Izrada 2D igara u Unityju

Zec, Marko

VELEUČILIŠTE SUVREMENIH INFORMACIJSKIH TEHNOLOGIJA

STRUČNI PRIJEDIPLOMSKI STUDIJ INFORMACIJSKIH TEHNOLOGIJA

Marko Zec

ZAVRŠNI RAD

IZRADA 2D IGARA U UNITYJU

Zagreb, listopada 2024.

Veleučilište suvremenih informacijskih tehnologija
10000 Zagreb, Ulica Vjekoslava Klaića 7

Studij:	Stručni prijediplomski studij informacijskih tehnologija smjer programiranje
Student:	**Marko Zec**
Matični broj:	2017022

Zadatak završnog rada

Predmet:	Programiranje u C#
Naslov:	**Izrada 2D igara u Unityju**
Zadatak:	Opisati izradu igara korištenjem Unity razvojne platforme. U okviru praktičnog rada napraviti oglednu 2D igru.
Mentor:	mr. sc. Julijan Šribar, v. pred.
Zadatak uručen kandidatu:	26.10.2023.
Rok za predaju rada:	3.10.2024.
Rad predan:	_____

Povjerenstvo:

Jurica Đurić, v. pred.	član predsjednik	_____
mr. sc. Julijan Šribar, v. pred.	mentor	_____
Mariza Maini, pred.	član	_____

SADRŽAJ

POPIS SLIKA

POPIS KODOVA

1. UVOD

U današnjem digitalnom dobu, u svijetu digitalne industrije zabave, koja se neprestano širi i razvija, videoigre su postale snažan medij interaktivne zabave i neizostavan dio života mnogih mladih i odraslih diljem svijeta. Bez obzira na dob, spol ili kulturnu pripadnost, milijuni ljudi svakodnevno se posvećuje raznim videoigrama na računalu, konzolama, mobilnim uređajima ili čak u virtualnoj stvarnosti. Moć videoigara da privlače publiku svih dobnih skupina poprima sve veće razmjere te pretvara igranje videoigara u globalnu pojavu. Tijekom godina, industrija videoigara je doživjela veliku transformaciju, od rudimentarnih igara iz prošlosti, do impresivnih i vizualno zapanjujućih remek-djela današnjice. Ta transformacija je dokaz nebrojenih inovacija i tehnološkog napretka koji su oblikovali svijet videoigara i gurnuli ga u svakidašnjicu današnjeg društva.

Ovaj završni rad se bavi razvojem videoigara, prvenstveno 2D videoigara korištenjem pokretača (engl. *game engine*) Unity i programskog jezika C#. Unity je jedan od vodećih pokretača videoigara dostupnih na tržištu. Otkako je predstavljena prva verzija prije skoro dva desetljeća, odigrao je ključnu ulogu u demokratizaciji razvoja videoigara, omogućujući početnicima, kao i iskusnim programerima, da efikasno stvaraju zanimljive i vizualno privlačne videoigre.

Poglavlja koja slijede u ovom radu pružit će uvid u povijest videoigara, prateći njihove korijene od ranih početaka do pojave visokobudžetnih naslova koji su preoblikovali digitalnu industriju zabave. Naglasit će se dinamika i vitalnost industrije videoigara, koja se oblikuje neprestanim inovacijama i tehnološkim napretkom sklopovlja na kojemu se izvode i alata kojima se one razvijaju. Nadalje, rad će se osvrnuti na područje pokretača videoigara, istražujući njihov značaj u pojednostavljivanju i ubrzavanju procesa razvoja, te pružiti objašnjenje osnovnih načela prema kojima oni funkcioniraju. Unity, pokretač koji je korišten u izradi praktičnog dijela rada, dobit će posebnu pozornost prilikom analize značajki, mogućnosti i doprinosa da razvoj videoigara postane dostupan široj publici. Bit će istraženi temeljni koncepti i alati koje pokretač Unity pruža i koji ga čine izborom za brojne programere diljem svijeta. Zatim slijedi detaljno ispitivanje funkcionalnosti pokretača Unity kroz razvoj vlastite 2D igre „Space Shooter”. U tom poglavlju proći će se kroz praktičnu implementaciju novostečenog znanja, pokazujući sinergiju Unityja i programskog jezika C# u stvaranju 2D videoigre. Proces razvoja bit će prikazan korak po korak, pružajući praktično razumijevanje onoga što je potrebno za stvaranje funkcionalne videoigre.

Do kraja ovog završnog rada čitatelji će steći sveobuhvatno razumijevanje povijesnog konteksta videoigara, značaja pokretača u razvoju videoigara i praktične primjene pokretača Unity i programskog jezika C# u razvoju 2D videoigara.

2. VIDEOIGRA

Videoigra je interaktivni računalni program namijenjen za zabavu (Slika 1). Popularnost videoigara u stalnom je porastu i danas su u društvu sveprisutne. Dostupne su na raznim platformama, od arkadnih aparata i igraćih konzola do osobnih računala i mobilnih uređaja. Globalna industrija videoigara danas (2024. godina) prelazi vrijednost od 400 milijardi dolara[1]. Ovo poglavlje obrađuje povijest videoigara i tehnologije koje se koriste za njihov razvoj i izvođenje.

Slika 1. Videoigra (Wikimedia Commons)

2.1. Povijest videoigara

Porijeklo videoigara seže do 1950-ih i 1960-ih godina, kada su računalni znanstvenici i inženjeri počeli eksperimentirati s interaktivnim oblikom elektroničke zabave. Rudimentarne, ali revolucionarne igre tog razdoblja, poput *Tennis for Two* i *Spacewar!*, pojavile su se na središnjim (engl. *mainframe*) sveučilišnim računalima. Kako se sklopovlje razvijalo, igre su postajale kompleksnije. 1970-ih godina, izlaskom igara kao što su *Pong* i *Space Invaders*, izniknula je industrija arkadnih igara. One su se izvodile na aparatima na kovanice koji su predstavili videoigre masama i postali kulturni fenomen (Slika 2).

[1] https://www.statista.com/statistics/1344668/revenue-video-game-worldwide/

Slika 2. Atari Pong aparat (Wikimedia Commons)

Pojava kućnih igraćih konzola, kao što su Magnavox Odyssey i Atari 2600, u kasnim 1970-ima označila je veliki pomak u industriji, donoseći videoigre u domove. 1980-e godine smatraju se zlatnim razdobljem arkadnih igara. Naslovi kao što su *Pacman*, *Donkey Kong*, *Galaga* i *Asteroids* postali su veliki hitovi i uspostavili igranje igara (engl. *gaming*) kao popularni oblik zabave. Unatoč uspjehu arkadnih igara, industrija videoigara doživjela je krah 1983. godine zbog prezasićenosti tržišta nekvalitetnim igrama. Pojava konzole NES (engl. *Nintendo Entertainment System*) 1985. godine revitalizirala je industriju. Igre poput *Super Mario Bros.* i *The Legend of Zelda* postavile su nove standarde. Istovremeno, rasla je popularnost igranja na osobnim računalima uz naslove kao što su *Doom*, *Quake* i *SimCity*.

S napretkom tehnologije, računala su postala dovoljno jaka da renderiraju 3D grafiku, što je revolucioniralo videoigre tijekom 1990-ih i ranih 2000-ih godina. Sony Playstation i kasnije Playstation 2 te konzole Sega Saturn i Dreamcast, pomaknule su granice mogućeg u pogledu 3D grafike i igrivosti (engl. *gameplay*).

Industrija videoigara zadržava brz tempo razvitka u 21. stoljeću. *Gaming* prerasta u globalnu pojavu, šireći se na platforme kao što su pametni telefoni i internet, putem kojeg se popularizira natjecanje u videoigrama (eSportovi). Veliki proizvođači poput Sonyja, Microsofta i Nintenda, nastavljaju razvijati tehnološki napredne konzole i igre. Naglasak je na fotorealističnoj grafici i sve većoj rezoluciji, kao i tehnološkim inovacijama poput virtualne stvarnosti VR (engl.

virtual reality), proširene stvarnosti AR (engl. *augmented reality*) i igranju u oblaku (engl. *cloud gaming*).

2.2. Pokretač videoigre

Pokretač videoigre (engl. *game engine*) je specijaliziran softverski sustav za razvoj i izvođenje videoigara, odnosno tehnologija putem koje se razvija i prikazuje interaktivna računalna grafika. Pokretači pružaju skup alata i sustava koje programeri koriste za učinkovito dizajniranje, izradu i upravljanje videoigrama. Mogu se klasificirati u dvije glavne vrste:

- 2D pokretači, dizajnirani za stvaranje 2D igara. Nude značajke kao što su upravljanje bitmapama (engl. *sprite*), pločicama mape (engl. *tilemap*) i 2D simulacije fizikalnih zakona. Primjeri 2D pokretača su Unity (na kojemu se mogu razvijati 2D i 3D igre), Godot Engine[2] i Cocos2d[3].
- 3D pokretači, namijenjeni za razvoj 3D igara, nudeći alate za prikaz 3D modela, upravljanje složenim scenama i rukovanje realističnom fizikom. Značajni predstavnici su Unreal Engine[4], Unity[5] i CryEngine[6].

Glavne komponente pokretača su:

- Grafički renderer, sustav za prikaz 2D bitmapa ili 3D modela u realnom vremenu, koristeći tehnike kao što su rasterizacija i praćenje zraka svjetlosti (engl. *ray tracing*).
- Sustav za fiziku, namijenjen za simulaciju zakona fizike, poput detekcije i obrade sudara između elemenata videoigre, gravitacije i drugih fizikalnih načela.
- Sustav za animaciju, namijenjen za animaciju 2D ili 3D objekata tehnikama kao što je *keyframing*.
- Sustav za zvuk namijenjen za reprodukciju zvuka, poput zvučnih efekata i glazbe.
- Umjetna inteligencija, sustav za implementaciju ponašanja računalno kontroliranih objekata videoigre.
- Sustav za kontrole koji obrađuje unos s ulaznih uređaja.
- Sustav za umrežavanje koji omogućava internetsku povezanost između igrača i poslužitelja (engl. *server*).

[2] https://godotengine.org/
[3] https://www.cocos.com/en
[4] https://www.unrealengine.com/en-US
[5] https://unity.com/
[6] https://www.cryengine.com/

Pokretači videoigara značajno pojednostavljuju razvojni proces jer nema potrebe za izgradnjom svakog aspekta igre od temelja. To programerima ubrzava izradu igara i olakšava posao jer se, umjesto na tehničke detalje niske razine, mogu fokusirati na stvaranje same videoigre. Pored učinkovitosti u razvoju, moderni pokretači pružaju visok stupanj skalabilnosti. Mogu se koristiti kako za manje, tako i za velike visokobudžetne projekte. Moderni pokretači također pružaju dobru višeplatformnu kompatibilnost. Na taj način programeri mogu stvarati igre za razne platforme kao što su PC, igraće konzole i mobilni uređaji te tako dosegnuti širu publiku.

3. UNITY

Unity je popularni višeplatformni pokretač za videoigre kompanije Unity Technologies. Može se opisati kao integrirano razvojno okruženje IDE (engl. *Integrated Development Environment*) za razvoj interaktivnih 2D i 3D igara. Objavljen je 2005. godine, s ciljem da bude pokretač s profesionalnim alatima pristupačan za amaterske programere i da na taj način približi razvoj videoigara široj publici. Danas je Unity u širokoj primjeni zahvaljujući pristupačnosti i mnoštvu funkcionalnosti. Ovo poglavlje bavi se detaljnim pregledom pokretača videoigara Unity.

3.1. Povijest razvoja

Prva inačica Unity 1.0 izdana je u srpnju 2005. godine za Mac OS X operacijski sustav i podržavala je isključivo MacOS kao ciljnu platformu projekta. Verzijom 1.1 uvedena je podrška za izvoz projekta na Microsoft Windows i internetskim preglednicima. Ta verzija je uvela hardverski ubrzanu 3D grafiku i podršku za eksterne dodatke napisane u jezicima C/C++. Verzijom 2.0, objavljenom 2007. godine, dodana je podrška za Microsoft DirectX aplikacijsko programsko sučelje (engl. *Application Programming Interface*, API) koje ubrzava izvođenje projekta na Windowsima, a verzija 2.5 objavljena 2009. omogućila je i razvoj na Windowsima[7]. Verzija Unity 3.0 izlazi 2010. godine i dodaje unaprijeđeno osvjetljenje pri renderiranju grafike, kao i druga poboljšanja, među kojima su automatsko UV mapiranje tekstura (proces 3D modeliranja) i tehniku odgođenog renderiranja. 2012. godine izlazi verzija Unity 4.0 koja dodaje podršku za DirectX 11 API i Adobe Flash, kao i novi alat za izradu animacija Mecanim. Godinu kasnije, izlazi verzija 4.3 koja donosi izvornu podršku za razvoj 2D projekata. Unity 5.0 stigao je 2015. godine i donio je brojna poboljšanja u grafičkim i audio mogućnostima, kao što je sustav globalne iluminacije u realnom vremenu, Nvidia PhysX 3.3 sustav za fiziku i unaprijeđeni audio sustav. Verzija 5.6 donijela je nove vizualne efekte i Vulkan grafički API.

Od 2017. godine Unity Technologies mijenja način imenovanja verzija tako da je sljedeća verzija Unity 2017 koja donosi poboljšanje u grafičkom sustavu i nove alate kao što je pametni sustav kamere Cinemachine i sustav Timeline koji omogućuje dodavanje animacija u igru korištenjem miša. Unity 2018 donosi sustav Scriptable Rendering Pipeline za modifikaciju grafike i alat za strojno učenje Imitation Learning, koji omogućuje igri da uči preko navika igrača. Unity 2020 donosi značajku MARS (engl. *Mixed and Augmented Reality Studio*) sa specijaliziranim alatima za poboljšani razvoj aplikacija proširene stvarnosti.

[7] Haas, John K. "A History of the Unity Game Engine." (2014).

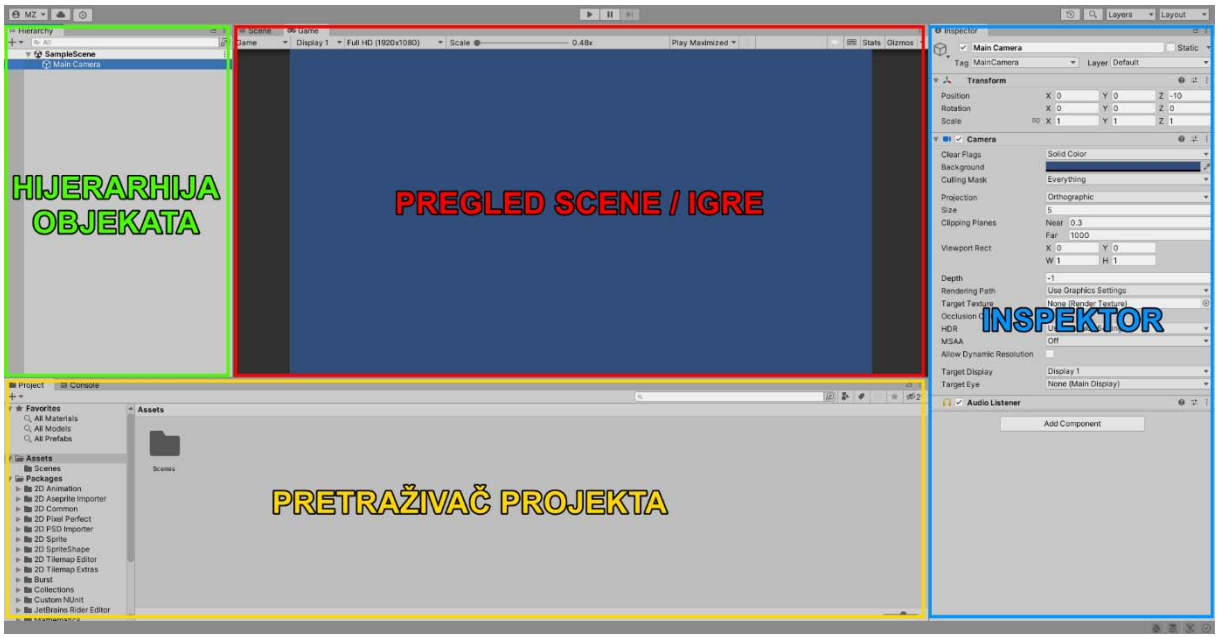
3.2. Značajke

Unity je dostupan u više verzija, ovisno o vrsti licence. Unity Personal je besplatna licenca za hobiste, studente i neovisne (engl. *indie*) programere i tvrtke. Uključuje osnovne funkcionalnosti i sadrži ograničenje na projekte s godišnjim prihodom manjim od 200.000 dolara[8]. Osim besplatne licence, Unity Technologies nudi Unity Plus, Pro i Enterprise verzije s proširenim značajkama i servisima, poput servisa baziranih u oblaku i napredne analitike namijenjene profesionalnim razvojnim studijima.

Za početak korištenja Unityja neophodno je registrirati se i s internetske stranice Unityja preuzeti aplikaciju Unity Hub, koja služi za upravljanje projektima i verzijama Unity Editora.

3.2.1. Unity Editor

Unity Editor je radno grafičko korisničko okruženje koje se koristi za stvaranje sadržaja za 2D, 3D i AR/VR iskustva te se sastoji od nekoliko integriranih prozora. Najvažniji i najkorisniji prozori su: Pretraživač projekta (*engl. Project window*), Inspektor (*engl. Inspector*), Pregled igre (*engl. Game view*), Pregled scene (*engl. Scene view*) i Hijerarhija objekata (*engl. Hierarchy window*) (Slika 3).



Slika 3. Unity Editor

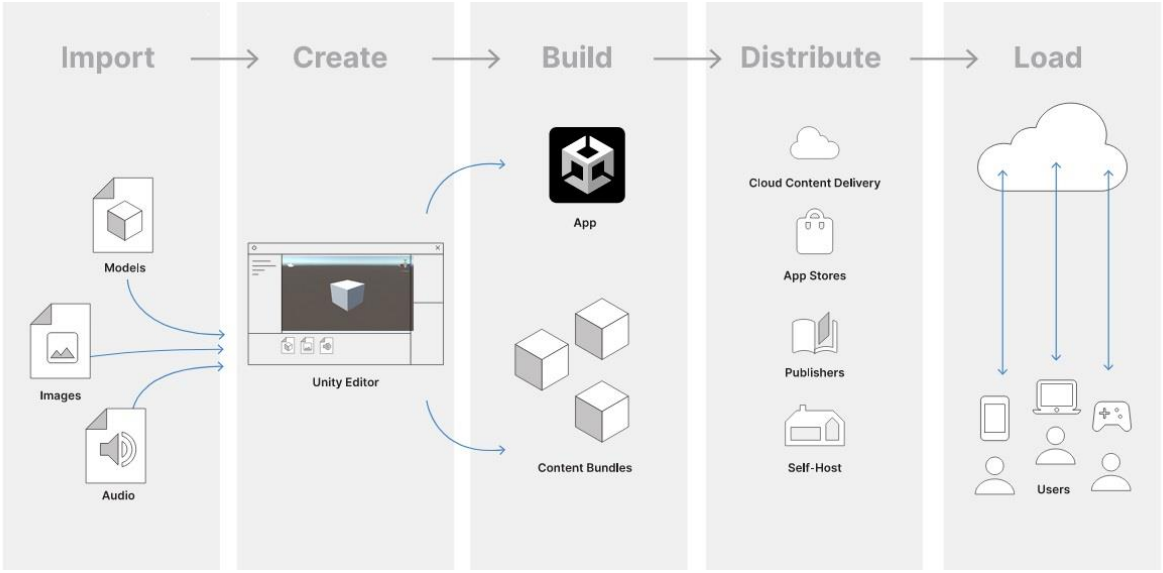
- Pretraživač projekta je prozor koji sadrži uvezene datoteke za razvoj projekta.

[8] https://unity.com/products/unity-personal

- Inspektor je prozor koji pruža uvid i omogućava izmjenu komponenata koje čine objekte igre.
- Pregled scene je prozor u kojem se odvija grafički dio izgradnje aplikacije.
- Pregled igre je prozor u kojem se prikazuje trenutačno stanje igre i omogućuje programeru testiranje unutar Editora.
- Hijerarhija objekata je prozor u kojem je, prema relaciji roditelj-dijete, poredana lista objekata igre unutar scene.

3.2.2. Koncept i tok razvoja

Ključna značajka Unityja je jednostavan tok izgradnje projekta (Slika 4).



Slika 4. Tok razvoja (Unity Technologies)

Prvi korak je uvoz datoteka koje se koriste za izradu aplikacije, poput slika, audio i video datoteka, 3D modela. Nakon što se datoteke uvezu, dostupne su unutar Editora i nazivaju se imovinom (engl. *asset*). Datoteke imovine se koriste kao elementi objekata unutar scene koji se nazivaju objekti igre (engl. *GameObject*). Proces izgradnje (engl. *build*) uključuje izvoz gotovog projekta u binarnu datoteku, koja se može pokrenuti i distribuirati na odabranoj platformi. Aplikacija se sastoji od jedne ili više scena. Svaka scena, bilo da se radi o sučelju ili kompleksnoj razini igre, sadrži objekte igre. Objekti igre se sastoje od komponenata. Komponente jednog objekta igre mogu biti ugrađene komponente ili individualne skripte. Primjer takve hijerarhije bio bi:

- *Scena* (Razina 1) koja sadrži podatke o svim objektima igre unutar scene
- *Objekt igre* (Igrač) je objekt razine 1

- *Komponenta* (Kreći se) je skripta prikvačena na objekt igre koja definira funkcionalnost.

Objekt igre u Unityju je osnovni građevni blok svake scene i služi kao spremnik za različite komponente koje definiraju njegove karakteristike, ponašanje i funkcionalnost. Prilikom razvoja, objektima igre mogu se dodavati razne komponente koje variraju ovisno o vrsti objekta i njegovoj predviđenoj ulozi. Neke od najkorisnijih i najčešće korištenih komponenti su:

- Transform je osnovna komponenta koja se automatski dodaje svakom objektu igre. Ona definira položaj, rotaciju i veličinu objekta u svijetu igre.
- Renderer, komponenta za prikaz objekta igre. Dodaje se objektima igre koji moraju biti vidljivi u svijetu igre. Kontrolira prikaz objekta, uključujući mrežu (engl. *mesh*), materijal i druga svojstva renderiranja.
- Collider, komponenta za detekciju sudara. Definira oblik fizičke interakcije objekta igre s drugim objektima kao i ponašanje prilikom sudara.
- Rigidbody, za interakciju objekta igre s Unityjevim podsustavom za fiziku. Dodaje fizička svojstva objektu, kao što su sile, masa i gravitacija.
- Camera, komponenta za prikaz različitih pogleda na svijet igre.
- AudioSource, omogućuje reprodukciju audio zapisa i kontrolu s ugrađenim ekvalizatorom.
- Animator, omogućuje kontrolu animacija i prijelaza između animacija.
- Particle System, komponenta za stvaranje efekata sustava čestica kao što su dim, eksplozija i vatra.
- Script, prilagođene skripte prikvačene na objekt igre koje definiraju njegovu funkcionalnost i ponašanje. Preko ugrađenih klasa i metoda, skripte jednog objekta komuniciraju s drugim objektima i njihovim komponentama.

3.2.3. Skriptiranje u Unityju

Skriptiranje u Unityju je proces stvaranja prilagođenog kôda za definiranje funkcionalnosti, ponašanje i interakcije objekata i sustava igre. Unity koristi C# kao zadani programski jezik, koristeći Mono .NET okruženje.

Jezik C# je statički tipiziran jezik koji nudi snažnu podršku za objektno orijentirano programiranje. Jezik C# omogućuje pristup Unity API-u za kontrolu objekata, fizike, grafike i drugih aspekta Unity pokretača. Skripte se prilažu objektima igre kao komponente. Kroz metode za događaje i povratne pozive (engl. *events and callbacks*), pruža se mogućnost odgovora na različite događaje u igri, kao što su sudari, okidači, klikovi mišem i sl. Prilagođivanjem kôda za rukovanje tim događajima mogu se stvoriti složene interakcije u igri.

Omogućeno je zadavanje redoslijeda izvršavanja skripti, što je važno za upravljanje ovisnostima i interakcijama. Unityjev sustav za upravljanje sredstvima omogućava referenciranje i korištenje sredstava preko skripte. Unity također nudi takozvane skriptabilne objekte (engl. *Scriptable Objects*), sredstva slična skriptama koja pohranjuju podatke i postavke neovisno o objektima igre.

3.2.4. Prednosti i nedostaci Unityja

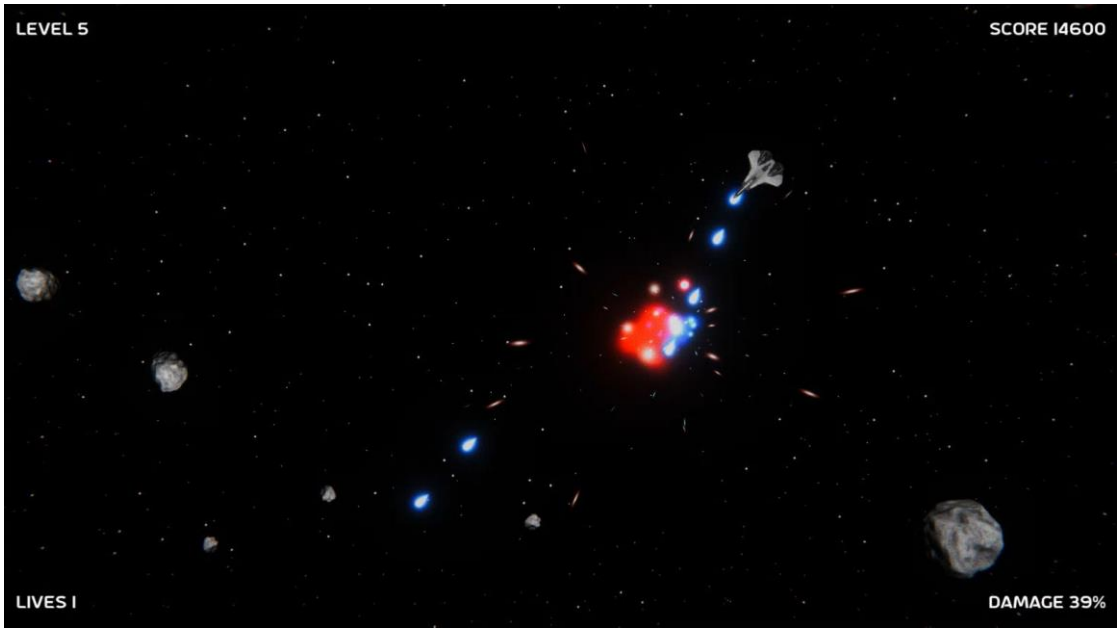
Efikasnost i jednostavnost radnog toka jedna je od glavnih prednosti Unityja. Unity omogućava jednostavan prijenos objekata iz vanjskih aplikacija i njihovo korištenje te automatsko spremanje pri modifikaciji na razini cijelog projekta. Unityjev prozor Inspektor omogućuje prilaganje komponenti i skripti objektima igre i prilagođivanje njihovih svojstava i varijabli izravno u Editoru, što olakšava podešavanje objekata bez dodatnog pisanja kôda. Veliki projekti s puno podataka mogu se dobro organizirati uz pretraživač projekta koji indeksira podatke po korisnički definiranim oznakama. Podržan je veliki broj platformi za izgradnju projekta te je moguće istovremeno razvijati za nekoliko platformi koristeći isti izvorni kôd. Također, programerima je na raspolaganju Asset Store, trgovina gotovih objekata kao što su animacije, grafike, efekti i slično, koji se mogu klikom dodati u projekt i na taj način razvoj dodatno ubrzati. Nadalje, Unity se može primijeniti izvan okvira video igara, primjerice u svrhu razvoja 3D prototipa, podatkovne vizualizacije ili čak za interaktivne umjetnosti. Dakako, najveća prednost Unityja jest činjenica da je pristupačan s relativno blagom krivuljom učenja, čineći ga dobrim izborom za amaterske programere. Istovremeno, moćni alati i mnogo značajki čine ga dobrim izborom za profesionalne programere.

Iako moćan i naširoko korišten pokretač koji je pridonio stvaranju brojnih uspješnih igara, kao i svaka druga softverska platforma, Unity ima nedostatke i mane. Slabe performanse, odnosno vjerojatnost za sporo izvođenje igara, glavna je kritika među programerima. Određene vrste igara, poput onih s visokim grafičkim zahtjevima osobito predstavljaju problem u smislu optimizacije performansi. Iako je Unity dosta napredovao u ovom smislu i napravio značajna poboljšanja tijekom godina, još uvijek ima poteškoća sa složenim scenama i velikim brojem objekata, osobito na slabijem hardveru. Realistično osvjetljenje, sjene i složeni efekti naknadne obrade (engl. *post-processing*) mogu značajno opteretiti hardver, što prisiljava programere da optimiziraju programe za sjenčanje (engl. *shader*), smanjuju broj poziva za renderiranje te pažljivo upravljaju brojem poligona unutar scena kako bi održali dobre performanse. Unity koristi sakupljač smeća za upravljanje memorijom (engl. *garbage collector*), što može uzrokovati probleme u izvedbi. Često sakupljanje smeća može dovesti do pada broja sličica u

sekundi. Loše optimizirane skripte ili skripte s puno nekorištenog ili redundantnog kôda mogu predstavljati usko grlo u izvedbi. Iako je C# skriptiranje u Unityju relativno brzo, neučinkovit kôd negativno utječe na brzinu izvođenja. Bitno je napomenuti da se ovi nedostaci mogu ublažiti iskustvom, pažljivim planiranjem i dobrom optimizacijom.

4. PRAKTIČNI RAD – SPACE SHOOTER

U okviru ovog rada razvijena je ogledna igra Space Shooter (Slika 5). Radi se o klonu klasične dvodimenzionalne arkadne igre *Asteroids*. Cilj igre je uništiti asteroide i leteće tanjure. Igrač upravlja svemirskim brodom, dodavajući potisak i skretanjem ulijevo ili udesno. Pucanjem projektila uništava leteće objekte koji se pojavljuju na ekranu, čime se stječu bodovi. Glavna mehanika igre jest metoda omotavanja (engl. *wraparound*). Naime, kada asteroidi i leteći tanjuri napuste ekran na određenoj strani, automatski se ponovno pojavljuju u polju igre na suprotnoj strani ekrana, zadržavajući pritom smjer kretanja. Space Shooter je koncipiran kao jednostavan uvod u Unity koji za cilj ima opisati općenitu funkcionalnost pokretača kroz implementaciju osnovnih elemenata videoigara, kao što su stvaranje objekata, kretanje, pucanje, sudaranje, skriptiranje i upravljanje zvukom. U ovom poglavlju opisat će se glavne značajke i mehanike igre, objasniti proces stvaranja igre i pružiti detalji o tehničkoj implementaciji glavnih elemenata igre. Radi sažetosti manje važni detalji bit će izostavljeni.

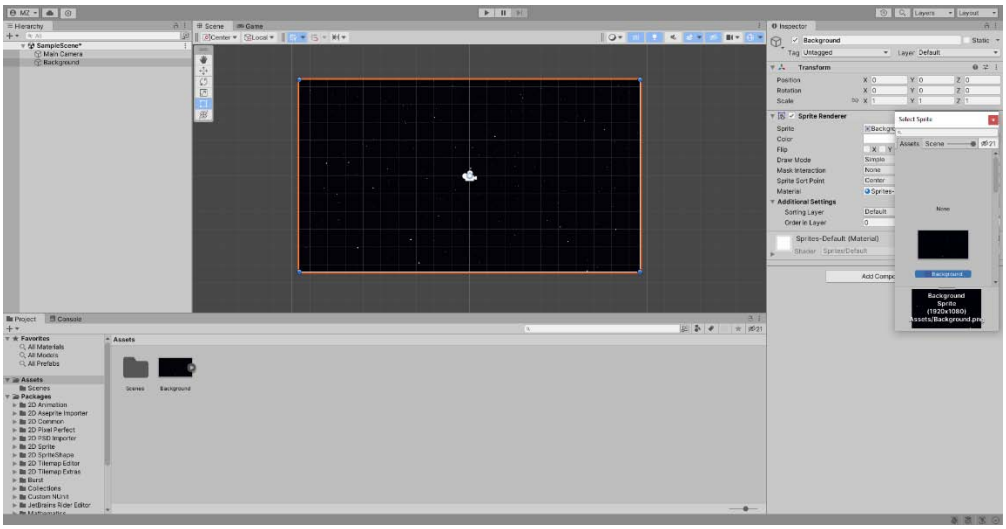


Slika 5. Prikaz Space Shootera

4.1. Postavljanje projekta

Projekt započinje odabirom konfiguracije za 2D aplikaciju u Unity Hubu. Projekt je pripremljen postavljanjem kamere na ortografski režim projekcije i podešavanjem omjera prikaza odnosno rezolucije. Potom slijedi uvoz sredstava, odnosno slikovnih (*sprite*), audio i ostalih datoteka koje će se koristiti u razvoju. Prilikom uvoza slikovnih datoteka koje imaju širinu i visinu jednake višekratnicima broja 4, one se automatski pretvaraju u komprimirani format teksture DXT, što značajno smanjuje veličinu na disku. U Space Shooteru akcija se događa u svemiru

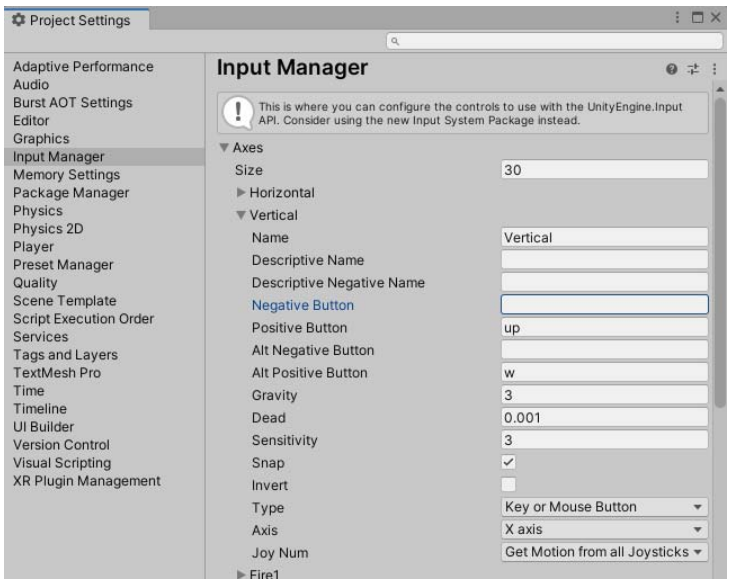
pa je potrebno stvoriti objekt igre Background koji će služiti kao pozadina svemira. Na pozadinu se priloži odgovarajuća tekstura preko komponente za renderiranje Sprite Renderer (Slika 6).



Slika 6. Postavljanje pozadine

4.2. Svemirski brod i kontrole

Rad na svemirskom brodu započinje stvaranjem praznog objekta igre u glavnoj sceni. Pored komponente Transform koja je automatski dio svakog objekta igre, dodaju mu se potrebne komponente, uključujući Rigidbody 2D za povezivanje s fizičkim sustavom, i skripta u kojoj će se definirati razne funkcionalnosti. Za kontrolu broda koriste se već definirane kontrole za horizontalnu i vertikalnu os u upravitelju kontrola. Potrebno je izmijeniti kontrole tako da se ukloni mapiranje za negativnu tipku na vertikalnoj osi, kako bi se onemogućilo kretanje broda unazad (Slika 7).



Slika 7. Upravitelj kontrola

U Unityju svaka skripta automatski nasljeđuje baznu klasu `MonoBehaviour`. Ta klasa pruža skup ugrađenih metoda kao što su `Start`, `Update`, `FixedUpdate`, koje Unity poziva automatski u određenim trenutcima izvođenja igre. Metoda `Start` se poziva samo prilikom instanciranja objekta na koji je skripta prikvačena, a `Update` se poziva za svaku sličicu (engl. *frame*) tokom izvođenja. Zbog toga u metodu `Update` unutar skripte za kontrolu svemirskog broda ShipControl.cs treba smjestiti kôd koji će detektirati unos s kontrole (Kôd 1).

```
void Update()
{
    thrustInput = Input.GetAxis("Vertical");
    turnInput = Input.GetAxis("Horizontal");
}
```

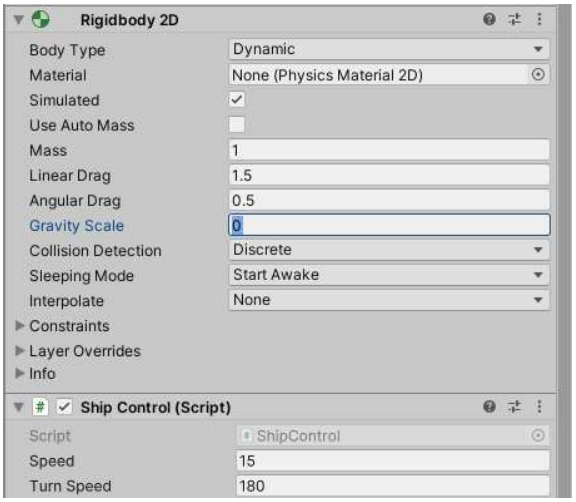
Kôd 1. Definiranje kontrole

Varijable `thrustInput` i `turnInput` pohranjuju rezultat metode `GetAxis`, koja će vratiti vrijednost između -1 i 1 ovisno o vrsti unosa, u ovom slučaju pritiska tipki (strelica gore i dolje) na tipkovnici. Varijable `thrustInput` i `turnInput` su deklarirane kao privatne varijable, što ih čini nedostupnim u Inspektoru kao i vanjskim objektima koji referenciraju skriptu ShipControl.cs. Implementacija kretanja u smjeru prema naprijed odrađena je u metodi `FixedUpdate` koja je sinkronizirana s podsustavom za fiziku, te se zbog toga koristi za primjenu sila na objekt odnosno interakciju s komponentom Rigidbody (Kôd 2).

```
void FixedUpdate()
{
    rb.AddRelativeForce(Vector2.up * thrustInput * speed);
}
```

Kôd 2. Implementacija kretanja

Metoda `AddRelativeForce` primjenjuje silu prema koordinatnom sustavu GameObjecta (svemirskog broda) na Rigidbody preko referentne varijable `rb`. To znači da je smjer vektora `Vector2.up` prema gore u odnosu na objekt, bez obzira na položaj objekta u svijetu igre. Parametar metode je sila, odnosno vektor smjera pomnožen s vrijednošću unosa s kontrole za potisak (`thrustInput`) i koeficijentom brzine (`speed`). Varijabla `speed` je javna varijabla, čija se vrijednost lako može podesiti putem Inspektora i na taj način kontrolirati akceleracija i brzina svemirskog broda. Pored toga, sama komponenta Rigidbody pruža nekoliko modifikatora u Inspektoru, kao što su koeficijenti mase, otpora i gravitacije. Budući da se radnja igre odvija u svemiru, gravitacija se postavi na nulu za sve GameObjecte koji su povezani sa sustavom fizike (Slika 8).



Slika 8. Modifikacija komponenti preko Inspektora

Skretanje ili rotacija svemirskog broda može se implementirati na sličan način primjenom sile, ali se koristi druga metoda koja poboljšava igrivost zbog bržeg odaziva na kontrolu. Rotiranje svemirskog broda postignuto je manipulacijom komponente Transform unutar funkcije `Update` (Kôd 3).

```
transform.Rotate(Vector3.forward, turnInput *
                 turnSpeed * Time.deltaTime);
```

Kôd 3. Rotacija svemirskog broda

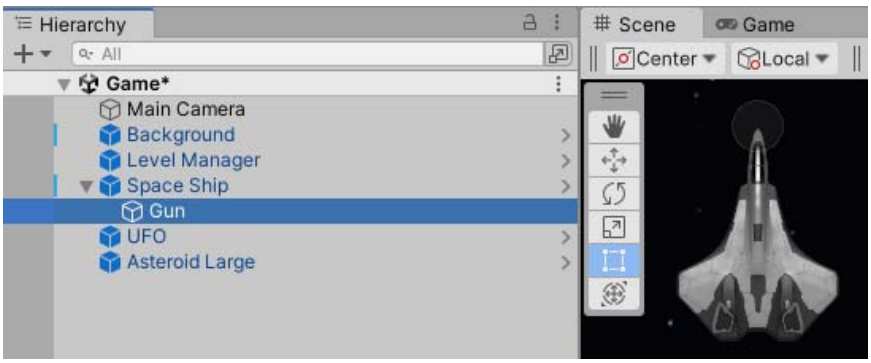
`Vector3.forward` označava *z*-os na koju se vrši rotacija. Umnožak vrijednosti unosa (`turnInput`) i brzine okretanja (`turnSpeed`) predstavlja stupanj rotacije. `Time.deltaTime` je interval između prethodnog i trenutačnog *framea*, i čini da rotacija nije ovisna o broju *frameova* (koji može varirati) već se izvodi konstantnom brzinom.

U Unityju, Prefab je GameObject koji je konfiguriran, spremljen i spreman za ponovnu upotrebu. Sustav Prefabova omogućuje programerima stvaranje predložaka za objekte i višestruko instanciranje kopije tog predloška u sceni. Također, drastično povećava efikasnost razvoja, jer se promjene na jednom Prefabu automatski preslikavaju na sve instance tog Prefaba kroz projekt. Jedan primjer korištenja Prefabova u Space Shooteru odnosi se na mehaniku pucanja (Kôd 4) koje se odvija u skripti ShipControl.cs. Uvjetom `if` se u svakom *frameu* provjerava postoji li unos s kontrole mapirane za pucanje.

```
if (Input.GetButtonDown('Fire'))
{
    GameObject bullet = Instantiate(bulletPrefab, gun.position,
                                    gun.rotation);
    bullet.GetComponent<Rigidbody2D>().AddRelativeForce(Vector2.up *
                                                        bulletSpeed);
    Destroy(bullet, 5.0f);
}
```

Kôd 4. Ispaljivanje projektila

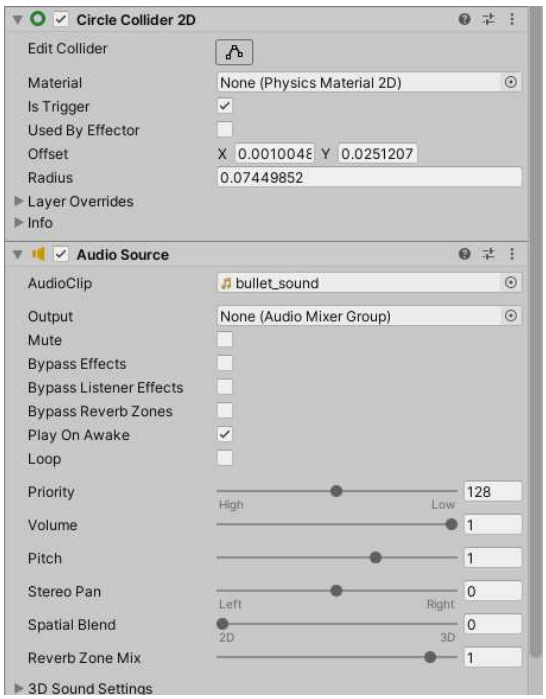
Unity hijerarhija objekata razvrstana je prema relaciji roditelj-dijete, što znači da je moguće definirati kompleksne objekte čiji su elementi drugi objekti odnosno podobjekti. Prefab projektil se instancira na lokaciji GameObjecta `Gun`, čiji je roditelj `Spaceship`, a njegov položaj (komponenta Transform) postavljen je ispred svemirskog broda (Slika 9).



Slika 9. Hijerarhija objekata

Osim preko referenci, komponenti GameObjecta se u skripti može pristupiti i preko metode `GetComponent`. Prefabu projektila dodijeli se oznaka (engl. *Tag*) koja služi za kategoriziranje i olakšava identificiranje u kôdu. Također mu se dodijeli komponenta Circle Collider kojoj se potom uključi opcija isTrigger. Trigger ili okidač je tip Collidera koji nema fizičku interakciju, odnosno neće se sudarati s drugim Colliderima. Njegova je uloga da detektira interakciju, a rezultat se definira u kôdu.

Zvuk u Unityju izvodi se preko komponenti Audio Listener i Audio Source. Na objekt koji reproducira zvuk priloži se Audio Source i zvučni efekt koji će se reproducirati preko komponente Audio Listener. Audio Listener je zadana komponenta glavne kamere u sceni. Tako se na Prefab projektila priloži Audio Source i doda mu se zvučni efekt pucnja, koji će se reproducirati prilikom instanciranja projektila u sceni (Slika 10). Da se ne bi nepotrebno zauzimali resursi, objekt klon Prefaba projektila uništava se 5 sekundi nakon ispaljivanja metodom `Destroy`, što je dovoljno da napusti ekran.



Slika 10. Komponente za koliziju i zvuk

4.3. Omotavanje zaslona

Omotavanje zaslona (engl. *screen wrapping*) je tehnika koja se koristi u videoigrama za umjetno proširivanje svijeta igre manipulacijom kretnje objekata kada napuste granicu zaslona. Umjesto da objekti nestanu kada izađu izvan okvira zaslona ili svijeta igre, omotavanjem se ponovno pojavljuju na suprotnoj strani stvarajući kontinuirano okruženje. Implementacija omotavanja vrši se kao prilagođena skripta pod nazivom ScreenWrap.cs (Kôd 5), koja se priloži kao komponenta svakom GameObjectu koji treba podržavati omotavanje, poput svemirskog broda i asteroida.

```
public class ScreenWrap : MonoBehaviour
{
    [HideInInspector]
    public Vector2 wrappedPosition;

    public bool CheckForWrapping(Vector2 position, float top, float right)
    {
        if (position.y > top)
        {
            wrappedPosition = new Vector2(position.x, -top);
            return true;
        }
        else if (position.y < -top)
        {
            wrappedPosition = new Vector2(position.x, top);
            return true;
        }
        else if (position.x > right)
        {
            wrappedPosition = new Vector2(-right, position.y);
            return true;
        }
        else if (position.x < -right)
        {
            wrappedPosition = new Vector2(right, position.y);
            return true;
        }
        else
            return false;
    }
}
```

Kôd 5. Omotavanje zaslona

Metoda `CheckForWrapping` prima poziciju od GameObjecta koji ju poziva, kao i unaprijed definirane koordinate za koje objekt napušta zaslon. Grananjem `if-else` detektira se i obrne koordinata osi na kojoj dolazi do prelaska granice zaslona, a u varijablu `wrappedPosition` se upiše vektor s korigiranom pozicijom i vraća rezultat logičkog tipa (Boolean) koji predstavlja okidač za omotavanje. Objekt igre se postavi na suprotni dio zaslona preko komponente Transform (Kôd 6).

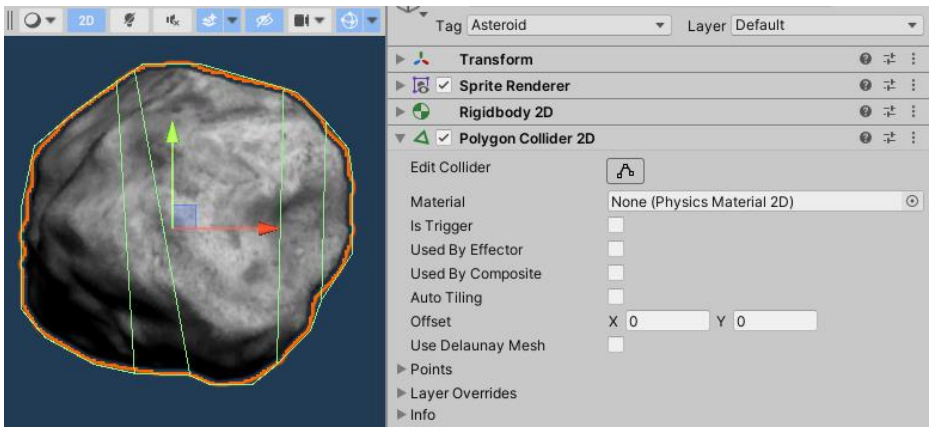
```
if(sw.CheckForWrapping(transform.position, top, right))
    transform.position = sw.wrappedPosition;
```

Kôd 6. Provjera omotavanja zaslona

4.4. Asteroidi i leteći tanjur

U Space Shooteru postoje tri vrste asteroida: veliki, srednji i mali. Kada projektil pogodi veliki asteroid, on se raspolovi na dva srednja. Pogotkom srednjeg, on se raspolovi na dva mala asteroida.

Implementacija počinje stvaranjem Prefabova za asteroide. Prilože se i konfiguriraju potrebne komponente, uključujući Collider za sudaranje s fizičkim GameObjectima. Unity nudi više vrsta Collidera koji se koriste ovisno o obliku objekta. Za asteroide i svemirski brod koristi se Polygon Collider koji automatski prepoznaje kompleksne oblike *spritea* (Slika 11). Ovaj put opcija IsTrigger nije uključena, što znači da će komponenta Rigidbody GameObjecta Asteroid prenositi silu na druge objekte pri sudaru.



Slika 11. Polygon Collider na objektu

U skripti koja kontrolira asteroide (AsteroidControl.cs) koristi se metoda `OnTriggerEnter2D` koja detektira interakciju s Trigger Colliderom projektila, uništava projektil i poziva funkciju koja sadrži logiku za razdvajanje asteroida (Kôd 7 i Kôd 8).

```
void OnTriggerEnter2D(Collider2D other)
{
    if (other.CompareTag('Bullet'))
    {
        Destroy(other.gameObject);
        Split();
    }
}
```

Kôd 7. Interakcija projektila s asteroidom

```
void Split()
{
    Vector2 position = new Vector2(transform.position.x,
                                   transform.position.y);
    switch (size)
    {
        case "Large":
            Instantiate(mediumAsteroid, new(position.x + 0.8f,
                                            position.y + 0.8f),
                                            transform.rotation);
            Instantiate(mediumAsteroid, position, Quaternion.identity);
            Destroy(gameObject);
            gameManager.SendMessage("UpdateScore", 50);
            break;
        case "Medium":
            Instantiate(smallAsteroid, new(position.x + 0.35f,
                                           position.y + 0.35f),
                                           transform.rotation);
            Instantiate(smallAsteroid, position, Quaternion.identity);
            Destroy(gameObject);
            gameManager.SendMessage("UpdateScore", 100);
            break;
        case "Small":
            Destroy(gameObject);
            gameManager.SendMessage("UpdateScore", 200);
            break;
    }
}
```

Kôd 8. Razdvajanje asteroida

Ovisno o veličini pogođenog asteroida (`size`), instanciraju se manji asteroidi i uništava pogođeni. Pozicija na koju se instanciraju raspolovljeni asteroidi dobije se od Transform komponente pogođenog asteroida. Prilikom instanciranja dva asteroida, jednom se blago izmjene koordinate kako se Collideri ne bi preklapali u trenutku pojavljivanja na zaslon.

Kvaternion je matematički prikaz trodimenzionalne rotacije. `Quaternion.identity` u Unityju konkretno predstavlja stanje bez rotacije. Koristi se kada se ne želi primijeniti rotacija na GameObject pri inicijalizaciji ili za resetiranje rotacije GameObjecta.

Ovisno o vrsti asteroida, metodom `SendMessage` šalje se broj bodova funkciji koja ih zbraja preko reference na skriptu u kojoj se to odvija (GameManager.cs). Leteći tanjur u Space Shooteru se pojavljuje i pokušava uništiti svemirski brod tako da prati igrača, pokušava se zabiti i puca na njega. Funkcionalnosti letećeg tanjura definirane su u skripti UfoControl.cs. Praćenje se implementira uz pomoć vektorske matematike. Kako bi se leteći tanjur mogao kretati prema svemirskom brodu potreban mu je smjer. Smjer se dobije oduzimanjem vektora položaja svemirskog broda od vektora položaja letećeg tanjura i postavljanjem magnitude rezultantnom vektoru na jedinicu, odnosno normalizacijom vektora (Kôd 9).

```
direction = (transformShip.position - transform.position).normalized;
rb.MovePosition(rb.position + direction * speed * Time.fixedDeltaTime);
```

Kôd 9. Implementacija pronalaženja i praćenja

Za kretanje se koristi metoda `Rigidbody.MovePosition` u funkciji `FixedUpdate`. Vektor smjera `direction` množi se s vrijednošću koeficijenta brzine `speed` za dobivanje konstantne brzine kretanja neovisno o udaljenosti. Vektor `direction` također se koristi za ispaljivanje projektila od letećeg tanjura prema igraču. Mehanika sustava za oštećivanje svemirskog broda pri sudaru odvija se u `OnCollisionEnter2D` funkciji u ShipControl.cs skripti. Preko objekta `Collision2D` dobije se sila sudara, koja se množi multiplikatorom i predstavlja štetu (Kôd 10). U ovisnosti od jačine udarca i akumulirane štete igraču se oduzima život.

```
float impactForce = collision.relativeVelocity.magnitude;
float damage = impactForce * damageMultiplier;
```

Kôd 10. Implementacija oštećenja

4.5. Razine i korisničko sučelje

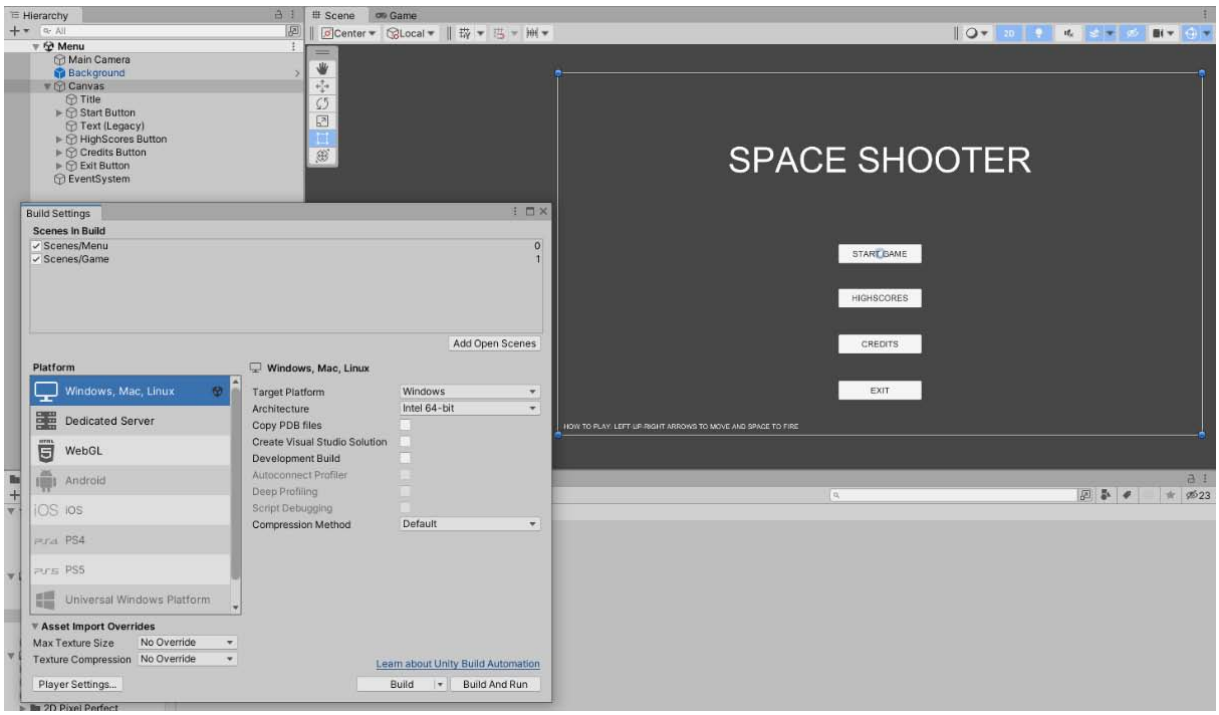
Upravljanje razinama odvija se u skripti GameManager.cs. U njoj se vrši promjena razina, zbrajaju i ažuriraju bodovi i upravlja korisničkim sučeljem. Kada tokom igranja igrač uništi sve asteroide i riješi se letećeg tanjura, automatski se pokreće nova razina s više asteroida i bržim i snažnijim letećim tanjurom, što čini svaku razinu težom od prethodne. Logika za promjenu razina okida se u trenutku kada igrač uništi leteći tanjur, pozivom metode

`CheckAsteroidCount` koja iterira kroz objekte u sceni, preko oznake dolazi do asteroida i računa njihov zbroj (Kôd 11). Ako petlja ne pronađe asteroide, počinje nova razina.

```
void CheckAsteroidCount()
{
    List<GameObject> allObjects = new List<GameObject>();
    Scene scene = SceneManager.GetActiveScene();
    scene.GetRootGameObjects(allObjects);
    int asteroidCount = 0;
    foreach(GameObject go in allObjects)
    {
        if (go.CompareTag("Asteroid"))
            asteroidCount++;
    }
    if (asteroidCount == 0)
            changeLevel();
}
```

Kôd 11. Provjera broja asteroida u sceni

Za kreaciju korisničkog sučelja (UI) stvori se objekt Canvas koji služi kao baza za sve elemente korisničkog sučelja. Na Canvas se dodaju UI elementi koji mogu biti tekst, slika, tipka i drugi. Upravljanje komponentama Canvasa odvija se preko prilagođene skripte. Glavni izbornik predstavlja zasebnu scenu koja se doda u postavkama izgradnje projekta te se postavi redoslijed scena tako da izbornik bude prva scena koja se učita pokretanjem aplikacije (Slika 12).



Slika 12. Canvas, elementi korisničkog sučelja i redoslijed scena

Stvaranje Glavnog izbornika i pod-izbornika za pauzu te izgradnja projekta u binarnu datoteku označava završetak projekta. Space Shooter je dostupan za preuzimanje na poveznici https://github.com/markanzec/SpaceShooterGame.

5. ZAKLJUČAK

Ovaj završni rad opisuje proces izrade 2D videoigara korištenjem pokretača videoigara Unity i programskog jezika C#.

Povijest videoigara koja se proteže od skromnih početaka do tehnološki naprednih remek-djela današnjice, služi kao svjedočanstvo ljudske kreativnosti i inovacije. Analizom povijesti videoigara pruža se razumijevanje evolucije igara kroz prizmu napretka tehnologije, te činjenice da su videoigre tokom godina postale globalna pojava i glavna grana industrije digitalne zabave.

Uloga pokretača je od iznimnog značaja u suvremenom razvoju videoigara. Pokretači videoigara, kao što je Unity, revolucionirali su industriju nudeći sveobuhvatan paket alata i resursa, omogućujući programerima da stvaraju igre velikom efikasnošću i zaobiđu složenost programiranja na niskoj razini. Pokretač Unity pruža pristupačno korisničko sučelje, robusne značajke, svestranost i prilagodljivost koje ga čine izborom za mnoge programere.

Praktična implementacija prikazana u 4. poglavlju pruža razumijevanje razvojnog procesa od koncepta do izvedbe, nudeći putokaz za stvaranje 2D videoigara korištenjem pokretača Unity i programskog jezika C#. Stvorena je jednostavna 2D igra po uzoru na klasičnu arkadnu igru *Asteroids*. Kroz njeno stvaranje predstavljen je tok rada pokretača Unity, način rukovanja sučeljem Unity Editora kao i stvaranje funkcionalnosti skriptiranjem u programskom jeziku C#. Prikazan je postupak uvoza sredstava te stvaranja objekata koji sačinjavaju scenu. Opisane su i prikazane glavne komponente koje definiraju karakteristike i funkcije objekata. Prikazano je rukovanje objektima u sceni te dodavanje i prilagođavanje komponenata putem korisničkog sučelja. Nadalje, prikazani su postupci i principi kodiranja skripti koje upravljaju objektima i komponentama, definirajući razne interakcije, funkcionalnosti i mehanike igre. Implementirane su mehanike kretanja i pucanja za igrača, mehanika omotavanja zaslona i logika za pronalaženje i praćenje igrača. Također je implementirana logika upravljanja razinama i elementima igre. Pritom su, kroz primjere kodova prikazane i objašnjene glavne funkcionalnosti i metode ugrađene u Unityjev API.

Kroz sve navedeno, vidljivo je kako izrada videoigara predstavlja spoj umjetnosti i tehnologije, gdje kreativnost i tehničko znanje idu ruku pod ruku kako bi se stvorila funkcionalna i zabavna videoigra. Očekuje se da će se u budućnosti ovo područje nastaviti širiti i razvijati, otvarajući nove mogućnosti za programere da ostvare svoje vizije i doprinesu bogatom svijetu videoigara uz pristupačne pokretače kao što je Unity.

Na kraju ovog rada, bitno je prepoznati da je svijet razvoja videoigara područje koje se neprestano razvija. Nove tehnologije, trendovi i izazovi oblikovat će budućnost razvoja kao i samu industriju videoigara.

LITERATURA

1. Clement, J., (2023), Video game market revenue worldwide from 2018 to 2028, https://www.statista.com/statistics/1344668/revenue-video-game-worldwide (pristupljeno 10.12.2023.)
2. godotengine.org, Homepage, https://godotengine.org/ (pristupljeno 10.12.2023.)
3. Cocos, Homepage, https://www.cocos.com/en (pristupljeno 10.12.2023.)
4. Unreal Engine, Homepage, https://www.unrealengine.com/en-US (pristupljeno 10.12.2023.)
5. Unity, Homepage, https://unity.com/ (pristupljeno 10.12.2023.)
6. Cryengine, Homepage, https://www.cryengine.com/ (pristupljeno 10.12.2023.)
7. Haas, J., (2014), *A History of the Unity Game Engine*, Worcester, Worcester Polytechnic Institute
8. Unity, Unity Personal, https://unity.com/products/unity-personal (pristupljeno 12.12.2023.)
9. wikipedia.org, Video game, https://en.wikipedia.org/wiki/Video_game (pristupljeno 12.12.2023.)
10. Chikhani, R., (2015), The History Of Gaming: An Evolving Community, https://techcrunch.com/2015/10/31/the-history-of-gaming-an-evolving-community/?guccounter=1 (pristupljeno 10.12.2023.)
11. museumofplay.org, Video Game History Timeline, https://www.museumofplay.org/video-game-history-timeline/ (pristupljeno 12.12.2023.)
12. Youtube, The History of Video Games Documentary, https://www.youtube.com/watch?v=z7-BN0qdZDk (pristupljeno 12.12.2023.)
13. wikipedia.org, Game engine, https://en.wikipedia.org/wiki/Game_engine (pristupljeno 15.12.2023.)
14. Arm, Gaming Engines, https://www.arm.com/glossary/gaming-engines (pristupljeno 15.12.2023.)
15. Butler, S., (2023), What is a Game Engine?, https://www.howtogeek.com/888619/what-is-a-game-engine/ (pristupljeno 10.12.2023.)

16. wikipedia.org, Unity (game engine), https://en.wikipedia.org/wiki/Unity_(game_engine) (pristupljeno 10.12.2023.)
17. wikipedia.org, Asteroids (video game), https://en.wikipedia.org/wiki/Asteroids_(video_game) (pristupljeno 10.12.2023.)
18. Unity User Manual 2022.3 (LTS), https://docs.unity3d.com/Manual/index.html (pristupljeno 10.12.2023.)
19. Unity Documentation, https://docs.unity3d.com/ScriptReference/index.html (pristupljeno 10.12.2023.)

SAŽETAK

Ovaj završni rad bavi se razvojem 2D videoigara, pružajući sveobuhvatan pregled razvojnog procesa. Rad počinje s povijesnim pregledom evolucije videoigara od njihovih rudimentarnih početaka do sadašnjih tehnološki naprednih oblika. Primarni fokus stavljen je na značaj pokretača u modernom razvoju videoigara, pri čemu je pokretač videoigre Unity u središtu pozornosti zbog svoje svestranosti i široke primjene. Detaljno su opisane značajke Unityja, naglašavajući pristupačno korisničko sučelje, robustan skup alata i njegovu ulogu u demokratizaciji razvoja videoigara. Nadalje, praktična primjena prikazana je kroz razvoj 2D videoigre, pojašnjavajući proces, korak po korak, od konceptualizacije do implementacije. Zaključno, rad prepoznaje dinamičnu prirodu razvoja videoigara i kontinuiranu evoluciju tehnologija i trendova, međutim naglašava da će temeljna načela opisana u rada ostati važna za programere u ovom području koje se stalno razvija.

Ključne riječi: razvoj 2D videoigara; povijest videogara; pokretač videoigre; Unity; C#

SUMMARY

This final thesis deals with the development of 2D video games by providing a comprehensive overview of the development process. The paper begins with a historical overview of the evolution of video games from their rudimentary beginnings to the current technologically advanced forms. The primary focus is on the significance of game engines in modern video game development, with the Unity game engine taking center stage due to its versatility and wide application. Unity features are described in detail, highlighting its user-friendly interface, robust toolset, and its role in democratizing video game development. Furthermore, a practical application is shown through the development of a 2D video game, clarifying the step-by-step process from conceptualization to implementation. Finally, the paper recognizes the dynamic nature of video game development and the continuous evolution of technologies and trends but emphasizes that the fundamental principles described in the paper will remain important for developers in this ever-evolving field.

Keywords: 2D Game Development, Videogame History, Game Engine, Unity, C#