Izrada računalne igre pomoću Arcade Python okvira za izradu 2D igara

Budić, Andrea

Undergraduate thesis / Završni rad

2022

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **University of Split, University of Split, Faculty of science / Sveučilište u Splitu, Prirodoslovno-matematički fakultet**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:166:755143

Rights / Prava: Attribution 4.0 International/Imenovanje 4.0 međunarodna

Download date / Datum preuzimanja: **2024-04-29**

Repository / Repozitorij:

Repository of Faculty of Science

zir.nsk.hr

UNIVERSITY OF SPLIT

dabar

DIGITALNI AKADEMSKI ARHIVI I REPOZITORIJI

SVEUČILIŠTE U SPLITU

PRIRODOSLOVNO MATEMATIČKI FAKULTET

ZAVRŠNI RAD

IZRADA RAČUNALNE IGRE POMOĆU ARCADE PYTHON OKVIRA ZA IZRADU 2D IGARA

Andrea Budić

Split, rujan 2022.

Temeljna dokumentacijska kartica

Završni rad

Sveučilište u Splitu
Prirodoslovno-matematički fakultet
Odjel za Informatiku
Ruđera Boškovića 33, 21000 Split, Hrvatska

IZRADA RAČUNALNE IGRE POMOĆU ARCADE PYTHON OKVIRA ZA IZRADU 2D IGARA

Andrea Budić

SAŽETAK

Ovim završnim radom su prvotno razjašnjeni pojmovi kao što su računalna igra i okvir za izradu računalne igre. Također, opisan je Arcade Python okvir kao i sve njegove mogućnosti korištene u ovom radu. Nakon uvoda o Arcade okviru objašnjen je jedan od načina instalacije okvira i početak rada u istoimenom okviru razvojem računalne igre platformer. Poslije teorijskog dijela, objašnjen je postupak izrade praktičnog dijela završnog rada kao i izrada same računalne igre.

Ključne riječi: Python, game okvir, računalna igra, Arcade okvir

Rad sadrži: 49 stranica, 38 grafičkih prikaza i 14 literaturnih navoda. Izvornik je na hrvatskom jeziku.

Mentor: **izv. prof. dr. sc. Ani Grubišić**, *izvanredni profesor Prirodoslovno-matematičkog fakulteta, Sveučilišta u Splitu*

Ocjenjivači: **izv. prof. dr. sc. Ani Grubišić,** *izvanredni profesor Prirodoslovno-matematičkog fakulteta, Sveučilišta u Splitu*
izv. prof. dr. sc. Branko Žitko, *izvanredni profesor Prirodoslovno-matematičkog fakulteta, Sveučilišta u Splitu*
Ines Šarić-Grgić, mag. ing. rač., *asistent Prirodoslovno-matematičkog fakulteta, Sveučilišta u Splitu*

Rad prihvaćen: **rujan 2022.**

Basic documentation card

Thesis

University of Split
Faculty of Science
Department of Computer Science
Ruđera Boškovića 33, 21000 Split, Croatia

CREATION OF A COMPUTER GAME USING ARCADE PYTHON FRAMEWORK FOR DEVELOPMENT OF 2D GAMES

Andrea Budić

ABSTRACT

With this thesis, concepts of computer game and game development framework are initially explained. Moreover, Python game development framework Arcade is described as well as its possibilities which were used in this paper. After the introduction of the Arcade framework, one of the ways of installation and the starting process of game creation is explained with development of a platformer computer game. After the theoretical part, the procedure of the development of the practical part of this thesis is described alongside the development of the computer game.

Keywords: Python, game development framework, computer game, Arcade framework

Thesis consists of: 49 pages, 38 figures and 14 references. Original language: Croatian

Mentor: **Ani Grubišić, PhD**, *Associate Professor, Faculty of Science, University of Split*

Reviewers: **Ani Grubišić, PhD**, *Associate Professor, Faculty of Science, University of Split*
Branko Žitko, PhD, *Associate Professor, Faculty of Science, University of Split*
Ines Šarić-Grgić, mag. ing. comp. *Assistant, Faculty of Science, University of Split*

Thesis accepted: **September 2022.**

Sadržaj

1. UVOD 1

2. RAČUNALNE IGRE 2

2.1 POVIJEST RAČUNALNIH IGARA 2

3. PYTHON OKVIRI ZA IZRADU RAČUNALNIH IGARA 5

3.1 PYGAME 7

3.2 ADVENTURELIB 8

3.3 PANDA3D 9

4. ARCADE 11

4.1 KONCEPTI ARCADE OKVIRA 12

4.1.1 Inicijalizacija 12

4.1.2 Prozori i koordinate 12

4.1.3 Crtanje 13

4.1.4 Objektno-orijentiran dizajn 14

4.1.5 Game Loop 16

4.2 POGODNOSTI ARCADEA 17

4.2.1 Ugrađeni resursi 17

4.2.2 Tiled mape 18

4.3 NEDOSTACI ARCADEA 20

5. IGRA PLATFORMER 21

5.1 MODELIRANJE ZAHTJEVA 21

5.2 MODELIRANJE TIJEKA 22

5.3 MODELIRANJE LOGIČKE STRUKTURE 25

5.4 MODELIRANJE VREMENSKOG REDOSLIJEDA 25

5.5. IZRADA IGRE PLATFORMER 27

5.6 RAD IGRE PLATFORMER 39

6. ZAKLJUČAK 41

7. LITERATURA 42

8. POPIS SLIKA 44

1. UVOD

Percepcija računalnih (video) igara ovisi o uzrastu i vrsti interesa osobe koja ju igra. Za jedno dijete računalna igra je „nešto“ gdje se pritiskom odgovarajuće tipke prikaz lika pomakne. Stariji uzrasti shvaćaju igre s više razumijevanja i znaju koji je cilj, misija ili pak priča. No, većini ljudi je taj stadij percepcije dovoljan i ne zamaraju se pitanjem kako nešto funkcionira u pozadini igara, ali za određene individualce je to najimpresivniji dio. Kako bi shvatili izradu samih igara treba prvo razdaniti logiku koja stoji iza pojedinačne igre, tj. definirati kakvu igru želimo napraviti i što želimo u njoj raditi. Svladavanjem spomenute logike koja je često jednostavna prelazimo na upoznavanje s alatima kojima ćemo izraditi igru. Danas se igre izrađuju na razne načine, ali u industriji su trenutno najpopularniji „game engine“ alati, tj. okviri, biblioteke posebno napravljene za izradu računalnih igara. Takvi su programi izričito kompleksni za početnike pa je kombinacija nekog programskog jezika i odgovarajućeg okvira sasvim dobar prvi korak. U ovom radu izrađena je jednostavna 2D igra „Platformer“ kao vrst upoznavanja sa izradom igara korištenjem programskog jezika Python i jednog od njegovih okvira za izradu računalnih igara pod imenom „Arcade“ uz pomoć funkcionalnosti koje on nudi. Prije predstavljanja izrade spomenute igre, upoznat ćemo korištenu tehnologiju i dublje objasniti pojmove računalne igre, okvira za izradu igara i „game engine“ alata te navesti primjere za svaki.

2. Računalne igre

Računalne igre su u suštini primjenski računalni programi za zabavu. No, u pravilu su one interaktivne igre koje se pokreću na računalima, igraćim konzolama, džepnim igraćim konzolama, te na mobitelima, dlanovnicima ili sličnim uređajima koji posjeduju video zaslon. U takve električne strojeve se ubrajaju i posebni automati na žetone ili pak novac, namijenjeni javnim prostorima ili zabavnim centrima. Računalne igre minimalno zahtijevaju izlazni uređaj i ulazni uređaj što bi u slučaju računala bili monitor i tipkovnica, a danas novije igre zahtijevaju odličan grafički procesor kao i mrežnu opremu. Današnje igre, pogotovo ako pokušavaju što vjernije prikazati privid stvarnog okoliša se smatraju kompleksnim programima zbog čega se one nerijetko igraju na osobnim računalima znatno boljih resursa i performansi no što su potrebne da se osigura njihov neometan rad. Dijele se na žanrove među kojima nema specifično definiranih granica, a neki od žanrova su: akcijski, avanturistički, horor, simulacijski, pucački u prvom licu (*eng. first-person shooter, FPS*), igranje usloga (*eng. role-playing game, RPG*), strategija u realnom vremenu (*eng. real-time strategy, RTS*) i drugi.

2.1 Povijest računalnih igara

Prijavljivanjem 1947. američkog patenta pod nazivom "Uređaj za zabavu s katodnom cijevi"[1] [1] započinje povijest računalnih igara. Patent opisuje igru u kojoj igrač upravlja elektronskim topom CRT-a kao pištoljem. Fokusirana točka zrake pištolja predstavlja projektil, a igrač pokušava kontrolirati točku kako bi pogodio mete postavljene na zaslonu pri čemu se svi pogoci mehanički detektiraju. [2]

Prve prave videoigre Dama (*eng. Draughts*) i križić-kružić (OXO) se pojavljuju 1950-ih na prvom komercijalnom računalu „Ferranti Mark I“, a prva računalna igra koja je predstavljena javnosti je „Tennis for Two“ Williama Higinbothama 1958. godine.

1961. godine je napravljena igra „Spacewar!“ koja se dugi niz godina smatrala prvom videoigrom ikad. Osmislili su je dva studenta Massachussetskog instituta za tehnologiju (MIT).

[1] (*eng. Cathode-ray tube amusement device*) osmislili su ga Thomas T. Goldsmith Jr. i Estle Ray Mann, a patentiran je 1948. godine. Spajanjem katodne cijevi s osciloskopom i osmišljavanjem gumba kojima su kontrolirali kut i putanju svjetlosnih tragova prikazanih na osciloskopu, uspjeli su izmisliti igru projektila koja je korištenjem preklapanja zaslona stvarala učinak ispaljivanja projektila na različite mete. Zbog troškova opreme i raznih okolnosti uređaj nije prodan.

Tijekom izrade došlo je do problema sa računanjem trigonometrijskih funkcija, točnije došlo je do nemogućnosti računanja trigonometrijskih funkcija koje su bile potrebne za izračun putanje svemirskog broda. Tada Stevenu Russellu priskače u pomoć Alan Kotok sa potrebnim izračunima [3]. Prvo računalo na kojem se igrao „Spacewar!“ je bio PDP-1 (Slika 1).

Slika 1 Igra „Spacewar!“ na PDP-1

Prva generacija videoigara su bile tekstualne igre gdje je igrač ručno unosio zapovijedi i tako određivao kretnje. U drugoj generaciji su vodeću ulogu preuzele igre koje su bile mješavina tekstualnih sa igrama sa statičkom grafikom. Vodeći žanr u 70-ima i 80-ima 20. stoljeća bile su tekstualne avanture. Sa 70-tim godinama je počela era prvih igraćih automata, a sredinom 80-ih se razvijaju i konzole za računalne igre. Pionir u igrama za konzole je Nitendo sa igrama poput „Mario Bros“, „Tetris“, „Super Mario 64“ i „Pokemon“. Kako se razvijalo sklopovlje tako su i igre postajale kompleksnije i spajanjem više ulazni uređaja tekstualno sučelje je zamijenjeno grafičkim, te je došlo razdoblje grafičkih avantura na arkadnim mašinama i igraćim konzolama.

Devedesetih godina dvadesetog stoljeća se događa revolucionarna promjena; predstavljena je prva cjenovno prihvatljiva 3D grafička kartica 3dfx-a Voodoo Graphics PCI. S novim napretkom nastaju i prve velike „first-person shooter“ igrice. Jedna od najvećih je bila „Wolfenstein 3D“ (Slika 2) iz 1992. godine. Usporedimo li „Wolfenstein 3D“ sa današnjim računalnim igrama ne možemo je smatrati pravom 3D. Dojam 3D igre je ostvarivala korištenjem pametnog „prečaca“ koji danas znamo pod nazivom „billboarding“. „Billboarding“ je tehnika

okretanja plošnih objekata prema kameri, a objekt posjeduje na sebi teksturu pomoću koje dobivamo dojam lažnog 3D objekta. Quake Engine je bio prvi pravi 3D „engine“ koji je debitirao 1993. igrom „DOOM“. Izrađen je kao pravi razvojni alat u C programskom jeziku i koristi OpenGL API [2] i Direct3D[3]. Uz pomoć kasnijih iteracija „Quake Engina“ napravljene su neke od najpoznatijih igara ikad kao što je „Call of Duty“ franšiza.

Slika 2 Igra „Wolfenstein 3D“

[2] OpenGL ili Javna Grafička Biblioteka je grafički 3D API napravljen od strane Silicon Graphics tvrtke. Sastoji se od preko 250 različitih funkcija koje definiraju načine crtanja kompliciranih 3D scena pomoću primitivnih (osnovnih) geometrijskih elemenata.
[3] Direct3D je jedan od dva API-a DirectX Graphics-a koji služi za prikazivanje 3D računalne grafike.

3. Python okviri za izradu računalnih igara

Za razvoj računalne igre bilo koje vrste, potrebno je odabrati grafički pokretač (*eng. Game engine*). Trenutno je na tržištu dostupna ogromna selekcija grafičkih pokretača ovisno o programskom jeziku koji odaberemo, a svaki od njih ima svoje prednosti i mane. Da još pobliže pojasnim, grafički pokretač je okvir za izradu računalnih igara (*eng. Game development framework*). Takav okvir nudi mnoštvo mogućnosti, od rada s animacijama do rada s umjetnom inteligencijom. Osnovni zadatak okvira za izradu računalnih igara je prikazivanje grafičkih elemenata, a treba istaknuti i zadatke detektiranja kolizija objekata kao i upravljanje memorijom. Većina okvira se u samoj srži sastoji od nekoliko osnovnih elemenata [4]:

1. Glavni program gdje se nalazi logika igre
2. Grafički pokretač za prikaz 2D/3D scene za rad
3. Pokretač zvučnih efekata sa potrebnim zvučnim algoritmima
4. Pokretač za simuliranje fizičkog ponašanja objekata
5. Modul za rad s umjetnom inteligencijom

U ovom radu je odabran programski jezik Python što jako utječe na sami odabir okvira koji se ubiti kod Pythona tretiraju kao Python biblioteke koje mogu biti instaliranje na razne načine. Nasuprot takvom tretiranju okvira za izradu igara imamo okvire koji se tretiraju kao zasebni grafički pokretači koji su kreirani točno za izradu igara. Neki od njih su „The Unreal Engine“, „Unity“ i „Godot“. Treba dati do znanja da se ovi zasebni grafički pokretači razlikuju od Pythonovih okvira u nekoliko ključnih aspekata:

1. Jezična podrška: programski jezici poput C++, C# i JavaScript su popularniji za izradu igara jer je i većina grafičkih pokretača napisana u njima. Jako malo zasebnih grafičkih pokretača podržava programski jeziku Python.
2. Podrška za skriptiranje: podržavaju i održavaju vlastite skriptne jezike, „Unity“ izvorno koristi C# dok „Unreal“ najbolje funkcionira sa C++.
3. Platformska podrška: moderni grafički pokretači proizvode igre za raznolike platforme što uključuje na primjer mobilne igre što je veliki pothvat u Pythonovom svijetu.
4. Licenciranje: mogućnosti licenciranja ovisi o korištenom grafičkom pokretaču.

Python je programski jezik opće namjene koji dopušta programerima korištenje nekoliko stilova programiranja poput objektno orijentiranog, proceduralnog i aspektno orijentiranog programiranja. Kao takav koristi se u podatkovnoj znanosti, strojnom učenju, izradi web aplikacija, ali i u izradi računalnih igara. Svoje okvire za izradu igara tretira poput biblioteka od kojih su većina dostupne na PyPI[4] i mogu se instalirati uz pomoć naredbe pip. No, kada pomislimo na izradu računalnih igara programski jezik Python je možda među zadnjima kojeg ćemo se sjetiti jer u svojoj srži Python nije dizajniran za računalne performanse, nego prije za performanse čistog kodiranja.

Glavna prednost Pythona je jednostavnost programskog jezika. Kako su Pythonovi okviri za izradu igara ubiti biblioteke, kreatori svoje dosadašnje znanje programiranja u Pythonu iskoriste u izradi igre i proučavanjem dokumentacije samo koriste mogućnosti odabranog okvira i tako smanjuju krivulju učenja i fokusiraju se na samo stvaranje igre. To također znači da je izrada igara sa Pythonom odličan prvi korak za početnike. Pythona zbog svoje jednostavnosti koriste i velika studija za izradu igara. Koriste ga najviše za prototipe projekata prije nego li ih „prevedu“ u neki drugi njima prikladniji programski jezik. Neke igre čak i u svojoj konačnoj verziji određene komponente koda imaju napisane u Pythonu. „Battelfield 2“ ima implementiran rezultat i balansiranje tima, a „Civilization 4“ unutarnju logiku i umjetnu inteligenciju napisanu u Pythonu [5].

Još jedna od jako bitnih činjenica vezana za Python je mogućnost ponovnog korištenja određenih dijelova koda iz drugih projekata u novim projektima. Ova prednost rezultira u manjem broju linija koda u konačnoj verziji igre. Također to znači da će igra provesti manje vremena u svojoj fazi izrade. Štoviše, dobivamo mogućnost korištenja koda napisanih od strane drugih programera korištenjem pozamašnih Pythonovih biblioteka što smanjuje pojavu mogućih problema brzine izrade računalne igre kao i sprječavanje nerješivosti nekakvog problema.

Treći razlog zašto bi netko napravio igru pomoću Pythona namjesto korištenjem nekog kompliciranijeg programskog jezika je lakoća pronalaženja grešaka i njihov popravak. Python je interpretirani programski jezik što znači da namjesto da prvo kompajlira kod on ga direktno izvršava. Postoje mane i prednosti izvršavanja tog procesa, ali najveća dobit je da se kod u

[4] Web stranica https://pypi.org/ na kojoj možemo pronaći bilo koju biblioteku potrebnu za instalaciju.

potpunosti zaustavlja kad naiđe na grešku i povratno ćemo dobiti informaciju o toj grešci. S time si olakšamo proces jer smo tako primorani fokusirati se na jednu grešku i njenim rješenjem prijeći na drugu ako postoji.

Neki od okvira za izradu igara koje ćemo pobliže predstaviti su PyGame, adventurelib i Panda3D, a za ovaj projekt je odabran okvir Arcade kojem ćemo posvetiti najviše pažnje.

3.1 PyGame

PyGame (Slika 3) je okvir koji enkapsulira i proširuje SDL biblioteku, što je skraćenica od eng. *Simple DirectMedia Layer*[5]. Kombinacija SDL biblioteke i PyGamea nam dopušta izradu bogatih višeplatformskih programa i zbog toga je organiziran u nekoliko različitih modula koji pružaju apstraktni pristup računalnoj grafici, zvučnom sklopovlju i ulaznim jedinicama. PyGame također definira brojne klase koje sažimaju koncepte koji ne ovise o sklopovlju. Na primjer, crtanje se odvija na *Surface* objektima čije su pravokutne granice definirane *Rect* objektom [6].

Svaka igra zahtjeva neku vrstu „game loop“ petlje pomoću koje se igra općenito pokreće i kroz koju kontroliramo tijek igre. PyGame pruža metode i funkcije za implementaciju same „game loop“ petlje, ali ju ne pruža automatski te od samog programera zahtjeva implementaciju njene funkcionalnosti. Svaka iteracija „game loop“ petlje se zove okvir (eng. *frame*) i tijekom svakog „framea“ igra radi sljedeće četiri radnje [6]:

1. Procesiranje korisničkog ulaza; što se u PyGameu izvodi uz pomoć modela događaja (eng. *event models*) jer PyGame ne pruža funkcionalnost rukovatelja događaja (eng. *event handlers*) nego ih sami moramo realizirati.
2. Ažuriranje stanja objekata igre; objekti igre se mogu realizirati korištenjem bilo koje PyGame podatkovne strukture ili specijalne PyGame klase. Sami objekti poput spirteova, slika, fontova i boja mogu biti kreirana i proširena u Pythonu da pruže što više potrebnih informacija o svome stanju.
3. Ažuriranje zaslona i audio izlaza; *display*, *mixer* i *music* pružaju apstraktni pristup zaslonu i zvučnom sklopovlju računala i pružaju fleksibilnost u dizajnu i implementaciji igre.

[5] SDL biblioteka pruža višeplatformski pristup temeljnim multimedijskim komponentama sklopovlja računala.

4. Održavanje brzine igre; PyGameova *time* modul pruža kontrolu nad brzinom igre osiguravajući da se svaki „frame“ izvrši unutar određenog vremenskog okvira.

PyGame je jedan od popularnijih Pythonovih okvira za izradu igara, a tome pridonose dvije činjenice koje su međusobno povezane. Prva je činjenica sadržana u njegovoj jednostavnosti i lakoći korištenja. Kao takav je korišten od strane početnika kao vrst uvoda u programiranje i od strane studenta za kompliciranije projekte. Druga činjenica leži u količini dokumentacije i količini informativnog materijala koju osoba može pronaći na PyGameovoj službenoj stranici, do članaka i videa na YouTubeu.

Slika 3 Službeni logo PyGame okvira

3.2 adventurelib

Ne zahtjeva svaka igra prikaz raznobojnog lika na zaslonu koji izbjegava prepreke i ubija negativce. Računalne igre poput „Zorka“ koje možemo smatrati klasicima pokazuju moć odličnog pripovijedanja dok istovremeno pruža sjajno iskustvo igranja. Izrada takvih tekstualno baziranih igara može biti komplicirano u bilo kojem programskom jeziku, ali zato za Python imamo posebnu biblioteku adventurelib koja pomaže pri izradi.

adventurelib pruža osnovne funkcionalnosti za izradu tekstualnih avanturističkih igara sa ciljem da mladim tinejdžerima bude dovoljno jednostavna. Osnova adventureliba leži u defniranju funkcija koje se pozivaju kao odgovor na zapovijedi (Slika 4) [7]. Ovu biblioteku su stvorili isti ljudi koji stoje iza još jednog Python okvira pod nazivom Pygame Zero[6]. adventurelib se bavi i naprednijim konceptima računalne znanosti kao što je upravljanje stanjem, poslovnom logikom, imenovanjem i referenciranjem, manipulacijom skupovima, itd. To ga čini izvrsnim alatom za učitelje, profesore, roditelje i mentore kao pomoć pri podučavanju računalne znanosti kroz izradu igara [6].

[6] Pygame Zero je okvir sličan PyGameu, ali dizajniran s namjenom za educiranje na bazi nekoliko principa savršen za početnike i mlade programere.

S obzirom da je adventurelib okvir pomoću kojeg se izrađuju tekstualne igre, postavlja se pitanje u kojem jeziku se mogu izrađivati igre. Kreatori adventurelib okvira savjetuju da se koristi engleski jezik jer je on korišten pri izradi samog API-a biblioteke. Pisanjem igre u nekom drugom jeziku stvara najviše problema u slučaju ako jezik nije anglofonski, ako nemaju koncept velikih i malih slova, ako je u pitanju jezik koji se čita s desna na lijevo, itd. adventurelib je zbog toga najiskoristiviji ako se radi o europskom jeziku, a manje iskoristiv ako se radi o jeziku iz ostatka svijeta [8].

```
@when("brush teeth")
def brush_teeth():
    print("You brush your teeth. They feel clean.")
```

If you start the game again you can try out the new command:

```
> brush teeth
You brush your teeth. They feel clean.
```

Slika 4 Primjer zapovijedi „brush teeth“

3.3 Panda3D

Panda3D (Slika 5) je okvir otvorenog koda za stvaranje 3D igara i 3D prikaza. Višeplatformski je okvir i podržava višestruke vrste sredstava poput slika ili zvukova, odmah se povezuje s brojnim bibliotekama trećih strana i pruža ugrađeno profiliranje cjevovoda. Iako je namijenjen za razvoj igara u programskom jeziku Python, sam pokretač je napisan u C++ jeziku i korištenjem omotača (eng. *wrapper*) izlaže kompletnu funkcionalnost okvira Pythonu. Ovakav pristup pruža programeru prednosti razvoja programa u Python programskom jeziku, kao što je brzi razvoj i napredno upravljanje memorijom, ali zadržava performanse jezgrenog programskog jezika.

Panda3D je stvoren za razvoj komercijalnih igara i zbog toga snaga, brzina, potpunost i tolerancija na pogreške su od iznimne važnosti. Snaga i brzina su razumljivi sami po sebi, ali pod potpunošću se misli na alate koji pomažu u samoj izradi igre: alat za kontrolu grafičkog prikaza, praćenje performansi, optimizatore animacije, itd. Kod tolerancije grešaka Panda3D se podosta razlikuje od ostalih okvira jer Panda3D gotovo nikad ne prestaje s radom ako naiđe na nekakvu vrstu greške jer je dosta koda Pande3D implementirano s posebnom posvetom na pronalaženje i izolaciju grešaka [9].

S obzirom da se igre izrađuju u Pythonu ne smijemo se zavarati i misliti kako je Panda3D alat za početnike. Panda3D podržava cijeli raspon funkcionalnosti koje su prisutne u mnogim drugim naprednim i modernim okvirima za izradu igara kao što su normal mapping“, „gloss mapping“, HDR, sjenčanje crtanih likova i „inking“, cvjetanje kao i mogućnost pisanja vlastitih „shadera“. Sami kreatori nalažu kako se Pandom3D ne mogu koristiti ljudi koji ne znaju što je API, stablo i koji nemaju iskustva sa programiranjem i kako bi tim ljudima Panda3D bila pozamašan zalogaj zbog svojih funkcionalnosti.

Sami okvir Panda3D je besplatan, ali dolazi s brojnim bibliotekama treće strane od kojih neke nisu besplatne. Neke biblioteke ograničavaju kreatore u korištenju ako nemaju licencu pri izradi komercijalnih igara. Panda3D uklanja taj problem sa lakim izbacivanjem tih zabranjenih biblioteka s treće strane i u ponudi nudi alternative.

Slika 5 Logo Panda3D i 3D rezultat korištenja okvira

U sljedećem poglavlju ćemo detaljnije opisati Pythonov okvir za izradu 2D igara Arcade koji je odabran za izradu projekta. Pobliže ćemo se upoznati sa njegovim konceptima i pogodnostima koje su kasnije korištene za izradu 2D računalne igre „Platformer“ kao i sa njegovim nedostacima.

4. Arcade

Arcade je moderni Pythonov okvir za izradu 2D igara s dojmljivom grafikom i zvukom. Objektno orijentiran po dizajnu, Arcade programerima igara pruža moderan skup alata za stvaranje izvrsnih igara koje pružaju odlično iskustvo igranja. Dizajnirao ga je profesor Simpson Collegea Paul Vincent Craven na bazi pyglet[7] prozora (eng. *windowing*) i multimedijske knjižnice [6]. Pruža skup poboljšanja i modernizacija koje se najbolje mogu usporediti sa PyGameom i Pygame Zeroom:

1. Podržava modernu OpenGL grafiku
2. Podržava Python 3 pomoć tipiranja (eng. *type hinting*)
3. Podržava animirane likove (eng. *sprite*) temeljene na okviru (eng. *frame-based*)
4. Uključuje dosljedne nazive naredbi, funkcija i parametara
5. Propagira odvajanje logike igre od koda za prikaz
6. Pruža dobro održavanu i ažurnu dokumentaciju što uključuje nekoliko vodiča kao i potpune primjere Python igara
7. Ima ugrađene fizičke pokretače (eng. *physics engine*) za „top-down“ igre i platformerske igre

Arcade je izrađen na bazi pygleta i OpenGla i može se pokrenuti na operacijskim sustavim Windows, Mac OS X i Linux. Za rad s njim potreban je Python verzije minimalno 3.6 ili novije, a za reprodukciju zvuka koristi SoLoud pokretač zvukova. Besplatan je i mogu se kreirati besplatne, „shareware“ i komercijalne igre otvorenog koda. Arcade je u konstantnom razvoju i podržan je od strane zajednice, a njegov kreator je ažuran u ispravljanju prijavljenih problema [10].

[7] pyglet je moćna, ali jednostavna Python biblioteka za razvoj igara i drugih vizualno bogatih aplikacija. Podržava „windowing“, rukovanje događajim, OpenGL grafiku, učitavanje slika i videa, te reprodukciju zvukova i glazbe.

4.1 Koncepti Arcade okvira

Kod napisan pomoću Arcade okvira se može izvršiti skoro na svakoj platformi koja podržava Python. Ovakav način rada iziskuje od Arcadea da se nosi s apstrakcijama platformi ovisno o njihovom sklopovlju. Razumijevanje tih apstrakcija, razlika i koncepata koje nudi Arcade će nam pomoći u samoj izradi i dizajnu igara kao i u shvaćanju što razlikuje Arcade od drugih Pythonovih okvira.

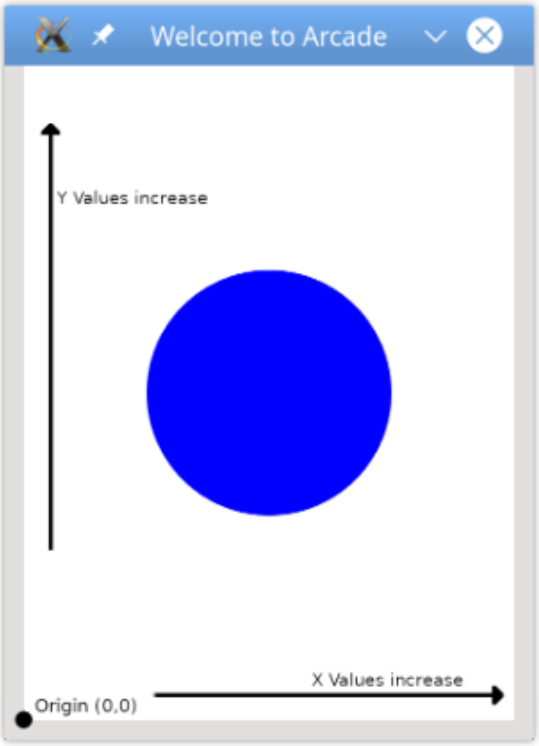
4.1.1 Inicijalizacija

S obzirom da se nosi sa različitim platformama, Arcade mora odraditi inicijalizacijski korak prije korištenja. Taj korak se obavlja automatski i odvija se kad god uvezemo (eng. *import*) Arcade biblioteku. Za razliku od drugih okvira poput PyGamea Arcadu nije potreban nikakav dodatan kod da bi se korak inicijalizacije pokrenuo. Kada uvezemo Arcade okvir odviju se sljedeći procesi:

1. Provjera da je Pythonova verzija barem 3.6 ili viša
2. Uvez *pyglet_ffmeg2* biblioteke koja je zadužena za rukovanje zvukom ako je dostupna
3. Uvez *pyglet* biblioteke za rukovanje sa prozorima i multimedijom
4. Postavljanje konstanti za boje i ključ mapiranja
5. Uvez ostatak Arcade biblioteke

4.1.2 Prozori i koordinate

Sve u Arcadeu se događa u prozoru kojeg stvorimo koristeći funkciju okvira *open_window()*. Parametri koje ova funkcija prima su *width, height, window_title, resizable* i *antialiasing*. U njih zapisujemo podatke o širini, visini, opcionalnom naslovu prozora kao i dopuštenju promjene veličine prozora i dopuštenju da OpenGL-ov anti-aliasing ispuni rubove korištenih objekata. Arcade u svom prozoru koristi Kartezijev koordinatni sustav (Slika 6) i početna nul-točka (0,0) se nalazi u donjem lijevom kutu prozora. Prozor zauzima područje prvog kvadranta koordinatnog sustav i vrijednosti apcise x se zbog toga povećavaju pomakom u desno, a vrijednosti ordinate y pomakom put gore. Ponašanje vrijednosti na apcisi i ordinati je drukčije za razliku od drugih okvira gdje je prozor često smješten u četvrtom kvadratnu.



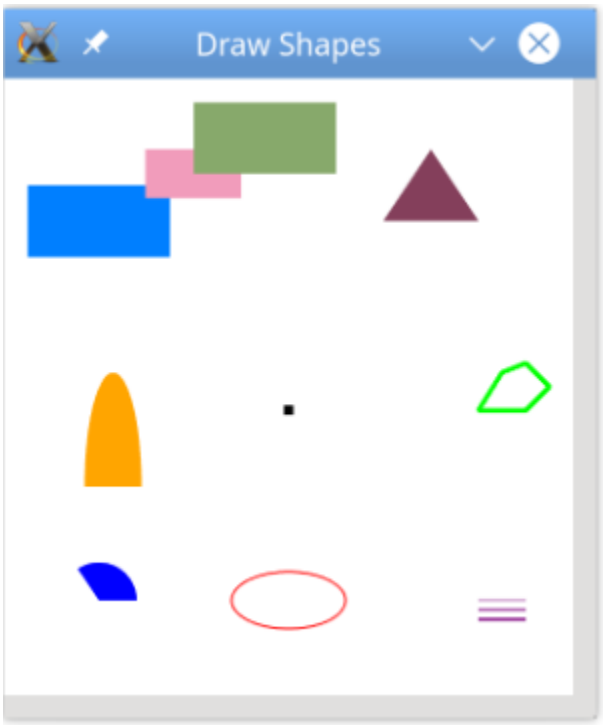
Slika 6 Korištenje koordinatnog sustava u Arcade okviru

4.1.3 Crtanje

Arcade posjeduje funkcije za crtanje različitih geometrijskih oblika što uključuje lukove, krugove, elipse, linije, parabole, točke, mnogokute, četverokute i trokute (Slika 7). Sve funkcije počinju sa *draw_* i zatim slijedi dosljedno imenovanje funkcije i njezin potpis ovisno o tome koji oblik želimo nacrtati. Također postoje različite funkcije ovisno o tome želimo li imati ispunjeni geometrijski ili pak ocrtani oblik ili nekad sana funkcija ovisi o koordinatnom sustavu. Na primjer, za crtanje četverokuta postoje čak tri različite funkcije čije ime u nastavku posjeduje **_filled* ili **_outlined*:

1. *draw_rectangle*()* – očekuje od parametara koordinate x i y koje označuju središte četverokuta, širinu i visinu
2. *draw_lrtb_rectangle*()* – od parametara očekuje lijevu i desnu koordinatu x i gornju i donju koordinatu y
3. *draw_xywh_rectangle*()* – od parametara očekuje x i y koordinatu donjeg lijevog kuta četverokuta, širinu i visinu

Osim korištenja funkcija koje počinju sa *draw_* možemo koristiti i funkcije crtanja koje koriste međuspremnik (eng. *buffered drawing funkcitons*). Takve funkcije koriste međuspremnike za vrhove koje šalju izravno na grafičku karticu što uzrokuje poboljšanje performansi. Takve funkcije započinju s *create_* i slijede dosljedne uzorke imenovanja i parametara.



Slika 7 Primjeri oblika koji se mogu nacrtati koristeći Arcade

4.1.4 Objektno-orijentiran dizajn

Kod u Arcadeu možemo pisati proceduralno. Na primjer za proceduralni kod za crtanje ispunjenog kruga (Slika 8) na samom početku uvezemo Aracde biblioteku i zatim otvorimo prozor sa naredbom *aracde.open_window()* koja prima širinu, visinu i naslov prozora. Postavimo pozadinsku boju pomoću *arcade.set_background_color()* prije postavljanja crtačeg (eng. *draw mode*) bloka koji počinje sa naredbom *arcade.start_render()* i završava sa naredbom *arcade.finish_render()*. U crtači blok pomoću kojeg se prikazuju nacrtani oblici napišemo naredbu za crtanje ispunjenog kruga *aracde.draw_circle_filled()* sa njezinim parametrima. Na kraju naredbom *arcade.run()* prikazujemo proceduralni kod crtanja kruga.

```
import arcade

SCREEN_WIDTH = 600
SCREEN_HEIGHT = 800
SCREEN_TITLE = "Welcome to Arcade"
RADIUS = 150

arcade.open_window(SCREEN_WIDTH, SCREEN_HEIGHT, SCREEN_TITLE)

arcade.set_background_color(arcade.color.WHITE)

arcade.start_render()

arcade.draw_circle_filled(
    SCREEN_WIDTH / 2, SCREEN_HEIGHT / 2, RADIUS, arcade.color.BLUE
)

arcade.finish_render()

arcade.run()
```

Slika 8 Proceduralni kod za crtanje ispunjenog kruga

Iako je omogućeno proceduralno programiranje, Aracde je u svojoj srži objektno-orijentiran gdje leži njegova moć. Na primjeru crtanja ispunjenog kruga prva razlika u objektno-orijentiranom kodu (Slika 9) od proceduralnog je u definiranju klase *Welcome* koja je bazirana na roditeljskoj klasi *arcade.Window.* Ovakav način definiranja klase nam omogućuje premošćivanje (eng. *override*) metoda, tj. funkcija roditeljske klase. U klasi *Welcome* zatim postavimo konstruktor *.__init__()* kojim inicijaliziramo prozor koji postavljamo uz pomoć konstruktora roditeljske klase koju pozivamo pomoću naredbe *super().__init__()* i zatim postavimo boju pozadine inicijaliziranog prozora. Nakon kontruktora *.__init__()* premošćujemo metodu *.on_draw()* kako bi crtala ispunjeni krug. Premošćena metoda *.on_draw()* predstavlja crtaći blok koji počinje sa *arcade.start_render()*, ali za razliku od crtaćeg bloka u proceduralnom kodu ne moramo pisati naredbu *aracde.finish_render()* jer će Arcade odraditi završetak crtanja čim završi metoda *.on_draw()*. Na kraju, van klase *Welcome* odredimo glavnu ulaznu točku koda gdje nakon što stvorimo novi objekt klase *Welcome()* pozovemo naredbu *aracde.run()* za prikaz prozora. Ono što možemo primijetiti jest da će Aracde zvati *.on_draw()* metodu kad god želi nešto nacrtati na prozor. Da bi smo to mogli kontrolirati koristit ćemo još jedan Aracdeov koncept „Game Loop“.

```python
import arcade

SCREEN_WIDTH = 600
SCREEN_HEIGHT = 800
SCREEN_TITLE = "Welcome to Arcade"
RADIUS = 150

# Classes
class Welcome(arcade.Window):
    """Main welcome window
    """
    def __init__(self):
        """Initialize the window
        """

        # Call the parent class constructor
        super().__init__(SCREEN_WIDTH, SCREEN_HEIGHT, SCREEN_TITLE)

        # Set the background window
        arcade.set_background_color(arcade.color.WHITE)

    def on_draw(self):
        """Called whenever you need to draw your window
        """

        # Clear the screen and start drawing
        arcade.start_render()

        # Draw a blue circle
        arcade.draw_circle_filled(
            SCREEN_WIDTH / 2, SCREEN_HEIGHT / 2, RADIUS, arcade.color.BLUE
        )

# Main code entry point
if __name__ == "__main__":
    app = Welcome()
    arcade.run()
```

Slika 9 Objektno-orijentirani kod za cratnje ispunjenog kruga

4.1.5 Game Loop

Sve akcije u gotovo svakoj igri se odvijaju u središnjoj petlji igre (eng. *game loop*). „Game Loop“ počinje nakon što je igra postavljena i inicijalizirana i završava kada i igra završava. Nekoliko se stvari događa uzastopno u samoj petlji, a minimalno sljedeće četiri akcije:

1. Program utvrđuje je li igra gotova. Ako je, petlja završava.
2. Obrađuju je ulazni podaci korisnika.
3. Stanja objekata u igri se osvježavaju i ažuriraju ovisno ulaznim podacima korisnika ili vremenu.
4. Ovisno o stanju igra prikazuje vizualne elemente i reproducira zvučne efekte.

U Arcade okviru „game loop“ je enkapsulirana u naredbi *arcade.run()*. U njoj se poziva skupina *Window* metoda pomoću kojih možemo implementirati gore navedene četiri akcije. Imena ovih metoda počinje sa *on_* i mogu se smatrati metodama ili bolje rečeno rukovateljima događaja (eng. *event handlers*). Ove metode u svojim zadanim postavkama ne rade ništa korisno, ali tijekom izrade vlastite klase čiji je roditelj klasa *arcade.Window*. možemo ih premostiti kako nam je potrebno ovisno o funkcionalnostima igre i što prema njima trebaju raditi. Neke od tih metoda su zadužene za:

1. Obradu ulaznih podataka sa tipkovnice: *.on_key_press(), .on_key_release()*
2. Obradu ulaznih podataka sa miša: *.on_mouse_press(), .on_mouse_release(), .on_mouse_motion()*
3. Ažuriranje objekata u igri: *.on_update()*
4. Crtanje: *.on_draw()*

Ne trebamo premostiti sve ove metoda nego samo one koje su nam potrebne za neometano funkcioniranje igre. Također se ne moramo brinuti o tome kada se pozivaju tijekom izvršavanja igre nego se moramo brinuti samo o tome što potrebne premoštene metode rade kada se pozovu.

4.2 Pogodnosti Arcadea

Nakon što smo razumjeli osnovan rad Arcadea kroz njegove koncepte možemo prijeći na njegove dodatne pogodnosti koje možemo koristiti. Riječ je o ugrađenim resursima (eng. *built-in resources*) Arcadea i mapama koje možemo koristiti pri izradi igara.

4.2.1 Ugrađeni resursi

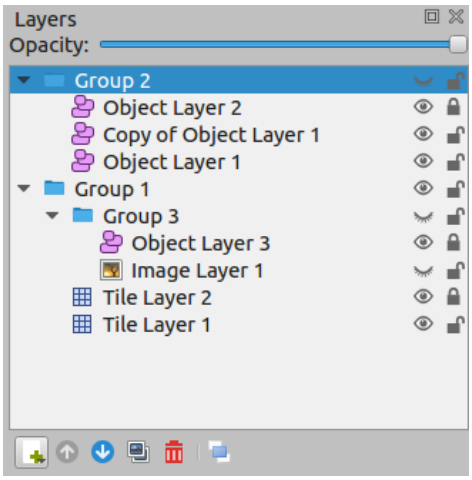
Kad spomenemo ugrađene resurse Arcadea (Slika 10) mislimo na slike i zvukove koje možemo lagodno koristiti pri izradi igara pri tom da se ne moramo brinuti o kopiranju datoteka u mapu projekta. Svaka datoteka koja je učitana iz resursa počinje sa *:resources:*. S tim početkom učitava se iz same biblioteke Arcadea umjesto iz nekakvog direktorija sa dodatnim resursima kojeg smo sami mogli napraviti u projektu. Većina ugrađenih resursa dolazi sa web stranice *Kenny.nl* na kojoj možemo pronaći još veći izbor slika, zvukova pa čak i mapa.

Slika 10 Neki od ugrađenih resursa

4.2.2 Tiled mape

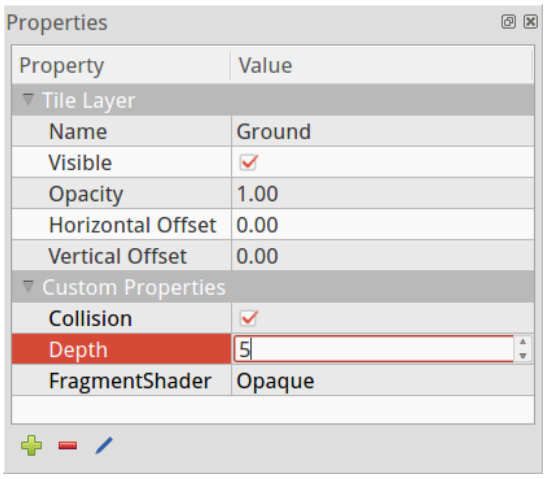
Iako već u ugrađenim resursima imamo već gotove mape koje su u Json obliku, moramo znati kako su točno napravljene da bi smo ih mogli u potpunosti koristiti. Mape koje su u resursima možemo učitati u programu pod nazivom *Tiled*. *Tiled* je 2D level editor koji pomaže u izradi sadržaja za igre. Njegova primarna značajka je uređivanje mape pločica (eng. *tile maps*) različitih oblika, ali isto podržava slobodno postavljanje slika kao i moćne načine označavanja mapa sa dodatnim informacijama koje koristimo u igri. Fokusira se na generalnu fleksibilnost dok pokušava ostati intuitivan [11]. Popločana mapa podržava različite vrste sadržaja koji je organiziran u različite slojeve. Najčešći slojevi su sloj pločica (eng. *Tile Layer*) i sloj objekata (eng. *Object Layer*) (Slika 11). Njihov poredak označava i poredak kojim se prikazuje naš sadržaj. Sloj pločica pruža učinkovit način ispunjavanja velikog područja s potrebnim podacima o mapi. Ti podaci su jednostavan niz referenci na pločice koje također ograničavaju pohranjivanje dodatnih podataka na lokacije pločica. Sloj objekata su korisni jer mogu u sebe spremiti dodatne podatke koje nismo

mogli spremiti u sloj pločica. Objekti se mogu slobodno postavljati i rotirati po mapi. Također im možemo mijenjati veličinu. Najbitnija značajka objekata je da mogu u sebi posjedovati individualna, osobna svojstva kao i klasu kojom možemo upravljati tim svojstvima [12].



Slika 11 Primjer slojeva u progamu Tiled

Jedna od glavnih prednosti Tileda je ta što omogućuje postavljanje prilagođenih svojstava na sve njegove osnovne strukture podataka. Na ovaj način moguće je uključiti mnoge oblike prilagođenih informacija koje kasnije možemo koristiti u igri ili u okviru koji koristimo za integraciju popločanih karata. Prilagođena svojstva prikazana su u prikazu *Properties* (Slika 12). Ovaj prikaz je osjetljiv na kontekst i obično prikazuje svojstva posljednje odabranog objekta. Također podržava višestruki odabir, za promjenu svojstava više objekata odjednom.



Slika 12 Prikaz Properties okvira

4.3 Nedostaci Arcadea

Prvi nedostatak Arcadea leži u njegovoj starosti. Kako je okvir relativno nov smatrao se nestabilnim i često su programeri nailazili na probleme nakon što bi nadogradili okvir s novom poboljšanom verzijom Arcadea. Već dovršeni projekti pri pokretanju bi javili pogrešku jer bi se neka od korištenih metoda promijenila u imenu ili potpisu. Trenutno takvi problemi su sve rjeđi, ali su i dalje mogući jer Arcade prima nove značajke i ažuriranja ujednačenim tempom od 2021. godine.

Drugi nedostatak Arcadea je da podržava rad samo sa određenim verzijama Pythona. Arcade podržava samo verzije Pythona 3.6 i na više. U slučaju da imamo neku nižu verziju Pythona okvir nam neće raditi jer jednostavno ne podržava tu verziju što je nezadovoljavajuće.

Treći nedostatak ovisi o samom programeru. Arcade koristi određene module koji nam olakšavaju izradu igre što je odlično za početnike, ali iskusni programeri bi radije sami isprogramirali funkcionalnosti određenih modula zbog potpune kontrole. Jedan od takvih modula je *Physics Engine* koji u svojoj suštini uvodi pravila fizike u igru i rukovodi određenim aspektima igre kao što su animacije i kolizije. Početnicima je taj modul od velike koristi jer se mogu fokusirati na glavnu logiku igre, ali iskusnim programerima koji u svojim igrama imaju nešto zahtjevniju fiziku će više odmoći nego pomoći.

Posljednji nedostatak je u nekonzistentnosti dokumentacije. Međusobno moduli Arcadea nisu popraćeni dokumentacijom u jednakoj količini. Oni moduli koji su najčešće korišteni su poduprti i tekstualnim objašnjenima i primjerima dok su neki samo tekstualno objašnjeni, a neki kao objašnjenje imaju naveden samo primjer. Početnicima će nekonzistentnost dokumentacije biti problem zbog razumijevanja, ali s vremenom i činjenicom da je Arcade nov dokumentacija postaje sve bolja i bolja.

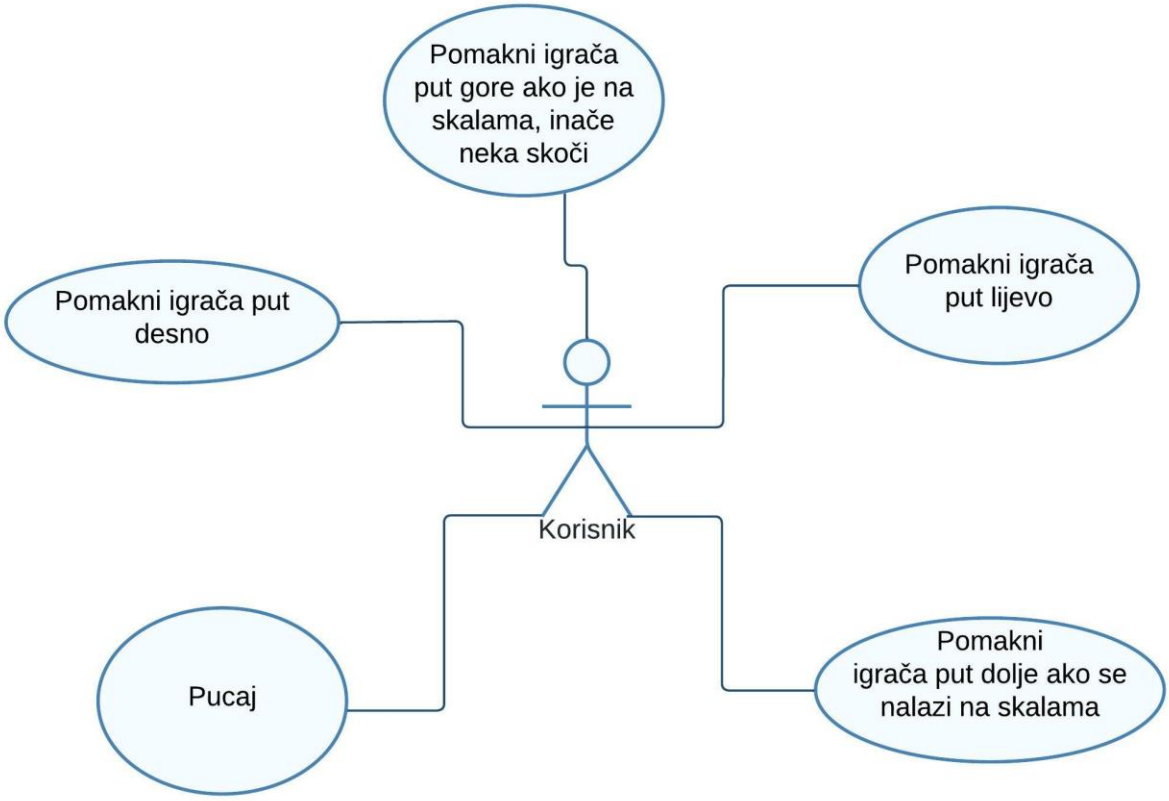
5. Igra Platformer

Prije nego što krenemo sa kodiranjem trebamo definirati što želimo da igra radi. S obzirom da nam je inspiracija igre *Super Mario* znamo da moramo imati lika koji utjelovljuje samog Super Maria kojim upravljamo preko tipkovnice koristeći tipke W, A, S i D ili strelice put gore, dolje, lijevo i desno. Osim utjelovljenja *Super Maria* kao igrača imamo i neprijatelje koje igrač ubija i pritom dobije određen broj bodova za rezultat. Osim ubijanjem neprijatelja, igrač dobiva bodove i skupljanjem novčića i zastavica. Tek kada igrač skupi sve novčiće, zastavice i ubije sve neprijatelje korisnik je pobjednik. U slučaju da dođe u doticaj sa neprijateljem ili se ne dočeka na platfomu, gubi igru.

5.1 Modeliranje zahtjeva

Dijagramom slučajeva korištenja (Slika 14) ćemo definirati funkcionalnosti igre. Korisnik dok upravlja animiranim likom igrača tipkama na tipkovnici može raditi sljedeće radnje:

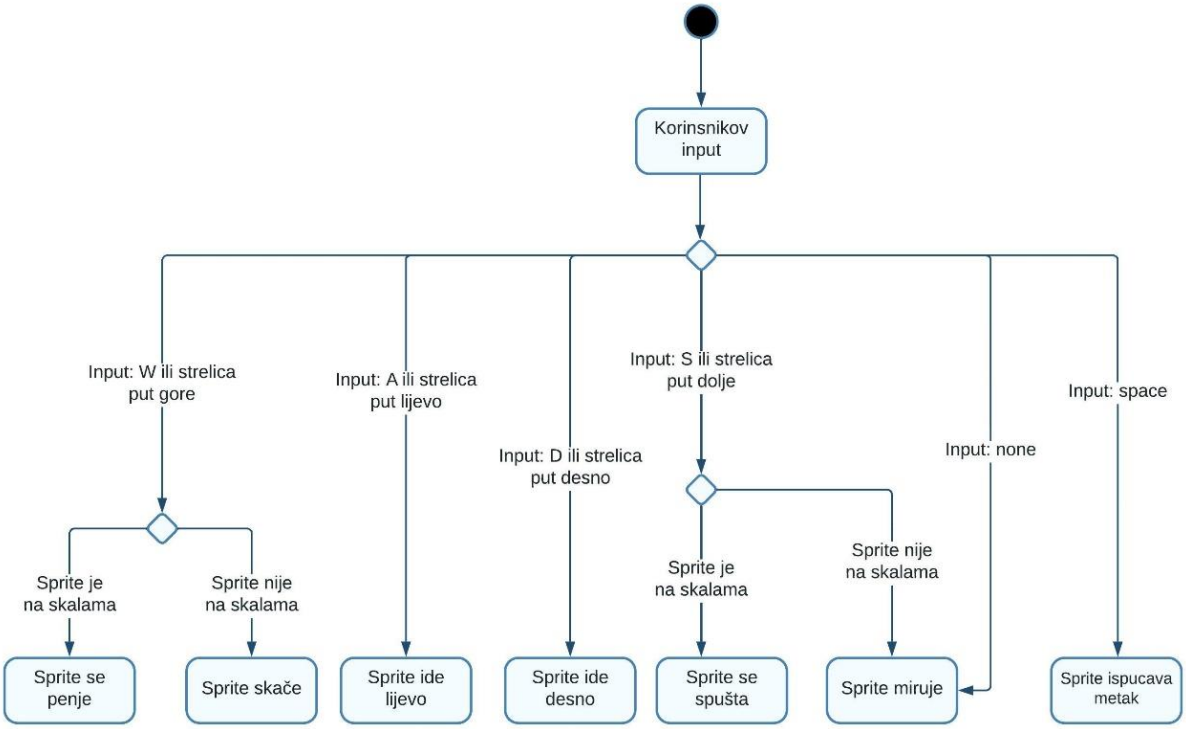
1. Pomaknuti igrača put gore ako je na skalama, inače neka igrač skoči
2. Pomaknuti igrača put lijevo
3. Pomaknuti igrača put desno
4. Pomaknuti igrača put dulje ako je na skalama
5. Pucati na neprijatelje



Slika 13 Dijagram slučajeva korištenja

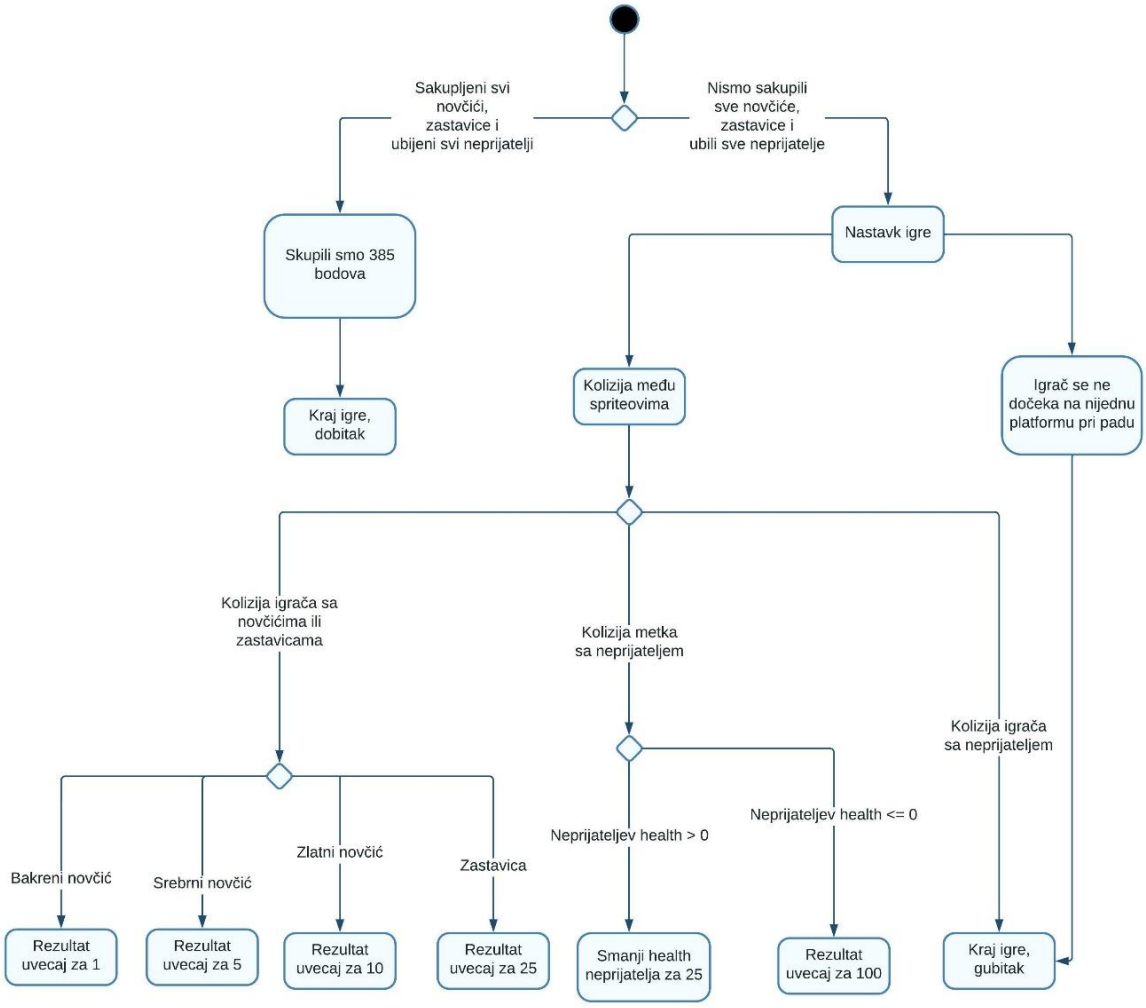
5.2 Modeliranje tijeka

Osnovna kretanja igrača se događaju kada korisnik pritisne određenu tipku. Bolje rečeno kretnje igrača se događaju kada se pritisnu tipke W, A, S i D ili strelice gore, dolje, lijevo i desno i tipka za razmak. S obzirom na pritisnute tipke igrač će se pomaknuti u odabranu stranu ili će skočiti ili će pucati, a dijagramom aktivnosti (Slika 15) ćemo još pobliže odrediti te funkcionalnosti kao i što se događa sa igračem kad nije pritisnuta ni jedna tipka.



Slika 14 Dijagram aktivnosti ovisno o ulaznim podacima korisnika

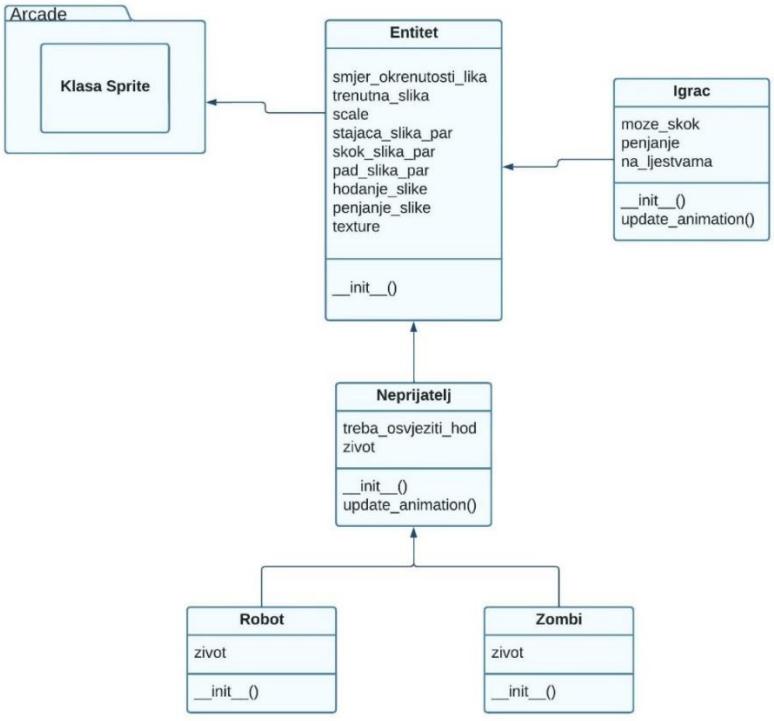
Još jednim dijagramom aktivnosti (Slika 16) ćemo opisati što se točno događa sa pojedinim animiranim likovima u slučaju kolizije među njima. Specifično govorimo o kolizijama između igrača sa neprijateljem, igrača sa novčićima, zastavicama i kolizijama između ispucanog metka i neprijatelja. Tim dijagramom aktivnosti je također opisan kada dolazi do kraja igre. Ako je igrač dotakao neprijatelja ili se igrač nije imao dočekati na nijednu platformu tada dolazi do kraja igre gdje smo izgubili. U slučaju da smo sakupili sve novčiće, zastavice i ubili sve neprijatelje tada smo pobjednici i također dolazi do kraja igre. Ako smo sakupili novčić, zastavicu ili ubili neprijatelja tijekom igre na rezultat dodamo određeni broj bodova.



Slika 15 Dijagram aktivnosti za tijek igre i kolizije

5.3 Modeliranje logičke strukture

Za definiranje samih animiranih likova koje ćemo koristiti u igri iskoristiti ćemo dijagram klasa (Slika 13). Kako se radi o animiranim likovima znamo da trebamo omogućiti njihovu animaciju da bi nam igra izgledala što bolje. Samo osvježavanje ćemo obaviti koristeći metodu *update_animation()* koju premošćujmo u klasama *Igrac* i *Neprijatelj* gdje ubiti pišemo logiku o tome koja će se slika pojaviti za određeno kretanje. U tim animaciji pomažu i varijable *moze_skok*, *penjanje* i *na_ljestvama* kod *Igraca* i varijabla *treba_osvjeziti_hod* kod *Neprijatelja*. Kako bi smanjili mogućnost greške pri učitavanju slika kretanja za pojedinu klasu ono se odvija u klasi *Entiteti* u čije varijable učitavamo slike iz resursa. *Entitet* nasljeđuje Arcadeovu klasu *Sprite* i tako svim ostalim klasama daje pristup varijabli *texture* kojom postavljamo točno onu sliku koju želimo prikazati. Klasu *Neprijatelj* nasljeđuju klase *Robot* i *Neprijatelj* u kojima je istaknuta varijabla *zivot* koja se smanjuje kada igrač upuca određenog neprijatelja.

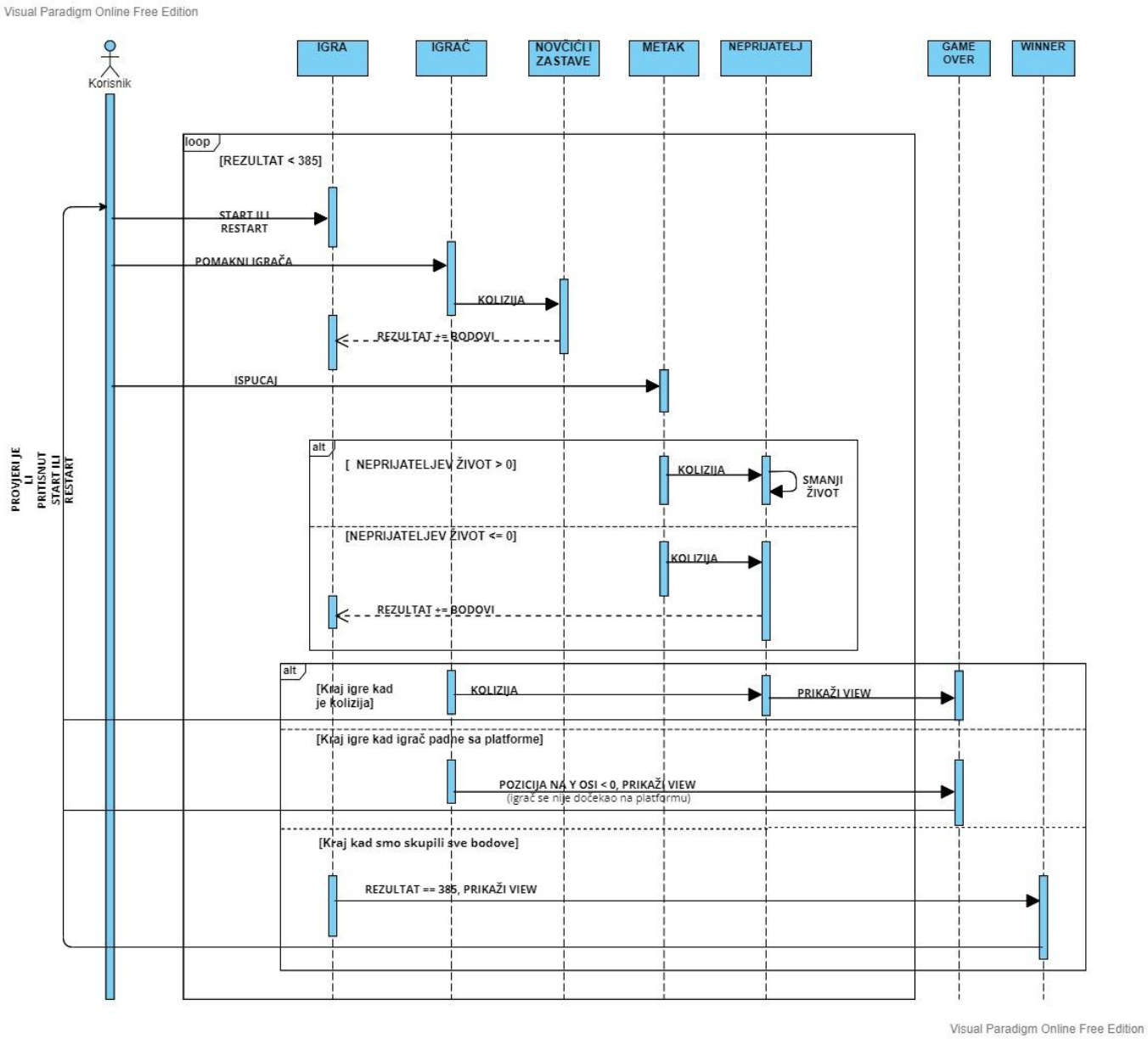


Slika 16 Dijagram klasa

5.4 Modeliranje vremenskog redoslijeda

Iako smo za sad odradili pozamašan posao modeliranja same igre uvesti ćemo dijagram redoslijeda (Slika 17) kojim pobliže modeliramo sami tijek igre kada će se koji zahtjev ili radnja

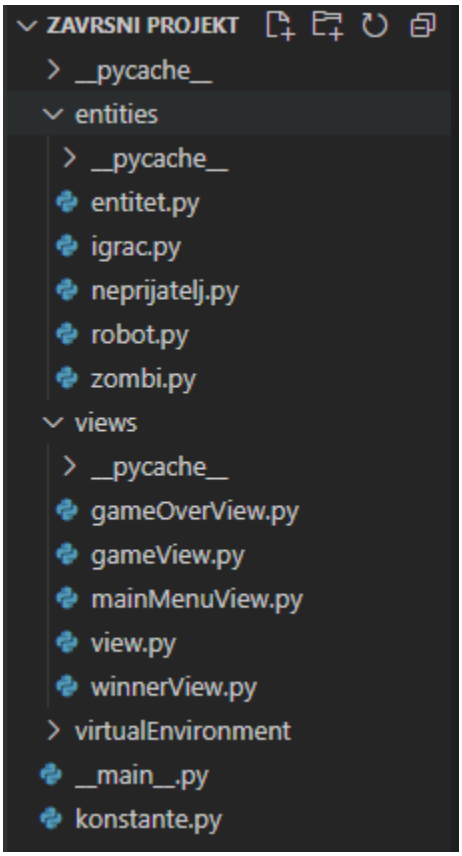
odviti. Njime smo definirali da će se igra odvijati dok ne skupimo svih tristo osamdeset i pet bodova u suprotnom prekidamo igru. Taj broj bodova nam predstavlja kraj gdje smo skupili sve novčiće i zastave i ubili sve neprijatelje. Igra se pokreće tako da ju pokrene sam korisnik koji pomiče igrača tipkama. Kad igrač dođe u koliziju sa novčićem povećavamo rezultat. Korisnik ispuca metak i kad on dođe u koliziju sa neprijateljem ili na rezultat dodamo bodove ili smanjimo neprijateljev život. Imamo dodatna dva scenarija kada ćemo prekinuti igru. Prvi se odnosi na koliziju igrača s neprijateljem, a drugi prekid se događa ako se igrač ne dočeka na platforme.



Slika 17 Dijagram redoslijeda

5.5 Izrada igre Platformer

Nakon što smo modeliranjem dobro opisali i predočili samu igru pregledati ćemo strukturu (Slika 18) samog projekta prije nego krenemo na sami kod.



Slika 18 Struktura projekta

Imamo dva direktorija: *entities* i *views*. U direktoriju *entities* imamo definirane klase za igrača i neprijatelje, a u direktoriju *views* nam se nalaze Python datoteke koje predstavljaju kod igre. Van direktorija *entities* i *views* imamo još dvije Python datoteke *__main__.py* i *konstante.py*.

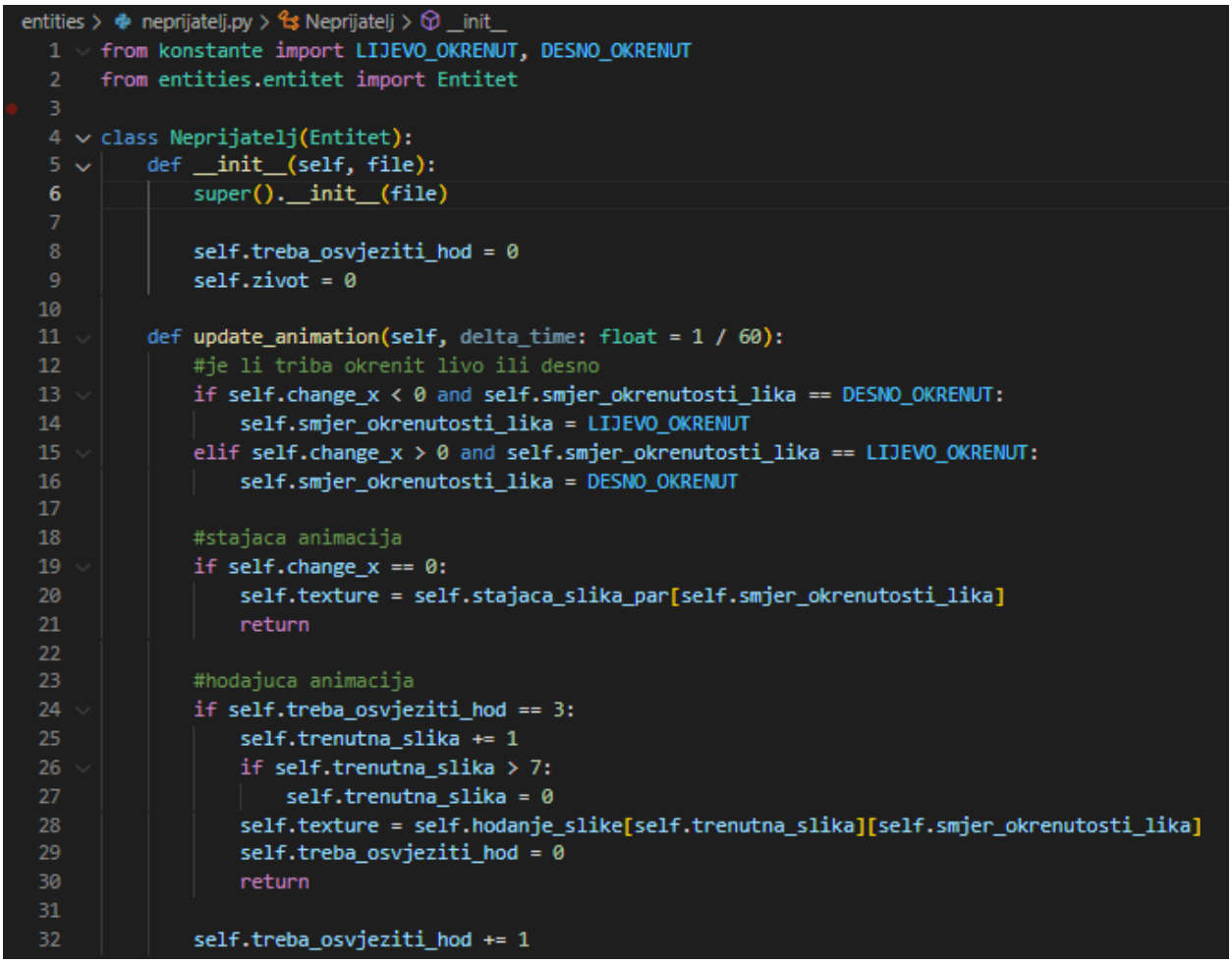
U klasama u mapi *entities* smo izdvojili određena svojstva kao što su *zivot* kod neprijatelja i *moze_skok, penjanje* i *na_ljestvama*. Osim tih svojstava u klasama *Entitet, Igrac* i *Neprijatelj* smo riješili problem animacije animiranih likova za što bolje iskustvo igranja. Kako imamo pristup ugrađenim resursima neki od njih su slike animiranih likova u različitim pokretima kao što su hodanje, skakanje, padanje. Te slike tretiramo kao nizove parova slika i ovisno o kojoj se klasi radi i njenim uvjetima izmjenjujemo pomoću indeksa niza. Ti indeksi su *trenutna_slika* koji označava broj trenutne slike i *smjer_okrenutosti_lika* koji označava na koju stranu animirani lik mora biti

okrenut. Najveći indeks *trenutna_slika* je sedam jer imamo osam slika hodanja, kod slika penjanja imamo dvije slike tako da taj indeks rješavamo cjelobrojnim dijeljenjem sa četiri. Za sva ostala stanja padanja, stajanja i skoka imamo po jednu sliku te se za njih moramo samo brinuti o indeksu *smjer_okrenutosti_lika*. Primjenu spomenutih indeksa i parove slika za animaciju možete vidjeti na slikama 19, 20 i 21

entities > entitet.py > ...

```python
import arcade

from konstante import PODESAVANJE_LIKA, DESNO_OKRENUT

def load_texture_pair(filename):
    '''ucitaj sliku i zrcali'''
    return [
        arcade.load_texture(filename),
        arcade.load_texture(filename, flipped_horizontally=True)
    ]

class Entitet(arcade.Sprite):
    def __init__(self, file):
        super().__init__()

        #defoltno okrenut lik
        self.smjer_okrenutosti_lika = DESNO_OKRENUT

        #koristi se za sekvence slika
        self.trenutna_slika = 0
        self.scale = PODESAVANJE_LIKA

        self.stajaca_slika_par = load_texture_pair(f"{file}_idle.png")
        self.skok_slika_par = load_texture_pair(f"{file}_jump.png")
        self.pad_slika_par = load_texture_pair(f"{file}_fall.png")

        #slike za hodanje
        self.hodanje_slike = []
        for i in range(8):
            slika = load_texture_pair(f"{file}_walk{i}.png")
            self.hodanje_slike.append(slika)

        #slike za penjanje
        self.penjanje_slike = []
        slika = arcade.load_texture(f"{file}_climb0.png")
        self.penjanje_slike.append(slika)
        slika = arcade.load_texture(f"{file}_climb1.png")
        self.penjanje_slike.append(slika)

        #set inicijalnu sliku (teksturu)
        self.texture = self.stajaca_slika_par[0]
```

Slika 19 Klasa Entitet



Slika 20 Klasa Neprijatelj

Slika 21 Klasa Igrac

Kao što je već rečeno *views* sadrži Python datoteke s kodom igre. Točnije sadrži „poglede“ igre, a ti „pogledi“ su *gameOverView.py*, *gameView.py*, *mainMenuView.py*, *winnerView.py* i *view.py*. *gameOverView*, *mainMenuView* (Slika 22) i *winnerView* su međusobno jako slične datoteke sa sličnim kodovima koji nakon što su pokrenuti na zaslonu prikazuju tekst i botune za pokretanje igre ili izlazak iz igre koje smo implementirali pomoću Aracdeovog GUI modula.

mainMenuView.py

views > mainMenuView.py > ...

```python
'''Main Menu View'''
import arcade
import arcade.gui

from views.view import View
from views.gameView import GameView
from views.gameOverView import GameOverView
from views.winnerView import WinnerView

class MainMenuView(View):
    def __init__(self):
        super().__init__()

        #vertikalni BoxGroup za botune
        self.v_box = None

    def setup(self):
        super().setup()
        self.ui_manager = arcade.gui.UIManager()

        self.setup_botune()

        self.ui_manager.add(
            arcade.gui.UIAnchorWidget(
                anchor_x="center_x", anchor_y="center_y", child=self.v_box
            )
        )

    def on_show_view(self):
        arcade.set_background_color(arcade.color.ALMOND)

    def setup_botune(self):
        self.v_box = arcade.gui.UIBoxLayout()

        play_botun = arcade.gui.UIFlatButton(text="Start Game", width=200)
        @play_botun.event("on_click")
        def on_click_play(event):
            if "game" not in self.window.views:
                self.window.views["game"] = GameView()
                self.window.views["game_over"] = GameOverView()
                self.window.views["winner"] = WinnerView()
            self.window.show_view(self.window.views["game"])
        self.v_box.add(play_botun.with_space_around(bottom=20))

        quit_botun = arcade.gui.UIFlatButton(text="Quit", width=200)
        @quit_botun.event("on_click")
        def on_click_quit(event):
            arcade.exit()
        self.v_box.add(quit_botun)

    def on_draw(self):
        arcade.start_render()

        arcade.draw_text(
            "Arcade platformer",
            self.window.width / 2,
            self.window.height - 125,
            arcade.color.ALLOY_ORANGE,
            font_size=44,
            anchor_x="center",
            anchor_y="center"
        )

        self.ui_manager.draw()
```

Slika 22 mainManeView.py

Najviše pažnje ćemo posvetiti *gameView* kodu jer se u njemu odvija cijela logika igre. U *gameView*-u imamo konstruktor *__init__()* (Slika 23) kojim inicijaliziramo sve potrebne varijable koje ćemo koristiti dalje u kodu. Neke od njih su *mapa*, *scena*, *igrac_sprite*, *rezultat*, zastavice za praćenje koja je tipka pritisnuta, zvukovi itd.

```
class GameView(View):
    def __init__(self):
        '''inicijalizacija igre i logika'''
        super().__init__()

        #prati koja je tipka trenutno pritisnuta
        self.lijevo_pritisnuto = False
        self.desno_pritisnuto = False
        self.gore_pritisnuto = False
        self.dolje_pritisnuto = False
        self.pucanje_pritisnuto = False
        self.reset_skoka = False

        #tile mapa
        self.mapa = None

        #scena
        self.scena = None

        #varijabla za igraca
        self.igrac_sprite = None

        #physics engine
        self.fizika = None

        #kamera za pomicanje backgrounda i kamera za gui elemente
        self.kamera = None
        self.gui_kamera = None

        self.kraj_mape = 0

        #za pracenje rezultata
        self.rezultat = 0

        #za pucanje
        self.moze_pucati = False
        self.pucanj_timer = 0

        #ucitaj zvukove
        self.novcic_zvuk = arcade.load_sound(":resources:sounds/coin1.wav")
        self.skok_zvuk = arcade.load_sound(":resources:sounds/jump1.wav")
        self.game_over = arcade.load_sound(":resources:sounds/gameover1.wav")
        self.pucanj_zvuk = arcade.load_sound(":resources:sounds/hurt5.wav")
        self.upucan_zvuk = arcade.load_sound(":resources:sounds/hit5.wav")
```

Slika 23 __init__()

Zatim slijedi metoda *setup()* kojom postavljamo cijelu igru. U njoj postavljamo vrijednosti u varijable koje smo prvotno naveli u metodi *__init__()* kao i postavljanje animiranih likova neprijatelja i igrača. Trebamo samo obratiti pažnju na varijablu *fizika* u metodi *setup()* jer njome postavljamo Arcadeov *Physics Engine* (Slika 24) za platformere kojim si znatno olakšavamo samo

kodiranje jer njime rješavamo kolizije sa platformama, zidovima, ljestvama i koji dodaje apstrakciju gravitacije u samu igru.

```
#kreiraj 'physics engine'
self.fizika = arcade.PhysicsEnginePlatformer(
    self.igrac_sprite,
    platforms=self.scena[POMICUCE_PLATFORME],
    walls=self.scena[PLATFORME],
    gravity_constant=GRAVITACIJA,
    ladders=self.scena[LJESTVE],
)
```

Slika 24 Physics Engine za platformere

Osim *PhysicsEngine*-a u *setup()* metodi postavljamo (Slika 25) mapu koju učitavamo iz ugrađenih resursa. Za njezino učitavanje moramo se sjetiti kako je ona napravljena uz pomoć programa *Tiled* i izvezena u Json obliku. Pri njenom učitavanju koristimo metodu *load_tilemap* kojoj šaljemo putanju do mape, broj kojim podešavamo veličinu mape i slojeve. Kada postavimo na *scenu* pomoću naredbe *arcade.Scene.from_tilemap()* ti slojevi će biti tretirani kao liste animiranih likova (eng. *SpriteLists*) i zbog toga im moramo omogućiti što bolju detekciju kolizija tako da postavimo *use_spatial_hash* na bool vrijednost *True.*

```
slojevi = {
    NOVCICI: {
        "use_spatial_hash": True
    },
    LJESTVE: {
        "use_spatial_hash": True
    },
    POMICUCE_PLATFORME: {
        "use_spatial_hash": True
    },
    PLATFORME: {
        "use_spatial_hash": True
    }
}

#ucitaj mapu (putanja u konstantama)
self.mapa = arcade.load_tilemap(MAPA, PODESAVANJE_PLOCICE, slojevi)

#inicijaliziraj scenu sa mapom sto ce automatski dodati slojeve
#iz mape kao SpriteLists u secnu u pravom redoslijedu
self.scena = arcade.Scene.from_tilemap(self.mapa)
```

Slika 25 Postavljanje mape u igru

Bitne metode za pomicanje samog igrača su *on_key_press()* (Slika 26), *on_key_release()* (Slika 27) i *process_keychange()* (Slika 28). Sa metodama *on_key_release()* i *on_key_press()* kontroliramo koje smo tipke točno pritisnuli ili otpustili pomoću zastavica koje smo prvotno inicijalizirali u konstruktoru. Zatim te iste zastavice koristimo za kontrolu uvjeta u metodi *process_keychange()* u kojoj točno definiramo gdje se animirani lik igrača pomiče ovisno o tome je li na ljestvama ili ne za što je zadužen *Physics Engine* sa svojom funkcijom *is_on_ladder()*.

```
211     def on_key_press(self, key, modifiers):
212         '''poziva se kad god je pritisnuta tipka'''
213         if key == arcade.key.UP or key == arcade.key.W:
214             self.gore_pritisnuto = True
215         elif key == arcade.key.DOWN or key == arcade.key.S:
216             self.dolje_pritisnuto = True
217         elif key == arcade.key.LEFT or key == arcade.key.A:
218             self.lijevo_pritisnuto = True
219         elif key == arcade.key.RIGHT or key == arcade.key.D:
220             self.desno_pritisnuto = True
221 
222         if key == arcade.key.SPACE:
223             self.pucanje_pritisnuto = True
224 
225         self.process_keychange()
```

Slika 26 Metoda on_key_press()

```
227     def on_key_release(self, key, modifiers):
228         '''poziva kad se pusti tipka'''
229         if key == arcade.key.UP or key == arcade.key.W:
230             self.gore_pritisnuto = False
231             self.reset_skoka = False
232         elif key == arcade.key.DOWN or key == arcade.key.S:
233             self.dolje_pritisnuto = False
234         elif key == arcade.key.LEFT or key == arcade.key.A:
235             self.lijevo_pritisnuto = False
236         elif key == arcade.key.RIGHT or key == arcade.key.D:
237             self.desno_pritisnuto = False
238 
239         if key == arcade.key.SPACE:
240             self.pucanje_pritisnuto = False
241 
242         self.process_keychange()
```

Slika 27 Metoda on_key_release()

```
182     def process_keychange(self):
183         '''poziva se kada mijenjamo tipku gore/dolje, kada smo i nismo na ljestvama,
184         kada skacemo, kada se micemo livo desno '''
185         #proces gore/dolje pri skakanju ili na ljestvama
186         if self.gore_pritisnuto and not self.dolje_pritisnuto:
187             if self.fizika.is_on_ladder():
188                 self.igrac_sprite.change_y = BRZINA_KRETANJA_IGRACA
189             elif (self.fizika.can_jump(y_distance=10) and not self.reset_skoka):
190                 self.igrac_sprite.change_y = BRZINA_SKOKA_IGRACA
191                 self.reset_skoka = True
192                 arcade.play_sound(self.skok_zvuk)
193         elif self.dolje_pritisnuto and not self.gore_pritisnuto:
194             if self.fizika.is_on_ladder():
195                 self.igrac_sprite.change_y = -BRZINA_KRETANJA_IGRACA
196 
197         #proces gore/dolje na ljestvama bez pomicanja
198         if self.fizika.is_on_ladder():
199             if not self.gore_pritisnuto and not self.dolje_pritisnuto:
200                 self.igrac_sprite.change_y = 0
201             elif self.gore_pritisnuto and self.dolje_pritisnuto:
202                 self.igrac_sprite.change_y = 0
203 
204         #proces livo desno
205         if self.desno_pritisnuto and not self.lijevo_pritisnuto:
206             self.igrac_sprite.change_x = BRZINA_KRETANJA_IGRACA
207         elif self.lijevo_pritisnuto and not self.desno_pritisnuto:
208             self.igrac_sprite.change_x = -BRZINA_KRETANJA_IGRACA
209         else:
210             self.igrac_sprite.change_x = 0
```

Slika 28 Metoda process_keychange()

U metodi *on_update()* se odvija cijela logika igre, animacija animiranih likova, pomicanje animiranih likova i kolizije animiranih likova. Samo pomicanje igrača (Slika 29) odrađujemo uz pomoć naredbe *update()* u kombinaciji sa *can_jump()* i *is_on_ladder() Physics Enginea* i metode *process_keychange()*.

```
def on_update(self, delta_time):
    '''micanje i logika'''
    #pomici igraca uz pomoc physic engina
    self.fizika.update()

    #updateaj animacije playera
    #skakanje
    if self.fizika.can_jump():
        self.igrac_sprite.moze_skok = False
    else:
        self.igrac_sprite.moze_skok = True

    #penjanje na skale
    if self.fizika.is_on_ladder() and not self.fizika.can_jump():
        self.igrac_sprite.na_ljestvama = True
        self.process_keychange()
    else:
        self.igrac_sprite.na_ljestvama = False
        self.process_keychange()
```

Slika 29 Pomicanje igrača

Također smo spomenuli animacije (Slika 30) neprijatelja i igrača koje ažuriramo uz pomoć naredbe *update_animation()* u metodi *on_update()*.

```
301         #updateaj animacije ostalih slojeva
302         self.scena.update_animation(
303             delta_time,
304             [
305                 NOVCICI,
306                 SLOJ_POZADINA,
307                 IGRAC,
308                 NEPRIJATELJI
309             ]
310         )
```

Slika 30 Ažuriranje animacija animiranih likova.

Kolizija između metka i neprijatelja (Slika 31) je riješena na način da kolizije tretiramo kao jednu listu kolizija između metka sa neprijateljima i platformama. Ako postoji ta lista u svakom slučaju uklonimo metak sa zaslona i zatim provjeravamo je li u koliziji sudjelovao neprijatelj. Ako je sudjelovao, ovisno o tome koliko mu je ostalo života radimo određene radnje. Uklanjamo neprijatelja iz liste i sa zaslona i dodajemo sto bodova u rezultat ako mu je život došao na nulu. U slučaju da je imao još života onda mu isti smanjimo za dvadeset i pet.

```
for metak in self.scena.get_sprite_list(METCI):
    hit_lista = arcade.check_for_collision_with_lists(
        metak,
        [
            self.scena.get_sprite_list(NEPRIJATELJI),
            self.scena.get_sprite_list(PLATFORME),
            self.scena.get_sprite_list(POMICUCE_PLATFORME),
        ],
    )

    if hit_lista:
        metak.remove_from_sprite_lists()

        for kolizija in hit_lista:
            if (
                self.scena.get_sprite_list(NEPRIJATELJI)
                in kolizija.sprite_lists
            ):
                # The collision was with an enemy
                kolizija.zivot -= STETA_METKA

                if kolizija.zivot <= 0:
                    kolizija.remove_from_sprite_lists()
                    self.rezultat += 100

                # Hit sound
                arcade.play_sound(self.upucan_zvuk)
        return
```

Slika 31 Kolizija metka sa neprijateljem

Još su nam ostale kolizije između igrača i neprijatelja i igrača i novčića i zastavica (Slika 32). Radimo ih na identičan način kao i koliziju metka sa neprijateljem. Kolizije tretiramo kao listu i ako je kolizija bila sa neprijateljem tada dolazi do kraja igre i prikazuje nam se *gameOverView*. U slučaju da je riječ o novčiću ili zastavici onda iz Json oblika mape iz sloja *NOVCICI* izvlačimo svojstvo *Points* koje dodajemo na rezultat.

```
379         igrac_kolizijska_lista = arcade.check_for_collision_with_lists(
380             self.igrac_sprite,
381             [
382                 self.scena.get_sprite_list(NOVCICI),
383                 self.scena.get_sprite_list(NEPRIJATELJI),
384             ]
385         )
386 
387         #petlja za svaki novcic koji dotaknemo i makni ga
388         for kolizija in igrac_kolizijska_lista:
389             if (self.scena.get_sprite_list(NEPRIJATELJI) in kolizija.sprite_lists):
390                 arcade.play_sound(self.game_over)
391                 self.window.show_view(self.window.views["game_over"])
392                 return
393             else:
394                 if "Points" not in kolizija.properties:
395                     print("Upozorenje, skupio novcic bez Points svojstva!")
396                 else:
397                     points = int(kolizija.properties["Points"])
398                     self.rezultat += points
399                 kolizija.remove_from_sprite_lists()
400                 arcade.play_sound(self.novcic_zvuk)
```

Slika 32 Kolizija igrača sa novičićima, zastavama i neprijateljima

U slučaju da se s igračem ne dočekamo na platformi tada nam se prikaže *gameOverView* (Slika 33), a u slučaju da smo skupili sve novčiće, zastavice i ubili sve neprijatelje što je jednako tristo osamdeset pet bodova tada nam se prikazuje *winnerView* (Slika 33).

```
402         if self.igrac_sprite.bottom < 0:
403             arcade.play_sound(self.game_over)
404             self.window.show_view(self.window.views["game_over"])
405             return
406 
407         if self.rezultat == 385:
408             self.window.show_view(self.window.views["winner"])
```

Slika 33 Dva moguća kraja igre

Van direktorija *view* i *entities* imamo glavnu ulaznu točku (Slika 34) cijele igre. To je datoteka *__main__.py* kojom stvaramo prozor i prikazujemo *mainMenuView*.

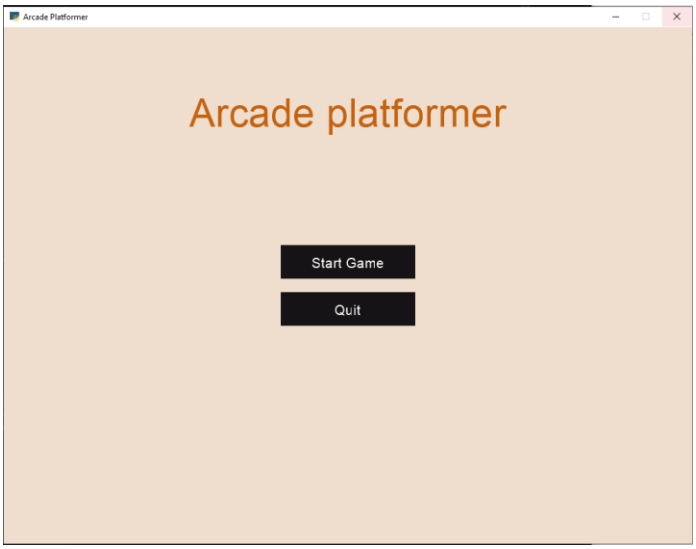
```python
import arcade

from konstante import VISINA_PROZORA, SIRINA_PROZORA, NASLOV_PROZORA
from views.mainMenuView import MainMenuView
class GameProzor(arcade.Window):
    def __init__(self):
        super().__init__(SIRINA_PROZORA, VISINA_PROZORA, NASLOV_PROZORA, resizable = False)

        self.views = {}
        self.views["main_menu"] = MainMenuView()

def main() -> None:
    window = GameProzor()
    window.show_view(window.views["main_menu"])
    arcade.run()

if __name__ == "__main__":
    main()
```

Slika 34 __main__.py datoteka

Osim *__main__.py* datoteke imamo i datoteku *konstante.py* u kojoj smo posebno izdvojili brojčane vrijednosti i stringove od cijelog koda da bi imali što manje grešaka tijekom izrade igre. Također smo si olakšali postupak promjena u samom kodu jer ne moramo tražiti svako spominjanje neke konstante da bi ju promijenili u kodu već ju samo promijenimo u spomenutoj datoteci.

5.6 Rad igre Platformer

Pri pokretanju igre nam se prikaže *mainMenuView* (Slika 35) sa naslovom „Arcade platformer“ i dva gumba „Start Game“ kojim pokrećemo igru i „Quit“ kojim prekidamo izvršavanje programa.

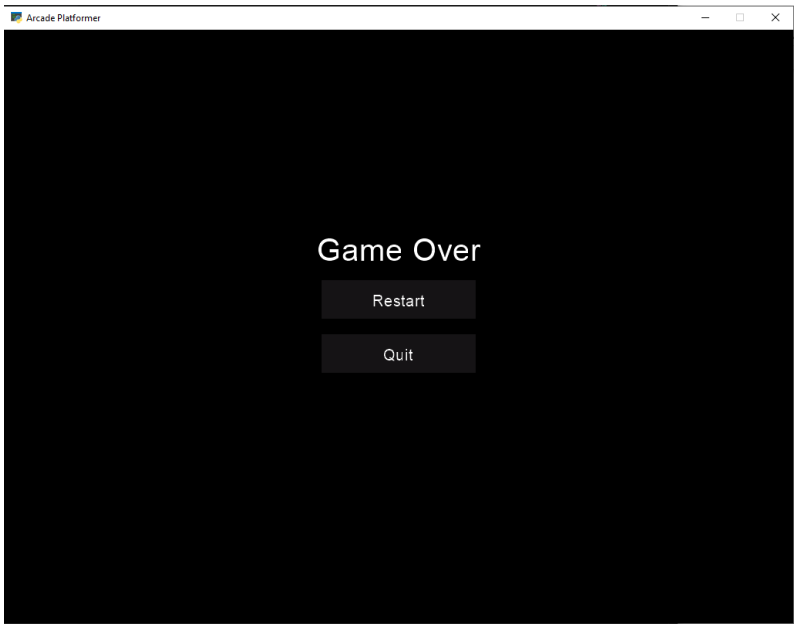


Slika 35 Prikaz na zaslonu mainMenuView.py

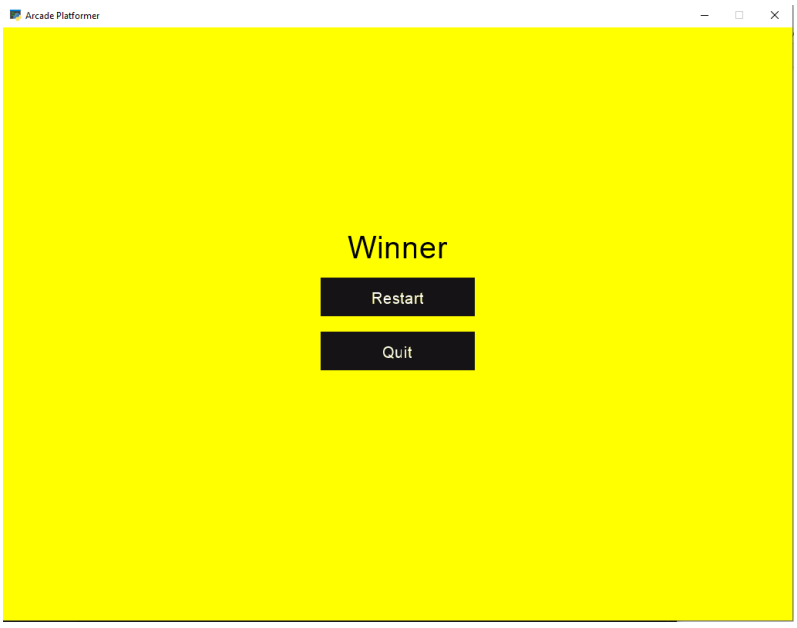
Nakon što smo pokrenuli igru pritiskom na gumb „Start Game“ na zaslonu prikazuje nam se *gameView.py* (Slika 36) i možemo slobodno igrati igru prema već opisanom načinu.

Slika 36 Prikaz gameView.py na zaslonu

U slučaju da tijekom igranja izgubimo prikazuje nam se *gameOverView.py* (Slika 37), a u slučaju da smo pobijedili prikazuje nam se *winnerView.py* (Slika 38) na kojima imamo dva botuna za ponovno pokretanje igre i za prekid izvršavanja programa.



Slika 37 gameOverView.py na zaslonu



Slika 38 winnerView.py na zaslonu

6. Zaključak

U ovom završnom radu je prikazana izrada 2D arkadne igre Platformer kojom smo prikazali rad s Pythonovim 2D okvirom za izradu igara Arcade. Iako postoji mnoštvo okvira na odabir od kojih su neki i zasebni programi dizajnirani na nekom puno prikladnijem programskom jeziku za izradu igara, Arcade nam nudi jednostavnost koja je u kombinaciji sa programskim jezikom Python savršena za osobe koje se žele uhvatiti u koštac sa izradom igara.

Sami proces izrade igara nipošto nije jednostavan jer je ono pozamašan proces koje zahtijeva planiranje. Prije nego započnemo cijeli proces trebamo se informirati što to znači izrađivati igru, kakvu vrst programiranja ono zahtijeva i kako treba teći taj proces. Da bi smo započeli s tim pothvatom prvo pitanje koje nam se postavlja je kakvu igru želimo napraviti. Nakon njega definiramo što radimo u toj igri, je li ima priču, koje likove koristimo, kako pokrećemo lika, postoje li neprijatelji, što se događa kad se igrač i neprijatelji susretnu, itd. Ako na svako pitanje takve vrste imamo odgovor znamo da smo na dobrom putu i da ćemo s lakoćom napraviti prikladne modele dijagrama. Nakon izrade dijagram tek možemo krenuti na odabir programskog jezika i okvira koji ovise o vrsti igre koju radimo. No prije nego započnemo s kodiranjem moramo se upoznati s odabranim okvirom, njegovim konceptima i mogućnostima da bi cijeli proces bio olakšan u suprotnom nailazimo na brdo problema koji će završiti odustajanjem od projekta jer smo u neznanju.

Izradu igara ne trebamo gledati kao nekakav jako komplicirani proces u kojem završna verzija projekta mora biti u skladu s današnjim igricama. Na cijeli proces treba gledati kao putanju napretka gdje krećemo od nečeg jednostavnog da se upoznamo sa procesom stvaranja ka nečem kompliciranijem s čime se možemo ponositi.

7. Literatura

[1] T. T. J. Goldsmith i E. R. Mann.Sjedinjene Američke Države Patent 2,455,992, 1948..

[2] »Wikipedia,« [Mrežno]. Available: https://en.wikipedia.org/wiki/Thomas_T._Goldsmith_Jr..

[3] J. M. Graetz, »The Origin of Spacewar,« *Creative Computing magazine (Volume 7, Number 8),* pp. 56-67, 1981..

[4] I. Engineering, »Interesting Engineering,« 2 Studeni 2016. [Mrežno]. Available: https://interestingengineering.com/innovation/how-game-engines-work.

[5] T. CodeWizardsHQ, »Ultimate Guide to Python Game Development: Best Python Games and Engines and How to Program Your Own,« 26 Kolovoz 2021. [Mrežno]. Available: https://www.codewizardshq.com/python-games/.

[6] J. Fincher, »Top Python Game Engines: Real Python,« 4 Svibanj 2022. [Mrežno]. Available: https://realpython.com/top-python-game-engines/.

[7] D. Pope, »adventurelib - easy text adventures: adventurelib,« 2016. [Mrežno]. Available: https://adventurelib.readthedocs.io/en/stable/.

[8] D. Pope, »Overview, Non-English speakers: adventurelib,« 2016. [Mrežno]. Available: https://adventurelib.readthedocs.io/en/stable/overview.html.

[9] »Introduction to Panda3D: Panda3D,« 2019. [Mrežno]. Available: https://docs.panda3d.org/1.10/python/introduction/index.

[10] anshitaagarwal, »Arcade Library in Python: GeeksforGeeks,« 2 Studeni 2020. [Mrežno]. Available: https://www.geeksforgeeks.org/arcade-library-in-python/.

[11] »Tiled documentation,« [Mrežno]. Available: https://doc.mapeditor.org/en/stable/manual/introduction/#about-tiled.

[12] »Layer Types,« [Mrežno]. Available: https://doc.mapeditor.org/en/stable/manual/layers/#layer-types.

[13] »Get startded with Unreal Engine,« [Mrežno]. Available: https://www.unrealengine.com/en-US/download.

[14] »Comparison of Game Engines 2020,« [Mrežno]. Available: https://indiegamedev.net/2020/02/11/comparison-of-game-engines-2020/.

8. Popis slika

Slika 1 Igra „Spacewar!“ na PDP-1 3
Slika 2 Igra „Wolfenstein 3D“ 4
Slika 3 Službeni logo PyGame okvira 8
Slika 4 Primjer zapovijedi „brush teeth“ 9
Slika 5 Logo Panda3D i 3D rezultat korištenja okvira 10
Slika 6 Korištenje koordinatnog sustava u Arcade okviru 13
Slika 7 Primjeri oblika koji se mogu nacrtati koristeći Arcade 14
Slika 8 Proceduralni kod za crtanje ispunjenog kruga 15
Slika 9 Objektno-orijentirani kod za cratnje ispunjenog kruga 16
Slika 10 Neki od ugrađenih resursa 18
Slika 11 Primjer slojeva u progamu Tiled 19
Slika 12 Prikaz Properties okvira 19
Slika 13 Dijagram slučajeva korištenja 22
Slika 14 Dijagram aktivnosti ovisno o ulaznim podacima korisnika 23
Slika 15 Dijagram aktivnosti za tijek igre i kolizije 24
Slika 16 Dijagram klasa 25
Slika 17 Dijagram redoslijeda 26
Slika 18 Struktura projekta 27
Slika 19 Klasa Entitet 28
Slika 20 Klasa Neprijatelj 29
Slika 21 Klasa Igrac 30
Slika 22 mainManeView.py 31
Slika 23 __init__() 32
Slika 24 Physics Engine za platformere 33
Slika 25 Postavljanje mape u igru 33
Slika 26 Metoda on_key_press() 34
Slika 27 Metoda on_key_release() 34
Slika 28 Metoda process_keychange() 35
Slika 29 Pomicanje igrača 35

Slika 30 Ažuriranje animacija animiranih likova. 36
Slika 31 Kolizija metka sa neprijateljem 36
Slika 32 Kolizija igrača sa novičićima, zastavama i neprijateljima..... 37
Slika 33 Dva moguća kraja igre..... 37
Slika 34 __main__.py datoteka..... 38
Slika 35 Prikaz na zaslonu mainMenuView.py 39
Slika 36 Prikaz gameView.py na zaslonu..... 39
Slika 37 gameOverView.py na zaslonu 40
Slika 38 winnerView.py na zaslonu..... 40