

U ovom poglavlju biti će opisano grafičko sučelje i način na koji igrač dobiva bodove i kako se mjeri vrijeme.

4.7.1. Glavni izbornik

U glavni izbornik ulazimo pri pokretanju igre (Slika 25). Prikazane su vrijednosti Score, Time i Final Score koje su pri prvom pokretanju na 0. Na ekranu se nalaze tri gumba: New game šalje na scenu Level 1 pomoću funkcije Play(), na kojoj se nalazi igra te igrač odmah započinje s igranjem. Drugi gumb Insctuctions šalje na scenu Instructions pomoću funkcije instructions(), na kojoj se nalaze upute (od kako se kretati s likom do koji je cilj igre). Treći gumb omogućuje da izađemo iz igre pozivanjem funkcije exit() koja pomoću klase Application poziva funkciju Quit() koja zatvara igru. Nakon završetka igre vraćamo se na glavni izbornik gdje će score i vrijeme biti prikazani uz final score koji je izračunat u skripti MainMenu u funkciji FinalScore, Slika 26. Konačni rezultat izračunat je po formuli (Score /Time)* 10. Varijabla tmp3 je tipa float, nju sa funkcijom ToString() pretvaramo u string kako bi je prikazali na glavnom izborniku. Vrijednost „F2“ nam zaokružuje rezultat na dvije decimale. Time pokušava prisiliti igrača na odabir najefikasnijeg načina rješavanja nivoa - skupljajući sve škrinjice ili preskačući neke kako bi se sačuvalo vrijeme.



Slika 25 - Glavni Izbornik

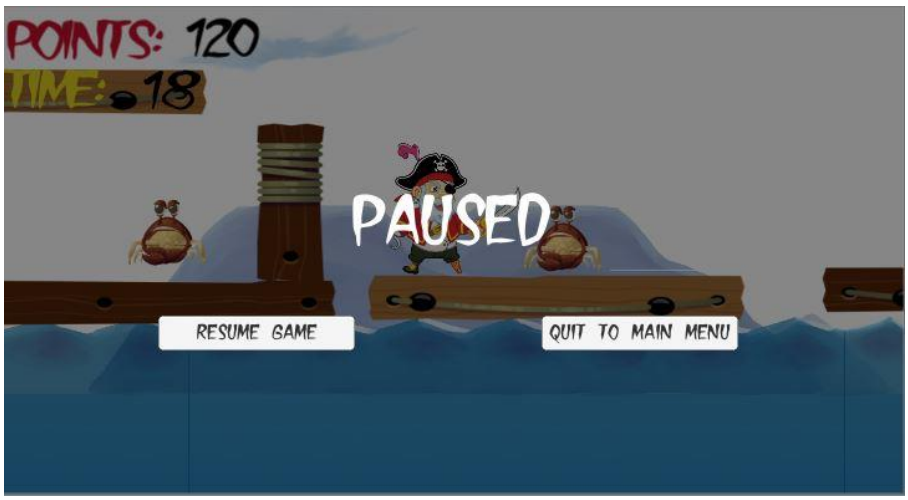
```
void Start(){
    HSFunction();
    TimeFunction();
    FinalScore();
}
void Update(){}
public void Play(){SceneManager.LoadScene(1);}
public void meni(){SceneManager.LoadScene(0);}
public void instructions(){SceneManager.LoadScene(2);}
public void exit(){Application.Quit();}
void HSFunction(){hS.text = PlayerPrefs.GetInt("HighScore").ToString();}
void TimeFunction(){tF.text = PlayerPrefs.GetInt("Time").ToString();}
void FinalScore(){
    tmp1 = PlayerPrefs.GetInt("HighScore");
    tmp2 = PlayerPrefs.GetInt("Time");
    tmp3 = tmp1/tmp2 ;
    tmp3 = tmp3 * 10;
    FS.text = tmp3.ToString("F2");
}
```

Slika 26 - MainMenu.cs

U pokušaju prelaska igre u što kraćem vremenskom periodu osvojeno je 170 bodova u 60 sekundi što na kraju iznosi 28.33 kao final score. U pokušaju prelaska igre sa skupljanjem svih škrinjica osvojeno je 595 bodova u 151 sekundu što na kraju iznosi 39.40 kao final score. Kako se može vidjeti, skupljanjem svih škrinjica dobije se više bodova, nego preskakanjem škrinjica. No, najviše bodova se dobije brzim skupljanjem škrinjica koje se nalaze blizu najbržeg puta. Takvim pokušajem osvojeno je 370 bodova u 79 sekundi što na kraju iznosi 46.84 kao final score. Jolly Roger se razlikuje od klasičnih platformerskih igara po tome što na score utječe koliko je vremena potrošeno da bi se došlo do kraja nivoa, dok u klasičnim platformerima vrijeme ne utječe na score. Ovaj način bodovanja daje „replayability“ pošto svaki pokušaj prelaska razine daje različite rezultate s obzirom na pristup rješavanju.

4.7.2. Pauza

Igrač može pauzirati igru u kojem god trenutku želi pritiskom tipke „Escape“. Tada se trenutna scena zamrzne i vrijeme prestaje teći (Slika 27). To je postignuto postavljanjem vrijednosti Time.timeScale na 0f. Na ekranu se pojavljuje tekst Pauza te dva gumba: jedan koji šalje na glavni izbornik (bez spremanja rezultata pošto igrač nije završio nivo) i drugi gumb koji nastavlja igru. Nakon pritiska gumba Time.timeScale se postavlja na defaultni 1f te se time vrijeme i igra nastavlja. Igrač može nastaviti igru ponovnim pritiskom na tipku „Escape“, Slika 28.



Slika 27 – Pauza

```
    void Update () {
        if (isPaused) {
            pauseMenuCanvas.SetActive(true);
            Time.timeScale = 0f;
        }
        else {
            pauseMenuCanvas.SetActive(false);
            Time.timeScale = 1f;
        }
        if (Input.GetKeyDown(KeyCode.Escape)) {
            isPaused = !isPaused;
        }
    }
    public void Resume() {
        isPaused = false;
    }
    public void Quit() {
        SceneManager.LoadScene(0);
    }
}
```

Slika 28 - PauseMenu.cs

4.7.3. Sučelje unutar nivoa

Tijekom igre igraču se prikazuje u gornjem lijevom kutu (Slika 29) Score koji se povećava skupljanjem škrinjica. Na svaku škrinjicu je dodana skripta CoinPickup u kojoj se određuje koji će bodova nositi. Broj bodova za svaku škrinjicu se sprema u varijablu pointsToAdd, Slika 31. Škrinjice su različite boje koje naglašavaju njihovu bodovnu razliku. Tako crveno-zlatna škrinjica nosi 10 bodova, plavo-ljubičasta nosi 25, a zelena 50 bodova. Osim toga, škrinjice su smještene tako da je teže doći do onih koje nose više bodova, to jest igrač će morati potrošiti više vremena kako bi došao do njih. Score se zbraja pomoću skripte ScoreManager koja sprema trenutni score u „PlayerPrefs“ kako se može vidjeti na Slici 30. Lokalno se sprema trenutni score te tako omogućuje da score prikazujemo na glavnom izborniku. Vrijeme se prikazuje u sekundama te se u skripti TimeManager na isti način sprema pomoću „PlayerPrefs“ funkcije.



Slika 29 - Sučelje

```
    public static void addPoints(int pointsToAdd){
        score += pointsToAdd;
    }
    public static int getScore() { return score; }
    public static void Reset() {score = 0;}
    public static int sendToMain(int score) {
        PlayerPrefs.SetInt("HighScore",score);
        return 1;
    }
```

Slika 30 - ScoreManager.cs

```
void OnTriggerEnter2D (Collider2D other)
{
    if (other.GetComponent<PlayerController>() == null)
        return;
    ScoreManager.addPoints(pointsToAdd);
    chestSoundEffect.Play ();
    Destroy(gameObject);
```

Slika 31 - CoinPickup.cs

5. Zaključak

Ovim radom prikazan je proces izrade video igre Jolly Roger koja je napravljena u Unity Engin-u uz korištenje programskog jezika C# u Microsoft Visual Studiu. Svi asseti, uključujući pozadinu, spriteove, zvučne efekte i animacije, preuzeti su besplatno s interneta ili sa Unity Asset Store-a. Jolly Roger prikazuje glavne elemente klasične platformerske igre u kojoj pratimo glavnog lika koji savladava prepreke skupljajući škrinjice s blagom. Jolly Roger se razlikuje od klasičnih platformerskih igara po tome što na score utječe koliko je vremena potrošeno da bi se došlo do kraja nivoa. Cilj ove igre je potaknuti igrača na efektivno razmišljanje korištenja jednog resursa na uštrb drugog. Igra je namjenjena prvenstveno mlađem uzrastu. Video igri Jolly Roger bi se u budućnosti moglo dodati više nivoa, novih neprijatelja i novih platformi različitih interakcija s igračem.

6. Literatura

Gameindustry.biz 2018. *Games industry generated $108.4bn in revenues in 2017*. Pokušaj pristupa 14. Lipnja 2018.https://www.gamesindustry.biz/articles/2018-01-31-games-industry-generated-usd108-4bn-in-revenues-in-2017

Game Type and Game Genre, Lindsay Grace, 2005.

Dice.com 2013. *How Unity3D Became a Game-Development Beast*. Pokušaj pristupa 14.Lipnja 2018.https://insights.dice.com/2013/06/03/how-unity3d-become-a-game-development-beast/

2018. *Unity Public Relations.* Pristupljeno 14.Lipnja.2018.

https://unity3d.com/public-relations

Gameindustry.biz 2016. *Unity dropping major updates in favour of date-based mode.* Pristupljeno 14.Lipnja.2018.

https://www.gamesindustry.biz/articles/2016-12-14-unity-dropping-major-updates-in-favour-of-date-based-model

2018. *Unity Store.* Pristupljeno 14.Lipnja.2018.

https://store.unity.com/

2018. *Unity Docs.* Pristupljeno 14.Lipnja.2018.

https://docs.unity3d.com/Manual/index.html

2018. *Unity Collaborate*. Lipanj. Pristupljeno 14.Lipnja 2018.
https://unity3d.com/unity/features/collaborate

7. Prilozi

7.1. Popis slika

Slika 1 - Unity platforme 6
Slika 2 - Unity Editor 8
Slika 3 – Jumping PlayerController.cs 11
Slika 4 - Sprite Jolly Roger 11
Slika 5 - Movement PlayerController.cs 12
Slika 6 – Grounded PlayerController.cs 12
Slika 7 - Flip PlayerController.cs 13
Slika 8 - FixedUpdate PlayerController.cs 13
Slika 9 – KillPlayer.cs 14
Slika 10 – Sprite EnemyCrab 15
Slika 11 – EnemyPatrol.cs 15
Slika 12 –Sprite EnemySkeleton 16
Slika 13 – ShootAtPlayerInRange.cs 16
Slika 15 - DestroyObjectOverTime.cs 16
Slika 14 –Sprite EnemyCannon 16
Slika 16 - LevelManager.cs 17
Slika 17 – Checkpoint.cs 17
Slika 18 - Drvene Platforme 18
Slika 19 - MovingPlatform.cs 18
Slika 20 - Parent&Child PlayerController.cs 19
Slika 21 –Teleport.cs 19
Slika 22 – EndGame.cs 20
Slika 23 - Pozadinske slike 20
Slika 24 - Animator 21
Slika 25 - Glavni Izbornik 22
Slika 26 - MainMenu.cs 22
Slika 27 – Pauza 24
Slika 28 - PauseMenu.cs 25
Slika 29 - Sučelje 25
Slika 30 - ScoreManager.cs 26
Slika 31 - CoinPickup.cs 26

7.2. Animacije



