SVEUČILIŠTE U ZAGREBU
FAKULTET ORGANIZACIJE I INFORMATIKE
VARAŽDIN

Barica

NASLOV RADA – LATEX PREDLOŽAK

SEMINAR

TEORIJA BAZA PODATAKA

Varaždin, 2022.

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Barica

Matični broj: 35918/07–R

Studij: Informacijski i poslovni sustavi

NASLOV RADA – LATEX PREDLOŽAK

SEMINAR

Mentor:

dr. sc. Bogdan Okreša Đurić

Varaždin, rujan 2022.

Barica

Izjava o izvornosti

Izjavljujem da je ovaj seminar izvorni rezultat mojeg rada te da se u izradi istoga nisam koristio drugim izvorima osim onima koji su u njemu navedeni. Za izradu rada su korištene etički prikladne i prihvatljive metode i tehnike rada.

Autorica potvrdila prihvaćanjem odredbi u sustavu FOI Radovi

Sažetak

Opsega od 100 do 300 riječi. Sažetak upućuje na temu rada, ukratko se iznosi čime se rad bavi, teorijsko-metodološka polazišta, glavne teze i smjer rada te zaključci.

Ključne riječi: riječ; riječ; ...riječ; Obuhvaća 7 ± 2 ključna pojma koji su glavni predmet rasprave u radu.

Sadržaj

1. Uvod

Tema ovog rada je izrada online igrice pomoću objektno-orijentirane baze podataka (OOBP). Kako je sve veća potreba za izgradnjom sustava sa što većom sličnošću baze podataka i aplikacijske domene dovelo je do objektno-orijentiranog pristupa izgradnje baze podataka. OOBP je baza podataka u koju pohranjujemo podatke u obliku objekata za razliku od klasičnih baza podataka gdje je podatak pohranjen u obliku tablice (relacije). Objektno-orijentirane aplikacije manipuliraju objektima te pri ponovnom pokretanju aplikacije dolazi do brisanja podataka objekata, stoga moramo imati nekakav mehanizam upravljana spremanja podataka. Za izgradnju online igrice uzeta je poznata igra Križić kružić te pomoću ZODB (Zope Object Data Base), koji omogućava pohranu objekata u programskom jeziku Python, kreirana aplikacija Križić kružić koja se može igrati u više igrača preko mreže te koja spremanja podataka za kasniju upotrebu. .

2. Korišteni alati

ZODB pohranjuje Python proširenjem (engl. Extend) objekte koristeći ugrađene Persistent klase. ZODB baza podataka ima jedan korijenski objekt tipa rječnik, koji je jedini objekt kojem baza podataka izravno pristupa. Svi ostali objekti pohranjeni u bazi podataka dostupni su preko korijenskog objekta. Objekti na koje upućuje objekt pohranjen u bazi podataka također se automatski pohranjuju u bazu podataka. Budući da je ZODB objektna baza podataka nema zasebnog jezika za operacije baze podataka, vrlo mali utjecaj na vaš kod kako bi objekti bili postojani, nema mapera baze podataka koji djelomično skriva bazu podataka. Korištenje objektno-relacijskog preslikavanja nije kao korištenje objektne baze podataka. Gotovo da nema spojeva između koda i baze podataka. Odnosima između objekata rukuje se vrlo prirodno, podržavajući složene objektne grafikone bez spojeva (engl. joins). [1]

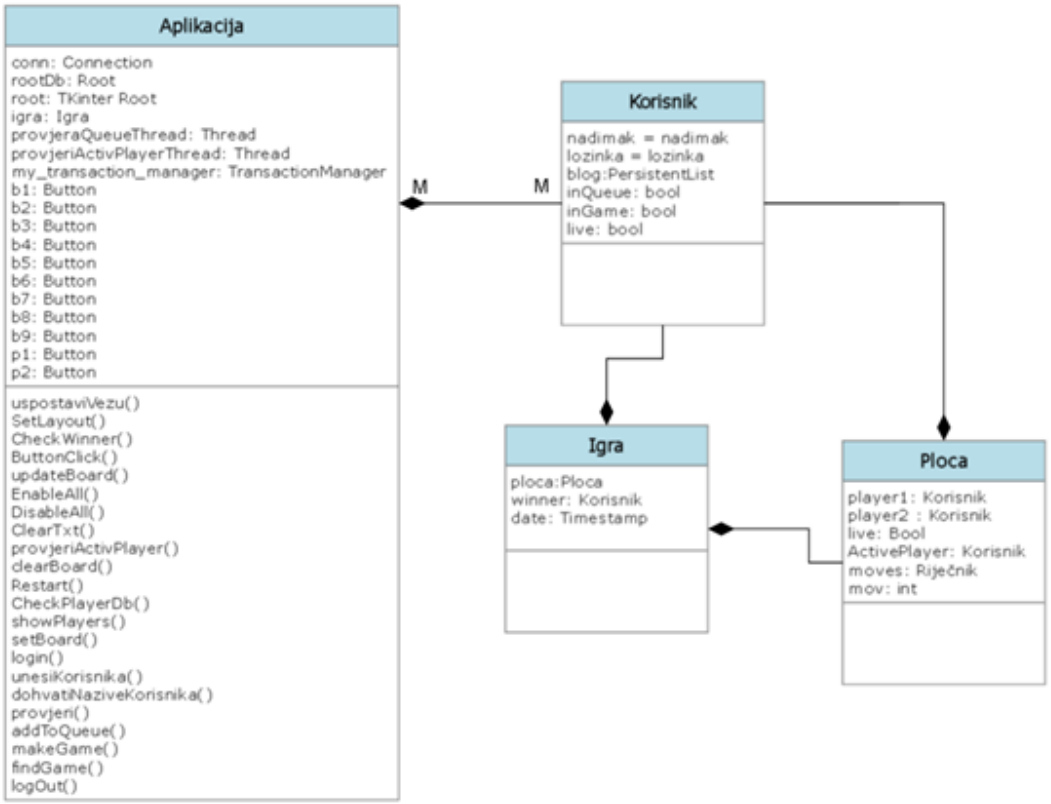
Zope Enterprise Objects (ZEO) implementacija je ZODB pohrane koja omogućuje višestrukim klijentskim procesima da pristupe spremljenim objektima na jednom ZEO poslužitelju. To omogućuje transparentno skaliranje. Kada koristite ZEO, pohrana niže razine, obično pohrana datoteka, otvara se u procesu ZEO poslužitelja. Klijentski programi povezuju se s ovim procesom koristeći ZEO ClientStorage. ZEO pruža dosljedan prikaz baze podataka svim klijentima. ZEO klijent i poslužitelj komuniciraju korištenjem prilagođenog protokola slojevitog na vrhu TCP-a.[3]

ZODB preglednik vam omogućuje pregled postojanih objekata pohranjenih u ZODB-u, pregled njihovih atributa i povijesnih promjena koje su na njima napravljene. [4]

3. Korišteni alati

Dijagram klasa prikazan na slici [1] prikazuje model baze podataka koji se sastoji od objekata Korisnik, Igra i Ploca koji su povezani agregacijskom vezom. Klasa Aplikacija je glavna klasa igrice Križić kružić te se on instancira na svakom pokretanju aplikacije. Sadrži varijable i funkcije za pokretanje, igranje i komunikaciju s bazom podataka.

Kreirana baza podataka u sebi sadrži korijenski objekt root tipa rječnik u kojega se spremaju podaci. U tablici [1] je prikazana hijerarhijska struktura root objekta gdje vidimo ostale objekte sadržane u root objektu, ključ objekta i tip.



Slika 1: Klas dijagram **Vlastita izrada**

root	Ključ	Tip		
	korisnici	Rječnik		
			Nadimak	Korisnik
	queue	Lista		
	live	Rječnik		
			Nadimak	Korisnik
	games	Rječnik		
			Datum	Ploča

Slika 2: Klas dijagram **Vlastita izrada**

4. Primjer korištenjai

U sljedećem poglavlju opisano je pokretanje servera sa bazom podataka i pokretanje izvođenje aplikacije. Također za potrebe prikaza sadržaja baze van aplikacije korištena je ZODB browser plugin.

4.1. Pokretanje servera

Kao bi naša baza podataka bila dostupna prvo ju pokrećemo iz terminala naredbom sa slike [2] i naredbom sa slike [3] pokrećemo zodb browser preglednik za ZODB.

4.2. Pokretanje aplikacij i prijava

Aplikaciju pokrećemo kroz naredbenu konzolu kao python skriptu. Nakon pokretanja aplikacije prikazati će se iskočni prozor, slika [4], s mogućnosti unosa nadimka igrača, u slučaju da igrač s nadimkom ne postoji kreira se novi igrač i igra može započet.

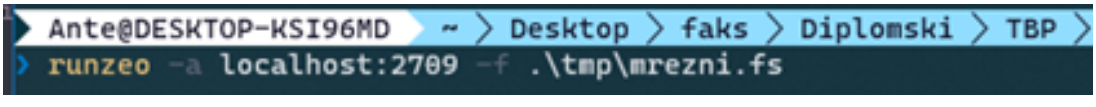
4.3. Igranje

Kada se korisnik spoji na aplikaciju unosom nadimka prikazuje se prozor s aplikacijom i gumbima koje može pritiskati nakon što se pronađe protivnik. Igrač može samostalno biti logiran ili ubaciti sebe u red čekanja za pronalazak protivnika. Na slici [5] vidimo dvije instance aplikacije pokrenute i svaka pokazuje listu aktivnih igrača s desne strane koja se ažurira svakih par minuta.

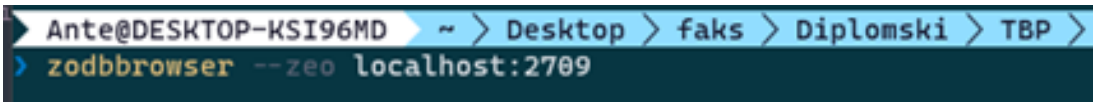
Pritiskom na gumb Play igrača se dodaje u red čekanja za pronalazak protivničkog igrače te će se gumbi tj. polja za poteze odblokirati kada se pronađe protivnički igrač prikazano na slici [5]. Slika [6] prikazuje dva igrača (a i b) koja su uparena iz reda čekanja te je pokrenuta nova igra. Na potezu je igrač „a“ i kao prvi igrač ima oznaka „x“

. Na pritisak korisnik na prazno polje unosi se oznaka sa simbolom korisnika koji je bio na redu te se žuta boja igrača mijenja na protivničkog igraća. Svaku sekundu u pozadini se ispituje dali je došlo do zapisa poteza u bazi podataka te se potez sa slike [7] igraća „a“ prikazuje na protivničkoj ploče i polja se blokiraju za daljnji unos dok protivnički „b“ igrač ne napravi potez..

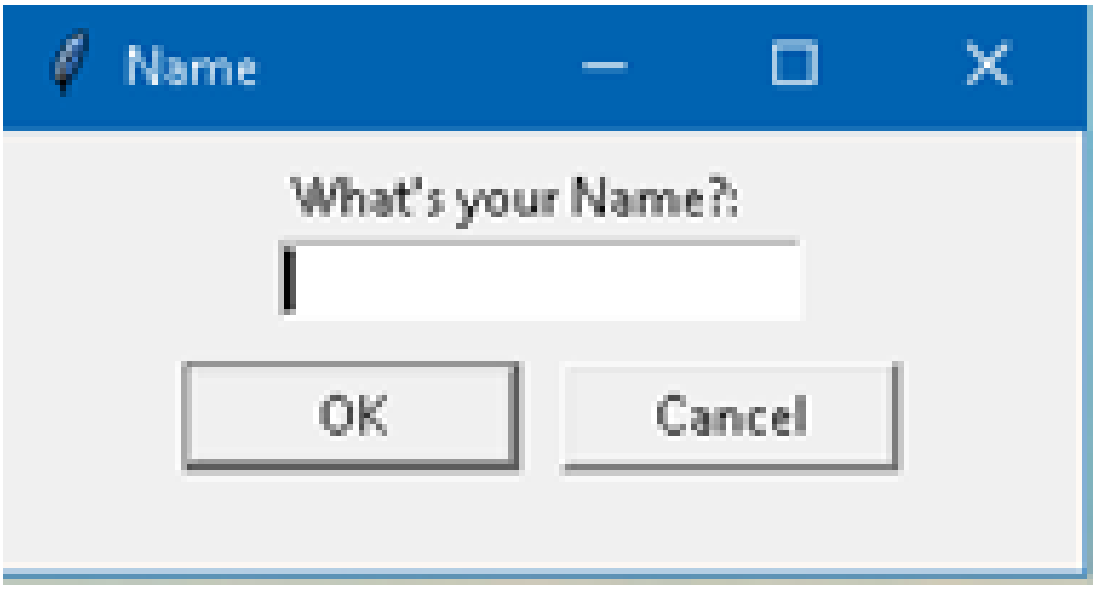
Na slici [8] je prikazana završena runda između igraća „a“ i „b“, nakon što je jedan od njih u ovom slučaju b igrač postavio tri uzastopna simbola „X“ osvježio se zadnji prikaz svih poteza



Slika 3: Runzeo naredba **Vlastita izrada**



Slika 4: ZODB browser naredba **Vlastita izrada**



Slika 5: Login prozor **Vlastita izrada**

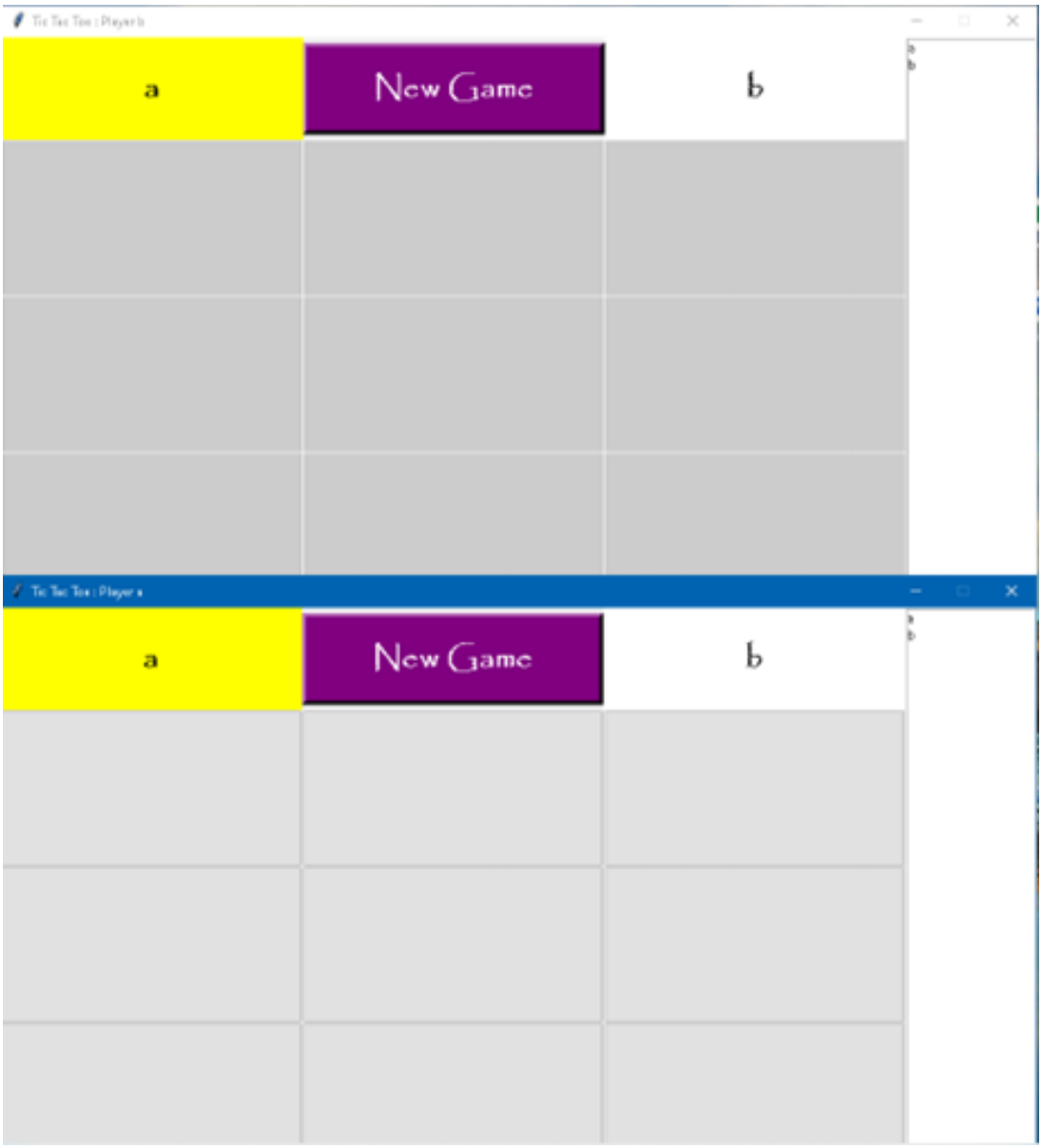
igra se zaustavila i prikazao se iskočni prozor s porukom o pobjedi igraća „b“ za obje pokrenute aplikacije. Pritiskom na gumb „Ok“ aplikacija obriše poteze igraća te se igraći ponovno mogu dodijeliti u red čekanja na protivnika.

4.4. Implementacija

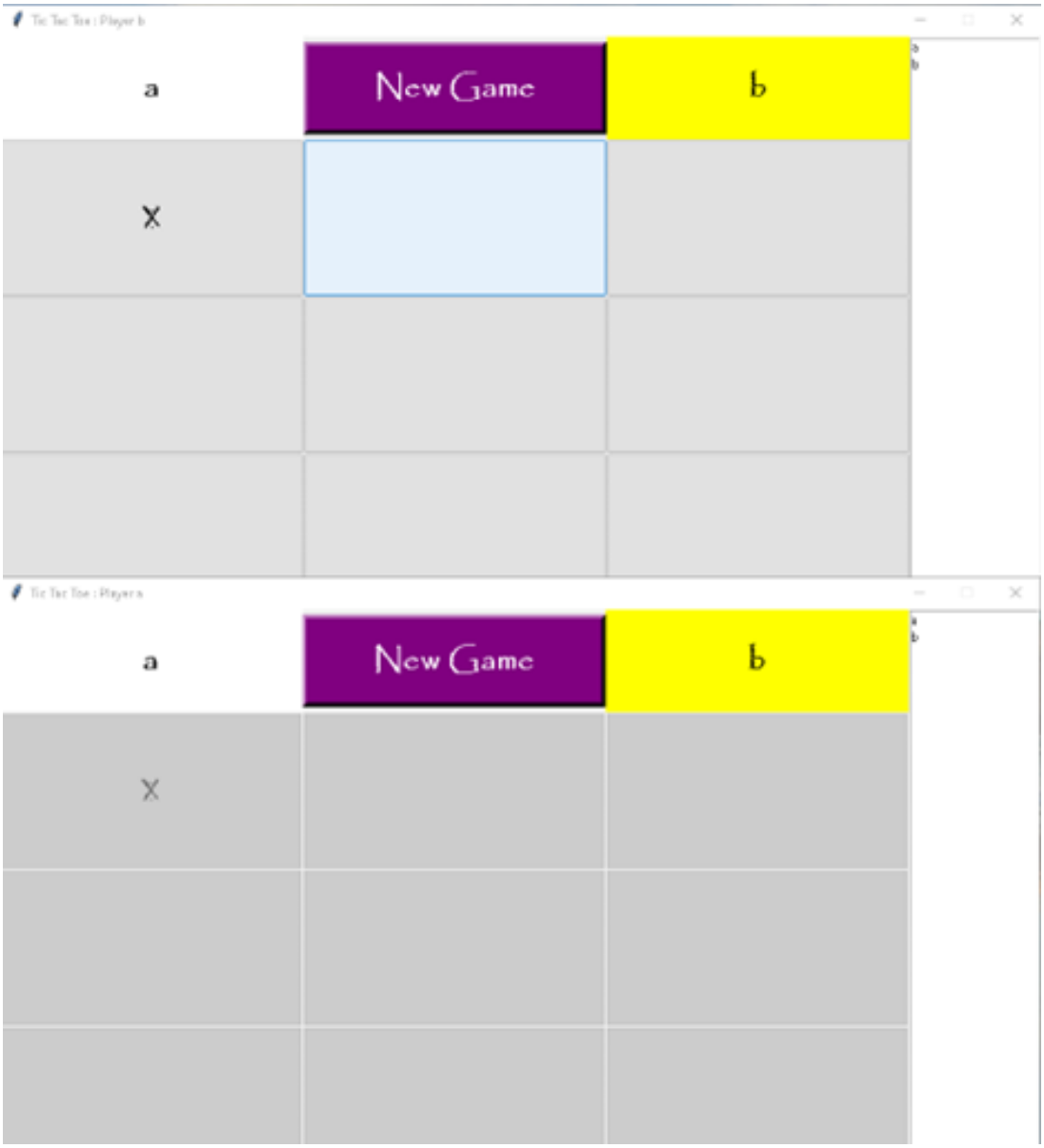
U sljedećih par poglavlja biti će opisane pojedine implementacije funkcionalnosti aplikacije.

4.5. Model

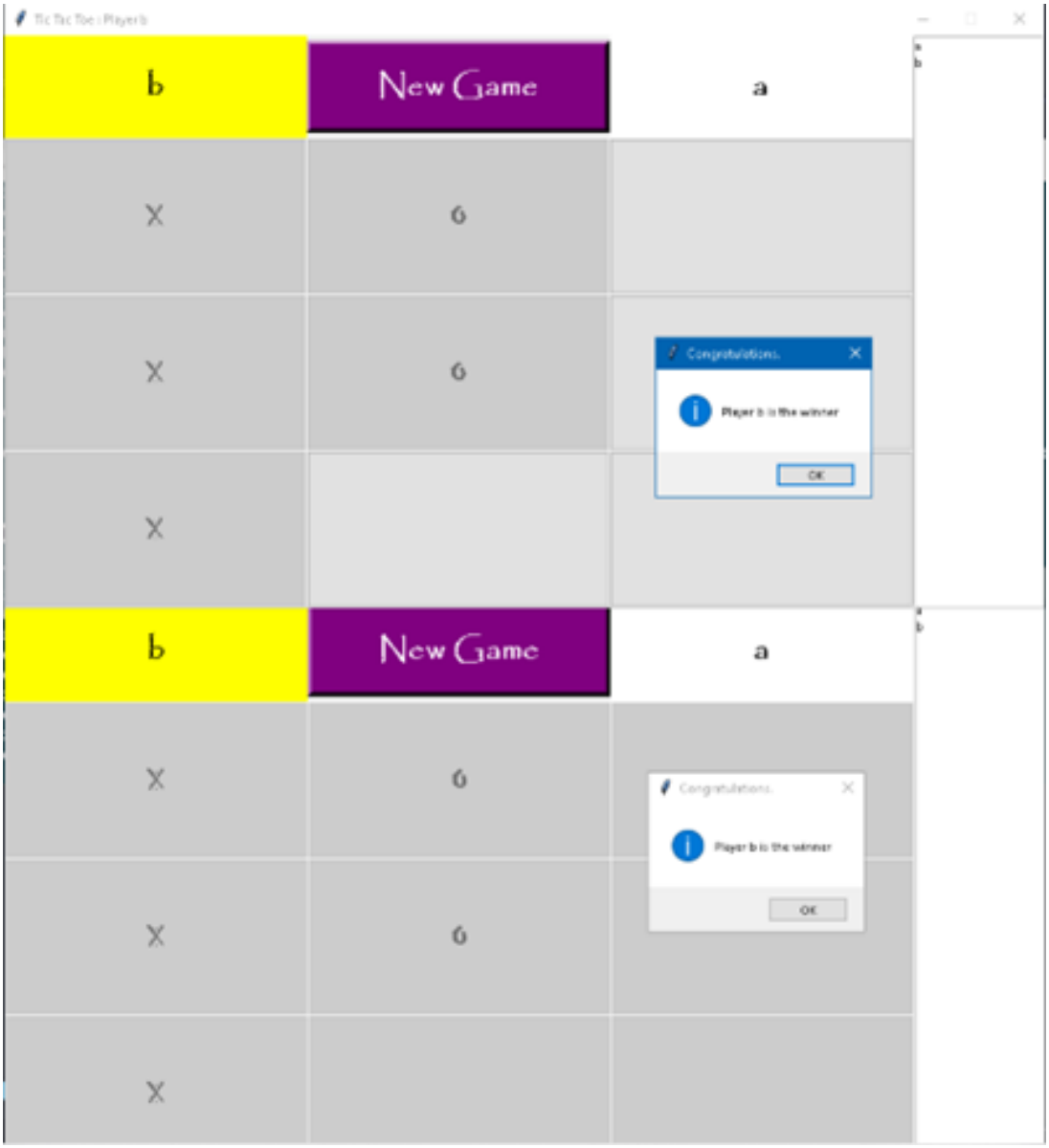
Klase sa slike [9] prikazuju objekte Igru, Plocu, Korisnika i aplikaciju koji se koriste u aplikaciji. Aplikacija je glavna klasa aplikacije koja upravlja objektima i podacima za igranje igre, self.provjeraQueueThread i self.provjeraActivePlayerThread su dvije niti koje se vrte u pozadini aplikacije sve dok se aplikacije ne zaustavi. Klase Igra, Ploca i korisnik se koriste u aplikaciji za prijeno podatak ali i ZODB pomoću njih kreira pogleda (view) na bazu podataka koje vidi aplikacija prilikom komunikacije s bazom.



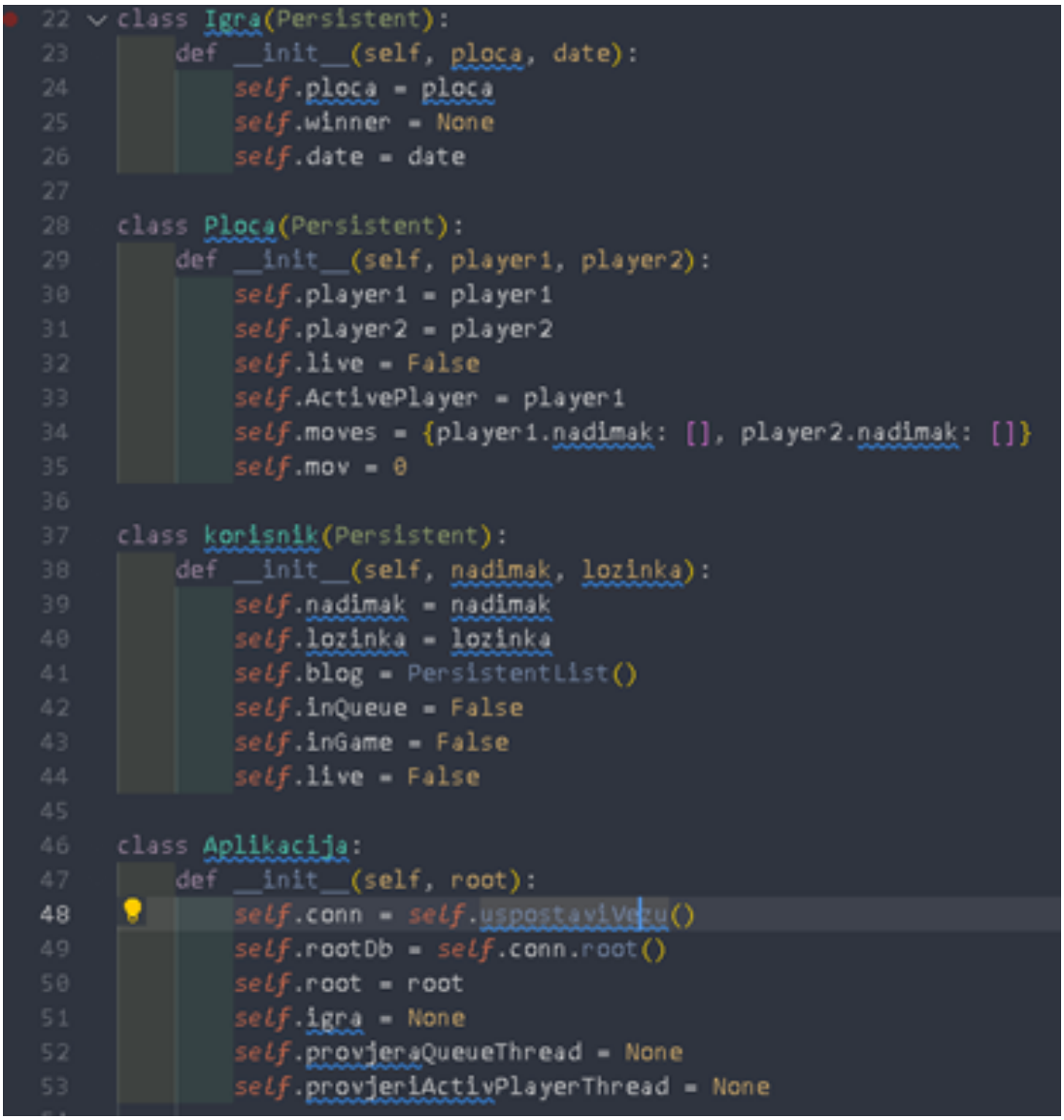
Slika 6: Započeta igra s dva igrača **Vlastita izrada**



Slika 7: Prvi potezi igrača **Vlastita izrada**

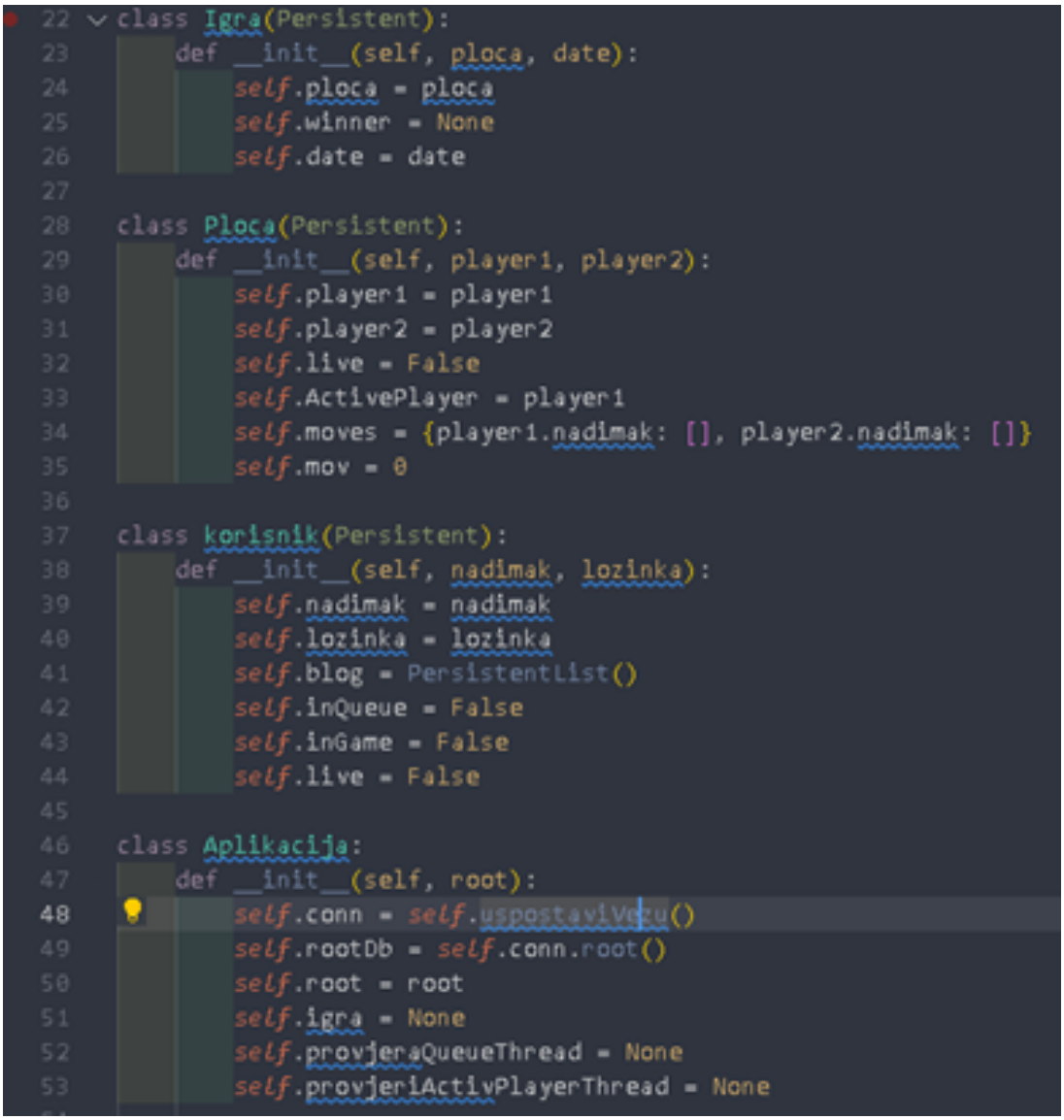


Slika 8: Kraj igre **Vlastita izrada**



```
22  class Igra(Persistent):
23      def __init__(self, ploca, date):
24          self.ploca = ploca
25          self.winner = None
26          self.date = date
27
28  class Ploca(Persistent):
29      def __init__(self, player1, player2):
30          self.player1 = player1
31          self.player2 = player2
32          self.live = False
33          self.ActivePlayer = player1
34          self.moves = {player1.nadimak: [], player2.nadimak: []}
35          self.mov = 0
36
37  class korisnik(Persistent):
38      def __init__(self, nadimak, lozinka):
39          self.nadimak = nadimak
40          self.lozinka = lozinka
41          self.blog = PersistentList()
42          self.inQueue = False
43          self.inGame = False
44          self.live = False
45
46  class Aplikacija:
47      def __init__(self, root):
48          self.conn = self.uspostaviVezu()
49          self.rootDb = self.conn.root()
50          self.root = root
51          self.igra = None
52          self.provjeraQueueThread = None
53          self.provjeriActivPlayerThread = None
```

Slika 9: Klase aplikacije **Vlastita izrada**



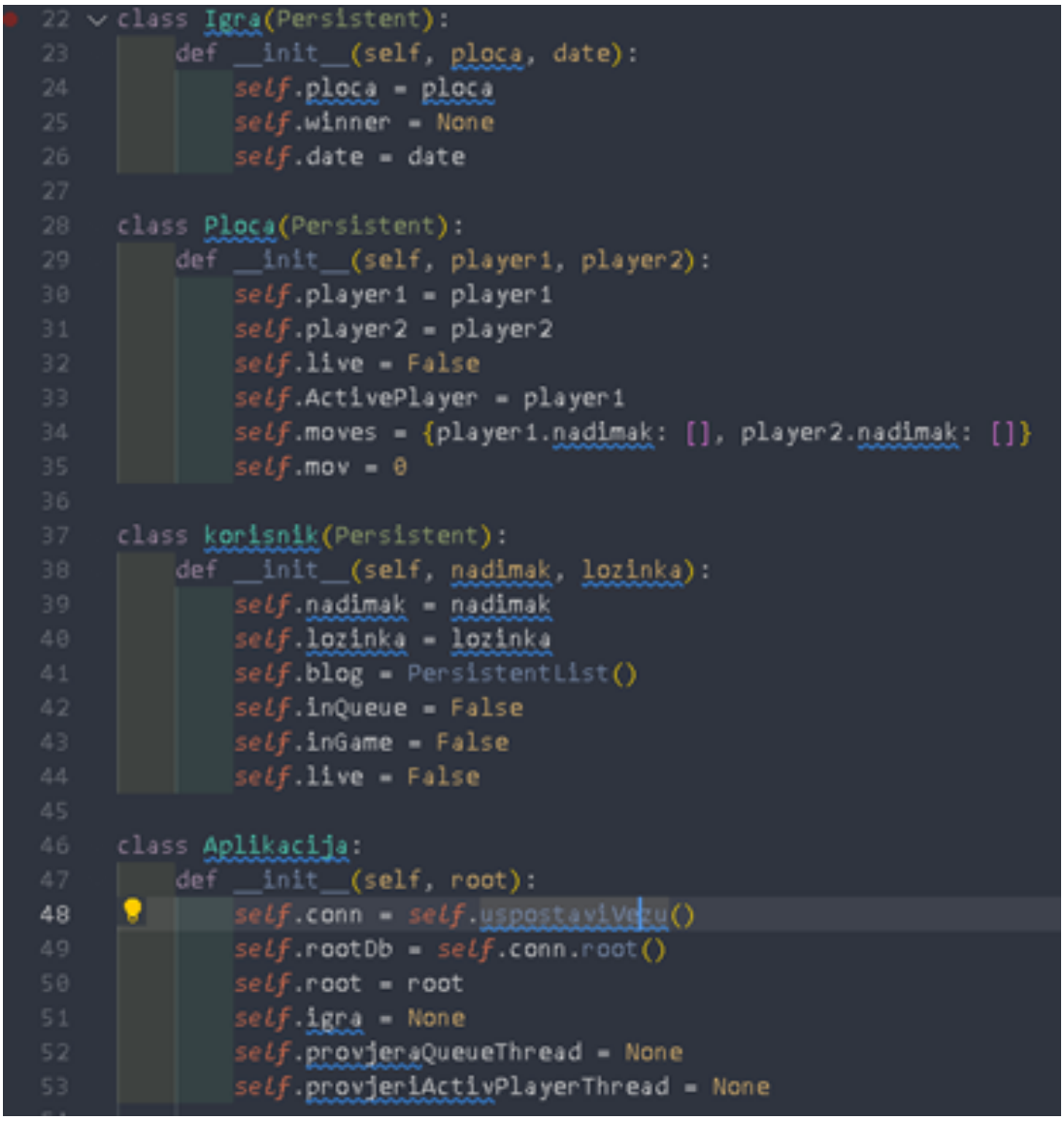
Slika 10: Metoda uspostaviVezu **Vlastita izrada**

4.6. Konekcija na server

Kako se u aplikaciji koristi više niti na kreiranju konekcije s bazom moramo imati tzv. Transactionmanagera koji se brine o našim transakcijama. Kreiramo ClienStorage na prije pokrenutom serveru [10] te instancu baze te otvaramo konekciju prema bazi s TransactionMaangerom i to je naša pristupna točka.

4.7. Traženje protivnika – queue

Pritiskom na gumb Play postavlja sebe na red čekanja za odabir protivnika. Slika [11] prikazuje dio koda kod koje ga se korisnik unosi u red čekanja tzv. queue i postavljaju se zastavice inQueue te se kreira transakcija i pohranjuje u bazu podataka s metodom commit(), ako je transakcija uspjela započinje izvođenje nove niti provjeraQueueThread s metodom



Slika 11: Metoda addToQueue **Vlastita izrada**

self.provjeraQueue

Metode provjeraQueue i dohvatuQueueKorisnika se izvode u novokreiranoj niti, slike [12], odgovorne su za dohvaćanje novog korisnika iz reda čekanja sve dok se takav korisnik ne postavi u red čekanja. provjeraQueue poziva metodu dohvatiQueueKorisnika koja joj vraća pronađenog korisnika u slučaju da je našla korisnika i trenutni korisnik nije više u queu (zastavica kor.inQueue) kreira se nova igra inače provjerava se dali je trenutni korisnik u igri (zastavica kor.inGame) te se pretražuje baza podataka s igrama, metoda findGame().

Metoda dohvatiQueueKorisnika, slika [13], dohvaća prvog korisnika u queue te ako je taj korisnik različit od trenutnog briše ga iz queue i briše trenutnog korisnika iz queue, postavlja atribute inQueu za oba korisnike na True, zapisuje promjene u bazu i pronađenog korisnika vraća kao povratnu vrijednost.

```
    def provjeraQueue(self):
        print( 'Dohvat iz queue thread' )
        while True:
            if self.kor.inQueue:
                print("player seraching")
                self.conn.sync()
                self.rootDb = self.conn.root()
                self.kor = self.rootDb[ 'korisnici' ][self.kor.nadimak]

                nextPlayer = self.dohvatiQueueKorisnika()
                self.nextPlayer = nextPlayer

                if self.nextPlayer != [] and self.kor.inQueue == False:
                    self.bs.config(text="New Game")
                    self.makeGame(self.kor, self.nextPlayer)
                elif self.kor.inGame == True:
                    self.bs.config(text="New Game")
                    self.findGame()

            time.sleep( 1 )
```

Slika 12: Metoda provjeraQueue **Vlastita izrada**

```
def dohvatiQueueKorisnika(self):
    print( 'Dohvat iz queue upitti' )
    try:
        if len(self.rootDb[ 'queue' ]) == 0:
            return []

        nextPlayer = self.rootDb[ 'queue' ][0]
        if nextPlayer.nadimak == self.kor.nadimak or nextPlayer.inGame == True or self.kor.inGame == True:
            return []

        del self.rootDb[ 'queue' ][0]
        """delete sebe"""
        for i in range(len(self.rootDb[ 'queue' ])):
            if self.rootDb[ 'queue' ][i].nadimak == self.kor.nadimak:
                del self.rootDb[ 'queue' ][i]
                break

        nextPlayer = self.rootDb[ 'korisnici' ][nextPlayer.nadimak]
        nextPlayer.inQueue = False
        nextPlayer._p_changed = True

        self.kor.inQueue = False
        self.kor._p_changed = True

        self.my_transaction_manager.commit()
        return nextPlayer
    except ConflictError or ValueError:
        print( 'trx abort' )
        time.sleep( 1 )
        pass
```

Slika 13: Metoda dohvatiQueueKorisnika **Vlastita izrada**

```
538     def makeGame(self, player1, player2):
539         print("make game ")
540         try:
541             self.conn.sync()
542             self.rootDb = self.conn.root()
543 
544             ploca = Ploca(player1, player2)
545             now = time.time()
546             igra = Igra(ploca, now)
547             igra.live = True
548             self.rootDb["games"][igra.date] = igra
549 
550             self.kor.inGame = True
551             player2.inGame = True
552             self.kor._p_changed = True
553             player2._p_changed = True
554 
555             self.my_transaction_manager.commit()
556             self.igra = igra
557             self.Restart(igra)
558         except ConflictError or ValueError:
559             print( 'trx abort' )
560             time.sleep( 1 )
561             pass
```

Slika 14: Metoda makeGame **Vlastita izrada**

4.8. Kreiranje igre

Kod kreiranja nov igre, metoda makeGame prihvaća argumente player1 i player2 te instanciranje objekt Ploca i Igra. Igri se dodjeljuju poslani player1 i player2 te se igri dodaje instancirana ploca. Igracima se postavlja parametar inGame na vrijednsot True kako bi protivnički igrač mogao pronać igru, slika [16], a ne kreirati novu.

Igrač koji je pronađen u queue od strane drugog igrača koji je kreirao igru, ne kreira novu igru već pronalazi već kreiranu igru preko atributa date objekta igra. Metoda findGame, slika [17], pronalazi prvu igru koja se izvodi iz baze s igračem koji je dodijeljen toj igri.

4.9. Potez igrača

Svakom polju na ploči dodijeljen je event pritiska tipke miša. Prilikom pritiska tipke miša poziva se metoda ButtonClick koja ažurira root objekt, provjerava koji je trenutni aktivni igrač u objektu ploca te ako je trenutni korisnik aplikacije aktivni igrač dopušta mu postavljane simbola

Slika 15: Metoda findGame **Vlastita izrada**

na pritisnuto polje. Mijenja se aktivni igrač u suigrača i promjene se pohranjuju u bazu. Slika [18]

OPIS

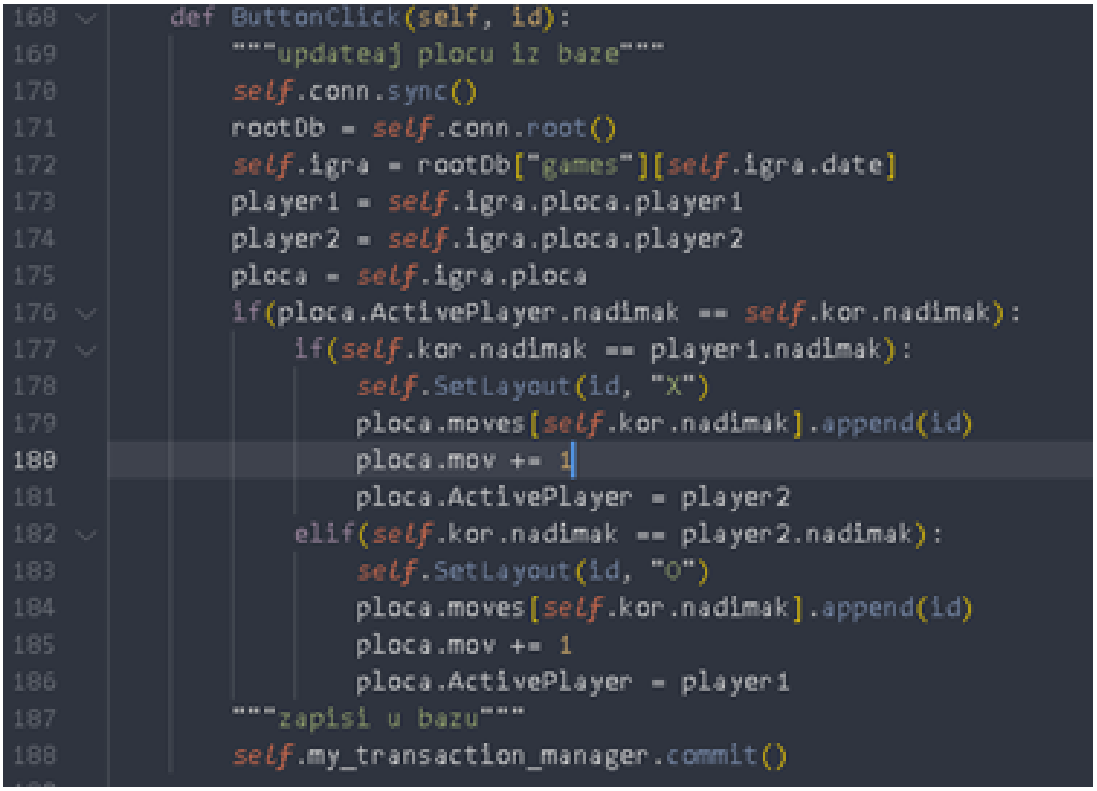
4.10. Potez igrača

Svaka ploča ima atribut s vrijednostima poteza za svakog igrača spremljenog u obliku rječnika. Ako se u nekoj listi korisnika nađe uzastopnih tri poteza slika [19] taj korisnik je pobjednik. Na kraju postavljamo atribute winner, inGame i live te prikazujemo poruku o pobjedniku slika [20].

4.11. Odjava

Prilikom odjave korisnika postavljaju se atributi live, inQueu, inGame na False kaok bi osigurali da ne dođe do rešenja aplikacije prilikom ponovnog pokretanja i brišemo objekte iz baze spremljen u live i queue rječnike. Slika [21]

Prilikom zatvaranja aplikacije postavljamo zastavice jos, next na fals kako bi prekinuli izvođenje petlji u nitima te uništava se korijen korisničkog sučelja i poziv metode logOut. Slika [22]



```
    def ButtonClick(self, id):
        """updateaj plocu iz baze"""
        self.conn.sync()
        rootDb = self.conn.root()
        self.igra = rootDb["games"][self.igra.date]
        player1 = self.igra.ploca.player1
        player2 = self.igra.ploca.player2
        ploca = self.igra.ploca
        if(ploca.ActivePlayer.nadimak == self.kor.nadimak):
            if(self.kor.nadimak == player1.nadimak):
                self.SetLayout(id, "X")
                ploca.moves[self.kor.nadimak].append(id)
                ploca.mov += 1
                ploca.ActivePlayer = player2
            elif(self.kor.nadimak == player2.nadimak):
                self.SetLayout(id, "O")
                ploca.moves[self.kor.nadimak].append(id)
                ploca.mov += 1
                ploca.ActivePlayer = player1
        """zapisi u bazu"""
        self.my_transaction_manager.commit()
```

Slika 16: Metoda ButtonClick **Vlastita izrada**

```
def provjeriActivPlayer(self):
    self.conn.sync()
    rootDb = self.conn.root()
    self.igra = rootDb["games"][self.igra.date]
    while True:
        if self.igra.live:
            try:
                t = self.my_transaction_manager.begin()
                """updateaj plocu iz baze"""
                self.conn.sync()
                rootDb = self.conn.root()
                self.igra = rootDb["games"][self.igra.date]
                player1 = self.igra.ploca.player1
                player2 = self.igra.ploca.player2
                ploca = self.igra.ploca

                self.updateBoard()
                self.CheckWinner()
                if self.igra.winner != None and self.igra.winner.nadimak == self.kor.nadimak:
                    self.my_transaction_manager.commit()

                if self.igra.live == False:
                    self.clearBoard()
                    continue

                if(ploca.ActivePlayer.nadimak == player2.nadimak):
                    self.bp2.configure(bg="Yellow")
                    self.bp1.configure(bg="white")
                elif(ploca.ActivePlayer.nadimak == player1.nadimak):
                    self.bp1.configure(bg="Yellow")
                    self.bp2.configure(bg="white")
                if(self.kor.nadimak == ploca.ActivePlayer.nadimak):
                    self.EnableAll()
                else:
                    self.DisableAll()
                time.sleep(0.5)
```

Slika 17: Metoda ButtonClick **Vlastita izrada**

```
101     def CheckWinner(self):
102         mov = self.igra.ploca.mov
103         self.p1 = self.igra.ploca.moves[self.igra.ploca.player1.nadimak]
104         self.p2 = self.igra.ploca.moves[self.igra.ploca.player2.nadimak]
105         winner = -1
106 
107         if(1 in self.p1) and (2 in self.p1) and (3 in self.p1):
108             winner = 1
109         if(1 in self.p2) and (2 in self.p2) and (3 in self.p2):
110             winner = 2
111 
112         if(4 in self.p1) and (5 in self.p1) and (6 in self.p1):
113             winner = 1
114         if(4 in self.p2) and (5 in self.p2) and (6 in self.p2):
115             winner = 2
116 
117         if(7 in self.p1) and (8 in self.p1) and (9 in self.p1):
118             winner = 1
119         if(7 in self.p2) and (8 in self.p2) and (9 in self.p2):
120             winner = 2
121 
122         if(1 in self.p1) and (4 in self.p1) and (7 in self.p1):
123             winner = 1
124         if(1 in self.p2) and (4 in self.p2) and (7 in self.p2):
125             winner = 2
126 
127         if(2 in self.p1) and (5 in self.p1) and (8 in self.p1):
128             winner = 1
129         if(2 in self.p2) and (5 in self.p2) and (8 in self.p2):
130             winner = 2
131 
132         if(3 in self.p1) and (6 in self.p1) and ( 9 in self.p1):
133             winner = 1
134         if(3 in self.p2) and (6 in self.p2) and (9 in self.p2):
135             winner = 2
136 
137         if(1 in self.p1) and (5 in self.p1) and ( 9 in self.p1):
138             winner = 1
139         if(1 in self.p2) and (5 in self.p2) and (9 in self.p2):
140             winner = 2
141 
142         if(3 in self.p1) and (5 in self.p1) and ( 7 in self.p1):
143             winner = 1
144         if(3 in self.p2) and (5 in self.p2) and (7 in self.p2):
145             winner = 2
```

Slika 18: Metoda ButtonClick a **Vlastita izrada**

```
147         if winner == 1:
148             messagebox.showinfo(title="Congratulations.", message="Player %s is the winner" % self.igra.ploca.player1.nadimak)
149             self.igra.winner = self.igra.ploca.player1
150             self.igra.ploca.player1.inGame = False
151             self.igra.ploca.player2.inGame = False
152             self.igra.live = False
153 
154         elif winner == 2:
155             messagebox.showinfo(title="Congratulations.", message="Player %s is the winner" % self.igra.ploca.player2.nadimak)
156             self.igra.winner = self.igra.ploca.player2
157             self.igra.ploca.player1.inGame = False
158             self.igra.ploca.player2.inGame = False
159             self.igra.live = False
160 
161         elif mov == 9:
162             messagebox.showinfo(title="Draw", message="It's a Draw!!")
163             self.igra.winner = None
164             self.igra.ploca.player1.inGame = False
165             self.igra.ploca.player2.inGame = False
166             self.igra.live = False
```

Slika 19: Metoda ButtonClick b **Vlastita izrada**

```
598     def logOut(self):
599         self.conn.sync()
600         self.rootDb = self.conn.root()
601         try:
602             t = self.my_transaction_manager.get()
603             self.kor.live = False
604             self.kor.inQueue = False
605             self.kor.inGame = False
606             self.kor._p_changed = True
607 
608             del self.rootDb[ 'live' ][self.kor.nadimak]
609 
610             for i in range(len(self.rootDb[ 'queue' ])):
611                 if self.rootDb[ 'queue' ][i].nadimak == self.kor.nadimak:
612                     del self.rootDb[ 'queue' ][i]
613                     break
```

Slika 20: Metoda logOut **Vlastita izrada**

```
625     def on_closing(self):
626         if messagebox.askokcancel("Quit", "Do you want to quit?"):
627             self.jos = False
628             self.logOut()
629             self.next = False
630             self.root.destroy()
```

Slika 21: Metoda logOut **Vlastita izrada**

5. Zaključak

6. Bibliografija