

Šolski center Novo mesto

Srednja elektro šola in tehniška gimnazija

Šegova ulica 112

8000 Novo mesto

IZDELAVA IGER - PREŽIVETJE V VESOLJU
(Maturitetna seminarska naloga)

Predmet: Računalništvo

Avtor: Nejc Cekuta

Razred: T4B

Mentor: dr. Albert Zorko univ. dipl. inž. el.

Novo mesto, april 2021/2022

POVZETEK

V tej seminarski nalogi se ukvarjamo s procesom izdelave iger. V nalogi je najprej dobro predstavljen proces izdelave igre, in kako po posameznih postopkih igra počasi nastaja. Glavni postopki so predproukcija, produkcija in postprodukcija. Nato so na kratko predstavljena vsa programska orodja, katera so bila uporabljena, tako pri načrtovanju, kot tudi pri oblikovanju, programiranju in izvrševanju igre. Med uporabljeno opremo spadajo Photoshop in Slikar, ki sta bila uporabljena za načrtovanje, oblikovanje in za izgled igre, Visual Studio Code, programsko okolje v katerem je zapisana vsa potrebna koda, in kot glavni je opisan igralni pogon Unity, s pomočjo katerega vse elemente združimo in lahko brez večjih težav igro igramo. V predstavitvi je natančno opisana njihova funkcionalnost in uporabnost, in kako pripomorejo različnim uporabnikom, pri najrazličnejših projektih, v vsakdanjem življenju. Nato sledi podroben opis načrtovanja in procesa izdelave moje igre. Na koncu naloge pa sledi še kratka predstavitev končane in delujoče igre, ki je bila narejena v igralnem pogonu Unity, skupaj z vsemi že prej opisanimi elementi, ki so uporabljeni v igri, ki se imenuje Preživetje v vesolju.

KLJUČNE BESEDE:

- Programska oprema
- Programska koda
- Preživetje
- Igralec
- Unity
- Igra

KAZALO VSEBINE

1	UVOD	1
2	IZDELAVA IGER - PREŽIVETJE V VESOLJU	2
2.1	PROCES IZDELAVE IGER.....	2
2.1.1	NAČRTOVANJE	3
2.1.2	PREDPRODUKCIJA	4
2.1.3	PRODUKCIJA	5
2.1.4	TESTIRANJE.....	6
2.1.5	PREDLANSIRANJE.....	6
2.1.6	LANSIRANJE	7
2.1.7	POSTPRODUKCIJA.....	7
2.2	PROGRAMSKA OPREMA UNITY	8
2.2.1	SPLOŠNO O UNITYU	9
2.2.2	UPORABA	9
2.2.3	SKLEP	13
2.3	IZDELAVA IGRE	14
2.3.1	IGRALNI PROSTOR.....	14
2.3.2	IGRALEC.....	15
2.3.3	METEK	17
2.3.4	ASTEROID	18
2.3.5	GENERATOR ASTEROIDOV	19
2.3.6	GAME MANAGER	19
2.3.7	MENI.....	21
3	ZAKLJUČEK.....	23
4	ZAHVALA	24
5	VIRI IN LITERATURA	25
6	STVARNO KAZALO	27
7	PRILOGE.....	28

KAZALO SLIK

Slika 1: Proces izdelave igre (Vir: g2.com)	2
Slika 2: Sodelovanje razvijalcev (Vir:logicsimplified.com)	4
Slika 3: Ustvarjanje igre (Vir: g2.com)	5
Slika 4: Igralna konvencija (Vir: screenrant.com)	6
Slika 5: Logotip VS Code (Vir:dev.to)	8
Slika 6: Logotip Unitya (Vir:develop4fun.fr)	9
Slika 7: Prikaz projektov v Unity Hubu	9
Slika 8: Hierarhija	10
Slika 9: Scena	11
Slika 10: Igra	11
Slika 11: Pregledovalnik	12
Slika 12: Projekt	12
Slika 13: Konzola	13
Slika 14: Meje ekrana.....	14
Slika 15: Preverjanje vnosa uporabnika	15
Slika 16: Dodajanje sile igralcu.....	15
Slika 17: Streljanje igralca	16
Slika 18: Preverjanje trkov	16
Slika 19: Lastnosti metka	17
Slika 20: Metodi za metek.....	17
Slika 21: Preverjanje trkov	18
Slika 22: Metoda razpolovi	18
Slika 23: Generiranje asteroidov	19
Slika 24: Začetek igre.....	19
Slika 25: Število življenj	20
Slika 26: Točkovanje.....	20
Slika 27: Konec igre	21
Slika 28: Metode gumbov	21
Slika 29: Nastavitve resolucije	22
Slika 30: Nastavitve glasnosti	22

1 UVOD

Igre na našem planetu obstajajo že tako dolgo kot človeštvo. Ko pa smo ljudje začeli izumljati računalnike, so začele nastajati računalniške igre. Načrtovanje in proces izdelave igre pa je daleč od tega, da bi lahko rekli da je preprost. In zato je moj glavni namen te seminarske naloge, da vam predstavim svet oblikovanja iger in vam morda celo obudim zanimanje zanj.

Moj glavni cilj pri izdelavi te seminarske naloge je izdelati interaktivno in zanimivo igro, ki bo uporabnika pritegnila. Pomembno je tudi, da je narejena preprosto in pregledno, saj jo bo uporabnik le tako lahko brez težav preizkusil. Poleg praktičnega dela, pa je moj cilj predstaviti tudi nekaj teoretičnih osnov o tem kako poteka proces izdelave igre, še predvsem pa dobro opisati igralni pogonu Unity, ki je glavni program, s pomočjo katerega je bila igra narejena.

Pri pisanju maturitetne naloge bom povzemal literaturo drugih avtorjev, ki jo bom našel tako na spletu, kot tudi v knjigah. Pri praktičnem delu naloge, kjer bom predstavil igro, bom za izdelavo in prikaz igre uporabil igralni pogon Unity, v katerem bom za programiranje uporabil programski jezik C#.

2 IZDELAVA IGER - PREŽIVETJE V VESOLJU

2.1 PROCES IZDELAVE IGER

Prvi primer razvoja iger je mogoče zaslediti v letu 1940, ko je dr. Edward Uhler Condon izdal igralni stroj, ki je temeljil na starodavni matematični igri Nim. Vendar pa je bil prvi sistem iger za komercialno domačo uporabo zasnovan in predstavljen šele tri desetletja pozneje, in sicer leta 1967 s strani Ralph Baerja in njegove ekipe. Igralni sistem, znan kot Brown Box, je bil vezje, ki ga je bilo mogoče povezati s televizijskim sprejemnikom. Lahko si ga programirali za igranje različnih iger, kot so dama in ping-pong. (1).

Leta 1972 je Atari postalo prvo igralno podjetje, ki je postavilo merilo za obsežno igralniško skupnost. Začelo je prodajati svojo elektronsko video igrico. In tako so se arkadni stroji začeli pojavljati v nakupovalnih centrih, barih, restavracijah in kegljiščih po vsem svetu. Nato je med letoma 1972 in 1985 več podjetij začelo razvijati video igre za nenehno rastoči trg. In od takrat naprej pa vse do danes, so se zmogljivosti računalnikov tako izboljšale, da se zdi, da vsaka nova serija iger in konzol prekaša prejšnjo generacijo, čemur pa še daleč ni videti konca (1).

Razvoj in izdelava iger je proces ustvarjanja, ki opisuje zasnovo igre, njen razvoj in na koncu izdajo igre. Razvojni del iger na splošno vključuje programiranje, kodiranje, upodabljanje, inženiring in testiranje igre in je ponavadi razdeljen na 7 faz, kot lahko vidimo tudi na Slika 1 (1):

- Načrtovanje
- Predprodukcija
- Produkcija
- Testiranje
- Predlansiranje
- Lansiranje
- Postprodukcija



Slika 1: Proces izdelave igre (Vir: g2.com)

2.1.1 NAČRTOVANJE

Preden pisci začnejo pisati, oblikovalci začnejo oblikovati in razvijalci začnejo razvijati, se mora najprej pojaviti ideja za video igo. To je prvi del faze načrtovanja in korenin, iz katerih bo zrasla vsaka video igra.

V fazi načrtovanja je treba odgovoriti na najbolj osnovna vprašanja, kot so (2):

- Kakšno vrsto videoigre izdelujemo?
- Ali bo 2D ali 3D
- Katere ključne lastosti mora imeti?
- Kdo so liki, ki bojo nastopali?
- Kdaj in kje poteka?
- Kdo je ciljna skupina igre?
- Na kateri platformi bo igra?

Morda se ne zdi tako, vendar je idejna zasnova videoigre eden najtežjih, vendar najpomembnejših delov razvoja iger. Ideja, ki jo razvije igralni studio, bo služila kot osnova celotne igre. To nas pripelje do naslednjega dela razvoja - preverjanja koncepta. Pri preverjanju koncepta se upoštevajo vse ustvarjene zamisli in začne se ugotavljati, kako izvedljive so za igralni studio. Nato bo treba odgovoriti na nekatera dodatna vprašanja, kot so (2):

- Kolikšni bojo stroški razvoja te igre?
- Kako velika bo morala biti ekipa?
- Kolikšen je predviden časovni okvir za lansiranje igre?
- Ali imamo dovolj dobre tehnološke zmogljivosti?

Nato je ključnega pomena narediti poskusno različico koncepta, saj z njo predstavite ideje ciljni publiki in tako dobite odziv ali je publika sploh zainteresirana za vaš projekt in ali se splača nadaljevati. Ko dobite potrditev s strani publike in morda kakšno denarna podporo je čas za začetek predprodukcije.

2.1.2 PREDPRODUKCIJA

Naslednja faza razvoja igre, je imenovana predprodukcija, razmišlja se o tem, kako oživiti številne ideje, predstavljene v fazi načrtovanja. Tukaj pisci, umetniki, oblikovalci, razvijalci, inženirji, vodje projektov in drugi ključni oddelki sodelujejo med seboj in razpravljajo kako bo igra narejena na najboljši način. To predstavlja tudi Slika 2. Nekateri primeri tega sodelovanja so lahko vidni kot (2):

- Potek srečanja pisateljev z vodji projekta, na katerem se podrobneje predstavi zgodba. Dopolnjujejo kdo je glavni junak, kakšna je njegova zgodba, kako so liki povezani med seboj in kakšen je konec ali več njih.
- Inženirji se srečujejo s pisci in jim sporočajo, da zaradi trenutnih tehnoloških omejitev, jim ne morejo omogočiti določenih stvari, saj bi to pomenilo možnost sesutja igre.
- Umetniki se srečujejo z oblikovalci, da bi zagotovili, da so vizualni elementi, barve in umetniški slogi skladni in usklajeni s tistim kar je bilo določeno v načrtovanju.
- Razvijalci se srečujejo z inženirji, da bi razjasnili mehaniko v igro, fiziko in način izrisa predmetov na igralčevem zaslonu.

Od tu naprej studii običajno izdelujejo prototipe likov, okolij, vmesnikov, nadzornih shem in drugih elementov v igri, da vidijo, kako so videti, kako se počutijo in kako delujejo drug na drugega. To je v bistvu trenutek preden preidemo na glavni del razvoja - proizvodnjo.



Slika 2: Sodelovanje razvijalcev (Vir: logicsimplified.com)

2.1.3 PRODUKCIJA

Večino časa, truda in sredstev, porabljenih za razvoj videoiger, je v fazi produkcije. To je tudi ena najzahtevnejših faz razvoja videoiger. Proizvodnja poteka v treh glavnih korakih (3):

1. **Grafika in oblikovanje** - 2D/3D umetniki sodelujejo pri ustvarjanju likov, sredstev, vizualnih učinkov, okolij in ostalih elementov. Oblikovalci okolja določijo strukturo okolja in glavne ovire na poti igralca, kar prikazuje Slika 3.
2. **Programiranje** - Čeprav so se programerji v igro vključili že med izdelavo prototipa, se njihovo glavno delo začne tukaj. Na podlagi obstoječega okolja ali okolja razvitega od začetka, ustvarijo osnovo izdelka, ki ga je že mogoče igrati in pri katerem je mogoča interakcija z vsemi glavnimi elementi igre.
3. **Oblikovanje zvoka** - Zvočni inženirji ali oblikovalci zvoka oblikujejo zvok za igro. To delo vključuje zvočne učinke, glasovne vložke in glasbo, ki so pogosto dinamični in se spreminjajo glede na igralčeva dejanja in dogajanje v igri.

Proizvodnja je eden od najdaljših korakov razvoja igre, saj ekipa v izvede veliko število prilagoditev in popravkov. Preden bo dosežen popoln vizualni in tehnični prikaz igre, bo potrebno odpraviti veliko napak. (3)

Pri razvoju videoiger se pogosto zgodi, da se po dokončanju igre zavrže celoten segment igre, ki je vreden več mesecev dela. Lahko si predstavljate, kako neprijetno je to za zaposlene. Te vrste sprememb se običajno pojavijo v fazi testiranja.



Slika 3: Ustvarjanje igre (Vir: g2.com)

2.1.4 TESTIRANJE

Vsako funkcijo in mehaniko v igri je potrebno preizkusiti. Igra, ki ni bila temeljito preizkušena ni pripravljena niti za alfa izdajo igre. V tej fazi je pomembno, da preizkuševalec igre opozori razvijalce na določene stvari (3):

1. Ali obstajajo območja v igri, kjer se pojavljajo napake
2. Ali lahko igralec na kakršenkoli način igro ogoljufa
3. Ali je igra dolgočasna
4. Ali je igra pretežka/prelahka
5. Ali so vsi elementi igre prikazani na zaslonu

Ko testerji ugotovijo težave, igra ponovno pride v roke programerjev ali umetnikov, ki izvedejo spremembe. Ta cikel se lahko večkrat nadaljuje, dokler testerji ne odobrijo lansiranja igre. In po neštetih urah testiranja in popravljanja naj bi bila igra pripravljena za izdajo. To je prvič, ko gre igra v javnost in jo lahko ljudje preizkusijo.

2.1.5 PREDLANSIRANJE

Postopek izdelave videoigre včasih ne vključuje te faze. Če pa govorimo o velikem projektu, je ta faza pred zagonom nujno potrebna. Tu govorimo o trženju, torej o vnaprejšnjem obveščanju ljudi o začetku novega projekta igre. Načini trženja so lahko različni: kratek video v katerem je igra predstavljena, članki in recenzije, različne slike in podobe iz igre. Včasih so ekskluzivni predogledi igre organizirani tudi na konvencijah o igrah ali tematskih srečanjih, kot lahko vidimo na Slika 4.

Neodvisni studii si ne morejo vedno privoščiti velikih proračunov za trženje, da bi pritegnili pozornost k svojim igram. Na srečo sta lahko množično financiranje in oglaševanje prav tako uspešna. Pogosta metoda neodvisnih studiev je pošiljanje kopij znanim osebnostim spletnega igranja, da njihovo igro še dodatno promovirajo (3).



Slika 4: Igralna konvencija (Vir: [screenrant.com](https://www.screenrant.com))

2.1.6 LANSIRANJE

Promocijska kampanja je končana, ostal je samo še najpomembnejši korak – izdaja ali lansiranje igre. Do tega trenutka lahko ekipa še vedno izvede nekaj manjših popravkov in dopolnitev. Običajno gre za vizualne popravke: izboljšanje kakovosti tekstur, izboljšanje animacije in izpopolnjevanje modelov. Tudi navidezno majhne podrobnosti so pomembne, da je igra še bolj barvita, zabavna in poglobljena (4).

Lansiranje igre pomeni njeno posredovanje trgovinam z igrami. Nato jo bodo igralci lahko kupili ali brezplačno prenesli, odvisno od tega, za katero platformo bo izdana in pod kakšnimi pogoji. Toda tudi ko je igra že na svetu, se postopek oblikovanja videoigre s tem ne konča. Sledi faza postprodukcije (4).

2.1.7 POSTPRODUKCIJA

Ko je končna različica igre izdana, preide v zadnjo fazo procesa razvoja igre - postprodukcijo. Glavni namen te faze je vzdrževanje igre, ki vključuje predvsem (5):


- **Odpravljanje napak** - Kljub prizadevnim preizkuševalcem večina iger ob izidu še vedno vsebuje manjše napake. Prvih nekaj mesecev med postproduksijsko fazo je običajno namenjenih odkrivanju in odpravljanju teh napak.
 - **Dodajanje nove vsebine** - vključuje redne posodobitve programske opreme za igro, od popravkov za uravnoteženje igre do novih dodatkov, kar je prikazano na **Error!**
- Reference source not found.**

Nobena igra ni enaka in tudi izkušeni studii za razvoj iger z več sto igrami v rokah se pogosto spopadajo s spremembami v zadnjem trenutku, kratkimi roki, ustvarjalnimi razlikami in drugimi izzivi. K uspehu igre prispeva veliko stvari, na katere razvijalec nima neposrednega vpliva. To je še razlog več, da poskrbite za stvari, na katere lahko imate vpliv. Strukturiran proces razvoja igre z jasnimi roki in cilji je za to še vedno najbolj enostaven način (5).

2.2 PROGRAMSKA OPREMA UNITY

Danes je večina iger ustvarjena s pomočjo grafičnih pogonov ali orodji za izdelavo iger, ki delujejo na različnih igralnih platformah. Poznamo različne igralne pogone in platforme. Najbolj znani igralni pogoni, ki spadajo med tri najbolj priljubljene so Unity, Unreal Engine in Godot. Platforme so razdeljene po skupinah glede na uporabo. Od računalniških do mobilnih in konzolnih. Znane platforme so Microsoft Windows, Android, Xbox, Nintendo, iOS, macOS, Linux (6).

Pri razvoju iger pa je pomembno, da se poslužujemo uporabe tudi nekaterih drugih programskih orodji, s pomočjo katerih zagotovimo igri lažje načrtovanje, oblikovanje določenih elementov igre in programiranje v okolju, ki nam za to predstavlja najmanjši napor in s katerim najlažje pridemo do željenih rezultatov. Programska orodja, ki so bila uporabljena pri izdelavi te igre so:

- **Photoshop** - programska oprema za urejanje fotografij in grafično oblikovanje elementov, ki uporabnikom omogoča ustvarjanje, urejanje in manipulacijo različnih grafik ter digitalne umetnosti. Omogoča tudi ustvarjanje in urejanje slik z več sloji ter uvoz slik v različnih datotečnih formatih. Program Photoshop je razvil Adobe Systems za operacijska sistema Windows in MacOS (7).
- **Visual Studio Code** - je brezplačni urejevalnik besedila podjetja Microsoft. Na voljo je za operacijske sisteme Windows, Linux in macOS. Čeprav je urejevalnik razmeroma lahek, vsebuje nekaj zmogljivih funkcij, zaradi katerih je v zadnjem času eno najbolj priljubljenih orodij razvojnega okolja. Podpira širok nabor programskih jezikov, od Java, C# in Pythona do CSS. Poleg tega omogoča dodajanje in celo ustvarjanje novih razširitev, vključno z

Visual Studio Code
Slika 5: Logotip VS Code (Vir: dev.to)
razhroščevalniki kode, ter nudi podporo za razvoj v oblaku in spletu (8).
- **Unity** – igralni pogon, ki ima številne vgrajene funkcije, kot je na primer fizika in zaznavanje trkov. To omogoča razvijalcem, da se osredotočijo na težje elemente v njihovi igri, in hitreje pridejo do željenih rezultatov. Več podrobnosti o Unityu pa v nadaljevanju.

2.2.1 SPLOŠNO O UNITYU

Unity je 2D in 3D pogon za igre, ki je na voljo že od leta 2005. Razvila ga je družba Unity Technologies, da bi več razvijalcem omogočila dostop do orodij za razvoj iger, kar je bilo v tistih časih novost. V svoji dolgi življenjski dobi se je močno spremenil in razširil, pri tem pa mu je uspelo slediti najnovejšim tehnologijam.

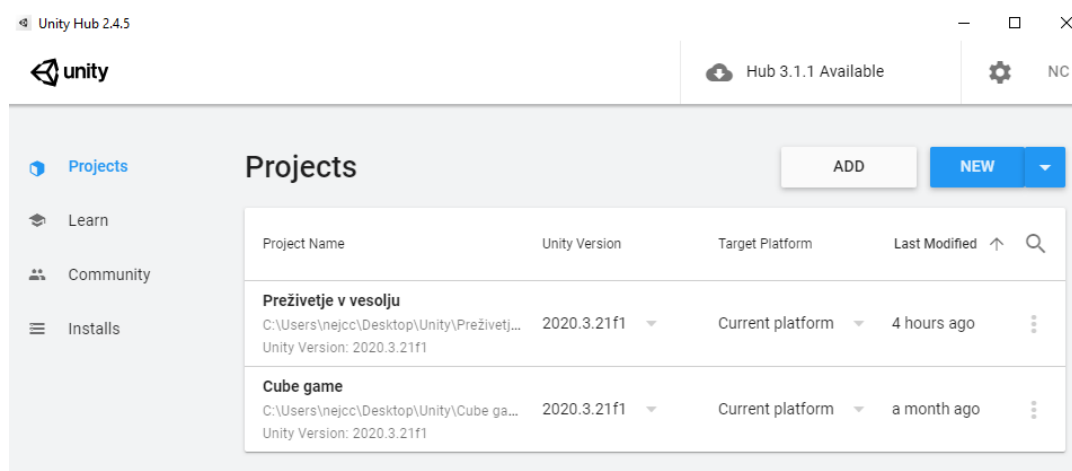
Še danes je glavni cilj pogona zagotoviti čim boljši nabor orodij za industrijo razvoja iger in čim bolj olajšati uporabo pogona razvijalcem iger na vseh ravneh znanja (vključno z začetniki). Svoj doseg so razširili tudi na druge panoge in se osredotočili na razvoj 3D, zaradi česar je to eden najzmogljivejših pogonov na voljo (9).



Slika 6: Logotip Unitya (Vir: develop4fun.fr)

2.2.2 UPORABA

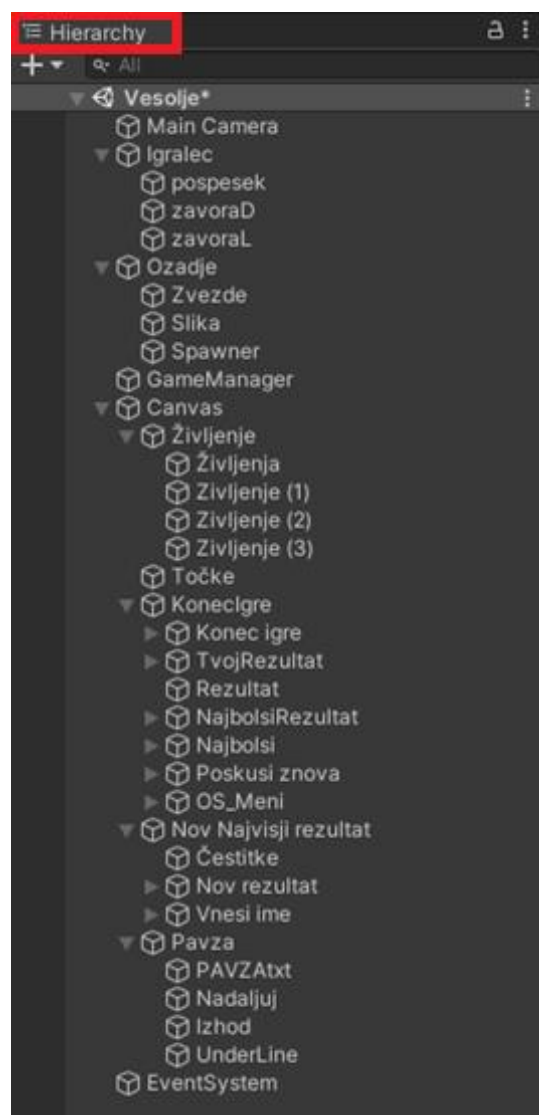
Program Unity je zelo preprosto prenesti in namestiti. Če ga želite prenesti, obiščite uradno spletno stran Unitya. Tam boste lahko prenesli programsko vozlišče Unity Hub, ki je upravitelj prenosov, s katerim boste lahko upravljali različne različice programske opreme Unitya in vse dodatne funkcije, ki jih boste potrebovali. Ko imate program Unity Hub, lahko izberete različico pogona, ki jo želite prenesti. Namestitveni program vas bo vodil skozi preproste korake. Ko imate naloženo določeno različico Unitya, lahko vanj naložite že obstoječe projekte ali pa ustvarite nove, kot je prikazano na Slika 7 (10).



Slika 7: Prikaz projektov v Unity Hubu

Ko s pomočjo Unity Huba odprete ali ustvarite določen projekt, se bo odprl program Unity, kateri pa se vam bo na začetku morda zdel zakompliciran, saj vsebuje veliko število oken, ikon in najrazličnejših možnosti za urejanje elementov. Na srečo pa so stvari preprostejše, kot je videti. V nadaljevanju so predstavljena glavna okna in njihove funkcije.

1. **Hierarhija (Hierarchy)** - Privzeto je na skrajni levi strani, prikazan je dolg seznam vseh igralnih objektov v sceni, kot je prikazano na Slika 8. Tako lahko hitro poiščete in izberete kateri koli objekt, ter spremenite njegove lastnosti. Objekti igre so preprosto elementi, ki so vključeni v vašo igro (10).



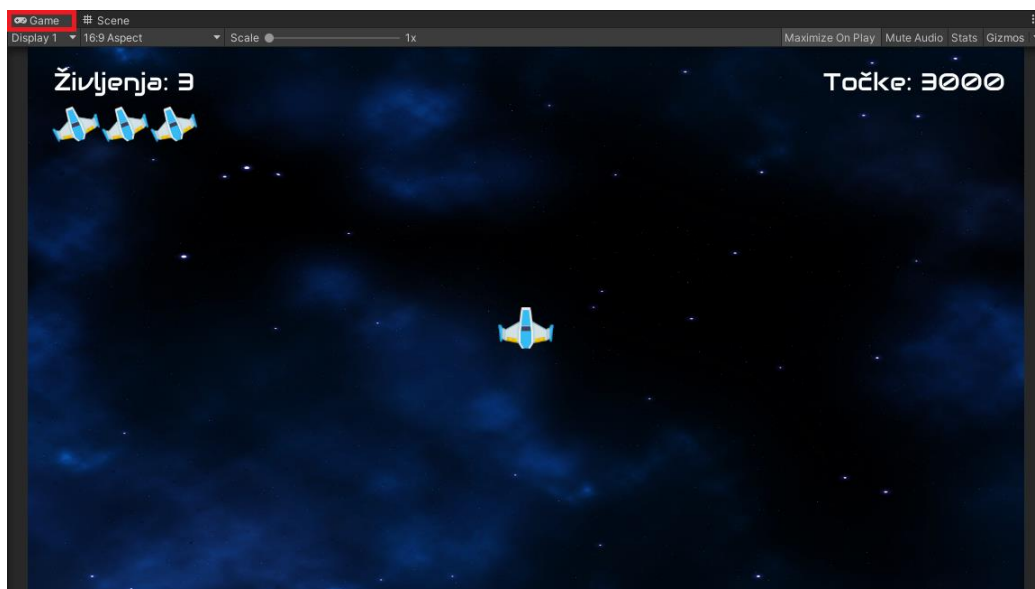
Slika 8: Hierarhija

2. **Scena (Scene)** – Največje okno v sredini programskega pogona Unity. Prikaže se pogled na trenutno raven, meni ali igralni svet, s katerim trenutno delate. V tem oknu lahko prosto vlečete, spuščate, povečujete in krčite objekte igre. Prikazano na spodnji Slika 9 (10).



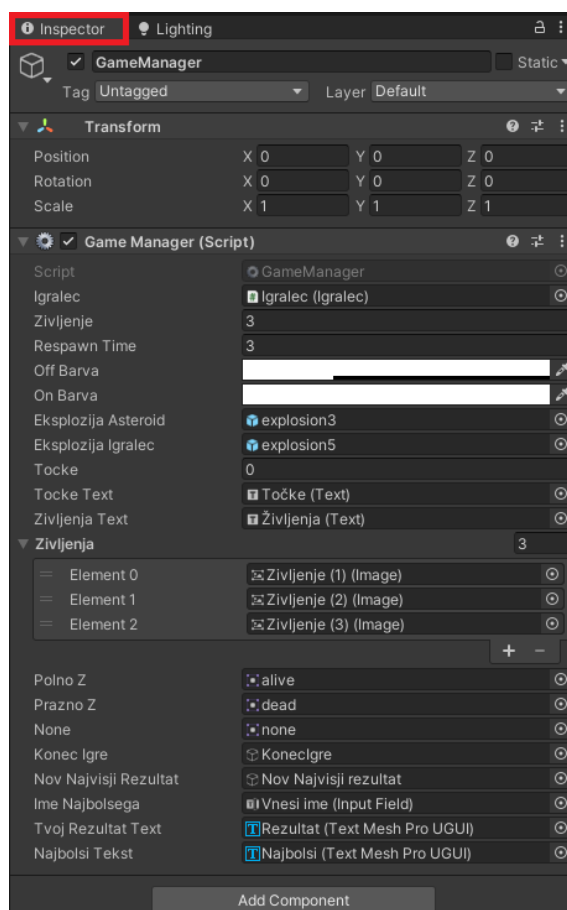
Slika 9: Scena

3. **Igra (Game)** – Običajno je zraven okna scena, do katerega lahko dostopate tako, da pritisnete zavihek na vrhu. Pogled igre prikazuje pogled na vašo sceno, kot je prikazana v igri. To prikazuje Slika 10. To pomeni, da boste imeli enako perspektivo kot kamera in ne boste mogli premikati objektov. Na tem mestu se lahko igro tudi preizkusi (10).



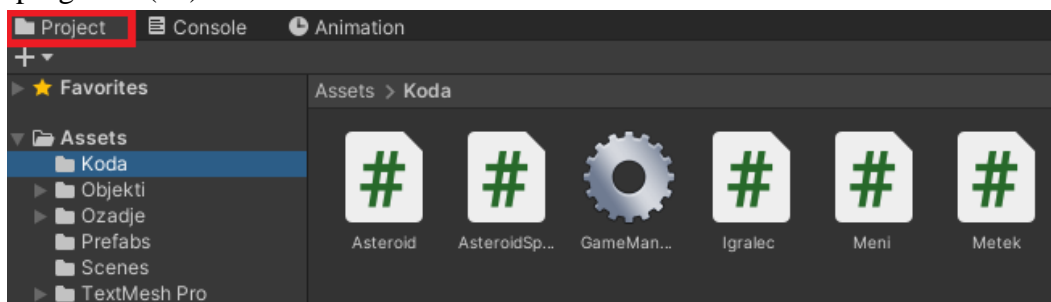
Slika 10: Igra

4. **Pregledovalnik (Inspector)** – To okno se nahaja na skrajni desni strani programa. Prikazan na Slika 11. V pregledovalniku si lahko ogledate in urejate lastnosti izbranega objekta v igri. To pomeni, da lahko spreminjamo velikost ali položaja objekta. Tu pa lahko objektu dodajamo tudi druge različne komponente kot so na primer razne skripte, s katerimi lahko igralcu omogočimo premikanje ali komponente za lažje zaznavanje trkov (11).



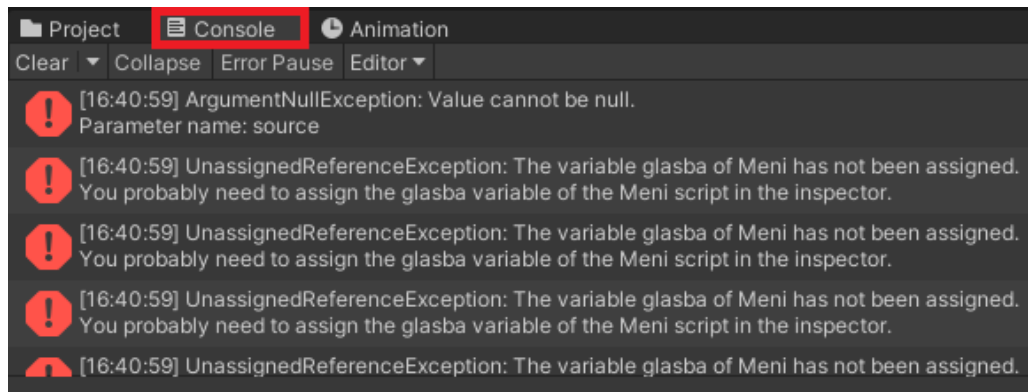
Slika 11: Pregledovalnik

5. **Projekt (Project)** - Se nahaja na dnu zaslona in prikazuje vse datoteke, ki sestavljajo igro. Tu lahko ustvarimo skripte, ki jih nato dodamo k različnim objektom. Sem lahko dodajamo tudi datoteke ali texture, ki so bile narejene s pomočjo kakšnega drugega programa (11).



Slika 12: Projekt

6. **Konzola (Console)** – Kot je prikazana na spodnji Slika 13, se nahaja na dnu zaslona, poleg »projekta«. Je mesto, kjer lahko vidite informacije, ki vam jih sporoča orodje Unity. Tako boste vedeli, ali so napake v vaši kodi ali pa je treba odpraviti težave s samo nastavitvijo programske opreme Unity (11).



Slika 13: Konzola

2.2.3 SKLEP

Unity je še vedno eden izmed najbolj priljubljenih pogonov, in to z dobrim razlogom. Ponuja številne funkcije in prednosti, ne glede na to, ali izdelujete igre ali potrebujete 3D grafiko za kakšno drugo panogo. Omogoča uporabo najrazličnejših orodji, katera razvijalcem zelo olajšajo uporabo pogona. Je eden najzmogljivejših pogonov na voljo in v svoji dolgi življenjski dobi se je močno spremenil in razširil, pri tem pa mu je uspelo slediti najnovejšim tehnologijam. Programski pogon Unity vse uporabnike že od samega začetka navdušuje s svojo preprostostjo in uporabnostjo, čemur pa ni videti konca, saj je iger v današnjem svetu čedalje več, s tem pa tudi narašča število razvijalcev (9).

2.3 IZDELAVA IGRE

V tem delu seminarske naloge, pa vam bom predstavil celoten potek, kako sem s pomočjo programskega pogona Unity ustvaril svojo igro. Igra je zasnovana na način preživetja, kar pomeni, da igralec poizkuša v igri preživeti čim dlje. Ko vstopimo v igro, se nam najprej prikaže osnovni meni, kjer lahko urejamo nastavitve igre, pogledamo kakšna so pravila igre in kako se jo sploh igra, ali pa kar takoj pričnemo z igro. Med igranjem se igralcu prištevajo točke, ki jih pridobiš ko uničiš določen objekt (asteroid), katere pa se ob zaključku igre prikažejo na ekranu in se tudi primerjajo s dozdajšnjim najboljšim rezultatom.

2.3.1 IGRALNI PROSTOR

Igra vsebuje zelo veliko elementov in uporabljene je tudi kar precej kode, saj sem ustvaril 6 različnih skript kode, katere pa bom v nadaljevanju podrobneje predstavil. Zato je dobro da se lotimo stvari kar na začetku. Da sploh lahko začnemo dodajati igri elemente in začnemo pisati kodo, je bilo najprej potrebno v programu Unity ustvariti igralni prostor, kateremu pa smo določili tudi meje in s tem določili do kod se igralec lahko premika (koda na Slika 14), za lepši izgled pa sem mu dodal še sliko za ozadje.

```
private Vector2 MejeEkrana;
3 references
private float SirinaIgralca;
3 references
private float VisinaIgralca;
0 references
void Start()
{
    MejeEkrana = Camera.main.ScreenToWorldPoint(new Vector3(Screen.width, Screen.height, Camera.main.transform.position.z));
    SirinaIgralca = transform.GetComponent().bounds.extents.x;
    VisinaIgralca = transform.GetComponent().bounds.extents.y;
}
0 references
void LateUpdate()
{
    Vector3 pozicija = transform.position; //trenutna pozicija objekta
    pozicija.x = Mathf.Clamp(pozicija.x, MejeEkrana.x * -1 + SirinaIgralca, MejeEkrana.x - SirinaIgralca);
    pozicija.y = Mathf.Clamp(pozicija.y, MejeEkrana.y * -1 + VisinaIgralca, MejeEkrana.y - VisinaIgralca);
    transform.position = pozicija;
}
```

Slika 14: Meje ekrana

S pomočjo te kode, najprej z metodo `Start()`, ki se izvede na začetku igre, poiščemo širino in višino zaslona na katerem igramo in širino in višino igralca. Te podatke potem uporabimo v spodnji metodi `LateUpdate()`, v kateri pa je določeno, da pozicija igralca (koordinati x in y) nesmeta biti večji kot je velikost zaslona in tako igralec ostane v igralnem polju ves čas.

2.3.2 IGRALEC

Ko imamo narejen igralni prostor, pa mu lahko začnemo dodajati elemente. Najprej sem ustvaril objekt, ki sem ga poimenoval igralec, s pomočjo Photoshopa sem mu dodelil izgled, ki predstavlja vesoljsko ladjo, nato pa sem zanj ustvaril skripto, ki pa vsebuje kodo za premikanje, streljanje, prikaz raznih animacij in kodo, ki določi kaj se z igralcem zgodi ko trči v drug objekt.

```
private void Update() //vedno deluje
{
    Pospesek = Input.GetKey(KeyCode.W) || Input.GetKey(KeyCode.UpArrow);
    Zavora = Input.GetKey(KeyCode.S) || Input.GetKey(KeyCode.DownArrow);

    if(Input.GetKey(KeyCode.A) || Input.GetKey(KeyCode.LeftArrow)){
        Obracanje = 1.0f;
    }else if(Input.GetKey(KeyCode.D) || Input.GetKey(KeyCode.RightArrow)){
        Obracanje = -1.0f;
    }else{
        Obracanje = 0.0f;
    }

    if(Input.GetKeyDown(KeyCode.Space) || Input.GetMouseButtonDown(0)){
        Streljanje();
    }
}
```

Slika 15: Preverjanje vnosa uporabnika

Na zgornji Slika 15, je predstavljena koda, s katero preverimo vnos uporabnika, in na podlagi tega se določenim spremenljivkam določijo določene vrednosti. Na primer, če uporabnik pritisne tipko W ali puščico za gor, se spremenljivka nastavi na true in se izvede nadaljna koda. Podobno velja tudi za ostale tipke.

Na Slika 16 je predstavljena metoda, ki na podlagi vnosa uporabnika izvede določeno kodo. Če sta spremenljivki pospešek ali zavora true, se izvede koda in doda igralcu silo za premik gor ali dol. Zavrti se tudi animacija. Podobno velja tudi za obračanje, le da je tu igralcu sila dodana v desno ali v levo smer, odvisno od pritisnjene tipke.

```
private void FixedUpdate() //določen cas deluje
{
    if(Pospesek){
        telo.AddForce(this.transform.up * this.HitrostPospeska);
        ogenjPospesek.Play(); // zavrti animacijo pospeksa (ogenj)
    }else if(Zavora)
    {
        telo.AddForce(this.transform.up * -this.HitrostPospeska);
        prikaziZavoraD(); //animacija
        prikaziZavoraL(); // animacija
    }

    if(Obracanje == 1.0f){
        telo.AddTorque(Obracanje * this.HitrostObracanja);
        prikaziZavoraL();
    }else if(Obracanje == -1.0f)
    {
        telo.AddTorque(Obracanje * this.HitrostObracanja);
        prikaziZavoraD();
    }else
    {
    }
}
```

Slika 16: Dodajanje sile igralcu

Spodnja koda na Slika 17, deluje podobno kot prejšnja. Na podlagi vnosa igralca iz prejšnje kode, če pritisne space ali levi klik na miški, se izvede metoda `Streljanje()`, ki v prostor prikaže drugi narejeni objekt, in sicer metek in pa izvede se metoda `Strel()`, ki se nahaja v skupku kode metek, katerega pa bom opisal v nadaljevanju.

```
private void Streljanje()
{
    1 reference
    Metek metek = Instantiate(this.MetekPrefab, this.transform.position, this.transform.rotation);
    metek.Strel(this.transform.up);
}
```

Slika 17: Streljanje igralca

In še zadnja metoda v s skripti igralec pa je metoda, ki je že ustvarjena s strani Unitya in je zato tudi lažja implementacija te metode v Unity. Na spodnji sliki lahko vidimo, da je v metodi if stavek, s pomočjo katerega preverjamo ali je prišlo do kakšnega stika med igralcem in asteroidom. In če je prišlo do stika, se objekt igralca ustavi in postane neaktiven, kar pomeni, da ga ne moremo več upravljati. Izvede se še metoda `IgralecMrtev()`, ki pa je predstavljena pod podnaslovom game manager.

```
private void OnCollisionEnter2D(Collision2D collision)
{
    if(collision.gameObject.CompareTag("Asteroid"))
    {
        telo.velocity = Vector3.zero; //hitrost gre na 0
        telo.angularVelocity = 0.0f;

        this.gameObject.SetActive(false); //igralec je neaktiven

        FindObjectOfType<GameManager>().IgralecMrtev(this);
    }
}
```

Slika 18: Preverjanje trkov

2.3.3 METEK

Naslednji objekt, ki sem ga ustvaril v Unityu in za katerega sem ustvaril script je objekt metek, ki predstavlja objekt, ki se prikaže na ekranu v trenutku, ko uporabnik pritisne tipko space. Kot lahko vidimo na Slika 19 in Slika 20, ima samo 3 lastnosti in pa 3 metode.

```
public float Hitrost = 500.0f;
1 reference
public float CasTrajanja = 10.0f;
2 references
private Rigidbody2D Telo;
0 references
private void Awake()
{
    Telo = GetComponent<Rigidbody2D>();
}
```

Slika 19: Lastnosti metka

Prva lastnost na Slika 19 mu določa, s kakšno hitrostjo bo potoval po igralnem prostoru, druga lastnost pa mu določa po kolikšnem času bo metek izginil. Tretja spremenljivka Telo, pa predstavlja spremenljivko, ki jo lahko uporabimo v nadaljevanju in ki je povezana z objektom ustvarjenim v Unityu.

Na Slika 20, pa sta predstavljeni še ostali 2 metodi, in sicer metoda Strel(), ki je uporabljena v skripti igralec, katera metku na podlagi zgornjih vrednosti, določi njegovo hitrost in po kolikšnem času bo izginil. Metoda OnCollisionEnter2D(), pa poskrbi za to, da če metek v karkoli trči takoj izgine.

```
public void Strel(Vector2 Smer)
{
    Telo.AddForce(Smer * this.Hitrost); //hitrost metka
    Destroy(this.gameObject, this.CasTrajanja); //metek izgine glede na casTrajanja
}
0 references
private void OnCollisionEnter2D(Collision2D collision)
{
    Destroy(this.gameObject); //metek izgine ko nekaj zadane
}
```

Slika 20: Metodi za metek

2.3.4 ASTEROID

Naslednji ustvarjen objekt je asteroid, ki se ga mora igralec izogibati in katerega lahko igralec s streljanjem tudi uniči. Za njegovo delovanje vsebuje 3 glavne metode.

Prva je metoda `DolociPot()`, v kateri podobno kot pri metku, objektu določimo kako hiter bo ko se bo prikazal v prostor in koliko časa bo v prostoru.

Druga je metoda `OnCollisionEnter2D()` (Slika 21), katera skrbi za to, da če metek in asteroid trčita, se asteroid uniči in izvede se metoda `AsteroidUnicen()`, katero bom razložil v nadaljevanju. Prisoten pa je še pogoj, da če je asteroid manjši kot 0.3f se izvede metoda `Razpolovi()`.

```
private void OnCollisionEnter2D(Collision2D collision)
{
    if(collision.gameObject.tag == "Metek") //ce se dotakne metka
    {
        if((this.Velikost * 0.5f) >= 0.3f) //ce je velikost vecja od 0.3f se razpolovi drugace ne
        {
            Razpolovi(); //2 nova asteroida
            Razpolovi();
        }
        FindObjectOfType<GameManager>().AsteroidUnicen(this);
        Destroy(this.gameObject);
    }
}
```

Slika 21: Preverjanje trkov

Metoda `Razpolovi()` (Slika 22), pa je metoda, ki v prostor prikaže nov asteroid, ki pa je za polovico manjši od prejšnjega (uničenega) in kateremu se spremeni pot. V metodi `OnCollisionEnter2D()`, je ta metoda uporabljena 2 krat, saj želimo, da se prikažeta 2 manjša asteroida.

```
private void Razpolovi()
{
    Vector2 Smer = this.transform.position;
    Smer += Random.insideUnitCircle * 0.5f; //nova smer objekta

    Asteroid pol = Instantiate(this, Smer, this.transform.rotation);
    pol.Velikost = this.Velikost * 0.5f; //nova velikost 1/2

    pol.DolociPot(Random.insideUnitCircle.normalized * this.Hitrost); //nova pot objekta
}
```

Slika 22: Metoda razpolovi

2.3.5 GENERATOR ASTEROIDOV

Da pa ima asteroid sploh smisel, ga je seveda potrebno nekako prikazati v igralni prostor. To pa dosežemo s pomočjo metode `Generiraj()`, ki se nahaja v skripti `Generator Asteroidov`.

```
private void Generiraj()
{
    for(int i = 0; i < this.spawnKolicina; i++) //generira asteroide
    {
        //kje se generira
        Vector3 spawnDirection = Random.insideUnitCircle.normalized * this.spawnOddaljenost;
        Vector3 spawnPoint = this.transform.position + spawnDirection;

        //kam se usmeri
        float usmerjenost = Random.Range(-this.rndUsmerjenost, this.rndUsmerjenost); //naključno od -15 do 15 stopinj
        Quaternion rotacija = Quaternion.AngleAxis(usmerjenost, Vector3.forward);

        Asteroid asteroid = Instantiate(this.asteriodPrefab, spawnPoint, rotacija); //ga generira
        asteroid.Velikost = Random.Range(asteroid.minVelikost, asteroid.maxVelikost); // mu določi naključno velikost
        asteroid.DolociPot(rotacija * -spawnDirection);
    }
}
```

Slika 23: Generiranje asteroidov

Na Slika 23 je prikazana metoda `Generiraj()`, katera ves čas skozi igro generira asteroide. Asteroidi se generirajo na naključni poziciji okoli igralnega prostora, določi jim naključno usmerjenost in pot po kateri bodo prečkali igralni prostor in vsakemu posebej določi naključno velikost.

2.3.6 GAME MANAGER

Game manager pa je objekt v Unityu, za katerega lahko rečemo, da je najpomembnejši, saj povezuje vse ključne elemente igre in skrbi da igre teče tako kot mora. V skripti za ta objekt je kar precej metod in kode, opisal pa bom le nekaj glavnih, brez katerih igra ne more potekati.

```
void Start(){

    tocke = 0; //tocke nastavi na 0

    novNajvisjiRezultat.SetActive(false); //screen odstrani
    konecIgre.SetActive(false); // konec screen odstrani

    TockeText.text = "Točke: " + tocke; //izpise se tocke 0
    zivljenjaText.text = "Življenja: " + zivljenje; //izpise se zaceto stevilo zivljenj

    for (int i = 0; i < zivljenja.Length; i++) //ce je i manjsi od stevila zivljenj kot ga ima igralec
    {
        if(i < zivljenje){
            zivljenja[i].sprite = polnoZ; //se pojavi slikza polnega življenja
        }else{
            zivljenja[i].sprite = none; //prikaze prazno sliko
        }
    }
}
```

Slika 24: Začetek igre

Na Slika 24 je prikazana metoda, ki praktično začne igro. Resetira točke igralca, zakrije vsa okna, ki prekrivajo igro in na ekran izpiše začetno število igralčevih življenj in točk.

Naslednja metoda je na Slika 25, in sicer to je `IgralecMrtev()`, katera pa predvsem skrbi za življenja igralca. Če se izvede metoda `igravec mrtev`, se igralcu odšteje 1 življenje in novo število življenj prikaže na ekran. Če pa igralec nima več življenj je konec igre in se izvede metoda `KonecIgre()`.

```
public void IgralecMrtev(Igralec igravec)
{
    this.eksplozijaIgralec.transform.position = this.igravec.transform.position;
    Instantiate(eksplozijaIgralec); //predvaja eksplozija animation

    this.zivljenje--; //zmanjsa zivljenje

    zivljenjaText.text = "Življenja: " + zivljenje; //prikaze zivljenja na ekranu

    if(this.zivljenje <= 0) //ce je zivljenje 0 je konec igre
    {
        KonecIgre();
    }else{
        Invoke(nameof(Respawn), this.respawnTime); //ce ne te respawn v dolocenem casu
    }
}
```

Slika 25: Število življenj

Naslednja metoda pa je metoda `AsteroidUnicen()`, katera skrbi za točkovanje. Kot lahko vidimo na sliki 27, so možne točke 25, 50, 100 in 500. Število točk, ki se prišteje igralcu, je odvisno od velikosti asteroida, ki ga igralec uniči. Točke se sproti seštevajo in se prikazujejo na ekran.

```
public void AsteroidUnicen(Asteroid asteroid)
{
    if(asteroid.Velikost < 0.3f){
        this.tocke = tocke + 500;
    }else if(asteroid.Velikost <= 0.5){
        this.tocke = tocke + 100;
    }else if(asteroid.Velikost <= 0.8){
        this.tocke = tocke + 50;
    }else
        this.tocke = tocke + 25;

    TockeText.text = "Točke: " + tocke; //prikaze rezultat na ekranu
}
```

Slika 26: Točkovanje

In še zadnja metoda, prikazana na Slika 27, brez katere seveda ne gre, saj se igra nebi nikoli končala, je metoda `KonecIgre()`. Ko se izvede ta metoda, se najprej prekinajo vse interakcije z igro. Potem preveri, če je presežen najboljši rezultat, in če je bil ti prikaže okno v katero lahko zapišeš svoje ime in se skupaj s tvojim rezultatom shrani. Če ne, se pojavi končno okno, na katerem se izpišejo dosežene točke in najvišji rezultat. Igralec pa ima možnost ponovnega igranja ali pa izhoda na začetni meni.

```
private void KonecIgre()
{
    CancelInvoke(); // prekine vse prilkllice

    if(PoisciNajvisjiRezultat(tocke)){
        novNajvisjiRezultat.SetActive(true);
    }else{
        konecIgre.SetActive(true); // ce ne se prikaze end ekran
        TvojRezultatText.text = "" + tocke; //izpise trenutne tocke
        NajbolshiTekst.text = PlayerPrefs.GetInt("tocke_najbolsega") + " - " +
        PlayerPrefs.GetString("ime_najbolsega");
    }
}
```

Slika 27: Konec igre

2.3.7 MENI

Meni je eden pomembnejših elementov igre, saj ko uporabnik prižge igro, se mu najprej odpre začetni meni na katerem so štirje gumbi, in sicer igray, pomoč, nastavitve in izhod. Če klikneš na gumb igray, te seveda takoj vrže v igro, oziroma če klikneš izhod, igro zapre. Če klikneš na gumb pomoč, si lahko tam prebereš pravila igre in tipke. Če pa klikneš na nastavitve, si lahko kot uporabnik nastaviš poljubno resolucijo igre, ali je čez cel zaslon ali ne, kakšna je kakovost igre in kolikšna je glasnost glasbe, ki se vrti v ozadju. Vsak gumb ima svojo metodo, ki se izvrši ko je gumb pritisnjen. Na spodnji Slika 28 je primer za gumba igray in izhod.

```
public void Igraj()
{
    SceneManager.LoadScene(SceneManager.GetActiveScene().buildIndex + 1);
}
0 references
public void Izhod()
{
    Application.Quit(); //ko klikneš izhod zapre igro
}
```

Slika 28: Metode gumbov

Na spodnji Slika 29 je prikazan del kode, ki omogoča uporabniku v nastavitvah spreminjanje resolucij in celozaslonskega načina. Celozaslonski način je narejen na način, da ga lahko uporabnik vklopi ali izklopi. Za resolucije pa se odpre tabela, kjer si lahko izbere poljubno resolucijo.

```
public void CelZaslon (bool jeCelZaslon)
{
    Screen.fullScreen = jeCelZaslon;
}
4 references
public TMPPro.TMP_Dropdown oknoResolucij;
7 references
Resolution[] resolucije;
0 references
public void NastaviResolucijo(int index)
{
    Resolution resolucija = resolucije[index];
    Screen.SetResolution(resolucija.width, resolucija.height, Screen.fullScreen);
}
```

Slika 29: Nastavitve resolucije

V nastavitvah, pa si je mogoče nastaviti še glasnost in kakovost igre, kar pa je prikazano na spodnji Slika 30. Metodi za to sta `VklopiGlas()` in `NastaviKakovost()`.

```
public void VklopiGlas(bool mute)
{
    if(mute)
    {
        glasba.volume = 0; //ce zvočnik prekrizan glasba 0
        slider.value = 0f; // in slider na 0
    }else
    {
        glasba.volume = 1; //ce ni glasba na 1
        slider.value = 0.5f; // in slider da na 0.5
    }
}
0 references
public void NastaviKakovost(int index)
{
    QualitySettings.SetQualityLevel(index);
}
```

Slika 30: Nastavitve glasnosti

3 ZAKLJUČEK

Pri izdelavi seminarske naloge, sem se naučil veliko novih stvari, katere pa se navezujejo predvsem na izdelavo iger. Izvedel sem, kako v večjih podjetjih od samega začetka poteka razvoj in iger, pa vse do konca, ko igro izdajo v javnost. Še bolj pa sem izpopolnil tudi svoje znanje na področju programiranja in pa uporabe programskega pogona Unity, s pomočjo katerega sem ustvaril vse najpomembnejše elemente igre in tako igro uresničil. Na začetku izdelave igre, sem si naredil dober načrt in si zadal jasne cilje kaj vse želim, da bi moja igra vsebovala, a vse ni šlo čisto kot načrtovano. Med samim delom je seveda prišlo do mnogih manjših težav in sprememb, katere pa sem z več ali manj truda popravil in razrešil. Z končnim izdelkom sem zelo zadovoljen, saj sem izpolnil vse zadane cilje, saj igra vsebuje vse elemente, ki so bili načrtovani in deluje brezhibno. Seveda, je tu še veliko prostora za izboljšave, za katere pa bi si moral vzeti malce več časa, saj če bi želel igro kam objaviti ali celo prodati, bo morala vsebovati še veliko elementov, za katere pa si je treba vzeti kar precej časa, da delujejo brezhibno.

4 ZAHVALA

Na koncu, pa bi se rad zahvalil vsem, ki so kakorkoli pripomogli pri izdelavi te seminarske naloge. Najprej gre zahvala profesorjema in mentorjema dr. Albertu Zorku univ. dipl. inž. el. in Gregorju Medetu univ. dipl. inž. rač. in inf., ki sta nas slozi vsa 4 leta učila računalništva in nam tako vzbudila željo in zanimanje po programiranju, hkrati pa sta bila vedno na voljo za kakršnokoli vprašanje. Nato pa bi se zahvalil še svoji družini, ki me je skozi vsa 4 leta šolanja podpirala in mi s tem dajala motivacijo za nadaljno uspešno delo.

5 VIRI IN LITERATURA

1. **Deepali. naukri.** *A Closer Look at Career in Game Development.* [Elektronski] 24. 1 2022. [Navedeno: 26. 3 2022.] <https://www.naukri.com/learning/what-is-game-development-st559-tg295#description>.
2. **Pickell, Devin. g2.** *The 7 Stages of Game Development.* [Elektronski] 15. 10 2019. [Navedeno: 26. 3 2022.] <https://www.g2.com/articles/stages-of-game-development>.
3. **Mozolevskaya, Victoria. kevruru games.** *6 KEY STAGES OF GAME DEVELOPMENT: FROM CONCEPT TO STANDING OVATION.* [Elektronski] 8. 2 2021. [Navedeno: 26. 3 2022.] <https://kevrurugames.com/blog/6-key-stages-of-game-development-from-concept-to-standing-ovation/>.
4. **Wikipedia.** *Video game development.* [Elektronski] 19. 3 2022. [Navedeno: 26. 3 2022.] https://en.wikipedia.org/wiki/Video_game_development.
5. **Nuclino.** *Video Game Development Process.* [Elektronski] 17. 2 2022. [Navedeno: 26. 3 2022.] <https://www.nuclino.com/articles/video-game-development-process>.
6. **Tyler, Dustin. GameDesigning.** *How to Choose the Best Video Game Engine.* [Elektronski] 9. 3 2022. [Navedeno: 26. 3 2022.] <https://www.gamedesigning.org/career/video-game-engines/>.
7. **Walker, Alyssa. Guru99.** *What is Photoshop? Introduction, Meaning, Definition & History.* [Elektronski] 12. 3 2022. [Navedeno: 27. 3 2022.] <https://www.guru99.com/introduction-to-photoshop-cc.html>.
8. **Mustafeez, Anusheh Zohair. educative.** *What is Visual Studio Code?* [Elektronski] 15. 1 2022. [Navedeno: 27. 3 2022.] <https://www.educative.io/edpresso/what-is-visual-studio-code>.
9. **Schardon, Lindsay. GameDev Academy.** *What is Unity? – A Guide for One of the Top Game Engines.* [Elektronski] 22. 2 2022. [Navedeno: 2. 4 2022.] https://gamedevacademy.org/what-is-unity/#What_is_Unity.
10. **Sinicki, Adam. Android Authority.** *What is Unity? Everything you need to know.* [Elektronski] 20. 3 2021. [Navedeno: 2. 4 2022.] <https://www.androidauthority.com/what-is-unity-1131558/>.

11. **Unity Documentation.** *Unity Manual.* [Elektronski] 28. 3 2022. [Navedeno: 2. 4 2022.] <https://docs.unity3d.com/Manual/ProjectView.html>.

12. **freeCodeCamp.** *What Is Game Development?* [Elektronski] 28. 12 2019. [Navedeno: 26. 3 2022.] <https://www.freecodecamp.org/news/what-is-game-development/>.

13. **Wikipedia.** *Unity (game engine).* [Elektronski] 1. 4 2022. [Navedeno: 2. 4 2022.] [https://en.wikipedia.org/wiki/Unity_\(game_engine\)](https://en.wikipedia.org/wiki/Unity_(game_engine)).

6 STVARNO KAZALO

cilj, 1, 9	metoda, 6, 15, 16, 17, 18, 19, 20, 21
elementov, 4, 5, 8, 10, 14, 23	načrtovanje, 3, 8
faz, 2, 5	objekt, 10, 14, 15, 16, 17, 18, 19
iger, 3, 1, 2, 3, 7, 8, 9, 13, 23	objekta, 12
igralca, 6, 5, 14, 16, 20	programskih orodji, 8
igralni pogon, 3, 1, 8	projekt, 10
izdelava iger, 2	razvoja igre, 4, 5, 7
kodo, 14, 15	rezultatov, 8
lastnost, 17	Unity, 3, 6, 1, 8, 9, 10, 11, 13, 14, 16, 23,
lastnosti, 10, 12, 17	25, 26
meni, 11, 14, 21	uporabnika, 6, 1, 15

7 PRILOGE

Priloga 1 – Povezava do igre

https://github.com/NCekuta/Prezivetje_v_vesolju/tree/main/Igra

Priloga 2 – Povezava do Word dokumenta

https://github.com/NCekuta/Prezivetje_v_vesolju/tree/main/Word%20dokument

Priloga 3 – Povezava do PowerPoint dokumenta

https://github.com/NCekuta/Prezivetje_v_vesolju/tree/main/PowerPoint