



UNIVERZITET U NOVOM SADU
FAKULTET TEHNIČKIH NAUKA U
NOVOM SADU



Matija Maksimović RA 132/2020

Analiza razvoja 2D igara upotrebom “PyGame” biblioteke

SEMINARSKI RAD
- Osnovne akademske studije -

Novi Sad, 2024.



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

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

SADRŽAJ

1. Uvod	
.....	2
2. Istorija 2D igara	
.....	3
3. PyGame biblioteka	
.....	6
4. Petlja igre	
.....	7
5. Kontrole i multimedija	
.....	9
6. Iscrtavanje objekata i sudari	
.....	12
7. UE, parametri igre i bodovanje	
.....	16
8. Uskršnja jaja	
.....	18
9. Osvrt na 3D grafiku	
.....	19
10. Zaključak	
.....	24
11. Literatura	
.....	25

1. UVOD

Svet video igara, kao oblik zabave i umetnosti, kontinuirano se razvija, donoseći nove izazove i mogućnosti za programere 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” (eng. Space Invaders) 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 posveta 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.



Slika 1: Napadač iz svemira [1]

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 godina. Sve što nam je bilo potrebno bilo je preduzeće koje bi moglo doneti kvalitet arkadnih igara kućnim konzolama.



Slika 2: Izbor likova u igrici Jazz Jackrabbit 2 [3]

Već 1985. godine, *Nintendo* je izdao svoju legendarnu konzolu - *Nintendo Entertainment System* - sa najpopularnijim igrama svih vremena kao što su *Super Mario*, *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 igrama i bile su gotovo izumrle. Sve najcenjenije franšize video igara prešle su

na 3D. Ovaj novi prikaz je dominirao na kućnim konzolama, zamenjujući 2D igre na prenosnim uređajima. Iako se 2D igre nisu potpuno povukle, premeštanje na prenosne uređaje im je naizgled umanjilo značaj. Prenosne verzije igara popularnih serijala uvek su bile tretirane kao sporedni naslovi, dok su konzolne igre uglavnom bile direktni nastavci jedne na drugu. Zato se 2D igre na prenosnim uređajima nisu mogle takmičiti sa velikim konzolnim igrama.



Slika 3: Igra simulacije farme u 2D stilu [4]

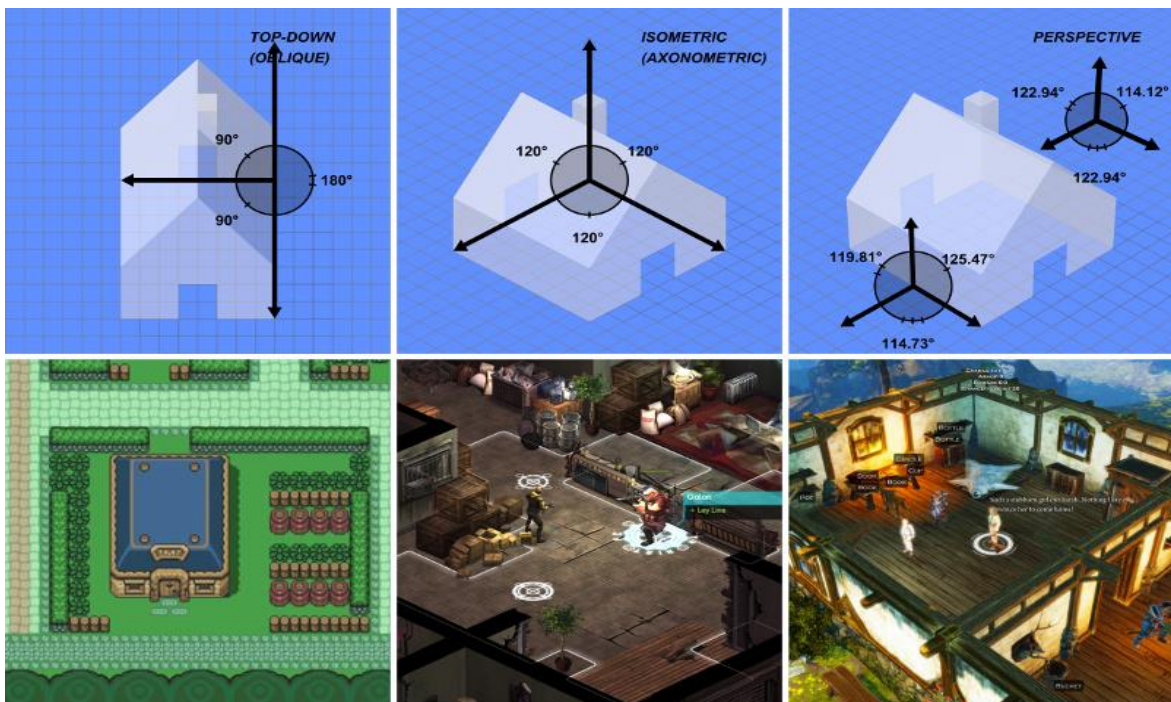
Međutim, uskoro su 2D igre doživele masovni povratak. Njihova obnova počela je sa nastankom *Xbox Live Arcade* servisa 2001. godine. Ova platforma je nudila neke jednostavne igre, kao i arkadne klasike. *Xbox Live Arcade* je postao mesto za igre malih razmera i stvorio je prostor za 2D igre u retro stilu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



Slika 4: Avanturistička igra u 2D svetu [5]

Dvodimenzionalne igre se mogu razlikovati po stilu umetnosti, žanru, perspektivi i platformi. Žanrovi video igara su gotovo isti za 2D i 3D igre. To su igre uloga (eng. role-playing games), platformeri (eng. platformers), akcione igre (eng. action games), borbe (eng. fighting games), simulacije (eng. simulations), pucačine (eng. shooters), slagalice (eng. puzzles) itd. Ali ključna razlika između 2D i 3D igara je izbor perspektive. Ograničenja perspektive 2D igara izuzetno su važna jer mogu uticati na sam način igranja i celokupnog korisničkog iskustva:

- **Pogled sa strane (eng. Side view):** 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 (eng. Top-down view):** Igra izgleda kao da je kamera iznad glave. Igračko polje se vidi kao iz ptičje perspektive ili ima blago nagnjanje kamere.
- **Izometrijska perspektiva (eng. Isometric perspective):** Igra se prikazuje iz određenog ugla kamere. Savršena je za stvaranje iluzije 3D prostora, prikazujući tri strane objekta.
- **Jednostrani ekran (eng. Single-screen):** Svaki nivo se postavlja u novoj sobi koja ispunjava ceo ekran. Kada se nivo završi, prelazi se na sledeću sobu.



Slika 5: Različiti tipovi perspektive u igrama [6]

3. PYGAME BIBLIOTEKA

PyGame koristi biblioteku *Simple DirectMedia Layer* (SDL) s ciljem da omogućiti razvoj računarskih igara u realnom vremenu bez potrebe za mehanizmima niskog nivoa (programski jezik C i njegovi derivati). Osnovna pretpostavka je da se najzahtevnije funkcije unutar igara mogu apstrahovati od same logike igre, čime postaje moguće koristiti jezike visokog nivoa, jezike poput Pythona za stvaranje strukture igre.

Ova biblioteka takođe nudi niz drugih mogućnosti, uključujući vektorsku matematiku, detekciju sudara, upravljanje 2D *sprite* scenama, podršku za *Musical Instrument Digital Interface*(MIDI), 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.



Slika 6: PyGame logo [7]

Postupak formiranja igre uz *PyGame* analizira se kroz projekat koji je inspirisan igricom *Space Invaders*. Ideja igre je u tome da svemirski brod obori što više flota neprijatelja i spase galaksiju. Flota se spušta ka dnu ekrana i dodir sa njom ili projektilima koje ispaljuje dovodi do gubitka jednog života igrača. Izgled predstavlja jednu vrstu kolaža piksel umetnosti koja je bila dostupna na internetu i odgovarala temi igre. Igra treba da ukaže na lepotu koju nosi piksel umetnost i istakne je svojim jednostavnim i pristupačnim pravilima igre.

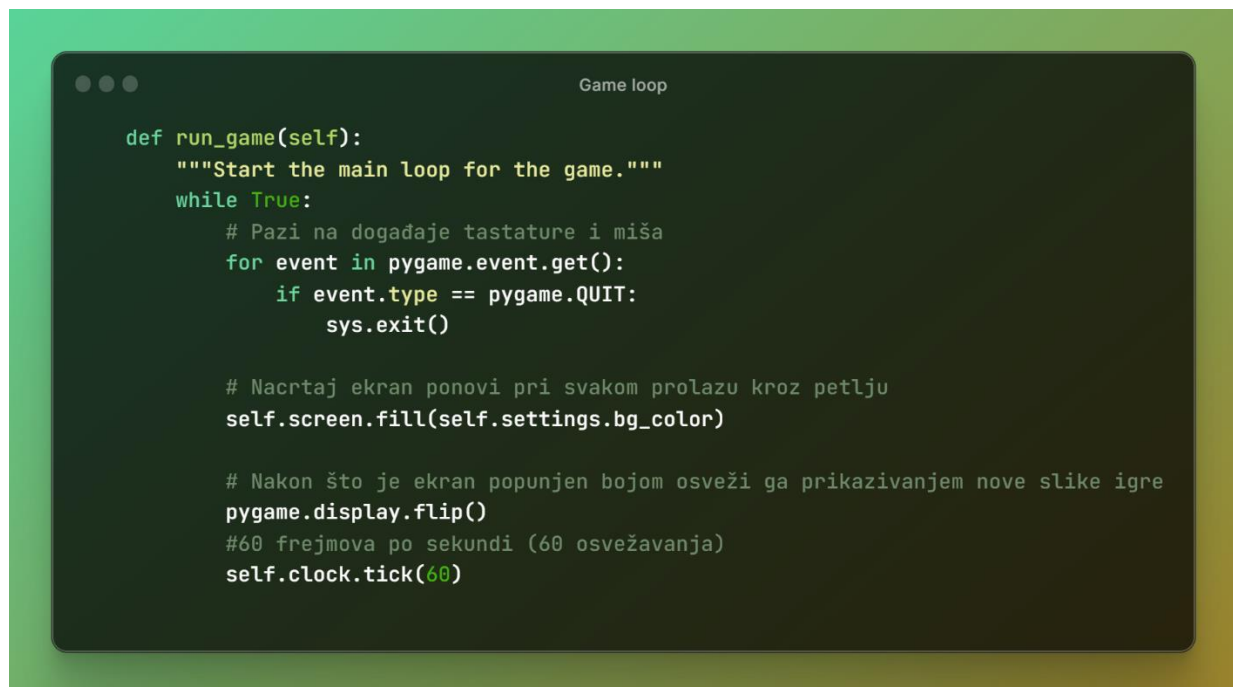


Slika 7: Obaranje flote vanzemaljaca [9]

4. PETLJA IGRE

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

- **Ulaz (eng. Input):** U svakoj iteraciji petlje, igra proverava korisnički ulaz, kao što su pritisci na tastere, klikovi miša i slično.
- **Logika (eng. Logic):** Nakon što se prikupe ulazni podaci, igra izvršava svoju logiku. To uključuje računanje fizike, upravljanje protivnicima, reagovanje na događaje itd.
- **Renderovanje (eng. Rendering):** Nakon što se ažurira igračka logika, igra crta novi okvir na ekranu. To uključuje iscrtavanje svih objekata, karaktera i pozadine kako bi se prikazalo trenutno stanje igre.
- **Ograničenje brzine (eng. 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 na mašinama koje nemaju iste hardverske specifikacije.
- **Provera uslova za završetak igre (eng. Game over conditions):** *Game loop* takođe proverava uslove za završetak igre, kao što su pobeda, gubitak ili napuštanje igre (uslovi za izlaz iz petlje).

A screenshot of a code editor window titled "Game loop". The code is written in Python and defines a function `run_game(self)`. The function starts with a docstring: `"""Start the main loop for the game."""`. It then enters a `while True:` loop. Inside the loop, it first checks for keyboard and mouse events using `pygame.event.get()`. If the `pygame.QUIT` event is detected, it calls `sys.exit()`. Next, it clears the screen with `self.screen.fill(self.settings.bg_color)`. Then, it updates the display with `pygame.display.flip()`. Finally, it uses `self.clock.tick(60)` to limit the frame rate to 60 FPS. Comments in the code explain each step: checking for events, clearing the screen, updating the display, and limiting the frame rate.

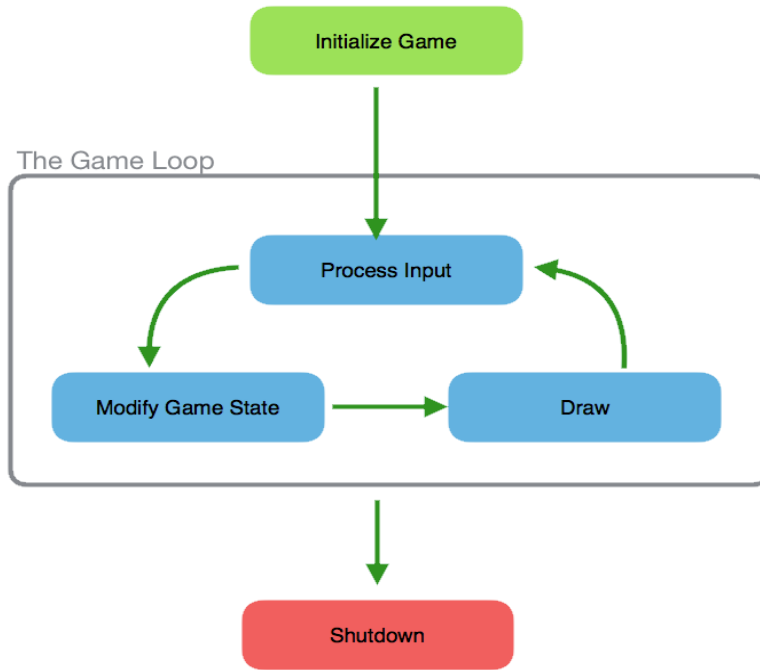
```
def run_game(self):
    """Start the main loop for the game."""
    while True:
        # Pazi na događaje tastature i miša
        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                sys.exit()

        # 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)
```

Slika 8: Funkcija petlje igre [9]

Ova petlja se ponavlja neprestano tokom izvršavanja igre, čime se omogućava korisnicima da interaguju 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.



Slika 9: Životni ciklus igre [10]

Ovo je funkcija gde inicijalizujemo parametre 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")
  
```

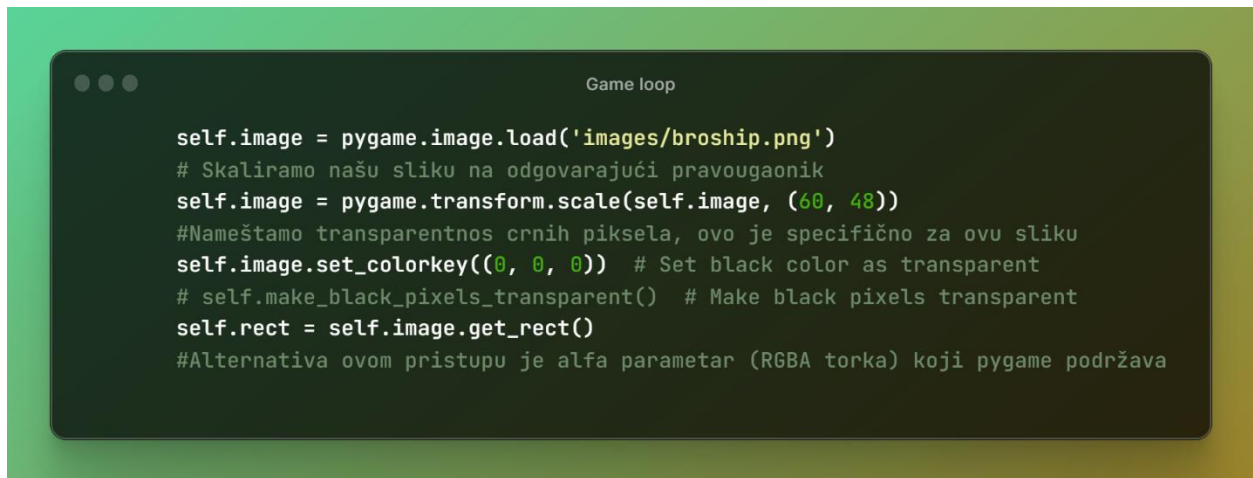
The screenshot shows a code editor window titled 'Game loop'. It contains a Python function `__init__(self)` that initializes the game. The code includes comments and calls to `pygame.init()`, `pygame.time.Clock()`, `Settings()`, `pygame.display.set_mode()`, and `pygame.display.set_caption()`.

Slika 10: Inicijalizacija parametara [9]

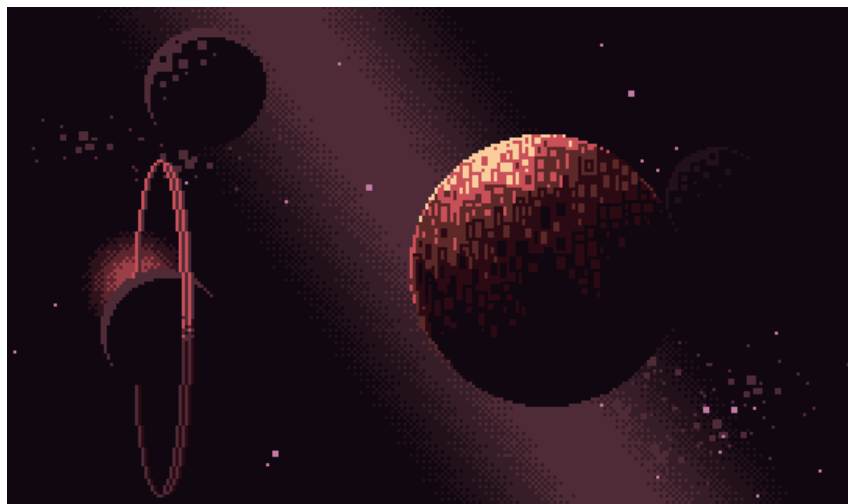
5. KONTROLE I MULTIMEDIJA

Multimediji igraju ključnu ulogu u svetu video igara i stvaranju bogatog i uzbuđljivog igračkog iskustva. Za ovu igricu sledeći aspekti su od posebnog značaja:

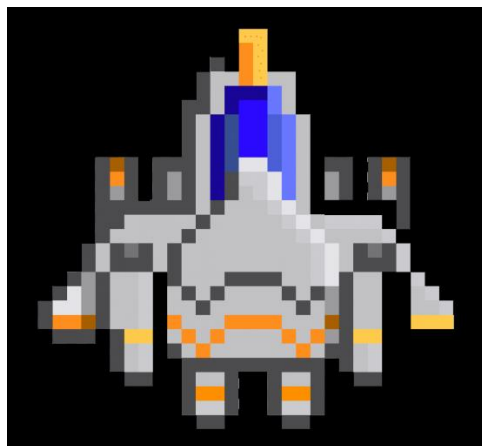
Vizuelni elementi: Grafika i animacije čine osnovu svake igre. Kvalitetna grafika, detalji i estetičan dizajn igre privlače igrače, stvarajući zanimljivu i uverljivu virtuelnu stvarnost. Vizualni elementi takođe pomažu u stvaranju emotivnih reakcija i atmosfere u igri. U ovoj igri, svaki objekat smatra se pravougaonikom (eng. rectangle). Ovaj pravougaonik može postati kontejner neke slike i uključivanjem transparentnosti postići iluziju uklapanja sa pozadinom igre. Kontaktna zona (eng. hitbox) objekta će ostati pravougaonik, samo što igrači to neće videti, pa dizajner treba obratiti pažnju na ovo.



Slika 11: Rad sa slikama [9]



Slika 12: Pozadina igre [9]



Slika 13: Brod koji igrač kontroliše [9]

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

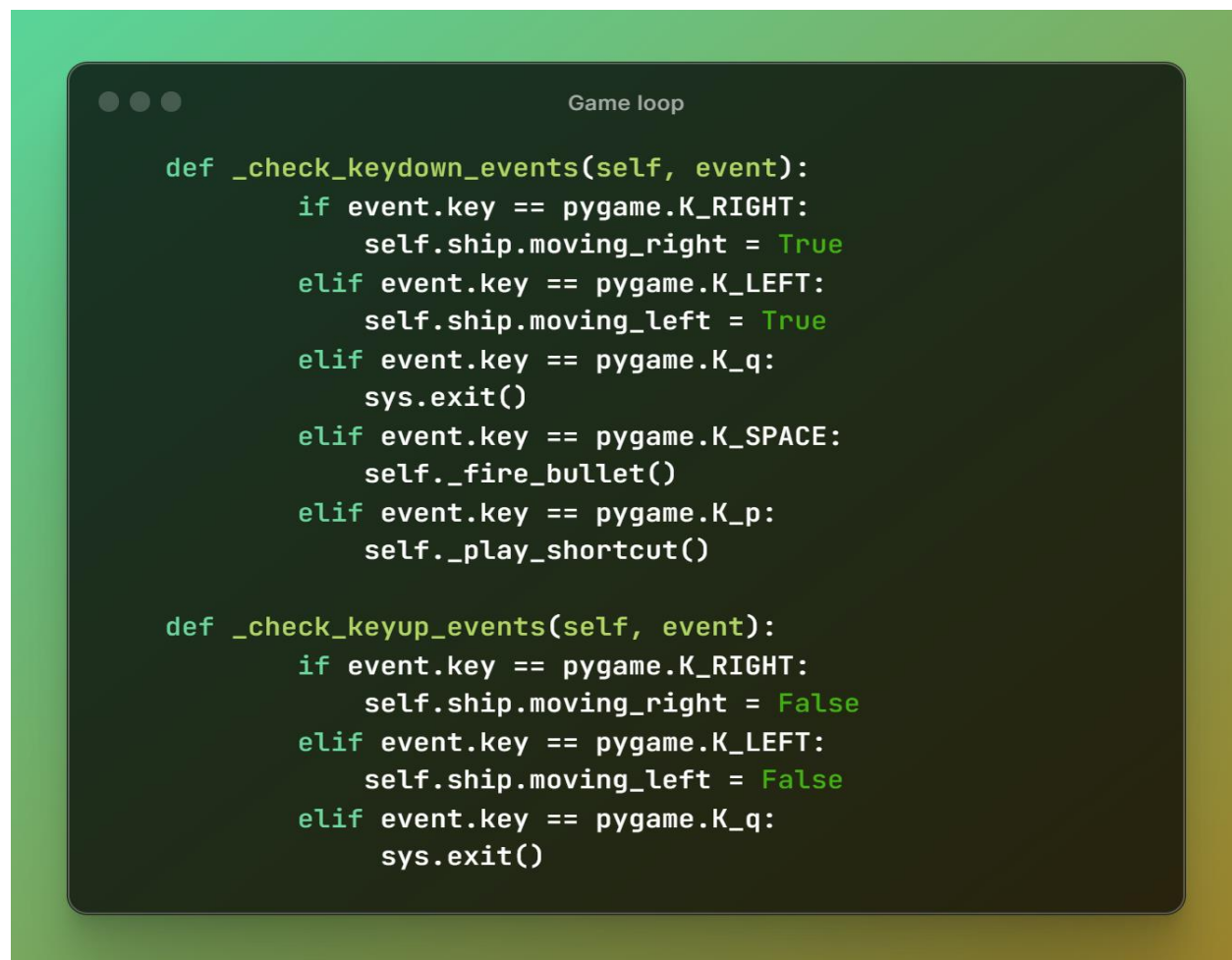
```
Game loop

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)
```

Slika 14: Implementacija zvučnih efekata [9]

KeyPress događaj (eng. event) u *PyGame* biblioteci ima veliku važnost jer omogućava igračima da interaguju sa igrom putem tastature. *KeyDown* i *KeyUp* događaji upotrebljeni su u sledećem kodu:



Slika 15: Implementacija *KeyPress* događaja [9]

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

6. ISCRTAVANJE OBJEKATA I SUDARI

PyGame podržava različite vrste objekata, uključujući pravougaonike, krugove, linije i slike. Za crtanje pravougaonika koristimo funkciju *draw*. 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))
```

Slika 16: Crtanje pravougaonika [9]

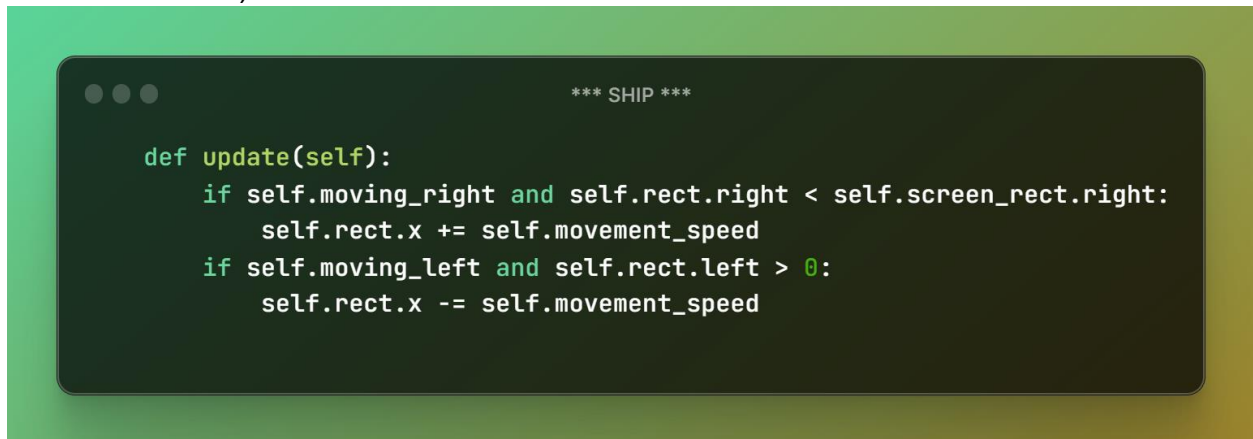
Važni objekti koji se crtaju na ekranu jesu vanzemaljci, igrač i projektili koje oni mogu ispaljivati. Za svaki objekat koji treba da se nacrtava imamo posebnu klasu sa ključnom metodom *blitme* koja daje objekat iscrtava na ekranu tokom petlje igre. Uz *blitme* mapiramo sliku našeg objekta na pravougaonik,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

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

Slika 17: *Blitme* funkcija [9]

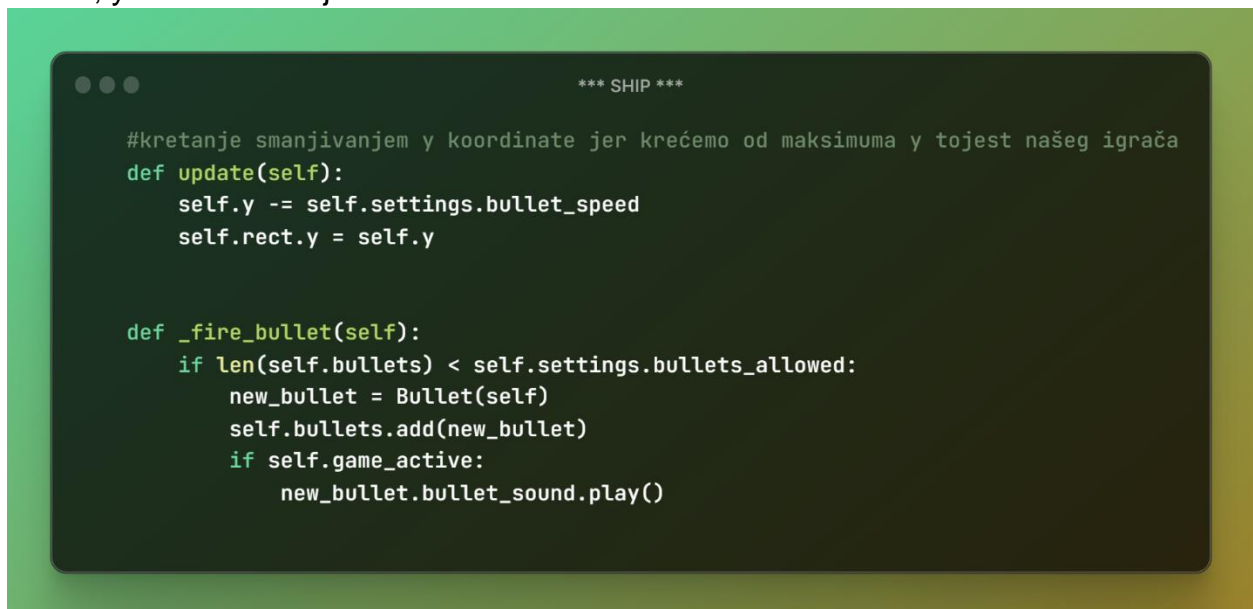
Za svemirski brod (igrača) imamo vezane osnovne kontrole kretanja i pucanja. Pošto je u pitanju 2D igra, kretanje se opisuje uz pomoć koordinata *x* i *y*. Važno je napomenuti

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 događaja) x koordinate igrača. Uslov jesu *boolean* koji nam je dao *KeyDown* događaj i ograničenje ekrana (brod logično ne sme izaći van ekrana).



Slika 18: Promena položaja broda [9]

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



Slika 19: Ispaljivanje projektila [9]

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

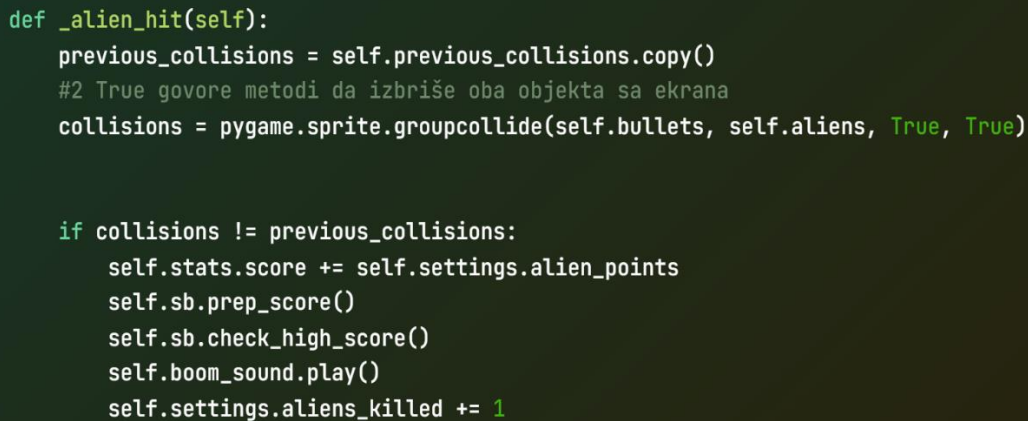
#Ako flota udari u granicu ekrana, ona mora promeniti smer kretanja i spustiti se dole
za neku predefinisanu vrednost
def _check_fleet_edges(self):
    for alien in self.aliens.sprites():
        if alien.check_edges():
            self._change_fleet_direction()
            break

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

Slika 20: Iscrtavanje flote neprijatelja [9]

Ključni deo ovog koda je *sprites* funkcija. Ako naš *rect* objekat (koristimo ovu klasu da definišemo dimenzije i poziciju objekata igre) pretvorimo u *sprite* objekat (proširena klasa *rect* stvorena da učini objekat interaktivnijim), dobijamo mogućnost grupisanja (primer flote) ali i mnogo važnije sudare objekata (eng. object collision).

Želimo da znamo kada je metak pogodio vanzemaljca i nakon toga ga uklonimo sa ekrana. Funkcija *groupcollide()* poredi objekte *rect* svakog elementa u jednoj grupi sa objektom *rect* svakog elementa u drugoj grupi. U ovom projektu, ona poredi *rect* svakog metka sa objektom *rect* svakog vanzemaljca i vraća rečnik koji sadrži metke i vanzemaljce koji su se sudarili. (kada god se dogodi preklop vraća se par ključ-vrednost) Svaki ključ u rečniku biće metak, a odgovarajuća vrednost će biti vanzemaljac koji je pogođen. Dva argumenta *True* govore PyGame modulu da izbriše metke i vanzemaljce koji su se sudarili.



```
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)

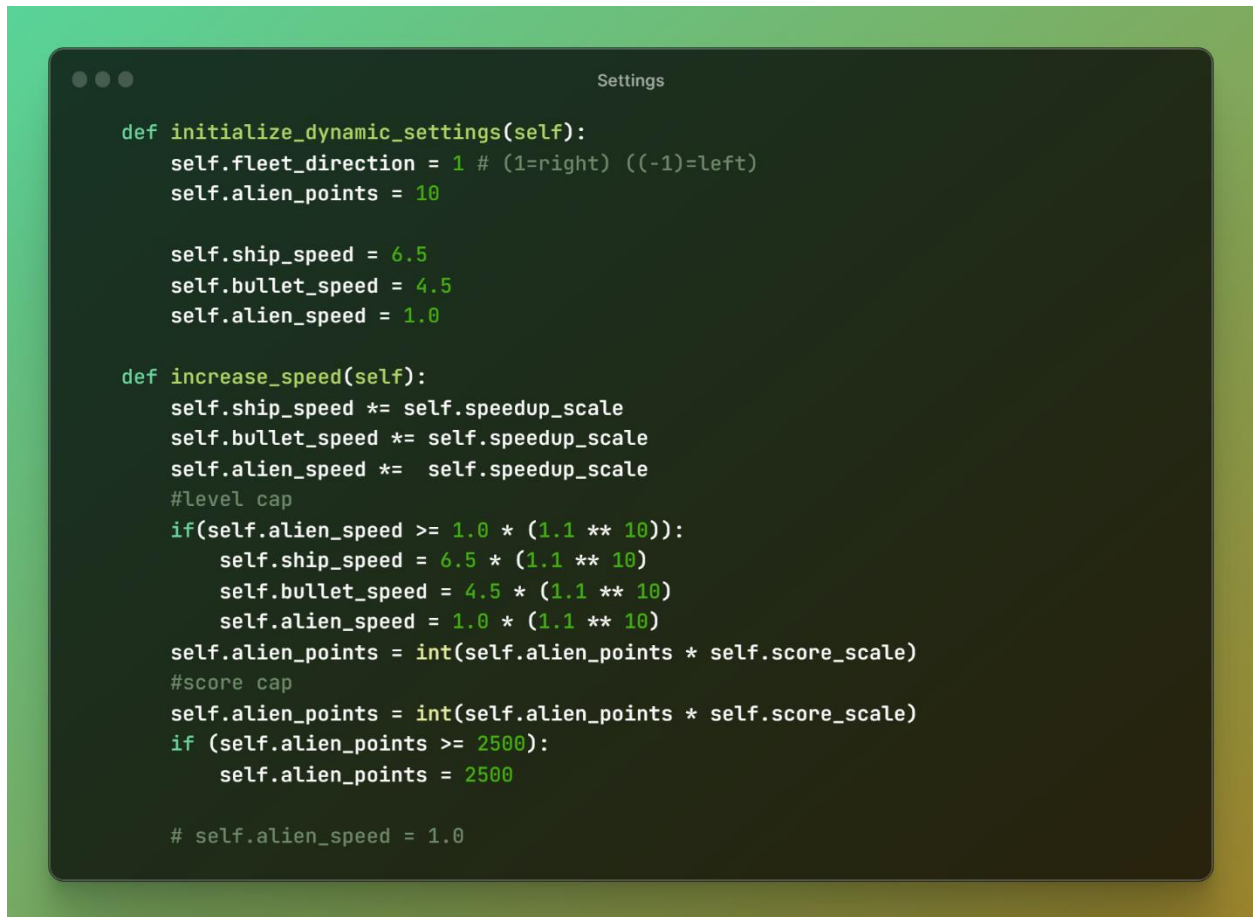
    if collisions != previous_collisions:
        self.stats.score += self.settings.alien_points
        self.sb.prep_score()
        self.sb.check_high_score()
        self.boom_sound.play()
        self.settings.aliens_killed += 1
```

Slika 21: Detekcija sudara [9]

7. UE, PARAMETRI IGRE I BODOVANJE

Korisčko iskustvo (eng. *User Experience (UE)*) igra ključnu ulogu u uspehu bilo koje igre ili aplikacije. *UE* se odnosi na percepciju, doživljaj i zadovoljstvo korisnika tokom korišćenja proizvoda ili usluge, u ovom slučaju igrice. Konzistentnost u korisničkom iskustvu postižemo uvođenjem globalne *settings* klase koja sadrži parametre igre. Glavni problemi bili su prekomerno ubrzavanje igre posle određenog nivoa zbog nedostatka ograničenja težine (eng. level cap) i nezanimljivi neprijatelji. Poboljšanje u ovim aspektima došlo je nakon predloga mog druga, koji je prvi igrao ovu igricu, suštinski jedinog testera kojeg sam imao. Nakon predloga 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, igranje bi postalo nemoguće, 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

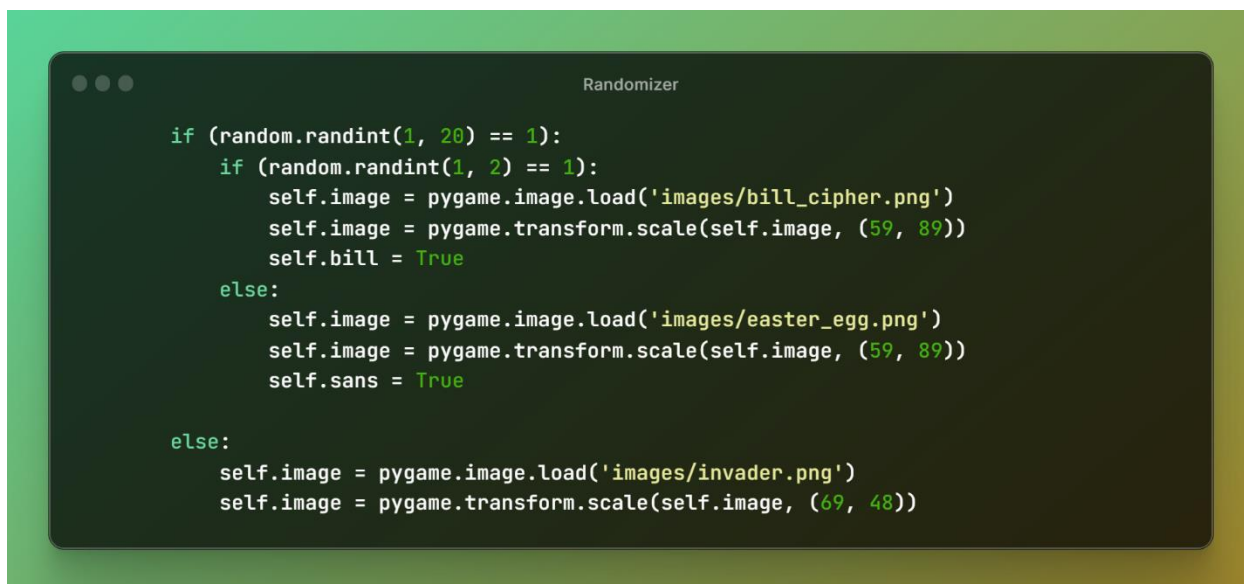
def increase_speed(self):
    self.ship_speed *= self.speedup_scale
    self.bullet_speed *= self.speedup_scale
    self.alien_speed *= self.speedup_scale
    #level cap
    if(self.alien_speed >= 1.0 * (1.1 ** 10)):
        self.ship_speed = 6.5 * (1.1 ** 10)
        self.bullet_speed = 4.5 * (1.1 **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

```

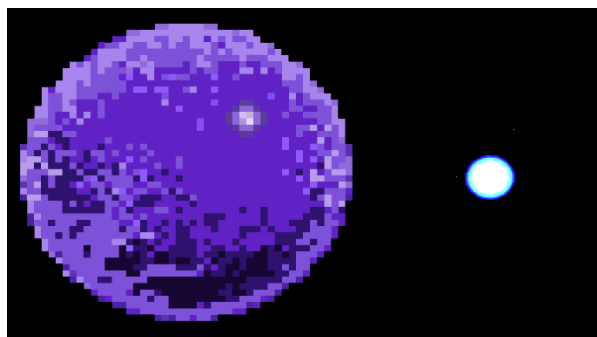
Slika 22: Parametri igre [9]

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



Slika 23: Nasumični neprijatelji [9]

U isečku koda za podešavanja 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 isečku koda za generisanje nasumičnih vrednosti vidimo 2 uskršnja jaja kao 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 igraču ako ih sretne na nekom višem nivou.



Slika 24: Razlika u hitbox-u između običnog i Bill Cipher projektila [9]

8. USKRŠNJA JAJA

Uskršnja jaja u multimediji su skriveni ili neobični sadržaji, elementi, poruke ili reference 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. Uskršnja jaja često predstavljaju posvetu različitim aspektima kulture, istorije ili same igre, te su postali popularni i prepoznatljiv deo gejmerske kulture. Tipovi ovih posveta u gejmingu:

- Referencijalna uskršnja jaja: 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: Uskršnja jaja 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 programeri: 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 uskršnja jaja: Ova uskršnja jaja 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.

Uskršnja jaja su postala integralni deo gejming sveta i često dodaju dodatnu dimenziju zabavi i istraživanju igara. Otkrivanje uskršnjih jaja može biti izazov, a igrači često koriste zajednicu i forume kako bi delili svoja otkrića i pomažu jedni drugima u njihovom pronalaženju. Ovo su uskršnja jaja u mojoj igrici:



Slika 25: Uskršnja jaja [9]

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, bolje bi bilo razmotriti upotrebu neke druge biblioteke ili okvira, ili neki drugi specijalizovani *Python* alat za 3D grafiku.

Ono što možemo uraditi jeste simulacija, kao u retro igri *Doom*. Objekti su bili 2D “spriteovi”, a udaljenost temena zidova umanjivana je u odnosu na daljinu od igrača što je stvaralo efekat trodimenzionalnosti. Objekti neprijatelja su imali više različitih tekstura koje su bile prilagođene načinu kretanja neprijatelja i poziciji igrača. Iako grafika u ovoj igri možda nije tako napredna kao u savremenim igrama, njen uticaj na industriju igara i žanr pucačina iz prvog lica je značajan. Pogledaćemo isečak koda iz verzije ove igrice kreirane pomoću *PyGame* biblioteke.

```

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 tojest
        # 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

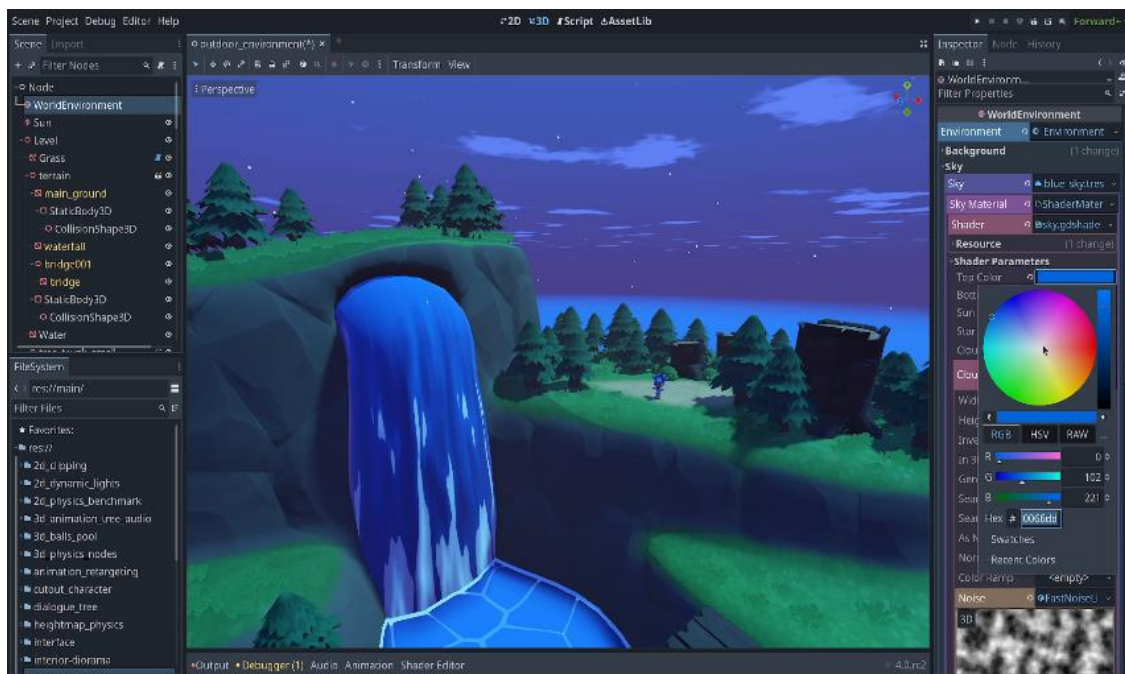
```

Slika 26: Ray casting [11]

U komentarima isečka koda 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 (eng. fishbowl) se odnosi na distorziju, tj. 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. Zraci se ispaljuju iz tačke gledišta igrača, tj. sa centra ekrana u scenu. Pomoću analize pogođenih objekata (čudovišta i zidova) određujemo vidljive delove scene, eliminišući nevidljive objekte.

Ako želimo da pravimo moderne 3D igre, *PyGame* nam nije od pomoći. Pogoni za razvoj video igara (eng. game engine) 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 usresređenost na kreativni proces umesto tehničkih izazova.



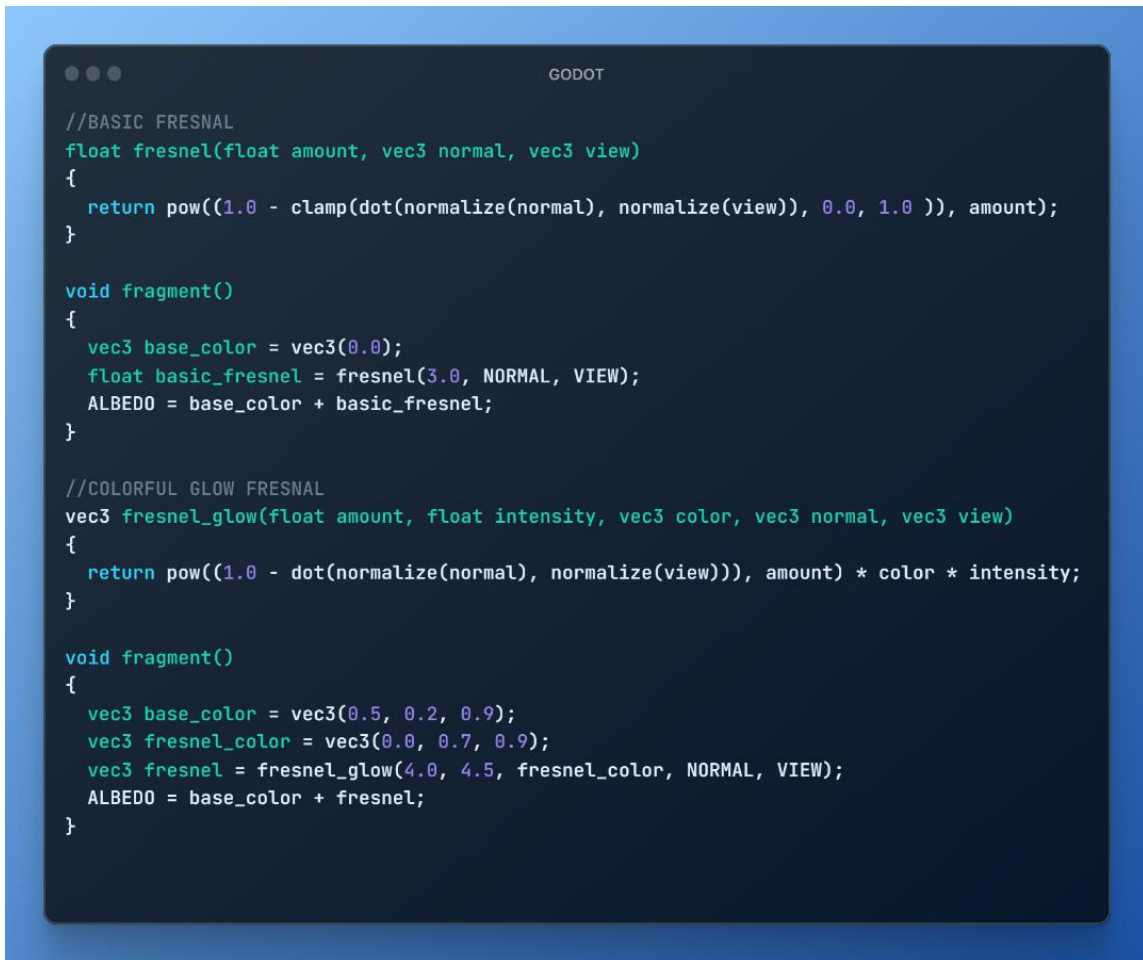
Slika 27: Godot korisnički interfejs [12]

Postoje kompleksniji, opšti pogoni poput *Unreal* (C++) i *Unity* (C#), njih karakterišu velika moć, podrška industrije ali i hardverska zahtevnost za naš računar. A tu je i *Godot*, otvorenog koda, hardverski nezahtevan i sa gomilom nezavisnih programera koji stoje iza njega umesto nekog tehnološkog giganta. *Godot* podržava C#, ali ima i svoj sopstveni jezik GDScript. Evo pregleda nekih 3D tehnika i funkcionalnosti u *Godot* pogonu:

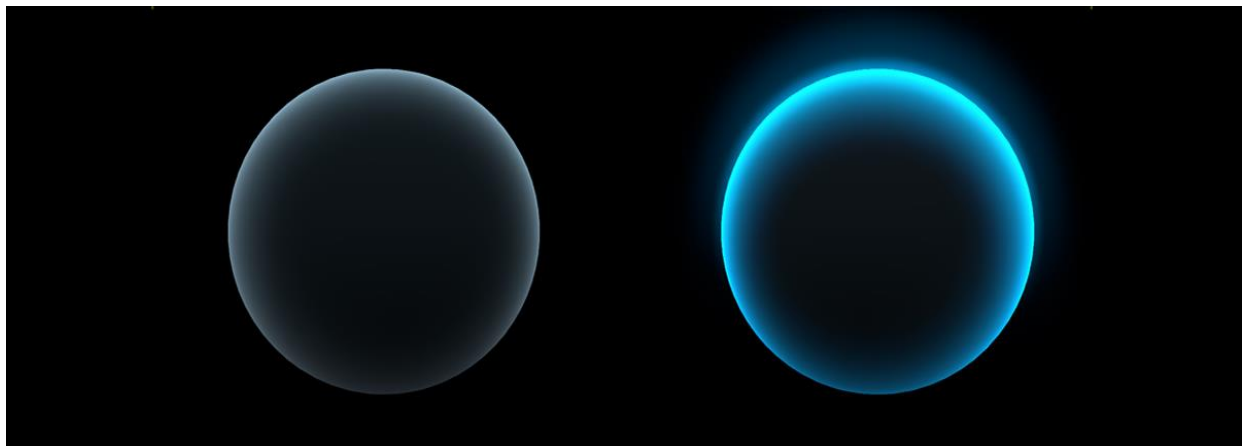
- **3D Rendering:** *Godot* koristi moderni *OpenGL ES 3.0* renderer za 3D grafiku. Podržava različite jezike za senčenje, uključujući VisualScript, ShaderScript i GDScript.
- **Sistem scena:** *Godot* koristi sistem scena gde se scene mogu sastojati od drugih scena, omogućavajući modularan i ponovljiv dizajn. Scene mogu sadržati 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 sudara 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 svetlosne mape 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 iskustava virtuelne stvarnosti koristeći ovaj pogon.

FRESNEL

Jedan od najboljih efekata za prostorno senčenje, fresnel, koristi se u 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 ćemo videti dolazeći sa te površine. Što je površina više okrenuta od nas, to ćemo videti više svetlosti (svetle ivice).



Slika 28: Frensel funkcije [12]



Slika 29: Efekti fresnel i frensel_glow funkcija [12]

ROTACIJA KINEMATIČKOG TELA

Ovaj GDScript kod upravlja rotacijom 3D objekta. Skripta koristi linearnu interpolaciju za glatku rotaciju prema ciljnom uglu na osnovu događaja (pritiskanje dugmića). Dugmići se skrivaju i ponovo pojavljuju u intervalu koji kontroliše tajmer. Može se uticati čak i na brzinu same rotacije pomoću promenljive *speed*.

```
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()
```

Slika 30: Rotiranje kinematičkog tela [13]

10. ZAKLJUČAK

Kroz analizu klona *Space Invaders* igrice razvijanog u *PyGame* biblioteci, uočavaju se ključni elementi koji čine uspešnu 2D igru. *PyGame* se pokazao kao snažan alat za brzu i jednostavnu implementaciju igara, pružajući širok spektar funkcionalnosti za manipulaciju grafičkim objektima i događajima.

Na primeru *Doom* igre je istražena prilagodljivost *PyGame* biblioteke za različite žanrove, dodajući elemente slobodnog kretanja i pucanja. Iako ograničena za prave 3D igre, ova biblioteka omogućuje stvaranje primamljivih igara sa raznolikim elementima.

Pregled razvoja 3D igara kroz *Godot* pogon naglasio je napredne mogućnosti alata otvorenog koda za stvaranje visokokvalitetnih 3D iskustava. *Godot* pruža pristup velikom broju resursa i podršku za *scripting* na različitim jezicima, otvarajući vrata kreativnosti i budućnosti igara u 3D prostoru.

U celini, ovaj rad pruža uvid u različite detalje razvoja igara, istražujući alate dostupne programerima i tako omogućuje razumevanje kreativnih vizija u domenu 2D i 3D igara.



Slika 31: Različite dimenzije igara [14]

11. LITERATURA

- [1] *Space Invaders*
- [2] *RetroStyle Games*, <https://retrostylegames.com/blog/are-2d-games-still-popular>
- [3] *Jazz Jackrabbit 2*
- [4] *StackExchange*, <https://gamedev.stackexchange.com>
- [5] *Adventurer*, <https://github.com/prateek271/Adventurer>
- [6] *RPG Playground*, <https://rpgplayground.com/>
- [7] *Wikipedia PyGame*, <https://en.wikipedia.org/wiki/Pygame>
- [8] *PyGame dokumentacija*, <https://www.pygame.org/docs>
- [9] *Moj PyGame projekat*, <https://github.com/MatijaMax/alien-invasion-game>
- [10] *CS 165*, <https://openequella.github.io>
- [11] *Doom-style-Game*, <https://github.com/StanimislavPetrovV/DOOM-style-Game/tree/main>
- [12] *Godotshaders*, <https://godotshaders.com/snippet/fresnel/>
- [13] *Godot 3D transformacije*, <https://forum.godotengine.org/t/how-to-rotate-a-3d-object-on-its-y-axis>
- [14] *3D Services India*, <https://www.3dservicesindia.com/>