# Igra skijanja upravljana praćenjem lica

**Sršić, Marko**

SVEUČILIŠTE U ZAGREBU
**FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA**

ZAVRŠNI RAD br. 1346

# IGRA SKIJANJA UPRAVLJANA PRAĆENJEM LICA

Marko Sršić

Zagreb, lipanj 2024.

SVEUČILIŠTE U ZAGREBU
**FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA**

ZAVRŠNI RAD br. 1346

# IGRA SKIJANJA UPRAVLJANA PRAĆENJEM LICA

Marko Sršić

Zagreb, lipanj 2024.

**SVEUČILIŠTE U ZAGREBU**
**FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA**

Zagreb, 4. ožujka 2024.

# ZAVRŠNI ZADATAK br. 1346

| | |
|---|---|
| Pristupnik: | **Marko Sršić (0036539814)** |
| Studij: | Elektrotehnika i informacijska tehnologija i Računarstvo |
| Modul: | Računarstvo |
| Mentor: | prof. dr. sc. Igor Sunday Pandžić |

Zadatak: **Igra skijanja upravljana praćenjem lica**

Opis zadatka:

Osnovna ideja ovog zadatka je igra skijaške utrke veleslaloma, gdje igrač upravlja modelom skijaša i nastoji završiti utrku prolaskom kroz vrata u što kraćem vremenu. Posebna značajka ove igre je inovativno korištenje praćenja i analize ljudskog lica za upravljanje igrom. Vaš zadatak je osmisliti, implementirati i testirati opisanu igru s posebnim naglaskom na specifične funkcije praćenja lica. Kao osnova za izvedbu, na raspolaganju Vam je sustav za praćenje lica koji prati poziciju i orijentaciju glave, položaje ključnih točaka lica, otvorenost očiju i smjer pogleda. Svu potrebnu literaturu i uvjete za rad osigurat će Vam Zavod za telekomunikacije.

Rok za predaju rada: 14. lipnja 2024.

*A, I, J, Z <3*

# Sadržaj

# Uvod

Od kada znam za sebe, uvijek sam na neki način bio uključen u svijet računalnih igara. Kada bi mi netko pokazao koliko sam sati proveo igrajući ili gledajući druge ljude kako igraju video igre, vjerojatno bih ispao iz cipela. Još od kada sam bio dijete fascinirali su me ti kompleksni i interaktivni digitalni svjetovi, a u pozadini je uvijek postojala i želja da naučim kako ti svjetovi nastaju. Ovaj završni rad predstavlja ispunjenje te želje iz djetinjstva kroz prikaz razvoja skijaške računalne igre uz inovativno korištenje praćenja lica, nazvane „SKI-FER“.

Razvoj računalnih igara u posljednjim godinama jedna je od najpopularnijih i najunosnijih grana računarske znanosti. On objedinjuje područja dizajna i umjetnosti, programiranja, glazbe, pa čak i psihologije, koja zajedno omogućavaju stvaranje složenih virtualnih svjetova. Cilj ovog rada je prikazati cjelokupni razvoj igre, od početne ideje do konačne implementacije, uključujući sve korake i tehnologije korištene u razvoju. Također, u ovom radu bit' će prikazano kako jedna napredna tehnologija poput praćenja i analiza pokreta lica i glave može postati integralni dio neke igre.

Prvo poglavlje fokusirat' će se na Unity, jedan od najpopularnijih alata za razvoj igara. Unity omogućuje izradu visokokvalitetnih 2D i 3D igara za različite platforme. Kroz ovo poglavlje bit' će prikazan osnovni pregled Unity okoline, njezinih funkcionalnosti i komponenata potrebnih za stvaranje igre.

U drugom poglavlju detaljno će se analizirati VisageSDK, alat specijaliziran za praćenje i analizu lica. On omogućuje precizno praćenje pokreta glave, što je ključna značajka za interaktivnost igre „SKI-FER“. Ovdje će biti objašnjeno kako VisageSDK funkcionira, koje su njegove mogućnosti te kako je integriran u Unity.

Treće poglavlje posvećeno je samom razvoju igre. Prikazat' će se potpuni proces iz-

rade igre: dizajn i razvoj skijaške staze, implementacije logike igre, uključivanje interaktivnih elemenata te integracija alata VisageSDK za praćenje pokreta lica i glave. Osim tehničkih dijelova, u ovom poglavlju bit' će vidljivi i raspravljeni izazovi s kojima sam se susretao tijekom razvoja te konačni rezultat – razvijena skijaška računalna igra upravljana praćenjem lica.

# 1. Unity

Pokretački sustavi za igre (engl. *game engine*) su programska sučelja visoke razine prvenstveno dizajnirani za razvoj računalnih igara. Oni pružaju skup vizualnih razvojnih alata uz ponovno iskoristive softverske komponente [1]. Jedan takav pokretački sustav je i Unity, razvijen od strane Unity Technologies i prvi put objavljen u lipnju 2005. kao pokretački sustav za igre za Mac OS X. Kroz godine je postupno proširen te sada podržava razne desktop, mobilne i konzole platforme, kao i platforme za proširenu i virtualnu stvarnost. Unity je posebno popularan za razvoj mobilnih igara za iOS i Android te se smatra jednostavnim za korištenje za programere početnike. Sustav se može koristiti za izradu dvodimenzionalnih i trodimenzionalnih igara te interaktivnih simulacija, a često je prihvaćen i u industrijama poput filma, automobilske industrije, arhitekture i drugih [2].

Poglavlje 1.1 donosi pregled Unity sučelja i njegove najvažnije prozore.

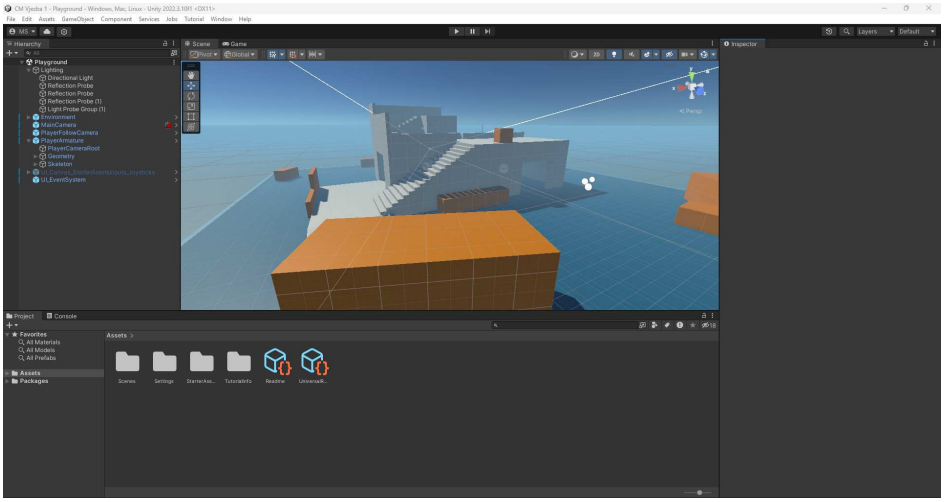
Poglavlje 1.2 objašnjava kako se definiraju 3D objekti u Unity-j i njihove najvažnije komponente.

Poglavlje 1.3 opisuje nekoliko koncepata koji su od iznimne važnosti prilikom razvoja igre – kretanje objekta (poglavlje 1.3.1), praćenje objekta kamerom (poglavlje 1.3.2), sudar (poglavlje 1.3.3).

Konkretna realizacija poglavlja 1.1, 1.2 i 1.3 po pitanju razvoja igre „SKI-FER“ bit će detaljnije objašnjena u poglavlju o razvoju igre.

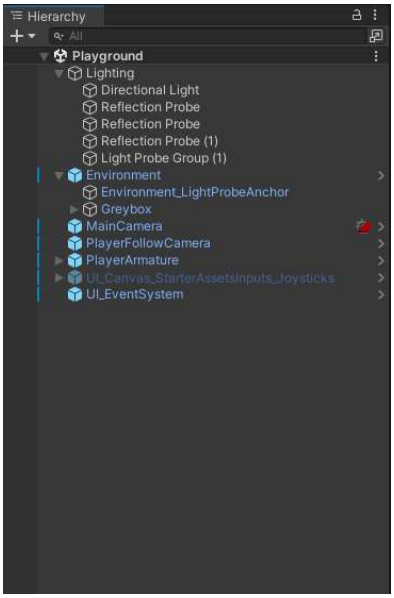
## 1.1. Pregled Unity sučelja

Osoba koja prvi put koristi Unity sigurno će imati problema sa snalaženjem u Unity sučelju zbog brojnih opcija i prozora koji su joj na raspolaganju. Ipak, svaki od tih prozora i opcija ima svoju ulogu i kao cjelina predstavljaju moćni alat za razvoj igara. Na slici 1.1 je prikaz sučelja Unity verzije 2022.3.10f1. Glavni dijelovi sučelja su *Hierarchy*, *Scene*, *Game*, *Inspector* i *Project*.



**Slika 1.1.** Unity sučelje

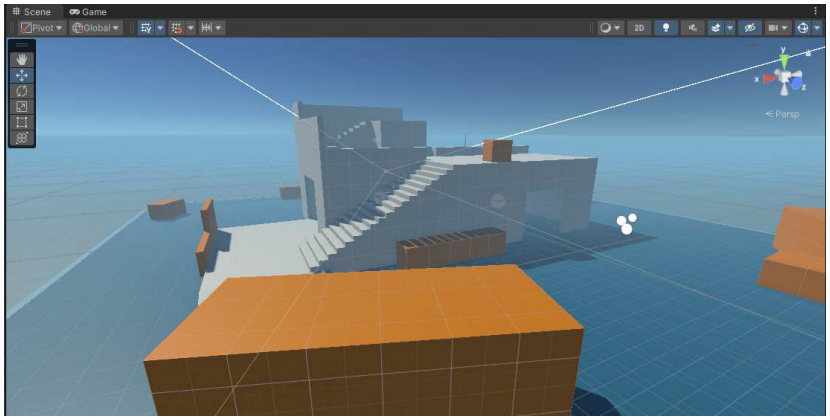
*Hierarchy* prikazuje sve objekte koji su trenutno na sceni. Ovaj prozor služi za pregledavanje i upravljanje scenom jer on omogućuje programeru da odredi hijerarhijski raspored objekata, što uključuje odnose roditelj-dijete između objekata. (slika 1.2)



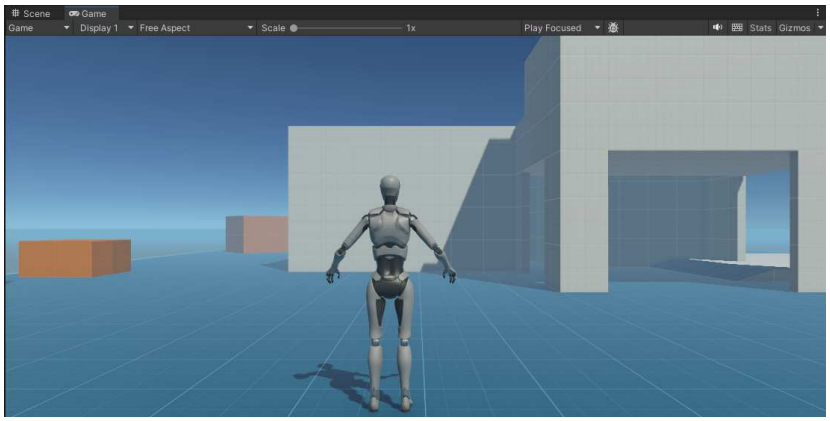

**Slika 1.2.** Hierarchy

*Scene* i *Game* donose pregled i interakciju s projektom. *Scene* je područje gdje se odvija proces uređivanja i izrade igre; ovdje se postavljaju, uređuju i pregledavaju svi objekti unutar scene. Primjeri objekata su kamera, izvor svjetlosti, izvor zvuka itd. (slika 1.3)

*Game* također prikazuje scenu, ali iz perspektive kamere - na taj način simulira kako će igra izgledati kada se pokrene. (slika 1.4)



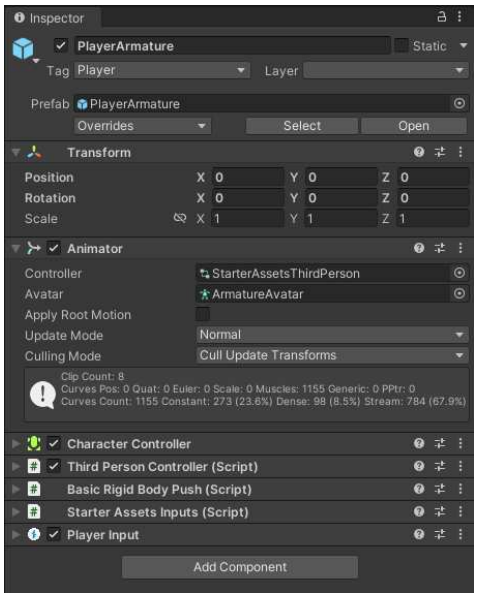

**Slika 1.3.** Scene




**Slika 1.4.** Game

Kada odaberemo objekt u *Hierarchy* ili *Scene* prozoru, njegova svojstva pojavljuju se u *Inspector* prozoru. *Inspector* nudi detaljno pregledavanje i uređivanje svojstava odabranih objekata. (slika 1.5)

*Project* je mjesto za organizaciju datoteka i resursa. Skripte, modeli, teksture, zvukovi, materijali i ostale datoteke obično organiziramo u direktorij *Assets*, dok su u direktoriju *Packages* paketi koji omogućavaju proširenje funkcionalnosti Unity-a.

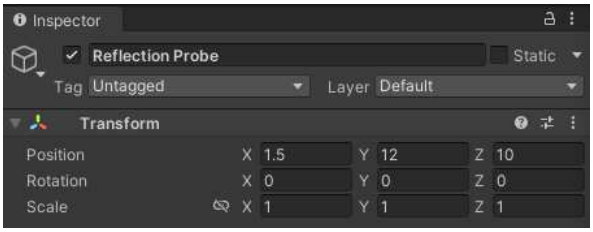



**Slika 1.5.** Inspector

## 1.2. 3D objekti i njihove komponente u Unity-u

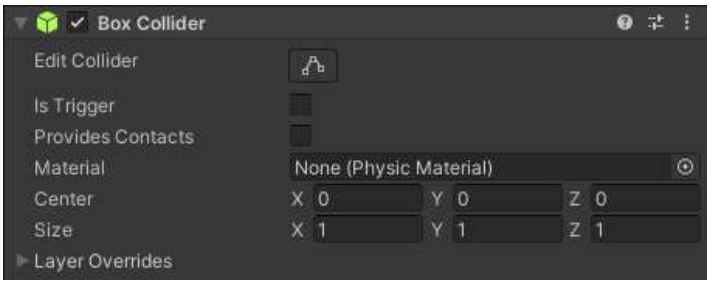
Unity sve 3D objekte kategorizira kao *GameObject* elemente. Budući da želimo osigurati hijerarhiju među elementima scene, *GameObject* zamišljamo kao čvor koji može imati svoje roditelje - nadčvorove i svoju djecu. Sam po sebi, *GameObject* nema funkcionalnosti. Iz tog razloga dodaju mu se različite komponente koje mu osiguravaju različite funkcije. Najvažnije komponente su *Transform*, *Collider*, *Rigidbody*, *Camera*, *Light*, *Audio Source*, *Script*, *Animator* i *CharacterController*.

*Transform* komponentu posjeduje svaki GameObject. Njome određujemo položaj, rotaciju i veličinu 3D objekta po osima x, y i z. (slika 1.6)



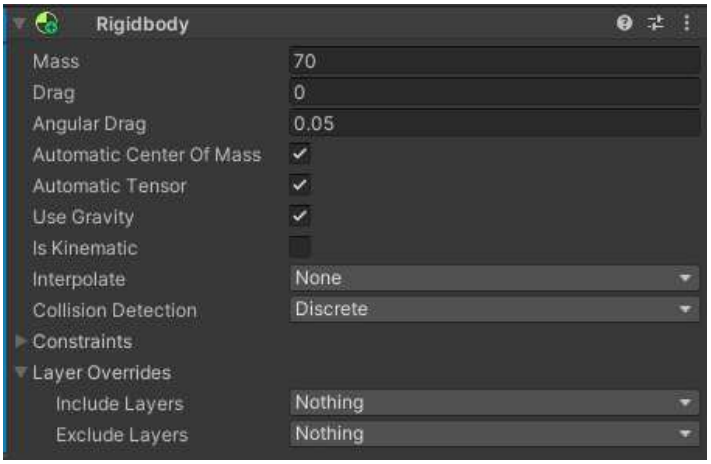

**Slika 1.6.** Transform komponenta

*Collider* komponenta donosi detekciju sudara i interakciju među objektima. Ona definira volumen objekta koji sudjeluje u kolizijama. Unity podržava veći broj Collider-a koji se razlikuju po volumenu koji pokrivaju. Na slici 1.7 je primjer BoxCollider-a, Collider-a koji ima oblik kvadra, pa je zato prikladan za objekte takvog oblika.




**Slika 1.7.** Box Collider komponenta

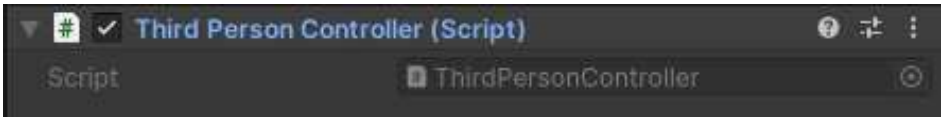
*Rigidbody* komponenta daje fizikalna svojstva objektu, omogućavajući mu da reagira na sile poput gravitacije i sudara. Bez ove komponente ne možemo dobiti realistične fizikalne interakcije u igri. (slika 1.8)




**Slika 1.8.** Rigidbody komponenta

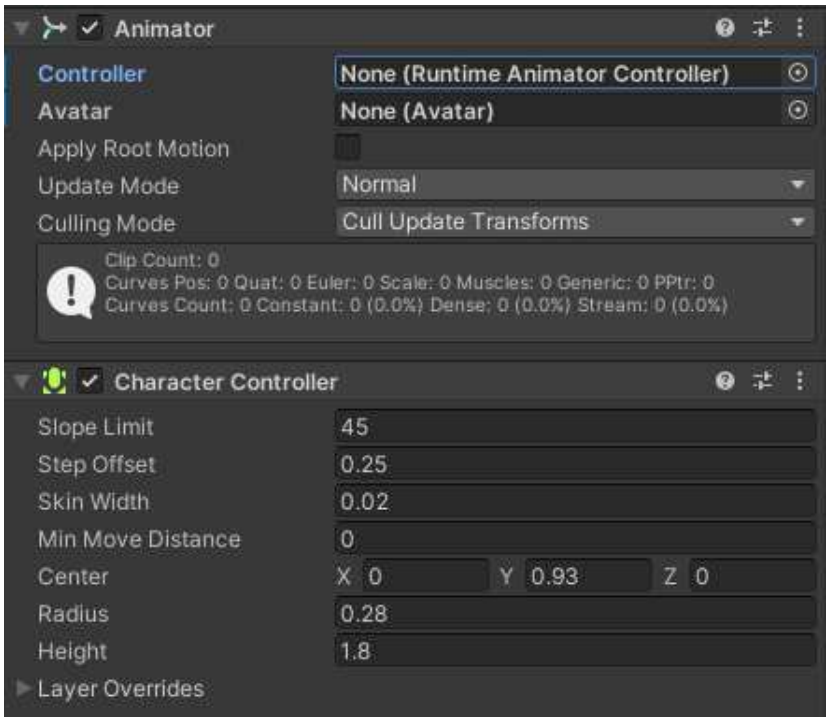
*Camera*, *Light* i *Audio Source* su komponente koje definiraju pogled prikaza scene, svjetlosne izvore i osvjetljavanje objekata te reprodukciju zvukova u sceni.

*Script* komponenta je programski kod – skripta napisana u programskom jeziku C# koja se dodaje objektu za implementaciju ponašanja objekta tijekom igre ili definiranje redoslijeda izvođenja radnji u igri. Svaka skripta sadrži dvije osnovne metode: *Start*() i *Update*(). Metodom *Start* definiramo što će se dogoditi kod prvog poziva skripte, dok u metodu *Update* pišemo radnje koje će se obavljati tijekom svakog frame-a igre, dok je skripta aktivna [3]. (slika 1.9)




**Slika 1.9.** Script komponenta

*Animator* i *CharacterController* su primjeri korisnih opcionalnih komponenti. *Animator* omogućuje upravljanje i reprodukciju animacija, dok s *CharacterController* komponentom možemo na jednostavan način implementirati upravljanje objektima u igri. (slika 1.10)



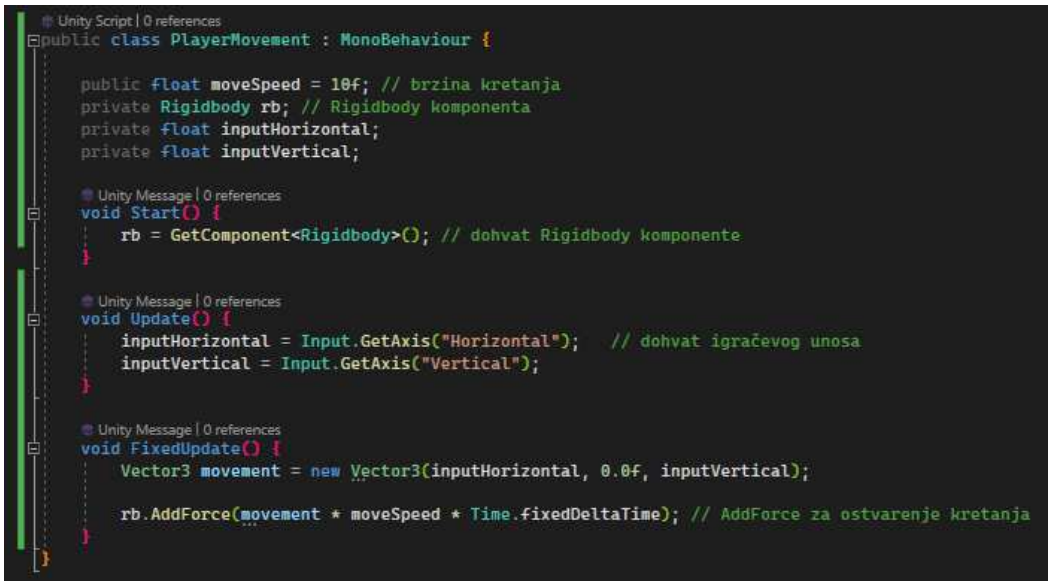

**Slika 1.10.** Animator i CharacterController komponente

## 1.3. Važni koncepti - kretanje objekta, praćenje kamere, sudari

Korištenjem komponenti *GameObject*-a i C# skripti možemo implementirati i prilagoditi bilo koji detalj igre, od upravljanja pokretima objekata i njihove interakcije s okolinom, do složenih vizualnih i zvučnih efekata, animacije i slično. U sljedećim potpoglavljima isječcima koda demonstrirat će se tri glavna koncepta u razvoju bilo koje igre, pa tako i igre skijaške utrke „SKI-FER“.

### 1.3.1. Kretanje objekta

Kretanje objekta je jedan od najvažnijih implementacijskih aspekata modernih igara, pa tako i igara s tematikom utrke kao što je i „SKI-FER“. Želimo li ostvariti kretanje objekta pod utjecajem raznih sila, naš objekt mora imati *Rigidbody* komponentu.




**Slika 1.11.** Skripta Rigidbody kretanja

U kodu sa slike 1.11 vidimo nekoliko ključnih stvari. Prvo, metodom *Start* na početku igre dohvaćamo *Rigidbody* komponentu objekta koji će se kretati. Metodom *Update* u svakom frame-u igre funkcijom *GetAxis* dohvaćamo igračev unos s tipkovnice za kretanje. Metoda *FixedUpdate* koristi se kada je potrebno računanje fizike u realnom vremenu u Unity-u. Razlika između ove metode i metode *Update* je što se *FixedUpdate* poziva prilikom svakog fiksnog okvira, tj. frekvencijom fizikalnog sustava, a ne prilikom osvježavanja svakog okvira. *AddForce* funkcija daje silu komponenti *Rigidbody*, a kao parametre koristi vektor kretanja *Vector3*, brzinu kretanja *moveSpeed* te *Time.fixedDeltaTime* vremenski interval koji se koristi kod ažuriranja fizike. [3]

Naravno, ovaj kod je samo osnova; složene igre imaju mnogo složenije skripte za kretanje objekta. Ovo će biti vidljivo na primjeru igre „SKI-FER“ u poglavlju o implementaciji kretanja skijaša.

### 1.3.2. Praćenje kamerom

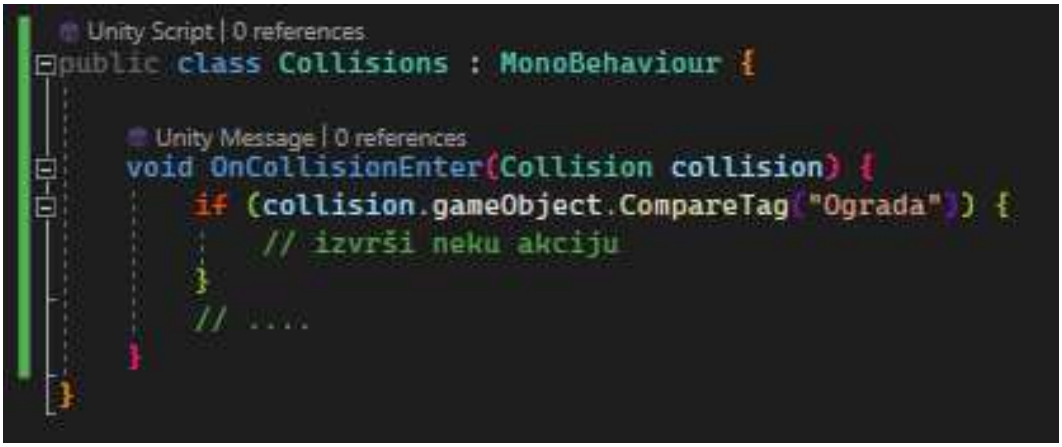
Ostvarivši kretanje objekta, sljedeći korak je upravljanje kamerom. Ideja je da kamera za vrijeme trajanja igre prati neki objekt, primjerice skijaša u igri „SKI-FER“. Ovisno o potrebi, kameru možemo ostvariti tako da bude u prvom licu ili trećem licu te tako da prati objekt cijelo vrijeme na jednakoj udaljenosti ili na različitim udaljenostima.

Praćenje objekta kamerom je također jedna od glavnih komponenta igre „SKI-FER“

i on će uz implementaciju kretanja skijaša detaljno biti objašnjen u zasebnom poglavlju.

### 1.3.3. Sudar

Kako su interakcije objekata prisutne u većini današnjih igara, bitno je razumjeti kako se to ostvaruje u Unity-u. Primjer interakcije objekata u igri „SKI-FER“ je skijašev sudar sa zaštitnom ogradom. Ako želimo ostvariti neko određeno ponašanje kada dva objekta dođu u kontakt, koristimo se komponentom *Collider* i funkcijom *OnCollisionEnter*.




**Slika 1.12.** Funkcija OnCollisonEnter

Na slici 1.12 je kratak isječak funkcije *OnCollisionEnter*. Kada dođe do kolizije, objekt kojemu je pridružena skripta s ovom metodom provjerava oznaku objekta s kojim se sudario. Ako objekt ima oznaku „Ograda“, izvršit’ će se neka akcija. [3]

# 2. VisageSDK

Za stvaranje kvalitetne računalne igre, često nam je pokretački sustav za igre poput Unity-a dovoljan da to i ostvarimo. Međutim, ako želimo da naša igra ima dozu inovativnosti i kreativnosti, možemo posegnuti za nekom naprednijom tehnologijom kako bi unaprijedili neki aspekt naše igre. U računalnu igru „SKI-FER“ integrirana je upravo jedna takva tehnologija – VisageSDK.

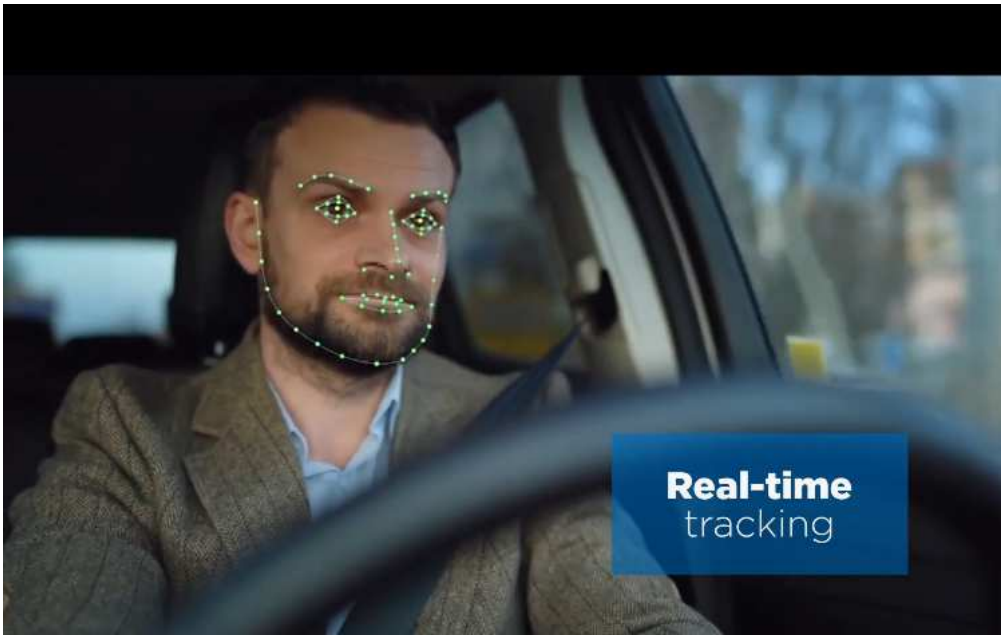
VisageSDK je softverski razvojni paket (engl. Software Development Kit - SDK) specijaliziran za praćenje i analizu lica. Razvijen je od strane tvrtke Visage Technologies i pruži napredne alate za praćenje, analizu i prepoznavanje lica. VisageSDK je dostupan na svim važnijim platformama, a neke od njegovih prednosti su jednostavnost integracije, mogućnost korištenja online i offline te podrška za Unity. [4]

Integracija VisageSDK u Unity projekt igre „SKI-FER“, kao i konkretna realizacija i implementacija praćenja pokreta glave za upravljanje kretanja skijaša bit’ će objašnjena u zasebnom poglavlju. U sljedećim poglavljima ukratko su predstavljene tri glavne značajke paketa VisageSDK – *FaceTrack* (poglavlje 2.1), *FaceAnalysis* (poglavlje 2.2), *FaceRecognition* (poglavlje 2.3).

## 2.1. *FaceTrack*

*FaceTrack* se koristi za praćenje lica, ekspresija lica i očiju u stvarnom vremenu. *FaceTrack* detektira i prati 151 točku na licu jedne ili više osoba na slikama i videima s kamere ili video datoteke, u boji i sivim tonovima. Korištenjem *FaceTrack*-a možemo dobiti 2D i 3D položaj glave i koordinate karakteristika lica, 3D model lica u trenutnom položaju i izrazu te akcijske jedinice koje opisuju trenutne izraze lica poput spuštanja čeljusti, zatvaranja očiju itd.

Uz ovo, *FaceTrack* pruža informacije u realnom vremenu o pokretima očiju i njihovom zatvaranju, smjeru pogleda, 2D i 3D koordinatama zjenica i sl. Ovo pomaže odrediti gdje osoba gleda kako bi se razumjelo što privlači i zadržava njihovu pažnju. [4]



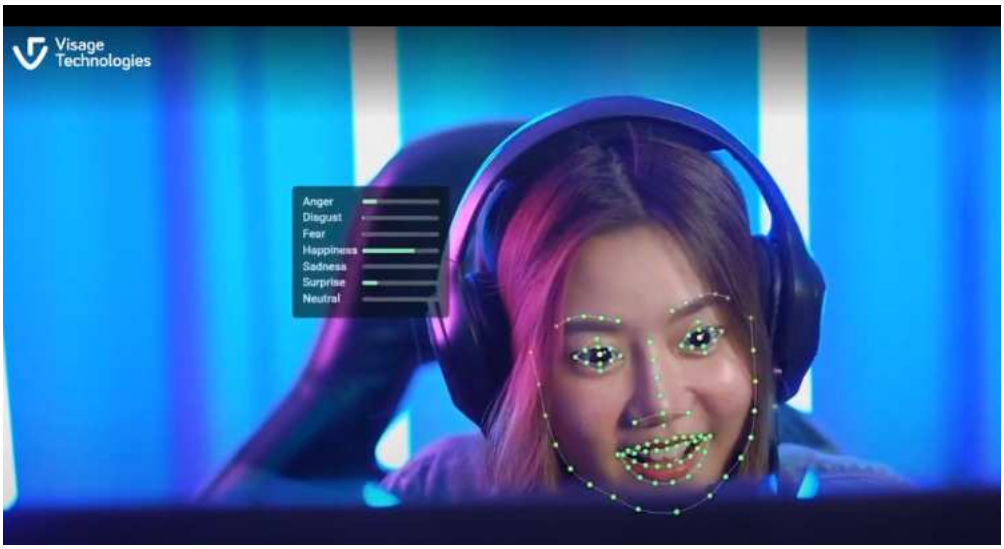

**Slika 2.1.** Primjer korištenja *FaceTrack*

Na slici 2.1 vidimo kako radi *FaceTrack*; lice je podijeljeno određenim brojem točaka koje definiraju čeljust i bradu, usnice, nos, oči te obrve. Ako neka karakteristika lica, npr. usta, promijene položaj, ažurirat će se i pripadne točke u analizi.

## 2.2. *FaceAnalysis*

Ako su nam podaci poput spola, godina ili emocija osobe u centru pozornosti, upotrijebit ćemo *FaceAnalysis* značajku. S pomoću nje možemo iz slika ili videa dobiti distribuciju vjerojatnosti svake od šest univerzalnih emocija – sreća, tuga, ljutnja, strah, iznenađenje i gađenje, te dodatno neutralna emocija. Također, koristeći određene značajke ljudskih lica, *FaceAnalysis* može u realnom vremenu raditi procjenu starosti i procjenu spola. [4]

Na slici 2.2 je demonstracija distribucije vjerojatnosti univerzalnih emocija. Graf pokazuje kako je najveća vjerojatnost da osoba pokazuje emociju "sreća", što gledajući u sliku možemo i potvrditi.




**Slika 2.2.** Primjer korištenja *FaceAnalysis*

## 2.3. *FaceRecognition*

*FaceRecognition* služi za mjerenje sličnosti između lica te prepoznavanje identiteta osobe. *FaceRecognition* se temelji na deskriptorima lica – jedinstvenim nizovima brojeva koji opisuju lica. U realnom vremenu izračunava sličnost između deskriptora lica novodetektiranog lica i svih prethodno registriranih deskriptora. [4]


**Slika 2.3.** Primjer korištenja *FaceRecognition*

Slika 2.3[1] pokazuje primjer identifikacije osoba u realnom vremenu. Četiri osobe istovremeno gledaju u kameru, i svakoj se pridodaje identifikator - ime.

[1]Slike 2.1, 2.2 i 2.3 su isječci iz videozapisa sa službenog YouTube kanala Visage Technologies [5]

# 3. Razvoj igre

Nakon upoznavanja s alatima Unity i VisageSDK, postupak razvoja igre može započeti. Razvoj igre (engl. game development) je složen proces i sadrži nekoliko ključnih faza. Svaka faza bit će razrađena zasebno, počevši s razradom ideje i koncepta igre sve do završnog proizvoda – igre skijanja upravljana praćenjem lica, „SKI-FER“.

Poglavlje 3.1 prikazuje motivaciju za odabir teme igre te postupak određivanja ideje i koncepta igre.

Poglavlje 3.2 bavi se dizajnom i organizacijom scene igre. Ovdje ulazi odabir modela skijaša i ostalog sadržaja vezanog uz skijanje i skijašku stazu, modeliranje staze te stvaranje glavne scene igre.

U poglavlju 3.3 objašnjena je implementacija osnovnog kretanja skijaša te stvaranje dinamičke kamere koja će pratiti skijaša za vrijeme kretanja.

Poglavlje 3.4 pokriva sve važnije „sustave“ koji definiraju logiku igre. Sustavi se sastoje od C# skripti i po potrebi od *GameObject* elementa u hijerarhiji scene. Primjer jednog takvog sustava je sustav za računanje vremena utrke.

Poglavlje 3.5 usredotočuje se na dodavanje preostalih scena u igru. Scene koje će biti dodane su scena glavnog menija i scena *About*.

U poglavlju 3.6 obrađen je posljednji implementacijski detalj igre – integriranje VisageSDK tehnologije u Unity i implementiranje praćenja pokreta lica u skriptu za kretanje skijaša.uspješno dođe do cilja.

Poglavlje 3.7 donosi pregled završnog proizvoda – razvijene igre. U ovom poglavlju prikazan je postupak izgradnje aplikacije (engl. *build*) iz Unity projekta kao priprema za

samo pokretanje i korištenje igre, a potom i upute koje objašnjavaju cjelokupni proces korištenja – igranja igre „SKI-FER“.

## 3.1. Faza 1: Ideja i koncept

Prilikom odabira teme igre, plan je bio odabrati tip igre u kojem praćenje pokreta lica može dobro doći do izražaja. Zbog osobnih afiniteta, počeo sam razmišljati u smjeru sporta, i nakon nekog vremena odabrao temu – skijanje.

Koncept igre izveden je iz skijaške discipline veleslaloma. Veleslalom je skijaška disciplina u kojoj skijaš-natjecatelj nastoji što brže proći zadanu stazu prolazeći kroz označeni put između „vrata“. Razlika između ove discipline i popularne discipline slalom je u tome što je spust strmiji i vrata su postavljena na malo široj međusobnoj udaljenosti. Budući da se radi o utrci, u igri će morati biti implementiran sustav praćenja vremena, a kako je u skijaškim disciplinama za neprolazak kroz vrata kazna diskvalifikacija, sličan sustav također će biti potrebno uključiti u igru. Ovime su definirane temeljene premise/postavke igre. Detaljnija razrada koncepata bit će obavljena u trenutku samog razvoja nekog specifičnog dijela igre.

Dodatnu inspiraciju za detalje igre pronašao sam u igri *Mario & Sonic at the Olympic Winter Games* za konzolu *Nintendo Wii*, s kojom sam često imao doticaja u djetinjstvu. Među dvadesetak zimskih disciplina koje igra nudi, discipline *Downhill* i *Giant Slalom* su skijaške utrke koje prate slične koncepte kojima će se voditi i igra „SKI-FER“.


**Slika 3.1.** Isječak iz igre *Mario & Sonic at the Olympic Winter Games*

Slika 3.1[1] prikazuje isječak iz igre. Ovdje možemo prepoznati neke osnovne elemente

[1]Slika 3.1 je isječak iz YouTube videa kanala NeoTechnoman

scene koje će biti potrebno implementirati – staza, zaštitna ograda, vrata. Uz to, vidimo brojač vremena i brojač prođenih vrata; to će također biti uključeno u igru „SKI-FER“. Na kraju uočavamo i neke elemente koji nisu bili spomenuti – isječak staze i trenutna pozicija igrača, prikaz najboljeg vremena – rekorda, te mogućnost ubrzanja skijaša pritiskom gumba **A** na upravljaču. Ovo su također zanimljivi elementi, no oni neće biti uključeni u konačnu igru razvijenu u sklopu ovog rada. Oni ostaju kao potencijalne nadogradnje igre u budućnosti.

## 3.2. Faza 2: Dizajn i organizacija scene

Razradivši ideju i koncept, sljedeća faza je dizajn i organizacija scene. Dizajn nudi beskonačne mogućnosti u razvoju igre – njime možemo u potpunosti odrediti vizualni prikaz igre.

Jedan od dijelova dizajna igre jest modeliranje. Modeliranje je proces stvaranja prikaza nekog objekta. U našem slučaju, potrebno je modelirati elemente poput skijaša, ograde, vrata, ciljne ravnine i sve ostale elemente koji će činiti stazu. Za potrebe ovog rada, svi potrebni modeli su preuzeti iz *Unity Asset Store*-a i drugih dostupnih izvora. Poglavlje 3.2.1 donosi kratak pregled korištenih modela.

Nakon pribavljanja modela bilo je potrebno stvoriti stazu. Opis i postupak modeliranja staze obrađen je u poglavlju 3.2.2.

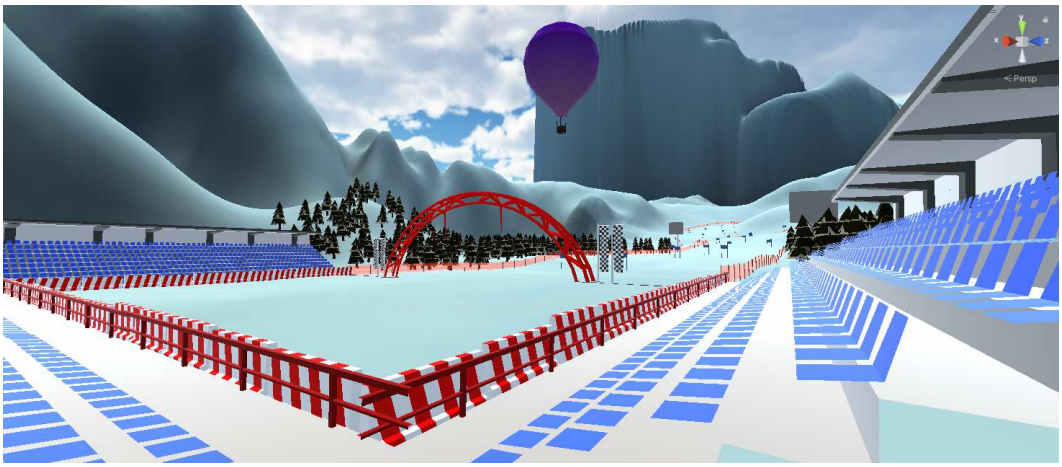
Posljednji korak je organizacija scene (poglavlje 3.2.3). Organizacija scene je uspostavljanje prostornih i drugih odnosa među elementima scene u svrhu vršenja učinkovitijih operacija nad scenom. [6] Ovaj koncept biti će ukratko objašnjen na primjeru skijaša.

### 3.2.1. Korišteni modeli


**Slika 3.2.** Paket *Polygon Snow Kit*

Na slici 3.2 prikazan je paket *Polygon Snow Kit* (autor: syntystudios) [2]. Paket donosi mnoštvo modela specifično napravljenih za skijaške igre i animacije.

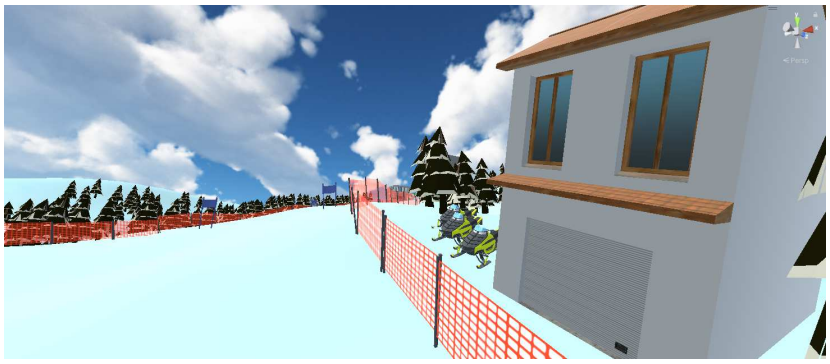
Kako bi okolina staze bila zanimljivija i ispunjenija, u igru su uključeni dodatni modeli. Slike 3.3, 3.4 i 3.5[3] daju uvid u te modele.



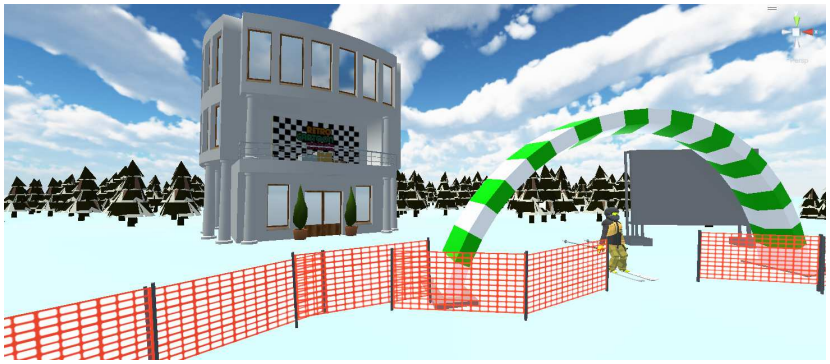
**Slika 3.3.** Tribine i ciljna ravnina

[2]Za razliku od paketa i modela u nastavku poglavlja, ovaj paket jedini nije besplatan.

[3]Modeli korišteni u dizajnu staze: *Billboard* (autor: Mariam Sarishvili), *Low-Poly Simple Nature Pack* (autor: JustCreate), *Environment Track Lowpoly: Cartoon Props Mobile Free* (autor: BE DRILL ENTER), *Cartoon Race Track - Oval* (autor: RCC Design), *Low-poly Medieval Free Pack* (autor: VanillaArt), *Stylized Snow Forest* (autor: Frag112)



**Slika 3.4.** Garaža i motorne sanjke



**Slika 3.5.** Start utrke

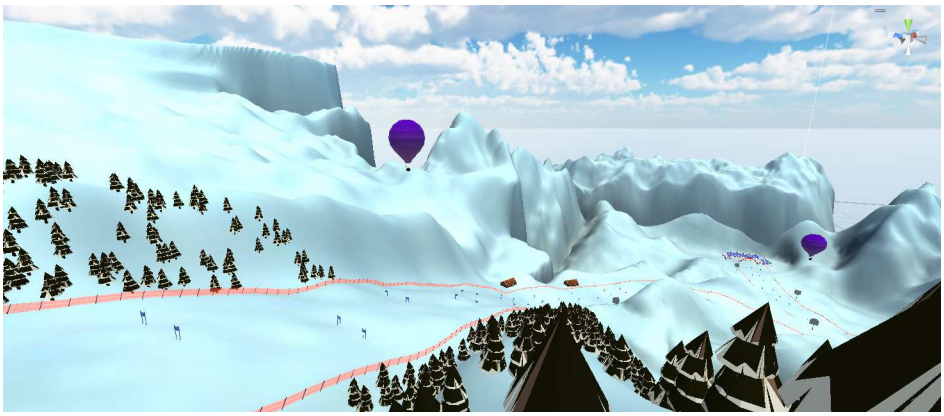
### 3.2.2. Modeliranje staze

Za modeliranje staze korišten je dodatni Unity paket *Terrain*. Paket omogućava manipulaciju i stvaranje površina/terena u Unity-u, pa je zato odličan izbor za ovu zadaću.

*Terrain* ima nekoliko važnijih alata. U ovom radu najviše su korišteni *Paint Terrain* i *Paint Trees*. *Paint Terrain* omogućava jednostavno stvaranje površina u raznim oblicima, dok *Paint Trees* omogućava postavljanje stabala na stvorene površine. Nakon stvaranja površine staze, u komponenti *Terrain* odabran je odgovarajući materijal[4] kako bi površina poprimila izgled skijaške staze, te je u *Terrain Collider* komponenti postavljen *Terrain Data* parametar koji definira teksture i ostale potrebne značajke. Ovime je modeliranje staze bilo dovršeno.

Slika 3.6 prikazuje isječak scene igre koji prikazuje dio staze na kojem se vidi element *Terrain*.

[4]Materijali korišteni u modeliranju staze su iz paketa *Stylize Snow Texture* autora LowlyPoly



**Slika 3.6.** *Terrain*

### 3.2.3. Organizacija scene

Zadnji preostali korak u dizajnu scene bio je posložiti sve elemente na stazu. Ovo je bio prvi pravi izazov u dosadašnjem radu. Organizacija scene idejno ne zvuči problematično, no u većim projektima to može biti vrlo dugotrajan i suhoparan proces. Primjera radi, u završnoj sceni igre „SKI-FER“ nalazi se gotovo tisuću pojedinačnih objekata zaštitne ograde, i svaka od njih je morala biti zasebno postavljena na stazu.

Postupak organizacije scene moguće je vidjeti kod hijerarhije skijaša. Slika 3.7 pokazuje tu hijerarhiju. Korijen objekta je čvor *„Player“*; i on sadrži sve komponente i skripte potrebne u idućim fazama. Ispod njega je čvor *„Character_Skier_02“*, a on predstavlja model skijaša. Svaki dio tijela odvojen je u veće, pa sve manje i preciznije skupine. Objekti poput kacige, skija i štapova moraju biti dodani na odgovarajuće mjesto u hijerarhiji kako bi se mogli transformirati zajedno s odgovarajućim dijelom tijela. Nakon dodavanja ovih elemenata, rotiranjem određenih dijelova skijaša postavljena je konačna poza skijaša, vidljiva na slici 3.8. Skijašu je također pridodana komponenta *BoxCollider* oko skija, koja će do izražaja doći u narednim fazama razvoja.


**Slika 3.7.** *Hijerarhija skijaša*


**Slika 3.8.** *Konačna poza skijaša*

## 3.3. Faza 3: Osnovno kretanje skijaša

Kretanje skijaša tema je 3. faze razvoja igre i označava početak "programerske" strane razvoja. Poglavlje 3.3.1 donosi skriptu *PlayerMovement.cs* koja definira logiku kretanja skijaša, kao i kratak opis praćenja zrake, metode iscrtavanja čiji se koncept koristi u jednoj od metoda skripte.

Ako nakon implementiranja kretanja pokrenemo igru, skijaš će vrlo brzo otići izvan vidljivog područja, stoga je potrebno napisati skriptu za kameru kako bi pratila skijaša za vrijeme utrke - *CameraMovement.cs*. Ona će ukratko biti pojašnjena u poglavlju 3.3.2.

### 3.3.1. Skripta *PlayerMovement.cs*

U utrci veleslaloma skijaši skijaju velikim brzinama po strmom spustu te nemaju razloga za zaustavljanje sve do cilja (jedino ako promaše vrata pa budu diskvalificirani). Imajući ovo u vidu, kretanje skijaša u igri „SKI-FER“ funkcionira na način da skijaš ima stalnu brzinu prema naprijed, a igrač upravlja njegovim skretanjem lijevo i desno, najprije preko tipkovnice, a kasnije pokretima glave.

Skijaš će za vrijeme utrke imati interakcije s raznim objektima, odnosno s njihovim *Collider*-ima. Iz tog razloga skijaš ima *Rigidbody* komponentu, koja osigurava da skijaš ima fizikalna svojstva. Skripta *PlayerMovement.cs* prati slične ideje opisane u poglavlju 1.3.1 o osnovnom *Rigidbody* kretanju. Metodom *Start* dohvaća *Rigidbody* komponentu, dok metoda *FixedUpdate* definira što će se događati u svakom fiksnom okviru. Ona izvršava druge dvije metode – *Move* i *AlignWithTerrain*.

*Move* je metoda koja upravlja kretanjem. Stvara dva *Vector3* objekta – *moveDirection* (smjer kretanja dobiven igračevim unosom i konstantnom brzinom) i *moveVelocity* koji koristi *moveDirection* kao jedan od parametara (brzina kretanja skalirana s vremenom). Pomak skijaša vrši se funkcijom *movePosition* nad *Rigidbody* komponentom, a funkcija *transform.Rotate* nad samim objektom rotira skijaša po osi y.

*AlignWithTerrain* automatski poravnava skijaša s nagibom terena koristeći tehniku *Raycast*. Ideja je da se iz pozicije objekta prema dolje emitira zraka koja će pogoditi površinu – teren. To se postiže funkcijom *Physics.Raycast*. Zatim se uspoređuje kut između normale pogođene površine i vertikalne osi s maksimalnim kutom nagiba definiranim varijablom *maxSlopeAngle*. Ako je kut manji od maksimalnog, objekt se rotira i poravnava s terenom.

Gledajući u retrospektivi, ova skripta mi je zadala najviše problema. Male promjene u kodu dovodile su do neprirodnog, pa čak i komičnog kretanja skijaša. Ipak, upotreba tehnike *Raycast* dovela je do zadovoljavajućih rezultata.

### 3.3.2. Skripta *CameraMovement.cs*

Kamera u skripti *CameraMovement*.cs određena je s nekoliko varijabli; *distance* i *height* opisuju horizontalnu i vertikalnu udaljenost kamere od ciljnog objekta, a *rotati-*

*onDamp* i *heightDamp* definiraju brzinu kojom se kamera prilagođava rotaciji i visini ciljnog objekta. Jedina metoda skripte je *LateUpdate*. Ova metoda je dobra za rad s kamerama jer se poziva u svakom *frame*-u nakon što su svi ostali objekti ažurirani, a to osigurava da se pozicija kamere neće promijeniti prijevremeno. Metoda računa podatke poput trenutne i željene rotacije i visine kamere, računa željenu poziciju kamere uzimajući u obzir zadane parametre te postavlja kameru na izračunatu željenu poziciju funkcijom *transform.position*. Konačno, funkcija *transform.LookAt* usmjerava kameru prema ciljnom objektu.

## 3.4. Faza 4: Glavni sustavi igre

Kretanje skijaša je implementirano, no za pravi doživljaj skijaške utrke to nije dovoljno. U svrhu lakšeg organiziranja, ostatak logike igre će biti podijeljen u nekoliko zasebnih sustava. Ti sustavi su redom *CountdownManager* (poglavlje 3.4.1), *TimerManager* (poglavlje 3.4.2), *GatesCounterManager* (poglavlje 3.4.3) *PlayerCrash* (poglavlje 3.4.4) i *RaceEndManager* (poglavlje 3.4.5). Sustavi se sastoje od C# skripti i pripadnih *GameObject* elemenata u hijerarhiji scene.

U ovu fazu također ulazi i zvučna podrška igre. Kratak osvrt na zvučne efekte i glazbu u igri dan je u poglavlju 3.4.6.

### 3.4.1. *CountdownManager*

Skijaške utrke, ali i utrke bilo kakve vrste, obično započinju odbrojavanjem dok ne dođe do nekakvog signala za početak utrke. *CountdownManager* sustav služi upravo toj svrsi.

*CountdownManager* je *GameObject* čvor s jednom komponentom – skriptom *Countdown.cs*. Za obavljanje željene funkcije korišteno je nekoliko različitih *GameObject* čvorova; *CountDown* koji predstavlja tekst koji će ispisivati na ekranu, te *TimerManager* i *SkierStartControl* objekti koji će se aktivirati kada završi odbrojavanje. *TimerManager* sustav objašnjen je u sljedećem poglavlju, a *SkierStartControl* je pomoćni *GameObject* sa skriptom koja aktivira *PlayerMovement.cs* skriptu, kako skijaš ne bi mogao krenuti prije početka utrke.

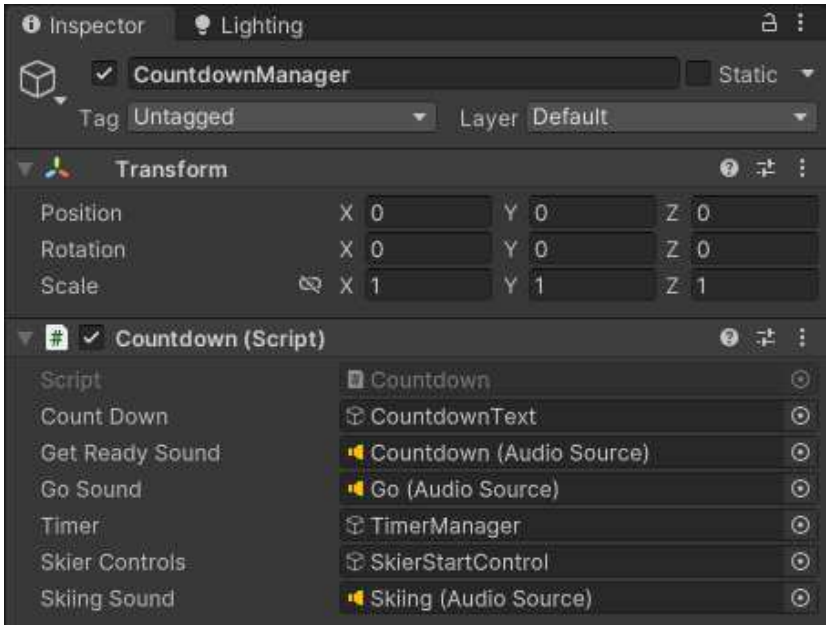
```
void Start() {
    StartCoroutine(CountStart());
}

1 reference
IEnumerator CountStart() {

    Timer.SetActive(false);

    yield return new WaitForSeconds(0.5f);
    CountDown.GetComponent<TextMeshProUGUI>().text = "5";
    GetReadySound.Play();
    CountDown.SetActive(true);

    yield return new WaitForSeconds(1);
    CountDown.SetActive(false);
    CountDown.GetComponent<TextMeshProUGUI>().text = "4";
    GetReadySound.Play();
    CountDown.SetActive(true);
```

**Slika 3.9.** Isječak skripte *Countdown.cs*

Slika 3.9 prikazuje isječak kode iz skripte *Countdown.cs*. Funkcionalnosti koje uključuju vrijeme obično se izvode korištenjem metode *StartCoroutine* koja poziva funkciju korutine. Korutine su funkcije koje mogu zaustaviti svoje izvođenje i nastaviti ga nakon što protekne neko vrijeme. Metodom *StartCoroutine* poziva se korutina *CountStart*. Na početku metode *TimerManager* postaje neaktivan, kako računanje vremena utrke ne bi započelo prije nego odbrojavanje završi. Linijama 27 i 38 zaustavlja se izvođenje korutine na 0.5, odnosno jednu sekundu. Kada prođe to vrijeme, dohvaća se *TextMeshProUGUI* komponenta objekta *CountDown* i postavlja se u brojeve od 5 do 1 nakon svake sekunde. Valja napomenuti kako su tekst objekti korišteni u ovom sustavu, ali i u sustavima *TimerManager* i *GatesCounterManager* smješteni unutar platna *GameUICanvas*. To platno, zajedno s *CrashCanvas* i *FinishCanvas*, o kojima će više riječi biti u sljedećim poglavljima, čini objekt *CanvasComplete* koji je zaslužan za organizaciju svih elemenata korisničkog sučelja. Slika 3.10 prikazuje hijerarhiju *CanvasComplete* čvora.


**Slika 3.10.** Hijerarhija *CanvasComplete* čvora

Na kraju je bilo potrebno povezati odgovarajuće objekte s varijablama deklariranim u skripti u Unity-evom *Inspector* prozoru. To je vidljivo na slici 3.11. U narednim poglavljima se ovaj postupak također provodi, tako da neće više biti posebno naglašen.




**Slika 3.11.** *Inspector* prozor objekta *CountdownManager*

Za primijetiti je da je skriptom definirano i nekoliko *AudioSource* objekata. Ovi i ostali *AudioSource* objekti koji se pojavljuju u skriptama narednih sustava bit će spomenuti u poglavlju o zvučnoj podršci.

### 3.4.2. *TimerManager*

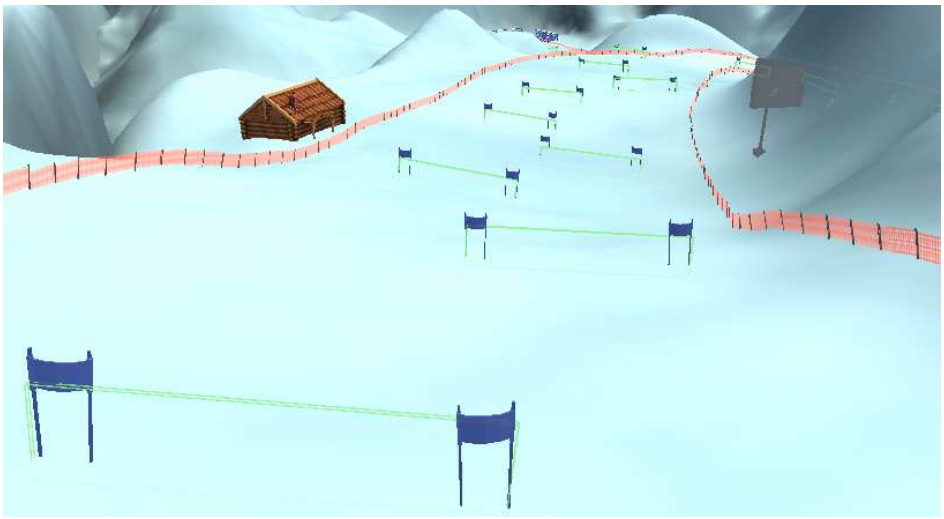
Sustav računanja vremena *TimerManager* definiran je skriptom *Timer.cs*. Funkcijom Start varijable u kojima će biti pohranjene milisekunde, sekunde i minute inicijaliziraju se na 0. Funkcijom *Update* služi kako bi se metoda *TimeUpdate* izvršavala u svakom *frame*-u igre. Metoda *TimeUpdate* služi za ažuriranje tekst objekata iz platna *GameUICanvas* koji igraču prikazuju trenutno vrijeme utrke. Milisekunde koje su protekle dobivaju se pomoću intervala *Time.deltaTime* pomnoženog s 10. Iz milisekundi se zatim dobivaju sekunde, a iz sekundi minute. Slika 3.12 je isječak skripte *Timer.cs* i prikazuje metodu *TimeUpdate*.

```
void TimeUpdate() {
    MilliCount += Time.deltaTime * 10;
    RawTime += Time.deltaTime;

    if (MilliCount >= 10) {
        MilliCount = 0;
        SecondCount++;
    }

    MilliDisplay = Mathf.Floor(MilliCount).ToString("F0");
    MilliBox.GetComponent<TextMeshProUGUI>().text = "" + MilliDisplay;

    if (SecondCount <= 9) {
        SecondBox.GetComponent<TextMeshProUGUI>().text = "0" + SecondCount + ".";
    } else {
        SecondBox.GetComponent<TextMeshProUGUI>().text = "" + SecondCount + ".";
    }

    if (SecondCount >= 60) {
        SecondCount = 0;
        MinuteCount++;
    }

    if (MinuteCount <= 9) {
        MinuteBox.GetComponent<TextMeshProUGUI>().text = "0" + MinuteCount + ":";
    } else {
        MinuteBox.GetComponent<TextMeshProUGUI>().text = "" + MinuteCount + ":";
    }
}
```

**Slika 3.12.** Isječak skripte *Timer.cs*

### 3.4.3. *GatesCounterManager*

U ovom sustavu do izražaja dolazi komponenta *Collider*, odnosno njezin poseban tip – *Trigger*. Ako neki *Collider* označimo kao *Trigger*, on postaje „ne-fizički“ te više ne uzrokuje sudare, nego može pokrenuti događaje ako neki drugi *Collider* uđe u njegov prostor. Ovi nevidljivi *Collider*-i postavljeni su između vrata, što je vidljivo na slici 3.13.



**Slika 3.13.** Nevidljivi *Collider*-i vrata

Kada skijaš uđe u prostor ovih *Trigger*-a, odnosno kada skijašev *Collider* uđe u prostor, izvršava se *OnTriggerEnter* metoda skripte *GatesCounterManager.cs* koja, u slučaju

da sudareni objekt ima oznaku *„Gate“* povećava brojač prođenih vrata.

Skripta *GatesCounterManager.cs* koja implementira ovu logiku dodaje se kao komponenta čvoru *Player*. Razlog za to što upravo skijašev *Collider* ima interakciju s *Gate-Collider*-ima.

### 3.4.4. *PlayerCrash*

Kao i sustav *GatesCounterManager*, sustav *PlayerCrash* također se u obliku skripte (*PlayerCrash.cs*) dodaje čvoru *Player*. U ovom slučaju skijašev *Collider* prilikom sudara s *Collider*-om objekta zaštitne ograde mora obaviti neku akciju. Ta će akcija biti prekid igre. Kada se aktivira metode *OnCollisonEnter*, a sudareni objekt ima oznaku *„Fence“*, zaustavlja se kretanje skijaša, *GameUICanvas* postaje neaktivan i aktivira se *CrashCanvas*. *CrashCanvas*[5] sadrži odgovarajuću poruku koja igraču daje do znanja da je igra gotova te nudi dvije opcije u obliku dva gumba: *TryAgain* za ponovno pokretanje utrke i *MainMenu* za povratak u glavni meni. Slika 3.14 prikazuje situaciju sudara skijaša sa zaštitnom ogradom.


**Slika 3.14.** Sudar skijaša sa zaštitnom ogradom

[5]CrashCanvas, EndCanvas te scene MainMenu i AboutMenu uređene su alatom Canva - besplatnim alatom za uređivanje i stvaranje prezentacija, plakata i slične vizualne elemente

### 3.4.5. *RaceEndManager*

Kada skijaš uspješno prođe kroz cilj, aktivira *EndGateTrigger* kojim se pokreće sustav *RaceEndManager*, definiran skriptom *RaceEndManager.cs*. On zaustavlja daljnje kretanje skijaša, gasi *GameUICanvas* i aktivira *EndCanvas* na kojemu je prikazano igračevo vrijeme te broj prođenih vrata. *EndCanvas* ima dva dodatna gumba – *RaceAgain* za pokretanje nove utrke, te *MainMenu* za odlazak u glavni meni.


**Slika 3.15.** *EndCanvas* kod završetka utrke

Slika 3.15 vizualizira ovaj scenarij. Na slici je moguće primijetiti crveni tekst *Penalty*. Ovo je dodatna karakteristika uključena u skriptu *RaceEndManager.cs*; za svaka propuštena vrata, kazna od pet sekundi dodana je na ukupno vrijeme. Kako je promašeno 16 vrata, ukupan *penalty* je 80 sekundi.

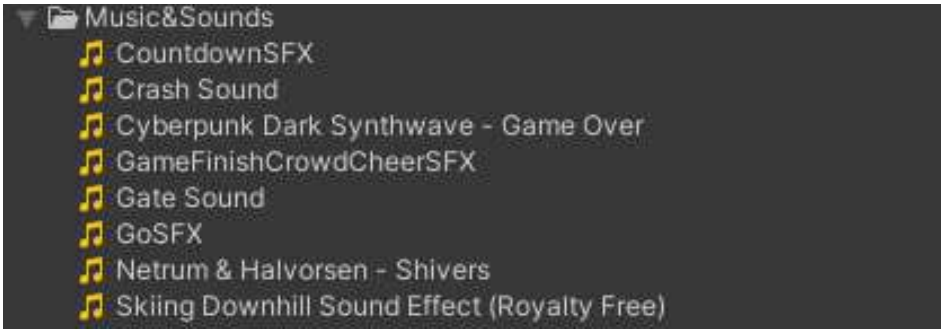
Još jedna značajka implementirana skriptom *RaceEndManager.cs* je prikaz teksta uspješnosti. Ako se utrka završi s prođenih svih 48 vrata, ispisat će se tekst *„Perfect!“*. Ako se prođe između 40 i 48 vrata, ispisat će se *„Great“*. Za sve ispod 40 prođenih vrata ispisuje se tekst *„Good“*, kao što je to i na slici 3.15.

### 3.4.6. Zvučna podrška

Zvučna podrška uključuje pozadinsku glazbu i zvučne efekte.

Pozadinska glazba uključuje glazbu u glavnom meniju i glazbu *GameAudio* koja se izvodi za vrijeme same utrke. U slučaju sudara u zaštitnu ogradu, *GameAudio* se prekida.

Zvučni efekti uključuju: zvuk odbrojavanja, zvuk početka utrke, zvuk skijanja, zvuk prolaska kroz vrata, zvuk sudara u ogradu te zvuk završetka utrke [6]. Organizirani zvučni efekti i glazba prikazani su na slici 3.16.




**Slika 3.16.** Organizacija glazbe i zvučnih efekata

## 3.5. Faza 5: Ostale scene

Ako želimo da prilikom pokretanja aplikacije i ulaska u igru ne započnemo odmah s glavnom scenom, potrebno je napraviti neku sporednu scenu kao poveznicu na glavnu scenu. Ova faza razvoja bavi se stvaranjem scene glavnog menija i dodatne scene *About-Menu* koja će služiti za demonstraciju prebacivanja iz jedne scene u drugu.

*MainMenu* scena sadrži tri gumba: *Play*, *About* i *Quit*. Prelaskom miša preko svakog gumba pojavljuje se pripadni okvir kako bi korisnik bolje vidio na kojoj je opciji. Akcije gumba definirane su u skripti *MainMenu.cs*.


**Slika 3.17.** *MainMenu* scena

[6] Glazba i zvukovi korišteni: glazba glavnog menija - *Game Over* (autor: White Bat Audio), glazba igre - *Shivers* (autori: Netrum, Halvorsen (NCS)), zvuk skijanja - *Skiing Downhill Sound Effect* (autor: Free stock footages and sound effects), ostali zvukovi - paket *CasualGameSounds* (autor: Dustyroom)

Na slici 3.17 je vidljiv izgled scene *MainMenu*. *Play* pokreće scenu Base – glavnu scenu igre, *About* pokreće scenu *AboutMenu*, a gumbom *Quit* se izlazi iz aplikacije.

Scena *AboutMenu* sadrži kratak tekst koji opisuje igru „SKI-FER“. Iz ove scene moguće je vratiti se u scenu *MainMenu* preko gumba Back.

## 3.6. Faza 6: Praćenje pokreta lica

Igra je gotovo završena. Posljednja faza razvoja je integracija napredne tehnologije praćenja pokreta lica u igru "SKI-FER".

Za integraciju VisageSDK tehnologije, u Unity projekt uključen je direktorij *StreamingAssets*, koji omogućuje pohranu datoteka potrebnih aplikaciji u izvornom formatu i dodan je direktorij *visage* za organizaciju i pohranu resursa i konfiguracija koje koristi VisageSDK.

Skripta koja upravlja inicijalizacijom VisageSDK za praćenje lica i korištenje podataka o praćenju za kontrolu kretanja skijaša je *VisageInit.cs* i ona se kao komponenta dodaje objektu *Player*. Skripta služi za dohvat i primjenu podataka o praćenju lica.

Metoda *Awake* poziva se prije nego što se pokrene bilo koja druga metoda u skripti. Ona stoga služi za inicijalizaciju licence i praćenja lica funkcijama *initializeLicence* i *initTracker*, te za pokretanje kamere funkcijom *openCamera*.

Metoda *Update* dohvaća trenutni okvir s kamere funkcijom *grabFrame*, izvodi praćenje funkcijom *track* i dohvaća status praćenja s *getTrackerStatus*. Provjerava se kontrolna *bool* varijabla *isTracking* koja je postavljena na *true*, pa se izvršava metoda *updateSkier*.

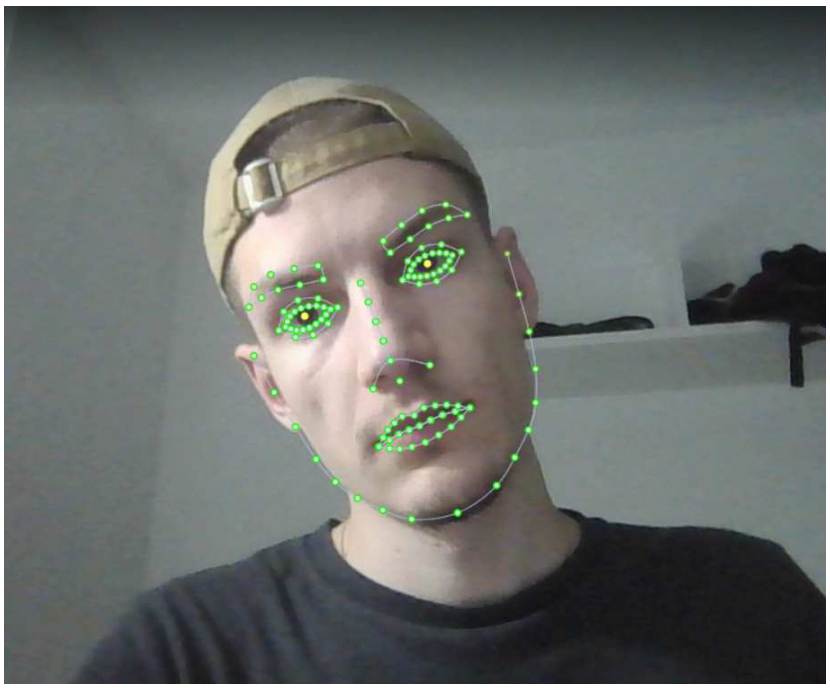
*UpdateSkier* ažurira podatke o praćenju lica i koristi ih za upravljanje kretanjem skijaša. Metoda prima parametar *faceIndex* postavljen na 0, jer se prati samo jedno, prvo detektirano lice. Funkcija *getHeadTranslation* popunjava varijablu *translation* vrijednostima translacije glave. Te se vrijednosti kopiraju u polje *Translation*, koje se zatim prosljeđuje u metodu *MoveSkier*.

*MoveSkier* iz primljene *Vector3* varijable translacije izvlači translaciju po x osi, koja predstavlja pomak glave lijevo ili desno. Budući da je ideja da igrač upravlja skijaševim

kretanjem upravo pomakom glave lijevo-desno, ta se translacija prosljeđuje metodi iz skripte *PlayerMovement.cs*, metodi *setInput*.

Metoda *setInput* radi ono što inače radi funkcija *Input.GetAxis*; postavlja igračev unos za horizontalni smjer u translaciju dobivenu iz skripte *VisageInit* i množi ga varijablom *horizontalScaler* kako bi se postigla optimalna osjetljivost praćenja pokreta glave.

Na ovaj način se postigla željena funkcionalnost: upravljanje skijašem pokretima glave.



**Slika 3.18.** *ShowCaseDemo*

Slika 3.18 prikazuje praćenje lica s pomoću ključnih točaka alatom *ShowCaseDemo* tvrtke Visage Technologies. Pomakom glave u lijevo, kao na slici, dobivena je negativna vrijednost translacije po x osi. Da je pomak bio u desno, translacija po x osi bila bi pozitivna. Na taj način određuje se smjer kretanja skijaša u igri „SKI-FER“.

## 3.7. Završni proizvod

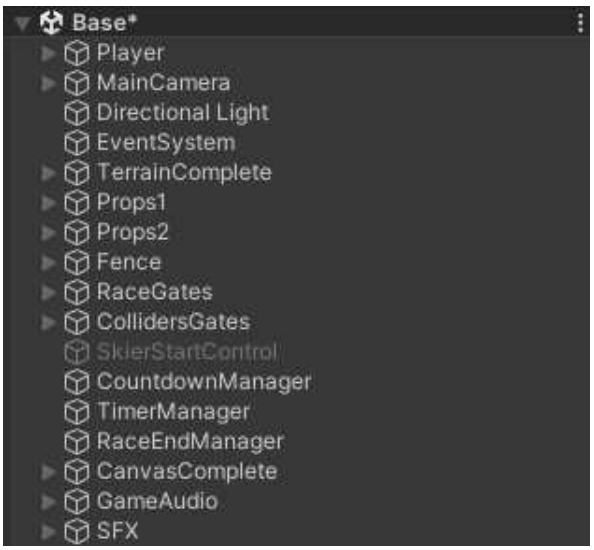
Na kraju je ostalo pregledati završni proizvod. Poglavlje 3.7.1 donosi završni izgled hijerarhije glavne scene i komponente objekta *Player*.

Po završetku implementacije i razvoja igre potrebno je izgraditi aplikaciju. Postupak izgradnje predstavljen je u poglavlju 3.7.2.

Poglavlje 3.7.3 su upute za korištenje aplikacije/igre „SKI-FER“. Osobi koja je pažljivo prošla kroz cijeli rad zasigurno nisu potrebne upute za korištenje jer je svaki dio igre detaljno objašnjen. Ovo poglavlje zato služi kao primjer kako bi trebala izgledati neka osnovna podrška aplikacije/igre za korisnike koji nisu upoznati s igrom, njezinim funkcionalnostima ili procesom igre.

### 3.7.1. Završni izgled hijerarhije i komponente objekta *Player*

Kako bi hijerarhija bila pregledna i razumljiva, dobra praksa je organizirati istovjetne objekte u zajednički nadčvor.




**Slika 3.19.** Hijerarhija igre SKI-FER

Na slici 3.19 moguće je vidjeti završnu hijerarhiju igre „SKI-FER“. Objekti uključuju igrača *Player*, sustave logike igre poput *TimerManager* i *RaceEndManager*, objekte zvučne podrške *GameAudio* i *SFX*, objekte površine staze organizirane u čvor *TerrainComplete*, te objekte ograde, vrata i ostalog sadržaja objedinjene zajedničkim čvorovima poput *RaceGates*, *Fence*, itd.

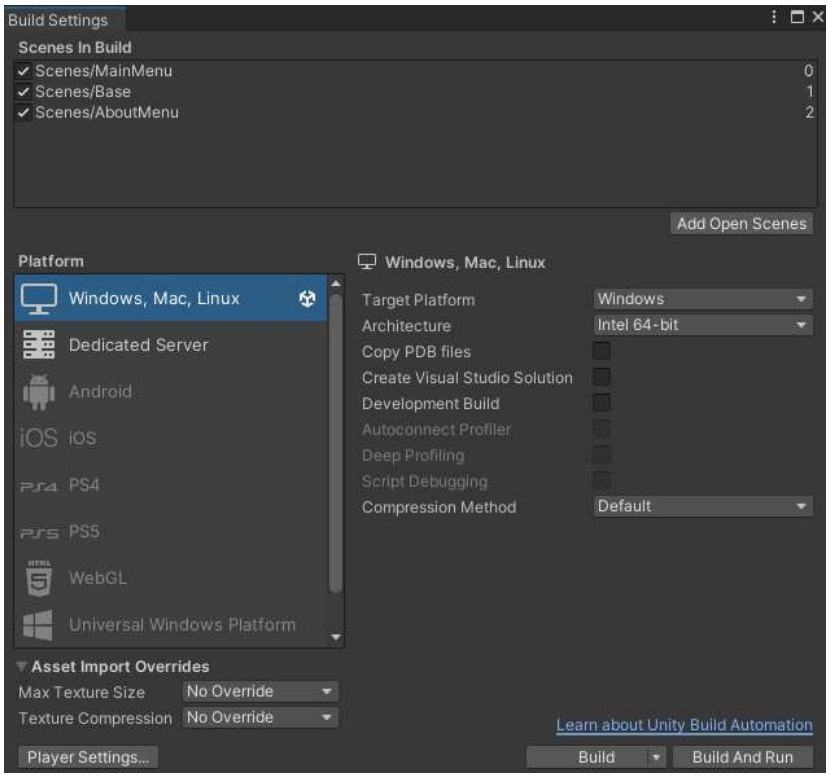
Komponente *Player* objekta vidljive su u *Inspector* prozoru. To je prikazano na slici 3.20. Komponente uključuju *BoxCollider* i *Rigidbody* kao osnovne Unity komponente, te komponente skripti *Player Movement*, *Gates Counter Manager*, *Player Crash* i *Visage Init*.


**Slika 3.20.** Komponente objekta *Player*

### 3.7.2. Izgradnja aplikacije

Izgradnja Unity igre vrši se opcijom *Build Settings*. Odabirom opcije otvara se prozor prikazan slikom 3.21. Odabirom odgovarajuće platforme i provjerom da su uključene sve relevantne scene, možemo odabrati opciju *Build* za izgradnju aplikacije.

Igra će biti izgrađena u odabranom direktoriju, te ju je moguće isprobati pokretanjem *.exe* datoteke unutar odabranog direktorija.




**Slika 3.21.** *Build Settings*

### 3.7.3. Upute za korištenje

Glavni preduvjet ispravnog korištenja/igranja igre je uključena i funkcionalna kamera.

Nakon preuzimanja i pokretanja igre korisnik se nalazi u glavnom meniju iz kojeg može pokrenuti utrku pritiskom na *Play*, otvoriti dodatne informacije pritiskom na *About* ili izaći iz igre pritiskom na *Quit*. Izgled glavnog menija je na slici 3.22.


**Slika 3.22.** *Main Menu*

Pritiskom na *About* otvara se nova scena na kojoj je moguće pročitati nekoliko informacija o igri. Izlaz iz scene i povratak u glavni meni radi se pritiskom na *Back*. Slika 3.23. prikazuje scenu *About*.


**Slika 3.23.** *About*

Pritiskom na *Play* započinje utrka. Nakon početnog odbrojavanja, korisnik pokretima glave upravlja skijašem i nastoji završiti utrku sa što bržim vremenom. Dvije situacije mogu uslijediti: igra može biti prekinuta u slučaju da se skijaš sudari u zaštitnu ogradu, ili može biti uspješno završena prolaskom skijaša kroz ciljnu ravninu.

U slučaju prekida zbog sudara, korisniku se ispisuje odgovarajuća poruka i dobiva mogućnost ponovno pokrenuti igru pritiskom na Try *Again*, ili vratiti se u glavni meni pritiskom na *Main Menu*. Slika 3.23 pokazuje ovu situaciju.


**Slika 3.24.** Sudar

U slučaju uspješno završene utrke, korisnik može pregledati svoje vrijeme i broj vrata koje je prošao, kao i poruku o uspješnosti koja može biti *Good*, *Great* ili *Perfect*! . Korisnik zatim može resetirati igru opcijom *Race Again* ili otići u glavni meni opcijom *Main Menu*. Tipičan prikaz završene utrke vidljiv je na slici 3.24.


**Slika 3.25.** Završetak utrke

# Zaključak

Razvoj igre „SKI-FER“ upravljane praćenjem lica pokazao se kao zahtjevan, ali vrlo zanimljiv i poučan zadatak. Proces razvoja obuhvatio je mnoge faze, od početne ideje i koncepta, preko modeliranja staze i implementacije kontrola, do integracije tehnologije praćenja lica. Svaka faza bila je po nečemu različita od prošle i problematična na svoj način. Kroz ove faze dobio sam dobar dojam o tehničkim aspektima razvoja igre, kao i o izazovima koji prate razvijatelje igri na svakom koraku, pogotovo u slučaju implementacije inovativnih značajki poput praćenja lica.

„SKI-FER“ ima mnogo prostora za budući razvoj. Osim proširenja postojećih funkcionalnosti, neke zanimljive ideje mogle bi biti dodavanje novih staza, implementiranje zabavnih značajki poput pojačanja (engl. *Power-Up*) ili prepreka na stazi te stvaranje sustava za praćenje najboljih postignutih rezultata.

Ovaj rad i razvijena igra „SKI-FER“ rezultat je dječačke želje za shvaćanjem pozadine digitalnog svijeta video igre. Zbog toga cilj ovog rada nije bio samo šturo navođenje činjenica i svih implementiranih detalja igre; ideja rada je da na zanimljiv način prikaže cijeli postupak razvoja jednostavne igre kako bi razvijatelji, bilo početnici ili iskusniji programeri, ali i osobe koje nisu nikada programirale, dobile inspiraciju i motivaciju za stvaranjem vlastite igre ili bilo kojeg vlastitog proizvoda.

# Literatura


[1] Wikipedia, "Game engine," *Wikipedia*, https://en.wikipedia.org/wiki/Game_engine, pristupljeno 12. lipnja 2024.

[2] Wikipedia, "Unity (game engine)," *Wikipedia*, https://en.wikipedia.org/wiki/Unity_(game_engine), pristupljeno 12. lipnja 2024.

[3] I. Pandžić, "Graf Scene Unity Upute," *FER*, https://www.fer.unizg.hr/_download/repository/V02GrafSceneUnityUpute[1].pdf, pristupljeno 10. lipnja 2024.

[4] Visage Technologies, "Visage Technologies," *Visage Technologies*, https://visagetechnologies.com/, pristupljeno 9. lipnja 2024.

[5] Visage Technologies, "Visage Technologies YouTube channel," *YouTube*, https://www.youtube.com/@VisageTechnologies, pristupljeno 9. lipnja 2024.

[6] Pandžić, I., Pejša, T., Matković, K., Benko, H., Čereković, A., Matijašević, M. *Virtualna okruženja: Interaktivna 3D grafika i njene primjene*. 1. izdanje. Zagreb: Element, 2011.

[7] FER, "Nastavni materijali," *FER*, https://www.fer.unizg.hr/predmet/ovo/materijali, pristupljeno 25. svibnja 2024.

[8] Visage Technologies, "Visage Technologies Documentation," *Visage Technologies*, https://docs.visagetechnologies.com/, pristupljeno 9. lipnja 2024.

[9] Unity Technologies, "Unity Documentation," *Unity*, https://docs.unity.com/, pristupljeno 25. svibnja 2024.

[10] Unity Technologies, "Unity Forum," *Unity*, https://forum.unity.com/, pristupljeno 25. svibnja 2024.

[11] Unity Technologies, "Unity Asset Store," *Unity*, https://assetstore.unity.com/, pristupljeno 25. svibnja 2024.

[12] SoloGameDev, "Unity TERRAIN Tutorial," *YouTube*, https://www.youtube.com/watch?v=DbJB9534PZQ&ab_channel=SoloGameDev, pristupljeno 25. svibnja 2024.

[13] GameDevBeginner, "Raycasts in Unity," *YouTube*, https://www.youtube.com/watch?v=B34iq4O5ZYI&ab_channel=GameDevBeginner, pristupljeno 9. lipnja 2024.

# Sažetak

„SKI-FER“ je računalna igra skijaške utrke napravljena popularnim alatom za razvoj videoigara Unity. Inovativna značajka igre jest integracija razvojnog alata VisageSDK za praćenje i analizu lica i glave za upravljanje skijašem na stazi. Cilj igrača je pokretima glave lijevo i desno usmjeravati skijaša po skijaškoj stazi, proći kroz što više vrata i izbjeći sudaranje u zaštitnu ogradu kako bi završio utrku u što manjem vremenu. Model skijaša i popratni sadržaji preuzeti su iz Unity Asset Store-a i drugih izvora te ukomponirani u cjelokupni dizajn staze pomoću dostupnih Unity alata. Logika i sve kontrole dio su skripti pisanih u programskom jeziku C#.

**Ključne riječi:** SKI-FER; igra skijanja; praćenje lica; Unity; Visage

# Abstract

„SKI-FER“ is a skiing computer game developed using the popular Unity game engine. An innovative feature of the game is the integration of the VisageSDK development tool for face and head tracking for controlling the skier on the course. The player’s goal is to steer the skier left and right by moving their head, pass through as many gates as possible and avoid crashing into the safety fences to finish the race in the fastest time possible. The skier model and accompanying assets were attained from the Unity Asset Store and other sources, then integrated into the overall course design using available Unity tools. The game logic and all controls are implemented through scripts written in the C# programming language.

**Keywords:** SKI-FER; skiing game; face tracking; Unity; Visage