Univerzitet u Novom Sadu
Prirodno-matematički fakultet
Departman za matematiku
i informatiku

Matija Kljajić 137/22

„Želite li da postanete milioner?“
Adaptacija TV kviza u video-igru

- seminarski rad iz predmeta „Skript jezici“ -

Novi Sad, 2023.

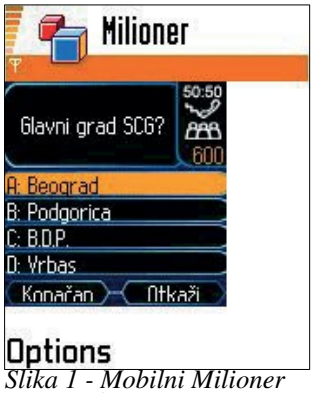
Sadržaj

1 Uvod

1.1 Istorijat

Sve je otpočelo 1998. godine kada je britanska televizijska mreža došla na ideju da napravi kviz „*Želite li da postanete milioner?*“. Čim se videlo da takva vrsta emisije doseže ogromnu gledanost, počela je da se lokalizuje u 106 zemalja među kojima je bila i Srbija. Početkom 2002. godine počinje da se emituje na malim ekranima u Srbiji i ubrzo postaje veoma popularna.

Oko 2002/03. godine nastaje prva srpska adaptacija kviza u video-igru i pojavljuje se u jednom od tadašnjih izdanja poznatog časopisa „*Svet kompjutera*“. Godinu dana kasnije izlazi i mobilna verzija kviza na srpskom jeziku napravljena za „*Symbian*“ operativni sistem i primer njenog UI-a se može videti na slici 1.



Slika 1 - Mobilni Milioner na “Symbian” OS-u

1.2 Cilj rada

1.2.1 Primarna ideja

Konačan cilj jeste rekreiranje i modernizacija ovih starih verzija video-igre kroz *Python* programski jezik.

1.2.2 Struktura kviza i programa

Kao i originalni kviz, igra će korisniku zadati 15 pitanja. Da bi stvari napravili još više zanimljivim, svako pitanje će imati vremensko ograničenje od 30 sekundi. Struktura programa će se sastojati od više *Python* fajlova i *resource* foldera koji će sadržati materijal korišćen u igri.

1.2.3 Pomoći unutar video-igre

Primarno su se u pravom kvizu pojavljivale tri vrste pomoći koje korisniku olakšavaju put do pobede: pomoć publike, pola-pola i pomoć prijatelja. Možemo videti kako je izgledao prikaz tih pomoći u pravom kvizu na slici 2. Pošto ne možemo perfektno simulirati pomoć prijatelja, mi ćemo u našoj verziji kviza nuditi sledeće vrste pomoći: pomoć publike, pola-pola, zamena pitanja i dupliranje vremena.

Slika 2 - Pomoći iz originalnog kviza

1.2.4 Korišćena pitanja

Pitanja korišćena u igri će se nalaziti u jednom *Excel* fajlu unutar već spomenutog *resource* foldera organizovanog tako da je u svakom redu pitanje, tačan odgovor, a zatim ostali netačni odgovori i to tačno po takvom redosledu (Tabela 1). Ovakvim rešenjem se mogu lokalno menjati pitanja i odgovori ukoliko to neko poželi.

Pitanja koja dolaze uz program će biti ona ista pitanja korišćena u starijim verzijama video-igre i zato su kategorisana u više listova unutar *Excel* fajla radi kategorizacije težine. *Excel* fajl će se čitati preko *pandas*[5] biblioteke.

Pitanje	Tačan odgovor	Netačan odgovor	Netačan odgovor	Netačan odgovor

Tabela 1 - Prikaz kako izgleda zaglavlje tabele u korišćenom Excel fajlu

2 Opis programa

2.1 Struktura programa i pregled fajla main.py

Struktura programa je prvobitno bila dosta jednostavnija i sastojala se samo od jednog *.py* fajla. Kasnije je izvršena rekonstrukcija organizacije tako da su metode programa kategorisane uz pomoć više *.py* fajlova radi lakšeg pregleda. Program se sastoji od *main.py* fajla koji se može videti u listingu 1. On sadrži globalne promenljive i poziv metode u kojoj se nalazi sama igra. Ovde takođe i inicijalizujemo *pygame*[1] biblioteku na kojoj se praktično zasniva ceo projekat.

```
# local imports
from game import *

# pygame init & other pygame properties
pygame.init()
width = 1280
height = 720
window = pygame.display.set_mode((width,height))
pygame.display.toggle_fullscreen()
clock = pygame.time.Clock()

# call
game(window, width, height, clock)
```

Listing 1 - main.py

2.2 Pregled fajla game.py

Osim inicijalizacije potrebnih biblioteka, ovde se nalaze četiri metode. Dve su metode iz *pygame*[1] i *cv2*[3] dokumentacije sa nekim blagim izmenama, a preostale dve su metode *endcard* i *game*.

2.2.1 Metoda drawArcCv2

Sa jednom od *cv2*[3] ugrađenih metoda možemo lepše da nacrtamo oblike uz pomoć *AntiAliasing*-a, dok sa *numpy*[6] bibliotekom možemo da izračunamo željeni luk kruga koji nam je potreban za izradu kružnog tajmera. Kao što vidimo na listingu 2, prvo što radimo jeste izračunavanje i kreiranje luka uz pomoć unetih argumenata, a onda ono što je izračunato crtamo na prozor preko metode *blit*.

```
def drawArcCv2(surf, color, center, radius, width, end_angle):
    circle_image = numpy.zeros((radius*2+4, radius*2+4, 4), dtype = numpy.uint8)
    circle_image = cv2.ellipse(circle_image, (radius+2, radius+2),
      (radius-width//2, radius-width//2), 0, 90, end_angle+90,
      (*color, 255), width, lineType=cv2.LINE_AA)
    circle_surface = pygame.image.frombuffer(circle_image.flatten(),
      (radius*2+4, radius*2+4), 'RGBA')
    surf.blit(circle_surface, circle_surface.get_rect(center = center))
```

Listing 2 - Metoda drawArcCv2

2.2.2 Metoda drawText

Ova metoda[2] je gotovo potpuno uzeta iz *pygame*[1] dokumentacije i možete baciti pogled na sekciju za literaturu ako vas zanima detaljnija logika iza ove metode. Ono što radi jeste poveliko formatiranje teksta na osnovu određenih širina i još par argumenata. Konačan rezultat svega toga jeste crtanje nečeg nalik *textbox*-u gde poduži tekst ne može izaći van zadatih granica. Glavna izmena koja je specifična za naš program jeste to da je sam tekst centriran unutar napravljenog *textbox*-a prilikom kreiranja istog.

2.2.3 Metoda endcard

Koristeći modul *mixer* koji je ugrađen unutar *pygame*-a[1] možemo da definišemo neki zvuk, pa da ga pustimo ili pauziramo kada poželimo. Jedna od glavnih funkcionalnosti *mixer*-a koju ćemo koristiti kroz ceo projekat jeste činjenica da možemo da pravimo kanale (*channels*). Najlakše je da zamislimo kanale kao slojeve koji nam dopuštaju da puštamo različite zvučne zapise istovremeno.

Ovo je metoda koja prikazuje kako izgleda prozor prilikom gubitka ili pobede na osnovu broja pitanja do kog je neki korisnik stigao. Možemo videti primer kako takav prozor izgleda na slici 3. Njene argumente čine veliki deo globalnih promenljiva, broj pitanja do kog je neki korisnik stigao, jedan od korišćenih kanala i odgovori koji su poslednje prikazani.

Na početku metode, kanal pomenut u argumentu polako zaustavljamo koristeći metodu *fadeout* i učitavamo potrebne resurse od kojih će se sačinjavati prikazan prozor. Sledeće što radimo jeste provera koji je poslednje učitan odgovor tačan i to čuvamo u nekoj pomoćnoj promenljivoj. To radimo da bi mogli da prikažemo u finalnoj poruci koji je zapravo odgovor bio tačan. Posle toga ispisujemo finalnu poruku koja takođe zavisi od konačnog broja pitanja a zatim crtamo ostale resurse.

Prilikom crtanja resursa, takođe proveravamo da li trenutna pozicija miša prelazi preko pozicija tastera i na osnovu toga menjamo koji resurs specifično crtamo. Trenutnu poziciju miša dobijamo preko funkcije *pygame.mouse.get_pos()*. Na taj način lažiramo efekat animacije tastera kroz *endcard* i *game* metode. Za crtanje vidimo da se koriste metode *blit* koju smo već jednom pomenuli i nakon svega metoda *update*. Prva metoda crta tekst ili oblik negde na prozoru preko unetih koordinata, dok druga metoda te promene/crtanja čuva i prikazuje tokom osvežavanja prozora.

Zatim sledi *event handling* koji nam omogućuje *pygame*[1] biblioteka. On ima sličnu svrhu kao i u metodi *game* i na listingu 3 se može videti kako izgleda u ovoj metodi. Ako bi se ponovo osvrnuli na sliku 3, primetili bi da postoje dva dugmeta, jedno za ponovni pokušaj i jedno za izlaz. Mi aktiviramo tastere tako što proveravamo da li je miš pritisnut i onda sa ugnježdenim *if*-ovima proveravamo da li pozicija kursora jeste na nekom od tih tastera u tom trenutku. Ako jeste, tom prilikom pokrećemo adekvatan deo koda. U ovom slučaju, ako je pritisnuto dugme za ponovni pokušaj onda se izvršava kod sa listinga 4 dok se u suprotnom pokreće kod iz listinga 5. *Event handler* uvek proverava da li je forsirano zatvoren program i u tom slučaju prekida glavnu petlju metode da ne bi došlo do izbacivanja sistemske greške.

```
for event in pygame.event.get():
      if event.type == QUIT:
            run = False
      if event.type == MOUSEBUTTONDOWN:
            if event.button == 1:
                  if (pozicija miša na prvom dugmetu):
                        # adekvatan kod
                  if (pozicija miša na drugom dugmetu):
                        # adekvatan kod
```

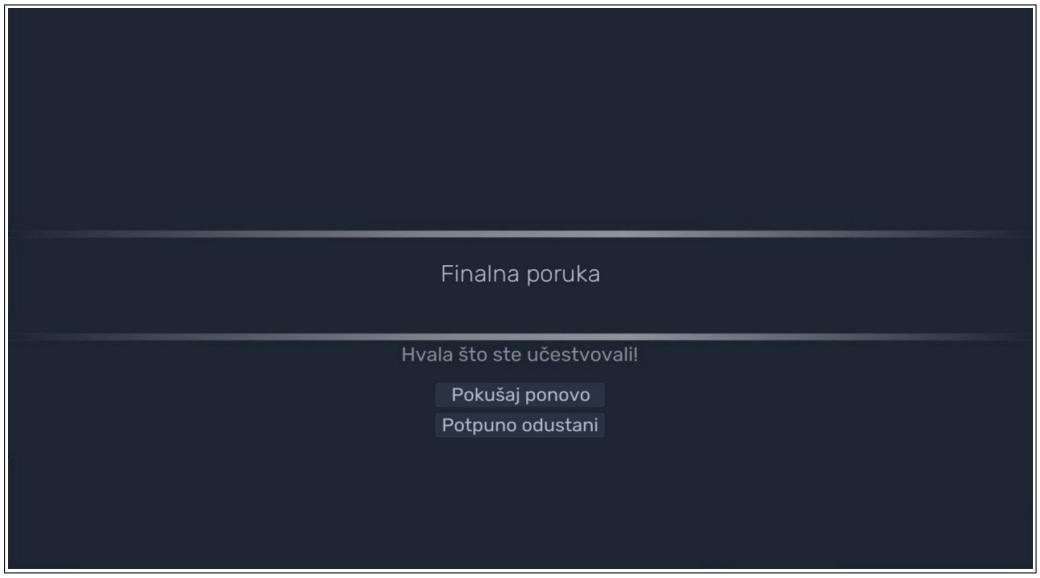
Listing 3 – Event handler u metodi endcard

```
run = False
game(window, width, height, clock)
```

Listing 4 – Kod koji se izvršava nakon pritisnutog tastera za ponovni pokušaj

```
pygame.quit()
exit()
```

Listing 5 - Kod koji se izvršava nakon pritisnutog tastera za izlaz



Slika 3 - Prozor pri završetku partije

2.2.4 Metoda game

Argumenti naše glavne metode jesu prozor na kom iscrtavamo igru (*window*), zadata širina ekrana (*width*), zadata visina ekrana (*height*) i *pygame*[1] ugrađen tajmer (*clock*) koji isključivo služi za *intro*. Nakon učitavanja resursa, definišemo nekoliko promenljivih koje će nam kasnije potrebne. To su:

- *starttime* – trenutno vreme preko ugrađene *pygame*[1] funkcije *get_ticks()*
- *counterclock* i *cc* – pomoćne promenljive koje služe za izračunavanje vrednosti na tajmeru, njihova početna vrednost je 30
- *counterquestion* – redni broj trenutnog pitanja, njegova početna vrednost je 0
- *dcao* – pomoćna promenljiva koja je potrebna da stopira novo izbacivanje pitanja zbog korišćene petlje, njegova početna vrednost je 0
- *shuffle* – pomoćna promenljiva na osnovu koje proveravamo da li su već izmešani prikazani odgovori
- *h* promenljive – pomoćne promenljive na osnovu kojih pamtimo da li je neka vrsta pomoći već iskorišćena
- *graph* – pomoćna promenljiva na osnovu koje znamo da li treba da se prikaže grafik prilikom iskorišćene *pomoći publike*
- *running* – promenljiva na osnovu koje znamo da li se još izvršava metoda

Ovom prilikom takođe učitavamo pitanja iz *Excel* fajla preko *qna* i *qnad* promenljiva. Učitavanje se vrši tako što se u *qna* promenljive učitaju pitanja kategorisana po težini i onda čuvaju u *dataframe* formi preko *pandas*[5] biblioteke. Nakon toga ih konvertujemo u rečnike i njih čuvamo u *qnad* promenljive radi kasnijeg lakšeg pristupa.

2.2.4.1 Pregled koda obeleženog sa # intro

Prvo što vidimo kada pokrenemo igru jeste snimak koji služi kao neka vrsta uvoda. Korišćenjem biblioteke *cv2*[3] možemo uspešno da prikažemo snimak na nekom prozoru tako što ga učitavamo frejm po frejm i onda prikazujemo na osnovu FPS-a u igri što možemo videti u listingu 6. Korišćenjem *tick* metode sinhronizujemo FPS igre sa FPS-om snimka i time omogućujemo tačan prikaz. Prvi frejm je učitan pre petlje, a nakon toga u petlji uz to proveravamo i da li je korisnik pritisnuo miš i time preskočio uvodni video. Ta provera se izvršava preko *event handling*-a. Onda se izvršava crtanje.

Pošto ovakvav način prikaza sa *cv2*[3] bibliotekom ne podržava reprodukovanje audio zapisa, koristimo već-pomenute kanale da bi pustili audio zapis od uvodnog snimka.

```
video = cv2.VideoCapture("resources/game/intro/intro.mp4")

audio = pygame.mixer.Sound("resources/game/intro/intro.ogg")
channel = pygame.mixer.Channel(2)

success, camera_image = video.read()
channel.play(audio)
run = success
while run:
      clock.tick(30)
      # event handling
      for event in pygame.event.get():
            if event.type == QUIT:
                  run = False
            if event.type == MOUSEBUTTONDOWN:
                  if event.button == 1:
                        run = False
                        channel.stop()
      # dalja provera
      success, camera_image = video.read()
      if success:
            camera_surf = pygame.image.frombuffer(camera_image.tobytes(),
                  camera_image.shape[1::-1], "BGR")
      else:
            run = False
      window.blit(camera_surf, (0, 0))
      pygame.display.update()
```

Listing 6 – Sekcija intro

2.2.4.2 Glavni deo metode game

Glavni deo naše metode se izvršava u *while* petlji koja zavisi od promenljive *running*. Pre te petlje, kroz jedan kanal puštamo našu pozadinsku muziku. Prva stvar koja nam je potrebna jeste trenutna pozicija miša i nju skladištimo u promenljivu *posm*. Nakon toga na osnovu vrednosti *counterclock*-a (izračunava se preko koda u listingu 7) definišemo tekst koji će se prikazati unutar tajmera. Ako je *counterclock* manji od nule, to znači da je igrač izgubio svoje vreme i onda se poziva *endcard* metoda.

```
counterclock = cc - (pygame.time.get_ticks()-starttime)/1000
```

Listing 7 – Računanje counterclock-a

Sada prelazimo na crtanje informacija koje već znamo na naš prozor. Prvo crtamo pozadinu, tekst koji smo definisali za tajmer i tabelu dokle je igrač stigao i kolika mu je suma novca zagarantovana. Tu takođe proveravamo da li je *counterquestion* prešao svoju granicu. U slučaju da jeste, korisnik je prošao finalno pitanje i pozivamo metodu *endcard*.

Sledeće što radimo jeste učitavanje odgovora za neko pitanje. Preko već-pomenute *dcao* promenljive postavljamo argument i proveravamo da li je izjednačena sa rednim brojem pitanja. Ako jeste, to znači da za trenutno pitanje nisu učitani odgovori već da su ostali od prošlog ili da je u pitanju prvo pitanje.

Odgovore učitavamo tako što preko indeksa izvlačimo praktično rečnik iz rečnika i od toga formiramo niz odgovora na koje dodajemo pomoćnu vrednost koja nam pokazuje koji je odgovor tačan. Nakon učitavanja odgovora, povećavamo *dcao* promenljivu za 1 da se ne bi ponovo definisali odgovori pre sledećeg pitanja. Učitavanje odgovora moramo da ovako proverimo jer u suprotnom ne bi radilo nasumično prikazivanje odgovora koje sledeće izvršavamo. Pre izvršavanja, oganičavamo i pravljenje nasumičnosti da se odradi samo jednom preko pomoćne promenljive *shuffle*.

Svaku nasumičnost u našem programu postižemo metodom koja se takođe zove *shuffle iz biblioteke random*[7]. Ona zapravo uzima neki niz ili rečnik i samo nasumično raspodeljuje elemente po indeksu.

Sada crtamo tekst koji predstavlja pitanje i odgovore. Kao što smo rekli i u metodi *endcard*, na osnovu *posm*-a biramo koje resurse crtamo radi lažiranja animacije i tako radimo za sve što može da se klikne na prozoru. Specifično za odgovore se uz efekat animacije menja i boja fonta. Međutim ovde uvodimo još jedan nivo provere. Što se tiče tastera sa kojima se bira konačan odgovor, ako su odgovori za neke tastere prazni stringovi onda ih gledamo kao da ne mogu da se kliknu i nije nam potrebna animacija. Isto tako radimo proveru za tastere sa kojima se aktiviraju pomoći preko *h* promenljivih. U ovom delu koda takođe proveravamo da li je ispunjen uslov preko *graph* promenljive za crtanje jednog grafika. Ovo je vezano za kasnije objašnjenu *pomoć publike* u slučaju da je aktivirana.

Sledeće na redu jeste *game*-ov *event handler.* Strukturno je isto kao i u *endcard*-u. Razlika je u tome što se akomodiraju tasteri vezani za prozor koji je specifičan za metodu *game*. Taster za odustajanje na primer samo zatvara igru.

Za tastere koji služe za biranje odgovora se aktivira kod u listingu 8. Postavljaju se *shuffle* i *counterquestion* na nove vrednosti da bi se učitalo novo pitanje. Zatim resetujemo tajmer preko *starttime*-a i *cc*-a. Onda pauziramo pozadinsku muziku preko odgovarajuće metode *pause*. Proveravamo da li je tačan odgovor i ako nije puštamo adekvatan zvučni efekat i prelazimo na metodu *endgame*. U suprotnom će se samo pustiti odgovarajući zvučni efekat za tačan odgovor, ponovo pokrenuti pozadinska muzika i nastaviti dalje.

```
shuffle = True
counterquestion = counterquestion + 1
starttime = pygame.time.get_ticks()
cc = 30
graph = False
channel1.pause()
if (odgovor tačan):
      channel2.play(false1, 0)
      # poziv endcard metode
else:
      channel2.play(true1, 0)
      channel1.unpause()
```

Listing 8 – Kod koji se izvršava kad se klikne neki od tastera za biranje odgovora

Na listingu 9 možemo videti šta se dešava kad se aktivira pomoć publike. Aktiviramo odgovarajući zvučni efekat i menjamo vrednost potrebnih pomoćnih promenljivih. Promenljivu *starttime* konfigurišemo tako da tajmer krene ponovo ali od 15 sekundi. Pozivamo onda metodu *makegraph* koju ćemo kasnije detaljnije objasniti. Ukratko, ona pravi grafik na osnovu odgovora za neko pitanje i onda nasumično prikazuje koji je odgovor moguće tačan.

```
channel3.play(h1ogg, 0)
graph = True
h1 = True
starttime = pygame.time.get_ticks()+1000 - (cc/2)*1000
makegraph(opcije)
```

Listing 9 – Kod koji se izvršava pri aktivaciji pomoći publike

Listing 10 nam prikazuje kod prilikom aktivacije pomoći *pola-pola*. Puštamo adekvatan zvučni efekat i menjamo vrednost pomoćne promenljive. Uzimamo dva nasumična odgovora koja su netačna preko metode *sample*, menjamo ih u prazne stringove i dodeljujemo preko pomoćne promenljive *rn* u originalni niz.

```
channel3.play(h2ogg, 0)
h2 = True
rn = random.sample(opcije,2)
while rn[0][1]=='T' or rn[1][1]=='T':
      rn = random.sample(opcije,2)
for i in range(0, len(opcije)):
      if(rn[0] == opcije[i] or rn[1] == opcije[i]):
            opcije[i][0] = ''
```

Listing 10 – Kod koji se izvršava pri aktivaciji pomoći pola-pola

Na listingu 11 vidimo šta se dešava kad se pritisne taster za *zamenu pitanja*. Puštamo adekvatan zvučni efekat i menjamo vrednost jedne od *h* promenljiva. Promenljive *cc* i *graph* vraćamo na početne vrednosti u slučaju da su aktivne neke izmene zbog aktivacije drugih pomoći. Zatim resetujemo tajmer tako što ponovo definišemo *starttime*. Onda vršimo samo prostu zamenu trenutno prikazanog pitanja u rečniku *qnad* za pitanje koje nije jedno od prvih 15 elemenata preko pomoćne promenljive *pom*. Da bi prikazali novo pitanje, osvežavamo prozor tako što dopuštamo ponovno učitavanje pitanja i odgovora u prozor preko *dcao* i *shuffle* promenljiva i time završavamo ovaj deo koda.

```
channel3.play(h3ogg, 0)
starttime = pygame.time.get_ticks()
cc = 30
h3 = True
graph = False
pom = qnad[counterquestion]
qnad[counterquestion] = qnad[15]
qnad[15] = pom
dcao = counterquestion
shuffle = True
```

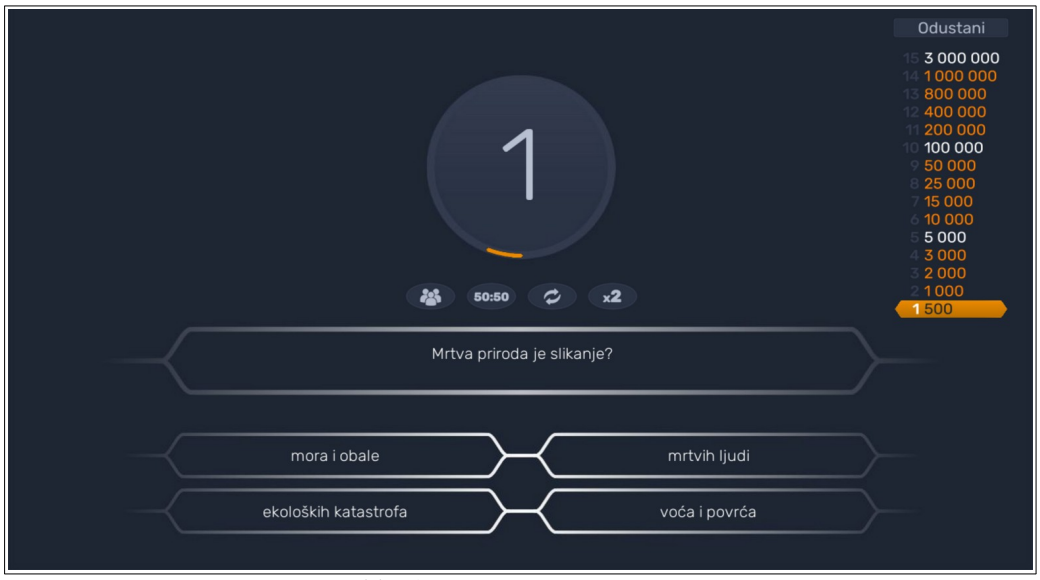
Listing 11 – Kod koji se izvršava pri aktivaciji pomoći zamena pitanja

Poslednju pomoć (*dupliraj vreme*) možemo da vidimo kako funkcioniše na listingu 12. Da bi postigli efekat dupliranja vremena, sve što treba da uradimo jeste da izmenimo promenljivu *cc*. Uz to takođe puštamo adekvatan zvučni efekat.

```
channel3.play(h4ogg, 0)
h4 = True
cc = cc * 2 - (30 - counterclock)
```

Listing 12 – Kod koji se izvršava pri aktivaciji pomoći dupliraj vreme

Na slici 4 možemo da vidimo kako izgleda prozor pri radu *game* metode sad kad smo prošli kroz sav njen kod.



Slika 4 - Prozor za vreme partije

2.2.5 Pregled fajla graph.py i metode makegraph

Fajl *graph.py* je zapravo fajl koji sadrži metodu *makegraph* koju smo rekli da ćemo detaljnije objasniti. Kao što smo rekli, ova metoda gradi grafik i to koristeći biblioteku *matplotlib*[4]. Da bi mogli da koristimo istovremeno ovu biblioteku sa *pygame*-om[1], moramo nakon importovanja *matplotlib*-a[4] pokrenuti sledeći kod u listingu 13. Od *matplotlib*-a[4] takođe importujemo metodu *plotpy* kao *plt* radi lakšeg pisanja koda.

```
matplotlib.use('module://pygame_matplotlib.backend_pygame')
```

Listing 13 – Konfiguracija matplotlib-a

Sad možemo da konačno pređemo na samu metodu i kako to ona funkcioniše. Prvo moramo da generišemo podatke koji će se prikazivati na grafiku. Ono što treba grafik zapravo da prikaže jeste koliko je publike odgovorilo na koje pitanje. Našu publiku će da simulira niz od 100 elemenata i svaki element će zapravo biti indeks odgovora što će označavati ko je kako odgovorio. Kod za ovaj deo je u listingu 14. Dodela vrednosti ko je kako odgovorio se izvršava tako što prvo nasumično izaberemo koliko je ljudi tačno odgovorilo na neko pitanje. Da bi povećali šansu da je najviše ljudi odgovorilo tačno, uzimamo nasumičnu vrednost između 42 i 88 preko metode *randrange* iz *random*[7] biblioteke. Nakon toga nasumično biramo između vrednosti 0 i 12 za neki netačan odgovor. Ostale dve netačne opcije praktično dopunjavamo do 100 na osnovu prethodnih vrednosti i još malo nasumičnosti.

```
t = random.randrange(42, 88)
f1 = random.randrange(0,12)
f2 = (100 - (t + f1)) // random.randrange(1,4)
f3 = 100 - (t + f1 + f2)
```

Listing 14 – Generisanje koliko je ljudi kako odgovorilo

Zatim tražimo indeks pravog tačnog odgovora i ove vrednosti iznad dodeljujemo pravim odgovorima. To možemo da vidimo kroz listing 15.

```
ti = 0
for i in range(0, len(opcije)):
      if opcije[i][1] == 'T':
            ti = i
if ti == 0:
      data = {'A':t, 'B':f1, 'C':f2, 'D':f3}
elif ti == 1:
      data = {'A':f1, 'B':t, 'C':f2, 'D':f3}
elif ti == 2:
      data = {'A':f1, 'B':f2, 'C':t, 'D':f3}
else: data = {'A':f1, 'B':f2, 'C':f3, 'D':t}
```

Listing 15 – Dodela generisanih odgovora

Ovim smo završili sa podacima i sad moramo da konstruišemo grafik. Prvo definišemo labele i vrednosti za grafik preko *data* promenljive. Onda definišemo osobine grafika i to se radi preko koda u listingu 16.

```
# definisanje grafika
fig = plt.figure(figsize=(4,4),facecolor='#1e2633')
ax = fig.add_subplot(111)
# definisanje osobina grafika
plt.bar(labels, values, color ='#ea8a00')
ax.tick_params(axis='x', colors='#ea8a00')
ax.tick_params(axis='y', colors='#ea8a00')
ax.spines[['left','right','top','bottom']].set_color('#ea8a00')
ax.set_facecolor('#1e2633')
# čuvanje grafika u png formatu
plt.savefig('resources/figure.png')
```

Listing 16 – Definisanje osobina grafika i njegovo čuvanje u formatu slike

Kao što vidimo, na samom kraju čuvamo grafik kao sliku koju kao što smo pomenuli u glavnom delu metode *game* crtamo u slučaju da *graph* promenljiva ima tačnu vrednost.

3 Zaključak

Inicijalni cilj smo ostvarili i program može da se distribuira uz još par promena radi bezbednosti i naravno poštujući autorska prava oficijalnog kviza. Ovaj kod naravno uvek može da se malo unapredi da bude efikasniji i kompaktniji. Ono što takođe možemo naknadno da dodamo da bi igru napravili pristupačnijom jeste neka vrsta tabele sačinjene od najboljih 10 igrača. To možda može da se uradi preko statičkog sajta i *python* skripte koja menja *html* fajl, ali međutim onda već zalazimo u sasvim drugi projekat koji se može odraditi u budućnosti da bi ovde poboljšali generalno iskustvo igranja.

4 Literatura

[1] Dokumentacija za biblioteku *pygame*, https://www.pygame.org/

[2] Dokumentacija za metodu *TextWrap*, https://www.pygame.org/wiki/TextWrap

[3] Dokumentacija za biblioteku *cv2*, https://docs.opencv.org/4.x/

[4] Dokumentacija za biblioteku *matplotlib*, https://matplotlib.org/stable/index.html

[5] Dokumentacija za biblioteku *pandas*, https://pandas.pydata.org/docs/

[6] Dokumentacija za biblioteku *numpy*, https://numpy.org/doc/

[7] Dokumentacija za biblioteku *random*, https://docs.python.org/3/library/random.html