Minolovac u Pythonu

Budiša, Petra

Undergraduate thesis / Završni rad

2023

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

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:166:272784

Rights / Prava: Attribution 4.0 International/Imenovanje 4.0 međunarodna

Download date / Datum preuzimanja: **2024-05-09**

Repository / Repozitorij:

Repository of Faculty of Science

zir.nsk.hr

UNIVERSITY OF SPLIT

dabar

DIGITALNI AKADEMSKI ARHIVI I REPOZITORIJI

PRIRODOSLOVNO-MATEMATIČKI FAKULTET U SPLITU

ZAVRŠNI RAD

MINOLOVAC U PYTHONU

Mentor: doc. dr. sc. Ani Grubišić

Student: Petra Budiša

Temeljna dokumentacijska kartica

Završni rad

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

MINOLOVAC U PYTHONU

Petra Budiša

SAŽETAK

Glavni cilj završnog rada je prikazati i objasniti logiku aplikacije Minolovac. Pomoću Python jezika isprogramirana je cijela aplikacija, koristeći Pygame biblioteku zajedno sa bibliotekom Random, kako bi se moglo nasumičnim redosljedom postavljati mine na igraču ploču. Osim samog programa korišten je i grafički jezik za unificirano modeliranje (eng. Unified modeling language – UML) kako bi se dijagramima prikazala sama logika, te aktivnost kroz cijelu igru. Za samo oblikovanje aplikacije također je dijagramom klasa prikazano kakvi su odnosi između njih i kojim se redom pozivaju. Igra je osmišljena da prati svaki korisnikov potez te mjenja stanje zaslona.

Ključne riječi: Pygame, oblikovanje igre, korisnikov potez, prikaz zaslona

Rad je pohranjen u knjižnici Prirodoslovno-matematičkog fakulteta, Sveučilišta u Splitu

Rad sadrži: 37 stranica, 32 grafičkih prikaza, 0 tablica 4 literaturnih navoda. Izvornik je na hrvatskom jeziku.

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

Ocjenjivači: **Dr.sc. Ani Grubišić,** *izvanredni profesor Prirodoslovno-matematičkog fakulteta, Sveučilišta u Splitu - predsjednica*

Dr. sc. Branko Žitko, *izvanredni profesor Prirodoslovno-matematičkog fakulteta, Sveučilišta u Splitu - član*

Ines Šarić-Grgić , *asistent Prirodoslovno-matematičkog fakulteta, Sveučilišta u Splitu* - članica

Rad prihvaćen: Rujan,2023.

Basic documentation card

Thesis

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

MINESWEEPER IN PYTHON

Petra Budiša

ABSTRACT

The main goal of the final paper is to show and explain the logic of the Minesweeper application. Using the Python language, the entire application was programmed, using the Pygame library together with the Random library, in order to be able to place mines on the player board in a random order. In addition to the program itself, a graphical language for unified modeling (Unified modeling language - UML) was also used in order to show the logic and activity throughout the entire game with diagrams. For the design of the application itself, the class diagram also shows the relationships between them and the order in which they are called. The game is designed to follow the user's every move and change the state of the screen..

Key words: Pygame, game design, user input, screen display

Thesis deposited in library of Faculty of science, University of Split

Thesis consists of: 37 pages, 32 figures, 0 tables and 4 references

Original language: Croatian

Mentor: **Ani Grubišić, Ph.D.** *Associate Professor of Faculty of Science, University of Split*

Reviewers: **Ani Grubišić, Ph.D.** *Associate Professor of Faculty of Science, University of Split - president*

Branko Žitko, Ph.D. *Associate Professor of Faculty of Science, University of Split - member*

Ines Šarić-Grgić *Assistant of Faculty of Science, University of Split-member*

Thesis accepted: September 2023.

SADRŽAJ:

1. UVOD 4

2. PYTHON 5

2.1. Pygame 5

3. MINOLOVAC 6

3.1. Način rješavanja 7

3.2. Modeliranje zahtjeva 7

3.3. Modeliranje tijeka 9

3.3.1. Pod aktivnost - generiranje igre 10

3.3.2. Pod aktivnost – korisnikov potez 11

3.4. Modeliranje logičke strukture 12

3.4.1. Klasa Menu 12

3.4.2. Klasa Leveli 15

3.4.3. Sprite klasa Tile 18

3.4.4. Sprite klasa Board 20

3.4.5. Klasa Game 23

3.5. Dijagram redosljeda i dijagram klasa 27

3.6. Modeliranje izgleda 29

3.6.1. Početni zaslon 30

3.6.2. Zaslon sa razinama 30

3.6.3. Zaslon sa igrom 31

4. ZAKLJUČAK 36

5. LITERATURA 37

1. Uvod

Programski jezik python jako je jednostavan za naučiti i shvatiti kako se koristi. Njime se koriste početnici zbog toga što mu je sintaksa jako jednostavna i ne zahtjevna kao kod drugih programskih jezika. Lako je napraviti bilo kakvu vrstu aplikacije u njemu zbog zoga što postoji jako mnogo biblioteka koje možemo koristit.

Za ovaj rad koristio se Pygame koji je namjenje za stvanje pokretnih igara, no puno lakše je isprogramirati statične 2D igre kao što je ovaj projekt. Minolovac ima jako specifičnu logiku, te pravila igranja. Kroz ovaj rad prikazat ćemo dijagramima i isječcima koda točno logiku iza same igre, odnoso aplikacije.

Za početak upoznat ćemo se samim programskim jezikom te bibliotekom Pygame koja je korištena. Nakon što se upoznamo sa alatom koji ćemo koristi moramo dobro proučiti igru te njenu funkcionalnost i pravila rješavanja.

Aplikacija je napravljenja u 3 Python datoteke. Prva glavna datoteka je Main.py u kojoj nam se nalaze fuunkcije koje reprezentiraju zaslone, odnosno Menu, odabir razine i sama igra. Sljedeća datoteka je Sprite.py u kojoj nam se nalaze sprite klase Tile i Board koje su nam potrebne da bi mogli generirat samu igricu. Osim njih nalaze nam se i klasa Botun koju pozivamo na zaslon u klasi Manu i Leveli kako bi mogli prebacivati se na sljedeći ili prethodni zaslon. Zadnja datoteka je Postavke.py gdje nam se nalaze sve slike potrebne za igru, te veličine kockica i broj stupaca i redaka.

Za vizualiziranje i dokumentiranje logike same aplikacije koristio se grafički jezik za unificirano modeliranje (eng. *unified modeling language* - UML) dijagramima. Alat se nalazi na internetskoj stranici draw.io, pomoću koje su prikazani svi dijagrami u ovom radu.

2. Python

Python je objektno-orijentirani programski jezik kojeg je osmisli Nizozemac Guido van Rossum, počevši s implementacijom 1989.godine. Glavne odlike jezika su produktivnost, portabilnost i modularnost [1].

Zbog svoje jednostavne sintakse što rezultira sa jako malo programskih linija, jezik ostvaruje produktivnost. Također postoji jako malo situacija gdje će sami program prijavljivati grešku te je lakši za korištenje i razumijevanje. S obziom da se Python može instalirati na bilo koje računlo sa bilo kojim operacijskim sustavom dobiva portabilnost koju je zamislio Guido van Rossum. Modularnost dobiva tako što postoji gomila biblioteka koje se mogu koristit tijekom programiranja python aplikacije. U sljedećem poglavlju ćemo se upoznati sa Pygame-om koji sam koristila za izradu ove aplikacije.

2.1. Pygame

Pygame je osnovno osmišljen za programiranje videoigara sa svojim višeplatformskim skupom Python modula. U sebi ima ugrađeno sve potrebne funkcije za obilježavanje korisnikovog poteza, pygame.events, ja sa koristila samo klik miša, no postoji za svaku tipku na tipkovnici svoj posebni event.type. Osim eventova sastoji se od dosta pomoćnih funkcija, koje za stvaranje ovaj projekt, nisu bile potrebne zbog toga što je Minolovac jako statična aplikacija. Pygame-om se većinom programiraju animirane video-igrice sa puno animacija i pokret na zaslonu, no postoji i puno statičkih igara, kao što je Minolovac, napravljenih pomoću njega [2].

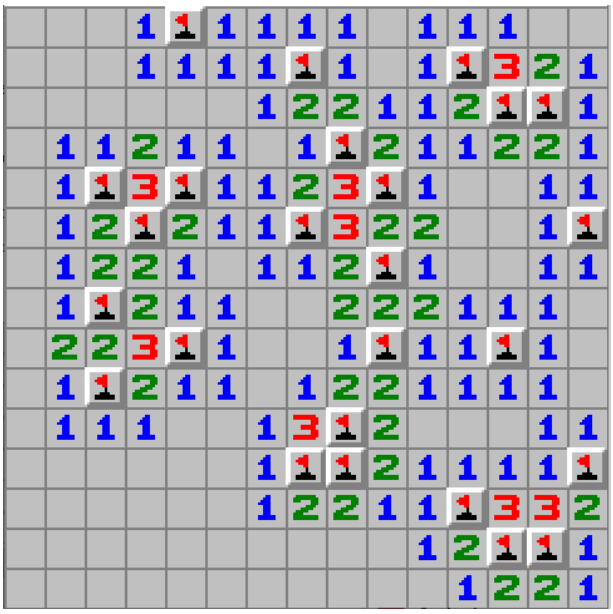
Za početak treba instalirati sa stranice https://www.pygame.org/news biblioteku na svoje računalo. Postoji dosta verzija, no ja sam koristila 2.5.0. Nakon instaliranja dovoljno je samo pozvati biblioteku u svojoj aplikaciji pomoću naredbe import pygame

3. Minolovac

Minolovac je logička igra namjenjena za igranje na računalu. Sastoji se od kvadratne igrače ploče na kojoj se nalaze zatvorena polja. Ispod nekoliko polja nalaze se nasumično postavljene mine koje se trebaju pronaći. Osim mina nalaze se brojevi, od 1 do 8, koji zapravo prikazuju broj mina koji se nalaze oko njih i obična prazna polja [3].

Cilj igre je otvoriti sva polja ispod kojih se ne nalazi mina kao što je prikazano na Slici 1. U početku igre sva su polja zatvorena, te korisnik mora nasumično odabrati prvo polje koje želi otvoriti. Naravno u nekim slučajevima korisnik može odma otvoriti polje sa minom i tada je igra gotova prije nego što je počela. Najčešće korsinik otvori polje sa brojem, ali postoji i mogućnost da otvori prazno polje, koje zapravo puno pomaže jer otvara sva prazna polja oko sebe dok ne dođe do polja sa brojem. Kako bi se igra lakše rješavala ponuđena je opcija zastavice koju postavite, desnim klikom miša, na polje ispod kojeg mislite da se nalazi mina.

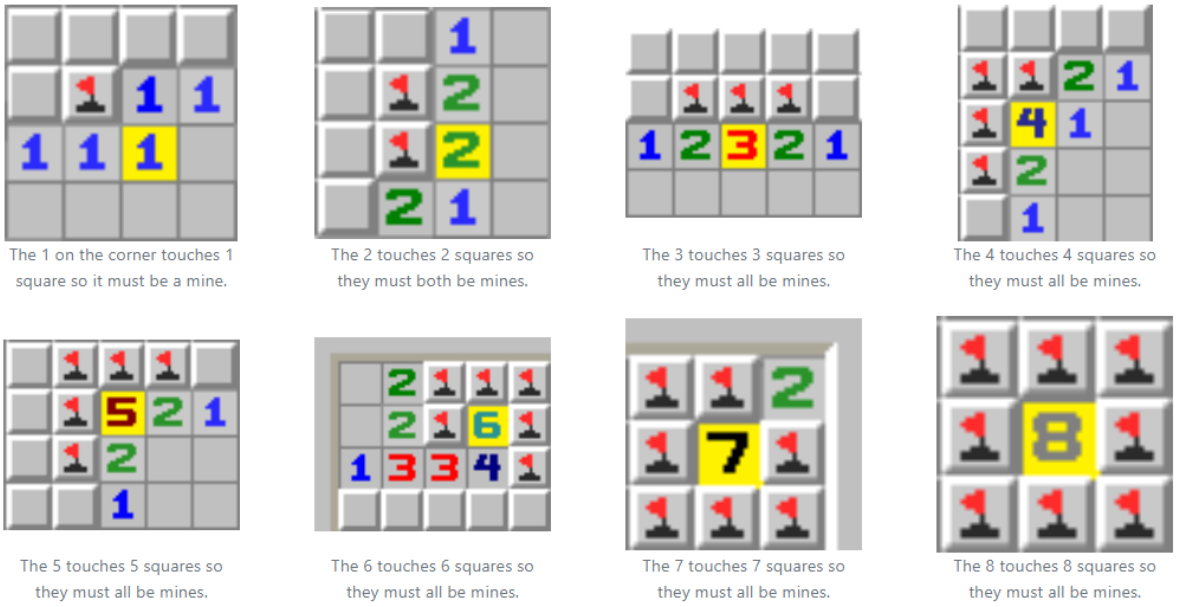
Ne postoji jedna verzija igre, odnosno broj polja i broj mina se može mijenjati i na taj način se igra otežava ili olakšava. Standardno je da se broj stupaca i broj redaka podudaraju, no ni toga se ne treba uvijek pridržavati.



Slika 1 Prikaz pobjede na Minolovcu

3.1. Način rješavanja

Postoji samo jedan logički način rješavanja igre i to je koristeći polja sa brojevima. Takva polja nam označavaju da se unutar 3x3 kvadrata, sa tim poljem u sredini, nalazi n broj mina oko njega i ostatak polja sa brojevima ili prazna polja. Naravno najlakše je kada polje prikazuje broj 8 jer to znači da su sve oko njega mine. Pomoću susjednih polja sa brojevima lako se može zaključiti na kojem se polju nalazi mina, a tek pogotovo ako su susjedna polja prazna. Na Slici 2 se nalaze najlakši primjeri za svaki broj [4].

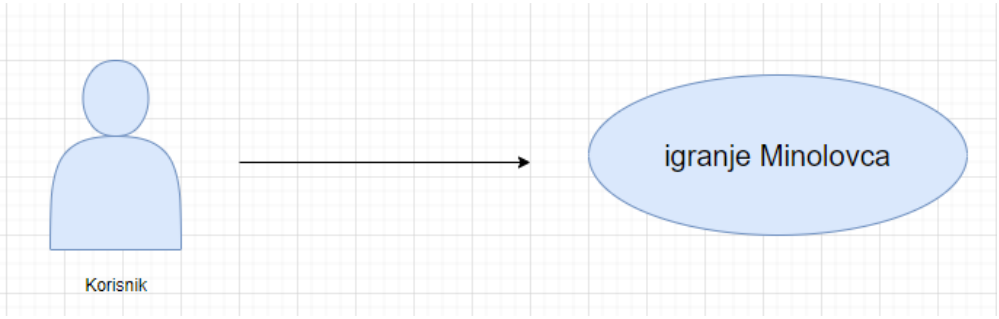


Slika 2 Prikaz pravila rješavanja

3.2. Modeliranje zahtjeva

Koristeći stranicu draw.io prikazat ćemo funkcionalnosti igre pomoću dijagrama korištenja.

Mogućnost jednog korisnka da igra Minolovca je osnovna funkcionalnost, te je prvi dijagram jako jednostavan i sastoji se od jednog korisnika i osnovne funkcionalnosti koji možemo vidjeti na Slici 3.

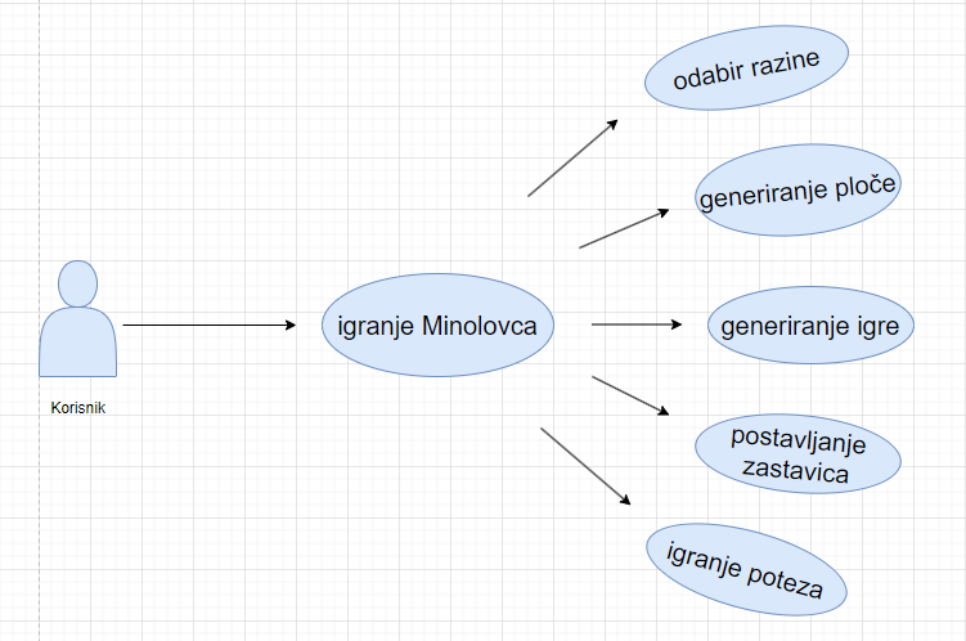


Slika 3 Dijagram korištenja

Dodatne funkcionalnosti igre su

- Odabir razine – imamo 6 razina gdje svaka razina ima određen broj mina koje treba pronaći
- Generiranje igrače ploče – ploča je napravljena ka prikaz 15x15 slika polja, koja se mogu otvoriti, lijevim klikom miša, i na koja možemo, desnim klikom miša, postaviti zastavicu
- Generiranje igre – postavljaju se nasumičnim redom mine, te se provjeravaju polja oko njih i ta polja postaju polja sa brojevima
- Postavljanje zastavice – zastavice se mogu, a i ne moraju, postavljati i brisati desnim klikom miša, te ako smo otkrili sva polja s brojevima automatski se na kraju postavljaju na polja sa minom
- Igranje poteza – lijevim klikom miša odabiremo polje koje želimo otvoriti

Sljedećim dijagramom, Slika 4, ćemo prikazati prethodno navedene dodatne funkcionalnosti.

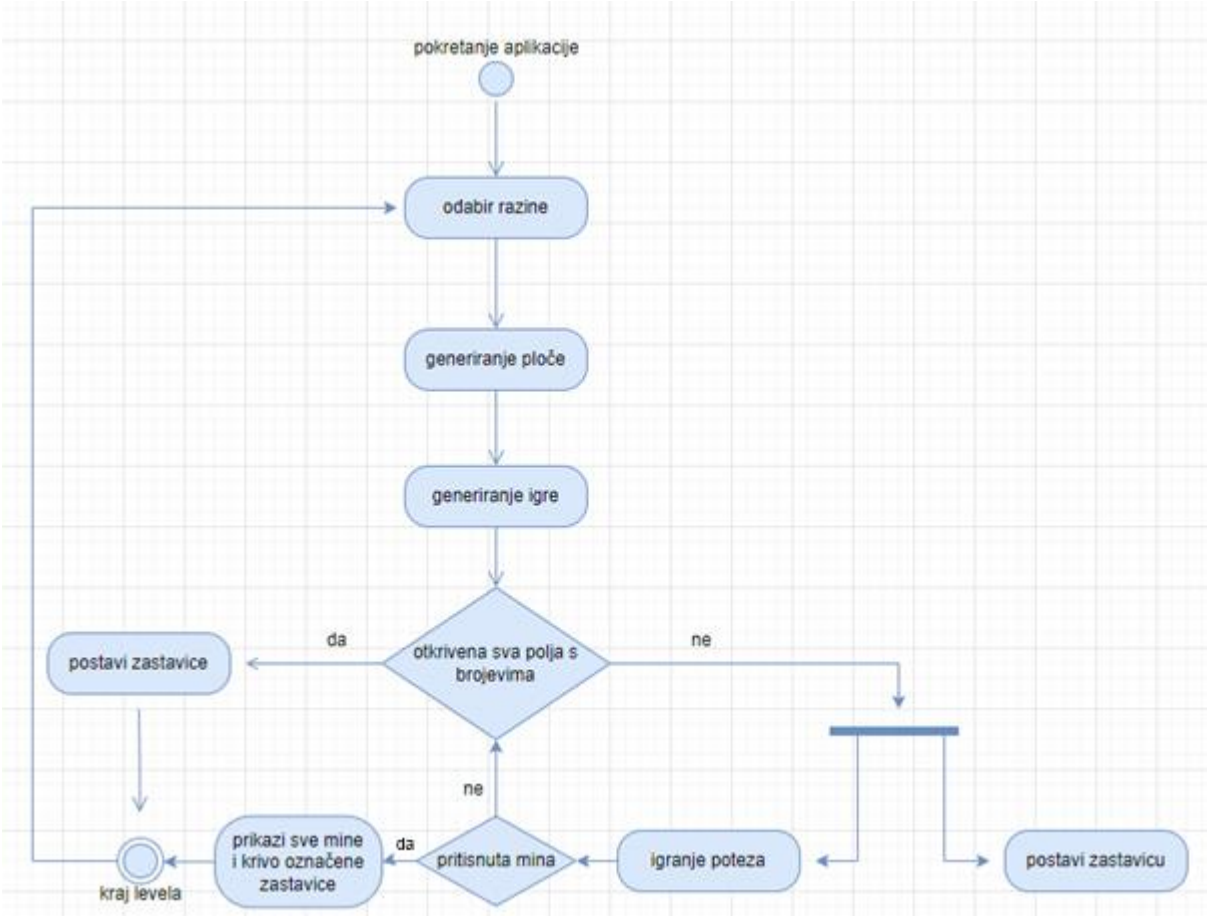


Slika 4 Dijagram korištenja sa dodatnim funnkcionalnostima

3.3. Modeliranje tijeka

Kod ovakvih kompleksnih projekata prethodno nacrtani dijagrami funkcionalnosti nisu dovoljni kako bi se prikazalo igranje igre tj. tijek događaja igrajući igru. Sljedećim dijagramima aktivnosti, na Slici 5, prikazat ćemo svaku aktivnost, kako slijde jedna drugu, te kako postoje aktivnosti sa uvjetima

Najjednostavniji tijek je da se generira igra i da se igraju potezi sve dok se ne ispuni glavni uvjet, a to je da su otvorena sva polja sa brojevima bez da smo otvorili minu. Igra provjerava jesmo li ispunili glavni uvjet, ako je uvjet ispunjen postavit će se zastavice koje nismo postavili i onda smo gotovi sa trenutnom razinom i možemo odabrat sljedeću, ako nije dopusti korisniku igranje poteza. Međutim igra nakon svakog poteza provjerava jesmo li otvorili minu. U slučaju da smo je otvorili igra nam prikazuje sva polja sa minama i polja koja smo označili sa zastavicom koja nisu točna, ako ih ima. Naravno ako nismo otvorili minu vraćamo se vidit jeli sada ispunjen glavni uvjet i tako sve dok se uvjet ne ispuni.



Slika 5 Dijagram aktivnosti Minolovca

U prethodnom dijagamu također nije sve prikazano već dovoljno da se shvati tijek događaja. Malo je detaljniji od dijagrama funkcionalnosti ali lakši za razumijeti. Kako bi prikazali sve aktivnosti u sljedećim pod poglavljima ćemo sve detaljno prikazati i opisati

3.3.1. Pod aktivnost - generiranje igre

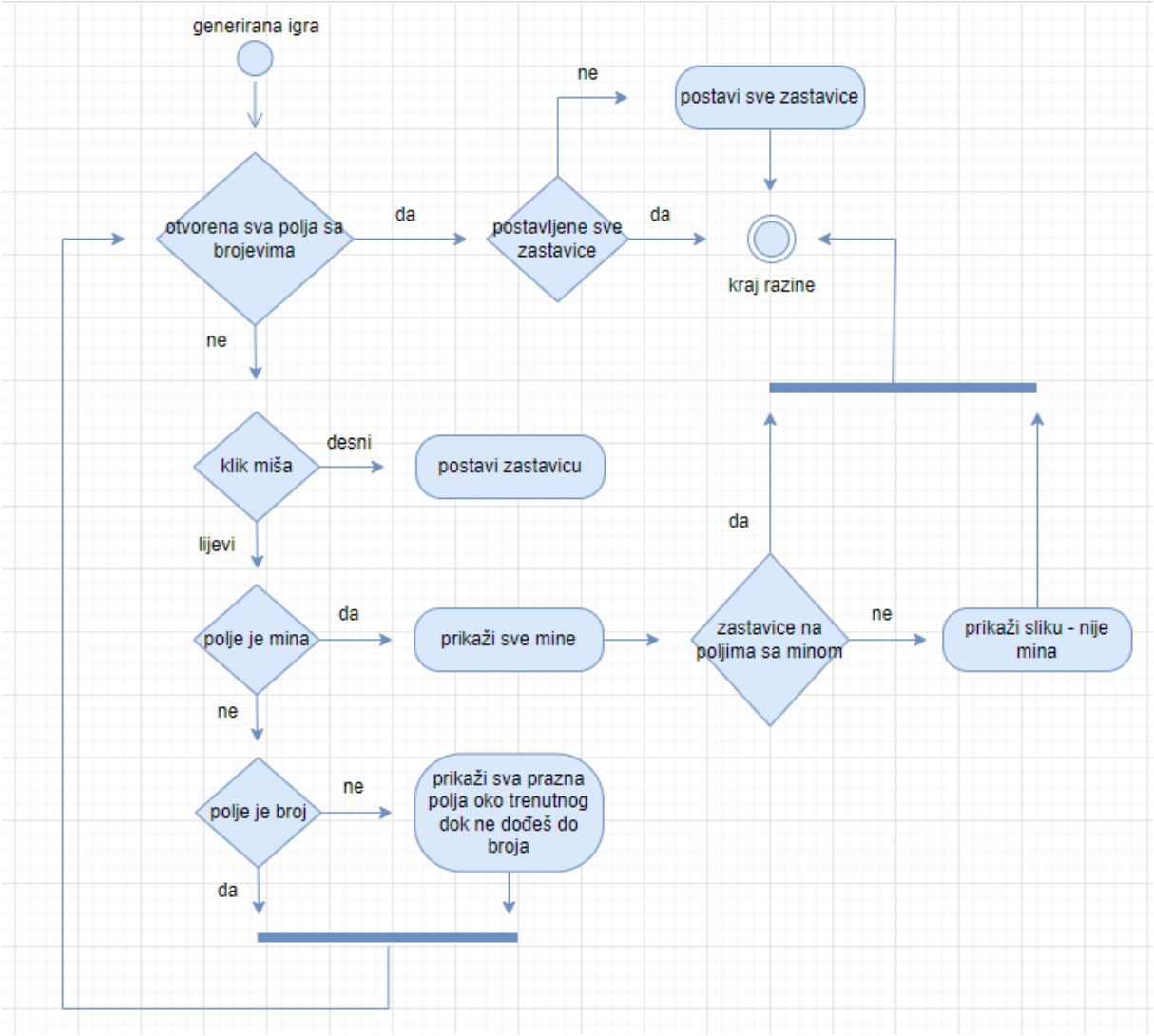
Prije generiranja same igre korisnik odabire između 2 botuna, prvi za prikaz razina i drugi za izgasit aplikaciju. Nakon odabira prvog botuna korisnik sada odabire jednu od šest razina koje imaju već određen broj mina koje će se generirati. Nakon toga nam se prikazuje 15x15 malih sličica koje prikazuju polja, te se generiraju mine „ispod“ tih polja nasumičnim redosljedom. Nakon što se ploča „nacrta“ slijedi korisnikov potez. Sve navedeno možemo vidjeti na Slici 6.

Slika 6 Dijagram generiranja Minolovca

3.3.2. Pod aktivnost – korisnikov potez

Kao i kod svake igre stalno se provjerava jeli ispunjen glavni uvjet i na početku se ne ispunjava te slijedi korisnikov potez. Kod minolovca imamo dvije opcije za potez. Desnim klikom miša korisnik ima opciju postavljati zastavice i također brisati već postavljene. Dok lijevim klikom miša otvaramo polje. U slučaju da smo otkrili minu prikazuju se sva polja sa minama i sa krivim zastavicama ako postoje i to označava kraj razine. Osim polja sa minama imamo opciju i otvoriti polje sa brojem, no ako nismo ni njega kliknuli onda se otvaraju sva prazna polja koja se dodiruju sa trenutnim praznim poljem, odnosno dok ne dođe do polja sa brojem. Ako je uvjet ispunjen i postavili smo sve zastavice točno to znači da nam je razina gotova, u slučaju da nismo igra automatski postavi zastavice na ne otvorena polja i to je kraj razine.

Na Slici 7 možemo proučiti cijeli dijagram aktivnosti korisnikovog poteza.



Slika 7 Dijagram korisnikovog poteza

3.4. Modeliranje logičke strukture

Za lakše razumijevanje logičke strukture prvo ćemo objasniti i slikama prikazati klase koje su se koristile kako bi aplikacija bila funkcionalna. Postoji 6 klasa: Menu, Leveli, Game, Board, Tile i Botun. Pomoću isječaka programskog koda i dijagramom klasa ćemo prikazati svaku klasu te njihove odnose.

3.4.1. Klasa Menu

Kod pygame-a moramo imati uvijek jednu glavnu klasu koja će se izvršavati sve dok je aplikacija upaljena kao što je prikazano na Slici 8. U ovom slučaju to je naša Menu klasa,

odnosno naš prvi okvir koji se prikazuje kada pokrenemo aplikaciju. U njoj nam se nalaze atributi samog okvira zajedno sa dva botuna: Leveli i Exit, koji imaju opciju da ih, lijevim klikom miša, korisnik pritisne.

```
menu = Menu()
while True:
    menu.run()
```

Slika 8 Početna klasa Menu koja je uvijek pokrenuta

Slijedi isječak programskog koda, Slika 9 u kojem se nalazi klasa Menu:

```
class Menu:
    def __init__(self):
        self.screen = pygame.display.set_mode((300, 400))
        pygame.display.set_caption("Minesweeper")
        self.leveli_img = pygame.image.load('slike/button-leveli.png').convert_alpha()
        self.LEVELI = Button(45,100,self.leveli_img,0)

        self.exit_img = pygame.image.load('slike/button-exit.png').convert_alpha()
        self.EXIT = Button(45,210,self.exit_img,0)

    def events(self):
        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                pygame.quit()
                quit(0)

            if event.type == pygame.MOUSEBUTTONDOWN:

                #jeli nam mis na botunu
                mouse = pygame.mouse.get_pos()
                if self.LEVELI.rect.collidepoint(mouse):
                    self.LEVELI = Leveli()
                    self.LEVELI.run()

                if self.EXIT.rect.collidepoint(mouse):
                    pygame.quit()
                    quit(0)

    def draw(self):
        self.screen.fill(WHITE)
        self.LEVELI.draw(self.screen)
        self.EXIT.draw(self.screen)
        pygame.display.flip()

    def run(self):
        self.playing = True
        while self.playing:
            self.events()
            self.draw()
```

Slika 9 Klasa Menu

Na isječku možemo primjetit da nam se klasa sastoji od glavne, inicijalne, funkcije i 3 dodatne: events, draw i run. Unutar glavne funkcije postavili smo glavne atribute okvira, veličinu i naslov, te dvije slike botuna i same botune. Botuni su zasebna klasa koja ima svoje atribute i funkciju draw koja crta botun na okvir pomocu, pygame-ove ugrađene, funkcije blit(). Na slici 10 možemo vidjeti isječak programskog koda za klasu Botun

```
class Button:
    def __init__(self, x, y, image, level):
        self.image = image
        self.rect = self.image.get_rect()
        self.rect.topleft = (x, y)
        self.level = level

    def draw(self, screen):
        screen.blit(self.image, (self.rect.x, self.rect.y))
```

Slika 10 Klasa Botun

Kod funkcije events() koristili smo pygame-ovu ugrađenu klasu event koja ima svoje funkcije za lakše programiranje pokreta. S obzirom da nam je za igrat dovoljan miš event.type nam je `pygame.MOUSEBUTTONDOWN` koji prepoznaje klik miša. S obzirom da on samo prepoznaje klik treba osigurati da samo kad stisnemo botun će se dogodit neka radnja i za to smo koristili također ugrađenu funkciju `pygame.mouse.get_pos()` koja nam daje trenutnu poziciju našeg miša. Međutim trebamo osigurati da program zna kada nam miš dodiruje botun i to smo osigurali sa ugrađenom funkcijom `collidepoint(mouse)` koja nam vraća vrijednost True ako se dodiruju, a u suprotnom False. Naravno imamo i glavni event.type `pygame.QUIT` koji nam zapravo gasi program kada pritisnemo na X od okvira igre pomoću funkcija `pygame.quit()`, koja zaustavlja pozivanje pygame biblioteke i `quit(0)` koja gasi program bez greške.

Kod funkcije draw() nam se nalaze funkcije za crtanje elemenata na zaslon. Pra funkcija nam je koja nam zapravo boja cijeli ekran u boju koja je spremljena pod varijablom `WHITE = (170, 170, 170)`. Nakon nje nam se pozivaju funkcije draw() za 2 botuna, Leveli i Exit, koji crtaju botune, na obojani zaslon. Te zadnja funkcija unutar ove funkcije je pygame-ova funkcija `pygame.display.flip()` koja zapravo ažurira cijeli zaslon prilikom pokretanja funkcije.

Zadnja funkcija unutar ove klase je run() koja zapravo obilježava početak igre, odnosno dok god je varijabla `self.playing = True` program će se pokretat, odnosno trenutna funkcija će pozivati funkcije `self.events() i self.draw()`

3.4.2. Klasa Leveli

Kao i kod prethodne klase imamo inicijalnu funkciju u kojoj nam se nalazi 6 botuna, svaki predstavlja svoju razinu sa različitim brojem mina i botun za nazad, odnosno za ponovni prikaz Menu-a. Sa prvom linijom untar funckije promijenili smo veličinu okvira i naslov, sa Menu na Leveli, kako bi se primjetila promjena zaslona. Slijedi Slika 11 sa prikazom inicijalne funkcije klase Leveli:

```
class Leveli:
    def __init__(self):
        self.screen = pygame.display.set_mode((WIDTH, HEIGHT))
        pygame.display.set_caption("Leveli")

        self.prvi_img = pygame.image.load('slike/button-prvi.png').convert_alpha()
        self.drugi_img = pygame.image.load('slike/button-drugi.png').convert_alpha()
        self.treci_img = pygame.image.load('slike/button-treci.png').convert_alpha()
        self.cetvrti_img = pygame.image.load('slike/button-cetvrti.png').convert_alpha()
        self.peti_img = pygame.image.load('slike/button-peti.png').convert_alpha()
        self.sesti_img = pygame.image.load('slike/button-sesti.png').convert_alpha()


        self.PRVI = Button(30,50,self.prvi_img,10)
        self.DRUGI = Button(250,50,self.drugi_img,15)
        self.TRECI = Button(30,180,self.treci_img,25)
        self.CETVRTI = Button(250,180,self.cetvrti_img,35)
        self.PETI = Button(30,310,self.peti_img,45)
        self.SESTI = Button(250,310,self.sesti_img,55)

        self.nazad_img = pygame.image.load('slike/button-nazad.png').convert_alpha()
        self.NAZAD = Button(0,400,self.nazad_img,0)
```

Slika 11 Inicijalna funkcija klase Leveli

Zatim u funkciji events() nam se nalaze sve radnje, odnosno kada pritisnmo na određeni botun dogodit će se određena radnja. U ovom slučaju svaki botun predstavlja svoju razinu, te

pritiskom na određeni botun poziva se klasa Game() i započinje se igra sa brojem mina koje botun predstavlja. Naravno botun Nazad ne predstavlja razinu već samo radnju za povratak na Menu zaslon.

S obzirom da smo se „prebacili“ na sljedeći okvir potrebno je dodati i mogućnost zaustavljanja aplikacije kao i kod početnog okvira Menu sa događajem `pygame.QUIT`.

Na Slici 12, u nastavku smo prikazali isječak programskog koda sa funkcijom events() unutar klase Leveli.

```
def events(self):
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            pygame.quit()
            quit(0)
        if event.type == pygame.MOUSEBUTTONDOWN:

            #jeli nam mis na botunu
            mouse = pygame.mouse.get_pos()
            if self.PRVI.rect.collidepoint(mouse):
                self.game = Game(self.PRVI.level)
                self.game.new()
                self.game.run()

            if self.DRUGI.rect.collidepoint(mouse):
                self.game = Game(self.DRUGI.level)
                self.game.new()
                self.game.run()

            if self.TRECI.rect.collidepoint(mouse):
                self.game = Game(self.TRECI.level)
                self.game.new()
                self.game.run()

            if self.CETVRTI.rect.collidepoint(mouse):
                self.game = Game(self.CETVRTI.level)
                self.game.new()
                self.game.run()

            if self.PETI.rect.collidepoint(mouse):
                self.game = Game(self.PETI.level)
                self.game.new()
                self.game.run()

            if self.SESTI.rect.collidepoint(mouse):
                self.game = Game(self.SESTI.level)
                self.game.new()
                self.game.run()

            if self.NAZAD.rect.collidepoint(mouse):
                self.menu = Menu()
                self.menu.run()
```

Slika 12 Funkcija events() klase Leveli

Funkcija draw() kao i u prethodnoj klasi služi nam za crtanje elemenata na zaslon. U ovom slučaju su to botuni sa razinama i botun za natrag. Naravno prije botuna opet će se cijeli zaslon obojati kako bi dobili privid da smo zapravo promjenili zaslon, što u pygame-u nije moguće, već se svaki element samo crta jedan preko drugog.

Funkcija run() također kao u prethodnoj klasi ima varijablu `self.playing = True` koja označava da se funkcije events() i draw() pozivaju sve dok je igra upaljena.

Obe funkcije ćemo prikazati Slikom 13:

```
def draw(self):
    self.screen.fill(WHITE)
    self.PRVI.draw(self.screen)
    self.DRUGI.draw(self.screen)
    self.TRECI.draw(self.screen)
    self.CETVRTI.draw(self.screen)
    self.PETI.draw(self.screen)
    self.SESTI.draw(self.screen)
    self.NAZAD.draw(self.screen)
    pygame.display.flip()

def run(self):
    self.playing = True
    while self.playing:
        self.events()
        self.draw()
```

Slika 13 Funkcije draw() i run() klase Leveli

3.4.3. Sprite klasa Tile

Klasa Tile nam služi kako bi mogli napravit svaku kockicu na ploči. U njoj nam se nalaze potrebni atributi kao što su `revealed=False i flagged=False` kako bi znali stanje kockice, naravno u početku su oboje false jer se stanje mijenja tek kada mišom kliknemo na nju.

Slijedi Slika 14 sa prikazom klase Tile i njenim atributima i funkcijama:

```
class Tile:
    def __init__(self, x, y, image, type, revealed=False, flagged=False):
        self.x, self.y = x * TILESIZE, y * TILESIZE
        self.image = image
        self.type = type
        self.revealed = revealed
        self.flagged = flagged

    def draw(self, board_surface):
        if not self.flagged and self.revealed:
            board_surface.blit(self.image, (self.x, self.y))
        elif self.flagged and not self.revealed:
            board_surface.blit(tile_flag, (self.x, self.y))
        elif not self.revealed:
            board_surface.blit(tile_unknown, (self.x, self.y))

    #za vratit u obliku stringa
    def __repr__(self):
        return self.type
```

Slika 14 Klasa Tile

Kao i sa svakom sprite klasom najprije inicijaliziramo poziciju na kojoj se nalazi, te potrebne atribute, u ovom slučaju su to: `self.image, self.type, self.revealed i self.flagged` koje primaju vrijednost prilikom stvaranja novog objekta Tile. S obzirom da za početak igre sve kockice su ne otkrivne odma stavljamo `revealed=False i flagged=False`

Također imamo funkciju draw() koja provjerava stanje kockice te prikazuje određenu sliku kockice. Slike sam postavila u posebne varijable(Vidi sliku) kako bi ih bilo lakše pozivati kod klasa.

Funkcija __repr__() nam služi kako bi tip kockice mogli dobiti u obliku stringa koji ćemo kasnije u klasi Board vidjeti čemu zapravo služi.

Na Slici 15 nam se nalaze bitne varijable sa stvaranje same ploče, odnosno nalaze nam se veličine svake pločice, te u varijblama spremljene sve slike koje su nam potrebne pri stvranju ploče.

```
ROWS = 15
COLS = 15
TILESIZE = 32
WIDTH = TILESIZE * ROWS
HEIGHT = TILESIZE * COLS


tile_numbers = []
for i in range(1, 9):
    tile_numbers.append(pygame.transform.scale(pygame.image.load(os.path.join("slike", f"Tile{i}.png")), (TILESIZE, TILESIZE)))
tile_empty = pygame.transform.scale(pygame.image.load(os.path.join("slike", "TileEmpty.png")), (TILESIZE, TILESIZE))
tile_exploded = pygame.transform.scale(pygame.image.load(os.path.join("slike", "TileExploded.png")), (TILESIZE, TILESIZE))
tile_flag = pygame.transform.scale(pygame.image.load(os.path.join("slike", "TileFlag.png")), (TILESIZE, TILESIZE))
tile_mine = pygame.transform.scale(pygame.image.load(os.path.join("slike", "TileMine.png")), (TILESIZE, TILESIZE))
tile_unknown = pygame.transform.scale(pygame.image.load(os.path.join("slike", "TileUnknown.png")), (TILESIZE, TILESIZE))
tile_not_mine = pygame.transform.scale(pygame.image.load(os.path.join("slike", "TileNotMine.png")), (TILESIZE, TILESIZE))
```

Slika 15 Potrebne varijable za stvaranje igrače ploče

3.4.4. Sprite klasa Board

Klasa Board sadrži listu kockica i zapravo sve potrebne funkcije za igranje igre. U njoj ćemo kasnije vidjeti funkcije koje postavljaju mine, postavljaju brojeve koliko je mina oko određene kockice, te otkrivanje kockica koje nisu blizu mina. Unutar ove klase nalaze se nam se i funkcije koje prikazju kraj igre, odnosno prikazivanje slika kada je krivo postavljena zastavica i općenito kada smo pritisnuli minu.

Za početak Slika 16 prikazuje inicijalizaciju klase i sve njene potrebne atribute, te funkciju za crtanje same ploče

```
class Board:
    def __init__(self, bombe):
        self.board_surface = pygame.Surface((WIDTH, HEIGHT))
        self.board_list = [[Tile(row, col, tile_empty, ".") for row in range(ROWS)] for col in range(COLS)]
        self.bombe = bombe
        self.place_mines()
        self.place_numbers()
        self.pritisnuto = []

    def draw(self, screen):
        for row in self.board_list:
            for tile in row:
                tile.draw(self.board_surface)
        screen.blit(self.board_surface, (0,0))
```

Slika 16 Klasa Board sa inicijalnom i draw funkcijom

Zatim ćemo isječkom, Slika 17, prikazati funkcije koje postavljaju mine i brojeve:

```
def place_mines(self):
    for _ in range(self.bombe):
        while True:
            x = random.randint(0, ROWS-1)
            y = random.randint(0, COLS-1)

            if self.board_list[x][y].type == ".":
                self.board_list[x][y].image = tile_mine
                self.board_list[x][y].type = "X"
                break

@staticmethod
def na_ploci(x, y):
    return 0 <= x < ROWS and 0 <= y < COLS


def check_neighbours(self, x, y):
    mines_counter=0
    for x_okolo in range(-1,2):
        for y_okolo in range(-1,2):
            x_neighbour = x + x_okolo
            y_neighbour = y + y_okolo
            if self.na_ploci(x_neighbour,y_neighbour) != 0 and self.board_list[x_neighbour][y_neighbour].type == "X":
                mines_counter += 1
    return mines_counter

def place_numbers(self):
    for x in range(ROWS):
        for y in range(COLS):
            if self.board_list[x][y].type != "X":
                total_mines = self.check_neighbours(x, y)
                if total_mines > 0:
                    self.board_list[x][y].image = tile_numbers[total_mines-1]
                    self.board_list[x][y].type = "C"
```

Slika 17 Funkcije za postavljanje mina i brojeva

Za postavljanje mina kreirali smo funkciju `place_mine()` u kojoj smo izabrali ugrađenu funkciju random koja nam vraća random broj, u rasponu od broja stupaca i redaka, kako bi na tu poziciju mogli postaviti minu. Kako bi postavili minu trebamo promijeniti .type na Tile kako bi program znao da je to mina, zajedno sa slikom mine.

Kako bi postavili brojeve trebamo prvo provjerit koliko se mina nalazi oko određene kockice. Zbog toga smo napravili funkciju `check_neighbours()` pomoću koje prolazimo sve kockice oko trenutne i ako je kockica mina brojač nam se povećava. Naravno mina nam se može nalaziti i na samom rubu kao i kockica sa brojem, te imamo statičku funkciju `na_ploci()` koja provjerava jeli se trenutna kockica nalazi na rubu, tj jeli postoje susjedne kockice na ploči.

Nakon što smo postavili mine i provjerili susjedne kockice sada možemo postaviti brojeve na kockicu. Koristeći funkciju `place_numbers()` prolazimo kroz cijelu ploču i ispitujemo svaku kockicu koja nije mina postoji li oko nje kockica koja je mina. Ako postoji pozvat će se slika sa brojm mina i .type će se promijeniti u C (Clue).

Sad kad smo napravili sve potrebno za prikaz igre proći ćemo funkciju `prikazi()` koja je zapravo najbitnija kako bi se moglo igrati. Slijedi isječak sa funkcijom, Slika 18:

```
def prikazi(self, x, y):
    self.pritisnuto.append((x, y))
    if self.board_list[x][y].type == "X":
        self.board_list[x][y].revealed = True
        self.board_list[x][y].image = tile_exploded
        return False

    elif self.board_list[x][y].type == "C":
        self.board_list[x][y].revealed = True
        return True

    self.board_list[x][y].revealed = True

    #za automatski pritisnit prazna polja oko kojih nema brojeva
    for row in range(max(0, x-1), min(ROWS-1, x+1)+1):
        for col in range(max(0, y-1), min(COLS-1, y+1)+1):
            if (row,col) not in self.pritisnuto:
                self.prikazi(row,col)
    return True
```

Slika 18 Funkcija za otvaranje polja

Za početak ćemo spremat u listu pritisnute kockice, zatim ćemo provjeravat koja je kockica pritisnuta, te što će se dogoditi nakon što smo je pritisnuli. U slučaju da imamo kockicu koja nije blizu mine zajedno s njom će se otvoriti i ostale prazne kockice sve dok ne naiđe na kockice s brojem, koje će se također otvoriti. Znači glavna uloga ove funkcije je da za svaku pritisnutu kockicu postavlja atribut `.revealed = True` kako bi igra znala da je korisnik otvorio kockicu i kako bi znala koju smo kockicu zapravo otvorili.

Kako bi nam bilo lakše provjeravti točnost brojeva oko mine postavili smo dodatnu funkciju koju možemo vidjeti na Slici 19:

```
#za prikazat u shellu
def display_board(self):
    for row in self.board_list:
        print(row)
```

Slika 19 Funkcija display_board() unutar klase Board

Unutar klase nalazi nam se još jedna funkcija `display_board()` koja mi je zapravo služila za provjeru točnosti igre, odnosno kako bi znala koja kockica je mina, broj ili prazna. U klasi Tile smo vidjeli funkciju __repr__ koja vraća u obliku stringa tip kockice, ona nam je potrebna samo za ovu funkciju kako bi u shellu mogli prikazati ploču. Na Slici 20 se vidi primjer jedne razine u shellu:

```
[., ., ., C, C, C, ., C, X, X, C, ., ., ., .]
[C, C, ., C, X, C, C, C, C, C, C, C, C, C, C]
[X, C, C, C, C, X, X, X, X, C, C, C, X, C, X]
[X, C, X, X, X, C, C, C, C, X, C, C, C, X, C]
[X, X, X, C, X, X, C, C, C, C, X, C, X, X, C]
[C, X, C, C, C, X, X, X, C, C, C, C, C, C, C]
[C, C, C, C, C, C, C, C, C, X, X, C, ., C, C]
[., ., C, X, C, ., C, X, C, C, C, C, ., C, X]
[., ., C, C, C, C, C, X, C, C, ., C, C, C, C]
[., ., ., C, X, C, C, C, X, C, C, C, X, C, C]
[C, C, C, C, C, C, C, X, C, C, X, X, C, C, X]
[X, X, C, X, X, C, C, C, C, C, C, C, C, C, C]
[X, X, C, X, C, C, C, C, C, C, C, C, ., ., .]
[C, X, C, C, X, C, X, C, C, X, X, C, ., ., .]
[C, X, C, C, C, X, C, C, C, C, C, C, ., ., .]
```

```
#za lakse testiranje u shellu cemo nacrtat plocu pomocu ovih znakova
# "." -> unknown
# "X" -> mine
# "C" -> clue
```

Slika 20 Prikaz Minolovca u shellu

3.4.5. Klasa Game

Posljednja i najbitnija klasa nam je klasa Game u kojoj se nalazi sve bitne akcije za igranje igre. Za početak postavljamo broj bombi ovisno o odabranoj razini, te naslov okvira također da piše broj mina. Slijedi isječak klase Game na Slici 21:

```
class Game:
    def __init__(self,level):
        self.level = level
        self.screen = pygame.display.set_mode((WIDTH, HEIGHT))
        pygame.display.set_caption("Broj mina = {}".format(self.level))

    def new(self):
        self.board = Board(self.level)
        self.board.display_board()

    def run(self):
        self.playing = True
        while self.playing:
            self.events()
            self.draw()

        else:
            self.end_screen()

    def provjeri_pobjedu(self):
        for row in self.board.board_list:
            for tile in row:
                #ako postoji tile koji nije mina i nije pritisnuto nije kraj
                if tile.type != "X" and not tile.revealed:
                    return False
        return True
```

Slika 21 Klasa Game

Pomoću funkcije `new()` zapravo kreiramo razinu, tj igru tako da kreiramo Board() sa svim kockicama i svim pravilima navedenim u pod poglavlju Sprite klasa Board, zajedno sa prikazom ploče u shellu.

Funkcija `run()` je do sada uvijek bila ista, odnosno poziva funkcije `events() i draw()` sve dok je igra upaljena. Kada je kraj igre poziva funkciju `end_screen()`

Kako bi igra znala kad je kraj imamo funkciju `provjeri_pobjedu()` koja provjerava jesu li otvorene sve kockice koje nisu mina. U slučaju da nisu vraća False, a ako ne vraća False, onda vraća True.

Najpotrebnija funkcija nam je `events()` koja sadrži cijelu logiku igre, odnosno što će se događati tijekom igre. Slijedi isječak funkcije na Slici 22:

```
def events(self):
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            pygame.quit()
            quit(0)

        if event.type == pygame.MOUSEBUTTONDOWN:
            my, mx = pygame.mouse.get_pos()
            mx = mx // TILESIZE
            my = my // TILESIZE

            #lijevi klik misa
            if event.button == 1:
                if not self.board.board_list[mx][my].flagged:

                    #ako smo pritisnili bombu
                    if not self.board.prikazi(mx,my):
                        for row in self.board.board_list:
                            for tile in row:
                                #ako smo stavili zastavicu na tile koji nije bomba
                                if tile.flagged and tile.type != "X":
                                    tile.flagged = False
                                    tile.revealed = True
                                    tile.image = tile_not_mine

                                #prikazi sve bombe
                                elif tile.type == "X":
                                    tile.revealed = True
                                    pygame.display.set_caption("Game over")
                        #kraj igre
                        self.playing = False

            #desni klik misa
            if event.button == 3:
                if not self.board.board_list[mx][my].revealed:
                    self.board.board_list[mx][my].flagged = not self.board.board_list[mx][my].flagged

    if self.provjeri_pobjedu():
        pygame.display.set_caption("Bravo!")
        self.win = True
        self.playing = False

        #postavi zastavice na sve bombe
        for row in self.board.board_list:
            for tile in row:
                if not tile.revealed:
                    tile.flagged = True
```

Slika 22 Funkija events() klase Game

Prilikom pokretanja klase „prebacujemo“ se na novi okvir te je potrebno imat opciju zaustavljanja igre pomoću X botuna na okviru. To nam je zapravo prvi događaj unutar ove funkcije.

Nakon toga sljedeći događaj nam je klik miša u kojem zapisujemo poziciju miša kako bi mogli kasnije provjeravati koju kockicu pritišćemo. Kod ovog događaja imamo ugrađeni atribut za botun na mišu. Nama su potrebna samo lijevi i desni klik pa ćemo samo njih koristiti.

Za `event.button == 1,` što je zapravo lijevi klik provjeravamo, za početak sve kockice koje nemaju zastavicu, jesmo li pritisnuli na minu. U slučaju da jesmo prikazat će se slika sa minom koju smo pritisnili. Međutim moramo provjeriti sve kockice na koje smo postavili zastavicu, odnosno jesmo li točno postavili zastavice. U slučaju da nismo pokazat će se slika sa prekriženom minom koja predstavlja da se tu ne nalazi mina, u supronom se neće mjenjati slika, odnosno ostat će zastavica. To nam naravno predstavlja kraj igre, te se natpis u okviru minjea u „Game over“ i sljedećim klikom miša vraća nas na okvir sa Levelima,razinama.

Za `event.button == 3`, što je desni klik miša postavljamo zastavicu na kockicu ili ako je već postavljena uklanjamo je.

Kao što smo prikazali sa dijagramom aktivnosti, u pod poglavlju Modeliranje tijeka, igra prije svakog poteza provjerava jesu li ispunjeni svi zahtjevi za pobjedu, odnosno jesu li otvorena sva polja koja nisu mine. U slučaju da su zahtjevi ispunjeni natpis na okviru aplikacije se mijenja u „Bravo“ postavlja atrbute `pygame self.win = True i self.playing = False` i to obilježava kraj igre. U slučaju da nismo postavili sve zastavice igra ih na kraju automatski postavlja.

U funkciji `run()` vidjeli smo da kada da nam se poziva funkcija `end_screen()`. Ona nam zapravo prezentira novi zaslon na kojem se ne nalazi ništa, međutim ima svoje događaje. Standardno ima događaj `pygame.QUIT` koji prekida aplikaciju i događaj `pygame.MOUSEBUTTONDOWN` koji nas prebacuje natrag na zaslon sa Levelima, razinama.

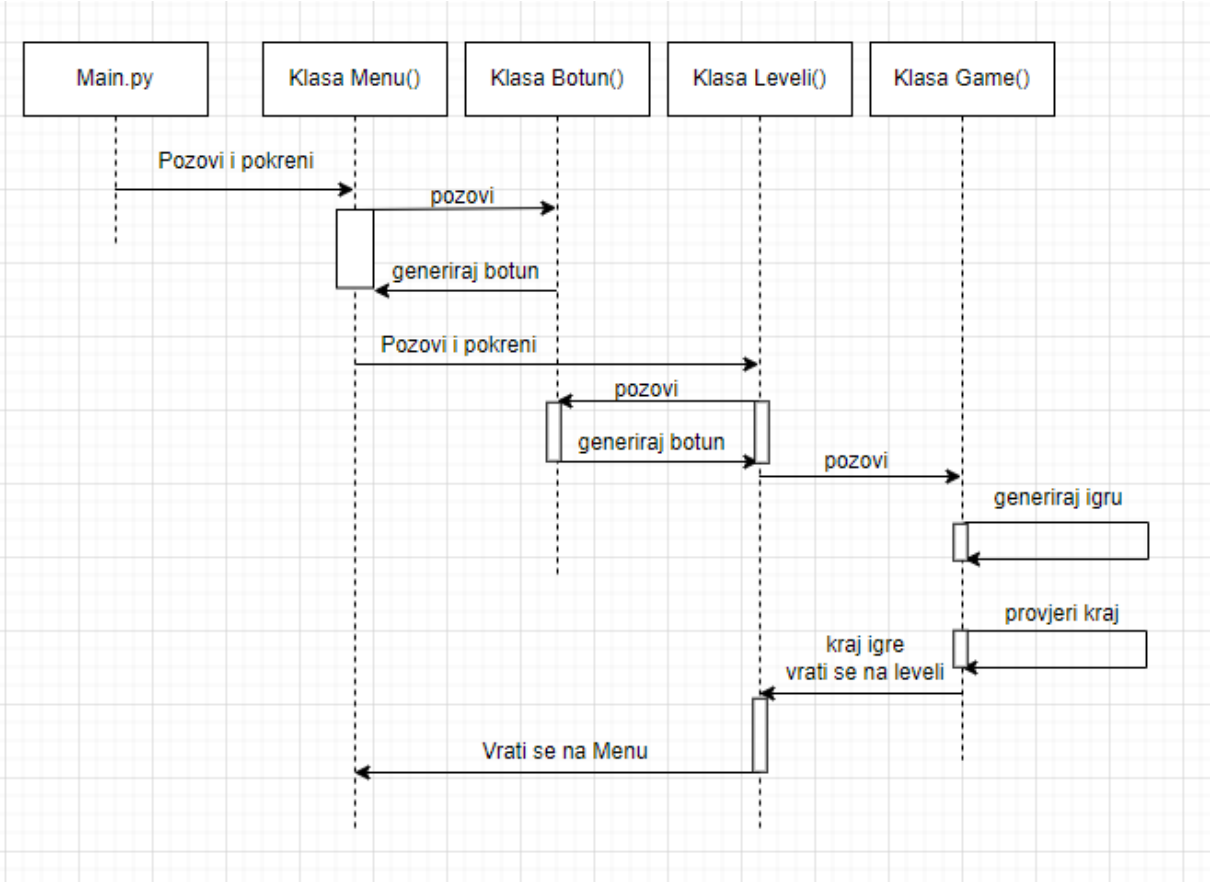
I stanrardno za svaku klasu imamo funkciju draw() koja nam zapravo prikazuje na zaslon cijelu klasu Game. Na sljedećoj slici, Slika 23 prikazane su funkcije end_screen i draw()

```
def end_screen(self):
    while True:
        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                pygame.quit()
                quit(0)

            if event.type == pygame.MOUSEBUTTONDOWN:
                pygame.display.set_caption("Leveli")
                return


def draw(self):
    self.screen.fill(WHITE)
    self.board.draw(self.screen)
    pygame.display.flip()
```

Slika 23 Funkcije end_screen() i draw() unutar klase Game

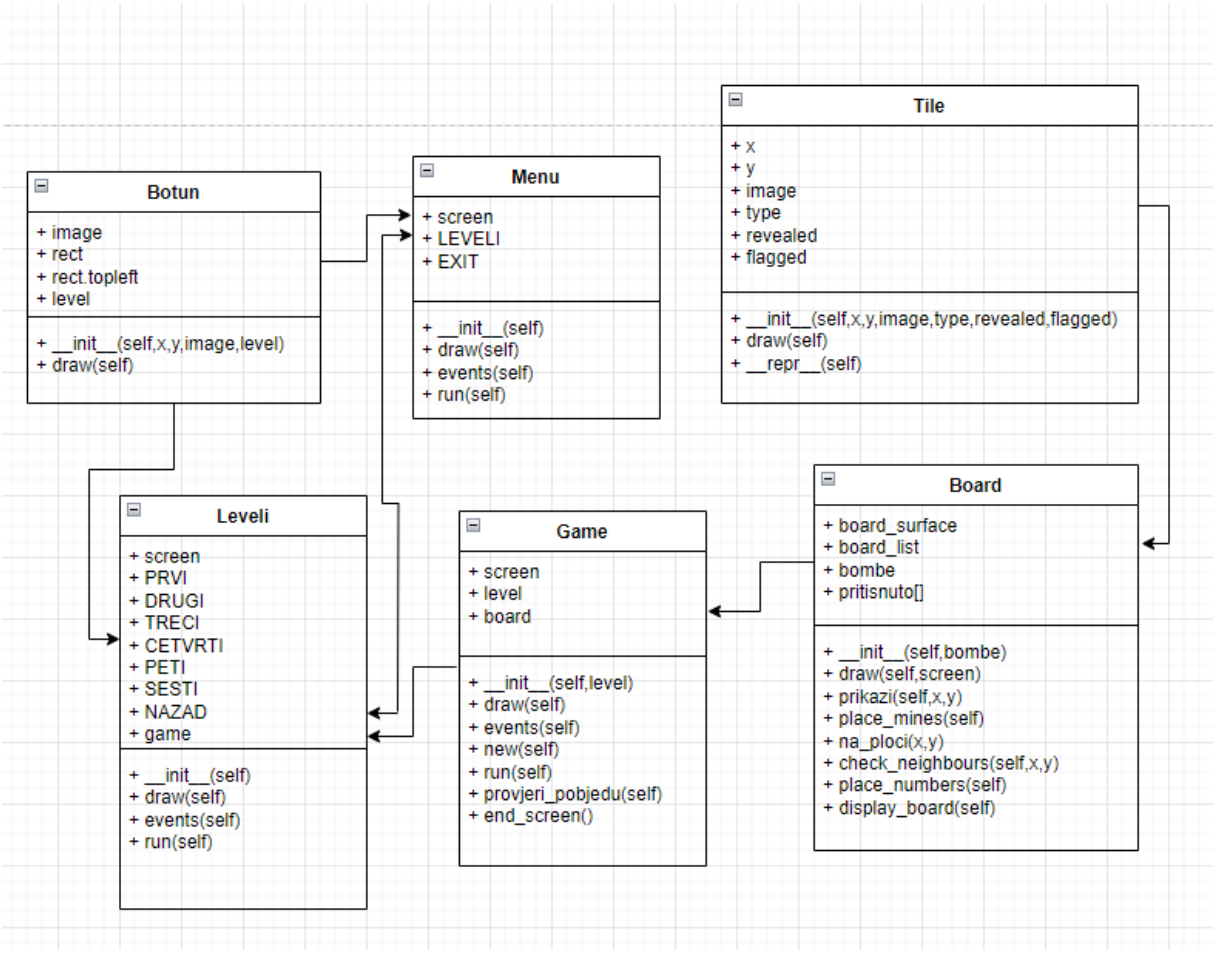
3.5. Dijagram redosljeda i dijagram klasa

Nakon što smo se upoznali sa svim klasama,njihovim atributima i funkcijama sada ćemo pomoću dijagrama redosljeda prikazati njihove odnose. Na sljedećem dijagramu, Slika 24, možemo vidjeti kako unutar Main.py se poziva i pokreće klasa Menu() koja kasnije poziva klasu Botun() kako bi mogla kreirati svoja 2 botuna. Taj botun nam je bitan kako bi mogli pozvati klasu Leveli() koja također poziva klasu Botun() jer trebamo odabrat razinu. Nakon što smo odabrali razinu pokreće se igra pozivajući klasu Game()



Slika 24 Dijagram redosljeda

Osim dijagrama redosljeda još jednostavnije možemo prikazati odnose između klasa sa dijagramom klasa, prikazanim na Slici 25. U sljedećem dijagramu se točno vidi gdje i koju klasu se poziva, te njihove funkcije i atributi. Klasu Botun pozivaju klasa Menu i klasa Leveli jer obe klase imaju botune za interakciju. Možemo vidjeti kako se klase Menu i Leveli međusobno pozivaju jer imaju botune za prijelaze između ta dva okvira. Zatim klasa Level poziva klasu Game koja poziva klasu Board kako bi mogla nacrtati ploču. S obzirom da se ploča sastoji od kockika poziva klasu Tile kako bi ih mogla napravit.



Slika 25 Dijagram klasa

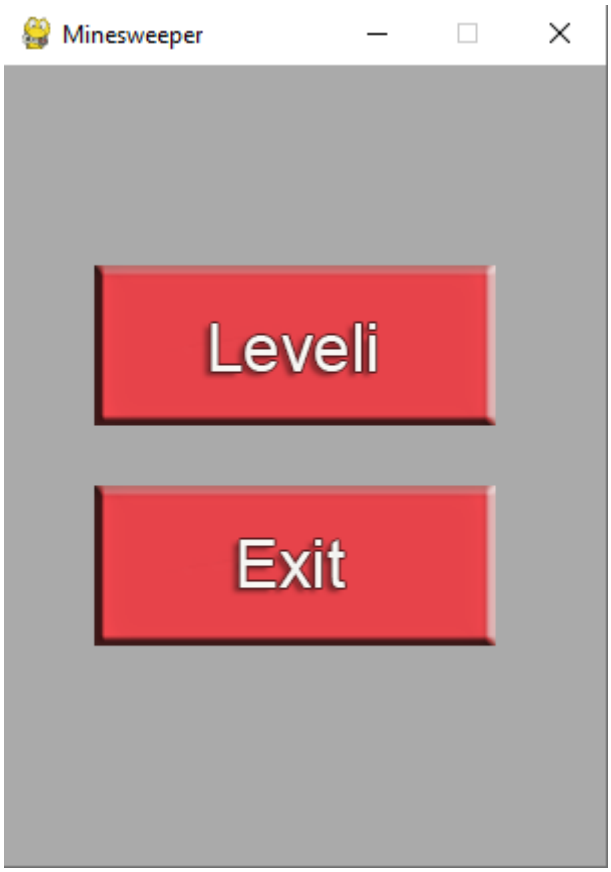
3.6. Modeliranje izgleda

Nakon što smo prikazali logičku strukturu dijagramima i isječcima programskog koda, kroz ovo poglavlje vidjeti ćemo kako sama aplikacija izgleda. Kao što je već rečeno, aplikacija se sastoji od 3 zaslona. Na prva dva zaslona nalaze nam se botuni koju su napravljeni pomoću alata Photoshop, te slike vezane za Minolovac također su pomoću Photoshopa napravljene, odnosno izrezane od isječka igre sa stranice https://minesweeperonline.com/.

Kroz sljedeća pod poglavlja prikazat ćemo svaki zaslon.

3.6.1. Početni zaslon

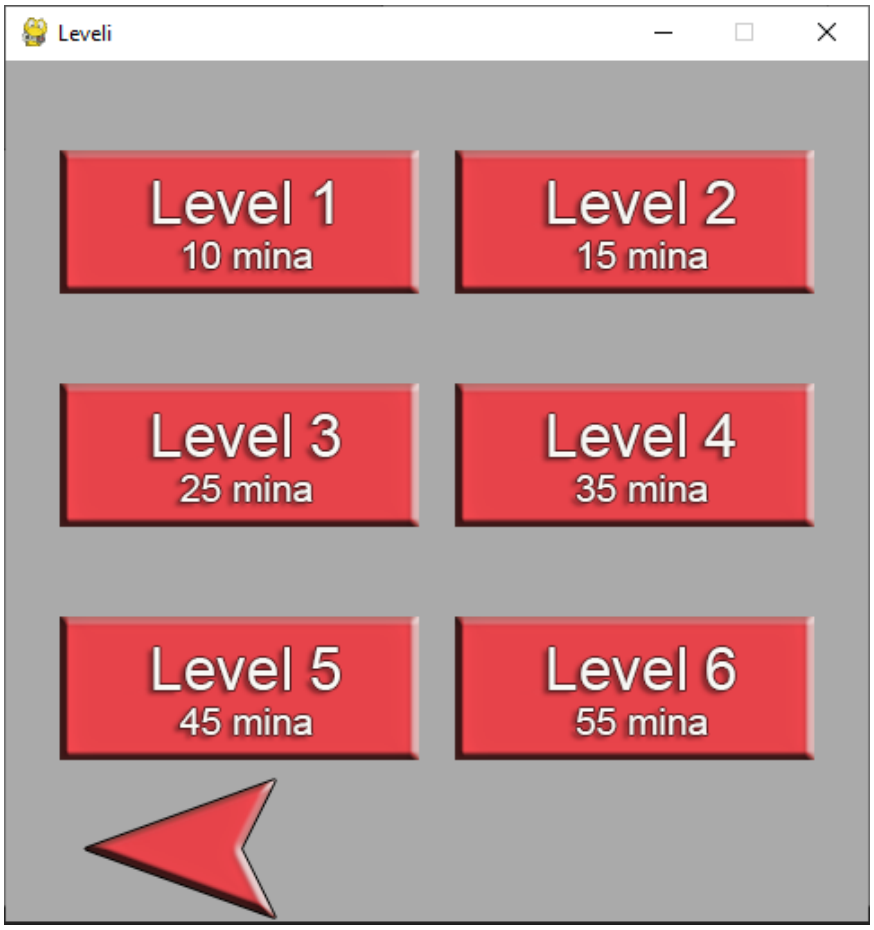
Prvi zaslon je početni na kojem imamo pozadinu i 2 botuna. Prvi botun, Leveli, nas prebacuje na sljedeći zaslon, a drugi botun, Exit, nam gasi aplikaciju. Sami izgled nije ništa posebno, odnosno imamo pozadinu koja je samo siva boja i okvir sa naslovom, imenom igre. Na slici 26 je prikazan početni zaslon nakon pokretanja aplikacije.



Slika 26 Početni zaslon

3.6.2. Zaslon sa razinama

Nakon što smo prrtisnuli botun Leveli prebacujemo se na sljedeći, malo veći, zaslon gdje odabiremo između 6 razina. Osim razina imamo i strjelicu prema lijevo što obilježava povratak na prošli zaslon, odnosno na početni. Na Slici 27 možemo primjetit da nam je naslov okvira promjenjen u „Leveli“, kako bi dobili privid mijenjanja zaslona.

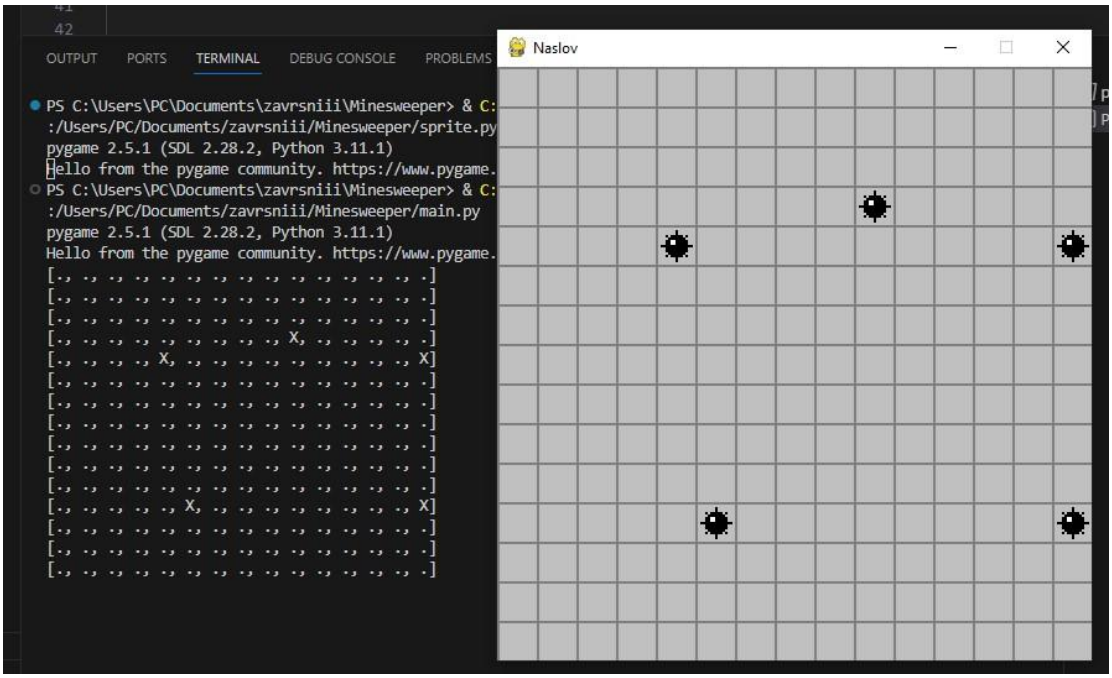


Slika 27 Prikaz zaslona sa razinama

3.6.3. Zaslon sa igrom

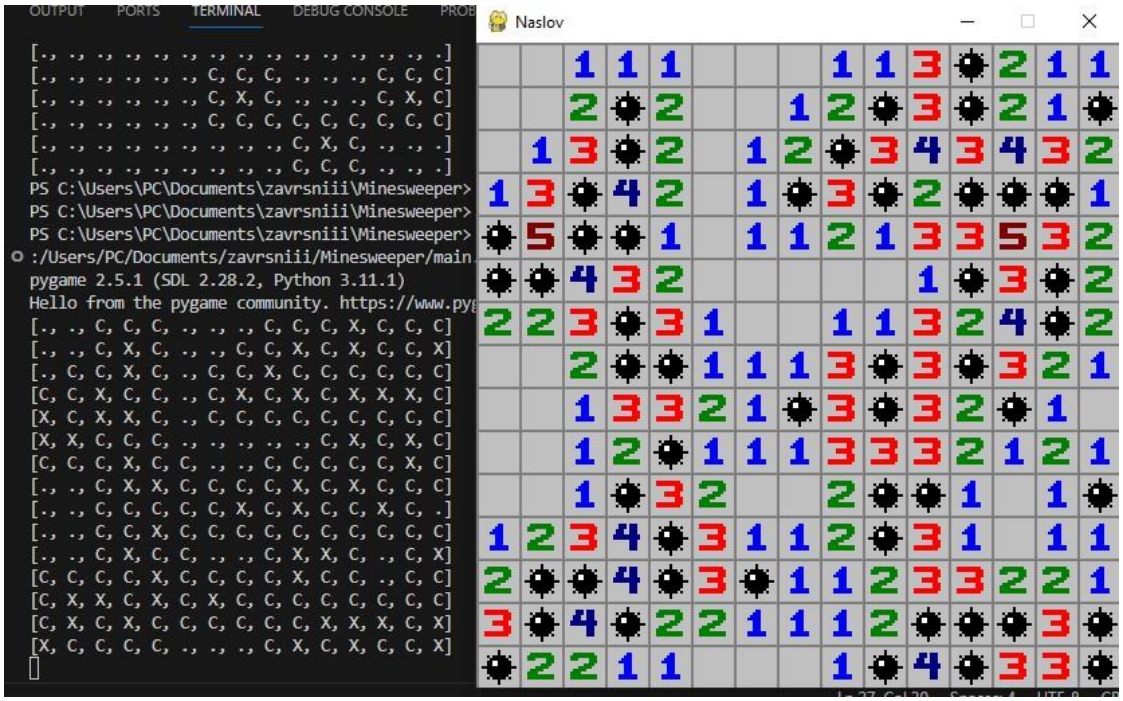
Zadnji, najbitnij, zaslon nam je zapravo zaslon sa igrom. Ovisno koju smo razinu odabrali tako se generira igra. Ukratko ćemo proći postupak modeliranja igre.

Na samom početku postavili smo na svako mjesto sliku praznog otvorenog polja. Nakon toga pozvali smo funkciju za nasumično postavljanje mina, te smo promjenili neka polja u slike mina. Za primjer smo uzeli prvu razinu, te na Slici 28 se vidi igrača ploča sa minama i ista ploča samo u shell-u.



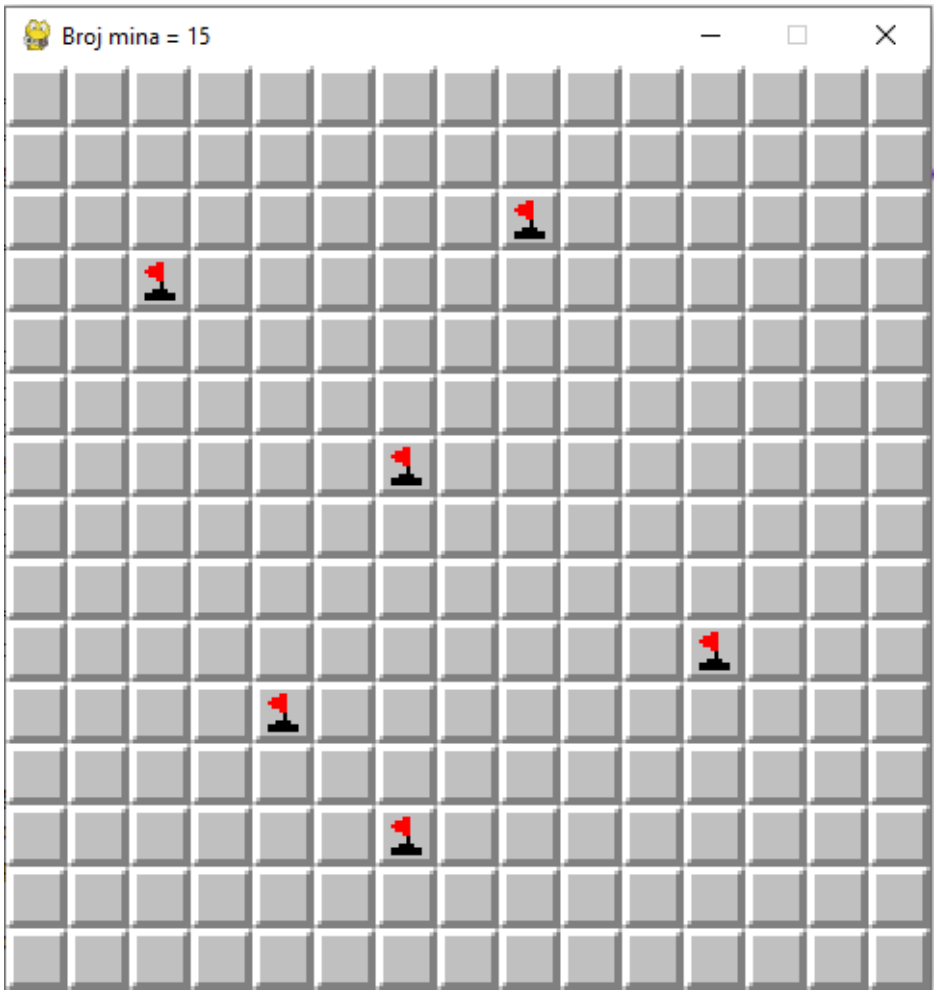
Slika 28 Prikaz samo mina na ploči

Nakon postavljanja mina napisana je funkcija za provjeravanje koliko se mina nalazi oko svakog polja, te su postavljene slike sa brojevima. Na Slici 29 je prikazana igrača ploča sa minama i brojevima oko njih, zajedno sa prikazom u shell-u.



Slika 29 Prikaz mina i brojeva na ploči

Sad kad znamo što se nalazi ispod zatvorenih polja možemo postaviti sloj zatvorenih polja na koje korisnik može postavljati zastavice. Na Slici 30 je prikazana početna ploča sa nekoliko postavljenih zastavica. Kako bi korisnik znao koja je razina, nakon modeliranja na okviru smo postavili koliko ima mina.



Slika 30 Prikaz zatvorenih polja i nekoliko zastavica

Jedino što nam je ostalo za prikazat je kraj igre, odnosno 2 opcije: pobjeda i gubitak.

U slučaju ako izgubimo, mina koju smo pritisnuli dobiva crvenu pozadinu koja simbolizira da je eksplodirala i sve točno postavljene zastavice ostaju na svojim mjestima, a krive se pretvaraju u prekriženu minu. Osim samog prikaza točnih i netočnih polja prikazuju nam se ostale mine i na okviru se natpis mijenja u „Game over“. Slika 31 nam prikazuje gubitak sa 3 točnih i 2 krive zastavice.

Slika 31 Gubitak sa 3 točne i 2 krive szatavice

Zadnje što ćemo kratko prikazati je kada korisnik pobjedi. Korisnik kada je pobjedio prazna mjesta koje su mine automatski se postave zastavice te se natpis u okviru mijenja u „Bravo!“ i to možemo vidjeti na Slici 32.

Slika 32 Prikaz pobjede

4. Zaključak

Python sam za sebe je jako zanimljiv i lagan jezik za razumjeti. Sintaksa mu je puno jednostavnija od ostalih programskih jezika koje smo koristili kao studenti na faksu. Također ima puno biblioteka s kojima se mogu programirati jako zanimljive aplikacije. Koristio se Pygame jer nam je već poznat od prije, no postoje i druge biblioteke za GUI aplikacije. Također jedna od najkorištenijih biblioteka je Random koja nam generira nasumični broj.

Trenutna napravljenja aplikacija, Minolovac, mogla se napraviti sa skroz drugačijim pristupom, te razine su mogle mijenjati veličinu okvira, odnosno mogao je biti veći broj redaka i stupaca. Izgled same aplikacije je moga biti bolji, međutim cilj rada je bio da se vidi logička struktura te da je igra zapravo moguća za igrati bez nekakvih problema ili naglih zaustavljanja.

Poznavanjem pravila za igranje bilo je jako lako smisliti način za provjeravanje pobjede i gubitka. Naravno postoji puno mjesta za poboljšavanje same aplikacije, no za početnike je jako dobra za shvatiti logiku i pravila igre, te sami programski kod je vrlo jednostavan za razumjeti.

5. Literatura

[1] L. Budin, P. Brođanac, Z. Markučić ,S. Perić: “Rješavanje problema programiranjem u Pythonu“, Element, 2013.

[2] Pygame: https://en.wikipedia.org/wiki/Pygame

[3]Minolovac: https://en.wikipedia.org/wiki/Minesweeper_(video_game)

[4] Minolovac pravila igranja: https://minesweepergame.com/strategy/how-to-play-minesweeper.php