Izrada igre utrkivanja u programskom alatu Unity

Baban, Deni

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Deni Baban

IZRADA IGRE UTRKIVANJA U PROGRAMSKOM ALATU UNITY

ZAVRŠNI RAD

Varaždin, 2019.

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Deni Baban

Matični broj: 44023/15–R

Studij: Informacijski sustavi

IZRADA IGRE UTRKIVANJA U PROGRAMSKOM ALATU UNITY

ZAVRŠNI RAD

Mentor/Mentorica:

Dr. sc. Mladen Konecki

Varaždin, rujan 2019

Deni Baban

Izjava o izvornosti

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

Autor/Autorica potvrdio/potvrdila prihvaćanjem odredbi u sustavu FOI-radovi

Sažetak

Ovaj rad se bavi izradom računalne igre utrkivanja sa pojačanjima (eng. Powerup) pomoću programskog alata Unity. Igra je namjenjena za 2-5 računala povezana u lokalnu mrežu (LAN) preko koje se igrači međusobno utrkuju. Kako bi samo utrkivanje postalo malo dinamičnije, igra sadrži razna pojačanja koja igrači mogu koristiti kako bi si stvorili prednost nad drugim igračima. Pojačanja se mogu nasumično pokupiti na stazi, a sama utrka se sastoji od nekoliko krugova po istoj stazi. Igra također ima mogućnost podešavanja raznih korisničkih opcija poput tipki za kretanje i sadrži posebnu mapu na kojoj igrač može isprobavati sva pojačanja. Vozilo igrača, uz standardno kretanje naprijed i unazad, također može skočiti i koristiti ubrzanje (eng. Boost), a njihovom kombinacijom i odletjeti malo u zrak. Mape su pravljene ručno te su prilagođene ovim mogućnostima kretanja vozila na način da je dodana vertikalna komponenta u klasične ravne staze.

Ključne riječi: Unity, utrka, algoritmi, programiranje, računalna igra

Sadržaj

1. Uvod

Završni rad se bavi izradom igre utrkivanja između više igrača povezanih na lokalnu mrežu sa mogućnostima sakupljanja dodatnih ojačanja (eng. Powerup) za vozila. Razvoj ove igre je stavlja naglasak na sposobnost shvaćanja objektno orijentiranog programiranja, primjene algoritama za riješavanje određenih problema funkcionalnosti same igre i dizajnerske kreativnosti prilikom odabira 3D modela i 2D grafike. Igra se sastoji prvenstveno od glavnoga izbornika u kojemu korisnici mogu odabrati dali žele pokrenuti svoju igru kao klijent-poslužitelj (eng. Host) ili se pridružiti već postojećoj igri, mogu pokrenuti mapu za trening kako bi isprobai razna pojačanja ili podesiti opcije. Igra sadrži nekoliko mapa na kojima se može utrkivati i na svakoj je cilj prvi napraviti određeni broj krugova po stazi. Rad također stavlja naglasak uporabu fizike prilikom realizacije kretanja vozila, korištenje ojačanja i raznih interakcija između svega toga.

Inspiracija za ovaj tip su bile igre Crash Team Racing i Mario Kart u smislu utrka sa skupljanjem ojačanja te Rocket League u svojoj primjeni fizike za kretanje vozila. Mješavina ovih igara mi se činila kao da ima velikoga potencijala biti zanimljiva i još dinamičnija od klasničnih trkaćih igara. Pošto dosta često igram razne igre i u njima uživam, ova tema završnoga rada je bila idealan odabir za mene. Imao sam nešto malo iskustva rada sa Unity alatom kada sam ga isprobavao iz znatiželje, pa se izrada ovakvoga tipa igre nije činila teška.

2. Metode i tehnike rada

Za izradu ovoga rada sam koristion popularni alat izrade igara Unity. Imao sam nekih prijašnjih iskustava u korištenju ovoga alata pa mi nije bio problem priviknuti se na njegov način rada. Unity koristi programski jezik C# za pisanje skripti koji je objektno orijentirani jezik i za njihovo uređivanje sam koristio Microsoftov Visual Studio. Način na koji je izvedeno povezivanje klasa i objekata u Unity alatu je vrlo jednostavno čak i za ljude koji nisu toliko upoznati sa programiranjem jer se manje više sve može povezati jednostavnim povlačenjem objekata ili klasa na određena mjesta u vizualnom Unity Inspector prozoru.

Kao glavnu dokumentaciju sam koristio službenu dokumentaciju sa Unity stranica i pogledao razne tutoriale igara koje su izradili. Za izradu osnovnih kretnji vozila sam pratio Wheels Collider tutorial koji obješnjava kako napraviti vozilo samo pomoću jedne kocke i četiri Wheels Collidera koji predstavljaju kotače (Wheel Collider Tutorial, Unity Manual). Također sam se služio nekim video kursevima na stranici coursera koji objašnjavaju u više detalja sam rad Unity alata i načine na koje su određene stvari izvedene, tokove rada i dobre prakse pri izradi igara kao i samom programiranju skripti.

Plan izrade ovoga projekta mi se sastojao prvenstveno od izrade osnovnih funkcionalnosti samoga igranja kao što je kretnja vozila, korištenje powerupova i navigacija izbornicima. Nakon toga sam prešao na multiplayer i povezivanje klijenata na server, kao i grafičko sučelje u izborniku za pregled svih trenutno aktivnih servera na koje se igrači mogu povezati. Isto tako je bilo potrebno prilagoditi sve skripte za rad u multiplayeru tako da svaki igrač može kontrolirati samo svoje vozilo i vidjeti na HUD-u informacije vezane samo za njega. Na kraju sam krenuo raditi stazu po kojoj bi se igrači utrkivali i sustav za praćenje odvoženih krugova kao i praćenje trenutne pozicije igrača u utrci.

Nakon što je igra bila napravljena, još su mi samo bili potrebni 3D modeli i par slika za koje bi stavio na HUD. Modele sam skidao sa Unity Store-a i a za izradu slika sam koristion GIMP ili sam ih skidao sa raznih stranica. Kada je sve to bilo gotovo, samo je još preostalo sve provjeriti i malo ispolirati kako bi igra bila kvalitetna, a onda napisati dokumentaciju.

3. Unity

Ovo poglavlje se bavi opisivanjem samoga alata u kojemu će se igra izrađivati. Unity je alat razvijen od strane kompanije Unity Technologies i napravljen je 2005. godine. Dostupan je na Windows operacijskom sustavu, MacOS i Linuxu, a aplikacije napravljene u njemu se mogu izvesti i na Android iOS, PlayStation, razne VR platforme i mnoge druge. Iako je primarno napravljen za izradu 3D aplikacija, Unity također podržava dvodimenzionalne igre. Unity alat ima besplatnu verziju kao i verziju koja se plaća i obavezna je ukoliko kreator aplikacija kroz njih zaradi više od određene svote novca tjekom jedne godine. Za ovaj rad se koristi besplatna verzija alata što se može prepoznati po Unity logu prilikom paljenja igre koji se automatski dodaje na sve aplikacije u besplatnoj verziji (Unity, Wikipedia).

Programski jezik korišten za pisanje skripti u aplikacijama je C# i piše se u Microsoftovom Visual Studiu, dok je prije Unity imao ugrađeno Monodevelop okruženje. Unity pruža mogućnosti stvaranja 3D oblika, vizualnih efekata pomoću sjena, svjetlosti, odraza i slično, sadrži u sebi PhysiX Engine koji daje mnogobrojne opcije za upravljanje fizikom u igri bez potreba vlastite implementacije. Uvoz vanjskih resursa kao što su 3D modeli, 2D grafika, zvuk i ostali, je vrlo jednostavna i brzo se može pretvoriti u ostale formate unutar samoga alata. Jedan od problema na koji sam naišao tjekom razvoja igre je bio zastarjeli Unity Unet koji služi za povezivanje igara u multiplayer, a u razvoju je novo riješenje koje će biti u potpunosti implementirano krajem ove godine. Unet još uvijek radi i ima podršku do 2022. godine, ali svugdje se izbacuju poruke upozorenja kako se on više neće podržavati.

Dokumentacija na Unity web stranici je vrlo opširna i kvalitetno napravljena te razdvojena po cjelinama. Sadrži sve bitne informacije za rad sa osnovnim elementima alata kao i detalje vezane za programiranje skripti. Također postoje mnogobrojni video tutoriali koji objašnjavaju pojedine cjeline Unity-ja ili duži video koji sadrži kompletan razvoj jednostavne igre od početka do kraja zajedno sa objašnjenjima koraka što može biti iznimno korisno početnicima u razvoju igara.

4. Kretanje vozila

4.1. Osnovna kretnja

Glavni aspekt igre jest vožnja trkaćeg automobila po zadanoj stazi, a kako bi se ova funkcionalnost ostvarila, koristi se ugrađeni PhysiX Engine i njegove ponuđene opcije poput Rigidbody komponente i Wheels Collider komponenti. Vozilo je kockastog oblika i nad njime je konstantno primjenjena sila ubrzanja prema gore koja bi suzbila gravitaciju dok god vozilo stoji na zemlji. Na rubovima kocke se nalaze četiri kotača koja su napravljena pomoću Raycast funkcije i svaki od njih primjenjuje silu na određenome kraju vozila ovisno o tome koliko je udaljeno od zemlje. Dok je vozilo u zraku i niti jedan kotač ne dodiruje tlo, sila za stabiliziranje se ne primjenjuje i fizički engine preuzima kontrolu.

```
int groundedWheels = 0;
for (int i = 0; i < 4; i++)
{
    Physics.Raycast(wheels[i], -1 * gameObject.transform.up, out wheelsRay[i], 0.3f, 9);
    if (wheelsRay[i].collider != null)
    {
        rb.AddForceAtPosition(transform.up * (wheelForce - wheelForce * 2 *
              wheelsRay[i].distance / 0.3f), wheels[i], ForceMode.Acceleration);
        groundedWheels++
    }
}
if(groundedWheels > 0) rb.AddForce(Vector3.up * 9.81f, ForceMode.Acceleration);
//Driving
if (Input.GetAxis("Vertical") != 0)
    rb.AddRelativeForce(Vector3.forward * Input.GetAxis("Vertical") * acceleration);
if (Input.GetAxis("Horizontal") != 0)
    rb.AddRelativeTorque(Vector3.up * Input.GetAxis("Horizontal") *
       (Input.GetAxis("Vertical") >= 0 ? 1 : -1) * steeringForce);
```

4.2. Skok i rotacija

Igraču je također dozvoljeno da napravi kratki skok pritiskom tipke space čime se vozilo podiže sa tla i leti po zraku. Dok je u zraku, kontrole vozila neće dodavati gas nego će vozilo rotirati ovisno o pritisnutoj osi. Tipke za naprijed i nazad će vozilo rotirati prema gore ili dolje, a lijevo i desno ga rotiraju u stranu. Ukoliko se drži tipka shift i pritišću tipke za lijevo ili desno, onda će se vozilo okretati oko svoje osi. Ovo služi za upravljanje letenjem pomoću boosta ili ispravljanja vozila iz nezgodne pozicije ako je pogođeno sa nekim od pojačanja. Ukoliko je vozilo preokrenuto naopako, igraču je dozvoljen jedan skok prilikom kojega može koristiti rotaciju za ispravljanje vozila.

```
gameObject.transform.Rotate(new Vector3(1, 0, 0)*Input.GetAxis("Vertical")*rotationSpeed,
        Space.Self);
if(Input.GetKey(KeyCode.LeftShift))
        gameObject.transform.Rotate(new Vector3(0, 0, -1)* Input.GetAxis("Horizontal") *
                rotationSpeed, Space.Self);
else
    gameObject.transform.Rotate(new Vector3(0, 1, 0)* Input.GetAxis("Horizontal") *
        rotationSpeed, Space.Self);
```

Slika 1: Skok i rotacija vozila

4.3. Boost

Boost daje igraču ubrzanje iznad normalne brzine vozila, a u kombinaciji sa skokom i rotacijom vozila u zraku, moguće je i neko vrijeme letjeti sa vozilom sve dok igraču ne ponestane boosta. Kapacitet boosta je 100 jedinica a i potroši se u par sekundi kontinuiranog držanja desne tipke miša. Boost se također sam automatski polagana obnavlja brzinom 2 jedinice boosta po sekundi, ali samo ako ga igrač trenutno ne koristi, tj. nije moguće da se u isto vrijeme boost troši i obnavlja. Grafički prikaz kapaciteta boosta se nalazi na HUD-u kojega skripta pronalazi i osvježava prikaz sa novim vrijednostima.

```
if(Input.GetKey(KeyCode.Mouse1) && boost > 0)
{
        boost -= Time.deltaTime * 30;
        GameObject.FindGameObjectWithTag("HUD
                Boost").GetComponent<HUDBoost>().SetBoost(boost);
        rb.AddRelativeForce(Vector3.forward * boostForce);
        gameObject.GetComponent<TrailRenderer>().enabled = true;
}
else
{
        gameObject.GetComponent<TrailRenderer>().enabled = false;
}

void BoostRegen()
{
        if(!Input.GetKey(KeyCode.Mouse1))
        {
                if(boost < 100) boost += 2;
                if(boost > 100) boost = 100;
                GameObject.FindGameObjectWithTag("HUD
                        Boost").GetComponent<HUDBoost>().SetBoost(boost);
        }
}
```

Slika 2: Korištenje Boosta

5. Pojačanja

Pojačanja su jedan od sastavnih djelova igre i služe kako bi igrači sustigli svoje protivnike ili ih sabotirali te samim time sebi stvorili prednost. Igra sadrži četiri pojačanja od kojih svako ima normalan i alternativan način korištenja. Normalan način se koristi tako da se pritisne lijeva tipka miša dok alternativni zahtjeva držanje tipke shift i pritisak lijeve tipke miša.

5.1. Raketa

Raketa je jedan od klasičnih pojačanja koje se koriste u ne samo igrama ovoga tipa, već i u mnogim drugima. Ona leti po ravnoj crti, ovisno u kojem smjeru je igrač ispali, sve dok se ne zabije u drugoga igrača, okolinu ili granice mape. Prilikom sudara sa nekim objektom, raketa će eksplodirati i odbaciti sve u radijusu eksplozije u smjeru od centra rakete prema van. Normalni način korištenja će raketu ispaliti ravno u smjeru u kojem igrač trenutno gleda, a alternativno ispali raketu unazad od stražnjeg djela vozila suprotno od smjera u kojem igrač trenutno gleda.

Slika 3: Raketa

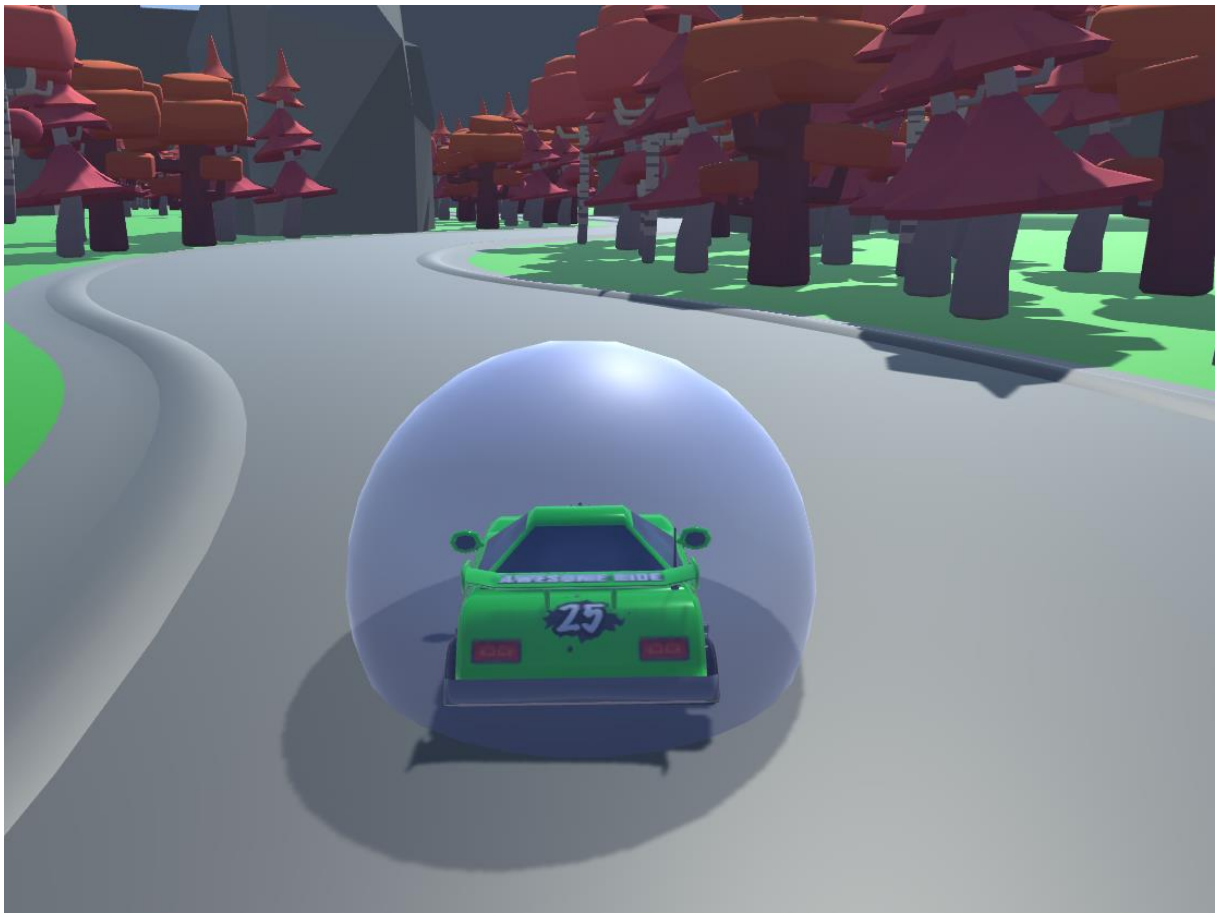
```
public class PowerupRocket : NetworkBehaviour
{
    Rigidbody rb;
    public float thrust = 10;
    public float explosionForce = 15000;
    public float explosionRadius = 5;
    public GameObject explosionPrefab;
    void Start()
    {
        rb = gameObject.GetComponent<Rigidbody>();
        rb.AddRelativeForce(Vector3.forward * 500, ForceMode.Impulse);
    }

    void FixedUpdate()
    {
        rb.AddRelativeForce(Vector3.forward * thrust);
    }

    private void OnCollisionEnter()
    {
        Collider[] explosionCollider =
              Physics.OverlapSphere(this.gameObject.transform.position, 1);
        Destroy(this.gameObject);
        foreach (Collider item in explosionCollider)
        {
            Vector3 explosionDirection = item.transform.position -
              this.gameObject.transform.position;
            if (item.GetComponent<Rigidbody>())
            {
                if(item.GetComponent<PowerupController>())
                {
                    if(item.GetComponent<PowerupController>().shield)
                        item.GetComponent<PowerupController>().BlowShield();
                    else
                    {
                        item.GetComponent<Rigidbody>().AddForce(explosionDirection *
                      explosionForce, ForceMode.Impulse);
                    }
                }
                else
                {
                    item.GetComponent<Rigidbody>().AddForce(explosionDirection *
                      explosionForce, ForceMode.Impulse);
                }
            }
        }
    }
}
```

5.2. Štit

Štit je također jedan od klasičnih pojačanja u igrama i služi kako bi blokirao bilo kakvu štetu i efekte ostalih pojačanja. Kada se štit aktivira, ostaje aktivan nekoliko sekundi i ako tjekom tog perioda igrač bude pogođen sa bilo kojim od ostalih pojačanja, ona neće imati nikakvog efekta. Alternativni način korištenja štita uzrokuje eksploziju oko samoga igrača i odbacuje ostale objekte kao što su drugi igrači, mine i rakete.



Slika 4: Štit

```
public class PowerupShield : MonoBehaviour
{
	public float duration = 10;
	float startTime;
	void Start()
	{
		startTime = Time.time;
	}
	void FixedUpdate()
	{
		if(startTime + duration < Time.Time) Blow();
	}
	public void Blow()
	{
		Destroy(this.gameObject);
	}
}
```

5.3. Kuka

Kuka je pojačanje koje služi za premještanje protivničkoga igrača unazad ili vlastitoga vozila unaprijed prema pogođenom objektu. Kuka može pogoditi i zahvatiti ili drugoga igrača ili minu te ovisno o načinu korištenja, premješta ili zahvaćeni objekt ili vozilo igrača. U normalnom načinu korištenja, zahvaćeni objekt će se privući natrag prema igraču, a dok

alternativni način korištenja privlači igrača prema zahvaćenom objektu. Uz sam objekt kuke, također se stvara i uže između kuke i igračevoga vozila koje nema collider nego samo služi radi vizualnog efekta. Za pomicanje kuke unaprijed se koristi funkcija Translate sa parametrom Vector3.forward i Space.Self, a kretnju između kuke i igrača nakon što ona zahvati neki objekt se koristi Vector3.Lerp. Uže se konstantno pozicira između kuke i igrača, rotira i uvećava.

Slika 5: Kuka

```
Vector3 hookLine = User.transform.position - gameObject.transform.position;
if (hookLine.magnitude > maxDistance) retracting = true;
else if (hookLine.magnitude < 2 && retracting)
{
    Destroy(this.gameObject);
    Destroy(Rope);
}
if (alternativeMode)
{
    if (retracting)
    {
        if (hookHit)
        {
            gameObject.transform.position = Hooked.transform.position;
            User.transform.position = Vector3.Lerp(User.transform.position,
              gameObject.transform.position, 0.02f);
        }
        else
        {
            gameObject.transform.position = Vector3.Lerp(User.transform.position,
              gameObject.transform.position, 0.98f);
        }
    }
    else
    {
         gameObject.transform.Translate(Vector3.forward * speed * Time.deltaTime,
              Space.Self);
    }
}
else
{
    if (retracting)
    {
        gameObject.transform.position = Vector3.Lerp(User.transform.position,
              gameObject.transform.position, 0.98f);
        if (hookHit)
        {
            Hooked.transform.position = gameObject.transform.position;
        }
    }
    else
        gameObject.transform.Translate(Vector3.forward * speed * Time.deltaTime,
              Space.Self);
}

//Rope
Rope.transform.position = User.transform.position - hookLine / 2;
Rope.transform.localScale = new Vector3(0.05f, hookLine.magnitude - 0.5f *
hookLine.magnitude, 0.05f);
Rope.transform.rotation = Quaternion.LookRotation(hookLine);
Rope.transform.Rotate(90, 0, 0);
```

5.4. Mina

Mina je kockasti objekt koji se postavi na stazu gdje i stoji sve dok se jedan od igrača ne zabije u nju, nakon čega stvori eksploziju iz smjera centra prema van i odbacuje sve što se nalazi u radijusu eksplozije. Normalan način korištenja mine će jednostavno stvoriti minu odmah iza igrača koji je iskoristi dok alternativni način baca minu određenu udaljenost ispred igrača kako bi mogao pogoditi protivničkog igrača ili mu zakrčiti put.

Slika 6: Mina

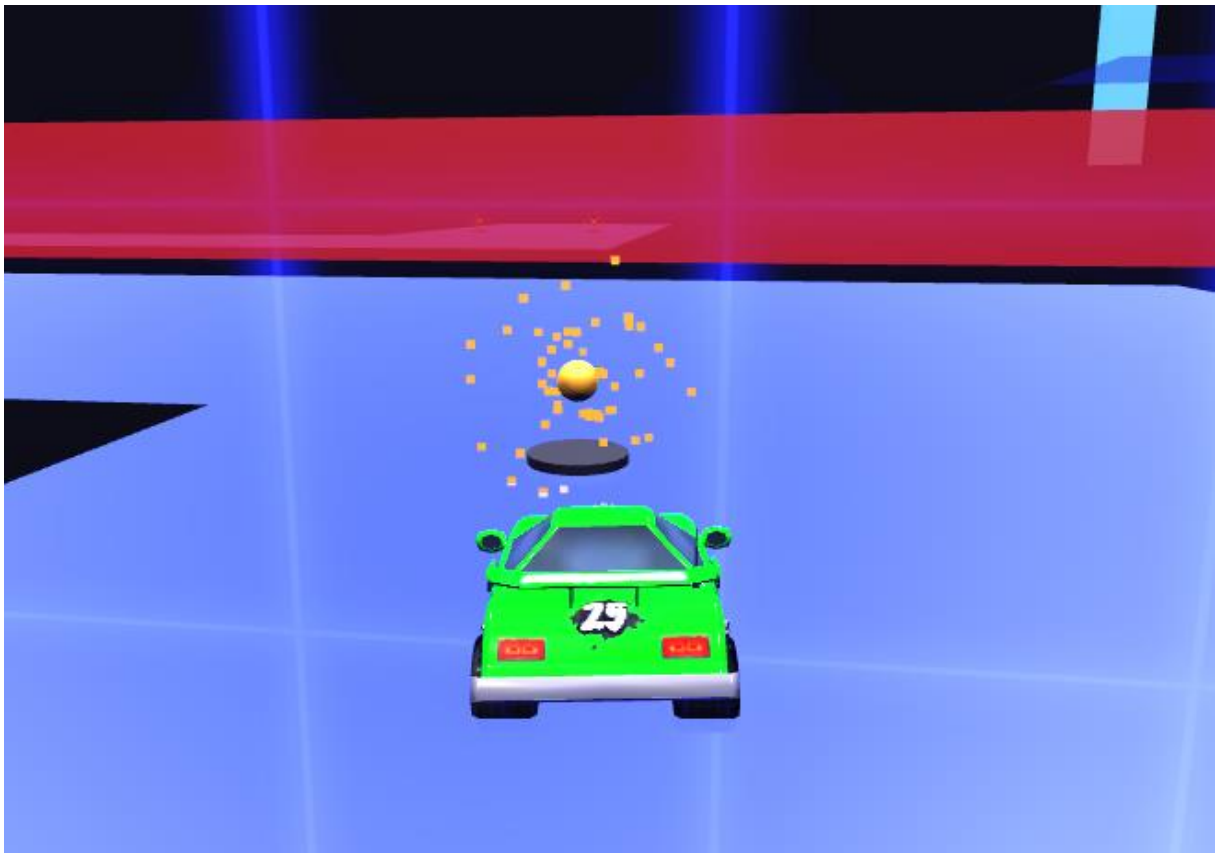
```
public class PowerupMine : NetworkBehaviour
{
    public float explosionForce = 10000;
    public float explosionRadius = 5;
    private void OnCollisionEnter(Collision collision)
    {
        if(collision.gameObject.GetComponent<Rigidbody>())
        {
            Collider[] explosionCollider =
              Physics.OverlapBox(gameObject.transform.position,
              transform.localScale * explosionRadius);
            foreach (Collider item in explosionCollider)
            {
                Vector3 explosionDirection = item.transform.position -
                     this.gameObject.transform.position;
                if (item.GetComponent<Rigidbody>())
                {
                    if (item.GetComponent<PowerupController>())
                    {
                        if (item.GetComponent<PowerupController>().shield)
                            item.GetComponentInParent<PowerupController>().BlowShield();
                        else
                            item.GetComponent<Rigidbody>().AddForce(explosionDirection *
                             explosionForce, ForceMode.Impulse);
                    }
                    else
                        item.GetComponent<Rigidbody>().AddForce(explosionDirection *
                             explosionForce, ForceMode.Impulse);
                }
            }
            Destroy(this.gameObject);
        }
    }
}
```

5.5. Sakupljanje pojačanja

Pojačanja se mogu pokupiti na određenim mjestima uzduž staze gdje se nalazi objekt Powerup Spawner. On iznad sebe stvara kuglicu koja nasumično odabire koji powerup će dati i prilikom prolaska igrača kroz nju, taj powerup se doda na igračevo vozilo. Ukoliko igrač već ima jedan powerup koji još nije iskoristio, onda se neće dobiti novi nego zadržava stari. Powerup Spawner konstantno prati dali je kuglica pokupljena ili nije, a kada je neko skupi, onda kreće odbrojavati određeni broj sekundi do stvaranja nove.

```
private void OnTriggerEnter(Collider collision)
{
    int powerupId = Random.Range(1, 5);
    if (collision.gameObject.tag == "Player")
    {
        Destroy(this.gameObject);
        collision.gameObject.GetComponent<PowerupController>().GainPowerup(powerupId);
    }
}

public class PowerupSpawner : MonoBehaviour
{
    public GameObject spherePrefab;
    GameObject sphere;
    float renewTime = 10;
    private void Start()
    {
        sphere = Instantiate(spherePrefab, this.transform.position + new Vector3(0, 0.8f,
0),
              Quaternion.identity);
    }
    void Update()
    {
        if(!sphere)
        {
            renewTime -= Time.deltaTime;
        }
        if(renewTime <= 0)
        {
            renewTime = 10;
            sphere = Instantiate(spherePrefab, this.transform.position
              + new Vector3(0, 0.8f, 0),
              Quaternion.identity);
        }
    }
}
```



Slika 7: Prikupljanje pojačanja

5.6. Korištenje pojačanja

Kao što je već ranije rečeno, pojačanja imaju dva načina korištenja, a njihovo aktiviranje i interakcija se prati iz skripte PowerupController. Ona sadrži funkcije za pridruživanje pojačanja igraču nakon što ga pokupi, funkcije za aktivaciju svakoga od pojedinih pojačanja i interakciju sa HUD-om za prikaz pojačanja kojega igrač posjeduje.

```
private void FixedUpdate()
{
    if (Input.GetKeyDown(KeyCode.Mouse0) && powerupId != 0 && gm.isGameActive &&
 isLocalPlayer)
    {
        if (Input.GetKey(KeyCode.LeftShift)) UsePowerup(true);
        else UsePowerup(false);
    }
}

public void GainPowerup(int id)
{
    if (powerupId == 0) powerupId = id;
    if (isLocalPlayer)
    {
        HUD.GetComponent<HUDPowerup>().SetImage(powerupId);
    }
}
```

```
void UseShield(bool altMode)
{
    if (altMode)
    {
        Collider[] explosionCollider =
              Physics.OverlapSphere(this.gameObject.transform.position, 5);
        foreach (Collider item in explosionCollider)
        {
            Vector3 explosionDirection = item.transform.position -
              this.gameObject.transform.position;
            if (item.GetComponent<Rigidbody>())
            {
                if (item.GetComponent<PowerupController>())
                {
                    if (item.GetComponent<PowerupController>().shield)
                        item.GetComponent<PowerupController>().BlowShield();
                    else
                    {
                        item.GetComponent<Rigidbody>().AddForce(explosionDirection *
                            shieldExplosionForce, ForceMode.Impulse);
                    }
                }
                else
                {
                    item.GetComponent<Rigidbody>().AddForce(explosionDirection *
                     shieldExplosionForce/100, ForceMode.Impulse);
                }
            }
        }
    }
    else
    {
        shield = Instantiate(shieldPrefab, gameObject.transform.position,
              Quaternion.identity);
        NetworkServer.Spawn(shield);
        shield.transform.parent = gameObject.transform;
    }
}

void UseMine(bool altMode)
{
    if (altMode)
    {
        GameObject mine = Instantiate(minePrefab, gameObject.transform.position + 3 *
              gameObject.transform.forward, gameObject.transform.rotation);
        NetworkServer.Spawn(mine);
        mine.GetComponent<Rigidbody>().AddRelativeForce(mineThrowForce * (Vector3.forward +
              Vector3.up/2), ForceMode.Impulse);
    }
    else
    {
        GameObject mine = Instantiate(minePrefab, gameObject.transform.position - 3 *
              gameObject.transform.forward, Quaternion.identity);
        NetworkServer.Spawn(mine);
    }
}
```

6. HUD

Ovo poglavlje objašnjava realizaciju HUD-a (eng. Heads-Up Display) unutar same igre. Svrha HUD-a je brzo i jednostavno pružanje osnovnih informacija igraču koje se odnose njegovo stanje ili stanje igre u kojoj se nalazi. HUD u ovoj igri prikazuje minimapu u gornjem ljevome kutu i trenutnu poziciju u utrci kao i količinu odvoženih krugova po stazi, količinu boost-a koju igrač trenutno ima na raspolaganju za korištenje u gornjem desnome kutu i odmah ispod je ikona za trenutno pokupljenji powerup koji igrač može iskoristiti. Glavni izbornih ne sadrži HUD već samo tipke za razne mogućnosti igre. Također vrijedi napomenuti da pod HUD ne spadaju izbornici u igri poput izbornika za izlaz prilikom pritiska na Escape tipku ili početnoga prozora za prikaz igrača trenutno spojenih na igru.

6.1. Minimapa

Minimapa je dio HUD-a koji prikazuje tlocrt mape po kojoj igrači voze te lokacije svih igrača na toj mapi. Ona je bitan dio trkaćih igara kako bi igrači mogli unaprijed vidjeti nadolazeće zavoje i put po kojime će morati ići i tako se unaprijed pripremiti. Ona također prikazuje gdje se na stazi nalaze ostali igrači što je bitno za efektivno korištenje pojačanja, pokazuje koliko je igrač u vodstvu ili zaostatku u odnosu na ostale igrače i omogućuje planiranje strategije daljnjeg igranja.

```
public class MinimapDisplay : MonoBehaviour
{
    public GameObject minimapPlayerPrefab;

    List<MinimapPlayerIcon> playerIcons;
    public float minimapScale = 1; // mapsize / 125
    private void Start()
    {
        playerIcons = new List<MinimapPlayerIcon>();
        foreach (GameObject player in GameObject.FindGameObjectsWithTag("Player"))
        {
            GameObject minimapIcon = Instantiate(minimapPlayerPrefab,
              this.gameObject.transform);
            minimapIcon.transform.localPosition = new Vector3(125, -125, 0);
            MinimapPlayerIcon newIcon = new MinimapPlayerIcon(player, minimapIcon);
            playerIcons.Add(newIcon);
        }
    }
    void FixedUpdate()
    {
        foreach (MinimapPlayerIcon icon in playerIcons)
        {
            icon.DotDisplay.transform.localPosition = new
              Vector3(icon.Owner.transform.position.x, icon.Owner.transform.position.z, 0)
              / minimapScale;
        }
    }
}
```

```
public class MinimapPlayerIcon
{
    public GameObject Owner { get; }
    public GameObject DotDisplay { get; }
    public MinimapPlayerIcon(GameObject owner, GameObject dotDisplay)
    {
        Owner = owner;
        DotDisplay = dotDisplay;
    }
}
```

Ovo prikazivanje igrača koristi jednostavan algoritam koji konstantno prati vektor pozicije svakoga od igrača i onda ga dvodimenzionalno prikazuje na minimapi. Mapa na kojoj igrači igraju je postavljena na koordinate ishodišta (0, 0, 0) i dohvaćanje gameObject.transform.position vektora igrača zapravo govori koliko je taj igrač udaljen od ishodišta na određenoj osi. Ako se taj isti vektor stavi na središte minimape pomoću njenih relativnih koordinata i makne dimenzija visine pošto se preslikava na dvodimenzionalnu podlogu, dobijemo jednake odnose udaljenosti na pravoj mapi i minimapi. Sada je samo potrebno podijeliti taj vektor sa mjerilom minimape, tj. odnosom veličine između prave mape i minimape kako bi smo taj vektor pravilno prikazali na minimapi. Ovo računanje se izvodi na svakom osvježavanju ekrana kako bi se dobio točan položaj protivnika u pravo vrijeme.

6.2. Pojačanje

HUD prikazuje pojačanja koja igrač trenutno ima sakupljenja ili prazan kvadratić ukoliko pojačanja nema. Skripta za upravljanje pojačanjima poziva funkcije iz skripte za prikaz pojačanja na HUD-u na način da proslijedi identifikator prikupljenog pojačanja prilikom samoga sakupljanja i da javi kako je pojačanje iskorišteno čime bi se uklonila slika sa HUD-a. Skripta za prikaz stavlja sliku na HUD ovisno o prosljeđenom identifikatoru pojačanja.

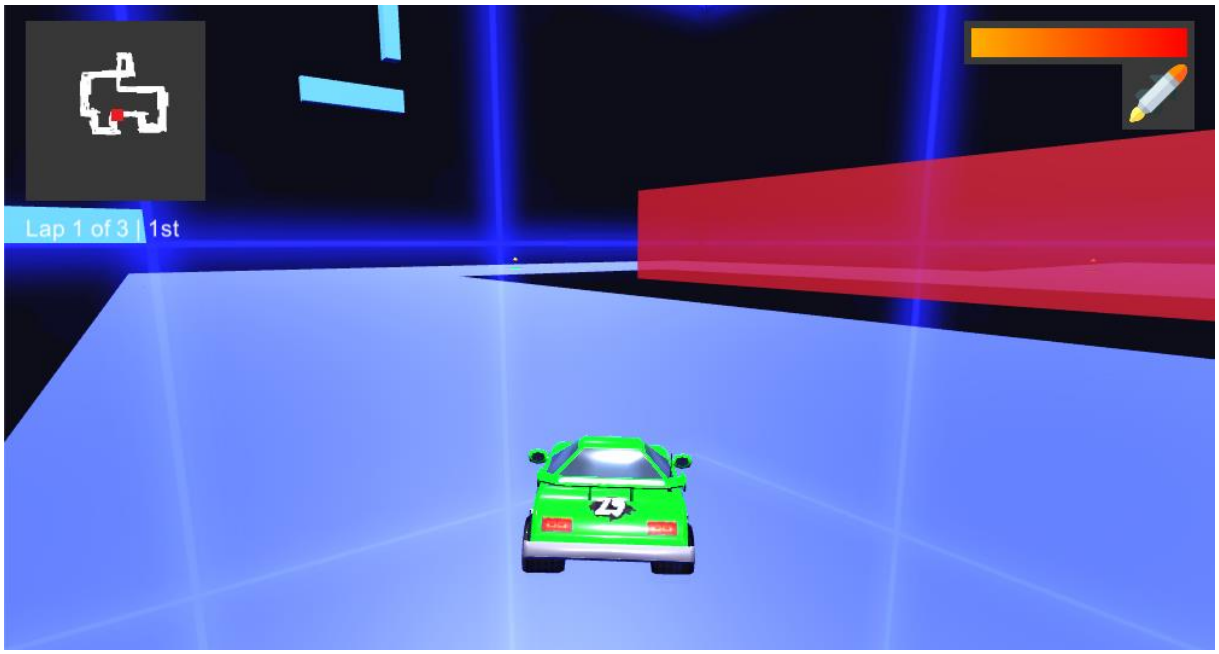
```
public class HUDPowerup : MonoBehaviour
{
    public void SetImage(int powerupId)
    {
        switch (powerupId)
        {
            case 1:
                gameObject.GetComponent<Image>().sprite =
                      Resources.Load<Sprite>("PowerupRocket");
                break;
            case 2:
                gameObject.GetComponent<Image>().sprite =
                      Resources.Load<Sprite>("PowerupHook");
                break;
            case 3:
                gameObject.GetComponent<Image>().sprite =
                      Resources.Load<Sprite>("PowerupShield");
                break;
            case 4:
                gameObject.GetComponent<Image>().sprite =
                      Resources.Load<Sprite>("PowerupMine");
                break;
```

```
        }
    }

    public void RemoveImage()
    {
        this.gameObject.GetComponent<Image>().sprite =
              Resources.Load<Sprite>("PowerupNone");
    }
}
```

6.3. Boost

Boost ima kapacitet od 100 jedinica što se lagano može pretvoriti u postotke i time se slika trenutnoga stanja boosta smanjuje ili povećava na određeni postotak svoje originalne veličine. Kako bi se ovo postiglo, koristi se atribut RectTransform-a sizeDelta. Skripta za upravljanje boost-om poziva ovu funkciju iz skripte za prikaz boosta na HUD-u sa prosljeđenim postotkom preostaloga boost-a.



Slika 8: HUD

7. Multiplayer

Mogućnost igranja u više igrača je jedna od osnovnih funkcionalnosti ove igre. Pošto igra utrkivanja zahtjeva da se igrač utrkuje protiv nekoga, a igra nema neku umjetnu inteligenciju protiv koje bi igrač mogao igrati sam, potrebno je omogućiti umreživanje računala kako bi igrali međusobno jedni protiv drugih. Za realizaciju ove mogućnosti se koristi Unity-jev HLAPI (High Level API) za multiplayer. Unity trenutno razvija novi sustav multiplayera no nije još u potpunosti gotov i dokumentacija za njegovo korištenje je jako rijetka pa se za ovaj rad koristi uskoro zastarijeli HLAPI. Podrška za njega će u potpunosti nestati oko 2022. godine, a novi sustav bi trebao biti funkcionalan već 2020. godine (UNET Deprecation FAQ, Unity Support).

7.1. Host

Igrač ima mogućnost pokretanja novih utrka kao i mogućnost priključivanja na neke druge. Kada igrač želi pokrenuti nove utrke na odabranoj mapi, on mora preuzeti ulogu host-a, tj. servera i klijenta u isto vrijeme. Prilikom odabira željene mape, igrač pokreće scenu putem Network Manager-a i postavlja svoju IP adresu. Nakon čega se pokreće Network Discovery komponenta koja oglašava (eng. Broadcast) postojanje utrke ostalim računalima u lokalnoj mreži te svoju IP adresu. Kada se dovoljan broj igrača pridruži na njegov server, host može pokrenuti igru i time se oglašavanje prekida kako se više igrači nebi mogli pridružiti na utrku koja je već u tijeku.

```
NetworkManager networkManager;
public GameObject mapPicker;

private void Start()
{
	networkManager = gameObject.GetComponent<NetworkManager>();
	networkManager.networkAddress = GetLocalIP();
}
private void StartHosting()
{
	networkManager.onlineScene = mapPicker.GetComponent<MapPicker>().GetSelectedMap();
	networkManager.StartHost();
}
```

7.2. Klijent

Klijent je računalo koje se pridružuje na već postojeću utrku prije njenoga početka. Kako bi se klijen povezao potrebno je kliknuti na tipku Join u izborniku sa popisom svih trenutno oglašavanih utrka. Time se postavlja IP adresa u Network Manageru na adresu poslužitelja i spaja se na njega kao klijent.

```
public void ConnectClient()
{
	CleanIP();
	netManager = GameObject.FindGameObjectWithTag("Net
		Menager").GetComponent<NetworkManager>();
	netManager.networkAddress = hostIP;
	netManager.StartClient();
}
```

7.3. Prikaz aktivnih igara

Kako igrač nebi morao ručno unositi IP adresu servera na kojega bi se spojio, koristi se Network Discovery komponenta koja služi za oglašavanje i slušanje mrežnih poruka u lokalnoj mreži. Host pokreće svoje oglašavanje prilikom stvaranja utrke i prekida ga onda kada odluči pokrenuti samu utrku, a klijenti u izborniku mogu vidjeti listu svih utrka na koje se trenutno mogu pridružiti. Network Discovery u izborniku osluškuje sve nadolazeće poruke sa IP adresama servera i onda ih prikazuje u listi sa tipkama za spajanje. Prilikom pritiska na tipku spajanja, IP adresa u Network Manageru se postavlja na IP adresu servera i klijent se nakon toga jednostavno može spojiti.

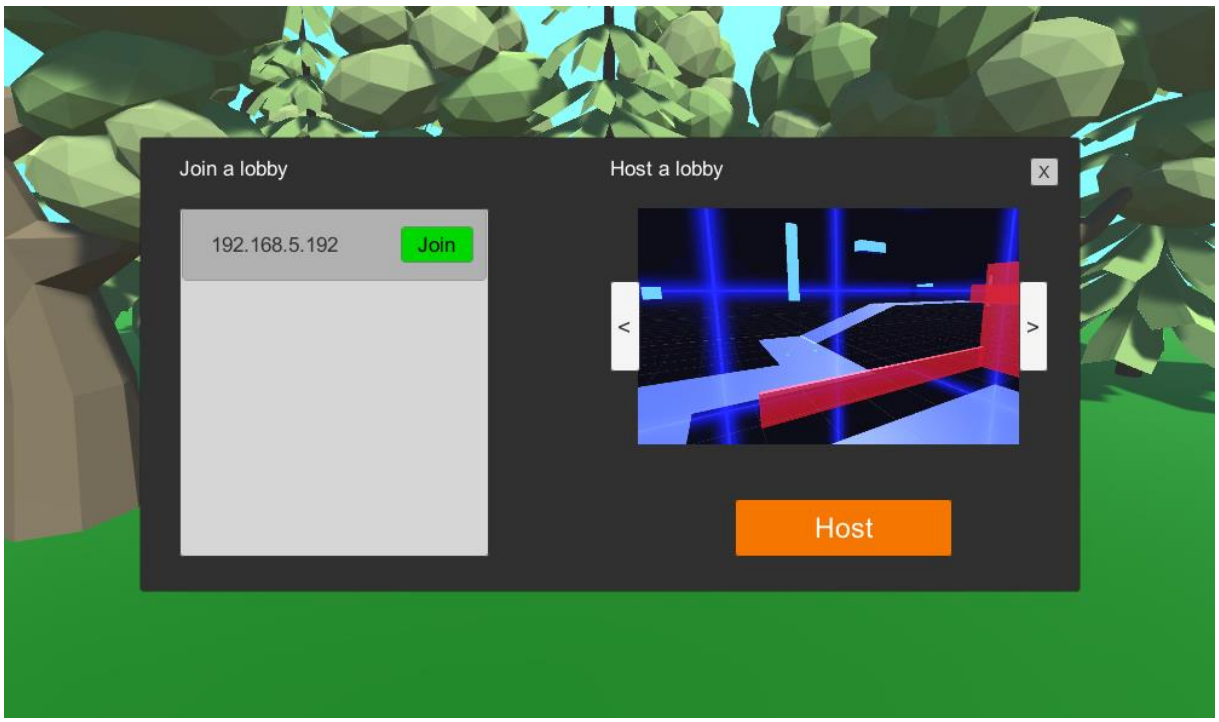
```
void CreateListings()
{
	for(int i = 0; i < this.transform.childCount; i++)
	{
		Destroy(this.transform.GetChild(i).gameObject);
	}
	int listingCounter = 0;
	foreach(string item in netDiscovery.GetComponent<NetworkFind>().GetBroadcasts())
	{
		GameObject newListing = Instantiate(listingPrefab, this.transform);
		newListing.transform.localPosition = new Vector2(0, -100 * listingCounter);
		newListing.GetComponent<NetworkJoin>().hostIP = item;
		newListing.GetComponent<NetworkJoin>().SetDescription();
	}
}

List<string> recievedBroadcasts = new List<string>();
public void StartBroadcasts()
{
	this.GetComponent<NetworkDiscovery>().Initialize();
	this.GetComponent<NetworkDiscovery>().StartAsClient();
	InvokeRepeating("BroadcastCleanup", 0f, 6f);
}
```

```
public override void OnRecievedBroadcast(string fromAddress, string data)
{
	UpdateBroadcastList(fromAddress);
}


void UpdateBroadcastList(string ip)
{
	bool found = false;
	foreach(string item in recievedBroadcasts)
	{
		if(item == ip) found = true;
	}
	if(!found) recievedBroadcasts.Add(ip);
}

void BroadcastCleanup()
{
	recievedBroadcasts.Clear();
}
```



Slika 9: Popis aktivnih igara

8. Utrkivanje

Glavna stavka ove igre je utrka između nekoliko vozila po zadanoj mapi. Kako bi se ova funkcionalnost ostvarila, potrebno je imati točke na mapi koje predstavljaju početnu liniju i nekoliko kontrolnih točaka. Igraču se broje prolasci kroz početnu liniju kao krugovi koje je odvozio, a igra završava onda kada napravi određeni broj krugova.

8.1. Početak i kraj

Utrka započinje onda kada host pritisne tipku za start prilikom čega se prikaže odbrojavanje od tri sekunde kako bi svi igrači stigli reagirati na početak utrke. Kada odbrojavanje bude gotovo, vozila se mogu početi kretati i sve njihove funkcionalnosti su omogućene. Igra je gotova za pojedinoga igrača onda kada pređe preko početne linije određeni broj puta, tj. napravi dovoljno krugova po mapi. Time se prikazuje prozor sa konačnom pozicijom igrača u utrci i tipka za povratak u glavni izbornik.

8.2. Kontrolne točke

Kontrolne točke služe za provjeru pozicije i stanja svakog pojedinog igrača u utrci. One su raspodjeljene na različita mjesta uzduž staze i reagiraju na prozalak igračevog vozila kroz njih. Također služe za kontrolu odvoženih krugova kako igrači nebi pokušali varati i skratiti si put po mapi. Svaka kontrolna točka ima u sebi varijablu tipa bool koja označava dali je igrač prošao kroz točku ili nije, a prilikom prolaska kroz početnu liniju, provjerava se dali je on prošao kroz sve kontrolne točke. Ukoliko je igrač prošao kroz sve točke i početnu liniju, priznaje mu se odvoženi krug i sve točke ponovno postavljaju vrijednosti varijabli na false jer se kroz njih ponovno mora proći u novome krugu.

```
public void CheckpointEnter(int id)
{
	if(checkpoints[id] == 0)
	{
		if(id == 0)
		{
			checkpoints[id] = 1;
			lastCheckpointId = id;
		}
		else if(checkpoints[id-1] == 1)
		{
			checkpoints[id] = 1;
			lastCheckpointId = id;
		}
```

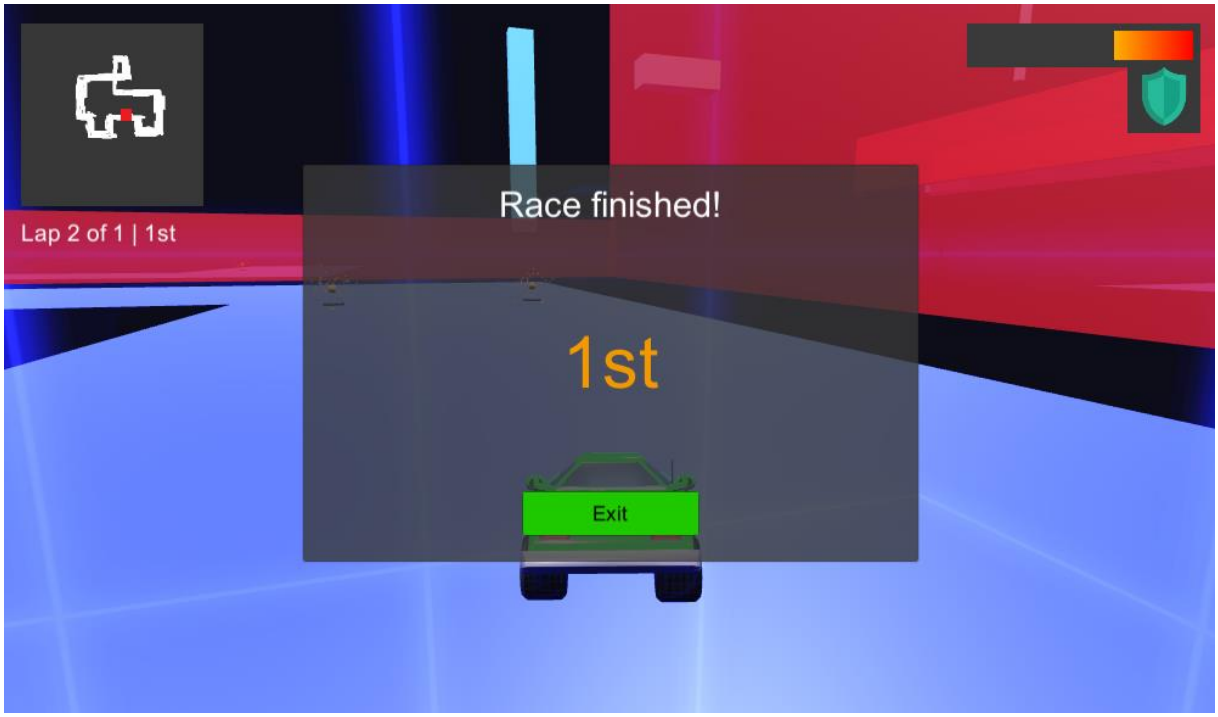
```
        }
}
public void StartlineEnter()
{
        bool allCheckpointsPassed = true;
        for(int i = 0; i < checkpoints.Length; i++)
                if(checkpoints[i] == 0) allCheckpointsPassed = false;
        if(allCheckpointsPassed)
        {
                laps += 1;
                lastCheckpointId = -1;
                for(int i = 0; i < checkpoints.Length; i++) checkpoints[i] = 0;
        }
}
```

8.3. Određivanje mjesta u utrci

Također je bitno u svakome trenutku utrke znati na kojoj poziciji se igrač nalazi u odnosu na ostale igrače. Za ovo je potreban algoritam koji računa odvožene udaljenosti mape za svakoga od igrača i onda ih poreda od većega prema manjemu. Skripta za određivanje pozicije koristi jednu izmišljenu vrijednost nazvanu positionValue koja se računa u svakome trenutku. Ona funkcionira na način da joj se pribroji određeni broj bodova za svaku prođenu kontrolnu točku uzimajući u obzir i broj odvoženih krugova, a zatim se na taj iznos oduzima preostala udaljenost koju igrač treba odvoziti do slijedeće kontrolne točke pomoću Vector3.magnitude. Ovi brojevi predstavljaju udaljenost koju je igrač prešao po mapi na način da veći broj predstavlja veću pređenu udaljenost i time veću poziciju u samoj utrci. Kako bi igrač saznao svoju poziciju, sve što se treba učiniti je pogledati koliko ostalih igrača ima veći positionValue od njega.

```
void SetPositionValue()
{
    int nextCheckpointIndex = 0;
    positionValue = laps * 1000 * GameObject.FindGameObjectsWithTag("Checkpoint").Length;
    for (int i = 0; i < checkpoints.Length; i++)
    {
        if (checkpoints[i] == 1) positionValue += 1000;
        else
        {
            nextCheckpointIndex = i;
            break;
        }
    }
    foreach (GameObject item in GameObject.FindGameObjectsWithTag("Checkpoint"))
    {
        if (item.GetComponent<TrackCheckpoint>().checkpointId == nextCheckpointIndex)
        {
            Vector3 nextCheckpointDistance =
              this.gameObject.transform.position - item.transform.position;
            positionValue -= nextCheckpointDistance.magnitude;
        }
    }
}
```

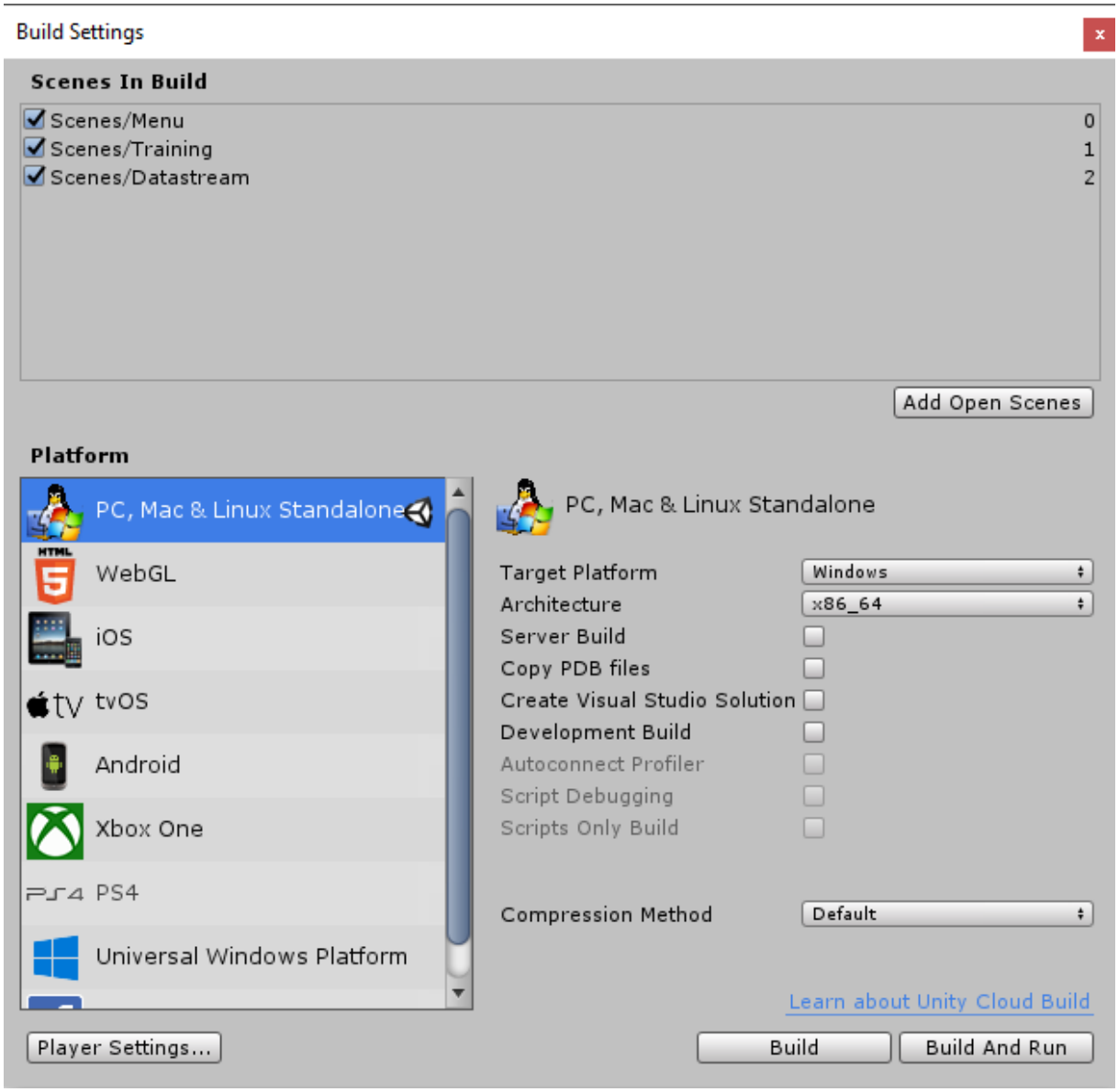
```
public string CalculatePosition()
{
    string position;
    int positionCounter = 1;
    foreach (GameObject kart in GameObject.FindGameObjectsWithTag("Player"))
    {
            if (kart != this.gameObject && kart.GetComponent<PlayerData>().positionValue >
              this.gameObject.GetComponent<PlayerData>().positionValue) positionCounter +=
       1;
    }
    if (positionCounter == 1) position = "1st";
    else if (positionCounter == 2) position = "2nd";
    else if (positionCounter == 3) position = "3rd";
    else position = positionCounter.ToString() + "th";
    return position;
}
```



Slika 10: Zaslon pri završetku utrke

9. Distribucija

Nakon što je igra u potpunosti napravljena i kvalitetno uređena, posljedni korak je distribucija igre. Pomoću distribucije pretvaramo Unity igru u prozivod koji ne zahtjeva da korisnik ima instaliran Unity ili Visual Studio na svome uređaju već jednostavnu aplikaciju koja se pokrene klikom na ikonu. Unity ima ugrađeni sustav koji izvozi igru u obliku za željeni operacijski sustav prilikom čega je potrebno unjeti neke od osnovnih podataka i postavki. Potrebno je odabrati koje sve scene će biti u konačnom proizvodu, platformu i onda pritisnuti Build tipku koja stvara izvršne datoteke igre.



Slika 11: Build postavke

10. Zaključak

Projekt razvoja igre u alatu Unity može biti jako zanimljivo i korisno iskustvo, ali samo ukoliko je osoba koja tu igru razvija spremna uložiti dovoljno vremena i truda u savladavanje svih aspekata alata, kao i principe objektno orijentiranoga programiranja. Općenito razvoj igre u Unity-ju nije toliko težak zadatak pošto je sam alat dobio ovakvu popularnost koju danas ima baš zbog svoje jednostavnosti korištenja i ubrzavanja procesa izrade, no postoje određene stvari za koje je potrebno posvetiti više vremena i pažnje kako bi se kvalitetno izradile.

Žanr igara utrkivanja donosi nekoliko problema koje je potrebno savladati i nisu toliko jednostavni za izradu. Prvi problem sa kojim sam se susreo je bio Unity-jev PhysiX Engine preko kojega sam morao izraditi vozilo i njegovo kretanje po mapi. Podešavanje raznih parametara fizičkih aspekata igre je jako naporan i monoton posao koji ponekada zna potrajati ovisno o željenoj razini kvalitete. Vozilo je bilo potrebno namjestiti za laganu i stabilnu kretnju po tlu, bez proklizavanja pri skretanju, također je trebalo sposobno za kretanje i rotaciju po zraku i borbu sa gravitacijom, a onda su na red dolazila i pojačanja. Drugi problem je bilo povezivanje u mrežni način igranja koji zahtjeva pravilno korištenje raznih mrežnih komponenti i konstantno čitanje dokumentacije. Dosta vremena je utrošeno na otkrivanje pogrešaka u kodu pri izradi ovih funkcionalnosti i dosta vremena je utrošeno na učenje kako se ove komponente koriste u samome Unity-ju.

Jedni od lakših i bržih dijelova izrade projekta je bilo pronalaženje 3D modela na internetu i njihovo uvoženje u projekt, izrada HUD-a kao i glavnoga izbornika te programiranje osnovne logike igre. Izrada izgleda mapa je također bio jedan od lakših dijelova koji u određenoj mjeri zahtjeva kreativnost pri dizajnu okoliša i atmosfere mape, a za kompletnu izradu mi je bio potreban samo jedan dan.

U konačnici smatram da mi je ovaj projekt dao dosta iskustva sa konkretnom primjenom objektno orijentiranoga programiranja i razvio organizacijske vještine koje se potrebne za upravljanje tokom razvoja video igara. Također sam dosta vremena proveo čitajući dokumentaciju za korištenje Unity-jevih komponenti i klasa što je dosta bitno za programiranje bilo čega u današnjem vremenu. Bilo je potrebno nekoliko dana aktivnoga rada, no smatram da se isplatilo i da je vrijeme korisno utrošeno.

Popis literature

[1] Unity User Manual. Dostupno na: https://docs.unity3d.com/Manual/index.html. [Pristupljeno: 9. Rujna 2019].

[2] Wheel Collider Tutorial, Unity User Manual. Dostupno na: https://docs.unity3d.com/Manual/WheelColliderTutorial.html. [Pristupljeno: 9. Rujna 2019].

[3] Unity (game engine), Wikipedia. Dostupno na: https://en.wikipedia.org/wiki/Unity_(game_engine). [Pristupljeno: 9. Rujna 2019].

[4] UNet Deprecation FAQ, Unity Support. Dostupno na: https://support.unity3d.com/hc/en-us/articles/360001252086-UNet-Deprecation-FAQ. [Pristupljeno: 9. Rujna 2019].

[5] Unity Asset Store, „Low poly styled rocks“. Dostupno na: https://assetstore.unity.com/packages/3d/props/exterior/low-poly-styled-rocks-43486. [Pristupljeno: 9. Rujna 2019].

[6] Unity Asset Store, „Toon Race Car – Low Poly“. Dostupno na: https://assetstore.unity.com/packages/3d/vehicles/land/toon-race-car-low-poly-18735. [Pristupljeno: 9. Rujna 2019].

[7] Unity Asset Store, „Cartoon explosive“. Dostupno na: https://assetstore.unity.com/packages/3d/props/weapons/cartoon-explosive-113242. [Pristupljeno: 9. Rujna 2019].

[8] Unity Asset Store, „Rockets, Milssiles & Bombs – Cartoon Low Poly Pack“. Dostupno na: https://assetstore.unity.com/packages/3d/props/weapons/rockets-missiles-bombs-cartoon-low-poly-pack-73141. [Pristupljeno: 9. Rujna 2019].

[9] Unity Asset Store, „Free Trees“. Dostupno na: https://assetstore.unity.com/packages/3d/vegetation/trees/free-trees-103208. [Pristupljeno: 9. Rujna 2019].

[10] Unity Asset Store, „Birch Tree – Proto Series - Free“. Dostupno na: https://assetstore.unity.com/packages/3d/vegetation/trees/birch-tree-proto-series-free-107476. [Pristupljeno: 9. Rujna 2019].

[11] Unity Asset Store, „Pine Tree – Proto Series - Free“. Dostupno na: https://assetstore.unity.com/packages/3d/vegetation/trees/pine-tree-proto-series-free-108767. [Pristupljeno: 9. Rujna 2019].

[12] Unity Asset Store, „Grassland Tree – Proto Series - Free“. Dostupno na: https://assetstore.unity.com/packages/3d/vegetation/trees/grassland-tree-proto-series-free-108585. [Pristupljeno: 9. Rujna 2019].

[12] Unity Learn, „Karting Template“. Dostupno na: https://learn.unity.com/project/karting-template. [Pristupljeno: 9. Rujna 2019].

[13] Flaticon, „Missile“. Dostupno na: https://www.flaticon.com/free-icon/missile_1537041. [Pristupljeno: 9. Rujna 2019].

[14] Flaticon, „Dynamite“. Dostupno na: https://www.flaticon.com/free-icon/dynamite_1592344. [Pristupljeno: 9. Rujna 2019].

[15] Flaticon, „Antivirus“. Dostupno na: Shield - https://www.flaticon.com/free-icon/antivirus_139732. [Pristupljeno: 9. Rujna 2019].

[16] HiveWorkshop, „Hook“. Dostupno na: https://www.hiveworkshop.com/threads/btn3hookgray.277863/#resource-66046. [Pristupljeno: 9. Rujna 2019].

Prilozi

[1] Igra utrkivanja „Završni Rad“ [Unity Project]. Dostupno na: https://drive.google.com/drive/folders/1WXP6F3rE9x6ZVRMY2rlTEP3cSKBaPQWw

[2] Distribucija igre. Dostupno na: https://drive.google.com/drive/folders/1IQuQroTudeWPs4ZBLmlJ96wfFkw1EEZ0