Zavod sv. Stanislava

Škofijska klasična gimnazija

Programiranje v Pythonu

Križci-krožci

Maturitetna seminarska naloga iz Informatike

Kandidat: Kogovšek Cene

Mentor: Starc Grlj Helena

Ljubljana Šentvid, marec 2026

Povzetek

Kot dijak, ki opravlja maturitetni predmet informatika, sem se odločil sprogramirati računalniško igro križci-krožci. Gre za preprosto, a strateško igro za dva igralca, pri kateri je cilj sestaviti tri enake znake v vrsto, stolpec ali diagonalo. Med izdelavo programa sem se soočil z različnimi izzivi, kot so logika preverjanja zmagovalnih kombinacij, izmenjevanje potez ter uporabniški vmesnik. S tem sem pridobil veliko novega znanja in izkušenj s programiranjem. Poleg same igre sem pripravil tudi spletno stran, na kateri predstavljam projekt in združujem vse izdelke, nastale v okviru maturitetne naloge.

Abstract

As a student taking the computer science matriculation subject, I decided to program a computer game called tic-tac-toe. It is a simple yet strategic two-player game in which the objective is to align three identical symbols in a row, column, or diagonal. During the development of the program, I encountered various challenges, such as implementing the logic for checking winning combinations, handling turn-taking, and designing the user interface. Through this process, I gained a great deal of new knowledge and programming experience. In addition to the game itself, I also created a website where I present the project and bring together all the work produced as part of my matriculation assignment.

Ključne besede

Programiranje, *Python*, križci-krožci

Kazalo vsebin

Kazalo slik

Kazalo razpredelnic

Kazalo grafov

Stvarno kazalo

1 Uvod

1.1 Vpeljevanje in predstavitev problema

V zadnjih letih ima programiranje vse pomembnejšo vlogo tako v izobraževanju kot v vsakdanjem življenju, saj razvija logično razmišljanje, natančnost in sposobnost reševanja problemov. Posebej pri začetnikih se pogosto uporablja pristop učenja skozi preproste projekte, ki omogočajo postopno razumevanje osnovnih programerskih konceptov. Med najpogosteje uporabljenimi primeri v literaturi in spletnih učnih virih se pojavlja prav igra križci–krožci, saj gre za problemsko področje z jasno definiranimi pravili, omejenim številom možnosti in pregledno logično strukturo. Različni učbeniki in programerski vodiči jo navajajo kot primer za učenje pogojev, zank, dela s podatkovnimi strukturami in razmišljanja v korakih.

Pri zbiranju podatkov za to nalogo sem se oprl predvsem na strokovne spletne vire, učne materiale za začetnike programiranja ter primere podobnih projektov, ki prikazujejo različne pristope k reševanju istega problema. Glavna problematika pri zbiranju podatkov je bila v tem, da so številni viri bodisi preveč poenostavljeni bodisi že preveč zahtevni, zato je bilo potrebno izbrati takšne, ki so ustrezni srednješolskemu znanju in omogočajo jasno razlago postopkov brez nepotrebne zapletenosti. Poseben poudarek je bil namenjen razumevanju logike igre, ne zgolj končni kodi.

Osrednji problem, s katerim se naloga ukvarja, je pretvorba preproste družabne igre v računalniški program, ki mora natančno slediti pravilom in hkrati pravilno obravnavati vse možne poteke igre. Program mora omogočiti izmenično igranje dveh igralcev, preprečiti neveljavne poteze ter samodejno prepoznati zmagovalne kombinacije ali neodločen izid. Gre za problem, ki na videz deluje preprost, vendar zahteva sistematično razmišljanje, jasno strukturiranje kode in natančno preverjanje vseh možnih primerov.

Zbrani podatki iz virov so pokazali, da je uspešna rešitev takšnega problema odvisna predvsem od dobrega načrtovanja pred samim programiranjem. Prav zato je v nalogi velik poudarek namenjen razlagi postopkov in logike, ki vodijo do končne rešitve. Cilj seminarske naloge je razviti delujoč program igre križci–krožci v programskem jeziku *Python* ter ob tem prikazati razumevanje osnov programiranja in sposobnost samostojnega reševanja jasno definiranega računalniškega problema.

1.2 Uporabljena tehnologija

1.2.1 Strojna oprema

Razpredelnica 1: Strojna oprema

Prenosni računalnik	Huawei MateBook X Pro
Procesor	Intel(R) Core(TM) i7-8550U CPU @ 1,80GHz
Grafična kartica	Intel(R) UHD Graphics 620 (128 MB)
Disk	477 GB SSD LITEON CA3-8D512
Delovni pomnilnik	16 GB 2133 MHz
Rezolucija monitorja	13,9-palcev 3K (3000x2000) IPS Touchscreen 59 Hz

1.2.2 Programska oprema

Razpredelnica 2: Programska oprema

Operacijski sistem	Microsoft Windows 10 Pro
Vrsta sistema	64-bitni OS, procesor na osnovi arhitekture x64
Različica BIOS-a	HUAWEI 1,28
Programi	Visual studio code, Google chrome

2 Teoretična rešitev

2.1 Analiza, razlaga in kritično ovrednotenje zbranih podatkov

Pri načrtovanju rešitve sem najprej zbral informacije o tem, kako se preproste igre običajno pretvorijo v računalniški program. Osredotočil sem se na vire, ki razlagajo osnovno strukturo programov, delo s pogoji, spremenljivkami in funkcijami ter uporabo grafičnega uporabniškega vmesnika v programskem jeziku *Python*. Večina uporabljenih virov je bila dostopna na spletu, predvsem v obliki učnih člankov in video posnetkov.

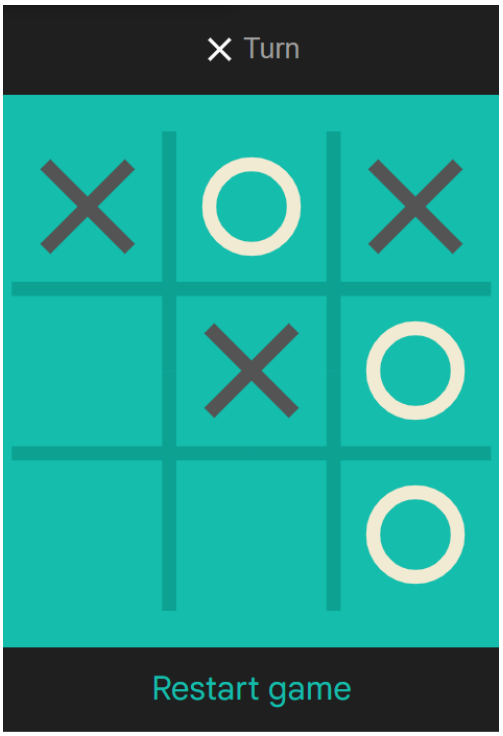
Pri zbiranju podatkov se je pokazalo, da obstaja veliko različnih načinov, kako rešiti isti problem. Nekateri viri so ponujali zelo kratke in poenostavljene razlage, drugi pa so bili preveč obsežni in zahtevni za začetno raven znanja. To je otežilo izbiro ustreznega pristopa, saj vsi viri niso bili enako uporabni za moj projekt. Zato sem moral podatke primerjati in izbrati tiste rešitve, ki so bile dovolj jasne in hkrati dovolj natančne, da sem jih lahko razumel in kasneje uporabil.

Posebno pozornost sem namenil virom, ki so obravnavali delo z grafičnim uporabniškim vmesnikom. Ugotovil sem, da se pri *Pythonu* pogosto uporablja knjižnica *Tkinter*, ki omogoča izdelavo preprostih oken, gumbov in drugih grafičnih elementov. Viri so pokazali, da je delo s *Tkinterjem* primerno za manjše projekte, vendar zahteva razumevanje načina, kako program reagira na uporabnikove dogodke, kot so kliki z miško.

Večino znanja o uporabi *Tkinterja* sem načrtoval pridobiti s pomočjo video posnetkov na platformi YouTube. Tak način učenja se je izkazal za uporabnega, ker omogoča postopno spremljanje celotnega procesa izdelave programa in hkrati prikazuje takojšnje rezultate sprememb v kodi. Slabost takšnih virov je, da pogosto ne razložijo vseh podrobnosti, zato je bilo potrebno nekatere informacije dodatno preveriti v drugih virih.

Na podlagi zbranih podatkov sem ugotovil, da je uspešna rešitev problema odvisna predvsem od dobrega načrta pred samim programiranjem. Brez jasne predstave o poteku igre, preverjanju pravilnosti potez in zaključku igre lahko hitro pride do napak ali nepregledne kode. Zato sem se odločil, da bom rešitve najprej razmislil na teoretični ravni in šele nato prešel na praktično izvedbo.

2.1.1 Primer igre na internetu



Slika 1: Primer križcev-krožcev

2.2 Opis rešitve

Teoretična rešitev temelji na postopni izdelavi programa, pri kateri ima vsak del jasno določen namen. Najprej je potrebno določiti osnovno strukturo igre križci–krožci. Igra poteka na mreži velikosti tri krat tri, kjer dva igralca izmenično zasedata prazna polja. Program mora v vsakem trenutku hraniti podatek o stanju vseh polj in o tem, kateri igralec je trenutno na potezi.

Naslednji korak je določitev načina vnosa potez. Uporabnik mora imeti možnost izbrati polje, pri čemer program preveri, ali je izbrano polje prazno in ali igra še poteka. Če pogoji niso izpolnjeni, se stanje igre ne sme spremeniti. S tem se preprečijo napačni vnosi in neveljavni poteki igre.

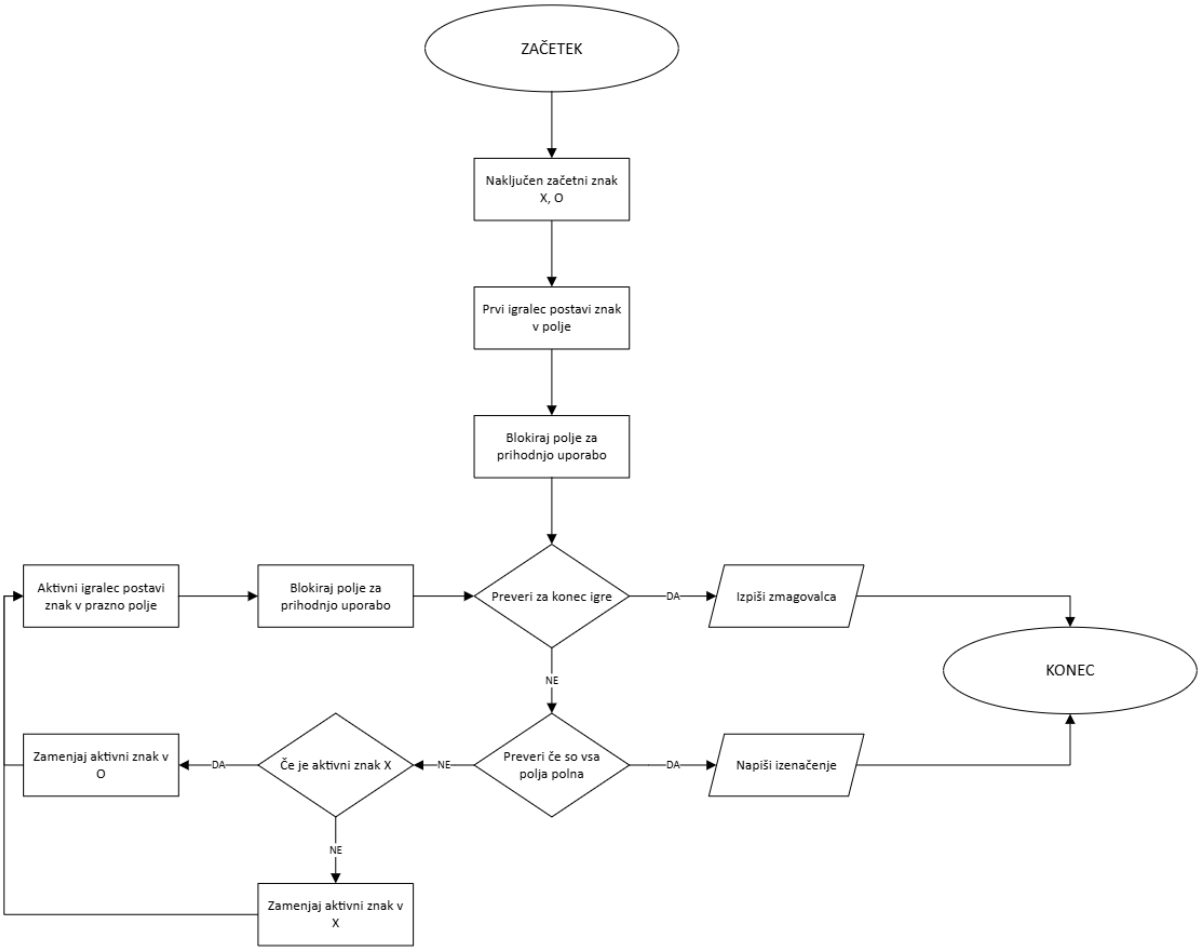
Pomemben del rešitve je preverjanje zmagovalnih pogojev. Program mora po vsaki potezi preveriti, ali obstaja zaporedje treh enakih znakov v isti vrstici, stolpcu ali diagonali. Če je takšno zaporedje zaznano, se igra zaključi in rezultat ustrezno prikaže. Če so vsa polja zapolnjena in zmagovalca ni, program zazna neodločen izid.

V teoretičnem načrtu je predvideno tudi jasno obveščanje uporabnika o poteku igre. Program mora prikazovati, kdo je na potezi, ter po koncu igre izpisati rezultat. To omogoča boljšo preglednost in razumevanje delovanja programa.

Za izvedbo grafičnega dela programa je predvidena uporaba knjižnice *Tkinter*. Najprej je načrtovana izdelava osnovnega okna, nato dodajanje igralne mreže v obliki gumbov ter dodatnih elementov, kot so besedilni prikazi in gumb za ponovni začetek igre. Učenje uporabe teh elementov bo potekalo s pomočjo video učnih vsebin na YouTubu.

V zadnjem delu rešitve je predvidena možnost ponastavitve igre, pri čemer se stanje vseh polj izbriše in igra začne znova. To zahteva ponovno nastavitev notranjih spremenljivk in grafičnih elementov. Celotna rešitev je zasnovana tako, da omogoča jasno strukturo programa in pravilno delovanje v vseh običajnih primerih igranja.

2.2.1 Potek igre



Slika 2: Potek igre križci-krožci

3 Praktična rešitev

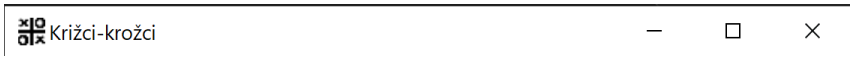
3.1 Osnovni del programa

Na začetku programa sem uporabil *import* s katerim uvozimo tri knjižnice, ki jih potrebujemo za delovanje. Knjižnica *os* omogoča delo z operacijskim sistemom, na primer upravljanje datotek in poti. *tkinter* se uporablja za ustvarjanje grafičnega uporabniškega vmesnika, pri čemer z ukazom *from tkinter import* * uvozimo vse njegove osnovne elemente, kot so okna, gumbi in besedila. Knjižnica *random* pa omogoča generiranje naključnih vrednosti, kar je uporabno pri različnih delih programa, kjer potrebujemo naključnost.

```
1  import os
2  from tkinter import *
3  import random
```

Slika 3: Začetek programa import

S pomočjo *tkinterja* sem ustvaril okno, kjer se bo odvijala igra. Okencu sem obarval ozadje, ga poimenoval in s pomočjo *os* sem nastavil ikono programu.



Slika 4: Ime programa in ikona

```
okno = Tk()
okno.config(bg="#ffebcd")
okno.title("Križci-krožci")
pot = os.path.join(os.path.dirname(__file__), "krizci-krozci-icon.png")
icon = PhotoImage(file=pot)
okno.iconphoto(True, icon)
```

Slika 5: Ustvarjanje okna

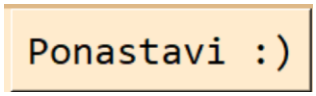
Definiral sem igralca in z uporabo *random* naključno izbral tistega, ki je prvi na potezi. Okencu sem dodal tudi besedilo, kjer piše ime igralca, ki je trenutno na potezi ter pobarval ozadje.

Slika 6: Besedilo igralca na potezi

```
 99   igralci = ["X", "O"]
100   igralec = random.choice(igralci)
101
102   poteza_text = Label(text= "Igralec: " + igralec, font=("Times New Roman", 40))
103   poteza_text.config(background="#deb887", padx=143, fg="white")
104   poteza_text.pack(side="top")
```

Slika 7: Koda igralca na potezi

Dodal sem tudi gumb za ponastavitev igre, ki sproži funkcijo *Nova_igra* ter definiral tabelo in obliko polja z *integerji* in *stringi*.

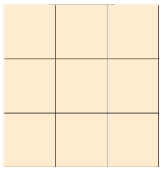


Slika 8: Gumb za ponastavitev

```
106   gumb_reset = Button(text="Ponastavi :)", font=("consolas", 20), command=Nova_igra)
107   gumb_reset.config(bg="#ffebcd", activebackground="white")
108   gumb_reset.pack()
109
110   tabela = [[0,0,0],
111             [0,0,0],
112             [0,0,0]]
113
114   igra = [["","",""],
115           ["","",""],
116           ["","",""]]
```

Slika 9: Koda za ponastavitev

Ustvaril sem okvir ter mu, s pomočjo dvojne *for* zanke, dodal gumbe razporejene v 3x3 obliko. Pri kliku vsakega iz med teh, se sproži funkcija *Naslednja_poteza*.



Slika 10: Igralno polje

```
okvir = Frame(okno)
okvir.pack()

for vrsta in range(3):
    for stolp in range(3):
        tabela[vrsta][stolp] = Button(okvir, text="", font=("Times New Roman", 40), width=5, height=2, activebackground="white",
                                      bg="#ffebcd", command=lambda v=vrsta, s=stolp: Naslednja_poteza(v, s))
        tabela[vrsta][stolp].grid(row=vrsta,column=stolp)

okno.mainloop()
```

Slika 11: Koda za igralno polje

3.2 Funkcije programa

3.2.1 Funkcija *Naslednja_poteza*

Funkcija *Naslednja_poteza* preveri, če je pritisneno polje prazno in če je, v gumb napiše rdeč križec ali pa moder krožec, odvisno od tega kdo je na potezi. Nato zamenja trenutnega igralca in požene funkcijo *if_konec*, ki preveri, za zmago ali pa izenačenje.

```
def Naslednja_poteza(vrsta, stolp):
    global igralec, igralci, zapri_polja
    gumb = tabela[vrsta][stolp]

    if gumb.cget("text") == "" and zapri_polja == False:
        igra[vrsta][stolp] = igralec
        tabela[vrsta][stolp].config(text=igralec)
        if igralec == igralci[0]:
            tabela[vrsta][stolp].config(fg="red")
            igralec = igralci[1]
        else:
            tabela[vrsta][stolp].config(fg="blue")
            igralec = igralci[0]
        poteza_text.config(text= "Igralec: " + igralec)
        if_konec()

zapri_polja = False
```

Slika 12: Funkcija Naslednja_poteza

3.2.2 Funkcija *if_konec*

Funkcija *if_konec* najprej, s pomočjo *for* zanke, pogleda vse vrste, če so znaki enaki, za tem pogleda vse stolpce in če so vsi enaki, pošlje v fukcijo *konec* katere kombinacije so zmagale in kater znak je zmagal.

```
26  def if_konec():
27      global igra, igralci
28      for n in range(3):
29      #Preveri za X:
30          #Preveri vse vrste
31          if igra[n][0] == igralci[0] and igra[n][1] == igralci[0] and igra[n][2] == igralci[0]:
32              konec(igralci[0], [(n,0),(n,1),(n,2)])
33          #Preveri vse stolpce
34          if igra[0][n] == igralci[0] and igra[1][n] == igralci[0] and igra[2][n] == igralci[0]:
35              konec(igralci[0], [(0,n),(1,n),(2,n)])
```

Slika 13: Funkcija if_konec 1

Podobno pogleda tudi za diagonale. Na koncu funkcije pa še sproži funkcijo *Polna_polja*.

```
    if igra[0][2] == igralci[1] and igra[1][1] == igralci[1] and igra[2][0] == igralci[1]:
        konec(igralci[1], [(0,2),(1,1),(2,0)])

    Polna_polja()
```

Slika 14: Funkcija if_konec 2

3.2.3 Funkcija *Polna_polja*

Funkcija *Polna_polja* gre s *for* zanko čez vsa polja in preveri, če je še kakšno polje prazno, če je se ne zgodi nič, če pa ni pa sproži funkcijo *konec*.

```
def Polna_polja():
    global igra
    for i in igra:
        if "" in i:
            return
    konec("Izenačenje", 0)
```

Slika 15: Funkcija Polna_polja

3.2.4 Funkcija *konec*

Ko se sproži funkcija *konec*, se takoj zaprejo polja, tako da ne moreš več klikati na gumbe. Za tem, če je izenačenje se to izpiše ter vsa polja se pobarvajo na sivo. Če pa ni izenačenja, se izpiše zmagovalec in zmagovalna polja se pobarvajo zeleno.

```
def konec(x, polja):
    global zapri_polja

    zapri_polja = True
    if x == "Izenačenje":
        poteza_text.config(text= "Izenačenje!", padx=126)
        for vrsta in range(3):
            for stolp in range(3):
                tabela[vrsta][stolp].config(bg="lightgray")
    else:
        poteza_text.config(text= "Zmagal je igralec "+str(x)+"!", padx=30)
        for vrsta, stolp in polja:
            tabela[vrsta][stolp].config(bg="lightgreen")

```

Slika 16: Funkcija konec

3.2.5 Funkcija *Nova_igra*

Ko se sproži funkcija *Nova_igra*, se vsa polja odprejo in ponastavi se igralno polje na prazno. Tekst ponovno kaže igralca, ki je na potezi. Začne pa tisti, ki je prejšnjo igro izgubil, če pa je bilo izenačeno, začne tisti, ki prejšnje ni začel. Na koncu se pa še vsa polja pobarvajo na prvotno barvo.

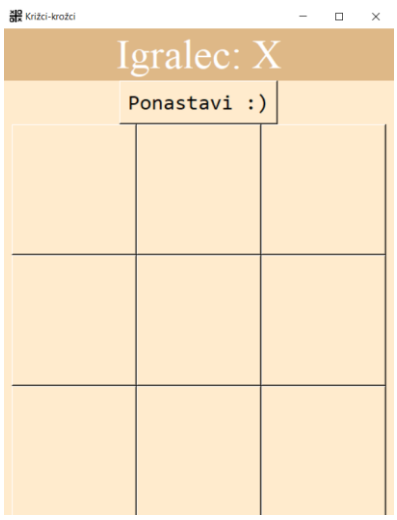
```
67  def Nova_igra():
68      global igra, zapri_polja
69
70      zapri_polja = False
71      igra = [["","",""],
72              ["","",""],
73              ["","",""]]
74
75      poteza_text.config(text= "Igralec: " + igralec, padx=143)
76
77      for vrsta in range(3):
78          for stolp in range(3):
79              tabela[vrsta][stolp].config(text="", bg="#ffebcd")
80
```

Slika 17: Funkcija Nova_igra

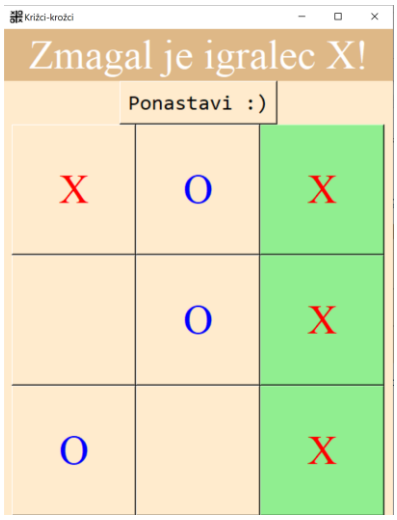
4 Sklep

4.1 Izgled končnega izdelka

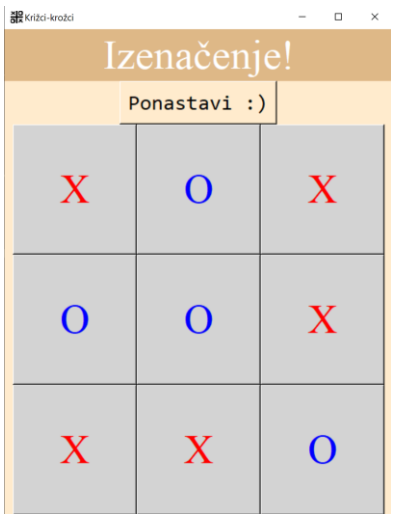
Začetek: Zmaga: Izenačenje:



Slika 18: Primer izenačenja



Slika 19: Primer zmage



Slika 20: Primer začetka

4.2 Navodila za uporabo

Program zaženeš in se s prijateljem dogovoriš kdo je kater uzorec (X ali O). Izmenično je vsak na potezi in tisti, prvi doseže tri v vrsto zmaga! Če se napolnijo polja preden pride do zmage je izenačenje. Med ali po končani igri lahko ponastaviš igro s klikom na gumb *Ponastavi*.

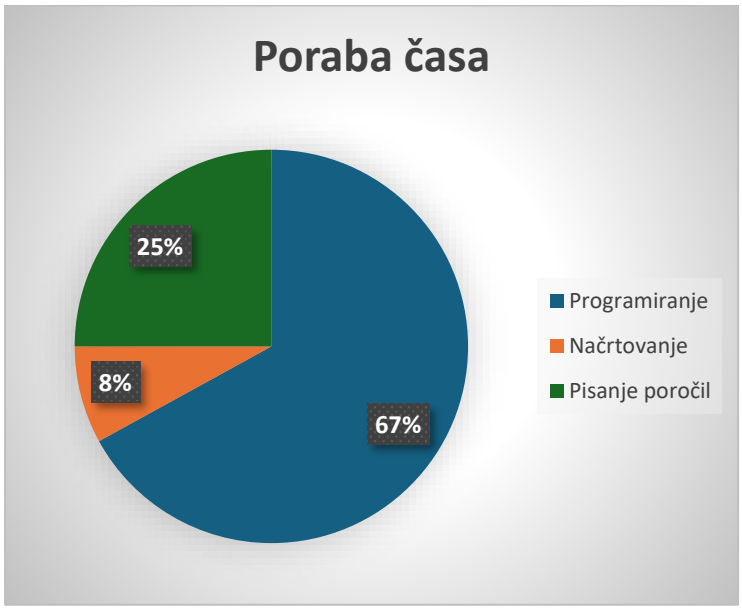
4.3 Možne izboljšave

Čeprav sem, glede na svoje znanje programiranja, zelo zadovoljen z izidom programa, je še vedno veliko možnih izboljšav. Na primer:

- Črta čez zmagovalna polja
- Zapisovanje preteklih zmag
- Zvočni efekti
- Osvetlitev polj ob hoverju
- AI nasprotnik

4.4 Poraba časa

Večino časa sem porabil na programiranju in reševanju problemov, ki so se mi pojavili na poti. Ta čas mi je hitro mineval, saj sem z navdušenjem programiral, dokler nisem bil pod časovnim pritiskom. Četrtino časa sem porabil na pisanju poročil, kar je bil še najbolj mučen del te seminarske naloge. Ostalo sem pa porabil za še potrebno načrtovanje in minimalno izobraževanje o *tkinterju* in *os-u*.



Graf 1: Poraba časa

5 Viri

- Učenje *Pythona* (online). [Uporabljeno 27. 2. 2026].

Dostopno na https://www.w3schools.com/python/default.asp

- Učenje *Pythona* (online). [Uporabljeno 27. 2. 2026].

Dostopno na https://lusy.fri.uni-lj.si/ucbenik/book/index.html#

- Pomoč pri razumevanju in programiranju (online). [Uporabljeno 27. 2. 2026].

Dostopno na https://chatgpt.com/

- Učenje *Pythona* (online). [Uporabljeno 27. 2. 2026].

Dostopno na https://www.youtube.com/

- Primer igre križci–krožci (online). [Uporabljeno 27. 2. 2026].

Dostopno na https://playtictactoe.org/

- Šolska spletna učilnica (online). [Uporabljeno 27. 2. 2026].

Dostopno na https://ucilnica.stanislav.si