Razvoj simulacijske digitalne igre AIR u Unity alatu

Vidiček, Marko

Undergraduate thesis / Završni rad

2020

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

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:195:059432

Sveučilište u Rijeci
Odjel za informatiku

Preddiplomski jednopredmetni studij Informatike

Marko Vidiček

Razvoj simulacijske digitalne igre AIR u Unity alatu

Završni rad

Mentor: izv. prof. dr. sc. Marina Ivašić Kos

RIJEKA, rujan 2020.

Rujan, 2020.

ODJEL ZA INFORMATIKU
SVEUČILIŠTE U RIJECI

Rijeka, 17. veljače 2020.

Zadatak za završni rad

Pristupnik: Marko Vidiček

Naziv završnog rada: Razvoj simulacijske digitalne igre AIR u Unity alatu

Naziv završnog rada na eng. jeziku: Development of simulation digital game AIR in Unity tool

Sadržaj zadatka: Proučiti Unity Engine alat za razvoj video igara i iskoristiti isti za razvoj igre sa naglaskom na simulaciju aerodinamike leta.

Osmisliti i razviti računalnu igru u kojoj igrač upravlja zrakoplovom u 3rd ili 1st Person perspektivi korištenjem tipkovnice i miša ili Xbox kontrolera s ciljem prolaska staze u što kraćem vremenu kako bi skupio što više bodova.

Objasniti ključne dijelove kodova i skripte koje daju funkciju elementima koji su ubačeni u Unity Engine poput kontrole leta, staza i te opisati korištenje dodatnih alata koji se koriste za dovršavanje funkcionalnosti igre.

Osim alata Unity Engine koristi se i Adobe Photoshop za izradu vlastitih grafika. Za izradu skripti koje pokreću sve elemente koristi se C# jezik.

Mentor

Izv. prof. dr. sc. Marina Ivašić-Kos

Voditelj za završne radove

Doc. dr. sc. Miran Pobar

Zadatak preuzet: 17.2.2020

(potpis pristupnika)

Marko Vidiček

Rujan, 2020.

1. Sažetak

Ovaj završni rad opisuje procese izrade video igre AIR: Airplane Immersive Racing u alatu Unity Engine. Cilj AIR računalne igre je upravljati zrakoplovom iz trećeg ili prvog lica te završiti stazu u što moguće kraćem vremenu i ponovno sletjeti na pistu ili samo uživati u letu.

U uvodu će biti opisana kratka povijest alata u kojem je igra rađena (Unity Engine) te povijest računalne igre koja je inspirirala rad ove igre, a to je Microsoft Flight Simulator.

Prilikom razvoja igre osim originalnih kodova i asseta korišteni su i neki asseti sa Interneta i Unity Store-a. U radu će biti opisani najbitniji dijelovi koda te korištenje asseta.

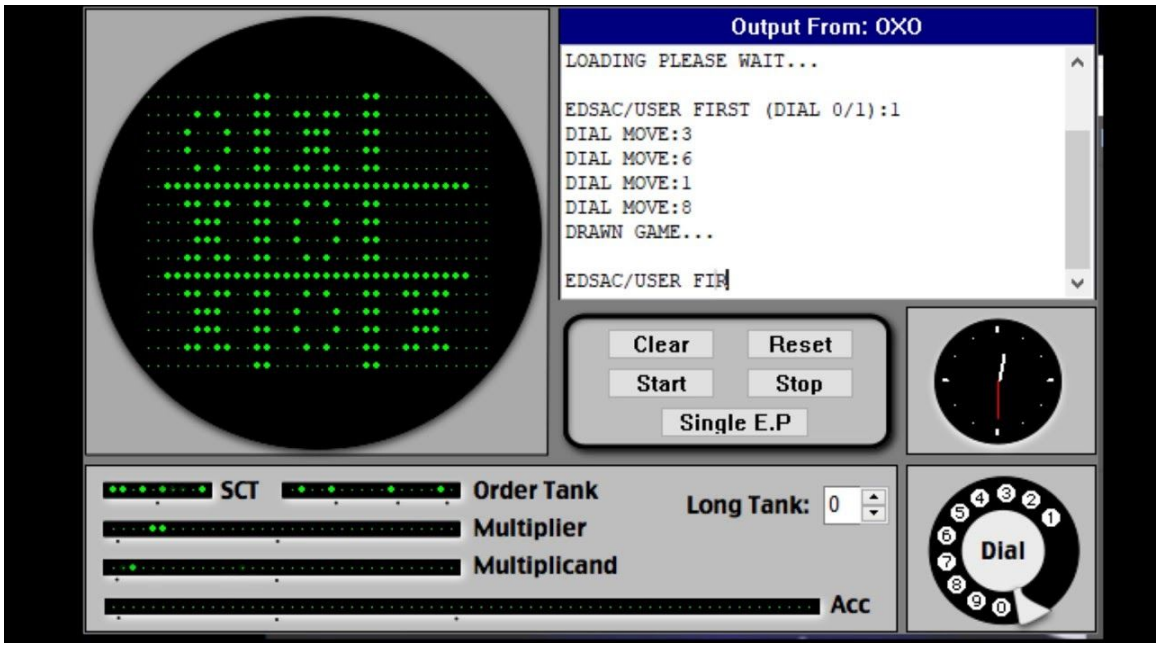
Ključne riječi: AIR, Airplane, Racing, Unity Engine, Microsoft Flight Simulator, Allsky, BOXOPHOBIC, Amplify Shader Editor, TextMeshPro

Rujan, 2020.

2. Uvod

Povijest razvoja igara **[1]** započinje već 1950-ih, iako koja je prva igra ovisi o definiciji video igre. Video igra je elektronička igra koja uključuje interakciju s korisničkim sučeljem ili uređajem za unos informacija, kao što su džojstik, kontroler ili tipkovnica radi generiranja vizualnih povratnih informacija na uređaju za prikaz. Glavne značajke video igara

Slika 1. prikazuje prvu računalnu igru koja je koristila digitalni zaslon, napisao ju je Alexander S. Douglas 1952. godine, i zvala se OXO **[2]**. 1958., igru pod nazivom Tenis za dvoje, koja je svoje rezultate prikazivala na osciloskopu, izradio je Willy Higinbotham, fizičar iz Nacionalnog laboratorija Brookhaven. 1961., glavna računalna igra nazvana Spacewar! razvila je skupina studenata Massachusetts Institute of Technology pod vodstvom Stevea Russella.

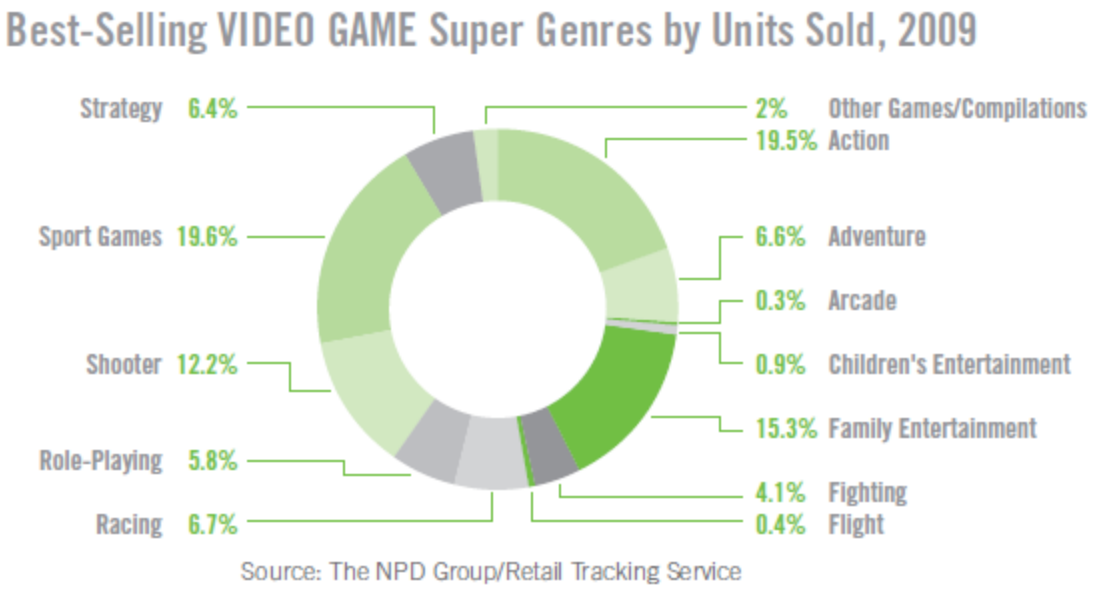


Slika 1. OXO, Alexander S. Douglas, 1952.

Istinski komercijalni dizajn i razvoj igara započeo je sedamdesetih godina prošlog stoljeća, kada su se prodavale arkadne video igre i konzole prve generacije. 1971. izdana je prva komercijalna video igra s novčićima, Computer Space. Za prikaz je koristio crno-bijeli televizor, a računalni sustav izrađen je od TTL čipova serije 74. 1972. godine objavljen je prvi sustav kućnih konzola nazvan Magnavox Odyssey, koji je razvio Ralph H. Baer. Iste godine Atari je objavio Pong, arkadnu igru koja je povećala popularnost video igara. Komercijalni uspjeh Ponga vodio je druge tvrtke da razvijaju Pong klonove, stvarajući industriju videoigara.

Žanr video igre **[3]** posebna je kategorija igara povezanih sa sličnim karakteristikama igranja. Žanrovi mogu obuhvaćati širok spektar igara, što dovodi do još specifičnijih klasifikacija nazvanih podžanrovi. Na primjer, akcijska igra može se klasificirati u mnoge podžanrove kao što su igre na platformi i borbene igre. Neke se igre, osobito pregledničke i mobilne igre, obično klasificiraju u više žanrova.

Od vremena prvih video igara pa do danas razvilo se mnogo različitih žanrova i podžanrova video igara **[4]**. Neki od najpopularnijih su: Akcijske igre (shooteri, platform, stealth, survival), akcijske avanture, igre igranja uloga (RPG), simulacije, strateške igre, sportske…itd.
Na slici 2 su prikazani najprodavaniji žanrovi i njihov postotak od ukupne prodaje video igara..



Slika 2. Najprodavaniji žanrovi video igara

Simulacijska video igra **[5]** opisuje raznoliku superkategoriju video igara, uglavnom dizajniranih za simulaciju aktivnosti u stvarnom svijetu.
Simulacijska igra pokušava kopirati razne aktivnosti iz stvarnog života u obliku igre u razne svrhe, poput treninga, analize ili predviđanja. Obično u igri ne postoje strogo definirani ciljevi, a igraču je umjesto toga dozvoljeno da slobodno kontrolira lik ili okolinu.

Simulacijske igre vozila **[6]**, žanr su videoigara koje igraču pokušavaju pružiti realnu interpretaciju upravljanja raznim vrstama vozila.

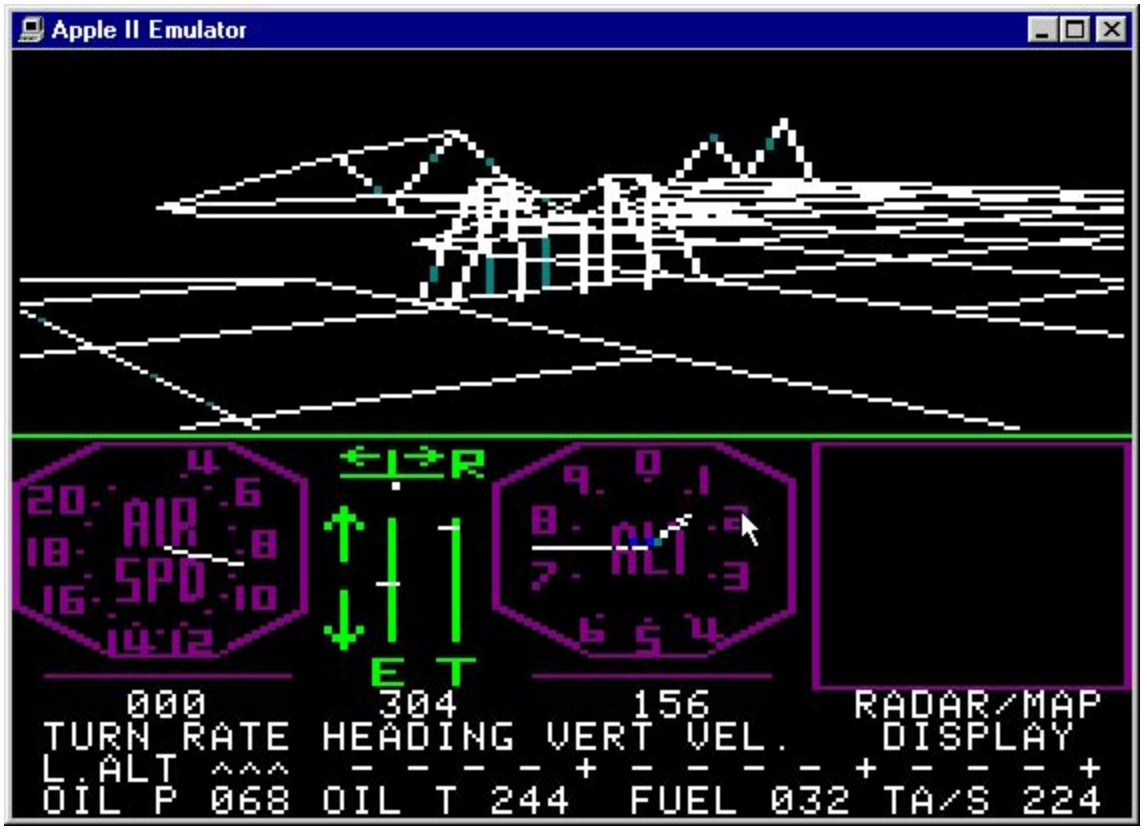
Simulatori borbenih letova najpopularniji su podžanr simulacije. Igrač kontrolira avion, ne samo da simulira čin letenja, već i borbene situacije. Postoje i civilni simulatori leta koji nemaju borbeni aspekt npr. Microsoft Flight Simulator.

2.1 Inspiracija

Microsoft Flight Simulator **[7]** (često skraćenica MSFS ili FS) niz je amaterskih programa simulator leta za operativni sustav Microsoft Windows i ranije za MS-DOS i Classic Mac OS.

To je jedan od najdugovječnijih, najpoznatijih i najopsežnijih simulatora kućnog letenja na tržištu i sa 37 godina postojanja, najdugovječnija serija PC igara svih vremena i najdugovječnija linija softverskih proizvoda Microsofta. Bio je to jedna od ranijih proizvoda u portfelju Microsoftovih aplikacija izdana tri godine prije Windowsima i značajno se razlikovao od ostalog Microsoftovog softvera koji je uglavnom bio poslovno orijentiran.

Početak Microsoft Flight Simulator bio je skup članaka koje je Bruce Artwick napisao 1976. o programu 3D računalne grafike. Kad je urednik časopisa rekao da pretplatnici žele kupiti program, Artwick je krenuo raditi na njegovom stvaranju i 1977. godine osnovao tvrtku nazvanu subLOGIC Corporation. 1979. godine subLOGIC je objavio FS1 Flight Simulator za Apple II Slika 3 prokazuje prozor igre FS1 pokrenut na Apple II emulatoru.



Slika 3. FS1 Flight Simulator, Apple II, 1979.

SubLOGIC **[8]** je objavio verziju za TRS-80, a 1982. Microsoftu su licencirali IBM PC verziju s CGA grafikom koja je objavljena kao Microsoft Flight Simulator 1.00.

subLOGIC se nastavio razvijati softver za druge platforme i portao je Flight Simulator II na Apple II 1983. godine.

U međuvremenu je Bruce Artwick napustio subLOGIC i osnovao The Bruce Artwick Organisation kako bi nastavio svoj rad na sljedećim Microsoftovim izdanjima, počevši od Microsoft Flight Simulator 3.0 1988. godine.

Microsoft je nastavio proizvoditi novije verzije softvera **[9]** prikazane na slici 4 , dodavajući značajke poput novih tipova zrakoplova i proširenog krajolika.

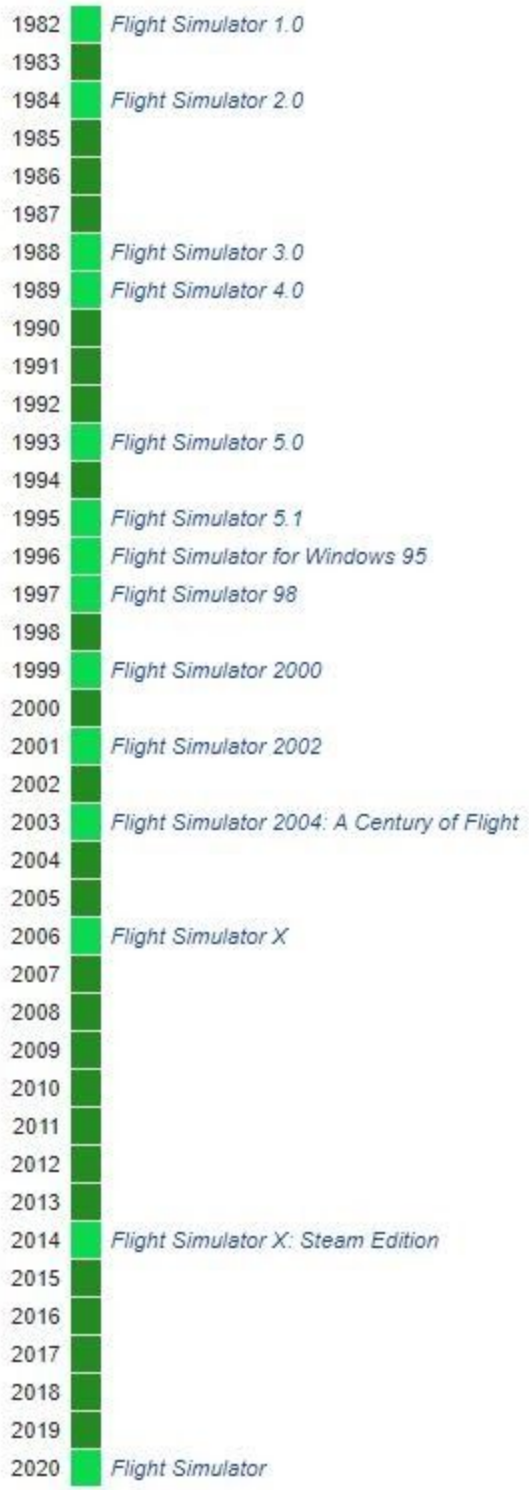
Verzije iz 2000. i 2002. bile su dostupne u izdanjima "Standard" i "Professional", gdje je Professional verzija uključivala više zrakoplova, alata i opcija krajolika.

Izdanje 2004. (verzija 9) obilježila je proslavu sto godina leta s motorom i imala je samo jedno izdanje.
Flight Simulator X **[10]**, objavljen 2006. godine, vratio se u dvostruke verzije s izdanjima "Standard" i "Deluxe".

Područje letenja obuhvaća planet Zemlju i uključuje preko 24,000 zračnih luka.

Tri najnovije verzije uključuju sofisticiranu simulaciju vremena, zajedno s mogućnošću preuzimanja podataka o vremenu u stvarnom svijetu (prvi put dostupno u Flight Simulator 2000).

Dodatne značajke u novijim verzijama uključuju okruženja zračnog prometa s interaktivnim funkcijama kontrole zračnog prometa, nove modele zrakoplova, interaktivne lekcije.



Slika 4. Flight Simulator Verzije

Microsoft Flight Simulator **[11]** (poznat kao Microsoft Flight Simulator 2020) simulator je leta koji je razvio Asobo Studio, a objavio Xbox Game Studios za Microsoft Windows.

Objavljen je 18. kolovoza 2020. za Microsoft Windows. To je jedanaesti glavni unos u seriji Microsoft Flight Simulator, kojemu prethodi Flight Simulator X.

Flight Simulator simulira cijelu Zemlju koristeći teksture i topografske podatke s Bing Maps **[21]**. Trodimenzionalni prikazi površine zemlje, poput terena, drveća, trave, zgrada i vode, generirani su tehnologijom Microsoft Azure.

Azure **[12]** se koristi za prikazivanje vizuala, poboljšanje vizualne vjernosti i simulaciju podataka i efekata iz stvarnog svijeta, kao i izračuna fizike i smatran je vrhuncem Microsoftove "power of the cloud" mantre.

Slika 5 prikazuje grad Rijeku u igri Microsoft Flight Simulator.

Slika 5. Microsoft Flight SImulator 2020

3. Alati za razvoj video igara

U ranoj povijesti industrije video igara **[13]**, alata za programiranje igara nije bilo. Međutim, ovo nije bila prepreka za game developere u to vrijeme.

Slike lika igrača su se crtale okvir po okvir naredbama izvornog koda. Čim je tehnološki naprednija upotreba spriteova postala uobičajena, počeli su se pojavljivati alati za razvoj igara, koje je programer prilagođavao. Danas alate za razvoj igara još uvijek često izrađuju članovi tima programera koji razvijaju i održavaju alate.

3.1 Unity Engine

Unity **[14]** je alat za razvoj igara na više platformi koji je razvio Unity Technologies, najavljen i objavljen u lipnju 2005.

Od 2018. godine engine je proširen tako da podržava više od 25 platformi, može se koristiti za stvaranje trodimenzionalnih, dvodimenzionalnih igara virtualne stvarnosti i proširene stvarnosti, kao i simulacija.

Unutar 2D igara, Unity omogućuje uvoz spriteova i naprednog 2D world rendera. Za 3D igre, Unity omogućuje kompresije teksture, mipmapa i postavke razlučivosti za svaku platformu koju igra podržava, pruža podršku za mapiranje izbočina, mapiranje refleksije, mapiranje paralaksa, okluzija ambijenta prostora (SSAO), dinamička sjene, efekata renderiranja tekstura i efekata naknadne obrade na cijelom zaslonu.

Od 2018. godine Unity se koristi za stvaranje približno polovice novih mobilnih igara na tržištu i 60 posto sadržaja proširene stvarnosti i virtualne stvarnosti.

Za ovaj projekt korištena je verzija 2019.3.14f1.

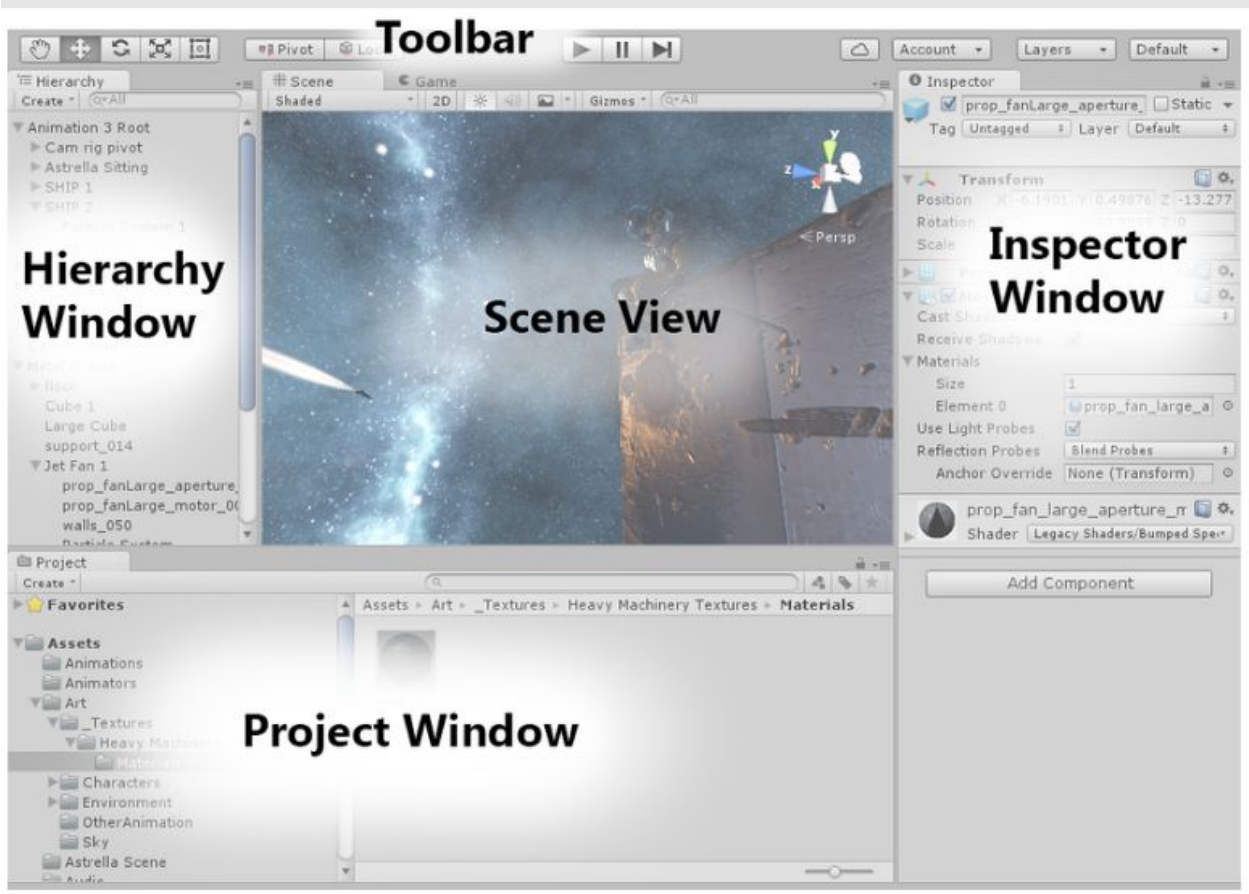
3.2 Adobe Photoshop

Adobe Photoshop **[15]** je uređivač rasterske grafike koji je razvio i objavio Adobe Inc. za Windows i macOS. Izvorno su ga stvorili 1988. godine Thomas i John Knoll. Od tada je softver postao industrijski standard ne samo u uređivanju rasterske grafike, već i u digitalnoj umjetnosti u cjelini. Photoshop može uređivati i komponirati rasterske slike u više slojeva i podržava maske, alfa komponiranje (postupak kombiniranja jedne slike s pozadinom kako bi se stvorio izgled djelomične ili potpune prozirnosti) i nekoliko modela boja.
Photoshop koristi vlastite formate datoteka PSD i PSB kako bi podržao ove značajke.

4. Upoznavanje sa Unity alatom

4.1 Unity Interface

Unity: Interface/Window
Prozor Unity alata, sastoji se od malih pojedinačnih prozora koji se mogu preuređivati, grupirati u setove, odvojiti od jednog položaja i vratiti natrag. Izgled alata može se razlikovati od projekta do projekta i od programera do programera. Slika 6 prikazuje prozor Unity alata.



Slika 6. Unity Interface/Window

Project Window
Prikazati će biblioteku objekata koi su dostupni za upotrebu te slike, glazbene datoteke i druge dodatne datoteke koje su implementirane u projekt.

Scene View
Omogućuje vizualnu navigaciju i uređivanja scene koju se stvara. Ovaj prikaz može biti 2D, kao i 3D prikaz, s obzirom na tip projekta na kojem se radi. U ovom prikazu mogu se premještati objekti.

Hierarchy Window
Prikazuje hijerarhijski prikaz svakog dostupnog objekta na sceni te kako se objekti vežu jedni za druge. Budući da je cijela Scena roditeljski objekt, dodani joj objekti postaju objekt djeteta, u Unity svijetu ovaj koncept poznat je i pod nazivom “Parenting”.

Inspector Window
Omogućuje programerima da pregledaju, analiziraju i uređuju sva svojstva odabranog objekta, različiti tipovi objekata imaju različit niz svojstava s različitim izgledom i sadržajem. Na primjer, kada odaberete Asset ispred prozora projekta, prozor inspektora prikazat će sve dostupne informacije o istom sa svojstvima koja se mogu uređivati.

Toolbar Window
Ovo je najvažniji prozor u programu Unity. S lijeve strane sadrži primarne alate za upravljanje prikazom scene zajedno s objektima koji se nalaze u njemu.

Kontrole reprodukcije, pauze i koraka također su dostupne u ovom prozoru. Pristup uslugama Unity Cloud-a također se mogu dobiti pomoću gumba s desne strane i Unity računa, plus izbornik vidljivosti i izbornik izgleda uređivača koji će pružiti neke alternativne izglede za prozore uređivača.

Game View
U ovom se prozoru prikazuje stvarni prikaz igre s kamerom u igri. To omogućuje provjeru stvarnog prikaza igre.

4.2 Elementi projekta

Postoji niz elemenata (ili komponenata) koji zajedno tvore Igru, razvijenu pomoću Unity alata. Ovi elementi igraju glavnu ulogu u tome da igra bude interaktivna, kao i u dodavanju značajki koje mogu izraziti cilj igre.

Assets
Assets su prikazi objekata koji se implementiraju u igru ili projekt. Objekt može biti datoteka uvezena izvan Unityja, poput 3D modela, audio i zvučne datoteke, slika (jpeg, gif, png itd.), tekstura ili bilo koje druge vrste datoteka koje podržava Unity.

Primjeri asset-a:

- Animator Controller
- Audio Mixer
- Render Texture
- Slike
- Datoteke za animacije

The Project

Projekt je mapa ili mjesto u kojem se nalazi cjelokupni projekt igre zajedno sa svim povezanim Asset-ovima koja može sadržavati i biblioteke te podmape asset-ova.

Packages

To je unaprijed sastavljena skupina game asset-a. Unity dolazi sa raznim paketima koji sadrže skup game asseta.

GameObject

Svaki objekt koji je prisutan u igri je GameObject. Tehnički, oni ne dodaju nikakvu funkcionalnost projektu, već samo djeluju kao držači komponenata poput komponenata Transform, Light, Script i RigidBody.

Components

Komponente su osnovni građevni blokovi, tj. matice i vijci objekata i njihove aktivnosti u igri. Oni djeluju kao funkcionalni dijelovi za svaki GameObject. Prema zadanim postavkama, svaki GameObjects automatski postavlja Transform Component, jer on diktira gdje se GameObject nalazi u Unity okruženju i kako se okreće i skalira.

Scenes

Scene se mogu definirati kao osnovni ili nadređeni objekt, gdje možete smjestiti svoje GameObjekte kako bi napravili neki level igre. Jedna ili više scena obično se stavljaju u igru i povezuju se, što će igrač prijeći ili proći jasno određenim ciljevima. Svi ovi ciljevi i logika igre bit će stavljeni u metode koje će se izvoditi zajedno s igračem unutar scene.

Prefab

Prefabovi su GameObject komponente za višekratnu upotrebu koje su postavljene u prozoru Project View. Prefabovi se mogu uvesti u bilo koji broj scena, bilo koliko puta. Nije važno koliko primjeraka postoji u projektu; kad napravimo bilo kakve promjene na Prefabu, možemo je vizualizirati i primijeniti na sve ostale instance.

Build

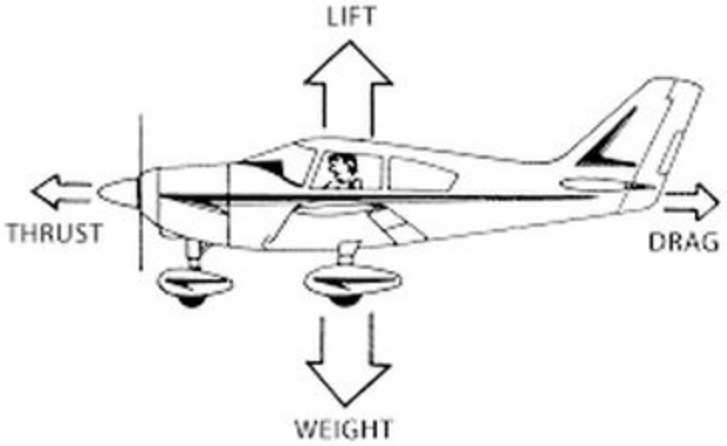
Izvezena je adaptacija igre koja će sadržavati sve bitne scene za reprodukciju na određenoj platformi.

5. Planiranje i opis igre

5.1 Proučavanje Aerodinamike leta

Aerodinamika **[16]** je način na koji se zrak kreće oko stvari. Pravila aerodinamike objašnjavaju kako je zrakoplov sposoban letjeti. Sve što se kreće zrakom reagira na aerodinamiku. Aerodinamika djeluje čak i na automobile, jer zrak struji oko automobila.

Četiri sile leta **[17]** su uzgon, potisak (thrust), otpor zraka (drag) i težina, prikazane na slici 7.



Slika 7. Četiri sile leta

Krila drže zrakoplov u zraku, ali četiri sile to omogućavaju. Guraju avion gore, dolje, naprijed ili ga usporavaju.

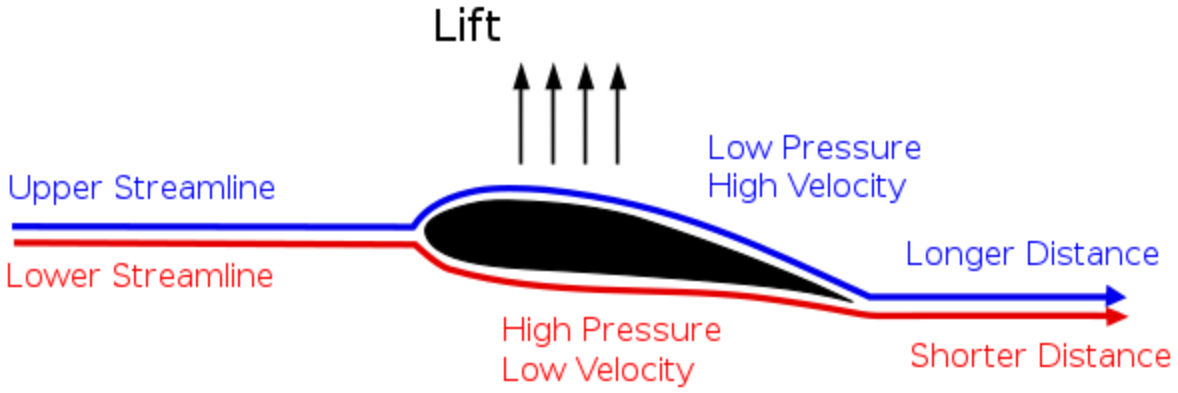
Potisak je sila koja pomiče zrakoplov u smjeru kretanja. Stvara se s propelerom, mlaznim motorom ili raketom. Zrak se uvlači, a zatim izbacuje u suprotnom smjeru. Jedan od primjera je ventilator za kućanstvo.

Otpor zraka je sila koja djeluje suprotno smjeru kretanja. Nastoji usporiti objekt. Uzrokovana trenjem i razlikama u tlaku zraka. Primjer je stavljanje ruke kroz prozor automobila koji se kreće.

Težina je sila uzrokovana gravitacijom.

Uzgon je sila koja drži zrakoplov u zraku. Krila stvaraju većinu uzgona koji koriste zrakoplovi.

Način na koji četiri sile djeluju na zrakoplov tjera avion da radi različite stvari. Svaka sila ima suprotnu silu koja djeluje protiv nje. Uzgon djeluje suprotno težini. Potisak djeluje suprotno od otpora zraka. Kada su snage uravnotežene, avion leti u ravnom smjeru. Avion se uspinje ako su sile uzgona i potiska veće od gravitacije i otpora. Ako su gravitacija i otpor veći od podizanja i potiska, avion se spušta. Tlak zraka je veći na donjoj strani krila, pa je gurnut prema gore.
Slika 8 prikazuje stvaranje uzgona zbog razlike u pritisku zraka.



Slika 8. Stvaranje uzgona (Lift)

5.2 Dizajn koda

Stvaranje nekog vizalnog pregleda koda pomoću besplatnog online draw.io alata
Svaki pravokutnik predstavlja pojedinu skriptu. Te skripte će se kasnije pridruživati Objektima u igri (GameObject) kako bi se stvorila funkcionalnost objekata i omogućili interakciju istih.

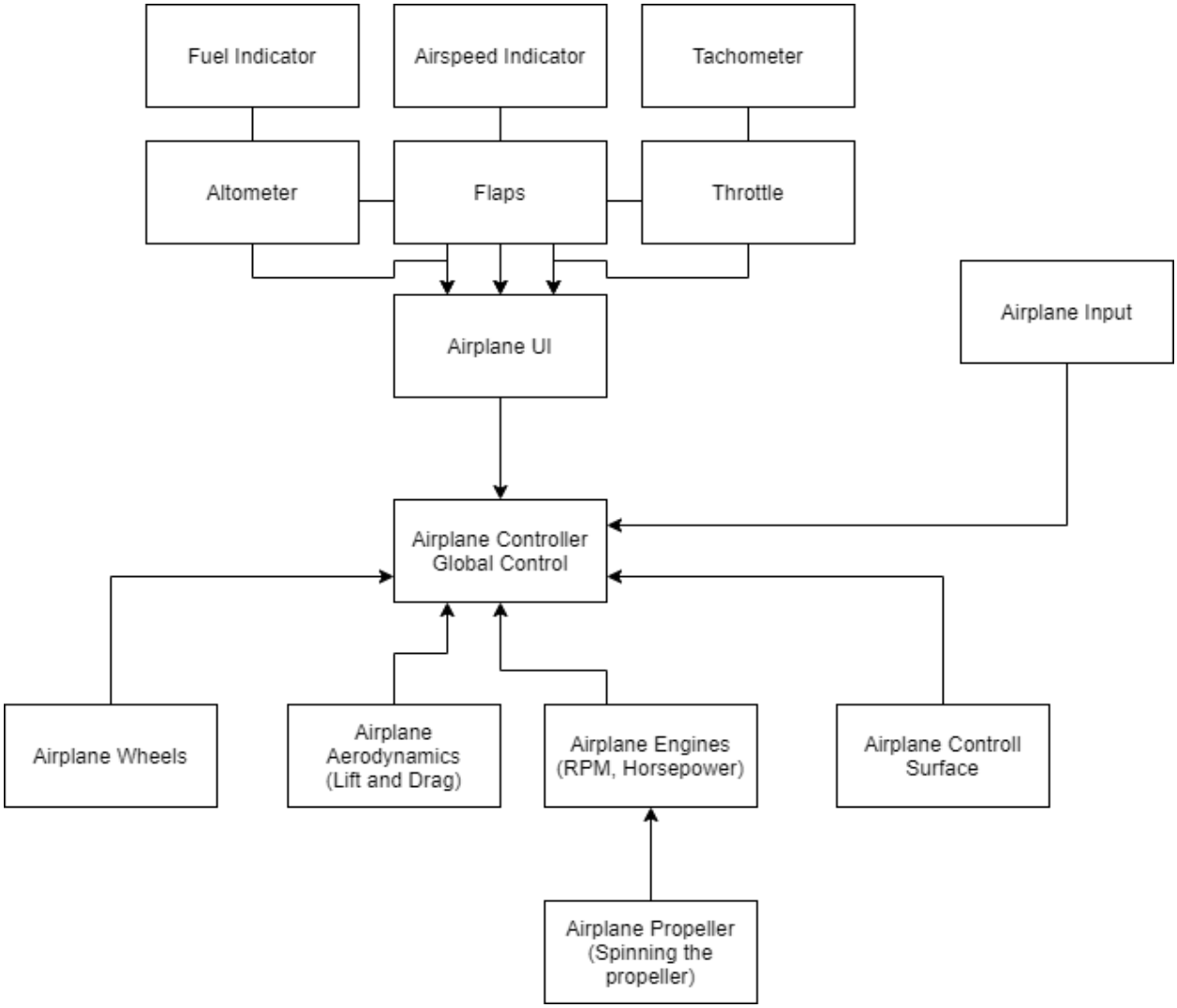
Prva i glavna C# skripta koju ćemo stvoriti je **Airplane Controller**, i to je skripta koja će upravljati svim ostalim komponentama i skriptama, Airplane Controller skripta će prosljeđivati većinu globalnih podataka ostalim skriptama.

Airplane Aerodynamics skripta će biti zadužena sa izračune i upravljanje aerodinamike leta tj. uzgon i otpor zraka.

Airplane Input skripta služi za procesiranje inputa sa kontrolera ili tipkovnice i prosljeđivanje istih u Airplane Controller kako bi imali funkcionalnost upravljanja zrakoplovom i pojedinim komponentama zrakoplova poput njihovih pozicija u sceni i rada motora.

Stvaramo **Airplane Engine** skriptu jer postoji više vrsti zrakoplova sa više vrsti motora pa ćemo imati jednu skriptu u kojoj ćemo moći mijenjati podatke o motoru, skripta će zatim te podatke obrađivati i proslijediti Airplane Controller skripti (Računanje RPM, Konjskih snaga) koje ćemo koristiti za brzinu vrtnje propelera za koje je zadužena **Airplane Propeller** skripta.

Airplane Control Surfaces skripta je zadužena za vizualnu reprezentaciju interakcije sa dijelovima aviona poput pomicanja zakrilca. Slika 9 prikazuje povezanost svih skripti.



Slika 9. Vizualna reprezentacija koda

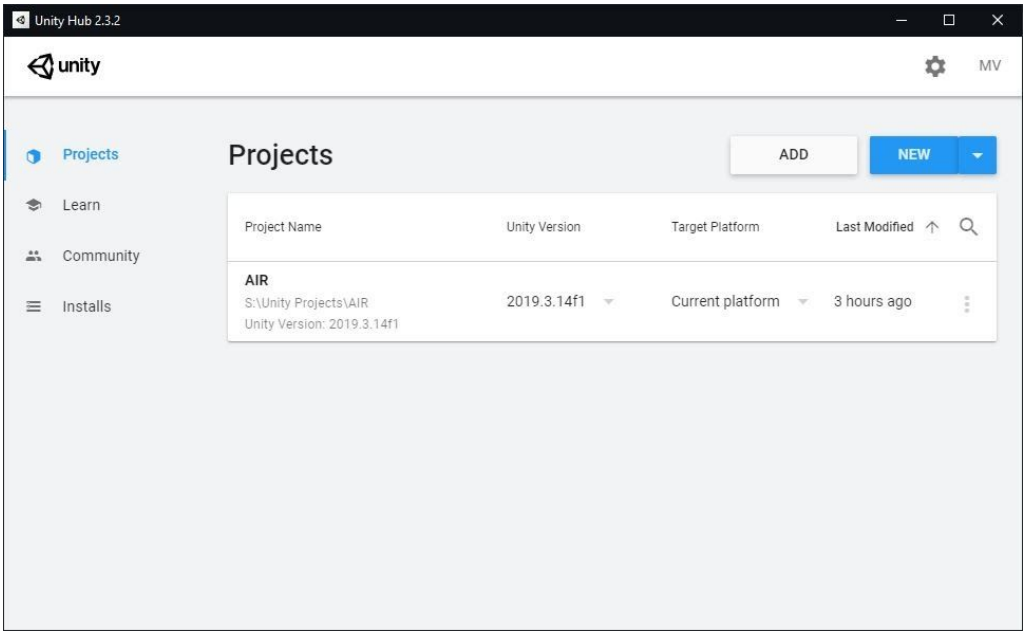
Airplane Wheel, skripta provjerava način upravljanja zrakoplovom na zemlji jer prednji kotači imaju kočnice dok zadnji služe samo za upravljanje na zemlji, ali način upravljanje ovisi od zrakoplova do zrakoplova. Podaci o upravljanju će se proslijediti Airplane Controller skripti

Airplane UI komponenta će primati informacije od Airplane Controller skripte i te podatke prosljeđivati određenim skriptama za upravljanje pojedinih UI elemenata za prikaz informacija o zrakoplovu ili letu kao što su. (Flaps - Zakrilca, Throttle - pogon, Altometer - visina, Airspeed - brzina zrakoplova). Fuel indicator i Tachometer su komponente koje nisu potpuno implementirane u završnoj verziji.

6. Izrada AIR: Airplane Immersive Racing igre

6.1 Stvaranje Projekta i početne scene

Prvi korak kod kreiranja igre je skinuti Unity Hub i odgovarajuću Unity verziju alata te kreirati novi projekt na toj verziji Unity-a. Slika 10 prikazuje Unity Hub prozor.



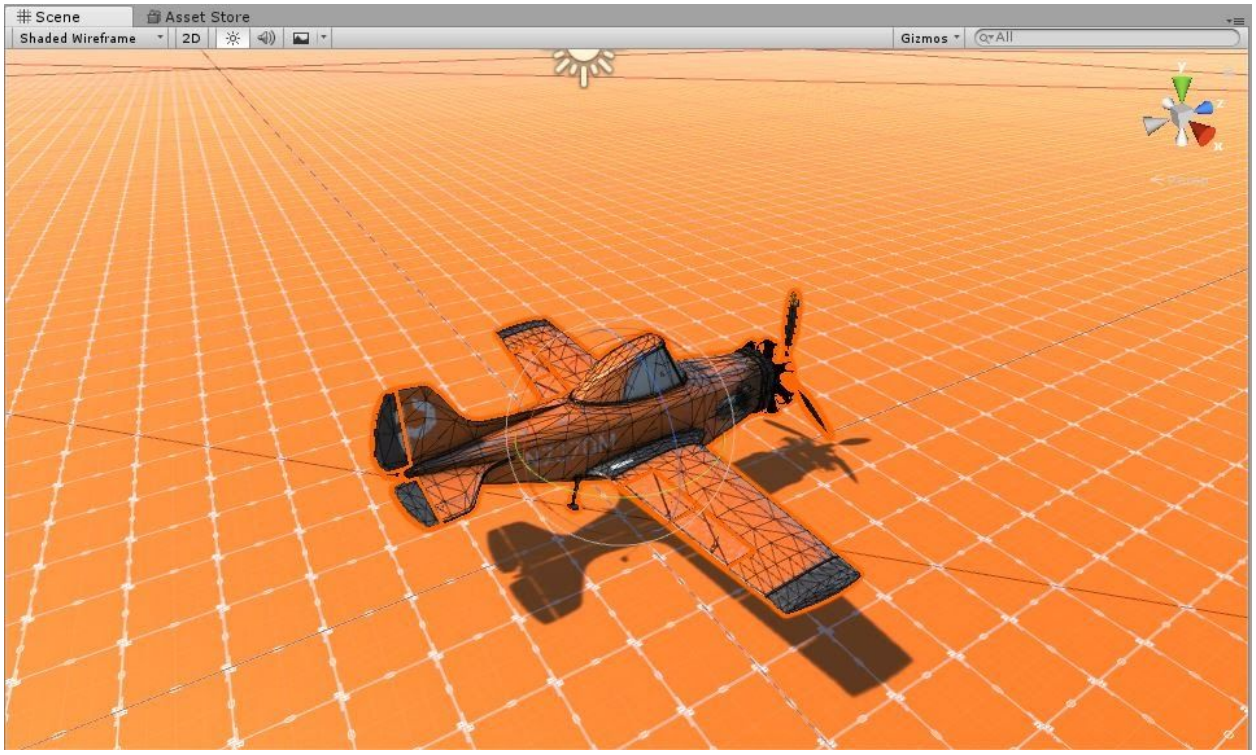
Slika 10. Unity Hub Projekt

Zatim otvaramo projekt i importamo sve Asset-ove koji su nam potrebni za izgradnju igrice poput 3D modela, tekstura, slika i zvukova.

3D model zrakoplova koji se koristi u ovoj igri je preuzet sa Interneta. Dakle njegovo modeliranje, rigging, teksturiranje i ostali komplicirani procesi nisu uključeni u izradu ovog projekta. Ostali asset-ovi koji su preuzeti sa interneta su zvukovi motora i prolaska kroz vrata staze, teren i pripadne teksture, skybox teksture, slike korištene za UI elemente i još neki elementi, a njihova upotreba će biti kasnije opisana.
Nakon importanja svih potrebnih elemenata postavljamo početnu testnu scenu u kojoj ćemo testirati sve funkcionalnosti igre i vidjeti kako naša igra izgleda u trenutnoj sceni.

Bitna je i organizacija datoteka. Potrebno je isplanirati njihovu hijerarhiju te posebno odvojiti različite tipove podataka poput skripti od umjetničkih dokumenata poput grafika i zvukova. Isto tako je pametno odvojiti zvukove od grafičkih elemenata te posebno napraviti folder za teksture, 3D modele, prefabove, UI elemente i sl.

Nakon kreiranja nove scene, u istu je importan Default_Plane prefab koji sadrži 3D model zrakoplova pripremljen za igru tako da je odvojen na dijelove (rep, zakrilca, kotači, krila...itd) te privremeni objekt pravokutne površina sa kolizijom koja nam služi za testiranje. Slika 11 prikazuje scenu za testiranje mehanika igre koje su u razvoju.



Slika 11. Početna scena za testiranje

6.2 Priprema Zrakoplova

Kako bi se zrakoplov ponašao prirodno i kao bi na njega djelovala pravila fizike moramo napraviti **Collider** objekte za pojedine dijelove zrakoplova poput krila, tijela zrakoplova, repa, kotača i propelera. Oni su prikazani na slici 12.

Koriste se Box, Capsule ili Wheel Collider komponente ovisno o obliku dijela zrakoplova za koji se on koristi.
Wheel Collider je posebna vrsta komponente specifična za kotače jer možemo posebno kontrolirati trijenje, amortizaciju, radijus, klizanje i ostale podatke posebne za kotače.

Slika 12. Plane Colliders

Moramo stvoriti i objekt koji će nam dati podatak o centru gravitacije zrakoplova koji ćemo nazvati **COG** i postavit ćemo ga ovisno o tome gdje bi se otprilike trebao nalaziti centar gravitacije.

Trebat će nam i **Engine** objekt koji će sadržavati Airplane Engine skriptu u kojoj se mogu mijenjati parametri ovisno o specifikaciji motora zrakoplova (Max Force, Max RPM, Power Curve) i koja ujedno kontrolira vrtnju motora.

Control Surfaces je objekt u kojem se nalaze pojedini objekti za kontrolu vizualnih elemenata zrakoplova pomoću skripti, tj. mijenjanje njihovih pozicija i rotacije ovisno o unosu sa tipkovnice.

Objekt **Audio_GRP** grupa je objekata koji sadrže audio komponente za zvukove motora u nepokretnom stanju i kod punog pogona.

Cockpit_Camera sadrži komponentu kamere za pogled iz prvog lica kokpita i post processing komponentu za bolju vizualnu reprezentaciju isto kao i glavna kamera.

Camera_Controller je objekt koji kontrolira položaj kamere ovisno o položaju zrakoplova i njegove brzine i udaljenosti od tla, te ovisno o pritisku tipke mijenja kameru iz trećeg u prvo lice ili obrnuto. Slika 13 prikazuje Default_Plane prefab koji sadrzi sve ove elemente

Slika 13. Default_Plane

6.3 Procesuiranje Inputa

Skripta **BaseAirplane_Input.cs** je skripta koja procesira različite unose tipkovnice (ako nismo pauzirali igru) .

```
protected virtual void HandleInput(){
            //Main Control Input
            pitch = Input.GetAxis("Vertical");
            roll = Input.GetAxis("Horizontal");
            yaw = Input.GetAxis("Yaw");
            throttle = Input.GetAxis("Throttle");
            //Brake inputs
            brake = Input.GetKey(brakeKey)? 1f : 0f;
            //Flap Inputs
```

```
            if(Input.GetKeyDown(KeyCode.F)){
                flaps +=1;
            }
            if(Input.GetKeyDown(KeyCode.G)){
                flaps -=1;
            }
            flaps = Mathf.Clamp(flaps, 0, maxFlapIncrements);
            cameraSwitch = Input.GetKeyDown(cameraKey);
        }
```

Virtualna metoda **HandleInput()** se poziva za svaki renderirani okvir (frame igre) u Update metodi koja je ugrađena metoda u Unity alatu i poziva se za svaki okvir.
“Vertical”, “Horizontal”, “Yaw”, i “Throttle” su tipke definirane u input editoru unutar Unity alata i moramo ih postaviti na vrijednosti koje želimo pod **Edit->Project Settings->Input Manager**
Naprimjer, tipka F se koristi za spuštanje zakrilca (Input.GetKeyDown(KeyCode.F)).

Postoji i **BaseAirplaneInput_Editor** skripta koja nam uglavnom služi za Debugging tako da vidimo dobivamo li vrijednosti za pritisnute tipke u inspector-u.

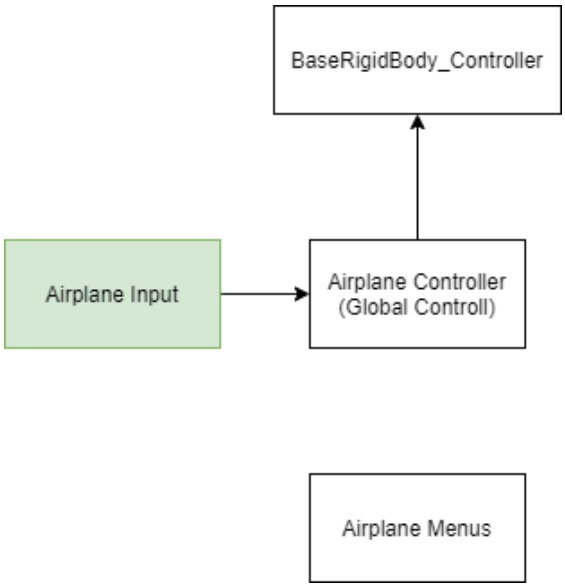
Osim tipkovnice i miša možemo koristiti i **Xbox kontroler** za koji postoji posebna skripta konfigurirana za Xbox unose koja nadjačava virtualnu metodu HandleInput() iz BaseAirplane skripte.

6.4 Airplane Controller skripta

Glavna skripta koja je središte svih funkcionalnosti koje ima zrakoplov i služi kao globalni kontroler svih komponenata sa kojima je povezana.

Ova skripta surađuje sa drugim komponentama i procesuira podatke koje iste proizvode te iz tih podataka dolazi do kranje fizike koja utječe na let zrakoplova poput težine zrakoplova.

Postavljena je kao komponenta na Default_Plane prefab na kojem je postavljena Rigidbody komponenta koja omogućuje upravljanje pozicijom tog objekta upotrebom fizike. Na slici 14 možemo vidjeti da Airplane Input prosljeđuje unose AIrplane Controller skripti koja pritom ovisno o unosu utječe na RigidBody objekt.



Slika 14. Airplane Controller

6.5 Motor i propeleri

Ova komponenta sadrži skriptu **Airplane_Engine.cs** koja kontrolira silu potiska (*finalForce*) koju generira motor zrakoplova i prosljeđuje je Airplane_Controller.cs skripti za pomicanje RigidBody objekta te podatke o vrtnji **Airplane_Propeller.cs** skripti koja ovisno o brzini okretaja rotira propelere.

Snaga motora (*finalThrottle*) se povećava koristeći animacijsku krivulja (*powerCurve.Evaluate*) koju podesimo željenim vrijednostima tako da postepeno dođemo do maksimalne snage. *Mathf.Clamp01(throttle)* limitira throttle na vrijednosti između 0 i 1.

```
public Vector3 CalculateForce(float throttle){
            // Calculate Power
            float finalThrottle = Mathf.Clamp01(throttle);
            finalThrottle = powerCurve.Evaluate(finalThrottle);
            // Calculate RPM
            currentRPM = finalThrottle * maxRPM;
            if(propeller)
            {
                propeller.HandlePropeller(currentRPM);
            }
            // Calculate Force
            float finalPower = finalThrottle * maxForce;
            Vector3 finalForce = transform.forward * finalPower;
            return finalForce;
        }
```

6.5.1 Animiranje propelera

Airplane_Propeller.cs skripta je zadužena za rotaciju probelera ovisno o broju okretaja koje nam daje Engine komponenta (*currentRPM*). U skripti se računaju stupnjevi po sekundi “dps” i taj podatak se koristi za rotaciju vektora propelera.

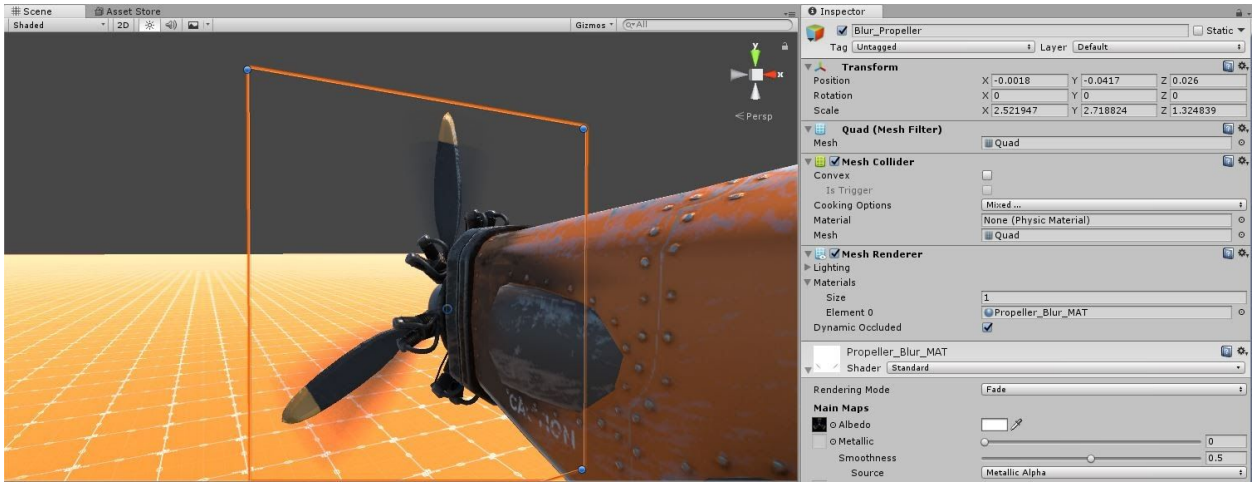
*dps = (((currentRPM * 360f) / 60f) + minRotationRPM) * Time.deltaTime;*

Kako bi dobili grafiku propelera koji se vrti, izoliran je objekt propelera u posebnu scenu i napravljen screenshot istoga, on je uvezen u Adobe Photoshop alat. Tamo je maknuta pozadina i stavljen efekt “Radial Blur” na sliku propelera. Ta slika je exportana iz Photoshop alata i uvezena nazad u Unity alat gdje Airplane_Propeller skripta mijenja texturu propelera ovisno o RPM podatku koje dobiva od Engine-a.

Propeler ima 3 teksture, normalna, sa blagim zamućenjem, i sa jakim zamučenjem koje se mijenjaju ovisno o okretajima po minuti. Naprimjer *minTextureSwap* = 600, što znači da ako je *currentRPM* > 600 onda se textura promijeni na blurLevel2. A sve dok je *currentRPM* < *minQuadRPMs* = 30 blurredProp element će biti isključen, nakon 30 on se aktivira dok se ujedno i deaktivira glavni 3D model propelera. Na slici 15 možemo vidjeti zamućenu teksturu propelera koja se nalazi na istoj poziciji kao i 3D objekt.

```
void HandleSwapping(float currentRPM){
            if(currentRPM > minQuadRPMs){
                blurredProp.gameObject.SetActive(true);
                mainProp.gameObject.SetActive(false);

                if(blurredPropMat && blurLevel1 && blurLevel2){
                    if(currentRPM > minTextureSwap){
                        blurredPropMat.SetTexture("_MainTex", blurLevel2);}
                    else{
                        blurredPropMat.SetTexture("_MainTex", blurLevel1);
                    }
                }
            }
            else{
                blurredProp.gameObject.SetActive(false);
                mainProp.gameObject.SetActive(true);}}
```



Slika 15. Propeller Blur

6.6 Karakteristike leta

Airplane_Characteristics.cs je glavna skripta za procesiranje i računanje glavnih sila potrebnih za let koje se koriste za kreiranje fizike koja utječe na poziciju RigidBody objekta i metode koje koristi za postizanje tih rezultata su navedene u kodu dolje.

```
public void UpdateCharacteristics() {
            if(rb){
                CalForwardSpeed();
                CalLift();
                CalDrag();
                HandleRigidBodyTransform();
                HandleControlSurfaceEfficiency();
                HandlePitch();
                HandleRoll();
                HandleYaw();
                HandleBanking();
            }
        }
```

6.6.1 Forward Speed

Metoda CalForwardSpeed(); se koristi za kalkuliranje brzine zrakoplova (*forwardSpeed*) relativno sa samim sobom (*localVelocity* u *CalFOrwardSpeed* metodi) tj. Koliko metara na sat se tijelo giba. Ove vrijednosti koriste se u izračunu uzgona koji stvaraju krila zrakoplova ovisno o brzini gibanja zrakoplova. *Mathf.Max(0f, localVelocity.z)* vraća veću od dvije vrijednosti dok *normalizedMPH* = *Mathf.InverseLerp* daje postotak "*mph*" ovisno o "*0*" i "*maxMPH*"

```
void CalForwardSpeed() {
            Vector3 localVelocity =
transform.InverseTransformDirection(rb.velocity);
            forwardSpeed = Mathf.Max(0f, localVelocity.z);
            mph = forwardSpeed * mpsToMph;
            normalizedMPH = Mathf.InverseLerp(0f, maxMPH, mph);}
```

6.6.2 Uzgon i napadni kut

Uzgon (*finalLiftForce* u CalLift() metodi) je koja se stvara zbog razlike u tlaku između gornje i donje površine krila. Bernulijev jednadžba **[18]** nam govori da zrakoplov može generirati uzgon zbog oblika svojih krila.
Krila su oblikovana tako da se zrak giba brže preko gornjeg dijela krila a sporije ispod. Zrak koji se brzo giba stvara područje manjeg pritiska dok je kod sporijeg obrnuto. Prema tome, zrak većeg pritiska ispod površine krila će gurati zrakoplov prema zraku manjeg pritiska kako bi se pritisak izjednačio. *Vector3 liftDir = transform.up* je smjer sile uzgona, a to je prema gore.

```
void CalLift(){
            angleOfAttack = Vector3.Dot(rb.velocity.normalized,
transform.forward);
            angleOfAttack *= angleOfAttack;
            Vector3 liftDir = transform.up;
            float liftPower = liftCurve.Evaluate(normalizedMPH) *
maxLiftPower;
            float finalLiftPower = flapLiftPower * input.NormalizedFlaps;
            Vector3 finalLiftForce = liftDir * (liftPower + finalLiftPower)
* angleOfAttack;
            rb.AddForce(finalLiftForce);}
```

Kut napada (**Angle of attack,** varijabla *angleOfAttack* u CalLift() metodi) **[19]** je kut između nadolazećeg zraka i referentne crte na zrakoplovu ili krilu. Koeficijent dizanja zrakoplova s fiksnim krilima varira ovisno o kutu napada. Povećavajući kut napada povezano je s povećanjem koeficijenta podizanja do maksimalnog koeficijenta podizanja, nakon čega se koeficijent podizanja smanjuje.

Zbog toga koristimo Vector3.Dot kako bi izračunali dot produkt vektora zrakoplova (smjer krila, *rb.velocity.normalized*) i vektora koji nosi RigidBody brzinu (*transform.forward*).
Kako se kut napada zrakoplova s fiksnim krilom povećava, odvajanje protoka zraka od gornje površine krila postaje sve izraženije, što dovodi do smanjenja brzine povećanja koeficijenta dizanja.

6.6.3 Otpor Zraka

Otpor je aerodinamička sila koja se suprotstavlja kretanju zrakoplova kroz zrak. Otpor generira svaki dio zrakoplova i generira se razlikom u brzini između čvrstog predmeta i fluida. Prema tome veće brzine generiraju veći otpor zraka.

Metoda koja procesira ovu silu naziva se CalDrag() i njezin kod je:

```
void CalDrag(){
            float speedDrag = forwardSpeed * dragFactor;
            float flapDrag = input.Flaps * flapDragFactor;
            float finalDrag = startDrag + speedDrag + flapDrag;
            rb.drag = finalDrag;
            rb.angularDrag = startAngularDraft * forwardSpeed;}
```

Prilikom računanja otpora moramo uzeti u obzir brzinu (*speedDrag*) i poziciju zakrilca (*flapDrag*) jer otpor ovisi o njima.
Povećanje napadnog kuta također povećava otpor zraka. (*rb.angularDrag*)

6.6.4 Upravljanje sa RigidBody tijelom

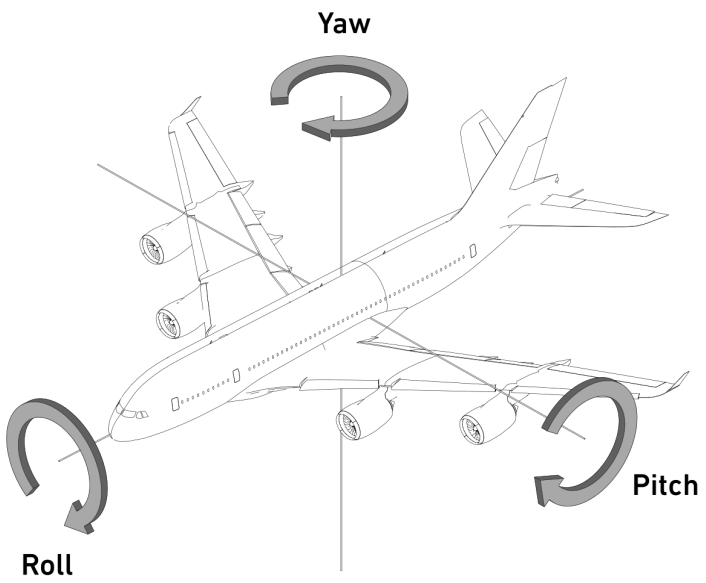
Sve gore izračunate sile koristimo kako bi kalkulirali smjer gibanja (*updatedVelocity*) i rotaciju (*updatedRotation*) tijela sa RigidBody komponentom. Sve smjerove gibanja zrakoplova prikazuje slika 16.

```
void HandleRigidBodyTransform() {
            if(rb.velocity.magnitude > 1f){
                Vector3 updatedVelocity = Vector3.Lerp(rb.velocity,
transform.forward * forwardSpeed, forwardSpeed * angleOfAttack *
Time.deltaTime * rbLerpSpeed);
                rb.velocity = updatedVelocity;
                Quaternion updatedRotation = Quaternion.Slerp(rb.rotation,
Quaternion.LookRotation(rb.velocity, transform.up), Time.deltaTime *
rbLerpSpeed);
                rb.MoveRotation(updatedRotation);}}
```

6.6.5 Pitch, Roll, Yaw i Banking

Pitch (*pitchAngle*) je rotacija zrakoplova po horizontalnoj osi koja je paralelna sa krilima
Za input se koriste “S” (-1) i “W” (1), *pitchSpeed* određuje brzinu nagiba.

```
void HandlePitch(){
            Vector3 flatForward = transform.forward;
            flatForward.y = 0f;
            flatForward = flatForward.normalized;
            pitchAngle = Vector3.Angle(transform.forward, flatForward);
            Vector3 pitchTorque = input.Pitch * pitchSpeed * transform.right
* csEfficiencyValue;
            rb.AddTorque(pitchTorque);}
```



Slika 16. Pitch, Roll, Yaw

Roll (*rollAngle*) je rotacija zrakoplova po horizontalnoj osi koja je paralelna sa tijelom zrakoplova
Za input se koriste tipke “A” (0) i “D” (1), u neutralnom stanju je 0. *rollSpeed* je brzina rotacije.

```
void HandleRoll() {
            Vector3 flatRight = transform.right;
            flatRight.y = 0f;
            flatRight = flatRight.normalized;
            rollAngle = Vector3.SignedAngle(transform.right, flatRight,
transform.forward);

            Vector3 rollTorque = -input.Roll * rollSpeed * transform.forward
* csEfficiencyValue;
            rb.AddTorque(rollTorque);}
```

Yaw (*yawTorque*) je rotacija zrakoplova po vertikalnoj osi okomitoj na zrakoplov
Za input se koriste strelica lijevo (-1) i desno (1). *yawSpeed* određuje brzinu skretanja.

```
void HandleYaw(){
            Vector3 yawTorque = input.Yaw * yawSpeed * transform.up *
csEfficiencyValue;
            rb.AddTorque(yawTorque); }
```

csEfficiencyValue je varijabla koja se povećava ovisno o brzini zrakoplova i sprječava bilo kakvu rotaciju zrakoplova kada on stoji na mjestu tj. Nema dovoljno brzine za kreiranje sila.

Banking [20] (*bankTorque*) se pojavljuje kada zrakoplov rotira po horizontalnoj osi paralelnoj sa tijelom zrakoplova (roll). Ono počinje okretati avion više nego što pilot to određuje sa kontrolama i lagano skreće zrakoplov u lijevu ili desnu stranu. Funkcija *Mathf.Lerp* linearno interpolira vrijednosti između neke dvije početne vrijednosti (*-1f, 1f*) .

```
void HandleBanking(){
            float bankSide = Mathf.InverseLerp(-90f, 90f, rollAngle);
            float bankAmount = Mathf.Lerp(-1f, 1f, bankSide);
            Vector3 bankTorque = bankAmount * rollSpeed * transform.up;
            rb.AddTorque(bankTorque);}
```

6.7 Kamera

Prvo je napravljena **Basic_Follow_Camera.cs** skripta koja može sljediti bilo koji gibajući objekt u igri i prilagođava poziciju (*wantedPosition*) kamere ovisno o visini i brzini objekta koji slijedi. SmoothDamp funkcija postupno mijenja vektor kako bi se kamera glatko gibala.

```
protected virtual void HandleCamera(){
            Vector3 wantedPosition = target.position + (-target.forward *
distance) + (Vector3.up * height);
            transform.position = Vector3.SmoothDamp(transform.position,
wantedPosition, ref smoothVelocity, smoothSpeed);
            transform.LookAt(target);}
```

Druga skripta je **Airplane_Camera.cs** koja nadjačava virtualnu metodu HandleCamera() iz prve skripte. Ona provjerava dali je zrakoplov na zemlji jer želimo da kamera bude na višoj poziciji kada sletimo. *Physics.Raycast* Baca zraku, iz točke *transform.position* (Zrakoplov), u smjeru *Vector3.down* (Dolje), vraća true ako zraka dodiruje collider objekt označenog sa “ground”.

```
protected override void HandleCamera(){
            RaycastHit hit;
            if(Physics.Raycast(transform.position, Vector3.down, out hit)){
                if(hit.distance < minHeightFromGround && hit.transform.tag
== "ground"){
                    float wantedHeight = origHeight + (minHeightFromGround -
hit.distance);
                    height = wantedHeight;}}
            base.HandleCamera();}
```

6.8 Animacija pokretnih dijelova

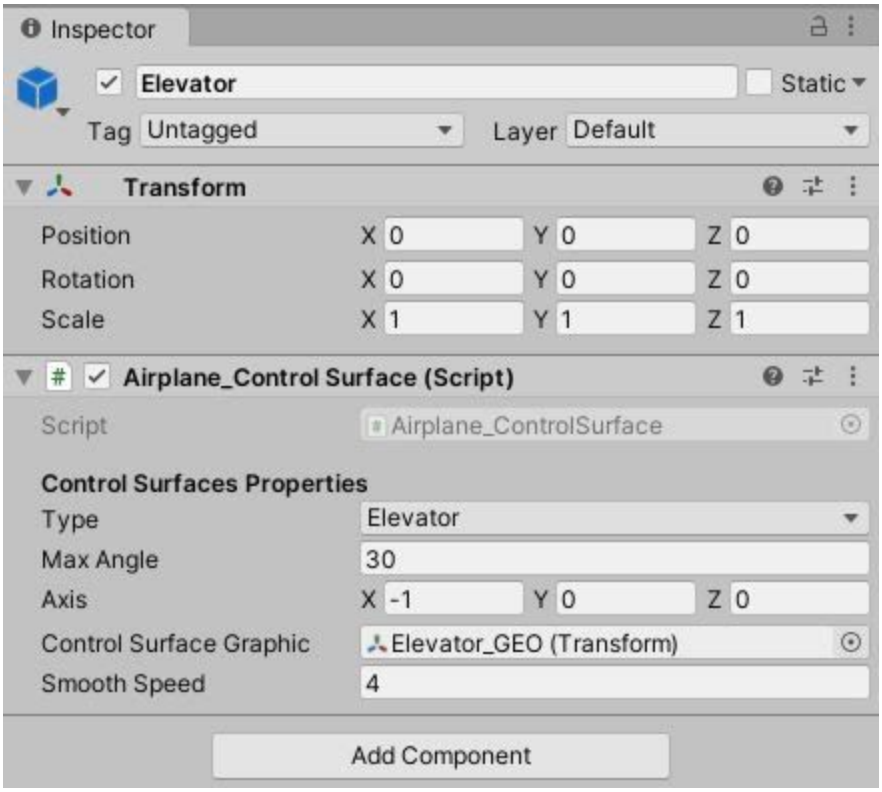
Skripta **Airplane_ControlSurface.cs** kako bi vizualizira gibanje dijelova zrakoplova koji određuju smjer i rotaciju objekta.
Dijelovi zrakoplova zaslužni za to su: krilca, zakrilca, kormilo i dizalo.

Glavni dio skripte je metoda **HandleControlSurface**() koja prima input iz BaseAirplane_Input skripte i ovisno o njemu kontrolira rotaciju pojedinih dijelova zrakoplova za kontroliranje leta.

```
public void HandleControlSurface(BaseAirplane_Input input){
            float inputValue = 0f;
            switch(type){
                case ControlSurfaceType.Rudder:
                    inputValue = input.Yaw;
                    break;
                case ControlSurfaceType.Elevator:
                    inputValue = input.Pitch;
                    break;
                case ControlSurfaceType.Flap:
                    inputValue = input.Flaps;
                    break;
                case ControlSurfaceType.Aileron:
                    inputValue = input.Roll;
                    break;
                default:
                    break;}
            wantedAngle = maxAngle * inputValue;}
```

Za svaki dio zrakoplova moramo napraviti Game Object sa skriptom Airplane_ControlSurface.cs i odrediti u Inspector prozoru da je to odabrani dio zrakoplova te povezati geometrijski dio modela zrakoplova sa skriptom u inspectoru.
Naprimjer ako je grafički element označen sa “Aileron” u komponenti skripte koja je na njemu (u inspector prozoru) onda će se za *inputValue* koristiti *input.Roll ().*
Sve objekte stavljamo pod Control_Surfaces roditelja.

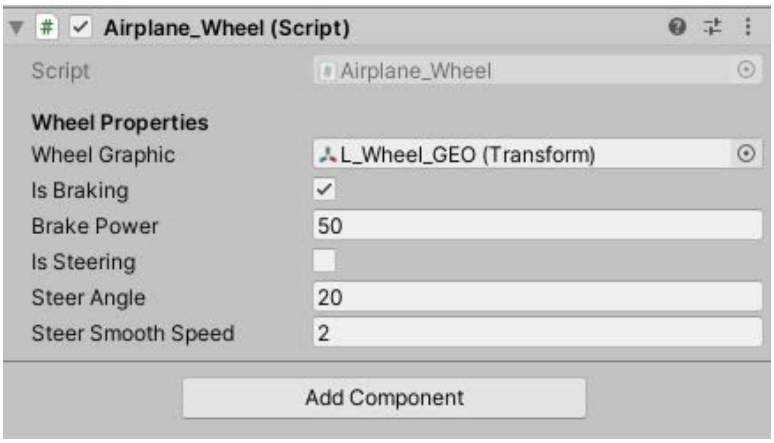
Smooth speed određuje koliko želimo izgladiti animaciju rotacije
Slika 17 prikazuje Inspector prozor za “Control Surface” objekt te prikazuje komponentu skripte koja ga kontrolira. U ovom slučaju tip površine je “Elevator” (moramo povezati grafiku objekta), možemo kontrolirati glatkost animacija i maksimalni kut i po kojim osima se element rotira te.



Slika 17. Control Surface Inspector

6.9 Dovršavanje kotača

U skripti **Airplane_Wheels.cs** određujemo dali kotač služi za kočenje ili skretanje te procesuiramo rotaciju kotača, njegovo kočenje ili skretanje ovisno o inputu koji dobivamo.
Ovu skriptu postavljamo na collision objekte kotača i u inspector prozoru određujemo dali on služi za skretanje ili kočenje. Na slici 18 možemo vidjeti Inspector prozor za element kotača na kojem je komponenta skripte koja upravlja kotačem. Ovdje se određuje dali kotač služi za kočenje ili za skretanje te koji mu je kut i glatkoća skretanja ili moć kočenja.



Slika 18. Airplane_Wheel Inspector

6.10 Ground Effect skripta

Za zrakoplove s fiksnim krilima Ground Effect je smanjeni aerodinamički otpor koji krila zrakoplova generiraju kada su blizu fiksne površine. Pri slijetanju pilotu može dati osjećaj da zrakoplov "lebdi".

Pri polijetanju efekt tla može privremeno smanjiti brzinu vertikalnog zaustavljanja. Tada pilot može letjeti tik iznad piste dok zrakoplov ubrzava zbog efekta i ne postigne sigurnu brzinu uspona.

```
protected virtual void HandleGroundEffect(){
            RaycastHit hit;
            if(Physics.Raycast(transform.position, Vector3.down, out hit)){
                if(hit.transform.tag == "ground" && hit.distance <
maxGroundDistance){
                    float currentSpeed = rb.velocity.magnitude;
                    float normalizedSpeed = currentSpeed / maxSpeed;
                    normalizedSpeed = Mathf.Clamp01(normalizedSpeed);
                    float distance = maxGroundDistance - hit.distance;
                    float finalForce = liftForce * distance *
normalizedSpeed;
                    rb.AddForce(Vector3.up * finalForce); }}}
```

Ova metoda iz skripte **Airplane_GroundEffect.cs** provjerava dali je zrakoplov blizu tla pomoću raycastinga i ovisno o tome dodaje dodatnu silu uzgona (*finalForce*) na već generiranu ovisno o brzini (*currentSpeed*) i udaljenosti (*distance*). Mathf.Clamp01 normalizira vrijednosti brzine na vrijednosti između 1 i 0.

6.11 Kontrola Zvuka

Airplane_Audio.cs skripta je zadužena za kontrolu audio elemenata u igri na način da provjerava koliki pogon dajemo zrakoplovu (*input.StickyThrottle*) i ovisno o jačini istog postepeno mijenja zvuk koji zrakoplov generira iz zvuka nepokretnog zrakoplova u zvuk zrakoplova prilikom punog pogona.

Ujedno povisuje i Pitch zvuka tako da se čini kao da motor brže radi.

HandleAudio() je metoda koja se poziva u ovoj skripti za kontrolu zvuka. *finalVolumeValue* linearno interpolira vrijednosti inputa za pogon motora (*StickyThrottle)* između vrijednosti 0 i 1, *finalPitchValue* radi isto za Pitch ali između vrijednosti 1 i *maxPitchValue*.

```
protected virtual void HandleAudio(){
            finalVolumeValue = Mathf.Lerp(0f, 1f, input.StickyThrottle);
            finalPitchValue = Mathf.Lerp(1f, maxPitchValue,
input.StickyThrottle);
            if(fullThrottleSource){
                fullThrottleSource.volume = finalVolumeValue;
                fullThrottleSource.pitch = finalPitchValue;}}
```

6.12 Prikazivanje i ažuriranje UI elemenata

Izgradnja 2D UI elemenata započinje izgradnjom glavnog Canvas objekta na koji ćemo postavljati sve UI elemente u obliku slika koje ćemo mijenjati ovisno o podacima koje dobivamo.

Canvas Render mode mora biti postavljen na Screen Space - Overlay. Ovaj način prikazivanja postavlja elemente korisničkog sučelja na zaslon prikazan na vrhu scene. Ako se promijeni veličina ili razlučivost zaslona Canvas će automatski promijeniti veličinu kako bi se podudarala s promjenom.

Kod skriptiranja UI elemenata koriste se interface **IAirplaneUI.cs** koji poziva metodu HandleAirplaneUI() što znači da svaka skripta koja nasljedi ovaj interface mora pozvati tu metodu.

Glavna skripta koja kontrolira UI je **AirplaneUI_Controller.cs** koja za svaki frame poziva HandleAirplaneUI() za svaki instrument koji ima pripadajući UI prikaz.
Ova skripta stvara listu svih instrumenata i za svaki poziva metodu koju oni moraju ispuniti.

```
void Update () {
            if(instruments.Count > 0){
                foreach(IAirplaneUI instrument in instruments){
                    instrument.HandleAirplaneUI();}}}
```

Svaki instrument ima skriptu koja regulira UI elemente određene za njega

Indikator brzine (**Airspeed indicator**) nam govori koliko brzo se zrakoplov giba. Skripta **Airplane_Airspeed.cs** uzima brzinu zrakoplova i prema tome računa rotaciju kazaljke (*pointer.rotation*) na UI objektu.
Kazaljka je posebna slika koja se nalazi iznad slike indikatora brzine i prema tome se može zasebno rotirati. Moramo paziti da limitiramo rotaciju (*maxIndicatedKnots*) i pretvorimo brzinu u stupnjeve (*wantedRotation = 360f * normalizedKnots*).

```
public void HandleAirplaneUI() {
            if(characteristics && pointer){
                float currentKnots = characteristics.MPH * mphToKnts;
                Debug.Log(currentKnots);
                float normalizedKnots = Mathf.InverseLerp(0f,
maxIndicatedKnots, currentKnots);
                float wantedRotation = 360f * normalizedKnots;
                pointer.rotation = Quaternion.Euler(0f, 0f,
-wantedRotation);}}
```

normalizedKnots = *Mathf.InverseLerp* daje postotak vrijednosti *currentKnots* između 0 i *maxIndicatedKnots*

Indikator visine (**Altimeter**) prikazuje visinu na kojoj se zrakoplov trenutno nalazi. Skripta **Airplane_Altimeter.cs** kontrolira rotacijom kazaljki za prikaz stotina (*currentHundreds*) i tisuća (*currentThousands*) metara na instrumentu ovisno o visini na kojoj se nalazimo (*currentAlt*). *Quaternion.Euler* vraća ordeđenu rotaciju (*thousandsRotation, hundredsRotation*) po z osi. (prvi vraća rotaciju za x os, drugi za y, a treći za z os)

```
public void HandleAirplaneUI(){
            if(airplane){
                float currentAlt = airplane.CurrentMSL;
                float currentThousands = currentAlt / 1000f;
                currentThousands = Mathf.Clamp(currentThousands, 0f, 10f);
                float currentHundreds = currentAlt -
(Mathf.Floor(currentThousands) * 1000f);
                currentHundreds = Mathf.Clamp(currentHundreds, 0f, 1000f);
                if(thousandsPointer){
                    float normalizedThousands = Mathf.InverseLerp(0f, 10f,
currentThousands);
                    float thousandsRotation = 360f * normalizedThousands;
                    thousandsPointer.rotation = Quaternion.Euler(0f, 0f,
-thousandsRotation);}
                if(hundredsPointer){
                    float normalizedHundreds = Mathf.InverseLerp(0f, 1000f,
currentHundreds);
                    float hundredsRotation = 360f * normalizedHundreds;
                    hundredsPointer.rotation = Quaternion.Euler(0f, 0f,
-hundredsRotation);}}}
```

Flap Lever nam govori na kojoj poziciji stupnjeva se nalaze zakrilca koja utječu na otpor zraka. Skripta koja upravlja ovim instrumentom je **Airplane_FlapLever.cs** i pomiče UI elemente napravljene od običnih tekstualnih elemenata i pravokutnika koji predstavlja ručicu koja se pomiče prema gore ili dolje ovisno o poziciji zakrilca.
Vector2.Lerp interpolira vrijednosti između originalne pozicije (*handleRect.anchoredPosition*) i željene (*wantedHandlePosition*) korakom Time.deltaTime * *handleSpeed.*

```
public void HandleAirplaneUI(){
            if(input && parentRect && handleRect) {
                float height = parentRect.rect.height;
                Vector2 wantedHandlePosition = new Vector2(0f, -height *
input.NormalizedFlaps);
                handleRect.anchoredPosition =
Vector2.Lerp(handleRect.anchoredPosition, wantedHandlePosition,
Time.deltaTime * handleSpeed); }}
```

Throttle Lever instrument nam pokazuje na kolikoj razini potiska se nalazi zrakoplov tj. motor. UI elementi se sastoje kao i kod Flap Lever-a od teksta i pravokutnika koji predstavlja ručicu koja se pomiće gore dolje od 0% do 100% potiska. Kod funkcionira isto kao i za pogon.

Slike UI elemata su uvezene unutar Image objekata koji se nalaze pod canvas objektom za pojedini instrument koji sadrži skriptu za kontrolu istoh i sadržan je u listi koju generira glavna skripta i poziva skripte svaki frame
Slika 19 prikazuje sve 2D UI elemente (Altimeter, Airspeed, RPM, Throttle, Flaps)

```
public void HandleAirplaneUI(){
            if(input && parentRect && handleRect){
                float height = parentRect.rect.height;
                Vector2 wantedHandlePosition = new Vector2(0f, height *
input.StickyThrottle);
                handleRect.anchoredPosition =
Vector2.Lerp(handleRect.anchoredPosition, wantedHandlePosition,
Time.deltaTime * handleSpeed);}}
```

Slika 19. UI elementi

6.13 Razvoj gameplay elemenata

Gameplay se sastoji od više staza koje igrač mora proći u što kraćem vremenu.
Na kraju svake staze igrač mora sletjeti sa zrakoplovom na pistu i tada se aktivira sljedeća.

Staze sadrže krugove (checkpoints) označene sa zelenom strelicom kroz koje igrač mora proletjeti sa pravilne strane i što manje udaljen od središta kruga. Prilikom prolaska aktivira se zvuk koji indicira da je igrač pravilno proletio kroz checkpoint.

Gate.cs je skripta koja provjerava je li checkpoint aktivan i je li igrač (objekt označen sa "Player") prošao kroz njih, ukoliko je onda je deaktiviran. Isto tako provjerava smjer vrata (*CheckDirection*) kako bi se utvrdilo dali je igrač prošao u pravom smjeru te računa udaljenost igrača od centra (*Vector3.Distance*) koja se koristi za praćenje bodova.

```
private void OnTriggerEnter(Collider other){
            if(other.tag == "Player" && !isCleared)
            {
                float dist = Vector3.Distance(other.transform.position,
transform.position);
                float distPercentage = dist / transform.localScale.x;
                CheckDirection(other.transform.forward, distPercentage);}}
```

Track.cs traži sva vrata (metoda *FindGates()*), inicijalizira ih (metoda *InitializeGates()*) i stvara listu istih. Služi primarno za provjeru prolaska cijele staze tj. svih vrata. Prilikom prolaska kroz jedna vrata, prebacuje trenutna vrata na sljedeća ovisno o nazivu i zapisuje score te provjerava dali je staza završena.

Osim toga gleda trenutno vrijeme staze, bodove i koliko od ukupnih vrata smo prošli kako bi to mogli prikazati teksutlano na ekranu te sprema sve podatke pri završetku staze. (metoda *UpdateStats()*)

```
void Start(){
            FindGates();
            InitializeGates();
            currentGateID = 0;
            StartTrack();}
void Update(){
            if(!isComplete){
                UpdateStats();}}
```

Track_Manager.cs je skripta na vrhu hijerarhije koja ima funkcionalnost pokretanja staze (*StartTrack()*) i aktiviranje objekata aktivne staze i deaktiviranje onih koji to nisu.
Pronalazi sve staze (*FindTracks()*) i inicijalizira iste (*InitializeTracks()*), ažurira UI (*UpdateUI()*) za prikazivanje podataka o stazi, provjerava dali smo sletjeli i završili stazu kako bi aktivirali sljedeću.

```
private void Start() {
            FindTracks();
            InitializeTracks();
            StartTrack(0);}
        private void Update(){
            if(currentTrack){
                UpdateUI();} }
```

6.14 Pause Menu i Main Menu

Za Main menu je korišten besplatan Asset sa Unity Store-a pod nazivom **SpeedTutor Full Menu System [22]** koji pruža osnovne elemente glavnog menija poput pokretanja igre, osnovnih postavki i izlaska iz igre. Elementi su prilagođeni kako bi pristajali zamišljenoj viziji glavnog menija.

Meni za pauzu je napravljen pomoću **PauseMenu_Controller.cs** skripte koja pritiskom na određeni gumb (Esc) postavlja Time.timeScale = 0f; kako bi se zaustavila igra i ujedno aktivira Canvas koji prikazuje meni sa odabirima za nastavak igre ili izlazak na Main Menu.

Prilikom odabira nastavka igre postavlja Time.timeScale = 1f; kako bi se nastavila igra i deaktivira objekt Canvas kako bi se sakrio meni. Gumb Main Menu učitava scenu za Main Menu koja je posebno odvojena od glavne scene igre.

Obje scene moraju biti odabrane pod Build Settings i pravilno poredane.
Za kreiranje gumbova za odabir se koristi TextMesh_Pro.

Slika 20 prikazuje meni pauze koji se prikazuje pritiskom tipke “Esc” i omogućuje izlazak na Main Menu ili nastavak igre. Kako bi zaustavili vrijeme igre *Time.timeScale = 0f*.

Slika 21 prikazuje glavni meni igre koji je posebna scena za sebe. Imamo opcije pokretanja nove igre, promijene nekih postavki u opcijama ili izlaska iz same igre.

Slika 20. Pause Menu

Slika 21. Main Menu

```
void Start(){
            pauseMenu.SetActive(false);
        }
        void Update() {
            if(Input.GetKeyDown(KeyCode.Escape)){
                if(isPaused){
                    ResumeGame();
                }
                else{
                    PauseGame();
                }}}
        public void PauseGame() {
            pauseMenu.SetActive(true);
            Time.timeScale = 0f;
            isPaused = true;}
        public void ResumeGame(){
            pauseMenu.SetActive(false);
            Time.timeScale = 1f;
            isPaused = false;}
        public void MainMenu(){
            Time.timeScale = 1f;
            SceneManager.LoadScene("MainMenu");}
```

Kod prikazuje **PauseMenu_Controller.cs** skriptu, u Update() metodi (metoda koja se poziva za svaki renderan okvir igre) se provijerava dali je tipka “Escape” pritisnuta na tipkovnici (*Input.GetKeyDown*).
Ako je provjerava se dali je igra već pauzirana ili nije, ako je, pritiskom na “Resume” poziva se *ResumeGame()* metoda koja će deaktivirati pauseMenu i postaviti Time.timeScale = 1f kako bi se pokrenulo vrijeme igre te ujedno postaviti isPaused = false kako bi znali da igra nije više pauzirana.

Ako pritisnemo gumb za povratak u glavni meni, učitava se scena "MainMenu" gdje se nalazi glavni meni.

Ako igra nije pauzirana i skripta vidi da je pritisnuta tipka “Escape”, postavlja se *Time.timeScale = 0f;* kako bi se zaustavilo vrijeme igre i aktivira se canvas na kojem je meni za pauzu.
isPaused je postavljen na true kako bi znali da je igra pauzirana pozivom svake metode *Update()*.

7. Zaključak

Razvoj video igara nije jednostavan proces. To je proces koji uključuje mnogo podprocesa za koje u današnje vrijeme postoje posebni stručnjaci. Za razvoj AAA igre potrebno je stotine stručnjaka koji imaju desetke godina iskustva. Potrebno je više godina za razvoj jedne igre uz najmodernije alate današnjice dok je u prošlosti to bio nezamislivo kompliciran pothvat.

Video igre su ponajprije umjetnička djela čiji je glavni cilj potaknuti osjećaje u onome tko uživa u njima poput filmova ili knjiga. Bio taj osjećaj sreća, zabava, strah ili tuga, poticanje na kreativnost ili dublje razmišljanje o životu. Kreativnosti nema granica, granica je samo naše znanje i poznavanje alata te njihova jednostavnost korištenja.

Unity Engine je jedan od najpopularnijih alata za razvoj video igara, njegova jednostavnost korištenja i community, koji je uvijek tu da pomogne drugima, omogućuju jednostavnost izražavanja umjetnika koji manje vremena ulaže u proučavanje alata, a više se može fokusirati na svoj rad.

No to ne znači da neće morati zasukati rukave, daleko od toga.

Razvoj simulacijske video igre zahtjevan je posao koji zahtjeva duboko poznavanje sustava kojeg želimo simulirati i dobro baratanje sa fizikom i matematikom.

Ova igra, kao i sve ostale, imaju veliki potencijal i uvijek se može napraviti još bolje i ambicioznije. Ovo je samo jednostavna demonstracija pojednostavljene simulacije leta zrakoplovom uz neke osnovne gameplay elemente no nadam se da igraču pruža uzadovoljstvo igranja kao što je autoru pružalo zadovoljstvo u procesu izgradnje.

Nadam se da će biti vremena za nastavak rada na ovome projektu i prilika za implementiranje novih funkcionalnosti te ambicioznijih i zanimljivih elemenata.

Rujan, 2020.

8. Literatura

[1] Povijest video igara “https://en.wikipedia.org/wiki/Video_game_development#History” , Pristupljeno 2.9.2020
[2] OXO “https://en.wikipedia.org/wiki/OXO” , Pristupljeno 2.9.2020
[3] Žanrovi video igara “https://en.wikipedia.org/wiki/Video_game_genre”, Pristupljeno 2.9.2020
[4] Lista žanrova “https://en.wikipedia.org/wiki/List_of_video_game_genres”, Pristupljeno 2.9.2020
[5] Simulacijske video igre “https://en.wikipedia.org/wiki/Simulation_video_game”, Pristupljeno 2.9.2020
[6] Simulacijske igre vozila “https://en.wikipedia.org/wiki/Vehicle_simulation_game”, Pristupljeno 2.9.2020
[7] Microsoft Flighjt Simulator “https://en.wikipedia.org/wiki/Microsoft_Flight_Simulator”, Pristupljeno 5.9.2020
[8] SubLOGIC “https://en.wikipedia.org/wiki/Sublogic”, Pristupljeno 5.9.2020
[9] Povijest Microsoft FS “https://en.wikipedia.org/wiki/History_of_Microsoft_Flight_Simulator”, Pristupljeno 6.9.2020
[10] Flight Simulator X “https://en.wikipedia.org/wiki/Microsoft_Flight_Simulator_X”, Pristupljeno 6.9.2020
[11] MFS 2020 “https://en.wikipedia.org/wiki/Microsoft_Flight_Simulator_(2020_video_game)”, Pristupljeno 6.9.2020
12] Azure Cloud “https://en.wikipedia.org/wiki/Microsoft_Azure”, Pristupljeno 6.9.2020
[13] Povijest alata za razvoj video igara “https://en.wikipedia.org/wiki/Game_development_tool”, Pristupljeno 6.9.2020
[14] Unity Engine “https://en.wikipedia.org/wiki/Unity_(game_engine)”, Pristupljeno 6.9.2020
[15] Adobe Photoshop “https://en.wikipedia.org/wiki/Adobe_Photoshop”, Pristupljeno 6.9.2020
[16] Aerodinamika leta “https://www.nasa.gov/audience/forstudents/5-8/features/nasa-knows/what-is-aerodynamics-58.html”, Pristupljeno 6.9.2020
[17]Četiri sile leta “https://www.nasa.gov/audience/foreducators/k-4/features/F_Four_Forces_of_Flight.html#:~:text=The%20four%20forces%20are%20lift,made%20the%20Frisbee%20slow%20down.”, Pristupljeno 7.9.2020
[18] Bernullijev princip “https://en.wikipedia.org/wiki/Bernoulli%27s_principle#:~:text=In%20fluid%20dynamics%2C%20Bernoulli's%20principle,his%20book%20Hydrodynamica%20in%201738.”, Pristupljeno 7.9.2020
[19] Napadni kut (Angle of attack) “https://en.wikipedia.org/wiki/Angle_of_attack”, Pristupljeno 7.9.2020
[20] Banking “https://en.wikipedia.org/wiki/Banked_turn”, Pristupljeno 7.9.2020
[21] Bing Maps “https://en.wikipedia.org/wiki/Bing_Maps”, Pristupljeno 6.9.2020
[22] SpeedTutor Full Menu System “https://assetstore.unity.com/packages/tools/gui/full-menu-system-free-158919”, Pristupljeno 7.9.2020

Rujan, 2020.

9. Slike

10. Prilozi

Uz ovaj projekt je priložena igra AIR: Airplane Immersive Racing sa svim svojim dijelovima i komponentama.

Rujan, 2020.