SVEUČILIŠTE U SPLITU
SVEUČILIŠNI ODJEL ZA STRUČNE STUDIJE
ODSJEK ZA INFORMACIJSKE TEHNOLOGIJE

PETAR IVANČEVIĆ

ZAVRŠNI RAD

IZRADA 3D IGRE U UNITY-u

Split, rujan 2015.

SVEUČILIŠTE U SPLITU
SVEUČILIŠNI ODJEL ZA STRUČNE STUDIJE
ODSJEK ZA INFORMACIJSKE TEHNOLOGIJE

PREDMET: OBJEKTNO PROGRAMIRANJE

ZAVRŠNI RAD

KANDIDAT: Petar Ivančević
TEMA ZAVRŠNOG RADA: Izrada 3D igre u Unity-u
MENTOR: dipl. ing. Ljiljana Despalatović

Split, rujan 2015.

Sadržaj

Sažetak

U ovom radu je prikazan proces izrade trodimenzionalne igre korištenjem Unity platforme. Tip igre koji je napravljen je utrka, gdje automobil skuplja bodove i treba proći dva kruga. Igra nakon izrade se može igrati na skoro svim platformama (android, Windows, Mac OSX, WebGL, itd.), znači na onim platformama na koje se proizvedu izvršne datoteke. Kako za igru trebaju modeli, izrađeni su korištenjem tehnologije SketchUp, te se nakon izrade uključe u Unity. Automobil posjeduje sve osnovne kretnje, te je stvoren ambijent za zanimljivije iskustvo tokom igranja. Konačna igra je skup svih pojedinih dijelova kao što su zvukovi, modeli i skripte koji čine jednu jedinstvenu cjelinu.

Summary

Development of a 3D game in Unity

In this paper the process of developing a three dimensional game is shown using the Unity platform. The game type is a race game, where a car collects points and needs to make two laps. After the creation of the game it can be played on almost any platform (android, Windows, Mac OSX, etc.), meaning on those platforms for which we make the executable files. Because the game needs models, they are built using the SketchUp technology, which are then imported in Unity. The car has all of the basic movements and the ambient is made for a better user experiance. The final game is a collection of sounds, models and scripts, which make a whole.

1 Uvod

Tema ovog završnog rada je izrada 3D igre u unity-u. Predznanje koje je potrebno za napraviti nešto slično ovom radu su C#, objektno orijentiranog programiranja, osnovno poznavanja fizike, te naravno kako voziti automobil. Kroz rad se postepeno upoznaje sa unity okruženjem i objašnjavaju najvažnije funkcionalnosti kako su realizirane. Motivacija za ovakav tip rada je zbog autorove želje za stvaranjem nečega što će i on sam moći koristiti, te graditi i razvijati kroz duži period.

U drugom poglavlju su navedene sve tehnologije koje su korištene za realizaciju igre. SketchUp su ukratko objašnjen, kako bi se znalo odakle dolaze modeli i nekakve osnove.

Treće poglavlje detaljnije opisuje unity, od osnovnih komponenti, do načina na koji se pokreće vozilo unutar igre. Najvažniji elementi su navedeni, te nakon čitanja ovog poglavlja bi čitatelj mogao dobiti sliku o kompleksnosti alata.

Četvrto poglavlje objašnjava skriptiranje unutar unity-a. Opisuje se što je skriptiranje, kako se izrađuju, osnovni sadržaj skripti, te neke tipove funkcija korištene u igri. Postoji još mnogo više funkcija koje je moguće vidjeti u dokumentaciji od unity-a, a ovdje su navedene najčešće korištene.

Peto poglavlje objašnjava izradu korisničkog sučelja, te što je korisničko sučelje usvari. Objašnjava kako se izrađuje, što je platno, kako napraviti izbornike, te manipulrati tekstovima preko referenci.

Šesto poglavlje objašnjava kako se koristi zvuk unutar unity-a. Na koji način se ustvari apstraktira zvuk, te neke od glavnih komponenti zvuka.

Sedmo poglavlje sadrži opis klasa koje su korištene unutar igre. Opisuje sve podatkovne članove i funkcije, odnosno metode.

Zadnje poglavlje je zaključak gdje se daje osvrt na rad, te misli autora nakon izrade igre.

2 Korištene tehnologije

2.1 Unity

Unity je jedan od najpopularnijih razvojnih platformi za izradu 2D i 3D igrica. Moguće je korištenjem ovog alata napraviti jednu verziju igrice koja će se moći pokretati na računalu, igrajućim konzolama, mobilnim uređajima i web stranicama.

Skripte se mogu pisati u C# ili javascriptu. Preporuka je pisati u C# zbog samog stila pisanja jer se u kratkom periodu dosegne više od pedeset linija kôda i koristi dosta ugrađenih metoda. Za sve ugrađene metoda treba znati koji su argumenti koje primaju, a ako se koristi javascript to se neće moći vidjeti. Framework koji se koristi za pisanje je Monodevelop jer se unity može koristiti na Windows mašinama i na Macintosh mašinama.

Moguće je besplatno koristit unity dok se ne dosegne prihod od 100,000 dolara. Ukoliko se zaradi ova svota novca preko igre razvijene u unity-u, tada je potrebno kupiti profesionalnu verziju. Trenutna verzija je 5.0, koja nažalost još uvijek ima problema sa linux platformom.

2.2 SketchUp

SketchUp je alat za izradu 3D geometrijskih tijela napravljen od strane Google-a. Koristi se većinom u građevinskim obrtima zbog vrlo jednostavnog načina izrade modela. Dvodimenzionalni elementima se jednostavno dodaje treća dimenzija preko gurni/povuci (*eng. Push/Pull*) alata. Ne preporuča se za izradu složenijih modela jer ne pruža dovoljno mogućnosti kao neki drugi alati.

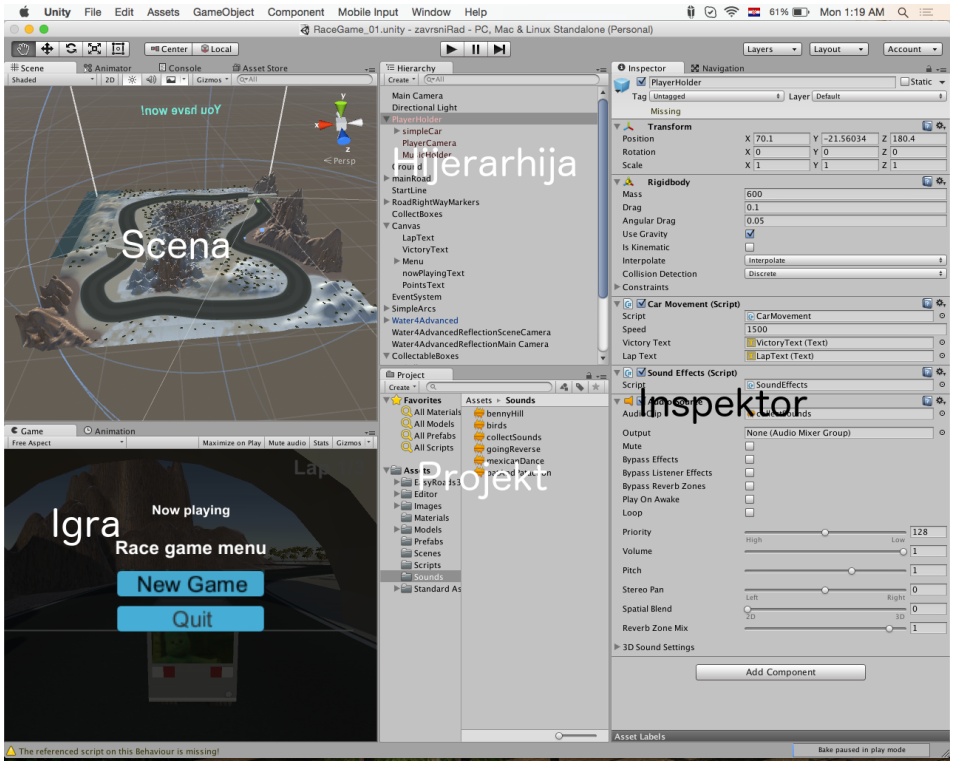
Vozilo korišteno kroz cijelu igru je također izrađen u SketchUp-u. Nakon što se izrade kompletni modeli i njihove animacije, treba se napraviti datoteka koja ima 3ds ekstenziju.

3 Unity

Glavno sučelje od unity-a se može vidjeti na slici 1. Raspored koji se može vidjeti je najoptimalniji za rad ukoliko programer ima samo jedan monitor. U slučaju da programer ima više monitora onda bi se mogao napraviti puno pregledniji raspored.

3.1 Inspektor

Inspektor (*eng. Inspector*) se koristi za provjeravanje pojedinih opcija komponenta igrajućih objekata (*eng. GameObject*). Pomoću njega je moguće dodavati elemente i obavljati bilo koju operaciju, te vidjeti sve moguće stavke o objektu.



Slika 1: Unity sučelje

3.2 Scena

Scena (*eng. Scene*) se koristi prilikom izrade igre kako bi se moglo iz neutralne pozicije gledati cijeli svijet. Sve preinake koje se rade u svijetu kao što su dodavanje objekata, određivanje položaja, dimenzija se obavljaju unutar ovog dijela. U gornjem desnom kutu se može vidjeti maleni objekt koji je crvene, zelene i plave boje. On programeru govori trenutnu orijentaciju u svijetu. Zelena boja je y, crvena je x, a plava z koordinata.

3.3 Hijerarhija

Hijerarhija (*eng. Hierarchy*) je dio u kojem se nalaze svi igrajući objekti koji su napravljeni. Ovdje je moguće vidjeti odnos svih igrajućih objekata. Pod odnos se misli na roditelj, dijete, gdje jedan igrajući objekt može biti dijete drugog. Na ovaj način je moguće iz roditelja pristupati djeci i njihovim komponentama. Iako unity pruža mogućnost i obrnutog procesa, ovo ugnježđivanje omogućuje jednu odličnu funkcionalnost koja se koristi u igri. Ako je kamera dijete nekog objekta koji se pokreće sa nekom silom, tada kamera prati kretenje tog igrajućeg objekta i dobiva se iluzija kao u 3D igricama gdje se glavni junak gleda iz trećeg lica.

3.4 Projekt

Projekt (*eng. Project*) segment omogućava programeru da vidi sve datoteke koje su uključene u igru i njihovu hijerarhiju. Zbog praktičnosti se naprave direktoriji za skripte, zvukove, modele i ostale objekte koji će se ponavljati kroz igru, kako bi se što lakše mogli pronaći prilikom rada. Postoji ugrađeni pretraživač za još jednostavnije pronalaženje željenih datoteka, kao i različiti filteri.

3.5 Igra

Ovaj dio je ono što igrač vidi kada pokrene igru, a omogućava programeru da što zornije postavi parametre kako bi igra izgledala kako je on zamislio. Moguće je postaviti da prilikom pokretanja igre u unity-u se poveća slika na maksimum (*eng. maxmizie on play*). Pokretanje igre je kao slušanje glazbe u bilo kojem alatu. Postoje igraj (*eng. play*), pauziraj i pomak po slici.

3.6 Konzola

Konzola (*eng. Console*) se koristi za provjeravanja koda unutar igre. Moguće je logove ispisivati za provjeru pojedinih varijabli tokom igranja i provjeravanja okidača da li uistinu rade. Bilo koja greška koja nije uzrokovana tijekom rada će prvo biti prikazana u konzoli crvenom bojom sa oznakom linije i imenom skripte u kojoj se dogodila greška.

3.7 Trgovina

Trgovina (*eng. Asset store*) se koristi za kupnju ili nabavljanje gotovih modela preko interneta, koje je netko drugi već napravio. U slučaju ove igre, cesta je preuzeta sa trgovine jer sama izrada ceste bi bio preduogtrajan proces. Trgovina ima svoje filtere za pojedine kategorije igara i tipova objekata koji su potrebni.

3.8 Igrajući objekti

Svaki objekt koji se može vidjeti u unity-u je **igrajući objekt**. Kada se napravi bilo koj objekt on treba imati svoju poziciju unutar svijeta. Kako bi se mogla znati njegova pozicija koristimo komponente. Svaki objekt mora imati svoju transformaciju (*eng. Transform*). U suštini igrajući objekti su samo kontenjeri koji sadrže komponente.

3.9 Kamera

Kamere se koriste za prikaz igre kako je programer zamislio. Koristeći više kamera moguće je napraviti različite efekte i animacije za igrača, te stvoriti jedinstveno iskustvo tijekom igranja igre. Kamere imaju dvije moguće projekcije (ortografijsku i perspektivnu). Ortografijska se koristi za 2D igrice ili ako nije bitna dubina u igricama. Najčešće su to neke platformske igre, puzzle ili slično. Perspektivna se koristi ako je želimo pokazati dubinu u igri. U igri se koristi ovaj oblik, te je postavljena kao dijete automobilskog igrajućeg objekta. Na ovaj način kamera slijedi automobil i dobiva se pravo iskustvo vožnje automobila.

3.10 Svijetlo

Svijetla se naravno koriste za osvijetljavanje svijeta, ali i za stvaranje ugođaja. Moguće je mijenjati boje svijetla, te tako stvarati prekrasne ambijente. Može se definirati spektar, doseg, boja, tip, intenzitet, intenzitet odbijanja, sjena i ostale naprednije funkcije. Zanimljivo je što se može definirati svijetlu da ne baca sjenu za igrajuće objekte.

Za bolju funkcionalnost se može preko padajućeg izbornika na svijetlu postaviti da je već ispečeno (*eng. Baked*). Ovo znači da će se sva svijetla prilikom pokretanja igre izračunati i postaviti za igrajuće objekte koji se ne kreću. Ovo je veoma praktično jer nema potrebe preračunavati za te objekt utjecaj sa svjetlom, već će se to obavljati ukoliko pokraj njega dođe ne-statičan objekt.

3.11 Montažni objekti

Montažni objekt (*eng. Prefab*) je objekt koji je već napravljen, te se želi multiplicirati više puta. Ako se koristi više istih igrajućih objekata, kao na primjer više istih automobila koji dijele sve funkcionalnosti. Tada nije praktično raditi promjene nad svakim automobilom zasebno, te se zato koriste prefabi. Kada se napravi novi igrajući objekt može se od njega napraviti prefab preko izbornika *Asset*, pa zatim *Create Prefab*. Nakon što se napravi prefab može se jednostavno povlačiti u svijet, te tako stvarati nove instance igrajućih objekata, koje dijele iste funkcionalnosti. Sada ako se nešto želi promijeniti, može se mijenjati bilo koji od objekata i unity će pitati, da li treba obaviti ove promjene i na ostale instance ovog prefaba.

3.12 Komponente

Komponente su kao što je rečeno dijelovi igrajućih objekata. One daju funkcionalnost objektima, te omogućavaju krajnjim korisnicima da razlikuju igrajuće objekte. Neke važnije komponente koje su korištene za izradu igrice su:

- Transformacija
- Kruta tijela
- Sudarači
- Kolni sudarači

3.12.1 Transformacija

Igrajući objekti moraju imati svoju **transformaciju**, inače se neće moći prikazati u svijetu. Transformacija definira širinu, poziciju i orijentaciju objekta. Svaka transformacija ima svoj pivot koji određuje centar, odnosno prema njemu se gledaju širina, pozicija i rotacija. Pivot se može gledati globalno ili lokalno.

Globalno gledanje je pozicija pivota gledajući koordinate x,y,z svijeta. Lokalno gledanje pivota je relativna pozicija naspram pozicije objekta. Primjer transformacijog elementa (*Gizmo*) se može vidjeti na slici broj 3.

3.12.2 Kruta tijela

Kruta tijela (*eng. Rigid Body*) igrajućim objektima daju ponašanje kao u stvarnom svijetu. Objektima daju dodatne informacije kao na primjer masa, primjena sile za pomicanje objekata, hoće li vjetar utjecati na kretanje i tako dalje. Dodavajući kruta tijela može se isto definirati hoće li gravitacija utjecati na element. U igri je najvažnija informacija bila masa zbog bolje simulacija pravog automobila.

3.12.3 Sudarači

Sudarači (*eng. Colliders*) su komponente koje omogućavaju igrajućim objektima fizičku koliziju. Oni se ne mogu vidjeti tokom igranja igre, a i nisu zbog toga napravljeni. Postoji više oblika sudarača (kvadar, sfera, kapsula, krug...) i svaki od njih se koristi za aproksimiranje mreže igrajućih objekata. Moguće je dodati sudarač tako da se savršeno slaže sa mrežom igrajućeg objekta, ali tada bi gubili na performansi. Jedan bolji način za bolju aproksimaciju je korištenje konveksnih (*eng. Convex*) sudarača. Ovaj način dodaje mrežu sličnu modelu, ali izbaci neke dijelove.

Unity u svakom trenutku provjerava da li je sudarač imao koliziju sa nekim drugim igrajućim objektom koji isto ima svoj sudarač. Kada bi se oblik postavio da savršeno odgovara mreži igrajućeg objekta, tada bi trebalo obavljati previše računanja. U igri se koristi kvadratični sudarač za tijelo modela jer veoma dobro aproksimira automobil, a za kotače se koriste kolni sudarači.

3.12.4 Kolni sudarači

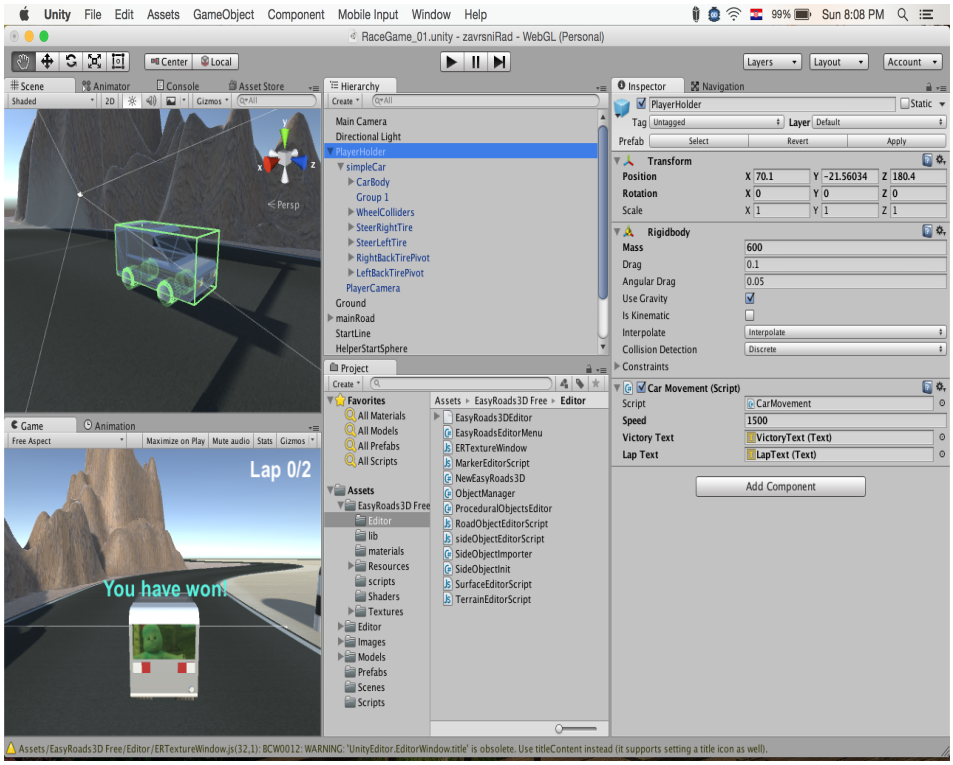
Unity ima predefinirane elemente za simulirati prave kotače prizemljenih vozila koje zovemo **kolne sudarače** (*eng. Wheel Colliders*). Prema dokumentaciji unity-a ovi sudarači se ne dodaju kao komponente, već trebaju biti komponente zasebnih igrajućih objekata. Znači za svaki kotač treba napraviti novi igrajući objekt, koji se zove prazni igrajući objekt (*eng. Empty GameObject*). Doda se komponenta praznom objektu i cijeli objekt se pomakne tako da je centriran sa kotačima.

Najvažnije metode za vožnju su moment motora (*eng. motor Torque*), moment kočnice (*eng. brake Torque*), te kut okretanja (*eng. steer angle*). Moment motora je sila koja djeluje na osovinu kotača izražena u Newton metrima. Predznak sile će odrediti smjer kretanja. Kut okretanja određuje za koliko će se okrenuti model prilikom skretanja, a moment kočnice određuje silu kočenja u Newton metrima. Primjer kôda za kretanje vozila se može vidjeti u ispisu 3

```
for(int i = 0; i < 4; i++)
   this.wheelsColliders[i].motorTorque = thrustTorque;
```

Ispis 1: Skripta za kretanje vozila

Jedan propust postoji kod ovih sudarača. Naime ukoliko se postave veće brzine kretanja model sam počima skretati prema desno. Prema forumima unity društva više je rješenja, iako nijedno od njih nije pomoglo u ovoj igri. Rješenje koje je primjenjeno u ovoj igri je optimizirana brzina.



Slika 2: Igrajući objekt

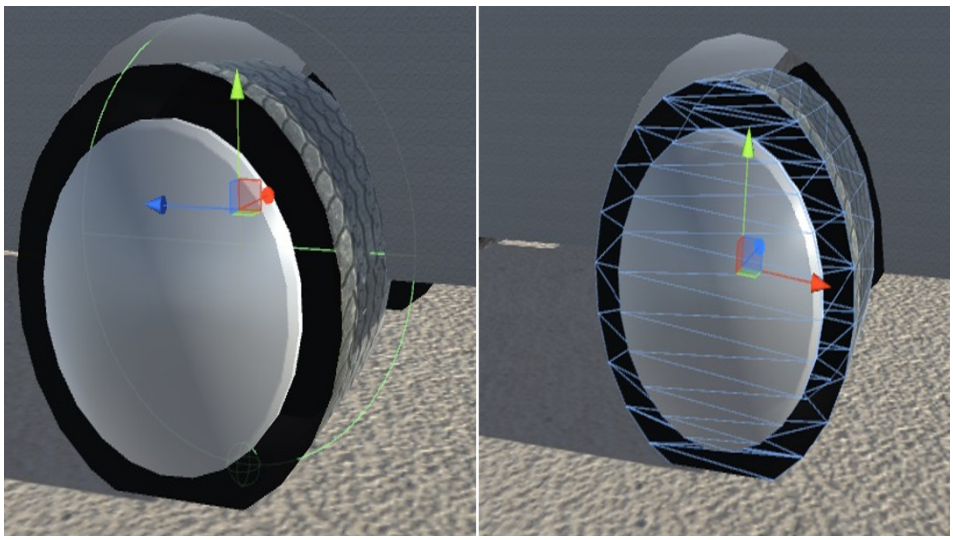
Na slici broj 2 se može vidjeti primjer unity radne okoline i jedan igrajući objekt koji se zove "PlayerHolder". Ovaj igrajući objekt ima tri komponente, transformaciju, kruto tijelo i skriptu koja se zove "CarMovement". O skriptama i kako one funkcioniraju u sljedećem poglavlju. Dodavanje komponenti se može klikom na dugme "Add Component". Nakon klika na dugme se otvara padajući izbornik za odabir željene komponente.

Svaki igrajući objekt pripada jednom sloju (*eng. Layer*) i ima oznaku (*eng. Tag*). Kasnije se pomoću skripti može manipulirati ovim elementima provjerom njihovih slojeva i oznaka. U igri se korisei oznake upravo za provjeravanje, da li je korisnik prošao cestu pravim putem. Na slici broj 4 se mogu vidjeti markeri koji provjeravaju navedeno.

3.13 Kotači

Kako je navedeno u prethodnom poglavlju kotači se sastoje od više komponenti. Te komponente su mreža (*eng. Mesh*), transformacija, te mrežni prevoditelj (*eng. Mesh Renderer*) koji dopušta korisniku da vidi konačni element. Ono što se koristi za kretanje modela je kolni sudarač. Mreža se može vidjeti na slici 3 desno, a sudarač na slici 3 lijevo.

Isto tako veoma bitna stvar je povećati masu modela, inače će početi nekontrolirano rotirati na sceni, kao da je upalo u crnu rupu. Vrijednost u konačnici definira kojom silom će gravitacija privlačiti model, odnosno definiramo brzinu. Što je masa veća sporiji je igrajući objekt i obrnuto.



Slika 3: Kotači modela

3.13.1 Rotiranje kotača

Prilikom vožnje automobila za bolju simulaciju potrebno je okretati i kola. Za isprogramirati ovu naizgled jednostavnu radnju više stvari treba unaprijed biti dobro definirano, inače se stvar komplicira. Ukoliko je igrajući objekt loše definiran i pivoti nisu dobro postavljeni na kotačima, zbog mehanike koju unity koristi objekti rotiraju krivo ili treba koristit naprednije metode koje povećavaju broj linija koda i stvaraju dodatne probleme.

Rotacija kotača bi trebala biti oko njegove osi, te se zato mora postaviti pivot u sredinu kotača. Ovo se obavlja tijekom izrade samog modela, te treba paziti na to prije unosa u unity. Ako se pivot nije centrirao prilikom izrade, onda se treba obavljati popravljanje. Koraci za popravljanje:

1. Pronaći objekt (kotač) unutar hijerarhije
2. Napraviti novi prazni objekt na istoj razini kao i kotač
3. Kotač ubaciti u prazni objekt

Sada za okretanje kotača se koristi novi prazni objekt jer je njegov pivot centriran. Ovakav pristup je jako učestal zbog dizajnera koji ne paze na pivote unutar svojih modela. Skripta za okretanje kola se može vidjeti u ispisu 2:

```
for (int i = 0; i < 4; i++)
     this.tirePivots[i].transform.Rotate(
        Vector3.forward,
        this.speed * move * Time.deltaTime
     );
```

Ispis 2: Skripta za okretanje kola

3.13.2 Problemi u zavojima

Prilikom vožnje vozila dodavanjem samo kolnih sudarača ne bismo dobili potpunu imitaciju pravog vozila. Jedan od problema koji se javlja je preokretanje u zavojima. Ova pojava je sasvim opravdana i nije nikakva greška programa. Što se ustvari događa? Kada vozilo uđe u zavoj i započme skretanje, na njega djeluje centrifugalna sila, podigne se prednji kotač sa tla i kako ne postoji protusila preokrene se.

Za ovaj slučaj postoji više rješenja, a onaj koji se koristi u igrici je sljedeći. Pri skretanju se provjeri koji kotač se podiže od tla i na njega se primjeni sila koja djeluje u smjeru gravitacije. Na ovaj način se smanji utjecaj centrifugalne sile, te samim time teže preokrenuti vozilo.

4 Skriptiranje

Skriptiranje (*eng. scripting*) je jedan od osnovnih dijelova svaki igre jer svaku igru je potrebno definirati određena pravila, te nekakav mehanizam koji će motriti sve igrajuće objekte i provjeravati da li se drže tih pravila. Skripte isto tako omogućavaju stvaranje grafičkih efekata korištenjem ugrađenih metoda ili mijenjanje same fizike igre tokom igranja.

4.1 Izrada i korištenje skripti

Izrada skripti se može na više načina. Najjednostvniji način je preko botuna za dodavanje komponenti igrajućem objektu, te odabirom nova skripta (*eng. new script*). Na ovaj način istovremeno se napravi skripta i pridruži igrajućem objektu. Kako je već navedno unity dopušta pisanje skripti u dva jezika:

- C# - industrijski standard, jezik koji je veoma sličan Javi ili C++
- Unityscript - jezik baziran na Javascriptu

4.2 Sadržaj skripti

Otvaranjem novoizrađene skripte može se vidjeti sadržaj. Unity omogućava da programer sam odabere program za otvaranje skripti, ali zadani program je MonoDevelop. U ispisu 3 je primjer C# skripte.

```
using UnityEngine;
using System.Collections;

public class MainPlayer : MonoBehaviour {

   void Start () {

   }

   void Update () {

   }
}
```

Ispis 3: Primjer skripte

Start metoda se koristi za incijalizaciju varijabli prilikom pokretanja programa. Ova metoda **nije konstruktor**, već unity preuzima odgovornost za sve konstruktore. Velika greška bi bila definiranje specijalnih konstruktora i vrlo vjerojatno ne bi ništa radilo.

Druga metoda koja je generirana se pokreće za broj slika u sekundi (*eng. frame per second fps*). Zanimljivo je da postoje tri varijacije ove metode.

- Update
- FixedUpdate - koji bi se trebao koristiti, ukoliko igrajući objekt sadrži kruto tijelo
- LateUpdate - se pokreće nakon svih drugih update funkcija, a korisna je za mijenjanje pozicije kamere jer se trebaju prvo pomaknuti svi objekti, a tek onda kamera.

Kod sve tri metode se često upotrebljava varijabla *Time.deltaTime*, koja je decimalna vrijednost vremena između svake slike. Ukoliko bi se željelo isprogramirati da se nešto kreće dvadeset metara po sekundi, tada se ova varijabla samo pomnoži sa brojem 20.

4.3 Pokretanje igrajućih objekata

Ako se napravi skripta preko *Assets ¿ Create ¿ C# Script*, tada prilikom pokretanja igra se skripta neće izvršavati jer nije pridružena igrajućem objektu. Samo skripte koje su pridružene igrajućim objektima će se pokretati. Za provjeru da li funkcionira skripta se može koristiti naredba za debugiranje *Debug.Log("Skripta radi!")*. Kako je navedeno igrajući objekti su samo kontenjeri za komponente, što znači da ukoliko je želja upravljati igrajućim objektima potrebno je upravljati komponentama. Primjer dohvaćanja komponenti se može vidjeti u ispisu 4.

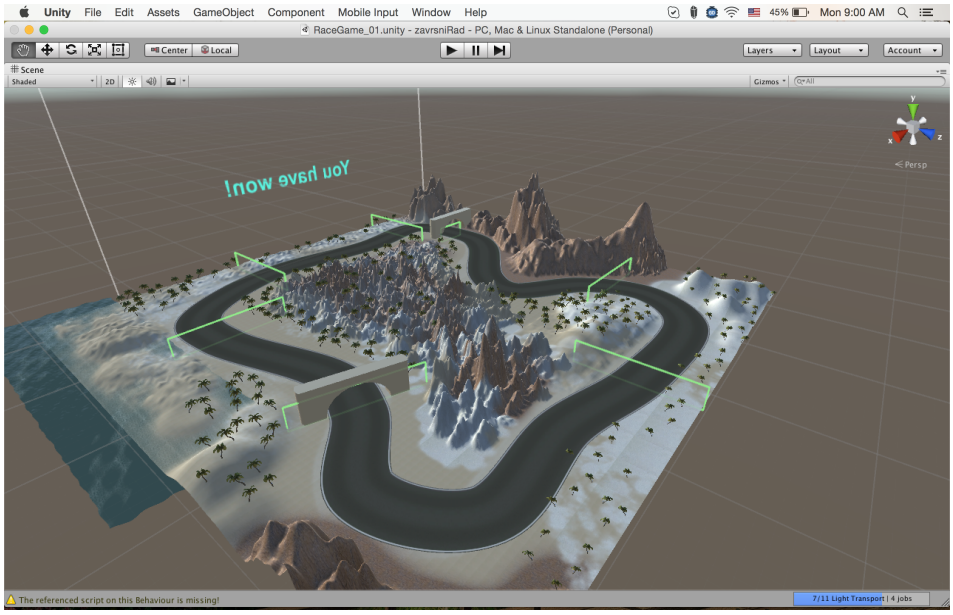
```
for (int i = 0; i < 4; i++) {
  this.tires[i] = GameObject.Find(this.tireNames[i]);
  this.tirePivots[i] = GameObject.Find(this.tirePivotNames[i]);
}
```

Ispis 4: Dohvaćanje objekata

4.4 Funkcije događaja

Skripte u unity-u nisu kao u drugim programskim jezicima da se vrte u beskonačnoj petlji dok ne završe dani zadatak. Unity funkcionira tako da prebacuje kontrolu funkcijama koje se trebaju izvršiti, koje nakon izvršavanja vrate kontrolu natrag unity-u. Ovakve funkcije se zovu funkcije događaja (*eng. Event functions*). Neke od najučestalijih koje se koriste su:

- Regularna ažuriranja događaja (*eng. Regular update events*) - sve funkcije koje se pokreću za animiranje ili manipuliranje likovima (Update, LateUpdate, FixedUpdate)
- Inicijalizacijski događaji (*eng. Initialization Events*) - funkcije koje se pokreću na početku pokretanja unity-a. Spomenuta je Start, ali postoji i buđenje (*Awake*). Awake se poziva za svaki igrajući objekt prije nego što se scena napravi. (Awake se pokreće prije Start)
- Fizički događaji (*eng. Physics Events*) - sve funkcije koje se pokreću kada se dogodi kolizija igrajućih objekata



Slika 4: Markeri putanje

4.4.1 Razlike fizičkih funkcija događaja

Unity razlikuje dva različita tipa ovih funkcija. One koje se pozivaju prilikom kontakta sa igrajućim objektom i one koje imaju na sudaračima označeno da je okidač (*eng. Trigger*). Svaka od ovih prima parametar, a isto tako se razlikuju i po tome. Imena funkcija moraju biti točno napisana i njihovi potpisi moraju biti točni, inače se neće pozivat i unity će izbaciti pogrešku.

Kolizijske funkcije (OnCollisionEnter, OnCollisionStay, OnCollisionExit) imaju potpis koji izgleda kao ispis 5. Ove metode se pozivaju kada se dva tijela koji imaju kruto tijelo ili sudarač, a sadrže informaciju o mjestu kontakta i brzini sudara.

```
void OnCollisionEnter(Collision col);
```

Ispis 5: Potpis kolizijskih funkcija

Funkcije koje se pokreću na okidač (OnTriggerEnter, OnTriggerStay, OnTriggerExit) imaju kao ulazni parametar sudarač, pa je moguće više informacija saznati iz njega. Isto tako se pozivaju kada se dva tijela sudare, ako barem jedno tijelo ima svoj sudarač označen kao okidač. Potpis ove funkcije i primjer kako izgleda u igri se može vidjeti u ispisu 6

```
void OnTriggerEnter(Collider col) {
  if (col.gameObject.layer == LayerMask.NameToLayer ("RoadMarkers"))
    this.checkTheRightMarker (col.ToString ().Split (' ') [0]);
}
```

Ispis 6: Potpis i primjer trigger funkcije

Na slici 4 se mogu vidjeti posebni markeri koji su korišteni za provjeravanje da li je automobil prešao sve markere i je li išao pravim smjerom. Ovo su obični igrajući objekti sa sudaračima kojima je označeno da su okidači. Ako se sudaraču postavi da je okidač, onda će se kroz njega moći prolaziti.

4.5 Generičke funkcije

Zbog jednostavnijeg načina dohvaćanja pojedinih komponenti igrajućih objekata postoje generičke funkcije. To su funkcije koje nije potrebno eksplicitno baciti u drugi tip, već prilikom dohvaćanja definiramo tip. Primjer se može vidjeti u ispisu 7

```
this.carBody = GetComponent<RigidBody>();
```

Ispis 7: Primjer generičke funkcije

5 Korisničko sučelje

Svaka igra mora imati svoje korisničko sučelje (*eng. User Interface UI*) gdje će korisnici moći mijenjati pojedine opcije igre. Unity za ovo pruža poseban igrajući objekt koji se zove platno (*eng. Canvas*). Ukoliko platno već nije izrađeno, onda prilikom izrade teksta za sučelje će unity sam napraviti i platno i tekst kao dijete platna. Dakle sve objekte koji pripadaju korisničkom sučelju moraju biti dijete platna u hijerarhiji.

5.1 Platno

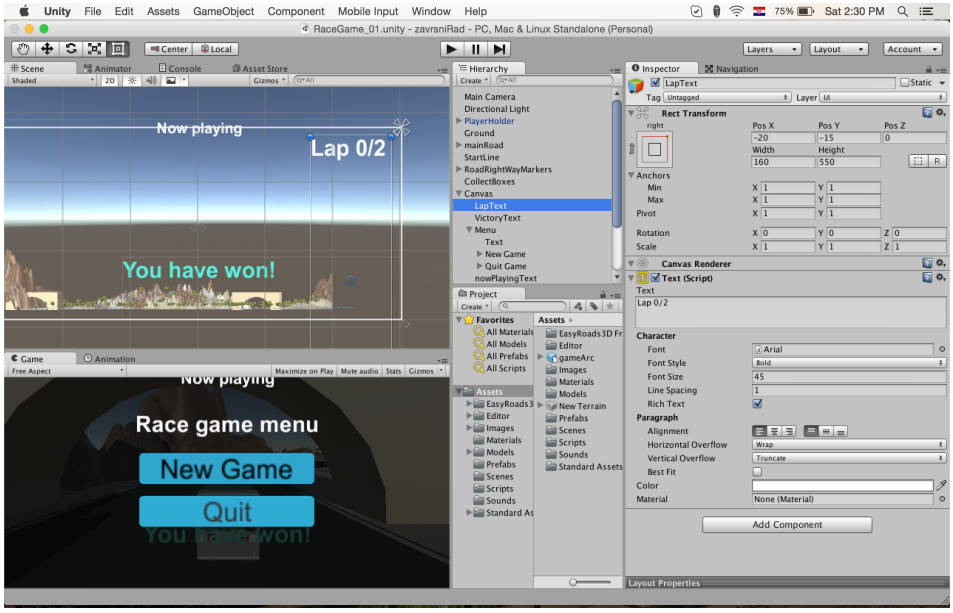
Sva platna su ustvari 2D objekti kojima je dodana dubina samo boljeg prikaza u 3D prostoru. Prebacivanjem u 3D pogled će se moći vidjeti da platna izgledaju kao običan list papira na kojima je nekakav tekst, botun ili slika. Igrajući objekti koji su dio platna će se prikazivati prema njihovom položaju u hijerarhiji. Dakle ako imamo više igrajućih objekata, onda će se prvo nacrtati prvo dijete i svi igrajući objekti koji njegova djeca, zatim će drugo dijete i tako dalje. Platno također ima različite modove izrade (*eng. Render modes*).

5.1.1 Raspored elemenata

Pod rasporedom (*eng. Layout*) se misli pozicija pojedinih elemenata unutar platna. Dakle dimenzije i odnos prema drugim elementima. Sidrišta (*eng. Anchors*) imaju jako veliku pri odnosu pojedinih elemata, a o njima će više biti rečenu u nastavaku. Veličina platna se može mijenjati korištenjem alata *Rect Transform*, koji se ponaša kao pravokutnik. Uglavnom se koristi za 2D objekte, ali naravno može i za 3D objekte. Isto tako se može mijenjati pozicija elemenata jednostavnim označavanjem i povlačenjem na željenu lokaciju. Potrebno je razlikovati prilikom mijenjanja veličine objekta da postoje dvije različite transformacije koje se čine jednake. Ako se koristi alat *Rect Tool* označen objekt tada mijenja svoju veličinu (*eng. Scale*), a ako je selektiran *Rect Transform* mijenjaju se dimenzije objekta (širina i visina).

5.1.2 Sidrišta

Sidrišta su četiri trokutasta elementa koja definiraju točku od koje će se kretati elementi i koju će pratiti za promjene. Zanimljiva funkcionalnost koja omogućavaju da element mijenja dimenzije ovisno o promjeni dimenzija elemenata na koji je pridružen, ako ujedno i je pridružen. Moguće je na drugi element postaviti da sidrište prati visinu odabranog elementa, te će se trenutni element mijenjati relativno s obzirom na pridruženi. Na slici 5 je moguće vidjeti tekstualni element kojemu je sidrište postavljeno da prati gornji desni kut roditelja. Na ovaj način će se tekst uvijek pomicati relativno sa gornjim desnim kutom.



Slika 5: Sidrište

Sa desne strane je moguće vidjeti sve komponente koje ima element, te standardne opcije kao što su x, y i z transformacija unutar roditelja, visina i širina, minimalna i maksimalna vrijednost sidrišta i veličina. Tekstu možemo definirati oblik, veličinu, poravnavanje i boju.

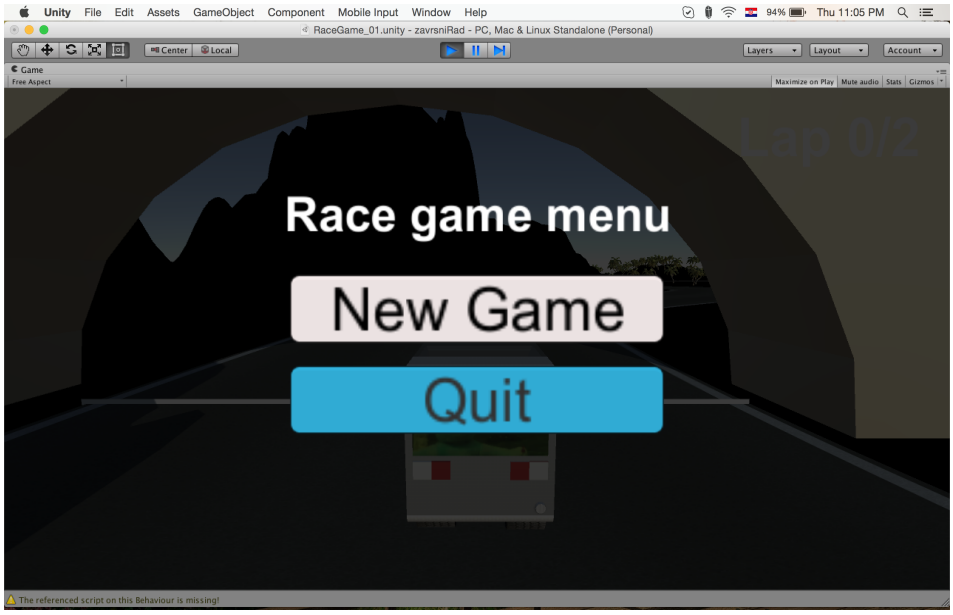
5.1.3 Modovi izrade

Postoje dvije glavne podjele ovih modova:

- Pozicija zaslona (*eng. Screen Space*) - platno se pozicionira prema poziciji zaslona.
  - Prekriti (*eng. Overlay*) - svi objekti se izrađuju na platnu, te mijenjaju svoju veličinu ovisno o veličini platna
  - Kamera - svi objekti se izrađuju ovisno o postavkama kamere. Ako se na kameri promijeni perspektiva, tada će se i sami izgled sučelja mijenjati ovisno o udaljenosti, kutu, te svim ostalim parametrima koji su postavljeni.
- Pozicija svijeta (*eng. World Space*) - platno se ponaša kao svaki drugi igrajući objekt. Moći će se postaviti u pozadini iza nekih drugih igrajućih objekata ili slično. Primjer bi bio prikaz teksta koji se mijenja na nekakvom računalu.

5.2 Glavni izbornik

Glavni izbornik se sastoji od tri igrajuća objekta, jednog teksta i dva botuna. Izborniku je pridružena i skripta koja kontrolira prikaz i funkcionalnost botuna. Koristi se od unity-a ugrađena funkcionalnost za pozivanja metoda unutar skripte. Svaki botun ima okidač na klik, te se samo treba postaviti metoda koja se želi pozvati. U igri su to dvije metode započmi novu igru (*StartNewGame*) i izađi iz igre (*StartNewGame*). Izgled izbornika se može vidjeti na slici 6.
Pauziranje same igre se obavlja postavljanjem razlike vremena između slika (*Time.deltaTime*) na 0. Na ovaj način se ništa ne kreće i dobije se dojam da je igra pauzirana. Nova igra i izlazak iz igre se izvršavaju preko klase aplikacija (*eng. Application*) koja se isto može koristiti za pokretanje novog levela igre.



Slika 6: Glavni izbornik

5.3 Tekst objekti

Tokom igranja se može u gornjem desnom kutu vidjeti tekst, koji pokazuje na kojem smo trenutno krugu, u gornjem lijevom kutu bodove koje je igrač ostvario, a prilikom završetka igre se prikazuje tekst koji govori da je igrač uspješno odvozio i ispisuje ukupne bodove koje je igrač sakupio. Ovo su ustvari tri različite tekstualna objekta preko čije se reference može mijenjati tekst iz skripti.

6 Zvuk

Teško je danas zamisliti igru bez ikakvih zvukova osim ako ciljano nije dizajnirana na taj način. U stvarnom svijetu svaka osoba različito čuje određene zvukove. Stariji ljudi ne čuju više frekvencije kao i mladi, osoba koja sluša glazbu preko računala na udaljenosti neće isto čuti kao i osoba koja stoji odmah do računala. Svi ovi faktori su imitirani unutar unity-a. Posebne metode provjeravanja pojedinih izvora zvuka, te ambijenta u kojem se nalazi omogućuju da igrač drugačije zvuk ovisno o svojoj poziciji. Formate koje unity podržava je moguće vidjeti u tablici 6.

Format	Extensions
MPEG layer 3	.mp3
Ogg Vorbis	.ogg
Microsoft Wave	.wav
Audio Interchange File Format	.aiff / .aif
Ultimate Soundtracker module	.mod
Impulse Tracker module	.it
Scream Tracker module	.s3m
FastTracker 2 module	.xm

Tablica 1: Tablica formata

Od unity verzije 5.0 unutar unity-a sami zvuk i zvukovna datoteka su odvojeno pohranjene. Zvuku se može pristupati preko skripti korištenjem komponente *Audio Clip*, koju je onda moguće odsvirat. Jedna od najćešćih grešaka prilikom dodavanja zvukova je ostavljanje sviranja na početku (*eng. play on awake*). Ako se ova opcija ne ugasi onda će se prilikom pokretanja igre početi svirati svi zvukovi kojima nije ovo isključeno. Na mobilnim uređajima audio komponente su komprimirane u mp3 format zbog bržeg načina dekompresije. Znimljiva komponenta je zvučni osluškivač (*eng. Audio Listener*)koji se koristi za snimanje zvukova unutar unity-a.

6.1 Zvučni osluškivač

Ova komponenta omogućava igračima pružiti iskustvo kao da su doista oni na pozicijama igrajućih objekata. Kroz ovu igru nije bilo potrebe za korištenje ove komponente na posebnim načinima jer je već predefinirano pridružena kameri. Potrebno je poznavati kako se zvuk ponaša u pojedinim ambijentima (*Reverb*) jer unity ima mogućnost simuliranja pojedinih okruženja ukoliko je to potrebno preko osluškivača.

Zanimljivo je da ukoliko se napravi izvor zvuk kao 2D objekt, tada neće paziti na lokaciju odakle izvire zvuk već će se svi ponašati kao da su globalni zvukovi i igrač će ih čuti jednako. Ukoliko se napravi da je izvor zvuka 3D objekt, onda će trebati podešavati jačinu, orijentaciju i lokaciju.

6.2 Postavke

Postavke (*eng. Settings*) je moguće mijenjati prije pokretanja igre ili čak tokom igre korištenjem klase *AudioSettings*. Nije praktično baš mijenjati postavke tokom igre jer će to uzrokovati ponovno učitavanje svih zvukova koji se koriste, te lošim performansama. Najbolje je prije početka ukoliko ima potrebe podesiti sve postavke. Ako je neki zvuk napravljen tokom igranja, promjena postavki će izbrisati taj zvuk i trebati će ga ponovno instancirati.

7 Klase

Svaka klasa koja se definira nalazi se unutar svoje zasebne skripte. Moguće je nasljeđivanje iako u ovoj igri nije bilo potrebe. Primjer klase B koja nasljeđuje klasu A je moguće vidjeti u ispisu 8. Potrebno je napomenuti da klase koje nasljeđuju *MonoBehavior* ne smiju imati svoje konstruktore, ali klase koje ne nasljeđuju *MonoBehavior* moraju imati svoj konstruktor.

```
using UnityEngine;
using System.Collections;

public class B : A {
  public B (string carType) {

  }
}
```

Ispis 8: Primjer nasljeđivanja

Ukoliko klasa nasljeđuje već nasljeđenu klasu onda je moguće definirati isto ulazne parametre za baznu klasu preko ključne riječi *base*. Primjer korištenja ove naredbe se može vidjeti u ispisu 9. Klasa C nasljeđuje gore definiranu klasu B i prosljeđuje joj ulazni argument tipa string ("mazda"). Ako se ne definira ulazni argument onda će se pozvati predefinirani konstruktor koji ne prima nijedan argument.

```
using UnityEngine;
using System.Collections;

public class C : B {
  public C () : base("mazda") {

  }
}
```

Ispis 9: Primjer nasljeđivanja nasljeđene klase

Kako trenutna igra nije toliko složena ne treba joj previše klasa. Sve klase koje su korištene unutar igre, kao i njihov opis je moguće vidjeti ispod. Svaka klasa je navedena kao i igrajući objekt kojem je pridružena, a detaljan opis se nalazi u podpoglavljima pojedine klase.

7.1 Klasa CarMovement

Glavna klasa koja se koristi za praćenje trenutnog kruga, koliko je bodova igrač skupio i kretanje. Pridružena je igrajućem objektu *PlayerHolder*, koji kao svoju djecu ima sve igrajuće objekte vezane uz vozilo. Kada se pokrene igra pozove se *Start()* funkcija koja dohvaća sve reference potrebne za pokretanje vozila, mijenjanje teksta i provjeravanje trenutnog kruga. Svakom promjenom slike se poziva *FixedUpdate()*, gdje se provjerava svaki unos od igrača i pozivaju druge privatne funkcije. Funkcije koje se koriste:

- (public void) Start() - sve reference se inicijaliziraju, prilikom pokretanja skripte
- (public void) MoveCar(float steer) - pokretanje vozila se izvršava pozivajući ovu funkciju. Ovisno o predznaku varijable *steer* će se vozilo kretati naprijed ili natrag
- (private void) RotateTheTires(float movement) - pomoćna funkcija koja okreće kotače. Ovo okretanje kotača nije povezano sa funkcionalnošću kretanja, već je samo zbog izgleda
- (private void) ApplyBrake() - funkcija koja zakoči vozilo
- (private void) centripetalForce() - pomoćna funkcija koja ne dopušta vozilu da se preokrene u zavojima. Ukoliko se jedan kotač podigne s tla, onda ova sila djeluje na taj kotač određenom silom kako se ne bi prevrnio
- (private void) updateTexts() - funkcija za mijenjanje teksta u korisničkom sučelju i za aktivaciju kutija pri prolasku kruga
- (private void) SteerCar(float steer) - pomoćna funkcija za skretanje vozila lijevo i desno. Ovisno o predznaku varijable *steer* će vozilo skretati u određenu stranu
- (private void) resetMarkers() - funkcija za resetiranje markera, koji se koriste za provjeravanje na koji način je igrač došao do cilja. Ovo se radi zbog novog kruga, inače bi igrač mogao proći samo jednom krug i stajat na istoj lokaciji dok mu ne dođe poruka da je pobjedio
- (private void) checkIfLap() - funkcija koja provjerava sve markere i provjeri da li je igrač prošao cijeli krug

- (private void) checkTheRightMarker(string marker) - ova funkcija postavi marker koji je igrač prošao na istinitu vrijednost, a zna točno koji treba jer se proslijedi iz *OnTriggerEnter()* funkcije
- (private void) activateTheBoxes() - pomoćna funkcija koja prošeta po nizu kutija i aktivira svaku od njih
- (private void) collectBox(GameObject go) - kada igrač prođe kroz jednu od kutija se pozove *OnTriggerEnter()* i preko nje se proslijedi prava kutija koja je skupljena. Ova funkcija samo ugasi taj igrajući objekt i promijeni bodove korisnika
- (void) OnTriggerEnter(Collider col) - funkcija događaja koja čeka za pozivanje već navedenih funkcija i prosljeđivanje pravih varijabli.
- (void) FixedUpdate() - funkcija koja se poziva za svaku sliku i provjerava da li je korisnik pritisnuo određene tipke za kretanje, skretanje ili kočenje

Reference se sve nalaze unutar privatnih ili javnih varijabli, koje se zovu podatkovni članovi. Ti članovi su:

- (private float) thrustTorque - broj koji definira moment motora
- (public float) speed - varijabla koja definira brzinu automobila
- (private RigidBody) carBody - varijabla koja sadrži kruto tijelo automobila
- (private bool[]) markersPassed - pomoćni niz boolean varijabli za provjeru markera jesu li prođeni ili ne
- (private int) lapsPassed - trenutni krug u kojem se igrač nalazi
- (private Vector3) lastPos - zadnja pozicija automobila koja sadrzi (x, y, z) koordinate
- (private int) points - bodovi igrača
- (public Text) victoryText - referenca na tekst koji pokazuje, ako je korisnik pobjedio

- (public Text) lapText - referenca na tekst trenutnog kruga
- (public Text) pointsText - referenca na tekst bodova
- (private string[]) tireNames - pomoćni niz koji se koristi za dohvaćanje referenci kola
- (private string[]) tirePivotNames - pomoćni niz koji se koristi za dohvaćanje pivota kotača
- (private string[]) wheelColliderNames - pomoćni niz koji se koristi za dohvaćanje kolnih sudarača
- (private WheelCollider[]) wheelColliders - reference na kolne sudarače
- (private GameObject[]) steerWheels - reference na dva kotača koja se zakreću prilikom skretanja
- (public float) antiRoll - sila koja se dodaje da se ne okrene automobil
- (private GameObject[]) tires - reference na kotače
- (private GameObject[]) tirePivots - reference na pivote kotača
- (private GameObject[]) collectableBoxes - reference na kutije koj se skupljaju tokom igranja
- (private AudioSource) collectSound - referenca na zvuk koji se čuje kada se skupi kutija

7.2 Klasa MenuController

Klasa koja se koristi za kontrolu glavnog izbornika i pauziranje cijele igre. Skripta je pridružena igrajućem objektu *Menu* koja ima pristup svim igrajućim objektima unutar izbornika. Početna funkcija koja se koristi je *Awake* koja se poziva prije *Start()* funkcija. Funkcije koje se koriste:

- (void) Awake() - funkcija koja se prva pokreće i postavlja referencu za platno unutar podatkovnog člana *platno*
- (private void) HandlePauseGame() - funkcija koja se poziva pritiskom tipke "esc", te ovisno o trenutnom stanju otvara ili zatvara glavni izbornik
- (void) Update() - funkcija koja se pokreće svaku sliku i koja provjerava da li je korisnik pritisnuo tipku "esc". Provjera se izvršava preko *Input.GetKeyUp(KeyCode.Escape)* jer bi se inače pozivalo svaku sliku i ne bi bilo učinkovito
- (private void) StartNewGame(int scene) - funkcija koja se poziva preko klika na dugme unutar glavnog izbornika. Funkcija je povezana zahvaljujući unity-u, pa ne treba posebno unutar skripte provjeravati. Naredba *Application.LoadLevel(scene)* se koristi za ponovno pokretanje igre jer je scena koja se pokreće početna scena igre
- (private void) QuitGame() - isto tako funkcija koja se poziva klikom na dugme unutar izbornika. Izlaz iz igre je ostvaren preko *Application.QuitGame()*

Ova klasa nema potrebe za previše podatkovnih članova jer su svi djeca trenutnog igrajućeg objekta i nema ih toliko mnogo. Podatkovni član je:

- (private Canvas) Menu - sadrži referencu na komponentu platna igrajućeg objekta *Menu*

7.3 Klasa SoundEffects

Ova klasa je isto tako pridružena igrajućem objektu *PlayerHolder*, a koristi se za kontroliranje glazbe unutar igre. Omogućava da klikom na tipku "X" je moguće mijenjati glazbu koja se čuje u pozadini tokom igranja. Početna funkcija je isto tako *Awake()* jer je potrebno učitati sve reference za glazbu prije samog pokretanja igre i sviranja te glazbe. Funkciju *Update()* koristimo za provjeravanje klika na dugme i mijenjanje glazbe. Funkcije su:

- (void) Awake() - početna funkcija koja dohvaća sve reference za glazbu i postavlja ih u podatkovne članove. Postavlja se indeks glazbe koja se sluša kako bi se kasnije znalo na koju glazbu prebaciti ako se pritisne tipka za mijenjanje glazbe
- (private void) playNextSong() - funkcija koja se poziva iz *Update* i koja mijenja glazbu kroz igru. Trenutni indeks provjerava da li je to zadnja pjesma unutar niza i poveća se ili postavi na nulu. Isto tako aktivira tekst kako bi igrač mogao znati koja trenutno glazba svira
- (private void) hideNowPlayingText() - pomoćna funkcija koja se pozove nakon nekog vremena da se sakrije tekst koji pokazuje koja glazba trenutno svira
- (void) Update() - funkcija koja se poziva svaku sliku i provjerava da li je korisnik pritisnuo tipku za promjenu glazbe ili ne. Isto tako ako korisnik drži tipku prema dolje se svira zvuk kamiona koji ide unatrag

Podatkovni članovi sadrže reference na glazbu i indeks glazbe koja trenutno svira, a oni su:

- (private AudioSource) reverse - sadrži referencu na snimku kamiona koji se kreće unutarga
- (private AudioSource[]) music - sva glazba koja se koristi unutar igre je u ovom nizu
- (private string[]) songNames - imena svih pjesmi koje sviraju tokom igre. Koriste se za tekst, kako bi igrač mogao točno vidjeti koja pjesma svira
- (private int) playingIndex - indeks trenutne pjesme koja svira

- (private Text) nowPlaying - referenca na tekst koji prikazuje koja pjesma trenutno svira
- (privat float) timeToHidePlayingText - vrijeme nakon kojeg se sakrije tekst koji pokazuje koja pjesma trenutno svira
- (private bool) changedSong - varijabla koja prati je li promijenjena pjesma ili nije kako bi se znalo da li treba sakriti ili ne treba sakriti tekst

7.4 Klasa RotateBoxes

Klasa koja je pridružena praznom igrajućem objektu *CollectableBoxes*, koji je roditelj svih kutija. Ovo je samo pomoćna klasa i koristi se samo zbog rotacije kutija zbog ljepšeg prikaza. Ima samo dvije funkcije *Start()* i *Update()*. Nakon svake slike se kutije okreću unutar *Update()*, a na početku se samo dohvaćaju sve reference od kutija. Podatkovni članovi su:

- (private int) numBoxes - broj kutija. Potrebna informacija jer se prolazi preko svake kutije i rotira
- (private GameObject[]) boxes - niz referenci na kutije. Šetajući preko ovog niza rotiramo svaku kutiju zasebno unutar funkcije *Update()*

8 Zaključak

Prvih tjedan dana bilo koje tehnologije programerima je najteži period i u tom vremenu će se moći vidjeti hoće li naučiti raditi s tom tehnologijom ili ne. Upoznavanje sa unity-om je veoma lako zbog brojnih video zapisa u kojima je detaljno objašnjen svaki dio. Ukoliko postoji želja za učenjem ove tehnologije i ustrajanost od godinu dana, programer će moći napraviti koju god igru poželi.

Za naučiti raditi igre u unity-u, a i za programiranje nije dovoljno samo znati programirati, već i želja za stvaranjem. To bi trebala biti nekakva umjetnost gdje se svaki problem može riješiti na više načina, a na programeru je da svojim stilom dođe do najboljeg. Proces izrade igre je veoma kompleksan ako se radi o većoj igri. Osobe koje vole znati *kako* nešto funkcionira bi trebali biti programeri jer su upravo programeri ti koji kada apstraktiraju nešto moraju poznavati kako to funkcionira.

Istina je da bi programeri trebali uvijek učiti. Jednim dijelom jer se tehnologija uvijek mijenja, a drugim ako rade različite tipove apstrakcija, moraju poznavati taj novi sustav kako bi ga mogli apstraktirati.

9 Literatura

1. https://docs.unity3d.com/Manual
2. https://docs.unity3d.com/ScriptReference
3. http://forum.unity3d.com/threads/how-to-make-a-physically-real-stable-car-with-wheelcolliders.50643/