



UNIVERZITET U NOVOM SADU
FAKULTET TEHNIČKIH NAUKA U
NOVOM SADU



Matija Maksimović RA/132/2020

Analiza razvoja 2D igara pomoću biblioteke pygame

SEMINARSKI RAD
- Osnovne akademske studije -

Novi Sad, 2023.



UNIVERZITET U NOVOM SADU • FAKULTET TEHNIČKIH NAUKA
21000 NOVI SAD, Trg Dositeja Obradovića 6

Predmet: Računarska grafika
Predmetni profesor: prof. dr Dragan Ivetić
Predmetni assistant: dipl.ing. Nedeljko Tešanović

SADRŽAJ

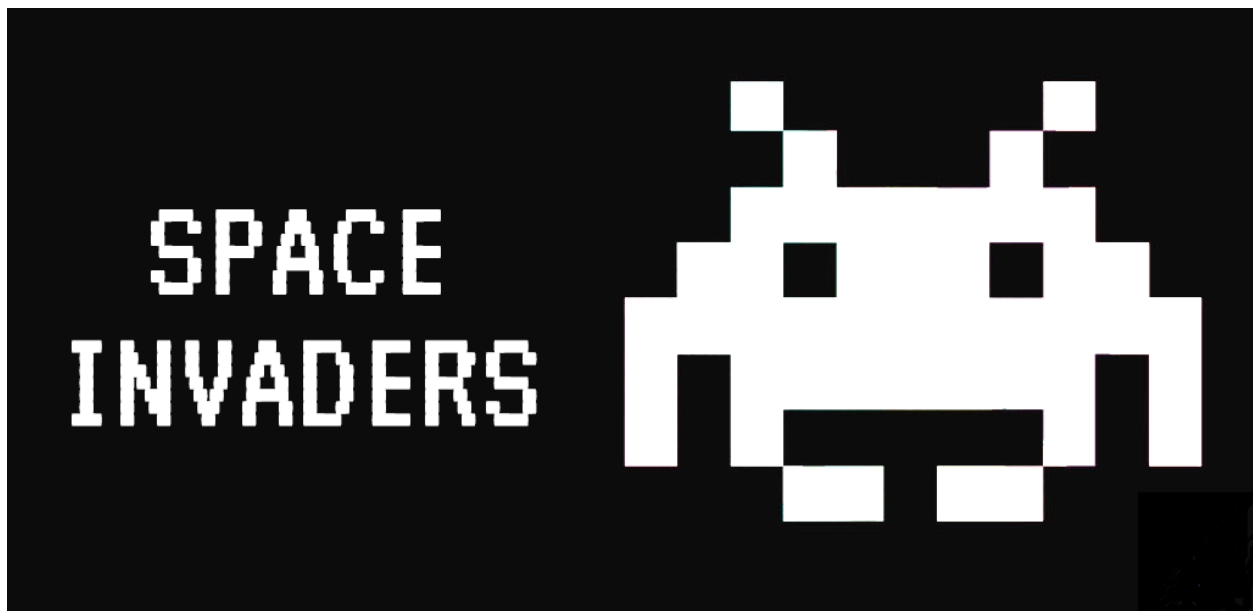
1. Uvod	
.....	2
2. Istorija 2D igara	
.....	3
3. Pygame	
.....	6
4. Game loop	
.....	7
5. Kontrole i multimedija	
.....	9
6. Iscrtavanje objekata i kolizije	
.....	12
7. UE, parametri igre i bodovanje	
.....	16
8. Easter eggs	
.....	18
9. Osvrt na 3D grafiku	
.....	19
10. Literatura	
.....	23

1. UVOD

Svet video igara, kao oblik zabave i umetnosti, kontinuirano se razvija, donoseći nove izazove i mogućnosti za developere i igrače. U tom razvoju, 2D grafika igra ključnu ulogu, pružajući jedinstveno iskustvo koje se oslanja na jednostavne, ali vizualno impresivne elemente. Ovo podseća na zlatno doba video igara, gde je igranje bilo u središtu pažnje, a svaki korak igrača predstavljao je nagradu samu po sebi. Ovaj rad istražuje svet 2D grafike kroz razvoj igre inspirisane klasikom "Napadači iz svemira" (Alien Invader) uz korišćenje Pygame biblioteke.

2D grafika ima sposobnost da nas prenese u fantastične svetove i probudi našu maštu. Ona nam omogućava da se vratimo temeljima video igara, gde je igračko iskustvo bilo neprikosnoveno, a svaka pobeda predstavljala je lični trijumf. Upravo u tom duhu je stvorena moja igra, kao omaž klasicima, ali koristeći alate i tehnike savremenog razvoja igara.

Ključna uloga u razvoju igre pripada Pygame biblioteci. Pygame, kao platformski nezavisan set Python modula, omogućava razvoj igara u 2D grafici i pruža bogat okvir za implementaciju raznih aspekata igre. Kroz ovaj rad, istražujemo kako koristiti Pygame za razvoj 2D igre, posmatrajući mehaniku igre, kreativni proces, izazove i lekcije naučene tokom razvoja.



2. ISTORIJA 2D IGARA

Istorija 2D igara započinje sedamdesetih godina s pojavom prvih arkadnih igara i kućnih konzola. Verovatno ste videli jednu od njih - Table Tennis za Magnavox Odyssey i Pong za Atari konzole. Ekran je bio podeljen na dva dela, a igrači su pomerali svoje platforme pokušavajući da pogode lopticu i postignu najbolji rezultat. Mreža je izgledala kao ravna linija, a loptica je bila samo tačka. Zapravo, sve igre tog ranog doba zahtevale su dosta maštovitosti da bi se u potpunosti razumele.

Sledeća revolucionarna igra nakon tenisa bila je Space Invaders, koja je počela kao arkadna igra 1978. godine. Prvi put viđena u Japanu, Space Invaders se smatra najuticajnijom igrom u istoriji video igara. Igrač mora ubiti svakog neprijatelja na ekranu, koristeći samo jedan metak po potezu. Igra se ubrzava kako igrač napreduje, čineći je sve težom za pobedu. Arkade su popularizovale same igre, jer su bile najpristupačniji način igranja od kraja sedamdesetih do početka osamdesetih. Sve što nam je bilo potrebno bilo je preduzeće koje bi moglo doneti kvalitet arkadnih igara kućnim konzolama.

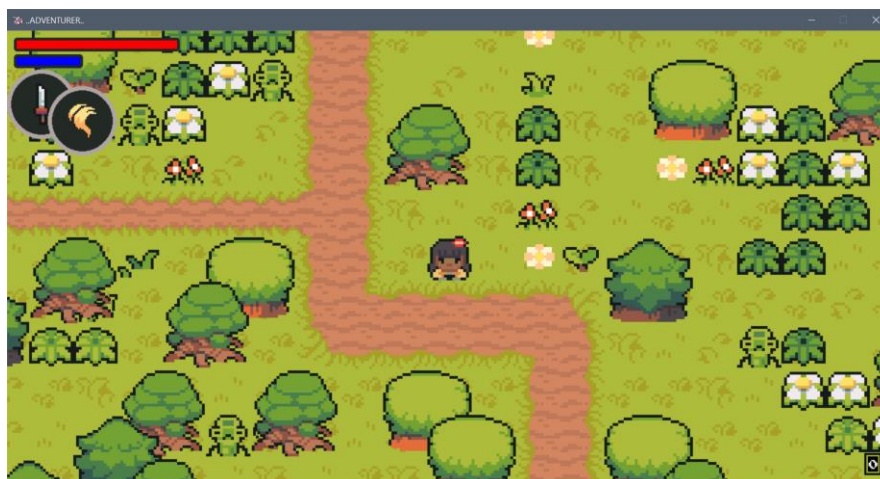


1985. godine, Nintendo je izdao svoju ikoničnu konzolu - Nintendo Entertainment System - sa najpopularnijim igrima svih vremena kao što su Super Mario Bros., The Legend of Zelda i Metroid. Narednu deceniju, 2D igre su dominirale svetom igara. Početkom devedesetih godina, četvrta generacija konzola je stigla - Super Nintendo i Sega Genesis su bili vladari sveta 2D igara. Krajem devedesetih i početkom 2000-ih, 2D igre su percipirane kao korak ka 3D igrima i bile su gotovo izumrle. Sve najcenjenije franšize video igara prešle su na 3D. 3D je dominirao na kućnim konzolama, zamenjujući 2D igre na

prenosnim uređajima. Da, 2D igre se nisu potpuno povukle, ali kako su se premestile na prenosne uređaje, činile su se manje važnim. Prenosne verzije popularnih franšiza uvek su bile tretirane kao sporedni naslovi, dok su konzolne igre uglavnom bile direktne nastavke jedna na drugu. Zato 2D igre na prenosnim uređajima nisu mogle konkurisati sa velikim konzolnim igrama.



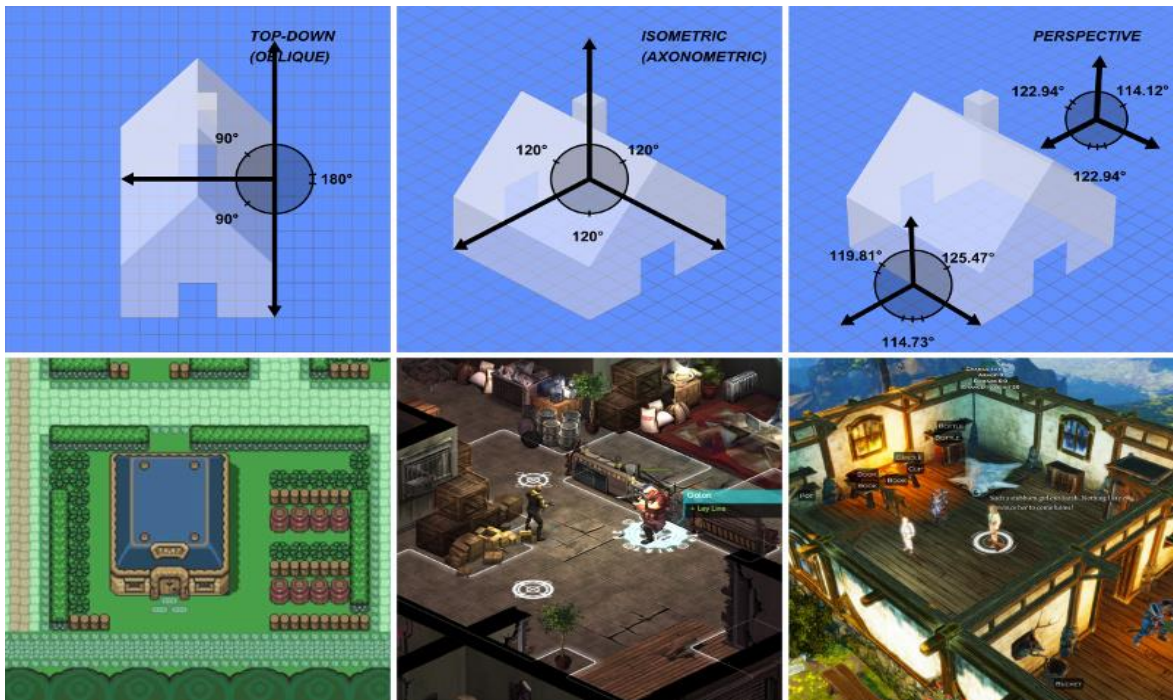
Međutim, uskoro su 2D igre doživele masovni povratak. Njihova obnova počela je sa početkom Xbox Live Arcade na originalnom Xbox-u 2001. godine. Ova platforma je nudila igre kao što su arkadne klasike i neke jednostavne igre. Xbox Live Arcade je postao mesto za igre malih razmera i stvorio je prostor za 2D igre retro stila na kućnim konzolama. Sada su 2D naslovi mogli koegzistirati sa visokobudžetnim 3D igrama.



Dvodimenzionalne igre se mogu razlikovati po stilu umetnosti, žanru, perspektivi i platformi. Žanrovi video igara su gotovo isti za obe 2D i 3D igre. To su igre uloga, platformeri, akcione igre, borbe, simulacije, pucačine, slagalice itd.

Ali ključna razlika između 2D i 3D igara je izbor perspektive. Evo najčešćih tipova 2D igara zasnovanih na njihovoj perspektivi:

- **Pogled sa strane:** To je jedan od najpopularnijih tipova 2D igara. Pogled sa strane je klasičan izbor za platformere, gde se likovi kreću uglavnom levo-desno, gore-dole.
- **Pogled odozgo:** Igra izgleda kao da je kamera iznad glave. Igračko polje se vidi kao iz ptičje perspektive ili ima blago naginjanje kamere.
- **Izometrijska perspektiva:** Igra se prikazuje iz određenog ugla kamere. Savršena je za stvaranje iluzije 3D prostora, prikazujući tri strane objekta.
- **Jednostrani ekran:** Svaki nivo se postavlja u novoj sobi koja ispunjava ceo ekran. Kada se nivo završi, prelazi se na sledeću sobu.



3. PYGAME

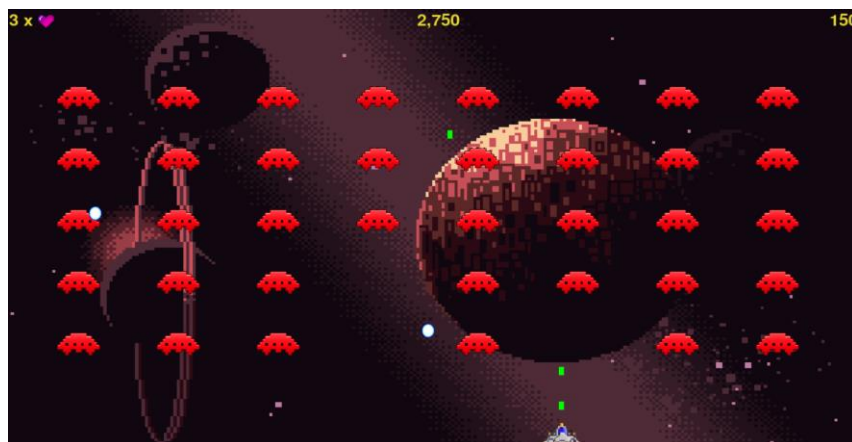
Pygame koristi biblioteku Simple DirectMedia Layer (SDL) s ciljem omogućavanja razvoja računarskih igara u stvarnom vremenu bez potrebe za mehanizmima niskog nivoa (programski jezik C i njegovi derivati). Osnova pretpostavka jest da se najzahtevnije funkcije unutar igara mogu apstrahovati od same logike igre, čime postaje moguće koristiti jezike visokog nivoa, jezike poput Pythona za strukturiranje igre.

SDL takođe nudi niz drugih mogućnosti, uključujući vektorsku matematiku, detekciju sudara, upravljanje 2D sprite scenama, podršku za MIDI (Musical Instrument Digital Interface), kameru, pixel-array manipulaciju, transformacije, filtriranje, naprednu podršku za freetype fontove i crtanje.

Aplikacije koje koriste Pygame mogu se pokretati na Android telefonima i tabletima uz pomoć Pygame Subset for Android (pgs4a). Na Androidu se podržava zvuk, vibracije, tastatura i akcelerometar.



Postupak formiranja igre uz pygame analiziraćemo kroz projekat koji možete pronaći na ovom linku: <https://github.com/MatijaMax/alien-invasion-game>



4. GAME LOOP

Game loop (petlja igre) je ključna komponenta u razvoju video igara. To je beskonačna petlja koja neprestano izvršava određene korake kako bi igra funkcionalno radila. Osnovni elementi game loop-a uključuju:

- **Ulaz (Input):** U svakoj iteraciji petlje, igra proverava korisnički ulaz, kao što su pritisci na tastere, klikovi miša i slično.
- **Logika (Logic):** Nakon što se prikupe ulazni podaci, igra izvršava svoju logiku. To uključuje računanje fizike, upravljanje protivnicima, reagovanje na događaje itd.
- **Renderovanje (Rendering):** Nakon što se ažurira igraća logika, igra crta ili renderuje novi okvir na ekranu. To uključuje iscrtavanje svih objekata, karaktera i pozadine kako bi se prikazao trenutni stanje igre.
- **Ograničenje brzine (Frame Rate Limiting):** Game loop često uključuje i ograničenje brzine igre na određeni broj frejmova u sekundi (FPS) kako bi se održala konstantna brzina igre.
- **Provera uslova za završetak igre (Game Over Conditions):** Game loop takođe proverava uslove za završetak igre, kao što su pobeda, gubitak ili napuštanje igre.

```
Game loop

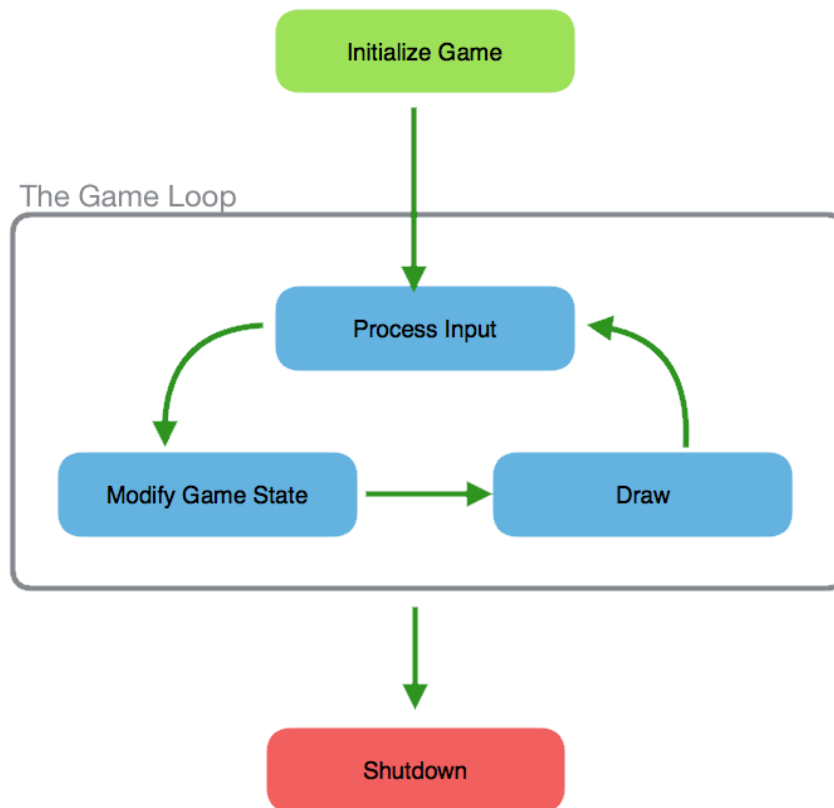
def run_game(self):
    """Start the main loop for the game."""
    while True:
        # Pazi na događaje tastature i miša
        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                sys.exit()

        # Nacrtaj ekran ponovi pri svakom prolazu kroz petlju
        self.screen.fill(self.settings.bg_color)

        # Nakon što je ekran popunjen bojom osveži ga prikazivanjem nove slike igre
        pygame.display.flip()
        #60 frejmova po sekundi (60 osvežavanja)
        self.clock.tick(60)
```

Ova petlja se ponavlja neprestano tokom izvršavanja igre, čime se omogućava korisnicima da interagiraju sa igrom, a igra reaguje na te interakcije, ažurira svoje

stanje i prikazuje promene na ekranu. Game loop je osnova svih video igara i omogućava im da funkcionišu kao kontinuirani i interaktivni softver.



Ovo je init funkcija gde inicijalizujemo parametre ove igre. Za početak to će biti dimenzije ekrana i boja.

```
def __init__(self):
    """Initialize the game, and create game resources."""
    pygame.init()
    self.clock = pygame.time.Clock()

    self.settings = Settings()

    self.screen = pygame.display.set_mode((1200, 800))
    self.bg_color = (230, 230, 230)
    pygame.display.set_caption("Alien Invasion")
```

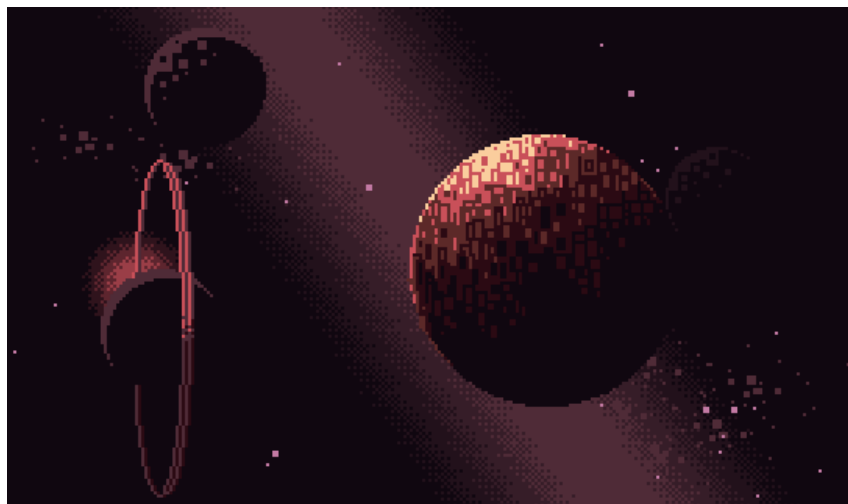
5. KONTROLE I MULTIMEDIJA

Multimediji igraju ključnu ulogu u svetu video igara i stvaranju bogatog i uzbudljivog igračkog iskustva. Za ovu igricu fokus će biti na sledećim aspektima:

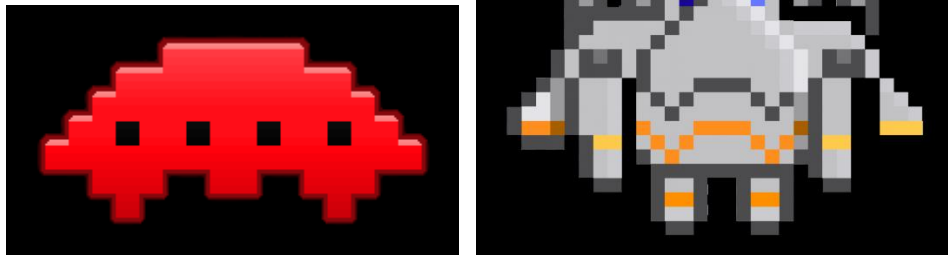
Vizuelni elementi: Grafika i animacije čine osnovu svake igre. Kvalitetna grafika, detalji i estetski dizajn igre privlače igrače, stvarajući privlačnu i uverljivu virtuelnu stvarnost. Vizualni elementi takođe pomažu u stvaranju emotivnih reakcija i atmosfere u igri. U ovoj igri, svaki objekat smatra se pravougaonikom (rect). Ovaj pravougaonik može postati kontejner neke slike i uključivanjem transparentnosti postići iluziju uklapanja sa pozadinom igre (hitbox objekta naravno biće pravougaonik, samo što mi kao igrači to nećemo videti, kao dizajner treba biti pažljiv sa ovom činjenicom):

```
Game loop

self.image = pygame.image.load('images/broship.png')
# Skaliramo našu sliku na odgovarajući pravougaonik
self.image = pygame.transform.scale(self.image, (60, 48))
# Nameštamo transparentnos crnih piksela, ovo je specifično za ovu sliku
self.image.set_colorkey((0, 0, 0)) # Set black color as transparent
# self.make_black_pixels_transparent() # Make black pixels transparent
self.rect = self.image.get_rect()
# Alternativa ovom pristupu je alfa parametar (RGBA torka) koji pygame podržava
```



Invasion background



Alien i spaceship

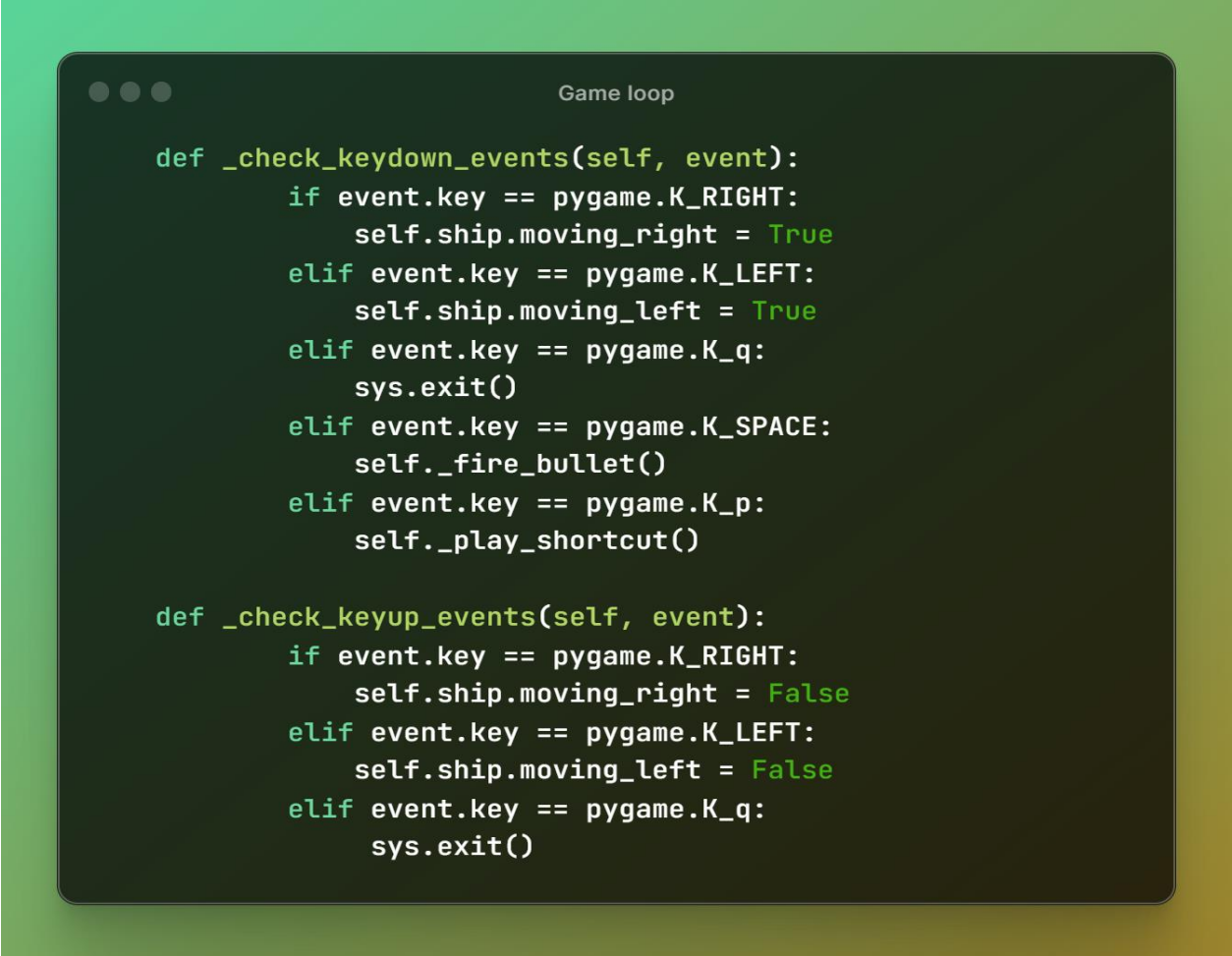
Zvuk i muziku: Zvuk i muzika su ključni za stvaranje atmosfere i emotivnog angažovanja igrača. Pravilno postavljeni zvučni efekti prate radnju igre, pomažu u prepoznavanju događaja i često doprinose uzbuđenju i napetosti. Muzika dodatno pojačava igračko iskustvo, usmerava emocije i često postaje prepoznatljiv deo igre.

```
Game loop

self.ship_hit = pygame.mixer.Sound("audio/shiphit.mp3")

def _ship_hit(self):
    if self.stats.ships_left > 0:
        self.stats.ships_left -= 1
        ### Izuzetno je važan redosled poziva funkcija
        self.ship_hit.play()
        ### Poziv mora logički da prati crtanje na ekranu
        time.sleep(1)
        self.bullets.empty()
        self.aliens.empty()
        self._create_fleet()
        self.ship.center_ship()
        self.wave_sound.play()
        self.sb.prep_hearts()
        time.sleep(1)
    else:
        self.game_active = False
        pygame.mouse.set_visible(True)
```

KeyPress događaj (event) u Pygame biblioteci ima veliku važnost jer omogućava igračima da interaguju sa igrom putem tastature. KeyDown i KeyUp eventovi utilizirani su u sledećem kodu:



```
def _check_keydown_events(self, event):
    if event.key == pygame.K_RIGHT:
        self.ship.moving_right = True
    elif event.key == pygame.K_LEFT:
        self.ship.moving_left = True
    elif event.key == pygame.K_q:
        sys.exit()
    elif event.key == pygame.K_SPACE:
        self._fire_bullet()
    elif event.key == pygame.K_p:
        self._play_shortcut()

def _check_keyup_events(self, event):
    if event.key == pygame.K_RIGHT:
        self.ship.moving_right = False
    elif event.key == pygame.K_LEFT:
        self.ship.moving_left = False
    elif event.key == pygame.K_q:
        sys.exit()
```

Event se mapira na promenu vrednosti nekog važnog parametra, pokretanje funkcije iscrtavanja ili pak izlazak iz same igre i prekidanju game petlje. Za igru ključni su: K_RIGHT, K_LEFT i K_SPACE (kretanje desno, kretanje levo i pucanje).

6. ISCRTAVANJE OBJEKATA

Pygame podržava različite vrste objekata, uključujući pravougaonike, krugove, linije i slike. Za crtanje pravougaonika koristimo funkciju `pygame.draw.rect()`. Ovo su osnovni koraci za crtanje pravougaonika na prozoru:

```
# Definišemo boju pravougaonika (u formatu RGB)
color = (255, 0, 0) # Crvena boja

# Definišemo koordinate gornjeg levog i donjeg desnog ugla pravougaonika
rect_x = 100
rect_y = 100
rect_width = 200
rect_height = 100

# Crtamo pravougaonik na ekranu
pygame.draw.rect(screen, color, (rect_x, rect_y, rect_width, rect_height))
```

Važni objekti koji se crtaju na ekranu jesu vanzemaljci, igrač i projektili koje oni mogu ispaljivati. Za svaki objekat koji treba da se nacrtava imamo posebnu klasu sa ključnom metodom `.blitme()` koja daje objektu iscrta na ekranu tokom game petlje. Uz `blitme` mapiramo sliku našeg objekta na pravougaonik, koji je suštinski njegov hitbox:

```
def blitme(self):
    self.screen.blit(self.image, self.rect)
```

Za svemirski brod (igrača) imamo vezane osnovne kontrole kretanja i pucanja. Pošto je u pitanju 2D igra, kretanje se opisuje uz pomoć x i y ose. Važno je napomenuti da pygame stavlja koordinatni početak u gornji levi ugao ekrana i ose se moraju pružati u skladu sa tim (suština je u tome je da ako želimo da naš igrač bude na dnu ekrana, naša y koordinata mora biti jednaka visini samog ekrana). Pomeranja ostvarujemo

inkrementom ili dekrementom (zavisno od eventa) x koordinate igrača. Uslov jesu boolean koji nam je dao KeyDown event i ograničenje ekrana (brod logično ne sme izaći van ekrana).

```
*** SHIP ***

def update(self):
    if self.moving_right and self.rect.right < self.screen_rect.right:
        self.rect.x += self.movement_speed
    if self.moving_left and self.rect.left > 0:
        self.rect.x -= self.movement_speed
```

Ispaljivanje metaka je funkcionalnost koja pripada i igraču ali i neprijateljima. Analiziraćemo iscrtavanje metaka igrača, x koordinata preuzeta je u konstruktoru metka, y se mora menjati:

```
*** SHIP ***

#kretanje smanjivanjem y koordinate jer krećemo od maksimuma y tojest našeg igrača
def update(self):
    self.y -= self.settings.bullet_speed
    self.rect.y = self.y

def _fire_bullet(self):
    if len(self.bullets) < self.settings.bullets_allowed:
        new_bullet = Bullet(self)
        self.bullets.add(new_bullet)
        if self.game_active:
            new_bullet.bullet_sound.play()
```

Da bismo iscrtili flotu potrebno je popuniti gornji deo ekrana neprijateljima, pratimo komentare u sledećem kodu:

```
*** ALIEN ***

#pomeranje na isti način koji smo videli ranije u kodu, ali sada uvodimo pojam flote,
imamo više objekata u floti, koji moraju poštovati njena pravila pomeranja
def update(self):
    self.x += self.settings.alien_speed * self.settings.fleet_direction
    self.rect.x = self.x
    #ovu su easter egg funkcionalnosti
    if(self.sans):
        if (random.randint(1, 100) == 1):
            self.fire_plasma()
    else:
        if (random.randint(1, 1600) == 1):
            self.fire_plasma()

#Ova petlja popunjava flotu, vanzemaljci kao objekti moraju biti na jasno definisanoj
udaljenosti jedni od drugih, flota se dakle popunjava po !definisanim granicama!
def _create_fleet(self):
    alien = Alien(self)
    alien_width, alien_height = alien.rect.size
    current_x, current_y = alien_width, alien_height
    while current_y < (self.settings.screen_height - 4 * alien_height):
        while current_x < (self.settings.screen_width - 2 * alien_width):
            self._create_alien(current_x, current_y)
            current_x += 2 * alien_width
        current_x = alien_width
        current_y += 2 * alien_height

#Ako flota udari u granicu ekrana, ona mora promeniti smer kretanja i spustiti se dole
za neku predefinisanu vrednost
def _check_fleet_edges(self):
    for alien in self.aliens.sprites():
        if alien.check_edges():
            self._change_fleet_direction()
            break

def _change_fleet_direction(self):
    for alien in self.aliens.sprites():
        alien.rect.y += self.settings.fleet_drop_speed
    self.settings.fleet_direction *= -1
```

Ključni deo ovog koda je `.sprites()` funkcija. Ako naš `rect` objekat pretvorimo u `Sprite` objekat, imamo mogućnost grupisanja (primer flote) ali i nečeg mnogo važnijeg, kolizije objekata.

Želimo da znamo kada je metak pogodio vanzemaljca i nakon toga ga uklonimo sa ekrana. Funkcija `sprite.groupcollide()` upoređuje objekte `rect` svakog elementa u jednoj grupi sa objektom `rect` svakog elementa u drugoj grupi. Ona uporedi `rect` svakog metka sa objektom `rect` svakog vanzemaljca i vrati rečnik koji sadrži metke i vanzemaljce koji su se sudarili.

```
Colisions

def _alien_hit(self):
    previous_collisions = self.previous_collisions.copy()
    #2 True govore metodi da izbriše oba objekta sa ekrana
    collisions = pygame.sprite.groupcollide(self.bullets, self.aliens, True, True)

    if collisions != previous_collisions:
        self.stats.score += self.settings.alien_points
        self.sb.prep_score()
        self.sb.check_high_score()
        self.boom_sound.play()
        self.settings.aliens_killed += 1
```

7. UE, PARAMETRI IGRE I BODOVANJE

User Experience (UX), ili korisničko iskustvo, igra ključnu ulogu u uspehu bilo koje igre ili aplikacije. UX se odnosi na percepciju, doživljaj i zadovoljstvo korisnika tokom korišćenja proizvoda ili usluge, u ovom slučaju igrice. Konzistentnost u korisničkom iskustvu postizemo uvođenjem globalne settings klase koja sadrži parametre igre. Glavni problemi igrice bili su level cap i nezanimljivi neprijatelji. Poboljšanje u ovim aspektima došlo je nakon sugestija mog druga, koji je prvi igrao ovu igricu, suštinski jedinog game testera kojeg sam imao. Nakon sugestija uvedeni su:

- **Level cap** (bar u ovoj igri) je ograničenje koje određuje koliku težinu igre igrač može dosegnuti. On je neophodan jer u jednom trenutku stalnim povećavanjem parametara težine, igra bi postala nemoguća za igrati, a to svakako nije cilj. Igrač dostiže neku maksimalnu težinu a izazov mu postaje preživljavanje na takvom nivou.

```
Settings

def initialize_dynamic_settings(self):
    self.fleet_direction = 1 # (1=right) ((-1)=left)
    self.alien_points = 10

    self.ship_speed = 6.5
    self.bullet_speed = 4.5
    self.alien_speed = 1.0

def increase_speed(self):
    self.ship_speed *= self.speedup_scale
    self.bullet_speed *= self.speedup_scale
    self.alien_speed *= self.speedup_scale
    #level cap
    if(self.alien_speed >= 1.0 * (1.1 ** 10)):
        self.ship_speed = 6.5 * (1.1 ** 10)
        self.bullet_speed = 4.5 * (1.1 ** 10)
        self.alien_speed = 1.0 * (1.1 ** 10)
    self.alien_points = int(self.alien_points * self.score_scale)
    #score cap
    self.alien_points = int(self.alien_points * self.score_scale)
    if (self.alien_points >= 2500):
        self.alien_points = 2500

    # self.alien_speed = 1.0
```

- **Raznolikost neprijatelja** je omogućena pri instanciranju objekta alien, gde postoji određena šansa da se u flotu pusti specijalni neprijatelj (easter egg). Ovo može učiniti igru izazovnijom, što je moj tester potvrdio.

```

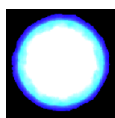
Randomizer

if (random.randint(1, 20) == 1):
    if (random.randint(1, 2) == 1):
        self.image = pygame.image.load('images/bill_cipher.png')
        self.image = pygame.transform.scale(self.image, (59, 89))
        self.bill = True
    else:
        self.image = pygame.image.load('images/easter_egg.png')
        self.image = pygame.transform.scale(self.image, (59, 89))
        self.sans = True
else:
    self.image = pygame.image.load('images/invader.png')
    self.image = pygame.transform.scale(self.image, (69, 48))

```

U settings code snippetu primetili smo mnogo parametara kao što su `alien_speed`, `bullet_speed`, `ship_speed`, `alien_points`. Pri svakom novom talasu neprijatelja oni se inkrementuju, ali postoji uslov koji ih ograničava na maksimalnu vrednost.

U randomizer code snippetu vidimo 2 easter egg neprijatelja koji se mogu pojaviti. To su Sans iz igre Undertale i Bill Cipher iz popularnog crtanog filma Gravity Falls. Oni osim drugačijeg izgleda imaju i drugačije projekte i postaju velika pretnja ako ih sretnemo na nekom višem nivou.



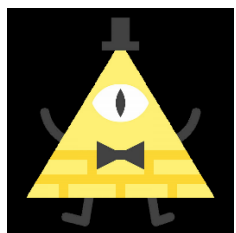
Razlika u hitbox-u između običnog i Bill Cipher projektila.

8. EASTER EGGS

Easter eggovi u gejmingu su skriveni ili neobični sadržaji, elementi, poruke ili referencije unutar video igara koje igrači mogu otkriti ili "otključati" ako pažljivo istraže igru ili izvrše određene radnje. Ovi skriveni elementi često predstavljaju izazov za igrače i dodaju dodatnu dubinu i zabavu igri. Easter eggovi često predstavljaju posvetu različitim aspektima kulture, istorije ili same igre, te su postali popularni i prepoznatljiv deo gejmerske kulture. Tipovi Easter eggova u gejmingu:

- Referencijalni easter eggovi: Ovo su često reference na druge igre, filmove, knjige ili popularnu kulturu. Na primer, igrači mogu pronaći likove ili objekte iz drugih igara unutar igre.
- Skriveni svetovi ili nivoi: Neki easter eggovi omogućavaju igračima pristup skrivenim svetovima ili nivoima unutar igre koji nisu deo glavnog toka igre. Ovi nivoi često donose neobične izazove ili sadrže humorističke elemente.
- Prikriti developeri: U mnogim igrama, razvojni timovi skrivaju svoje reference unutar igre. To može uključivati njihove imena, slike ili poruke koje igrači mogu pronaći.
- Nostalgija i retro easter eggovi: Ovi easter eggovi često se odnose na stare igre i konzole, i donose elemente iz prošlih vremena. Na primer, može se pojaviti mini-igra inspirisana klasičnim arkadnim igrama.

Easter eggovi su postali integralni deo gejming sveta i često dodaju dodatnu dimenziju zabavi i istraživanju igara. Otkrivanje easter eggova može biti izazov, a igrači često koriste zajednicu i forume kako bi delili svoja otkrića i pomažu jedni drugima u njihovom pronalaženju. Ovo easter eggovi u mojoj igrici:



9. OSVRT NA 3D GRAFIKU

Nažalost, Pygame je prvenstveno dizajniran za razvoj 2D igara i ne pruža ugrađenu podršku za 3D grafiku. Ako planiramo razvijati 3D igre, možda bismo trebali da razmotrimo korišćenje neke druge biblioteke ili okvira, ili neki drugi specijalizovani alat za 3D grafiku u Pythonu.

Ono što možemo uraditi jeste simulacija, kao u retro igri Doom. Objekti su bili 2D spriteovi koji su stvarali iluziju trodimenzionalnog prostora. Iako grafika u Doomu možda nije tako napredna kao u savremenim igrama, njegov uticaj na gejming industriju i žanr pucačina iz prvog lica je značajan. Pogledaćemo kod snippet iz verzije Doom igrice kreirane pomoću pygame biblioteke. (link do repozitorijuma dostupan u literaturi)

```
DOOM

def ray_cast(self):
    self.ray_casting_result = []
    ox, oy = self.game.player.pos
    x_map, y_map = self.game.player.map_pos
    ray_angle = self.game.player.angle - HALF_FOV + 0.0001

    for ray in range(NUM_RAYS):
        sin_a, cos_a = math.sin(ray_angle), math.cos(ray_angle)

        #Presek linija se koristi kako bi se utvrdilo gde tačno zrak prolazi kroz granice
        #između različitih oblika na mapi (na primer, zidovi u igri). Kada se pronade presek, može
        #se odrediti koja tekstura ili boja treba da bude prikazana na tom mestu na ekranu.
        # Izračunavanje preseka za horizontalne linije
        y_hor, dy = (y_map + 1, 1) if sin_a > 0 else (y_map - 1e-6, -1)
        depth_hor = (y_hor - oy) / sin_a
        x_hor, dx = ox + depth_hor * cos_a, (dy / sin_a) * cos_a

        # Izračunavanje preseka za vertikalne linije
        x_vert, dx = (x_map + 1, 1) if cos_a > 0 else (x_map - 1e-6, -1)
        depth_vert = (x_vert - ox) / cos_a
        y_vert, dy = oy + depth_vert * sin_a, (dx / cos_a) * sin_a

        # Određivanje teksture na osnovu preseka, šta je osvetljeno, šta vidi kamera to jest
        #naš igrač
        texture_hor = self.game.map.world_map.get((int(x_hor), int(y_hor)), texture_hor)
        texture_vert = self.game.map.world_map.get((int(x_vert), int(y_vert)),
        texture_vert)

        # Određivanje dubine, pomeraja teksture, i eliminacija fishbowl efekta
        depth, texture, offset = (depth_vert, texture_vert, y_vert % 1) if depth_vert <
        depth_hor else (
            depth_hor, texture_hor, (1 - x_hor) if sin_a > 0 else x_hor % 1
        )
        depth *= math.cos(self.game.player.angle - ray_angle)

        # Projekcija
        proj_height = SCREEN_DIST / (depth + 0.0001)

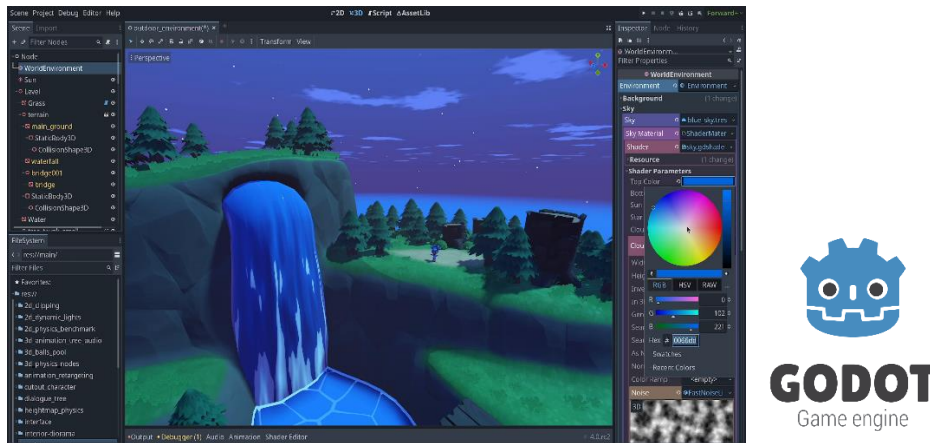
        # Dodavanje rezultata ray casting-a
        self.ray_casting_result.append((depth, proj_height, texture, offset))

        ray_angle += DELTA_ANGLE
```

U komentarima kod snippeta možemo naići na neke nove termine. Ukratko, dubina preseka je udaljenost od igrača do tačke preseka sa objektom, pomeraj teksture određuje kako će tekstura biti prikazana na ekranu, a efekat riblje činije (fishbowl) se odnosi na distorziju, tojest iskrivljenja koja mogu nastati pri simulaciji 3D prostora.

Ray casting se ovde koristi za određivanje vidljivih površina na ekranu. Ova metoda omogućava eliminaciju nevidljivih delova scene, poboljšavajući efikasnost daljih računanja u pajplajnu. Zrake, predstavljene pikselima, ispaljuju se sa ravni gledanja (doom igrač) u scenu. Pomoću analize pogođenih objekata (čudovišta) određujemo vidljive delove scene, eliminišući nevidljive objekte.

Ako želimo da pravimo moderne 3D igre, pygame nam nije od pomoći. Game engini su uvedeni kako bi ubrzali razvoj igara, omogućili pristup manjim timovima, pružili moćne alate za grafički dizajn, osigurali doslednost i optimizaciju na različitim platformama te podržali razvoj igara za mobilne uređaje, konzole i računare. Ovi alati omogućavaju fokusiranje na kreativni proces umesto tehničkih izazova.



Postoje heavyweight engini poput Unreal (C++) i Unity (C#), njih karakterišu velika moć, podrška industrije ali i memorijska zahtevnost za naš računar. A tu je i GODOT, otvorenog koda, memorijski nezahtevan i sa gomilom nezavisnih developera koji stoje iza njega umesto nekog tech giganta. Godot podržava C#, ali ima i svoj nativni jezik GDScript. Evo pregleda nekih 3D tehnika i funkcionalnosti u Godot-u:

- **3D Rendering:** Godot koristi moderni OpenGL ES 3.0 renderer za 3D grafiku. Podržava različite jezike za senčenje, uključujući VisualScript, ShaderScript i GDScript.
- **Sistem scena:** Godot koristi sistem scena gde se scene mogu sastojati od drugih scena, omogućavajući modularan i ponovljiv dizajn. Scene mogu sadržavati i 2D i 3D elemente, pružajući fleksibilnost u kreiranju kompleksnih igračkih okruženja.
- **3D Fizika:** Godot uključuje 3D fiziku koja omogućava realistične simulacije fizičkih interakcija u 3D prostoru. Podržava karakteristike kao što su kruta tela, oblici kolizije i zglobovi.

- **Navigacija i pronalaženje puta:** Godot pruža alate za navigaciju i pronalaženje puta u 3D prostoru, omogućavajući likovima i entitetima inteligentno kretanje u igračkom svetu.
- **3D Animacija:** Godot podržava skeletnu animaciju za 3D modele, omogućavajući realistično kretanje likova. Mešanje animacija i mašine stanja poboljšavaju kontrolu nad animacijama likova.
- **Osvetljenje i senke:** Godot podržava različite tehnike osvetljenja, uključujući globalnu iluminaciju i dinamičke senke. Senke u realnom vremenu i lightmap-e doprinose vizualnom kvalitetu 3D scena.
- **Efekti nakon procesiranja:** Godot omogućava primenu efekata nakon procesiranja kako bi se poboljšala vizuelna privlačnost 3D scena. Efekti poput sjaja i korekcije boja lako se mogu implementirati.
- **Podrška za VR:** Godot ima eksperimentalnu podršku za virtualnu stvarnost (VR), omogućavajući programerima kreiranje VR iskustava koristeći engine.

FRESNAL

Jedan od najboljih efekata za prostorno senčenje, fresnel, koristi se za različite svrhe. Od toga da voda izgleda prirodnije, simulacije rubnog osvetljenja pa čak do kreiranja forcefield efekta. Fresnel efekat simulira odnos između ugla površine i količine svetlosti koju ćete videti dolazeći sa te površine. Što je površina više okrenuta od vas, to ćete videti više svetlosti, stvarajući svetle ivice.

```

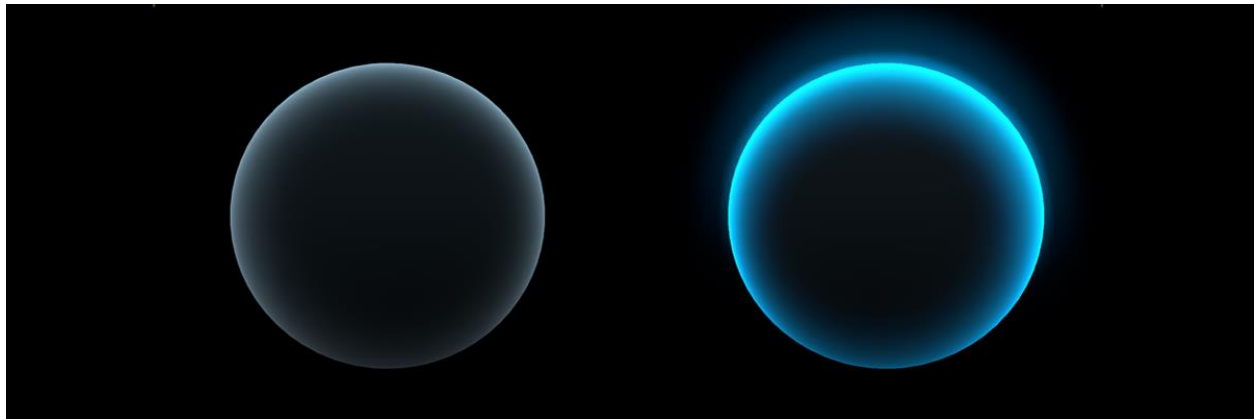
//BASIC FRESNAL
float fresnel(float amount, vec3 normal, vec3 view)
{
    return pow((1.0 - clamp(dot(normalize(normal), normalize(view))), 0.0, 1.0 )), amount);
}

void fragment()
{
    vec3 base_color = vec3(0.0);
    float basic_fresnel = fresnel(3.0, NORMAL, VIEW);
    ALBEDO = base_color + basic_fresnel;
}

//COLORFUL GLOW FRESNAL
vec3 fresnel_glow(float amount, float intensity, vec3 color, vec3 normal, vec3 view)
{
    return pow((1.0 - dot(normalize(normal), normalize(view))), amount) * color * intensity;
}

void fragment()
{
    vec3 base_color = vec3(0.5, 0.2, 0.9);
    vec3 fresnel_color = vec3(0.0, 0.7, 0.9);
    vec3 fresnel = fresnel_glow(4.0, 4.5, fresnel_color, NORMAL, VIEW);
    ALBEDO = base_color + fresnel;
}

```



Efekti fresnal i fresnal_glow funkcija

KINEMATIČKO TELO - ROTIRANJE (transformacije)

```
GODOT

extends KinematicBody

onready var left_b = get_node("Camera/UIControl/LeftButton")
onready var right_b = get_node("Camera/UIControl/RightButton")
onready var button_timer = get_node("Camera/UIControl/ButtonTimer")
var current_angle = Vector3();
var target_rotation = Vector3();
export var speed = 3.0;

func _process(delta):
    rotation_degrees = Vector3(0,lerp(rotation_degrees.y,target_rotation.y,speed*delta),0)

func _on_RightButton_pressed():
    target_rotation = target_rotation - Vector3(0,90,0)
    left_b.hide()
    right_b.hide()
    button_timer.start()

func _on_LeftButton_pressed():
    target_rotation = target_rotation + Vector3(0,90,0)
    left_b.hide()
    right_b.hide()
    button_timer.start()

func _on_ButtonTimer_timeout():
    left_b.show()
    right_b.show()
```


10. LITERATURA

- [1] <https://retrostylegames.com/blog/are-2d-games-still-popular>
- [2] <https://en.wikipedia.org/wiki/Pygame>
- [3] <https://www.pygame.org/docs>
- [4] <https://github.com/MatijaMax/alien-invasion-game>
- [5] <https://ray.so/>
- [6] <https://github.com/StaniislavPetrovV/DOOM-style-Game/tree/main>
- [7] <https://godotshaders.com/snippet/fresnel/>