# Proces izrade grafičke aplikacije u Pythonu

**Garmaz, Petar**

Sveučilište Josipa Jurja Strossmayera u Osijeku
Ekonomski fakultet u Osijeku
Preddiplomski studij Poslovne informatike

Petar Garmaz

# PROCES IZRADE GRAFIČKE APLIKACIJE U PYTHONU

Završni rad

Osijek, 2020.

Sveučilište Josipa Jurja Strossmayera u Osijeku
Ekonomski fakultet u Osijeku
Preddiplomski studij Poslovne informatike

Petar Garmaz

# PROCES IZRADE GRAFIČKE APLIKACIJE U PYTHONU

Završni rad

**Kolegij: Razvoj poslovnih aplikacija**

JMBAG: 0010218592
e-mail: pgarmaz@efos.hr

Mentor: doc. dr. sc. Domagoj Ševerdija

Osijek, 2020.

Josip Juraj Strossmayer University of Osijeku
Faculty of Economics in Osijek
Undergraduate study of Business informatics

Petar Garmaz

# PROCESS OF MAKING A GRAPHICAL APPLICATION IN PYTHON

Final paper

Osijek, 2020.

# IZJAVA

## O AKADEMSKOJ ČESTITOSTI, PRAVU PRIJENOSA INTELEKTUALNOG VLASNIŠTVA, SUGLASNOSTI ZA OBJAVU U INSTITUCIJSKIM REPOZITORIJIMA I ISTOVJETNOSTI DIGITALNE I TISKANE VERZIJE RADA

1. Kojom izjavljujem i svojim potpisom potvrđujem da je ZAVRŠNI (navesti vrstu rada: završni / diplomski / specijalistički / doktorski) rad isključivo rezultat osobnoga rada koji se temelji na mojim istraživanjima i oslanja se na objavljenu literaturu. Potvrđujem poštivanje nepovredivosti autorstva te točno citiranje radova drugih autora i referiranje na njih.
2. Kojom izjavljujem da je Ekonomski fakultet u Osijeku, bez naknade u vremenski i teritorijalno neograničenom opsegu, nositelj svih prava intelektualnoga vlasništva u odnosu na navedeni rad pod licencom *Creative Commons Imenovanje – Nekomercijalno – Dijeli pod istim uvjetima 3.0 Hrvatska.* (cc) BY-NC-SA
3. Kojom izjavljujem da sam suglasan/suglasna da se trajno pohrani i objavi moj rad u institucijskom digitalnom repozitoriju Ekonomskoga fakulteta u Osijeku, repozitoriju Sveučilišta Josipa Jurja Strossmayera u Osijeku te javno dostupnom repozitoriju Nacionalne i sveučilišne knjižnice u Zagrebu (u skladu s odredbama Zakona o znanstvenoj djelatnosti i visokom obrazovanju, NN br. 123/03, 198/03, 105/04, 174/04, 02/07, 46/07, 45/09, 63/11, 94/13, 139/13, 101/14, 60/15).
4. izjavljujem da sam autor/autorica predanog rada i da je sadržaj predane elektroničke datoteke u potpunosti istovjetan sa dovršenom tiskanom verzijom rada predanom u svrhu obrane istog.

**Ime i prezime studenta/studentice:** Petar Garmaz

**JMBAG:** 0010218592

**OIB:** 52496061863

**e-mail za kontakt:** petar.garmaz@gmail.com

**Naziv studija:** Poslovna Informatika

**Naslov rada:** Proces izrade grafičke aplikacije u Pythonu

**Mentor/mentorica diplomskog rada:** doc. dr. sc. Domagoj Ševerdija

U Osijeku, 2020. godine

Potpis Petar Garmaz

## Proces izrade grafičke aplikacije u Pythonu


## SAŽETAK

U ovom radu je prikazan proces izrade grafičke aplikacije u Pythonu. Ukratko su opisani korišteni resursi za izradu zadatka ovog rada (poput Pygame-a i Pygame GUI-a), zatim je opisana struktura, tj. arhitektura koda u kojoj je grafička aplikacija napisana (MVC) i s time sami kod i od čega se sastoji. Na kraju su prikazani rezultati zadatka završnog rada, u kojem se pokazuje kako sama aplikacija funkcionira.



**Ključne riječi:** Python, MVC, Pygame, Pygame GUI


## Process of making a graphical application in Python


## ABSTRACT

This paper shows the process of making a graphical application in Python. This paper has described couple of different resources that were used in the making of this papers' assignment (such as Pygame and Pygame GUI), after that the paper describes the structure or architecture of the code in which the graphical aplication was made (MVC) as well as the code itself and what it consists of. In the end the paper shows couple of results of the assignment, which demonstrates how the whole application works.



**Keywords:** Python, MVC, Pygame, Pygame GUI

# SADRŽAJ

# 1. UVOD

Izrada grafičkih aplikacija je složen proces, kako vremenski tako i u smislu uloženog truda i vještine razvijača grafičkih aplikacija. Glavna komponenta grafičke aplikacije, GUI (engl. Graphical User Interface), zahtjeva veliku razinu razumijevanja, kako izrade programskih aplikacija tako i određene razine umjetničkog razumijevanja za izradu dizajna same aplikacije.

Svrha ovog rada je pokazati proces izrade grafičke aplikacije u Python programskom jeziku kroz primjer koji je također bio i zadatak ovog završnog rada. Cilj ovog rada je obići različite procese koji se moraju proći kako bi se uspješno napravila grafička aplikacija te korišteni alati, tj. moduli koji su pomogli u izradi grafičke aplikacije.

Sami zadatak rada je bio napraviti grafičku aplikaciju, ili u ovom slučaju video igru, korištenjem Python programskog jezika. Kako bi sami zadatak bio lakši za izradu, korišten je Pygame modul za samu logiku video igre i Pygame GUI za grafičko sučelje video igre.

## 2. Teorijska podloga i prethodna istraživanja

Razvijanje i izrada velikih i kompleksnih grafičkih aplikacija je zahtjevan posao. Razvijači i programeri moraju obraćati pozornost na razne probleme u izradi grafičkih aplikacija. Kako bi programerima bilo lakše, postoje različiti načini na koje se grafičke aplikacije mogu izrađivati. Različita postojeća istraživanja predlažu korištenje MVC (engl. Model-View-Controller) arhitekture i PAC (engl. Presentation-Abstraction-Control) arhitekture kao obrazac za izradu pravilne strukture grafičke aplikacije. (Karagkasidis, 2020.)

MVC i PAC su slične u smislu da odvajaju kod na tri različita dijela. No za razliku od MVC-a, PAC se koristi kao hijearhijska struktura „agenata“, koji se sastoje od 3 dijela: prezentacija, apstrakcija i kontrola. „Agenti“ međusobno komuniciraju samo kroz kontrolni dio u hijearhiji. Isto tako se razlikuje od MVC-a u tome što svaki agent potupno izolira prezentaciju (slično „pogledu“ u MVC-u) i apstrakciju (slično „modelu“ u MVC-u). Ovaj rad je napravljen u MVC arhitekturi zbog lakšeg korištenja te zbog jednostavnosti implementacije. Zbog relativno niske kompleksnosti aplikacije, implementacija PAC arhitekture jednostavno ne bi išla u korist aplikaciji.

Također kod primjene ovih obrasca u praksi se mogu proizaći različiti problemi do čijih rješenja programeri moraju sami doći.

# 3. Metodologija rada

Za izradu ovog završnog rada je korištena metoda deskripcije. Deskripcija je opis pojava koje se istražuju. Svako istraživanje bi trebalo započeti s deskripcijom svih temeljnih pojmova ili pojava. U znanstvenom radu velika pozornost posvećuje se detaljnom opisivanju činjenica, pojava ili podataka kako bi se povećala objektivnost i točnost u svim drugim fazama istraživanja. (Žugaj i dr., 2006).

Glavna hipoteza ovog rada jest testiranje implementabilnosti video igre u MVC okruženju. Verifikaciju hipoteze temeljimo na izrađenoj funkcionalnoj aplikaciji koji implementira sve postavljene zahtjeve aplikacije unatar MVC paradigme. Temeljito osiguravanje kvalitete aplikacije prema standardima razvoja softvera bi bila završna etapa verifikacije ove hipoteze, ali zbog obujma rada ovaj segment je ispušten.

Glavni izvori podataka za opis ovog rada je korišten zadatak završnog rada, te ostali izvori literature koji su korišteni sa samu izradu tog zadatka, poput dokumentacije Pygame-a i PygameGUI-a.

Tijekom izrade zadatka završnog rada najviše poteškoća se pronalazilo kod pisanja koda za igrača, no takvi problemi su nastajali najčešće zbog nerazumijevanja dokumentacije Pygame-a te su se ti problemi sami sebe riješavali nakon određenog vremena i novonastalog iskustva.

# 4. Opis istraživanja i rezultati istraživanja

## 4.1. Opis zadatka završnog rada

Zadatak završnog rada je video igra u kojoj igrač, tj. igračev tenk mora uništiti što više neprijatelja, tj. neprijateljskih tenkova, što je moguće. Igrač kontrolira svoj tenk, tako da sa tipkama „W“ i „S“ pomiče svoj tenk unaprijed ili unazad te tipkama odnosno „A“ i „D“ rotira svoj tenk prema desno ili lijevo. Kupola tenka se automatski pomiče na onu poziciju na kojoj se nalazi pokazivač miša. Lijevim klikom miša igrač ispaljuje glavni projektil, koji ima sporiju brzinu projektila te sporiju brzinu punjenja, ali radi veću štetu. Desnim klikom miša igrač ispaljuje sporedni projektil koji je nešto brži od glavnog projektila te ima puno bržu brzinu punjenja, ali radi manje štete. Video igra se igra po rundama. Prije početka prve runde, generira se polje sa nasumičnim brojem zaklona (kamenje, kuće, itd...). Svaka runda završava kad igrač uništi sve neprijateljske tenkove na polju. Pet sekundi nakon završetka runde počinje nova runda, gdje se stvara jedan tenk više nego u prijašnjoj rundi, i tako u beskonačnost, sve dok neprijatelji ne unište igrača. Igrač također ima mogućnost popravka svog tenka, tako da pokupi kutije za popravljanje, koje imaju 50% šanse da se stvore na početku svake runde.

Implementacija projekta nalazi se u cijelosti na GitHub platformi koje se može preuzeti na: https://github.com/PetarGarmaz/Thunder-Run-Game (Prilog 1. Zadatak završnog rada, Thunder Run)

## 4.2. Izrada aplikacije

### 4.2.1. Pygame

Pygame je set Pythonovih modula koji su dizajnirani za lakše programiranje video igara. Pygame omogućuje izrađivanje potpuno opremljenih, u smislu funkcionalnosti, video igara i ostalih multimedijalnih programa u Python programskom jeziku (Pygame, 2020.). Pygame je također vrlo portabilan te se može pokretati na skoro svakoj platformi i operacijskom sustavu.

Pygame sadrži par značajki zbog kojih je jedan od najboljih modula za izradu video igara u Pythonu, a to su:

- Jednostavnost – jednostavan i lagan za korištenje, savršeno za početnike u Pythonu
- Pouzdanost – mnogo se video igara objavilo, koje su koristile Pygame, što znači da je sami modul bio testiran par miliona puta
- Kontrola – mogućnost kontrole glavne petlje te veća kontrola kod korištenja drugih riječnika
- Kompaktnost – pygame ne zahtijeva stotine i tisuće linija koda kako bi program radio neke sitnice, jezgra pygame-a je jednostavna te dodatne stvari poput GUI riječnika i različitih efekata se mogu nadodati posebno izvan igre
- Modularnost – pygame omogućava posebno korištenje zasebnih dijelova modula

Pygame se može instalirati korištenjem windows instalacije na web stranici modula ili upisivanjem sljedeće linije u komandnu ploču windowsa:

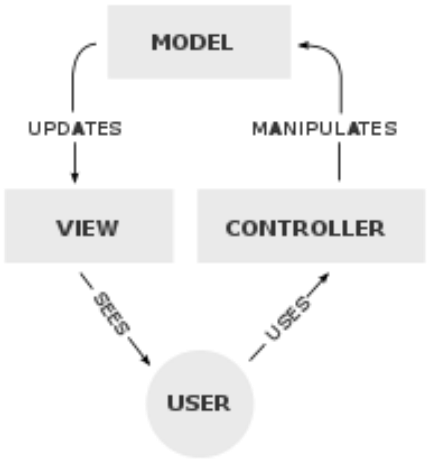
```
pip install pygame
```

Kako bi mogli koristiti Pygame, u kod se još mora dodati i sljedeća linija koda:

```
import pygame
```

### 4.2.2. MVC

Video igra je napravljena u MVC (engl. Model-view-controller) arhitekturi, koja se često koristi u izradi GUI-a. MVC služi kako bi se razdvojili interni prikazi informacija od načina na koje su te informacije prikazane korisniku. MVC se koristi kako bi se olakšao nezavisan razvoj, testiranje i održavanje određene aplikacije.




***Slika 1:*** *Struktura MVC arhitekture*

MVC se sastoji od 3 cjeline:

- Model (Model) – logika određene aplikacije, sastoji od podataka, poslovnih pravila, logike i funkcija ugrađenih u programsku logiku
- Pogled (View) – prikaz modeliranih podataka, tj. bilo kakav prikaz podataka kao što je tablica, obrazac ili dijagram
- Upravitelj (Controller) – upravljanje korisničkim zahtjevima, tj. prihvaća unos korisnika i pretvara ga u naloge modelu ili pogledu

### 4.2.3. Programski kod aplikacije

Aplikacija se sastoji od sveukupno 4 modula (Main.py, Player.py, Enemy.py, Enviroment.py). Glavni modul je Main.py kroz koju se pokreće cijela aplikacija i koja učitava sve informacije i podatke iz ostalih skripti. Na kraju samog modula se nalazi provjera da li korišteni Python interpreter izvodi Main.py modul kao glavni program. Ona služi kako bi se pokrenula jedna ili više linija koda samo ako je otvorena datoteka pokrenuta direktno, a ne ako je uvezena.

#### 4.2.3.1. Main.py modul

Main.py modul je glavni modul ove grafičke aplikacije. On se sastoji od 3 glavne klase, a to su GameController(), GameModel() i GameView().

```
import pygame
import pygame_gui #pip install pygame_gui
from pygame.math import Vector2
import math
import random

from Player import PlayerProjectile120mm, PlayerProjectile30mm, Player, Pla
yerTurret, PlayerHealthSprite, PlayerReloadBar120mm, PlayerReloadBar30mm, R
epairCrate
from Enemy import EnemyProjectile, Enemy, EnemyTurret, EnemyHealthSprite
from Enviroment import EnviromentSprite

class GameModel():
    # (...)

class GameView():
    # (...)

class GameController():
    # (...)


if __name__ == "__main__":
    GameController()
    pygame.quit()
```

***Slika 2:*** *Isječak Main.py skripte sa korištenim uvozima i korištenim klasama*

U inicijalizaciji klase GameController() se nalazi Pygameov inicijalizator, pygame.init(), što inicijalizira sve uvezene Pygameove module te se isto tako nalaze 2 varijable s kojima inicijaliziramo GameModel() i GameView(), također sadrži par ostalih varijabli koje se samo koriste kao provjere. GameController() se sastoji od 5 metoda:

- MainMenuEventHandler() – rukuje sa svim eventima, ili događajima unutar glavnog izbornika, poput pritiskanja gumbova, korištenja klizača, itd...
- GameEventHandler() – rukuje sa svim događanjima unutar igre, poput pokretanja ili rotiranja igrača
- UpdateMainMenu() – koristi se kako bi se za svaku skličicu (engl. Frame) ažurirale sve glavne metode koje se koriste unutar glavnog izbornika, poput ažuriranja statusa gumbova (da li su sakriveni ili ne, da li su se izbrisali ili ne, itd...)
- UpdateGame() – koristi se kako bi se za svaku sličicu ažurirale glavne metode koje se koriste unutar same igre, poput ažuriranja pozicije ili rotacije igrača, neprijatelja, njihovih životnih bodova, itd...
- NumToString() - koristi se za pretvorbu broja u niz znakova za odabir teme razine igre unutar glavnog izbornika

U inicijalizaciji klase GameModel() se nalaze varijable sa kojima su se definirale grupe tzv. Sprajtova (engl. Sprite). One služe kako bi se različiti sprajtovi, poput igračevog oklopa, igračevog topa, okoliša, itd. mogli zasebno dohvatiti za različite radnje. Također se nalaze varijable za definiranje igrača, njegove početne pozicije u prostoru, kojoj grupi sprajtova on pripada, itd. Nakon toga, klasa još sadrži i varijable koje se koriste za provjeru koliko neprijatelja ima na polju, varijabla za odbrojavanje runde, varijabla za odabir teme te varijable za mjerenje vremena. Klasa se sastoji od 4 metoda:

- GameEnviroment()
- GameEnemySpawner()
- GameLogic()
- GameDraw()

GameEnviroment() metoda se koristi za stvaranje igračeve okoline, poput kuća ili kamenja, kako bi ih igrač mogao koristiti kao zaklon od neprijateljskih projektila.

```
def GameEnviroment(self):
        if self.spawnEnviroment == True:

            numOfDecor = 3
            currentObject = None
            distanceBetweenObjects = 0

            for i in range(numOfDecor):
                spawnObject = False
                randomObject = random.randint(0,1)

                while spawnObject == False:
                    x = random.randint(100, 700)
                    y = random.randint(100, 500)

                    randomRotation = random.choice([0, 90, 180, 270])

                    if(currentObject == None):
                        currentObject = (EnviromentSprite((x, y), self.them
e, randomObject, randomRotation, self.allSprites, self.enviroment))
                        spawnObject = True
                    else:
                        distanceBetweenObjects = math.hypot(x - currentObje
ct.position.x, y - currentObject.position.y)

                        if(distanceBetweenObjects > 200 and distanceBetween
Objects < 500):
                            currentObject = (EnviromentSprite((x, y), self.
theme, randomObject, randomRotation, self.allSprites, self.enviroment))
                            spawnObject = True


            self.spawnEnviroment = False
```

***Slika 3:*** *Isječak metode GameEnviroment()*

Pošto se ova metoda odvija nakon svake sličice, potrebno je staviti provjeru kako bi se ta metoda odvila samo jednom. Nakon te provjere postavljene su 3 početne varijable koje će se kasnije koristiti u metodi te jednu „for“ petlju koja će se vrtiti onoliko puta koliko je definirano u varijabli „numOfDecor“. U toj petlji se nalaze 2 nove varijable, a to su „spawnObject“ (varijabla tipa boolean, provjerava hoće li se stvoriti neki objekat) i „randomObject“ (nasumična cjelobrojna varijabla koja definira kojeg će tipa biti neki objekat – ako je varijabla

0, onda će objekat biti kamen, a ako je varijabla 1 onda će stvoreni objekat biti kuća). Ispod toga postoji druga petlja, tj. „while“ petlja, koja će se vrtiti sve dok se objekat ne stvori pod određenim uvjetima. Ti uvjeti su definirani ispod te petlje, a to su udaljenost između sadašnjeg i prijašnjeg objekta koja se računa korištenjem metode „math.hypot“ (dužina hipotenuze pravokutnog trokuta). Kad su ti uvjeti zadovoljeni stvorit će se novi objekt klase EnviromentSprite() kojem dodjeljujemo nasumično generiranu poziciju, temu koju je igrač dodijelio u glavnom izborniku, nasumičan tip objekta, nasumičnu rotaciju te kojim grupama sprajtova on pripada. Nakon toga varijabli „spawnObject“ dodijeljujemo vrijednost True kako bi prekinuli „while“ petlju te nastavili sa „for“ petljom. Nakon stvorenih „numOfDecor“ broja objekata, „for“ petlja će se prekinuti te će se varijabli „spawnEnviroment“ dodijeliti vrijednost False, što znači da aplikacija više neće stvarati dodatne objekte.

Metoda GameEnemySpawner() funkcionira na sličnom principu kao EnviromentSpawner() te se koristi za stvaranje neprijatelja i kutija za popravljanje igračevog tenka nakon gotove runde, tj. nakon što se svi neprijatelji na polju unište.

```
def GameEnemySpawner(self):
        repairCrateChance = random.randint(0, 1)

        if(len(self.enemies) <= 0):
            self.roundStart += 1/60
        else:
            self.roundStart = 0

        if(self.roundStart >= 5):
            if(repairCrateChance == 1):
                self.SpawnCrate()

            for i in range(self.enemyNum):
                x = random.choice([0, 800])
                y = random.choice([0, 600])

                newX = random.randint(0, 800)
                newY = random.randint(0, 600)

                fireTiming = random.randint(1, 6)

                #Generate enemies
                enemy = Enemy((x, y), (newX, newY), self.player, self.allSp
rites, self.enemies)
                enemyTurret = EnemyTurret(enemy, self.player, fireTiming, s
elf.allSprites, self.enemyTurrets, self.enemyProjectiles)

                enemyHealthBar = EnemyHealthSprite(enemy, self.bars)

            self.enemyNum += 1
```

***Slika 4**: Isječak metode GameEnemySpawner()*

Metoda počinje sa varijablom koja određuje šansu stvaranja kutije za popravak (ako je varijabla jednaka nuli, kutija se neće stvoriti, ako je varijabla jednaka jedinici, kutija će se stvoriti, tj. kutija ima 50% šanse za stvaranje). Nakon toga je postavljena „for“ petlja koja će se vrtiti onoliko puta koliko se neprijatelja treba stvoriti (na početku igre je to samo jedan, no nakon svake runde taj se broj povećava za 1). Zatim, određujemo gdje će se taj neprijatelj stvoriti (x i y varijable) te u kojem će se pravcu kretati (newX i newY varijable).

Treća metoda u klasi je metoda GameLogic() gdje se sva logika aplikacije, znači koji će uzrok biti nekog događaja. Većinom se tu nalaze samo provjere kolizije, ili sudaranja, npr. da li je igrača pogodio neprijateljski projektil, ako je oduzmi mu životne bodove.

Zadnja metoda u klasi je metoda GameDraw() koja prima „view“ klasu koju je GameController() klasa poslala. Ova metoda se koristi za crtanje svih sprajtova i ostalih elemenata na zaslon nakon svake sličice.

### 4.2.3.2. Player.py i Enemy.py modul

Player.py modul se sastoji od sveukupno 7 klasa, a Enemy.py modul se sastoji od 4 klasa. Modul je korišten u svrhu kako bi se grupirale sve klase vezane uz igrača u jedan modul. Svaka klasa označava jedan sprajt, pošto te klase nasljeđuju varijable Pygameovog modula Sprite te ne mogu imati više od jednog sprajta u klasi. Svaka klasa se sastoji od 2 obvezne metode: init() i update() metode. U init() metodi se inicijaliziraju varijable koje su nasljeđene od Pygameovog Sprite modula, onda se definira izgled tog sprajta, tj. učitava se korištena slika, njegova pozicija na zaslonu i ostale varijable koje su bitne za tu klasu. Update() metoda je metoda koja se učitava svake sličice, u toj metodi se najčešće nalaze varijable koje se trebaju mijenjati svake sličice, poput pozicije, životnih bodova, trenutni kut kretanja, itd. Pošto su Player.py i Enemy.py moduli relativno slični, rad će proći samo kroz Player.py modul. Najbitnije i najkompleksnije metode u ovom modulu su klasa Player() i klasa PlayerTurret() te one definiraju izgled igračevog tenka, njegovo područje pogotka (engl. Hitbox), njegovi životni bodovi, smjer kretanja, smjer nišanjenja, itd.

Igrač pomiče svoj tenk pomoću metode Movement() u klasi Player(), gdje se računaju sljedeće varijable:

- Trenutni smjer (izražen u vektorima) – računa se pomoću funkcije rotate_ip (engl. rotate in place), gdje se uzima trenutna brzina rotacije (kontrolirana tipkama „A“ i „D“ na tipkovnici) te se rotira trenutni smjer za određenu rotacijsku brzinu po sekundi (ako je rotacijska brzina 3, onda će se smjer rotirati za 3 stupnjeva po jednoj sličici)
- Trenutni kut (izražen u stupnjevima) – računa se tako da se za svaku sličicu zbroji sa rotacijskom brzinom, koristi se u rotiranju samog sprajta

- Trenutna korištena slika – trenutna slika se rotira za onoliko stupnjeva, koliko je definirano u varijabli trenutnog kuta te se ista slika sprema u novu sliku koja se onda crta na zaslonu
- Nova pozicija – nova pozicija se računa tako da se uzme trenutna pozicija te se zbroji sa trenutnim smjerom kretanja koji je pomnožen sa brzinom kretanja (koja se kontrolira sa tipkama „W“ i „S“ na tipkovnici); Nova pozicija se također koristi kako bi se moglo provjeriti hoće li igrač izaći iza zaslona, u slučaju da izađe iz zaslona, trenutna pozicija se neće promijeniti, te će tako igrač biti forsiran da ostane unutar zaslona.

Player() klasa također sadrži manje metode poput metode PlayerStats(), koja mjeri koliko je životnih bodova ostalo kod igrača te ukoliko je igrač izgubio sve svoje životne bodove, njegov sprajt nestaje, te metode PlaySound() koja služi kako bi pustio određeni zvuk ako se igrač pomakne, ili ako stoji na mjestu.

```
class Player(pygame.sprite.Sprite):
    def __init__(self, pos, allSprites, players):
        # (...)


    def update(self):
        # (...)

    def Movement(self):
        self.dir.rotate_ip(self.rotationSpeed)
        self.currentAngle += self.rotationSpeed

        self.image = pygame.transform.rotate(self.original_image, -
self.currentAngle)
        self.rect = self.image.get_rect(center=self.rect.center)

        newPosition = self.position + self.dir * self.movementSpeed

        if(newPosition.x > 0 and newPosition.x < 800 and newPosition.y > 0 and newP
osition.y < 600):
            self.position += self.dir * self.movementSpeed
            self.rect.center = self.position

    def PlaySound(self):
        # (...)

    def PlayerStats(self):
        # (...)
```

***Slika 5:*** *Isječak metode Movement() u klasi Player()*

PlayerTurret() klasa se sastoji od 3 glavne metode: UpdatePosition(), Rotation(), i Fire().

UpdatePosition služi samo kako bi se pozicija topa ažurirala prema poziciji igračevog tenka.

Rotation metoda služi kako bi se top rotirao prema poziciji pokazivača miša. To se radi tako da se izračuna kut pravca (čija je točka A pozicija tenka, a točka B pozicija pokazivača miša na zaslonu) tako da se računa $\theta = \tan^{-1}(\frac{yB-yA}{xB-xA})$ , tj. računa se arkus tangens od omjera razlike y točke „A“ i y točke „B“, i razlike x točke „A“ i x točke „B“. Dobiveni kut se uzima te koristi kako bi se rotirala korištena slika koja predstavlja top.

Fire metoda provjerava koju tipku miša je igrač pritisnuo te ovisno da li je lijeva ili desna tipka miša, top će respektivno ispaliti jači ili slabiji projektil.

```
def UpdatePosition(self):

        self.position = self.hullPos
        self.rect.center = self.hull.rect.center

    def Rotate(self):
        x, y = pygame.mouse.get_pos()

        self.currentAngle = math.atan2((y - self.hullPos.y), (x - self.hull
Pos.x))
        self.currentAngle = math.degrees(self.currentAngle)

        self.image = pygame.transform.rotate(self.original_image, - self.cu
rrentAngle - 90)
        self.rect = self.image.get_rect(center=self.rect.center)

    def Fire(self):
        mouse_pressed = pygame.mouse.get_pressed()

        if(mouse_pressed[0] and self.cooldown == 7):
            self.cooldown = 0
            PlayerProjectile120mm(self.position, self.currentAngle, self.al
lSprites, self.projectiles)

        if(mouse_pressed[2] and self.cooldown2 == 0.5):
            self.cooldown2 = 0
            PlayerProjectile30mm(self.position, self.currentAngle, self.all
Sprites, self.projectiles)
```

***Slika 6:*** *Isječak klase PlayerTurret(), metode UpdatePosition(), Rotate(), Fire()*

## 4.3. Izrada GUI-a

### 4.3.1. Pygame GUI

Pygame GUI je modul koji pomaže pri izradi grafičkog korisničkog sučelja, tj. GUI-a (engl. Graphical User Interface) u video igrama napravljenim u Pygameu. Trenutna inačica Pygame GUI-a je namijenjena samo za Pygame 2 i Python 3.

Trenutne značajke Pygame GUI-a uključuju:

- Samostalno dizajniranje izgleda – mogućnost promjene izgleda cijelog korisničkog sučelja pomoću JSON (engl. JavaScript Object Notation) datoteka u kojoj se mogu definirati boje, izgled fonta, veličina fonta, itd.
- Potpora za oblikovanje teksta – podržano korištenje HTML-a za oblikovanje teksta
- Velik izbor različitih elemenata – podržani su svakaki elementi korisničkog sučelja, poput gumbova, klizača, prozora s unosom teksta, „drop-down“ izbornika, itd.
- Poštivanje načina funkcioniranja Pygamea

Sve vezano uz Pygame GUI, u kodu, se nalazi u klasi GameView, u čijoj Init() metodi se definiraju svi korišteni elementi korisničkog sučelja poput gumbova, klizača i tekstnih okvira. Za svaki element obvezno je definirati dimenzije tog elementa (pozicija na x osi, pozicija na y osi, širina, visina) i koji je glavni „menadžer“ tog elementa. Ostale opcije tog elementa su neobavezne, poput teksta, „sidra“ (tj. engl. Anchor) i kontejnera (engl. container).

Kako bi uspješno postavili Pygame GUI, na početku je obvezno inicijalizirati „menadžera“ GUI-a. Njega spremamo u posebnu varijablu koju koristimo kasnije kod inicijaliziranja elemenata GUI-a. Nakon toga je potrebno stvoriti vaijablu „Clock“ koja prati koliko je sekundi prošlo između svake petlje programa. Također se mora postaviti i „deltaTime“ varijabla koja se koristi kao brojač kojeg koriste neki GUI elementi.

```
def MainMenuEventHandler(self):

        self.model.deltaTime = self.model.clock.tick(60)/1000

        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                self.menuDone = True
                self.done = True

            if event.type == pygame.USEREVENT:
                if event.user_type == pygame_gui.UI_BUTTON_PRESSED:
                    if event.ui_element == self.view.startButton:
                        # (...)

                    elif event.ui_element == self.view.optionsButton:
                        # (...)

                    elif event.ui_element == self.view.quitButton:
                        self.done = True

                    if event.ui_element == self.view.backToMenuButton:
                        # (...)

                if event.user_type == pygame_gui.UI_HORIZONTAL_SLIDER_MOVED:
                    if event.ui_element == self.view.healthSlider:
                        # (...)
                    elif event.ui_element == self.view.movementSlider:
                        # (...)
                    elif event.ui_element == self.view.themeSlider:
                        # (...)

            self.view.guiManager.process_events(event)
```

***Slika 7:*** *Isječak metode MainMenuEventHandler()*

### 4.3.2. Izrada glavnog izbornika

Glavni izbornik se sastoji od 3 GUI elemenata, tj. gumbova, i pozadine. Pozadina je napravljena prije samog koda u programu GIMP, kako bi se olakšalo sami proces izrade izbornika, tako da se ne mora izrađivati različite linije, tekstni okviri i ostali ukrasi preko koda. Gumbovi koji se koriste u glavnom izborniku su:

- Start game – gumb za pokretanje igre
- Options – gumb za postavljanje igre
- Quit game – gumb za izlaz

Prilikom pritiskanja gumba „Start game“ pokrenut će se igra, s tim da se prije samog pokretanja određene varijable promijene. Te varijable uključuju temu igre (travnata, pustinjska, snježna tema) i igračevi maksimalni životni bodovi (50-200 životnih bodova), tj. varijable koje se podešavaju u opcijama na glavnom izborniku.

```
if event.ui_element == self.view.startButton:

      self.model.theme = self.view.themeSlider.get_current_value()

      if(self.model.theme == 0):
            self.view.background = self.view.grassBG
      elif(self.model.theme == 1):
            self.view.background = self.view.desertBG
      elif(self.model.theme == 2):
            self.view.background = self.view.snowBG

      self.model.player.maxHealth = self.view.healthSlider.get_current_value()
      self.model.player.health = self.model.player.maxHealth

      self.menuDone = True
      self.UpdateGame()
```

***Slika 8**: Isječak provjere pritisnutod "Start game" gumba*

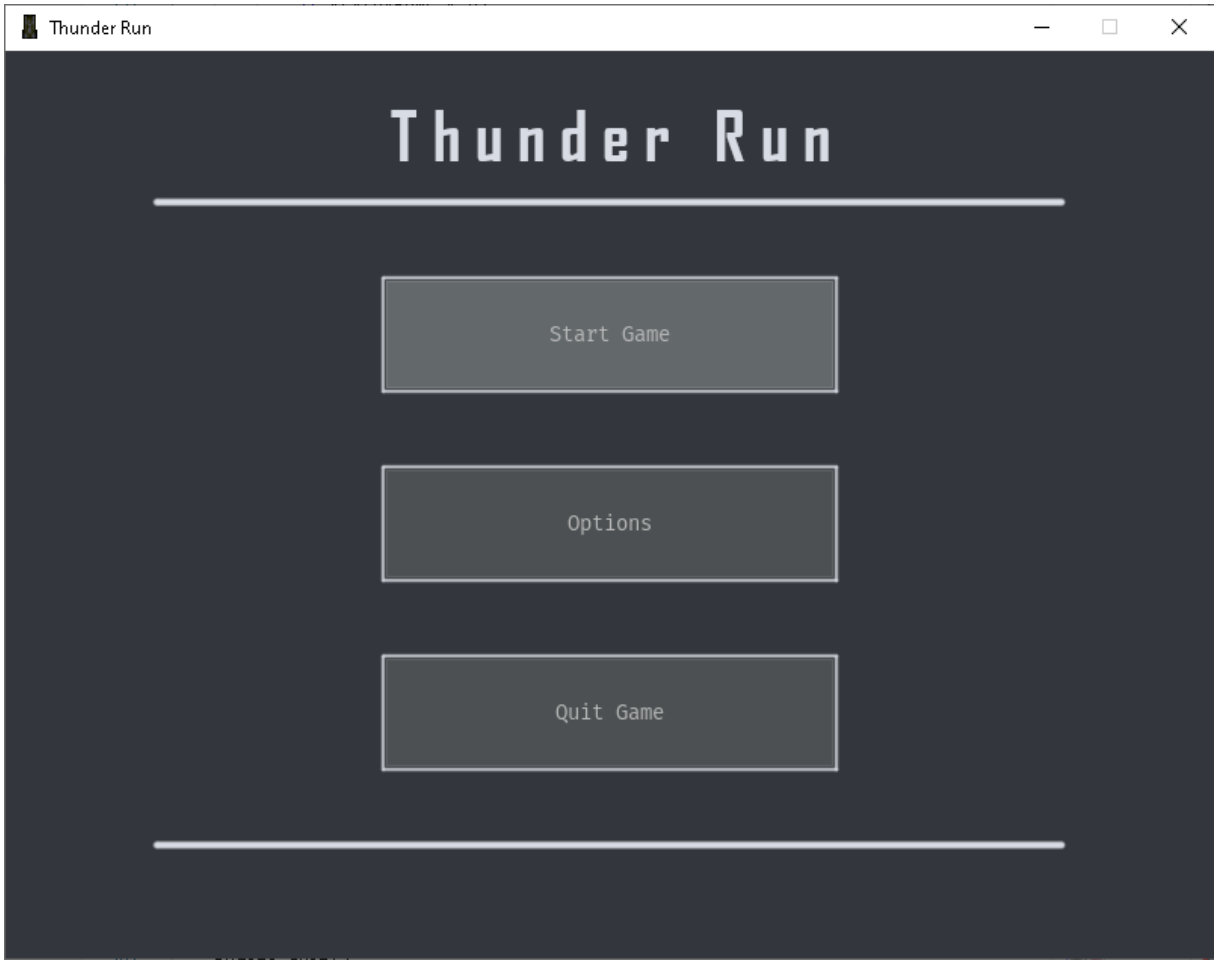
Pritiskom gumba „Options“ se otvaraju opcije za postavljanje igre. Opcije se otvaraju tako da se svi gumbovi korišteni u glavnom izborniku sakriju pomoću hide() funkcije, da se pokažu svi gumbovi, klizači i tekstualni okviri pomoću show() funkcije te da se promijeni pozadina.

```
elif event.ui_element == self.view.optionsButton:

      self.view.menuBackground = self.view.optionsBackground
      self.view.startButton.hide()
      self.view.optionsButton.hide()
      self.view.quitButton.hide()
      self.view.healthSliderNumber.show()
      self.view.movementSliderNumber.show()
      self.view.themeSliderNumber.show()
      self.view.healthSlider.show()
      self.view.movementSlider.show()
      self.view.themeSlider.show()
      self.view.backToMenuButton.show()
```

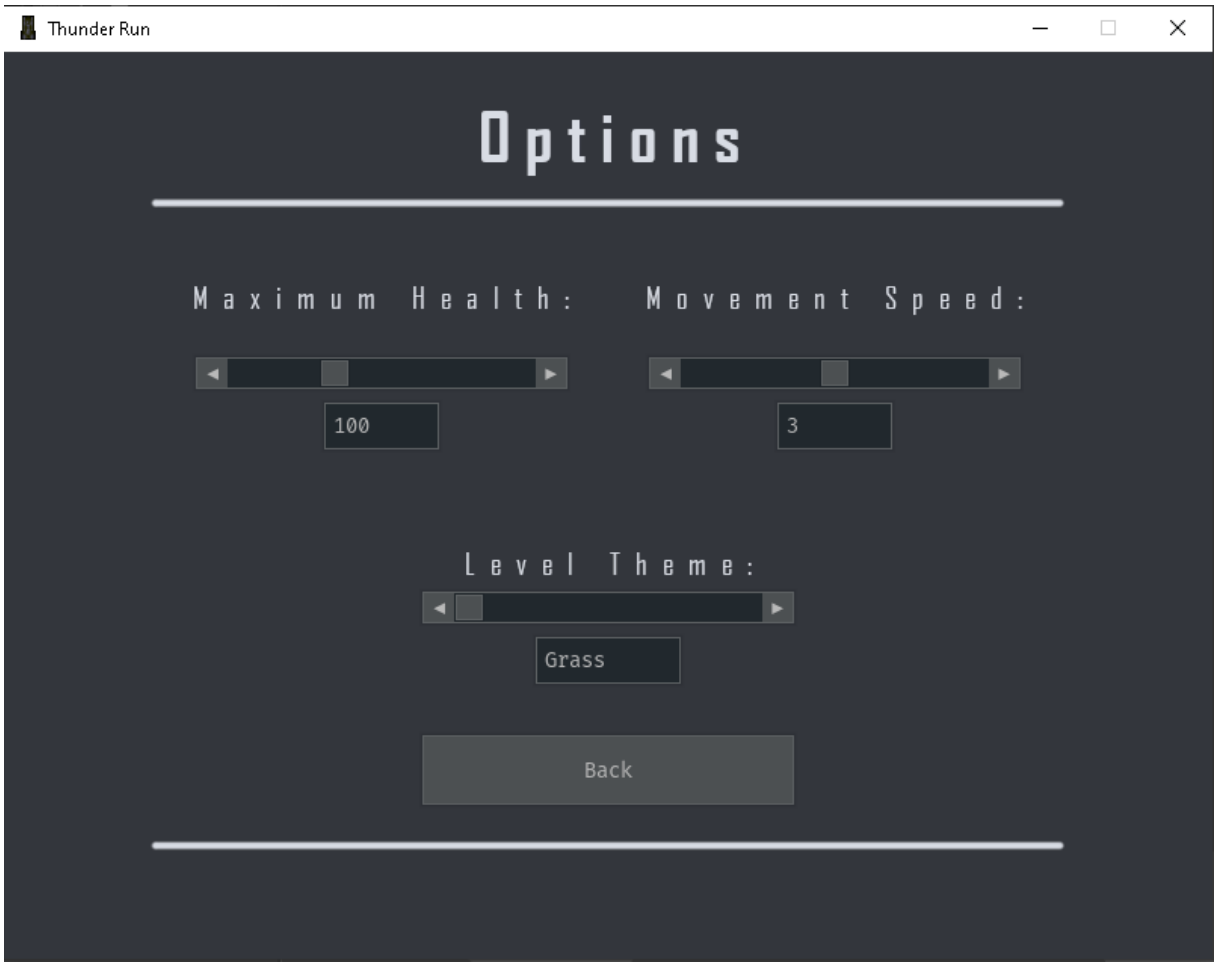
***Slika 9:*** *Isječak provjere pritisnutog "Options" gumba*

## 4.4. Rezultat



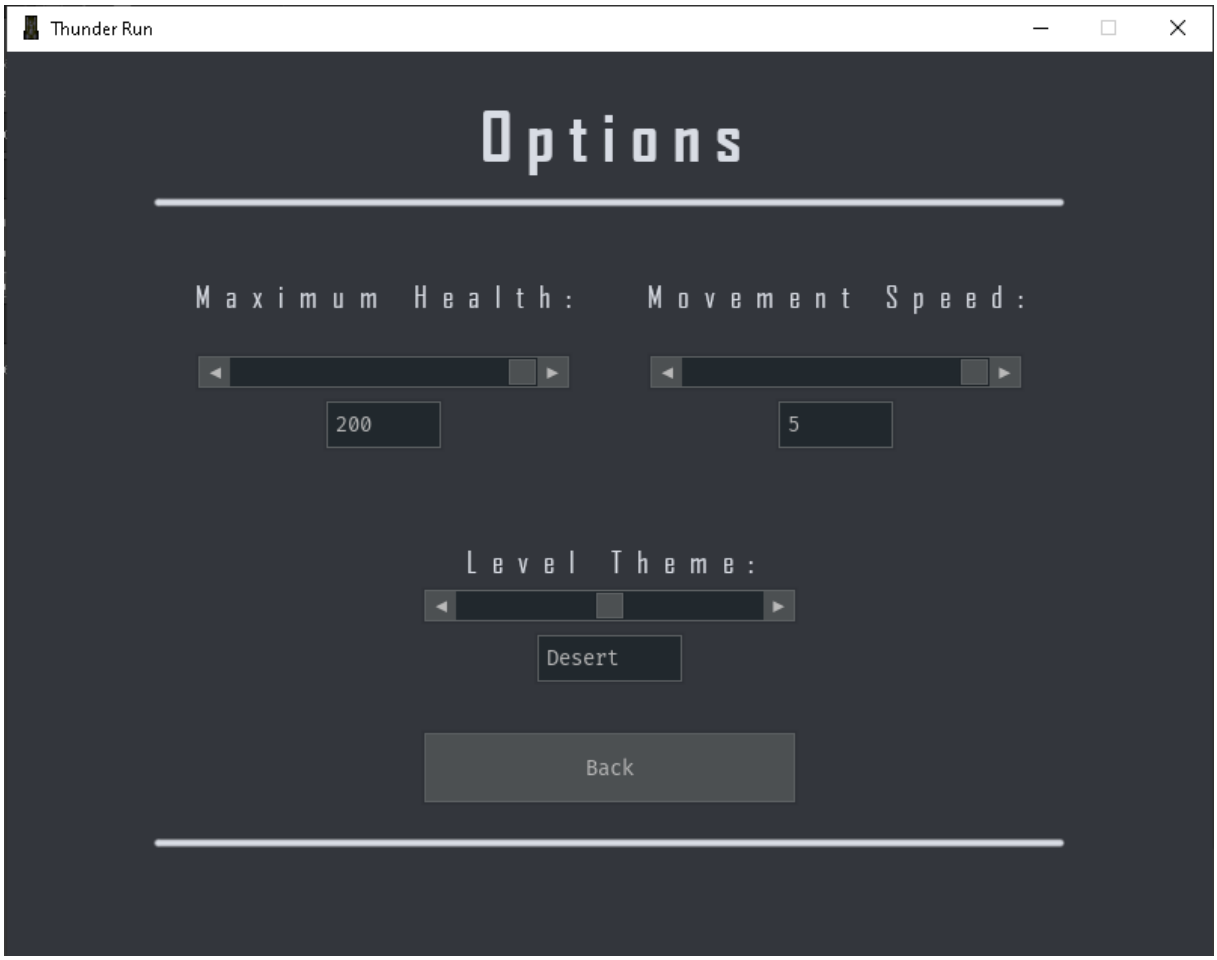

***Slika 10:*** *Prikaz glavnog izbornika*

Na slici iznad je prikazan rezultat glavnog izbornika, vidljiva su 3 elementa GUI-a (gumbovi) i pozadina. Klikom na gumb „Options“ će se prikazati sljedeći prikaz.




***Slika 11:*** *Prikaz opcija za postavljanje igre*

Sad je moguće postaviti igru po volji igrača, ako želi da video igra bude lakša, može si postaviti da mu je maksimalni život veći ili da mu brzina kretanja bude viša. Isto tako je i moguće postaviti temu igre. Promjene su vidljive na idućoj slici.

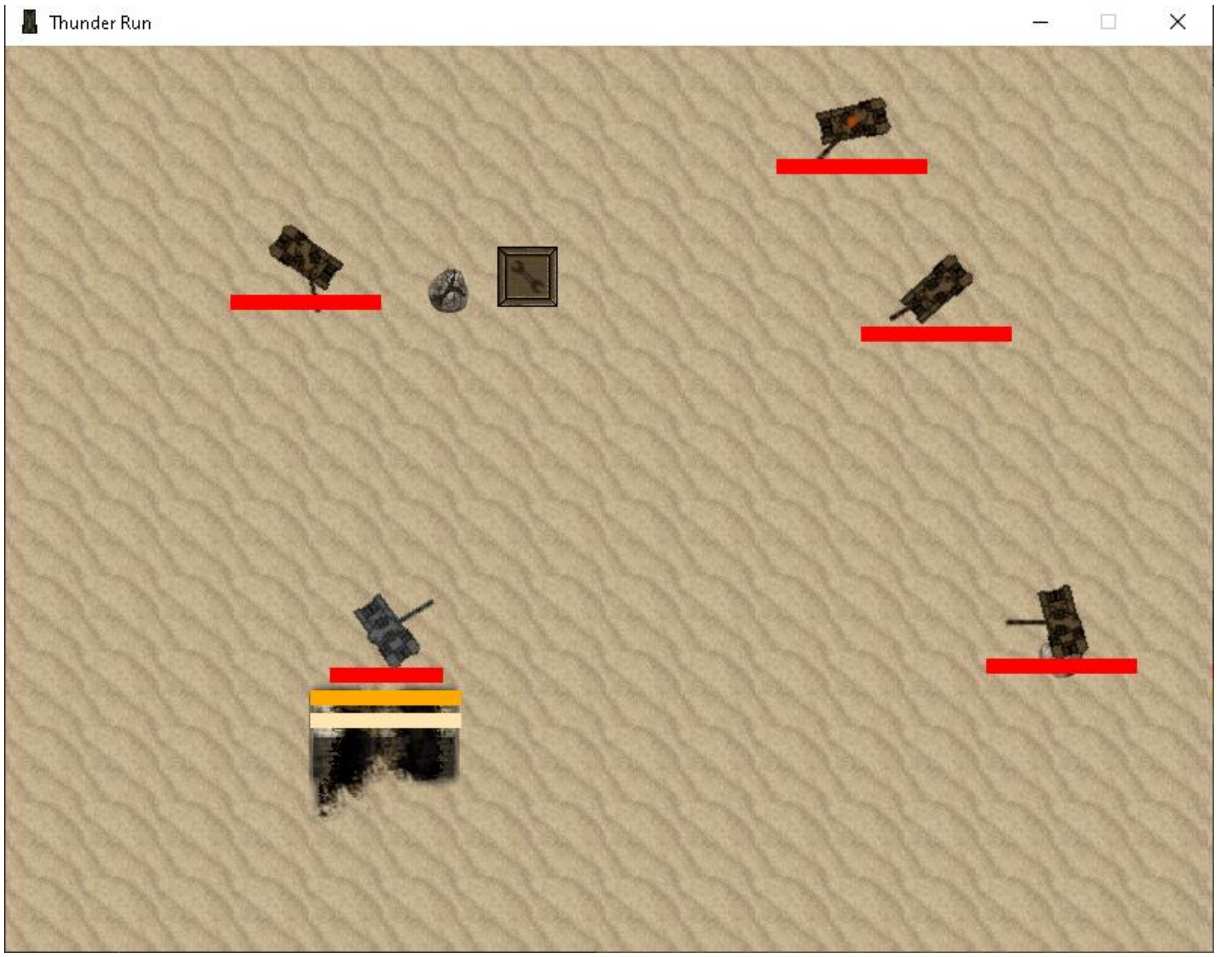



***Slika 12:*** *Opcije igre nakon male izmjene*

Nakon što je igrač završio sa izmjenama, može pritisnuti gumb „Back“ za povratak na glavni izbornik, prilikom čega će se sve promijene spremiti. S tim postavkama se sad može započeti igra koja je prikazana na idućoj slici.

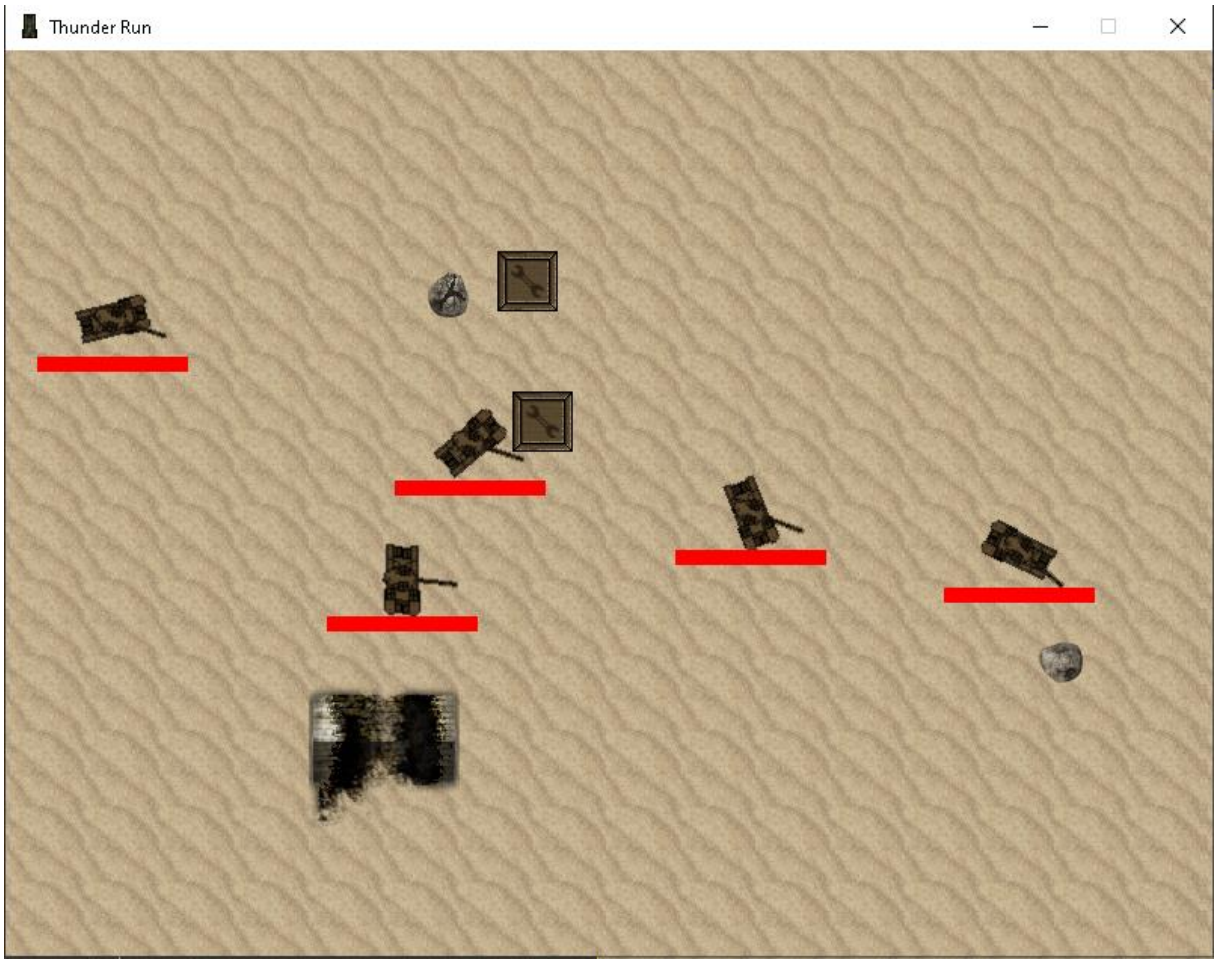

***Slika 13:*** *Prikaz početka igre*

U ovoj slici je prikazano kako se je nasumično generirao okoliš (jedna kučica i 2 kamena), generirana je i kutija za popravak te isto tako i neprijatelj kojeg je sad potrebno poraziti.




***Slika 14:*** *Prikaz 4. runde igre*

Nakon 3 odigrane runde već su vidljive promjene u okolišu, poput uništene kučice i kamena koji više ne blokiraju projektile te tako igra postaje teža. Isto tako je vidljiv veći broj neprijatelja.




***Slika 15:*** *Prikaz 5. runde*

Nakon 4 uspješne runde, igrač susreće svoj kraj u 5. rundi, protiv 5 neprijateljskih tenkova.

# 5. Rasprava

## 5.1. Prednosti korištenja Pythona

Prednost Pythona je lakoća te relativno velika brzina usavršavanja programerskih sposobnosti. To postiže svojom jednostavnosti, zbog koje programer ne mora imati veliko predznanje u programiranju. Isto tako omogućava programerima da posao obavljaju s manje redaka koda u odnosu na slične objektno orijentirane jezike. Python također nudi mogućnost lakog održavanja gdje se pogreške mogu uočiti i riješiti relativno brzo.

## 5.2. Nedostaci korištenja Pythona

Što se tiče nedostataka, Python je vrlo neprikladan za memorijske procese, pošto je Python interpreterski jezik (vrsta programskog jezika koji izvršava svoje upute izravno, bez prethodnog kompiliranja programa u strojni jezik) te zbog toga stvara pad performansi u odnosu na kompajlerske jezike (C, C++, C#), te se zbog toga Python često ne koristi za razvoj aplikacija koje koriste zahtjevniju 3D grafiku. Također jedan od nedostataka može biti nepotpuna dokumentacija, ili čak potpuno nepostojanje te dokumentacije, pogotovo kod novonastalih funckionalnosti.

## 5.3. Osiguranje kvalitete aplikacije

Osiguranje kvalitete je način sprječavanja pogrešaka i nedostataka u aplikacijama ili proizvodima te izbjegavanje problema prilikom izlaska aplikacije u javnu uporabu. Postoje par različitih pristupa testiranju kvalitete, no u testiranju ove aplikacije najviše bi pristajao bi pristup „Ispitivanje neuspjeha“ (engl. stress testing). Zbog relativno male složenosti aplikacije, nije potrebno raditi mnogo testiranja, no bitno je samo provjeriti može li igrač na neki način „iskorištavati“ mehaniku video igre u svoju korist, npr. izlazak igrača izvan „granica“ video igre kombiniranjem velike brzine igrača i usporavanja video igre.

# 6. Zaključak

Ovaj završni rad je pokušao predočiti zahtjevnost i kompleksnost izrade grafičkih aplikacija. Rad je ukratko prošao kroz korištenu arhitekturu u kojoj je kod zadatka završnog rada napravljen te kroz alate, tj. module koji su korišteni za njegovu izradu.

U radu je također prikazan kod zadatka završnog rada, koji se sastoji od 3 glavna modula (Main, Player i Enemy) te jednog sporednog modula (Enviroment) koji nije bio prikazan čisto zbog svoje jednostavnosti. Main modul je izrađen u MVC arhitekturi, te sadrži sve od same logike grafičke aplikacije do njenog grafičkog sučelja, dok su Player i Enemy moduli napravljeni kao „kontejneri“ za igračeve sprajtove (Player modul) i neprijateljeve sprajtove (Enemy modul). Na kraju su prikazani rezultati grafičke aplikacije te je ukratko opisano kako funkcionira aplikacija.

## Literatura


[1.] Al Sweigart, Making Games with Python & Pygame, CreateSpace Independent Publishing Platform, 2012. [Pristupljeno: lipanj 2020.]

[2.] Alexandros Karagkasidis, Developing GUI Applications: Architectural Patterns Revisited, [Pristupljeno: kolovoz 2020.]

[3.] Alexandros Karagkasidis, Developing GUI Applications: Architectural Patterns Revisited, [Pristupljeno: kolovoz 2020.]

[4.] https://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93controller, [Pristupljeno, srpanj, 2020.]

[5.] https://subscription.packtpub.com/book/application_development/9781785889196/10/ch10lvl1sec88/mvc-architecture, [Pristupljeno: svibanj 2020.]

[6.] https://www.pygame.org/docs/, [Pristupljeno: srpanj 2020.]

[7.] https://www.pygame.org/wiki/about, [Pristupljeno: srpanj 2020.]

[8.] https://pygame-gui.readthedocs.io/en/latest/quick_start.html, [Pristupljeno: srpanj 2020.]

[9.] http://pygametutorials.wikidot.com/tutorials-basic, [Pristupljeno: lipanj 2020.]

## Popis slika

# Prilozi