# Implementacija igri primjenom koncepata funkcijskog programiranja

**Baričević, Mateo**

**Undergraduate thesis / Završni rad**

**2020**

*Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj:* **University of Rijeka / Sveučilište u Rijeci**

*Permanent link / Trajna poveznica:* https://urn.nsk.hr/urn:nbn:hr:195:621356

*Rights / Prava:* In copyright / Zaštićeno autorskim pravom.

*Download date / Datum preuzimanja:* **2024-04-25**

Sveučilište u Rijeci
**Fakultet informatike i digitalnih tehnologija**

*Repository / Repozitorij:*

Repository of the University of Rijeka, Faculty of Informatics and Digital Technologies - INFORI Repository

zir.nsk.hr

uniri DIGITALNA KNJIŽNICA

dabar
DIGITALNI AKADEMSKI ARHIVI I REPOZITORIJI

Sveučilište u Rijeci – Odjel za informatiku

Preddiplomski jednopredmetni studij informatike

Mateo Baričević

# Implementacija igri primjenom koncepata funkcijskog programiranja

Završni rad

Mentor: izv. prof. dr. sc. Ana Meštrović

Rijeka, rujan 2020.

ODJEL ZA INFORMATIKU
SVEUČILIŠTE U RIJECI

Rijeka, 9.6.2020.

# Zadatak za završni rad

Pristupnik: Mateo Baričević

Naziv završnog rada: Implementacija igri primjenom koncepata funkcijskog programiranja

Naziv završnog rada na eng. jeziku: Implementation of games using functional programming concepts

Sadržaj zadatka:

Zadatak završnog rada je opisati mogućnosti primjene koncepata funkcijskog programiranja za implementaciju igri. U radu je potrebno ukratko opisati različite paradigme, te usporediti funkcijski stil programiranja s imperativnim stilom. Potrebno je analizirati primjenu funkcijskih koncepata na primjeru implementacije jednostavnijih igri u odabranom programskom jeziku.

Mentor

Izv. prof. dr. sc. Ana Meštrović

Voditelj za završne radove

Dr. sc. Miran Pobar

Zadatak preuzet: 9.6.2020.

Baričević Mateo

(potpis pristupnika)

## Sažetak

Cilj ovog rada je prikazati koncepte funkcijskog programiranja kroz njihovu primjenu u implementaciji jednostavnih računalnih igara. Za njihovo programiranje korišten je programski jezik Python koji podržava više paradigmi. Prije same implementacije tih koncepata, opisane su programske paradigme nakon kojih su opisani koncepti funkcijskog programiranja koje Python podržava, te njihova realizacija. Zatim su opisane prednosti i nedostaci funkcijskog programiranja u Pythonu. Nakon kojih slijedi njihova primjena u primjeru razvoja igre križić-kružić. U tom poglavlju je objašnjeno gdje se koji koncept primijenio, te na koji način. Također je opisana i usporedba programskog kôda napisanog proceduralnom paradigmom s kôdom nakon primjene koncepata funkcijskog programiranja. Potom slijedi poglavlje u kojem se također prikazuje primjena funkcijskih koncepata, ali u primjeru razvoja igre vješala. U ovom poglavlju se također opisuje gdje se i na koji način primijenio pojedini koncept te usporedba kôda. Na kraju rada objašnjene su korisnosti svakog koncepta.

## Ključne riječi

funkcijsko programiranje, programska paradigma, Python, čiste funkcije, nepromjenjivost varijabli, funkcija višeg reda, lambda izrazi, map, filter, sažet zapis listi, razvoj igri, minimax

# Sadržaj

Sažetak ..... I
Ključne riječi ..... I
1. Uvod ..... 1
2. Paradigme programiranja ..... 2
2.1. Proceduralna paradigma ..... 2
2.2. Funkcijska paradigma ..... 2
2.3. Objektno orijentirana paradigma ..... 3
3. Funkcijska paradigma u programskom jeziku Python ..... 5
3.1. Realizacija koncepata funkcijske paradigme u Pythonu ..... 5
3.1.1. Čiste funkcije ..... 5
3.1.2. Nepromjenjivost varijabli ..... 5
3.1.3. Funkcije višeg reda ..... 6
3.1.4. Lambda izrazi ..... 7
3.1.5. Ugrađene funkcije višeg reda ..... 8
3.1.6. Sažet zapis listi ..... 9
3.1.7. Rekurzija ..... 10
3.2. Tipovi podataka u Pythonu ..... 10
3.3. Prednosti funkcijskog programiranja u Pythonu ..... 11
3.4. Nedostaci funkcijskog programiranja u Pythonu ..... 11
4. Primjena koncepata funkcijskog programiranja u primjeru razvoja igre križić-kružić ..... 12
4.1. Strategija za optimalno igranje ..... 12
4.2. Naprednija verzija križić-kružića ..... 12
4.3. Čiste funkcije ..... 12
4.4. Funkcije višeg reda i nepromjenjivost varijabli ..... 13
4.5. Map i lambda izrazi ..... 13
4.6. Sažet zapis listi ..... 14
4.7. Filter ..... 14
4.8. Minimax algoritam i rekurzija ..... 15
4.9. Usporedba kôda ..... 17
5. Primjena koncepata funkcijskog programiranja u primjeru razvoja igre vješala ..... 19
5.1. Strategija za optimalno igranje ..... 19
5.2. Čiste funkcije i nepromjenjivost varijabli ..... 20
5.4. Map i lambda izrazi ..... 20

5.5. Filter ..... 21
5.6. Sažet zapis listi ..... 21
5.8. Usporedba kôda ..... 22
6. Zaključak ..... 23
7. Literatura ..... 25
8. Popis slika ..... 27
9. Popis tablica ..... 27
10. Popis priloga ..... 27
Prilog A ..... 28
Prilog B ..... 37
Prilog C ..... 46
Prilog D ..... 50

# 1. Uvod

U današnje vrijeme postoji puno programskih jezika, pa tako i kategorija u koje se mogu svrstati. Programska paradigma je jedan od načina koji pokušava klasificirati programske jezike na temelju njihovih značajki ili stilu pisanja kôda. Dakle, možemo reći da je programska paradigma stil ili način programiranja.

Većinu vremena na Python se gleda kao objektno orijentirani jezik, gdje se podaci modeliraju u obliku klasa, objekata i metoda. Međutim, postoji i nekoliko alternativa objektno orijentiranom programiranju, a jedna od njih je funkcijsko programiranje.

Klasična podjela programskih jezika se sastoji od dvije paradigme programiranja, a one su imperativna i deklarativna. Pod imperativnom paradigmom spadaju proceduralna i objektno orijentirana, a pod deklarativnom spadaju funkcijska, logička i matematička paradigma. U ovome radu ćemo se fokusirati na funkcijsku paradigmu.

Funkcijsko programiranje se bazira na funkcijama kao glavnim konceptom programiranja, sve se definira pomoću funkcija. Fokusira se na definiranje što činiti, umjesto izvršavanja neke akcije. Funkcije se tretiraju kao bilo koji objekt, mogu se pridružiti varijablama ili proslijediti u druge funkcije. Podaci moraju biti nepromjenjivi, što znači da je promjena podataka u nekoj listi nemoguća i da uvijek moramo napraviti novu varijablu s promijenjenim podacima umjesto mijenjanja već definirane varijable. Također, programi pisani funkcijskim programiranjem moraju biti neovisni o nekom stanju. To znači da bi svaki zadatak trebali izvršavati kao da im je prvi put da ga izvršavaju. Drugim riječima, funkcije moraju biti ovisne samo o podacima koji im se prosljeđuju kao parametri. Podaci izvan okvira funkcije ne smiju utjecati na izvršavanje te funkcije. Lijenost je još jedna karakteristika funkcijskog programiranja, gdje se vrijednosti ne izračunavaju sve dok nisu zatraženi.

Python podržava koncepte kao što su funkcije višeg reda, lambda izrazi, *map* i *filter* funkcije, te sažet zapis listi.

U drugom poglavlju će se detaljnije objasniti paradigme programiranja. U trećem poglavlju će se razraditi svi koncepti funkcijskog programiranja u Pythonu, te njihova realizacija. U četvrtom poglavlju će se prikazati kako su se ti koncepti primijenili u implementaciji igre križić-kružić, a u petom poglavlju u implementaciji igre vješala.

Funkcijsko programiranje u akademiji je aktivna grana istraživanja u teoriji programskih jezika. U industriji se funkcijsko programiranje koristilo za razvoj raznih aplikacija u kompanijama kao što su Facebook [1], WhatsApp [2]. Erlang je funkcijski programski jezik koji je dobar odabir za razvoj komunikacijskih aplikacija koje pružaju komunikaciju u stvarnom vremenu [3]. Scheme, dijalekt LISP-a, je korišten za razvoj aplikacija Apple Macintosh računala [4], te kod treniranja simulacijskog softvera [5] i kontrole teleskopa [6]. U edukaciji neki koriste funkcijsko programiranje kao uvod u programiranje, dok neki prvo uče imperativno programiranje. Također se koristi kao metoda učenja rješavanja matematičkih problema, algebre i geometričkih koncepata.

# 2. Paradigme programiranja

## 2.1. Proceduralna paradigma

Proceduralna paradigma je pristup programiranju koji se koristi od samog začetka programiranja. Koristeći ovu paradigmu, program se sastoji od lista instrukcija koje računalo izvršava redoslijedom kojim su napisane, osim ako drugačije nije definirano. Ovaj pristup je vrlo jednostavan i kôd pisan u ovoj paradigmi je vrlo čitljiv ako je program razumno kratak. Duži programi pisani koristeći proceduralnu paradigmu mogu biti nepregledni, što rezultira teškim daljnjim razvojem programa, te im nije lako otkloniti pogreške.

Većina proceduralnih programskih jezika ima strukture kontrole toka programa: *if* naredbe koje se koriste za grananje izvršavanja kôda ovisno o nekom uvjetu, *for* i *while* petlje za ponavljanje izvršavanja nekog kôda i *goto* naredba koje omogućavaju skok na neki drugi dio kôda. Jedna od podparadigma proceduralne paradigme je strukturalna paradigma koja izostavlja *goto* izjavu, što znači da se program mora izvršiti bez preskakanja bilo kojih naredbi [7].

Programski jezici zasnovani na proceduralnoj paradigmi uključuju većinu ranih jezika kao što su Fortran, COBOL, BASIC, Pascal i C.

Slijedi primjer kako se može implementirati funkcija faktorijela koristeći proceduralni programski jezik C.

```
#include <stdio.h>
int main()
{
    int n = 10;
    unsigned long long factorial = 1;

    for (int i = 1; i <= n; ++i)
    {
        factorial *= i;
    }
    printf("%d! = %llu", n, factorial);

    return 0;
}
```

## 2.2. Funkcijska paradigma

Funkcijska paradigma bitno se razlikuje od proceduralne paradigme u načinu na koji programeri razvijaju svoje programe. U proceduralnoj paradigmi programeri određuju način na koji program funkcionira, a u funkcijskoj paradigmi programer definira željeni ishod i ne može definirati način na koji se funkcije izvršavaju. Što znači da programi pisani deklarativnom paradigmom objašnjavaju što treba izračunati, ali ne nužno i kako se računa. Opisuje se logika kojom računalo može doći do rezultata bez opisivanja postupka kako doći do rezultata.

Njoj suprotna paradigma se zove imperativna paradigma. Imperativni programi računalu opisuju koje korake treba poduzeti za rješavanje nekog problema. Funkcijsko programiranje oslanja se na koncept lambda računa, koji definira način pretvaranja bilo koje računalne funkcije u matematičku funkciju koja se može izraziti funkcijskim jezicima [7].

Najvažniji funkcijski programski jezik je LISP (LISt Processing). Još neki primjeri funkcijskih jezika su Scheme (dijalekt LISP-a), Haskell, Mathematica, Erlang, Clojure i F#.

Slijedi primjer implementacije funkcije faktorijela koristeći funkcijski programski jezik Haskell:

```
module Main where

import Text.Printf

factorial :: Integer -> Integer
factorial 0 = 1
factorial n = n * factorial (n - 1)

main = printf "%d! = %d" 10 $ factorial 10
```

## 2.3. Objektno orijentirana paradigma

Objektno orijentiranu paradigmu razvio je Alan Kay dok je radio na objektno orijentiranom jeziku Smalltalku [8]. Ovaj pristup je razvijen kako bi olakšalo upravljanje velikim projektima te podjelu opterećenja među suradnicima, a jednostavan je zahvaljujući modularnosti objekata.

Glavni fokus objektno orijentirane paradigme je objekt. Objekt ili entitet je način grupiranja struktura podataka, koje nazivamo atributima, i funkcija koje se mogu provesti na tim podacima, koje nazivamo metodama [9]. Razina složenosti objekta ovisi o razini željene apstrakcije.

Nakon objekata, klasa je sljedeća glavna značajka objektno orijentirane paradigme. Objekti koji imaju iste ili slične karakteristike se klasificiraju u klase. Klasa je logičko grupiranje objekata bazirano na nekim kriterijima njihovih zajedničkih karakteristika.

Odnosi između predmeta i klasa općenito se opisuju kao odnos „je“. Objekt „je“ klasa. Takvi odnosi čine hijerarhijsku strukturu između objekata i klasa. Također, klasa može biti podklasa neke klase. Postoje dva tipa „je“ odnosa, a to su agregacija i generalizacija.

Agregacija je kombiniranje komponenata u neki entitet. Na primjer, komponente računala agregacijom tvore objekt računalo. Suprotnost agregaciji je dekompozicija, raščlanjivanje objekta na njegove komponente.

Generalizacija je kombiniranje objekata ili klasa niže razine u klase više razine generaliziranjem njihovih atributa. Na primjer, kombiniranjem objekta računalo s objektom stol je generalizacija klase ured.

Nasljeđivanje je koncept kojim se svojstva nadređenih objekata ili klasa prenose na podređene objekte ili klase. Nasljeđivanjem se mogu definirati kompleksniji objekti i klase. Korištenjem nasljeđivanja se kôd „reciklira“ i time je ponovna upotrebljivost kôda veća [10].

Primjenom pravilnog tipiziranja podataka i metoda, programer može postići visoku razinu enkapsulacije. Enkapsulacija se odnosi na grupiranje podataka s metodama koje djeluju na tim podacima ili ograničavanje izravnog pristupa nekim atributima objekta [7].

Prvi programski jezik koji je implementirao značajke objektno orijentirane paradigme bio je Smalltalk. Temelji se na objektima, što znači da ne postoje podaci ili metode izvan objekta.

Sljedeći programski jezik fokusiran na objektno orijentiranu paradigmu bio je skup proširenja za C kolektivno nazvan C with Classes, što je kasnije dovelo do razvoja jednog od najutjecajnijih objektno orijentiranih jezika današnjice, C++.

Drugi poznati objektno orijentirani jezik je Java, koji se danas široko koristi za stvaranje programa u rasponu od *servleta* za interaktivne web stranice do mobilnih i računalnih igara. Usredotočuje se na kompatibilnost s više platformi koristeći *Java Virtual Machine* za pokretanje istog kôda na širokom rasponu različitih vrsta i arhitektura hardvera. Iako je praktičan za prenosivost, programi u ovom jeziku su manje učinkoviti u korištenju resursa te sporiji od ostalih programskih jezika.

Još neki primjeri objektno orijentiranih jezika su C#, Python, R, PHP, JavaScript, Ruby i Perl.

Slijedi primjer implementacije funkcije faktorijela koristeći objektno orijentirani jezik Java:

```
public class Factorial
{
    public static void main(String[] args)
    {
        int n = 10;
        long factorial = 1;
        for (int i = 1; i <= n; ++i)
        {
            factorial *= i;
        }
        System.out.printf("%d! = %d", n, factorial);
    }
}
```

# 3. Funkcijska paradigma u programskom jeziku Python

Python je jezik opće namjene koji podržava više programskih paradigmi. Većim dijelom je imperativni jezik jer potpuno podržava objektno orijentiranu i strukturalnu paradigmu, ali je na njegov razvoj utjecala i deklarativna paradigma.

Na neke od značajki funkcijskog programiranja koje Python podržava taj utjecaj je bio čisto funkcijski jezik Haskell. Neki od koncepata Haskella koji mogu biti poželjni su čiste funkcije, nepromjenjivost varijabli te funkcije višeg reda. Haskell je također utjecao na iteratore i generatore u Pythonu svojim lijenim učitavanjem, ali ta značajka nije potrebna da bi jezik bio funkcijski jezik.

Python se inače više koristi kao imperativni jezik, ali po potrebi se može upotrijebiti i njegov deklarativni dio [11].

## 3.1. Realizacija koncepata funkcijske paradigme u Pythonu

### 3.1.1. Čiste funkcije

Čiste funkcije ne mijenjaju vrijednost varijable koja je proslijeđena kao parametar ili bilo kakvih varijabli koje nisu dio funkcije. Ona će za iste ulazne parametre uvijek dati isti izlaz.

Slijedi primjer čiste funkcije koja množi listu brojeva s dva.

```
def pomnozi2(brojevi):
    novi = []
    for n in brojevi:
        novi.append(n * 2)
    return novi

brojevi = [1, 2, 3, 4]
noviBrojevi = pomnozi2(brojevi)
print(brojevi) #  [1, 2, 3, 4]
print(noviBrojevi) #  [2, 4, 6, 8]
```

Izvorna lista brojeva je nepromijenjena i ne referenciramo nijednu drugu varijablu izvan funkcije, tako da možemo reći da je ova funkcija čista. Testiranje čiste funkcije je olakšano jer se nakon pozivanja čiste funkcije ne mijenja stanje niti jedne varijable izvan okvira te funkcije, te ćemo onda lakše shvatiti radi li funkcija ono što je potrebno. Za takve funkcije kažemo da nemaju nuspojave jer ne mijenjaju stanje programa.

### 3.1.2. Nepromjenjivost varijabli

Nepromjenjiva varijabla je varijabla kojoj se vrijednost ne može mijenjati tijekom izvođenja programa. Ako se vrijednost te varijable pokuša promijeniti u drugu vrijednost interpreter će javiti grešku. To je korisno jer će se greška pojaviti već pri samom mijenjanju vrijednosti varijable, a ne tamo gdje je promijenjena vrijednost već utjecala na drugi dio programa. Koristeći ovu grešku temeljni uzrok neke druge pogreške se može pronaći ranije.

Python nudi neke nepromjenjive vrste podataka, jedna od njih je uređena lista. Usporedimo ju s listom, koja je izmjenjivi tip podataka.

```
lista = [1, 2.3,  [4, 5], 'Hello world!']
uredenaLista = (1, 2.3,  [4, 5], 'Hello world!')
```

```
print(lista[1]) #  2.3
print(uredenaLista[1]) #  2.3

lista[0] = 6
uredenaLista[0] = 6
```

Ovdje se pojavljuje pogreška “TypeError: 'tuple' object does not support item assignment” jer uređena lista ne podržava mijenjanje vrijednosti elemenata.

Postoji zanimljiva situacija gdje se uređena lista može činiti izmjenjivom. Na primjer, ako želimo promijeniti listu koja se nalazi u uređenoj listi iz [4, 5] u [4, 5, 6], možemo učiniti sljedeće:

```
uredenaLista[2].append(6)
print(uredenaLista[2]) #  [4, 5, 6]
```

Ovo funkcionira jer je lista promjenjivi objekt, no ako pokušamo listu vratiti natrag u [4, 5] dobiti ćemo grešku.

```
uredenaLista[2] = [4, 5]
```

Ovdje očekivano dobijemo istu grešku kao i prije. U uređenoj listi možemo promijeniti sadržaj promjenjivog objekta koji je u toj uređenoj listi, ali ne možemo promijeniti referencu na taj promjenjivi objekt koji je pohranjen u memoriji.

### 3.1.3. Funkcije višeg reda

Funkcije višeg reda su funkcije koje ili prihvaćaju funkciju kao argument, ili vraćaju funkciju za daljnju obradu.

Slijedi primjer kako se oba slučaja mogu jednostavno napisati u Pythonu.

Razmotrimo funkciju kojom se poruka ispisuje određeni broj puta:

```
def ispisi(poruka, n):
    for i in range(n):
        print(poruka)

ispisi('Pozdrav!', 10)
```

Ako bismo htjeli tu poruku zapisati u datoteku kao grešku 10 puta ili ju samo ispisati 10 puta, umjesto da napišemo dvije različite funkcije gdje svaka ima svoju petlju, možemo napisati jednu funkciju višeg reda koja prihvaća funkciju kao argument:

```
def ispisi(poruka, n, akcija):
    for i in range(n):
        akcija(poruka)

ispisi('Pozdrav!', 10, print)
```

Moramo uključiti modul *logging* koji sadrži funkciju za spremanje grešaka *error*

```
import logging
ispisi('Pozdrav!', 10, logging.error)
```

Ako želimo stvoriti funkcije koje elemente neke liste povećavaju za 2, 5 i 10, možemo napisati tri vrlo slične funkcije ili napisati jednu funkciju višeg reda koja će ih stvoriti za nas.

Počnimo s prvim slučajem:

```
def dodaj2(brojevi):
    novi = []
    for n in brojevi:
        novi.append(n + 2)
    return novi

print(dodaj2([20, 60])) #  [22, 62]
```

Iako je trivijalno napisati funkcije dodaj5 i dodaj10, očito je da bi one radile na isti način. Uzimale bi elemente iz liste, dodavale neki inkrement i vratile rezultat. Dakle, umjesto stvaranja različitih funkcija dodavanja inkrementa, možemo napisati jednu funkciju višeg reda:

```
def dodaj(inkrement):
    def dodajInkrement(brojevi):
        novi = []
        for n in brojevi:
            novi.append(n + inkrement)
        return novi
    return dodajInkrement

dodaj5 = dodaj(5)
print(dodaj5([20, 60])) #  [25, 65]
dodaj10 = dodaj(10)
print(dodaj10([20, 60])) #  [30, 70]
```

Funkcije višeg reda nam omogućuju veću fleksibilnost našeg kôda. Apstraktno definirajući koje se funkcije primjenjuju ili vraćaju povisuje iskoristivost našeg kôda. U prijašnjem primjeru, koristeći funkciju dodaj, lako možemo napraviti novu funkciju koja pribraja elementima liste bilo koji drugi broj. Python pruža neke već ugrađene funkcije višeg reda koje mogu znatno olakšati rad s listama, a to su funkcije *map* i *filter*.

### 3.1.4. Lambda izrazi

Lambda izrazi, poznati i kao anonimne funkcije nam omogućuju stvaranje i korištenje funkcije u jednom retku.

Slijedi primjer kako napisati funkciju višeg reda koja vraća funkciju koja množi broj s unaprijed određenom vrijednosti:

```
def produkt(množitelj):
    return lambda x: x * množitelj

mnozi9 = produkt(9)
print(mnozi9(5)) #  45
```

Lambda izrazi su korisni kada nam treba kratka i jednostavna funkcija. Stvaranjem nove funkcije u Pythonu koristi se ključna riječ *def*, te joj se dodjeljuje jedinstveno ime. Korištenjem lambda izraza možemo brže definirati funkcije i ne moramo se brinuti oko dodjeljivanja imena funkciji. Najčešće se koriste u kombinaciji s *map*, *filter* i funkcijama sortiranja [12].

### 3.1.5. Ugrađene funkcije višeg reda

Python je iz funkcijskih programskih jezika implementirao neke od najčešće korištenih funkcija višeg reda što znatno olakšava obradu iterativnih objekata poput lista, uređenih lista, asocijativnih listi i skupova. Radi boljeg iskorištavanja računalne memorije, ove funkcije vraćaju iterator umjesto nove liste.

#### 3.1.5.1. Map

Funkcija *map* nam omogućuje primjenu neke funkcije na svaki element bilo kojeg iterativnog objekta. Na primjer, ako imamo listu imena i želimo pozdraviti svaku osobu iz te liste, možemo napisati sljedeći kôd:

```
imena = ['Mateo', 'Ivan', 'Luka']
pozdravi = map(lambda x: 'Hello, ' + x, imena)
```

Moramo paziti da ne pokušamo ispisati pozdravi kao listu jer nam funkcija *map* ne vraća listu, već iterator. Tako da moramo prolaziti iteratorom da bi došli do vrijednosti.

```
for pozdrav in pozdravi:
    print(pozdrav)
```

#### 3.1.5.2. Filter

Funkcija *filter* testira svaki element u iterativnom objektu s funkcijom koja vraća istinu ili laž, zadržavajući samo one za koje ta funkcija vraća istinu. Slijedi primjer kako možemo od liste brojeva filtrirati samo one koji su djeljivi s 5.

```
brojevi = [20, 47, 2, 15]
djeljivi5 = filter(lambda x: x % 5 == 0, brojevi)
```

Iteratore možemo promijeniti u liste tako da koristimo ugrađenu funkciju *list*.

```
print(list(djeljivi5)) #  [20, 15]
```

#### 3.1.5.3. Kombiniranje funkcija map i filter

Funkcije *map* i *filter* vraćaju iterator, te oboje prihvaćaju iterativne objekte. Zbog toga ih možemo koristiti zajedno za neke ekspresivne manipulacije podacima. Kao primjer možemo uzeti listu brojeva koju ćemo filtrirati tako da uzmemo brojeve djeljive s 5, između 1 i 20, te ih onda kubiramo.

```
brojevi = map(lambda x: x ** 3, filter(lambda x: x % 5 == 0,
range(1, 21)))
print(list(brojevi)) #  [125, 1000, 3375, 8000]
```

Izraz u listi brojevi može se rastaviti na 3 dijela:

- range(1, 21) je iterativni objekt koji predstavlja brojeve od 1 do 20.
- filter(lambda x: x % 3 == 0, range(1, 21)) vraća iterator za niz brojeva 5, 10, 15 i 20.
- te mapiranjem funkcije kubiranja na taj niz dobijemo iterator za novi niz brojeva 125, 1000, 3375 i 8000.

### 3.1.6. Sažet zapis listi

Jedan od jedinstvenih koncepta funkcijskog programskog jezika je sažet zapis listi (*engl. list comprehension*), koji opisuje listu tako da se koriste elementi neke druge liste. Iz jedne se liste uzimaju elementi, te se testiraju i/ili transformiraju tako da postanu elementi nove liste. Ova notacija je vrlo korisna pri pisanju programa koji rade s obradom listi [13, p. 79].

Slijedi jednostavni primjer kako se koristi sažet zapis listi da bi definirali novu listu.

```
xs = [1, 2, 3, 4]
print([2*x for x in xs]) #  [2, 4, 6, 8]
```

Ovaj primjer ispisuje listu [2, 4, 6, 8] koju smo definirali tako da se uzme svaki element x iz liste xs i onda se pomnoži s dva. Taj izraz možemo pročitati kao "Uzmi sve 2*x gdje je x iz liste xs".

Nakon izraza `for x in xs` možemo dodavati uvjete.

Ako želimo filtrirati elemente neke liste tako da dobijemo sve parne elemente, onda ćemo napraviti funkciju koja će nam vraćati istinu ili laž, ovisno je li broj paran ili nije.

```
def paran(x):
    if x % 2 == 0:
        return True
    else:
        return False
```

Koristeći tu funkciju kao uvjet u sažetom zapisu liste ćemo dobiti listu koja sadrži samo parne elemente neke liste.

```
xs = [1, 2, 3, 4]
print([x for x in xs if paran(x)]) #  [2, 4]
```

Ovaj izraz možemo pročitati kao "Uzmi sve x gdje je x iz liste xs ako je x paran".

Lijevo od izraza `for x in xs` možemo staviti bilo kakvu operaciju nad elementima liste. Za sljedeći primjer ćemo koristiti funkciju paran iz prijašnjeg primjera.

```
parovi = [(1, 2), (2, 3), (3, 4), (4, 5)]
print([x + y for (x, y) in parovi if paran(x) and y > 3]) #  [9]
```

Ovaj izraz ispisuje listu [9] jer je (4, 5) jedini par koji za x ima paran broj, a da je y veći od 3. Tako da jedino taj par zadovoljava sve uvjete te se jedino na njemu izvrši operacija koja je definirana lijevo od izraza `for x in xs`, a to je u ovom slučaju x + y, što kao rezultat vraća 9.

Sažet zapis listi može biti dio neke veće funkcije. Ako, na primjer, želimo provjeriti jesu li svi elementi liste neparni možemo napisati novu funkciju parni

```
def neparni(xs):
    if xs == [x for x in xs if x % 2 != 0]:
        return True
    else:
        return False

print(neparni([1, 3, 5])) #  True
```

### 3.1.7. Rekurzija

Ako imamo neki složeni problem, možemo ga probati raščlaniti na manje probleme te onda probati naći rješenje jednog od tih manjih problema. Tada ćemo možda moći nadograditi to rješenje i dobiti rješenje za cijeli problem. To je ideja rekurzije. Rekurzivni algoritmi raščlanjuju problem na manje dijelove na koje se ili već zna odgovor, ili ih se može riješiti primjenom istog algoritma na svaki manji dio problema, te se zatim svi rezultati manjih problema kombiniraju da bi došli do konačnog rezultata [14].

Općenito, rekurzija je postupak definiranja nečega pomoću samoga sebe. Ona uključuje uporabu procedure, potprograma, funkcije ili algoritma koji poziva sam sebe te ima uvjet prekida. Rekurzija se obrađuje te poziva sama sebe sve dok uvjet prekida nije ispunjen. U tom trenutku dobije neki rezultat i ostala ponavljanja se obrađuju od posljednje pozvanog prema prvom pozvanom.

Fibonaccijevi brojevi su definirani nizom definiranog sljedećom rekurzivnom relacijom:

$$F(n) := \begin{cases} 0 & ako\ je\ n = 0; \\ 1 & ako\ je\ n = 1; \\ F_{n-1} + F_{n-2} & ako\ je\ n > 1. \end{cases}$$

Slijedi primjer kako se može koristiti koncept rekurzije za izračunavanje fibonaccijevog broja.

```
def fibonacci(n):
    if n == 0:
        return 0
    elif n == 1:
        return 1
    else:
        return fibonacci(n-1) + fibonacci(n-2)
```

## 3.2. Tipovi podataka u Pythonu

Python je programski jezik koji ima dinamičku (*engl. dynamic typing*) i jaku (*engl. strong typing*) tipizaciju. Dinamička tipizacija znači da Pythonov interpreter provjerava tipove varijabli pri izvršavanju kôda što omogućuje mijenjanje tipa varijable pri izvođenju programa. Ukoliko dođe do toga da se neka operacija primjenjuje na tipove za koje nije specifirana interpreter će javiti grešku. Prednost ovakvog sustava tipizacije je ta što programer ne mora definirati tip varijable, ali je zato otežan pronalazak pogrešaka povezanih s tipovima varijabli [15].

Jezici s jakom tipizacijom ne dozvoljavaju implicitne konverzije tipova. Jedan tip podataka se ne može tretirati kao drugi tip podataka. Prednost jake tipizacije je ta da se logičke pogreške kod kojih smo koristili ili proslijedili varijablu krivog tipa gdje nismo trebali se lakše pronađu i otklone.

Haskell je programski jezik koji ima statičku (*engl. static typing*) i jaku tipizaciju, te automatsko zaključivanje tipa varijable (*engl. type inference*). Statička tipizacija znači da svi izrazi, uključujući i funkcije, imaju tip koji se mora ili navesti ili će se automatski zaključiti prije ili tijekom izvršavanja kôda.

Zaključivanje tipa varijable je postupak određivanja tipova izraza na temelju poznate vrste simbola koji se pojavljuju u njima. Razlika između zaključivanja tipa varijable i provjere tipa pri izvršavanju programa je malena. Algoritam za provjeru tipa varijable pri izvršavanju

programa provjerava slažu li se tipovi koje je programer napisao s tipovima koji su definirani u jeziku, a kod zaključivanja tipa varijable ideja je da neke informacije nisu navedene, te da se za određivanje tipa koristi neki oblik logičkog zaključivanja. Na primjer, identifikatori tipova u Haskellu obično nisu deklarirani kao određeni tipovi podataka, već sama tipizacija zaključuje tip identifikatora i izraza koji ih sadrže pomoću operacija koje se koriste u tom izrazu [16, p. 122].

## 3.3. Prednosti funkcijskog programiranja u Pythonu

Čistim funkcijama je lakše shvatiti svrhu jer one ne mogu imati nikakav utjecaj na kôd izvan okvira funkcije, ne mogu imati skrivene ulaze ili mijenjati stanje globalnih varijabli. Zbog toga je sigurno da opisi čistih funkcija govore točno što primaju kao parametre i što vraćaju kao izlaz funkcije. Zbog toga programer troši manje vremena na razumijevanje kôda, može pamtiti manje detalja o svakoj funkciji te se više fokusirati na korištenje tih funkcija. Također programeru pomažu imati na umu veći dio logike programa. Budući da čiste funkcije ovise samo o njihovim ulaznim parametrima da bi vratile njihov izlaz, lakše je uklanjanje pogrešaka. Naravno da je i dalje moguće pogriješiti pri pisanju čiste funkcije, ali jednom kada se ispiše pogreška, slijede se vrijednosti dok ne shvatimo što je uzrok pogreške. Budući da su funkcije čiste, ne moramo se brinuti o onome što se događa u ostatku programa, već je samo dovoljno znati koji su ulazni parametri proslijeđeni čistoj funkciji uzrokovali grešku [17].

Neke značajke koje kôd mogu učiniti čitljivijim i konciznijim su mogućnost prosljeđivanja funkcija u druge funkcije pomoću funkcija višeg reda, mogućnost pisanja kratkih i anonimnih funkcija, te pisanje velikih hijerarhija klasa nije potrebno. Pisanjem čitljivijeg kôda možemo biti sigurniji da ćemo ga, ako ga budemo čitali nakon što je prošlo neko vrijeme, lakše pročitati i razumjeti.

## 3.4. Nedostaci funkcijskog programiranja u Pythonu

Kod čisto funkcijskih programskih jezika posljedica korištenja čistih funkcija i nepromjenjivih varijabli je ta da se mora koristiti rekurzija jer u takvim jezicima ne postoje petlje. Jedini način iteracije po elementima neke liste je rekurzijom, koja nije baš intuitivna. Zato postoje neki načini kako zamijeniti korištenje rekurzije kao što je funkcija *map* kojom možemo primijeniti funkciju nad elementima liste i time zamijeniti neku rekurziju. Python nije čisto funkcijski jezik što znači da taj problem nije toliko izražen jer on podržava definiranje petlji [18].

Mijenjanje vrijednosti nepromjenjive varijable je nemoguće, ali zato možemo definirati novu varijablu i zatim u nju kopirati podatke iz nepromjenjive varijable mijenjajući samo ono što želimo. Svaki put stvaramo novu varijablu koja mora imati novo ime, te ju u daljnjem izvođenju programa koristimo umjesto stare varijable. Takva metoda zaobilaženja nepromjenjivosti varijabli nije komplicirana, ali rezultira nečitljivijim kôdom. U Pythonu postoje promjenjive varijable kojima možemo lako zamijeniti nepromjenjive varijable.

Python nije čisto funkcijski jezik pa se sve mane čisto funkcijskog programiranja mogu zaobići korištenjem drugih paradigmama koje Python podržava.

# 4. Primjena koncepata funkcijskog programiranja u primjeru razvoja igre križić-kružić

Koncepti funkcijskog programiranja će se prikazati na igri križić-kružić. Križić-kružić je jednostavna igra za dva igrača koja se igra na praznoj ploči koja se sastoji od tri redaka i stupaca. Prvi igrač crta križiće, a drugi kružiće. Igrači naizmjenično odabiru jedno od praznih polja i unutar njih crtaju svoj znak. Igrač pobjeđuje kada nacrta tri svoja znaka uzastopno u nekom retku, stupcu, glavnoj ili sporednoj dijagonali. Ako to ne uspije ostvariti niti jedan igrač, igra završava neriješeno [19].

## 4.1. Strategija za optimalno igranje

Igrač može igrati optimalno koristeći sljedeću strategiju. Ako igrač ima dva znaka zaredom i treće mjesto je slobodno, crta treći i pobjeđuje. U slučaju da to ne može napraviti, provjerava ima li protivnik dva znaka za redom. Ako ima, crta svoj znak na trećem mjestu da bi spriječio pobjedu protivnika. Zatim mu je cilj stvoriti situaciju u kojoj u sljedećem potezu ima dva moguća poteza kojim može pobijediti. Ako to ne može, mora paziti da spriječi protivnika da se račva tako da ili stvori dva znaka zaredom da protivnik mora zaustaviti njegovu pobjedu ili nacrta znak na polje koje protivnik treba za račvanje. Ako nema opasnosti od protivnikovog račvanja, igra sredinu. Ako to ne može, igra suprotni kut od protivnikovog prijašnjeg poteza. Ako je to polje zauzeto, igra proizvoljan kut. Ako su svi kutovi zauzeti, igra proizvoljno polje koje nije ni kut ni sredina [20].

## 4.2. Naprednija verzija križić-kružića

Križić-kružić se obično igra na ploči od tri retka i stupaca. Na takvoj ploči se može lako i brzo naučiti igrati optimalno, pa je rezultat igre vrlo predvidljiv. Ako oba igrača nauče igrati optimalno, igra će uvijek završiti neriješeno. Zato se veličina ploče može povećati na četiri, pet ili koliko god broj redaka i stupaca želimo. Povećanjem veličine ploče moramo i povećati i koliko uzastopnih znakova igrač mora nacrtati da bi pobijedio. Za ploču od četiri redaka i stupaca, optimalno je da igračima trebaju četiri uzastopna znaka za pobjedu, a za sve ploče veće od te možemo staviti da im trebaju pet uzastopnih znakova za pobjedu. Povećanjem ploče povećava se i kompleksnost igre.

## 4.3. Čiste funkcije

Koncept čistih funkcija primijenjen je na sve funkcije tako da se sve varijable potrebne za rad neke funkcije uvijek prosljeđuju kao parametri. Time se pri svakom pozivu funkcije vrijednost varijabli kopira u novu varijablu koja se može koristiti unutar granica funkcije.

Kôd pisan deklarativno, prije primjene koncepta:

```
def displayText(message):
    text = font.render(message, True, darkGray)
    window.blit(text,    text.get_rect(center=(displaySize/2+3,
displaySize/2+3)))
    text = font.render(message, True, white)
    window.blit(text,      text.get_rect(center=(displaySize/2,
displaySize/2)))
    pygame.display.update()
```

Kôd nakon primjene koncepta:

```
def displayText(message, displaySize, window, font):
    white = (255, 255, 255)
    darkGray = (39, 44, 48)
    text = font.render(message, True, darkGray)
    window.blit(text,    text.get_rect(center=(displaySize/2+3,
displaySize/2+3)))
    text = font.render(message, True, white)
    window.blit(text,      text.get_rect(center=(displaySize/2,
displaySize/2)))
    pygame.display.update()
```

Glavna svrha koncepta čistih funkcija je ta da smo sigurni da se vrijednosti neke globalne varijable neće promijeniti nakon poziva neke funkcije. Također, sve varijable potrebne za izvršavanje funkcije su na jednom mjestu što olakšava čitanje kôda. Dodavanjem svih varijabli u parametre funkcije složenost kôda se nije promijenila.

## 4.4. Funkcije višeg reda i nepromjenjivost varijabli

Koncept funkcije višeg reda je korišten da bi se napisala funkcija koja će vratiti funkciju za računanje broja uzastopnih znakova igrača koji je trenutno na potezu. Funkciji se kao parametar prosljeđuje koji smjer provjerava: red, stupac, glavna ili sporedna dijagonala. Ovisno o tom parametru ona stvara funkciju koja korištenjem ugniježdene *for* petlje prolazi po ploči i broji koliko igrač ima uzastopnih pojavljivanja njegovog znaka.

Ta funkcija višeg reda se koristi za provjeravanje postoji li pobjednik nakon svakog odigranog poteza. Ona generira sve potrebne funkcije koje se koriste da bi znali broj uzastopnih znakova igrača, te se onda provjerava je li taj broj dovoljan za pobjedu.

Koncept nepromjenjivosti varijabli se koristio pri stvaranju funkcija tako da se ovisno o smjeru provjeravanja stvori uređena lista koja sadrži koeficijente kojima se množe varijable petlji da bi se dobio smjer kojim se mora provjeravati. Ne želimo da se ti koeficijenti ikada promijene tijekom izvođenja programa. Definiranjem te varijable uređenom listom se osigurala nepromjenjivost te varijable.

## 4.5. Map i lambda izrazi

Funkcija *map* se koristila u kombinaciji s lambda izrazom da bi se lakše dobio broj slobodnih polja na ploči. Broj slobodnih polja se koristi za provjeru je li igra završila neriješeno.

Kôd prije primjene koncepta:

```
emptySpaces = 0
    for i in range(0, boardSize):
        for j in range(0, boardSize):
            if board[i][j] == ' ':
                emptySpaces += 1
```

Kôd nakon primjene funkcije *map* i lambda izraza:

```
emptySpaces = sum(list(map(lambda x: x.count(' '), board)))
```

Mapiranjem lambda x: x.count(‘ ’) izraza na ploču, lambda izraz kao parametar x dobiva listu koja predstavlja jedan redak ploče. Koristeći ugrađenu funkciju *count*, lambda izraz prebroji koliko ta lista ima elemenata koji su prazan znak. Korištenjem lambda izraza nije potrebno stvarati novu funkciju koja bi služila samo za ovu upotrebu. Funkcija *map* vraća iterator koji moramo pretvoriti u listu koristeći ugrađenu funkciju *list*. Ta lista sadrži brojeve praznih polja po redovima ploče, te da dobijemo broj praznih polja za cijelu ploču još samo moramo sumirati tu listu ugrađenom funkcijom *sum*.

## 4.6. Sažet zapis listi

Koncept sažetog zapisa listi primijenjen je kod stvaranja i inicijalizacije liste listi za ploču igre. Ta lista listi predstavlja dvodimenzionalnu ploču.

Kôd prije primjene koncepta:

```
boardSize = 3
board = []
for _ in range(0, boardSize):
    temp = []
    for _ in range(0, boardSize):
        temp.append(' ')
    board.append(temp)
```

Kôd nakon primjene koncepta:

```
boardSize = 3
board = [[' ' for _ in range(0, boardSize)] for _ in range(0,
boardSize)]
```

Ovaj koncept je jako koristan kod obrade listi jer dosta smanjuje broj linija kôda. Umjesto pisanja dvije ugniježdene *for* petlje i korištenja pomoćne varijable možemo svu funkcionalnost napisati u jednoj liniji kôda. Samim time je i kôd postao puno čitljiviji. Kod proceduralnog pristupa sama funkcionalnost kôda nije toliko očita.

## 4.7. Filter

Za funkciju *filter* nisam našao primjenu u ovoj igri, ali se također mogla koristiti za brojanje slobodnih polja na ploči.

Slijedi kako bi kôd izgledao primjenom funkcije *filter* u kombinaciji s sažetim zapisom liste i lambda izrazom.

```
emptySpaces = len(list(filter(lambda x: True if x == ' ' else
False, [y for row in board for y in row])))
```

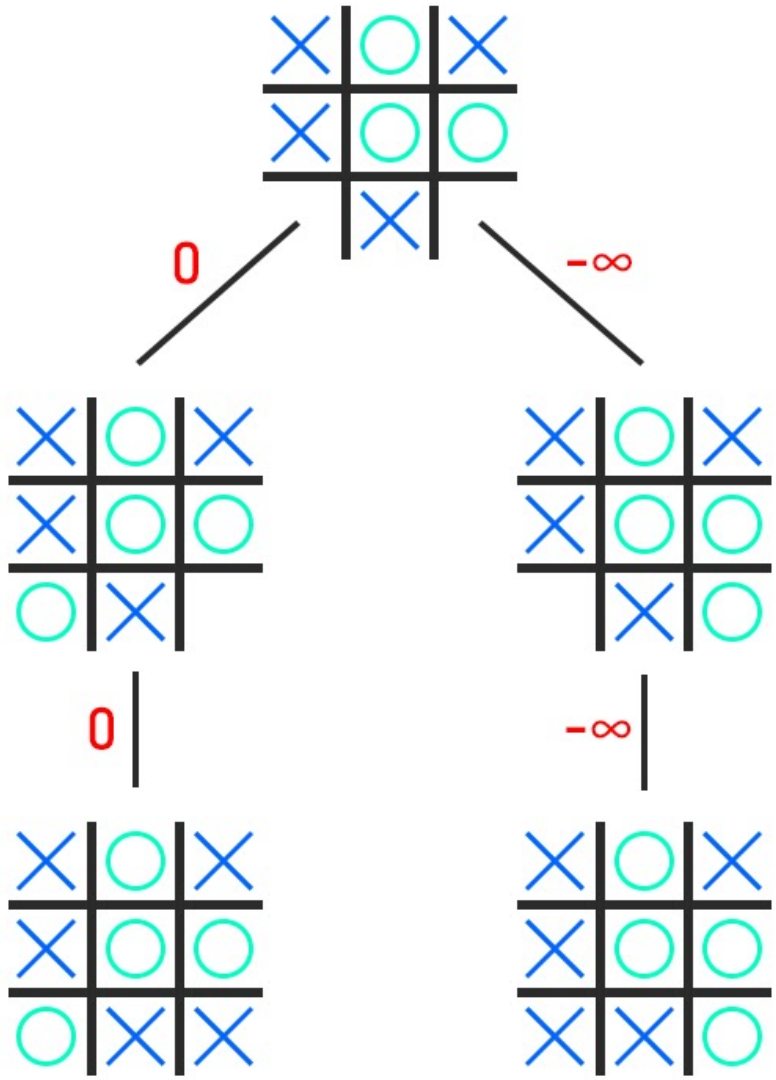
Lista board je lista koja predstavlja trenutno stanje ploče. U njoj se nalaze liste koje predstavljaju retke ploče. Prvo moramo dobiti jednodimenzionalnu listu koja sadrži stanje svih polja na ploči. Sažeti zapis liste `[y for row in board for y in row]` prvo uzima liste iz liste board, te onda uzima elemente tih listi. Time smo dobili listu koja sadrži svaki element ploče. Nju možemo filtrirati lambda izrazom koji vrati istinu ako je taj element prazno polje, a inače vrati laž. Funkcija *filter* vraća iterator koji moramo pretvoriti u listu ugrađenom funkcijom *list*. U toj listi se nalaze samo prazna polja. Toj listi ugrađenom funkcijom *len* izračunamo duljinu i time smo dobili broj praznih polja na ploči.

## 4.8. Minimax algoritam i rekurzija

Minimax je rekurzivni algoritam u teoriji odlučivanja koji služi za traženje optimalnog poteza kojeg igrač može odigrati. Algoritam uzima u obzir sve moguće poteze protivnika te pretpostavlja da protivnik igra optimalno. Cilj mu je minimizirati maksimalni mogući gubitak. Igrač pokušava postići najviši mogući rezultat, a njegov protivnik pokušava postići najniži mogući rezultat. Često se koristi u igrama za dva igrača koje se temelje na potezima kao što su križić-kružić, šah, i sl. [21].

Odabirom jednog od mogućih poteza igrača algoritam ocjenjuje svaki mogući potez protivnika i time stvara stablo svih mogućih ishoda igre. Stvaranje stabla se može zaustaviti do željene dubine, te se koristeći heuristiku može dobiti ocjena trenutnog stanja ploče. Svakom se stanju ploče može izračunati ocjena. Ako u trenutnom stanju igra igrač rezultat ploče će biti pozitivna vrijednost, u suprotnom će to biti negativna vrijednost. Ocjene ploče se izračunavaju pomoću funkcije heuristike koja je jedinstvena za svaku vrstu igre.

Količina posla koju minimax pretraživanjem jednog poteza generira raste eksponencijalno što je dubina pretraživanja veća. Da bi se smanjilo vrijeme izvršavanja algoritma, pretraživanje se mora ograničiti. Inače bi se gubilo vrijeme na pretraživanje poteza koji su očito loš odabir za igrača. Jedan od učinkovitih pristupa smanjenju broja grana koje se moraju pretražiti je implementacija alfa-beta rezanja. Alfa-beta rezanje prati najbolje poteze za svakog igrača pri pretraživanju stabla. U slučaju da je ocjena trenutnog poteza lošija od ocjene trenutno najboljeg poteza, prekinuti će se pretraživanje te grane stabla [22].




*Slika 1. Primjer izvršavanja minimax algoritma*

U ovom primjeru izvršavanja minimax algoritma u trenutnom stanju ploče postoje dva moguća poteza. Algoritam odabire prvi potez i stvara novu granu. Sada je protivnik na redu, ali on ima jedan mogući potez koji rezultira tako da igra završi neriješeno. Takav slučaj se ocjenjuje neutralnom ocjenom, koja je u ovom slučaju nula. Ocjena se prosljeđuje nazad kroz granu i sada znamo da ako igramo taj potez da će igra završiti neriješeno. Ostao je još drugi mogući potez za kojeg se stvara nova grana. Opet protivnik ima samo jedan mogući potez, ali sada taj potez rezultira njegovom pobjedom. Pošto je on minimizirajući igrač, tu ploču ocjenjujemo s najnižom ocjenom. Ta ocjena se također prosljeđuje nazad kroz granu i sada znamo da ako igramo taj potez da će protivnik pobijediti. Sada imamo odabir između dvije ocjene, nula i minus beskonačnost. Igrač je maksimizirajući igrač, on odabire najvišu ocjenu od te dvije što znači da će odabrati potez koji rezultira tako da igra završi neriješeno.

## 4.9. Usporedba kôda

Program pisan proceduralnom paradigmom ima 367 linija kôda, dok ih program pisan funkcijskom paradigmom ima 373.

| Funkcija | Opis funkcije | Broj linija kôda (proceduralna paradigma) | Broj linija kôda (funkcijska paradigma) |
|---|---|---|---|
| drawX | Crta križić na polje koje mu se proslijedi kao parametar | 12 | 17 |
| drawO | Crta kružić na polje koje mu se proslijedi kao parametar | 5 | 9 |
| displayText | Ispisuje tekst koji mu se proslijedi kao parametar | 7 | 9 |
| resetGame | Resetira stanje ploče i ponovno nacrta praznu ploču | 15 | 10 |
| getLimit | Vraća broj do kojeg funkcija getInARow smije prolaziti po ploči ovisno o smjeru koji se provjerava | 0 | 10 |
| getInARow | Vraća novu funkciju koja ovisno o smjeru koji se provjerava prolazi po poljima ploče i broji koliko igrač ima uzastopnih ponavljanja njegovog znaka | 0 | 44 |
| checkWinner | Provjerava je li broj uzastopnih ponavljanja igračevog znaka dovoljan za pobjedu. Ako je, vraća pobjednika, ako su sva polja popunjena, vraća neriješeno inače vrati 0 | 76 | 28 |
| heuristicEvaluate | Ocjenjuje stanje ploče koja se proslijedi kao parametar tako da broji koliko koji igrač ima znakova za redom | 75 | 75 |
| minimax | Minimax algoritam koji rekurzivno ocjenjuje ploču odabirući moguće poteze ovisno o trenutnom igraču i time gradi stablo igre | 47 | 47 |

*Tablica 1. Usporedba broja linija kôda igre križić-kružić*

Većina funkcija su ili ostale nepromijenjene ili su minimalno promijenjene. Kod funkcija drawX, drawO i displayText u funkcijskoj paradigmi jedina promjena je ta da su varijable koje funkcija koristi definirane u njima, i time su postale čiste funkcije. Promjene su minimalne, nema velike razlike u čitljivosti kôda. Kod funkcije resetGame korišten je sažet zapis liste i time se malo poboljšala čitljivost funkcije. Ondje gdje se vidi najveća razlika je u funkciji checkWinner koja se u funkcijskoj paradigmi raščlanila na tri funkcije. Postala je puno čitljivija

jer se koristi funkcija višeg reda getInARow kojoj samo proslijedimo koji smjer provjeravamo. Ona će nam vratiti funkciju kojom možemo provjeriti koliko igrač ima uzastopnih ponavljanja njegovog znaka na ploči.

U proceduralnoj paradigmi funkcija checkWinner nije toliko čitljiv, te se isti kôd ponavlja za svaki smjer provjeravanja. Koncept funkcije višeg reda je puno pomogao razumijevanju i čitljivosti kôda. Ostale funkcije su ostale identične.

# 5. Primjena koncepata funkcijskog programiranja u primjeru razvoja igre vješala

Igra vješala je igra za dva ili više igrača. Jedan igrač smisli neku riječ, a drugi igrači ju pokušavaju pogoditi tako da predlažu slova.

Riječ koja se pogađa predstavljena je nizom crtica od kojih svaka predstavlja jedno slovo riječi. U većini verzija ove igre, vlastite imenice, kao što su imena, mjesta i marke, nisu dopuštene za odabir kao riječ za pogađanje. Žargonske riječi, koje se ponekad nazivaju neformalnim ili skraćenim riječima, također nisu dopuštene. Ako igrač koji pogađa predloži slovo koje se pojavljuje u riječi, prvi će ga igrač zapisati u sve njegove ispravne položaje. Ako se predloženo slovo ne pojavljuje u riječi, prvi igrač crta jedan dio čovječuljka kao način brojanja krivih pogađanja.

Igrač koji pogađa riječ može u bilo kojem trenutku pokušati pogoditi cijelu riječ. Ako je riječ točna, igra je gotova i pogađač pobjeđuje. Inače, prvi igrač može odlučiti kazniti pogađača dodavanjem dijelova čovječuljka. Ako pogađač napravi dovoljno netočnih pogađanja i time se dovrši crtanje čovječuljka, igra je također gotova, ovoga puta s pobjedom prvog igrača. Međutim, pogađač može pobijediti i pogađanjem svih slova koja se pojavljuju u riječi, čime se dovršava riječ, prije nego što se crtanje čovječuljka dovrši.

## 5.1. Strategija za optimalno igranje

U hrvatskome jeziku prvih 12 najfrekventnijih slova su a, i, o, e, n, s, r, j, t, u, d i k [23]. Poredani od najfrekventnijeg prema manje frekventnom slovu. Korištenjem tih slova pogađač si može povećati šanse da pogodi slovo riječi. Ali ako igrač koji smišlja riječ poznaje tu strategiju, može smisliti riječ koja ne sadrži ta slova i time će ova strategija za pogađača biti loša.

Slijedi primjer odvijanja igre u kojoj imamo jednog pogađača te ako taj pogađač koristi strategiju najfrekventnijih slova hrvatskoga jezika. Riječ je vješalo.

| r.br. poteza | Vješalo | Riječ | Pokušaj | Promašaji |
|---|---|---|---|---|
| 1. | | _ _ _ _ _ _ _ | a | |
| 2. | | _ _ _ _ a _ _ | i | |
| 3. | | _ _ _ _ a _ _ | o | i |
| 4. | | _ _ _ _ a _ o | e | i |
| 5. | | _ _ e _ a _ o | n | i |

| | | | | |
|---|---|---|---|---|
| 6. | | _ _ e _ a _ o | s | i, n |
| 7. | | _ _ e _ a _ o | r | i, n, s |
| 8. | | _ _ e _ a _ o | j | i, n, s, r |
| 9. | | _ j e _ a _ o | t | i, n, s, r |
| 10. | | _ j e _ a _ o | u | i, n, s, r, t |
| 11. | | _ j e _ a _ o | | i, n, s, r, t, u |

*Tablica 2. Primjer odvijanja igre vješala*

Pogađač je izgubio igru jer se čovječuljak završio crtati prije nego su sva slova pogođena.

Druga strategija koja se može koristiti je prvo pogađati samoglasnike, u hrvatskome jeziku to su a, e, i, o i u. Skoro svaka hrvatska riječ sadrži samoglasnik, pa si koristeći ovu strategiju pogađač također može povećati šanse za pogađanjem slova.

## 5.2. Čiste funkcije i nepromjenjivost varijabli

Koncept čistih funkcija, kao i u primjeni kod igre križić-kružić, primijenjen je na sve funkcije prosljeđivanjem svih potrebnih varijabli kao parametre funkcije. Time se pri svakom pozivu funkcije vrijednost potrebnih varijabli kopira u nove varijable koje funkcija koristi za obradu.

Primjenom ovog koncepta složenost kôda se nije promijenila, ali je postao malo čitljiviji.

Koncept nepromjenjivih varijabli se koristio kod definicije boja.

## 5.4. Map i lambda izrazi

Funkcija *map* i lambda izraz korišteni su kod dobivanja trenutnog stanja riječi koja se pogađa. Lambda izraz kao ulaz prima pojedino slovo riječi te indeks i, a kao izlaz vrati ili to slovo ili donju crtu ovisno je li to slovo otvoreno ili nije. Mapirajući taj lambda izraz na samu riječ koja se pogađa i pomoćni iterator koji nam služi za prolaženje po riječi, funkcija *map* vraća iterator u kojemu se nalazi trenutno stanje riječi. Taj iterator moramo pretvoriti u listu ugrađenom funkcijom *list* te tu listu spojiti s praznim znakom, koristeći ugrađenu funkciju *join*, da bi dobili tekst koji nam treba za prikaz trenutnog stanja riječi.

Kôd prije primjene koncepata funkcije *map* i lambda izraza:

```
text = ''
for i in range(0, length):
    if opened[i]:
        text += word[i] + (' ' if i < length - 1 else '')
    else:
        text += '_' + (' ' if i < length - 1 else '')
```

Kôd nakon primjene koncepata funkcije *map* i lambda izraza:

```
text = ' '.join(list(map(lambda x, i: x if opened[i] else '_',
word, range(0, length))))
```

Primjenom ovih koncepata, složenost je ostala ista dok smo malo smanjili broj linija kôda i time povećali njegovu čitljivost.

## 5.5. Filter

Funkcija *filter* se koristila za filtriranje riječi iz rječnika. U rječniku se mogu naći riječi koje sadrže povlaku, točku i slične znakove koje igra ignorira ako se unese taj znak. Filtriranjem rječnika s lambda izrazom koji provjerava je li svaki znak te riječi iz abecede, takve riječi možemo izbaciti iz rječnika. Nakon filtriranja rječnik sadrži 61457 riječi od kojih se nasumice odabere jedna koja postane riječ koja se pogađa.

Kôd prije primjene koncepata funkcije *filter* i lambda izraza:

```
words = {"otorinolaringologija"}
file = open("hrvatski-rjecnik.txt", "r", encoding="utf8")
for row in file:
    if row.split("\t")[3] == "imenica\n":
        if row.split("\t")[1].lower().isalpha():
            words.add(row.split("\t")[1].lower())
words = list(words)
```

Kôd nakon primjene koncepata funkcije *filter* i lambda izraza:

```
words = {"otorinolaringologija"}
file = open("hrvatski-rjecnik.txt", "r", encoding="utf8")
for row in file:
    if row.split("\t")[3] == "imenica\n":
        words.add(row.split("\t")[1].lower())
words = list(filter(lambda x: True if x.isalpha() else False,
words))
```

U ovom slučaju kôd ima minimalne promjene te primjenom ovog koncepta njegova razumljivost i čitljivost su ostale nepromijenjene.

## 5.6. Sažet zapis listi

Koncept sažetog zapisa listi primijenjen je kod inicijalizacije liste opened koja sadrži istinu ili laž za svako slovo riječi koje se pogađa, te označava je li to slovo otkriveno ili nije. Također je primijenjen za dobivanje broja otvorenih slova riječi koja se pogađa.

Kôd prije primjene koncepta sažetog zapisa listi:

```
opened = []
for i in range(0, length+1):
    opened.append(False)
countOpened = 0
for i in opened:
    if i:
        countOpened += 1
```

Kôd nakon primjene koncepta sažetog zapisa listi:

```
opened = [False for i in range(0, length+1)]
countOpened = sum([1 if x else 0 for x in opened])
```

Broj linija kôda se smanjio s 7 na samo dvije i time se njegova čitljivost povećala.

## 5.8. Usporedba kôda

Program pisan proceduralnom paradigmom ima 150 linija kôda, dok ih program pisan funkcijskom paradigmom ima 140.

| Funkcija | Opis funkcije | Broj linija kôda (proceduralna paradigma) | Broj linija kôda (funkcijska paradigma) |
|---|---|---|---|
| displayText | Ispisuje tekst koji mu se proslijedi kao parametar | 8 | 9 |
| drawHangingPole | Crta vješalo | 5 | 6 |
| drawStick | Crta čovječuljka ovisno koliko je pogađač puta pogriješio | 22 | 24 |

*Tablica 3. Usporedba broja linija kôda igre vješala*

Većina funkcija su minimalno promijenjene, jedina promjena između proceduralnog i funkcijskog pristupa je ta da su varijable koje funkcija koristi definirane u njima. Time su one postale čiste funkcije, te nema razlike u čitljivosti kôda.

Koncept sažetih zapisa listi je u razvoju ove igre dao najveću razliku u razumijevanju i čitljivosti kôda. Ostali koncepti su ili nisu upotrjebljeni ili su minimalno promijenili kôd.

# 6. Zaključak

U ovom radu su se implementirale dvije igre proceduralnim programiranjem i onda su se na taj kôd primijenili koncepti funkcijskog programiranja. Prvo se implementirala igra križić-kružić, a zatim igra vješala. Time se prikazalo kako se ti koncepti mogu primijeniti za implementaciju igara.

Korištenje funkcijske paradigme nije teško niti komplicirano, a u isto vrijeme rezultira čitljivijim i razumljivijim programskim kôdom. Većina koncepata se može uvijek koristiti, ali njegova korisnost ovisi o njegovoj primjeni. Kod Pythona je dobro to što podržava više paradigmi programiranja i time se programer ne mora limitirati na jednu paradigmu. Programer može odabrati koji god koncept mu najviše odgovara za neku specifičnu situaciju, te se ne mora striktno držati svih pravila čistog funkcijskog programiranja. Ne moraju sve varijable biti nepromjenjive, već se taj koncept može koristiti gdje je potrebno.

Čitanje kôda čije su funkcije čiste je lakše zbog toga što se sve potrebne varijable nalaze unutar funkcije i jer se cijela funkcionalnost funkcije mora sadržati u toj funkciji. Takve funkcije je lako testirati jer nemaju utjecaj na varijable izvan te funkcije. Implementiranje ovog koncepta je vrlo jednostavno, a pruža sigurnost da se varijable izvan funkcije neće mijenjati.

Primjenu koncepta funkcije višeg reda nije tako lako pronaći, ali u situacijama gdje se može su vrlo korisne. Ako se funkcija višeg reda koristi za stvaranje i vraćanje novih funkcija, vrlo se lako može proširiti njena funkcionalnost. Ako se funkcija višeg reda koristi za upotrebu funkcije koja je proslijeđena kao parametar, uštedi se na količini kôda kojeg programer mora napisati. Implementacija ovog koncepta je malo zahtjevnija, ali njena primjena može biti dosta korisna.

Lambda izrazi su vrlo korisni kada programer treba neku jednostavnu funkciju koju neće upotrebljavati više puta, na primjer ako programeru treba funkcija koja vraća istinu ili laž ovisno o nekom uvjetu. Definiranjem takve funkcije lambda izrazom je brže i jednostavnije jer se ne mora definirati njeno ime. Također, lambda izraz se piše u jednoj liniji kôda, dok za standardnu definiciju funkcije trebaju barem dvije linije kôda.

Funkcijama *map* i *filter* programer može pojednostaviti kôd tako da umjesto korištenja ugniježdenih *for* petlji za prolaženje po elementima neke liste mapira neku funkciju na tu listu. Uz ove funkcije je korisno upotrijebiti lambda izraz za definiranje funkcije koja se mapira ili po kojoj se odlučuje koje elemente liste želimo zadržati filtriranjem. Lako im se može naći primjena nakon koje kôd postaje razumljiviji jer se lakše može shvatiti svrhu tog kôda.

Sažet zapis listi je koncept koji je vrlo koristan za inicijalizaciju i obradu listi. Lako se implementira, te kôd postaje kraći i jasniji. Programer ima mogućnost kombiniranja elemenata iz više listi, te ako je potrebno i vršiti nekakve operacije nad njima kako bi dobio novu listu za daljnju obradu. Velika prednost ovog koncepta je ta što se ovaj koncept vrlo lako može primijeniti na puno različitih načina, dodavanjem uvjeta kojim se testiraju elementi ili čak uzimanjem elemenata iz različitih listi.

Koncept rekurzije je zahtjevan za implementaciju, jer zahtjeva specifično razmišljanje o ponavljanju izvršavanja kôda. Zbog toga neki programeri izbjegavaju njeno korištenje, makar može biti vrlo korisna. Korištenjem rekurzije se mogu riješiti neki veći problemi, ali njen nedostatak je taj što može dovesti do velike potrošnje računalne memorije. Također, vrlo ju je lako krivo napisati i time program dovesti u stanje u kojem ne može izaći iz rekurzije.

Primjenom ovih koncepata na igri križić-kružić kôd je postao jasniji, a samim time i čitljiviji. Primjenom koncepta sažetog zapisa listi dosta se pojednostavilo čitanje kôda. Za razliku od proceduralnog pristupa u kojem za razumijevanje ugniježdenih *for* petlji treba više vremena da se shvati što radi koja petlja, funkcijskim pristupom cijela se funkcionalnost kôda može pročitati

u jednoj liniji. Kod ove igre, također je pomogao koncept funkcije višeg reda. Raščlanio je jednu funkciju na tri funkcije kojima je lakše shvatiti svrhu.

Ista situacija je i kod primjene koncepata funkcijske paradigme na igri vješala. Njezin kôd je također postao čitljiviji od proceduralnog pristupa, no igra vješala je dosta jednostavnija pa se za neke koncepte nije mogla naći primjena.

Primjenom koncepata funkcijskog programiranja na ovim igrama složenost kôda je ili ostala ista ili se smanjila. Što znači da se njihovom primjenom može samo pojednostaviti kôd.

# 7. Literatura


[1] E. L. Christopher Piro, »Functional Programming at Facebook,« 2009. [Mrežno]. Available: https://web.archive.org/web/20091017070140/http://cufp.galois.com/2009/abstracts.html#ChristopherPiroEugeneLetuchy. [Pokušaj pristupa 10. rujna 2020.].

[2] WhatsApp, »1 million is so 2011,« 2012. [Mrežno]. Available: https://blog.whatsapp.com/1-million-is-so-2011/. [Pokušaj pristupa 10. rujna 2020.].

[3] »What is Erlang,« [Mrežno]. Available: http://erlang.org/faq/introduction.html. [Pokušaj pristupa 10. rujna 2020.].

[4] W. Clinger, »Multitasking in MacScheme+Toolsmith™,« [Mrežno]. Available: http://preserve.mactech.com/articles/mactech/Vol.03/03.12/Multitasking/index.html. [Pokušaj pristupa 10. rujna 2020.].

[5] »Wikipedia,« [Mrežno]. Available: https://en.wikipedia.org/wiki/Computer_simulation. [Pokušaj pristupa 10. rujna 2020.].

[6] Wikipedia, »Functianal programming, Applications,« [Mrežno]. Available: https://en.wikipedia.org/wiki/Functional_programming#Applications. [Pokušaj pristupa 10. rujna 2020.].

[7] J. Bartoníček, »Programming Language Paradigms & The Main Principles of Object-Oriented Programming,« *CRIS - Bulletin of the Centre for Research and Interdisciplinary Study,* 2014.

[8] A. Sharp, Smalltalk by Example, 1997.

[9] »What Is an Object?,« Oracle, [Mrežno]. Available: https://docs.oracle.com/javase/tutorial/java/concepts/object.html. [Pokušaj pristupa 17. kolovoza 2020.].

[10] W. A. Newman i A. R. Hendrickson, »The object-oriented paradigm as an implementation of systems theory in IS,« *Journal of International Information Management: Article 7,* svez. 5, br. 2, 1996.

[11] M. Sanatan, »Functional Programming in Python,« StackAbuse, [Mrežno]. Available: https://stackabuse.com/functional-programming-in-python/. [Pokušaj pristupa 18. kolovoza 2020.].

[12] P. Pandey, »Elements of Functional Programming in Python,« Towards Data Science, 12. srpanj 2019. [Mrežno]. Available: https://towardsdatascience.com/elements-of-functional-programming-in-python-1b295ea5bbe0. [Pokušaj pristupa 19. kolovoza 2020.].

[13] S. Thompson, Haskell: The Craft of Functional Programming, 1996.

[14] »What is Recursion?,« SparkNotes, [Mrežno]. Available: https://www.sparknotes.com/cs/recursion/whatisrecursion/section1/. [Pokušaj pristupa 22. kolovoza 2020.].

[15] G. A. Hyelle, »Python Type Checking,« RealPython, [Mrežno]. Available: https://realpython.com/python-type-checking/. [Pokušaj pristupa 29. kolovoza 2020.].

[16] D. Stefan, »CSE 130: Type Systems, Type Inference and Polymorphism,« [Mrežno]. Available: https://cseweb.ucsd.edu/~dstefan/cse130-winter17/book/ch06.pdf. [Pokušaj pristupa 29. kolovoza 2020.].

[17] A. Alexander, »Benefits of Functional Programming,« [Mrežno]. Available: https://alvinalexander.com/scala/fp-book/benefits-of-functional-programming/. [Pokušaj pristupa 30. kolovoza 2020.].

[18] A. Alexander, »Disadvantages of Functional Programming,« [Mrežno]. Available: https://alvinalexander.com/scala/fp-book/disadvantages-of-functional-programming/. [Pokušaj pristupa 30. kolovoza 2020.].

[19] E. Arneson, »How to Play Tic-Tac-Toe,« 12. svibnja 2020. [Mrežno]. Available: https://www.thesprucecrafts.com/tic-tac-toe-game-rules-412170. [Pokušaj pristupa 30. kolovoza 2020.].

[20] »Križić i kružić,« Wikipedia, [Mrežno]. Available: https://hr.wikipedia.org/wiki/Kru%C5%BEi%C4%87_i_kri%C5%BEi%C4%87. [Pokušaj pristupa 30. kolovoza 2020.].

[21] »Minimax Algorithm in Game Theory,« GeeksforGeeks, 28. svibanj 2019. [Mrežno]. Available: https://www.geeksforgeeks.org/minimax-algorithm-in-game-theory-set-1-introduction/. [Pokušaj pristupa 31. kolovoza 2020.].

[22] A. Kamil, »CS61B, Minimax,« 2003. [Mrežno]. Available: http://web.eecs.umich.edu/~akamil/teaching/sp03/minimax.pdf. [Pokušaj pristupa 31. kolovoza 2020.].

[23] »Frekvencija slova u hrvatskom jeziku,« [Mrežno]. Available: http://www.mathos.unios.hr/kiss/kript_pred.pdf. [Pokušaj pristupa 8. rujna 2020.].

## 8. Popis slika

Slika 1. Primjer izvršavanja minimax algoritma ..... 16

## 9. Popis tablica

Tablica 1. Usporedba broja linija kôda igre križić-kružić ..... 17

Tablica 2. Primjer odvijanja igre vješala ..... 20

Tablica 3. Usporedba broja linija kôda igre vješala ..... 22

## 10. Popis priloga

Uz ovaj rad priložene su sljedeće datoteke:

A. tictactoe_proceduralno.py – igra križić-kružić prije primjene koncepata funkcijskog programiranja
B. tictactoe_funkcijski.py – igra križić-kružić nakon primjene koncepata funkcijskog programiranja
C. hangman_proceduralno.py – igra vješala prije primjene koncepata funkcijskog programiranja
D. hangman_funkcijski.py – igra vješala nakon primjene koncepata funkcijskog programiranja
E. hrvatski_rjecnik.txt – hrvatski rječnik iz kojeg se uzimaju riječi za igru vješala
F. Roboto-Regular.ttf – font za ispis teksta u igrama

## Prilog A

```
import pygame
import time

# Colors
blue = (0, 170, 255)
darkBlue = (0, 130, 196)
green = (0, 255, 213)
darkGreen = (0, 148, 124)
white = (255, 255, 255)
lightGray = (118, 133, 148)
gray = (70, 84, 94)
darkGray = (39, 44, 48)

# Display resolution
displaySize = 1000

# Initialize window
pygame.init()
window = pygame.display.set_mode((displaySize, displaySize))
window.fill(darkGray)
pygame.display.set_caption("Tic-Tac-Toe")
font = pygame.font.Font("Roboto-Regular.ttf", int(displaySize/7))

# Board
boardSize = 3
numForWin = 3
board = []
for _ in range(0, boardSize):
    temp = []
    for _ in range(0, boardSize):
        temp.append(' ')
    board.append(temp)

# Draw cells
borderWidth = displaySize/((boardSize+1)*9)
cellWidth = displaySize/(boardSize*1.125)
cells = [[0 for i in range(0, boardSize)] for i in range(0,
boardSize)]
for i in range(0, boardSize):
    for j in range(0, boardSize):
        cells[i][j] = pygame.draw.rect(window, lightGray,
(borderWidth + j * (cellWidth + borderWidth), borderWidth + i *
(cellWidth + borderWidth), cellWidth, cellWidth))


def drawX(i, j):
    lineWidth = cellWidth/8
```

```
    topLeftPoint = (cells[i][j].x + borderWidth + lineWidth/2,
cells[i][j].y + borderWidth)
    bottomRightPoint = (cells[i][j].x + cellWidth-borderWidth -
lineWidth/2, cells[i][j].y + cellWidth-borderWidth)
    topRightPoint = (cells[i][j].x + cellWidth-borderWidth -
lineWidth/2, cells[i][j].y + borderWidth)
    bottomLeftPoint = (cells[i][j].x + borderWidth + lineWidth/2,
cells[i][j].y + cellWidth-borderWidth)
    # Outline
    pygame.draw.line(window, darkBlue, (topLeftPoint[0] - 3,
topLeftPoint[1] - 3), (bottomRightPoint[0] + 3, bottomRightPoint[1]
+ 3), int(lineWidth + 3*2))
    pygame.draw.line(window, darkBlue, (topRightPoint[0] + 3,
topRightPoint[1] - 3), (bottomLeftPoint[0] - 3, bottomLeftPoint[1] +
3), int(lineWidth + 3*2))
    # X
    pygame.draw.line(window, blue, topLeftPoint, bottomRightPoint,
int(lineWidth))
    pygame.draw.line(window, blue, topRightPoint, bottomLeftPoint,
int(lineWidth))
    pygame.display.update()


def drawO(i, j):
    # Outline
    pygame.draw.arc(window, darkGreen, (cells[i][j].x + borderWidth
- 3, cells[i][j].y + borderWidth - 3, cellWidth - 2 * borderWidth +
3*2, cellWidth - 2 * borderWidth + 3*2), 0, 360, int(cellWidth / 10
+ 3*2))
    # O
    pygame.draw.arc(window, green, (cells[i][j].x + borderWidth,
cells[i][j].y + borderWidth, cellWidth-2*borderWidth, cellWidth-
2*borderWidth), 0, 360, int(cellWidth/10))
    pygame.display.update()


def displayText(message):
    # Shadow
    text = font.render(message, True, darkGray)
    window.blit(text, text.get_rect(center=(displaySize/2 + 3,
displaySize/2 + 3)))
    # Text
    text = font.render(message, True, white)
    window.blit(text, text.get_rect(center=(displaySize/2,
displaySize/2)))
    pygame.display.update()


def resetGame():
```

```
    global cells
    global board
    # Refill window
    window.fill(darkGray)
    # Redraw cells
    for i in range(0, boardSize):
        for j in range(0, boardSize):
            cells[i][j] = pygame.draw.rect(window, lightGray,
(borderWidth + j * (cellWidth + borderWidth), borderWidth + i *
(cellWidth + borderWidth), cellWidth, cellWidth))
    # Reset board
    board = []
    for _ in range(0, boardSize):
        temp = []
        for _ in range(0, boardSize):
            temp.append(' ')
        board.append(temp)


def checkWinner(x, y):
    # Row
    for j in range(0, boardSize):
        if board[x][j] == ' ' or boardSize-j < numForWin:
            continue
        inARow = 0
        player = board[x][j]
        for k in range(0, boardSize-j):
            if board[x][j+k] != player:
                break
            inARow += 1
            if inARow == numForWin:
                return player

    # Column
    for i in range(0, boardSize):
        if board[i][y] == ' ' or boardSize-i < numForWin:
            continue
        inARow = 0
        player = board[i][y]
        for k in range(0, boardSize-i):
            if board[i+k][y] != player:
                break
            inARow += 1
            if inARow == numForWin:
                return player

    # Main diagonal
    for i in range(0, boardSize):
        for j in range(0, boardSize):
```

```
            if board[i][j] == ' ' or boardSize-max(i, j) <
numForWin:
                continue
            k = x - min(x, y)
            l = y - min(x, y)
            inARow = 0
            player = board[x][y]
            for m in range(0, boardSize-max(k, l)):
                if board[k+m][l+m] != player:
                    inARow = 0
                    continue
                inARow += 1
                if inARow == numForWin:
                    return player

    # Secondary diagonal
    for i in range(0, boardSize):
        for j in range(0, boardSize):
            if board[i][j] == ' ' or min(i+1, boardSize-j) <
numForWin:
                continue
            k = x
            l = y
            while k < boardSize-1 and l > 0:
                k += 1
                l -= 1
            if min(k+1, boardSize-l) < numForWin:
                continue
            inARow = 0
            player = board[x][y]
            for m in range(0, min(k+1, boardSize-l)):
                if board[k-m][l+m] != player:
                    inARow = 0
                    continue
                inARow += 1
                if inARow == numForWin:
                    return player

    # Draw
    boardEmptySpaces = 0
    for i in range(0, boardSize):
        for j in range(0, boardSize):
            if board[i][j] == ' ':
                boardEmptySpaces += 1
    if boardEmptySpaces == 0:
        return 'draw'

    # Nobody won
    return 0
```

```
def heuristicEvaluate(board, isMaximizing):

    if isMaximizing:
        player = 'X'
    else:
        player = 'O'

    score = 0
    # Row
    for i in range(0, boardSize):
        for j in range(0, boardSize):
            if board[i][j] == ' ' or boardSize-j < numForWin:
                continue
            inARow = -1
            extraSpace = 0
            for k in range(0, boardSize-j):
                if board[i][j+k] != player and board[i][j+k] != ' ':
                    break
                if board[i][j+k] == ' ':
                    extraSpace += 1
                else:
                    inARow += 1
            if inARow >= 0 and inARow+1+extraSpace >= numForWin:
                score += 10**inARow
    # Column
    for i in range(0, boardSize):
        for j in range(0, boardSize):
            if board[i][j] == ' ' or boardSize-i < numForWin:
                continue
            inARow = -1
            extraSpace = 0
            for k in range(0, boardSize-i):
                if board[i+k][j] != player and board[i+k][j] != ' ':
                    break
                if board[i+k][j] == ' ':
                    extraSpace += 1
                else:
                    inARow += 1
            if inARow >= 0 and inARow+1+extraSpace >= numForWin:
                score += 10**inARow
    # Main diagonal
    for i in range(0, boardSize):
        for j in range(0, boardSize):
            if board[i][j] == ' ' or boardSize-max(i, j) <
numForWin:
                continue
            inARow = -1
```

```
            extraSpace = 0
            for k in range(0, boardSize-max(i, j)):
                if board[i+k][j+k] != player and board[i+k][j+k] !=
' ':
                    break
                if board[i+k][j+k] == ' ':
                    extraSpace += 1
                else:
                    inARow += 1
            if inARow >= 0 and inARow+1+extraSpace >= numForWin:
                score += 10**inARow
    # Secondary diagonal
    for i in range(0, boardSize):
        for j in range(0, boardSize):
            if board[i][j] == ' ' or min(i+1, boardSize-j) <
numForWin:
                continue
            inARow = -1
            extraSpace = 0
            for k in range(0, min(i+1, boardSize-j)):
                if board[i-k][j+k] != player and board[i-k][j+k] !=
' ':
                    break
                if board[i-k][j+k] == ' ':
                    extraSpace += 1
                else:
                    inARow += 1
            if inARow >= 0 and inARow+1+extraSpace >= numForWin:
                score += 10**inARow
    if isMaximizing:
        return score
    else:
        return -score


def minimax(x, y, alpha, beta, depth, isMaximizing, board):
    if boardSize > 3 and depth > 3:
        return heuristicEvaluate(board, isMaximizing)

    result = checkWinner(x, y)
    if result:
        if result == 'X':
            score = -100000000 + 5**depth
        elif result == 'O':
            score = 100000000 - 5**depth
        else:
            score = 0
        return score
```

```
    if isMaximizing:
        score = float("-inf")
        for i in range(0, boardSize):
            for j in range(0, boardSize):
                if score == 100000000 - 5**depth:
                    return score
                if board[i][j] == ' ':
                    board[i][j] = 'O'
                    score = max(score, minimax(i, j, alpha, beta,
depth+1, False, board))
                    # print("---------------[ O ]---------------")
                    # out = str(board).split(']', ['')
                    # print(i, j, '],\n     ['.join(out), score)
                    board[i][j] = ' '
                    alpha = max(alpha, score)
                    if alpha >= beta:
                        return score
        return score
    else:
        score = float("inf")
        for i in range(0, boardSize):
            for j in range(0, boardSize):
                if score == -100000000 + 5**depth:
                    return score
                if board[i][j] == ' ':
                    board[i][j] = 'X'
                    score = min(score, minimax(i, j, alpha, beta,
depth+1, True, board))
                    # print("---------------[ X ]---------------")
                    # out = str(board).split(']', ['')
                    # print(i, j, '],\n     ['.join(out), score)
                    board[i][j] = ' '
                    beta = min(beta, score)
                    if beta <= alpha:
                        return score
        return score


# Game
run = True
while run:

    # Events
    for event in pygame.event.get():
        # Quit
        if event.type == pygame.QUIT:
            run = False
        # Pressed space
        if event.type == pygame.KEYDOWN:
```

```
            if event.key == pygame.K_SPACE:
                resetGame()
        # Mouse pressed
        if event.type == pygame.MOUSEBUTTONUP:
            # Get position
            position = pygame.mouse.get_pos()
            for i in range(0, boardSize):
                for j in range(0, boardSize):
                    # Has player clicked in any cell that is free?
                    if cells[i][j].collidepoint(position) and
board[i][j] == ' ':
                        # Draw X and check if player won
                        drawX(i, j)
                        board[i][j] = 'X'
                        winner = checkWinner(i, j)
                        if winner:
                            if winner == 'draw':
                                displayText("Draw!")
                            else:
                                displayText("You " + ("win!" if
winner == 'X' else "lose!"))
                            # time.sleep(2)
                            # resetGame()
                        else:
                            # Computers turn
                            bestScore = float("-inf")
                            for k in range(0, boardSize):
                                for l in range(0, boardSize):
                                    if board[k][l] == ' ':
                                        board[k][l] = 'O'
                                        before = time.time()
                                        score = minimax(k, l,
float("-inf"), float("inf"), 0, False, board)
                                        after = time.time()
                                        print("\n---------------
[ O ]---------------")
                                        out = str(board).split(']],
[')
                                        print(k, l, ']],\n
['.join(out), score)
                                        print('\nTime: ', after-
before)
                                        board[k][l] = ' '
                                        if score > bestScore:
                                            bestScore = score
                                            i = k
                                            j = l
                            print("\nWaiting for player's turn...")
                            # Draw O and check if computer won
```

```
                                drawO(i, j)
                                board[i][j] = 'O'
                                winner = checkWinner(i, j)
                                if winner:
                                    if winner == 'draw':
                                        displayText("Draw!")
                                    else:
                                        displayText("You " + ("win!" if
winner == 'X' else "lose!"))

    pygame.display.update()
    pygame.time.delay(5)

pygame.quit()
```

## Prilog B

```
import pygame
import time

# Colors
lightGray = (118, 133, 148)
darkGray = (39, 44, 48)

# Display resolution
displaySize = 1000

# Initialize window
pygame.init()
window = pygame.display.set_mode((displaySize, displaySize))
window.fill(darkGray)
pygame.display.set_caption("Tic-Tac-Toe")
font = pygame.font.Font("Roboto-Regular.ttf", int(displaySize/7))

# Board
boardSize = 3
numForWin = 3
board = [[' ' for _ in range(0, boardSize)] for _ in range(0,
boardSize)]

# Draw empty cells
borderWidth = displaySize / ((boardSize + 1) * 9)
cellWidth = displaySize / (boardSize * 1.125)
cells = [[0 for _ in range(0, boardSize)] for _ in range(0,
boardSize)]
for i in range(0, boardSize):
    for j in range(0, boardSize):
        cells[i][j] = pygame.draw.rect(window, lightGray,
(borderWidth + j * (cellWidth + borderWidth), borderWidth + i *
(cellWidth + borderWidth), cellWidth, cellWidth))


def drawX(i, j, window, displaySize, boardSize, cells):
    blue = (0, 170, 255)
    darkBlue = (0, 130, 196)
    borderWidth = displaySize/((boardSize+1)*9)
    cellWidth = displaySize/(boardSize*1.125)
    lineWidth = cellWidth/8

    topLeftPoint = (cells[i][j].x + borderWidth + lineWidth/2,
cells[i][j].y + borderWidth)
    bottomRightPoint = (cells[i][j].x + cellWidth-borderWidth -
lineWidth/2, cells[i][j].y + cellWidth-borderWidth)
```

```
    topRightPoint = (cells[i][j].x + cellWidth-borderWidth -
lineWidth/2, cells[i][j].y + borderWidth)
    bottomLeftPoint = (cells[i][j].x + borderWidth + lineWidth/2,
cells[i][j].y + cellWidth-borderWidth)
    # Outline
    pygame.draw.line(window, darkBlue, (topLeftPoint[0] - 3,
topLeftPoint[1] - 3), (bottomRightPoint[0] + 3, bottomRightPoint[1]
+ 3), int(lineWidth + 3*2))
    pygame.draw.line(window, darkBlue, (topRightPoint[0] + 3,
topRightPoint[1] - 3), (bottomLeftPoint[0] - 3, bottomLeftPoint[1] +
3), int(lineWidth + 3*2))
    # X
    pygame.draw.line(window, blue, topLeftPoint, bottomRightPoint,
int(lineWidth))
    pygame.draw.line(window, blue, topRightPoint, bottomLeftPoint,
int(lineWidth))
    pygame.display.update()


def drawO(i, j, window, displaySize, boardSize, cells):
    green = (0, 255, 213)
    darkGreen = (0, 148, 124)
    borderWidth = displaySize/((boardSize+1)*9)
    cellWidth = displaySize/(boardSize*1.125)
    # Outline
    pygame.draw.arc(window, darkGreen, (cells[i][j].x + borderWidth
- 3, cells[i][j].y + borderWidth - 3, cellWidth - 2 * borderWidth +
3*2, cellWidth - 2 * borderWidth + 3*2), 0, 360, int(cellWidth / 10
+ 3*2))
    # O
    pygame.draw.arc(window, green, (cells[i][j].x + borderWidth,
cells[i][j].y + borderWidth, cellWidth-2*borderWidth, cellWidth-
2*borderWidth), 0, 360, int(cellWidth/10))
    pygame.display.update()


def displayText(message, displaySize, window, font):
    white = (255, 255, 255)
    darkGray = (39, 44, 48)
    # Shadow
    text = font.render(message, True, darkGray)
    window.blit(text, text.get_rect(center=(displaySize/2+3,
displaySize/2+3)))
    # Text
    text = font.render(message, True, white)
    window.blit(text, text.get_rect(center=(displaySize/2,
displaySize/2)))
    pygame.display.update()
```

```
def resetGame():
    global cells
    global board
    # Refill window
    window.fill(darkGray)
    # Redraw empty cells
    for i in range(0, boardSize):
        for j in range(0, boardSize):
            cells[i][j] = pygame.draw.rect(window, lightGray,
(borderWidth + j * (cellWidth + borderWidth), borderWidth + i *
(cellWidth + borderWidth), cellWidth, cellWidth))
    # Reset board and board empty spaces
    board = [[' ' for _ in range(0, boardSize)] for _ in range(0,
boardSize)]


def getLimit(direction, i, j, boardSize):
    if direction == 'row':
        return boardSize-j
    elif direction == 'column':
        return boardSize-i
    elif direction == 'main_diagonal':
        return boardSize-max(i, j)
    elif direction == 'secondary_diagonal':
        return min(i+1, boardSize-j)
    else:
        return None


def getInARow(direction):
    if direction == 'row':
        coef = (1, 0, 0, 1)  # [x*1 + k*0][y*0 + k*1] = [x][y+k]
    elif direction == 'column':
        coef = (0, 1, 1, 0)  # [x*0 + k*1][y*1 + k*0] = [x+k][y]
    elif direction == 'main_diagonal':
        coef = (1, 1, 1, 1)  # [x*1 + k*1][y*1 + k*1] = [x+k][y+k]
    elif direction == 'secondary_diagonal':
        coef = (1, -1, 1, 1)  # [x*1 + k*-1][y*1 + k*1] = [x-k][y+k]
    else:
        return None

    def countInARow(x, y, board, boardSize, numForWin):
        inARow = 0
        for i in range(0, boardSize):
            for j in range(0, boardSize):
                if board[i][j] == ' ' or getLimit(direction, i, j,
boardSize) < numForWin:
                    continue
```

```
                posX = x
                posY = y
                if direction == 'row':
                    posY -= y
                elif direction == 'column':
                    posX -= x
                elif direction == 'main_diagonal':
                    posX -= min(x, y)
                    posY -= min(x, y)
                elif direction == 'secondary_diagonal':
                    while posX < boardSize-1 and posY > 0:
                        posX += 1
                        posY -= 1
                if getLimit(direction, posX, posY, boardSize) <
numForWin:
                    continue
                inARow = 0
                player = board[x][y]
                for k in range(0, getLimit(direction, posX, posY,
boardSize)):
                    if board[posX*coef[0] + k*coef[1]][posY*coef[2]
+ k*coef[3]] != player:
                        inARow = 0
                        continue
                    inARow += 1
                    if inARow == numForWin:
                        return inARow
        return inARow

    return countInARow


def checkWinner(x, y, board, boardSize, numForWin):
    # Row
    row = getInARow('row')
    if row(x, y, board, boardSize, numForWin) == numForWin:
        return board[x][y]

    # Column
    column = getInARow('column')
    if column(x, y, board, boardSize, numForWin) == numForWin:
        return board[x][y]

    # Main diagonal
    mainDiagonal = getInARow('main_diagonal')
    if mainDiagonal(x, y, board, boardSize, numForWin) == numForWin:
        return board[x][y]

    # Secondary diagonal
```

```
    secondaryDiagonal = getInARow('secondary_diagonal')
    if secondaryDiagonal(x, y, board, boardSize, numForWin) ==
numForWin:
        return board[x][y]

    # Draw
    emptySpaces = sum(list(map(lambda x: x.count(' '), board)))
    # emptySpaces = len(list(filter(lambda x: True if x == ' ' else
False, [y for row in board for y in row])))
    if emptySpaces == 0:
        return 'draw'

    # Nobody won
    return 0


def heuristicEvaluate(board, isMaximizing, boardSize, numForWin):

    if isMaximizing:
        player = 'X'
    else:
        player = 'O'

    score = 0
    # Row
    for i in range(0, boardSize):
        for j in range(0, boardSize):
            if board[i][j] == ' ' or boardSize-j < numForWin:
                continue
            inARow = -1
            extraSpace = 0
            for k in range(0, boardSize-j):
                if board[i][j+k] != player and board[i][j+k] != ' ':
                    break
                if board[i][j+k] == ' ':
                    extraSpace += 1
                else:
                    inARow += 1
            if inARow >= 0 and inARow+1+extraSpace >= numForWin:
                score += 10**inARow
    # Column
    for i in range(0, boardSize):
        for j in range(0, boardSize):
            if board[i][j] == ' ' or boardSize-i < numForWin:
                continue
            inARow = -1
            extraSpace = 0
            for k in range(0, boardSize-i):
                if board[i+k][j] != player and board[i+k][j] != ' ':
```

```
                    break
                if board[i+k][j] == ' ':
                    extraSpace += 1
                else:
                    inARow += 1
            if inARow >= 0 and inARow+1+extraSpace >= numForWin:
                score += 10**inARow
    # Main diagonal
    for i in range(0, boardSize):
        for j in range(0, boardSize):
            if board[i][j] == ' ' or boardSize-max(i, j) <
numForWin:
                continue
            inARow = -1
            extraSpace = 0
            for k in range(0, boardSize-max(i, j)):
                if board[i+k][j+k] != player and board[i+k][j+k] !=
' ':
                    break
                if board[i+k][j+k] == ' ':
                    extraSpace += 1
                else:
                    inARow += 1
            if inARow >= 0 and inARow+1+extraSpace >= numForWin:
                score += 10**inARow
    # Secondary diagonal
    for i in range(0, boardSize):
        for j in range(0, boardSize):
            if board[i][j] == ' ' or min(i+1, boardSize-j) <
numForWin:
                continue
            inARow = -1
            extraSpace = 0
            for k in range(0, min(i+1, boardSize-j)):
                if board[i-k][j+k] != player and board[i-k][j+k] !=
' ':
                    break
                if board[i-k][j+k] == ' ':
                    extraSpace += 1
                else:
                    inARow += 1
            if inARow >= 0 and inARow+1+extraSpace >= numForWin:
                score += 10**inARow
    if isMaximizing:
        return score
    else:
        return -score
```

```
def minimax(x, y, alpha, beta, depth, isMaximizing, board,
boardSize, numForWin):
    if boardSize > 3 and depth > 3:
        return heuristicEvaluate(board, isMaximizing, boardSize,
numForWin)

    result = checkWinner(x, y, board, boardSize, numForWin)
    if result:
        if result == 'X':
            score = -100000000 + 5**depth
        elif result == 'O':
            score = 100000000 - 5**depth
        else:
            score = 0
        return score

    if isMaximizing:
        score = float("-inf")
        for i in range(0, boardSize):
            for j in range(0, boardSize):
                if score == 100000000 - 5**depth:
                    return score
                if board[i][j] == ' ':
                    board[i][j] = 'O'
                    score = max(score, minimax(i, j, alpha, beta,
depth+1, False, board, boardSize, numForWin))
                    # print("---------------[ O ]---------------")
                    # out = str(board).split(']', ['')
                    # print(i, j, ',\n     ['.join(out), score)
                    board[i][j] = ' '
                    alpha = max(alpha, score)
                    if alpha >= beta:
                        return score
        return score
    else:
        score = float("inf")
        for i in range(0, boardSize):
            for j in range(0, boardSize):
                if score == -100000000 + 5**depth:
                    return score
                if board[i][j] == ' ':
                    board[i][j] = 'X'
                    score = min(score, minimax(i, j, alpha, beta,
depth+1, True, board, boardSize, numForWin))
                    # print("---------------[ X ]---------------")
                    # out = str(board).split(']', ['')
                    # print(i, j, ',\n     ['.join(out), score)
                    board[i][j] = ' '
                    beta = min(beta, score)
```

```
                    if beta <= alpha:
                        return score
        return score

# Game loop
run = True
while run:

    # Events
    for event in pygame.event.get():
        # Quit
        if event.type == pygame.QUIT:
            run = False
        # Pressed space
        if event.type == pygame.KEYDOWN:
            if event.key == pygame.K_SPACE:
                resetGame()
        # Mouse pressed
        if event.type == pygame.MOUSEBUTTONUP:
            # Get position
            position = pygame.mouse.get_pos()
            for i in range(0, boardSize):
                for j in range(0, boardSize):
                    # Has player clicked in any cell that is free?
                    if cells[i][j].collidepoint(position) and
board[i][j] == ' ':
                        # Draw X and check if player won
                        drawX(i, j, window, displaySize, boardSize,
cells)
                        board[i][j] = 'X'
                        winner = checkWinner(i, j, board, boardSize,
numForWin)
                        if winner:
                            if winner == 'draw':
                                displayText("Draw!", displaySize,
window, font)
                            else:
                                displayText("You " + ("win!" if
winner == 'X' else "lose!"), displaySize, window, font)
                            # time.sleep(2)
                            # resetGame()
                        else:
                            # Computers turn, run minimax
                            bestScore = float("-inf")
                            for k in range(0, boardSize):
                                for l in range(0, boardSize):
                                    if board[k][l] == ' ':
                                        board[k][l] = 'O'
                                        before = time.time()
```

```
                                        score = minimax(k, l,
float("-inf"), float("inf"), 0, False, board, boardSize, numForWin)
                                        after = time.time()
                                        print("\n---------------
[ O ]---------------")
                                        out = str(board).split(']),
[')
                                        print(k, l, '],\n
['.join(out), score)
                                        print('\nTime: ', after-
before)
                                        board[k][l] = ' '
                                        if score > bestScore:
                                            bestScore = score
                                            i = k
                                            j = l
                                print("\nWaiting for player's turn...")
                                # Draw O and check if computer won
                                drawO(i, j, window, displaySize,
boardSize, cells)
                                board[i][j] = 'O'
                                winner = checkWinner(i, j, board,
boardSize, numForWin)
                                if winner:
                                    if winner == 'draw':
                                        displayText("Draw!",
displaySize, window, font)
                                    else:
                                        displayText("You " + ("win!" if
winner == 'X' else "lose!"), displaySize, window, font)

    pygame.display.update()
    pygame.time.delay(5)

pygame.quit()
```

## Prilog C

```
import pygame
import random

# Colors
white = (255, 255, 255)
green = (0, 255, 213)
red = (224, 67, 67)
darkBlue = (0, 130, 196)
lightGray = (118, 133, 148)
darkGray = (34, 39, 43)
black = (0, 0, 0)

# Display resolution
displayWidth = 1280
displayHeight = 900

# Initialize window
pygame.init()
window = pygame.display.set_mode((displayWidth, displayHeight))
window.fill(white)
pygame.display.set_caption("Igra vješala")

def displayText(message, color, fontSize, xOffset, yOffset):
    font = pygame.font.Font("Roboto-Regular.ttf", fontSize)
    # Shadow
    text = font.render(message, True, black)
    window.blit(text, text.get_rect(center=(xOffset+1, yOffset+1)))
    # Text
    text = font.render(message, True, color)
    window.blit(text, text.get_rect(center=(xOffset, yOffset)))
    pygame.display.update()

def drawHangingPole():
    # Draw hanging pole
    pygame.draw.line(window, darkGray, (displayWidth/2, 80),
(displayWidth/2, 40), 6)
    pygame.draw.line(window, darkGray, (displayWidth/2-2, 40),
(displayWidth/2+120+3, 40), 6)
    pygame.draw.line(window, darkGray, (displayWidth/2+120, 40),
(displayWidth/2+120, 360), 6)
    pygame.draw.line(window, darkGray, (displayWidth/2+40, 360),
(displayWidth/2+160, 360), 6)

# Dictionary
words = {"otorinolaringologija"}
file = open("hrvatski_rjecnik.txt", "r", encoding="utf8")
for row in file:
```

```
    if row.split("\t")[3] == "imenica\n":
        if row.split("\t")[1].lower().isalpha():
            words.add(row.split("\t")[1].lower())
words = list(words)

# Lives left
lives = 6

# Pick a word
word = random.choice(words)
length = len(word)
print(word, length)

# Is character opened
opened = []
for i in range(0, length+1):
    opened.append(False)

# Display empty word
text = ' '.join('_' * length)
displayText(text, darkGray, int(displayWidth/16), displayWidth/2,
displayHeight-100)
drawHangingPole()


def drawStick():
    global lives
    if lives == 6: return
    # Draw head
    pygame.draw.arc(window, darkGray, (displayWidth / 2 - 29, 80,
61, 61), 0, 360, 6)
    if lives == 5: return
    # Draw body
    pygame.draw.line(window, darkGray, (displayWidth/2, 140),
(displayWidth/2, 240), 6)
    if lives == 4: return
    # Draw left arm
    pygame.draw.line(window, darkGray, (displayWidth/2-45, 150),
(displayWidth/2, 165), 6)
    if lives == 3: return
    # Draw right arm
    pygame.draw.line(window, darkGray, (displayWidth/2, 165),
(displayWidth/2+45, 150), 6)
    if lives == 2: return
    # Draw left leg
    pygame.draw.line(window, darkGray, (displayWidth/2-25, 275),
(displayWidth/2, 240), 8)
    if lives == 1: return
    # Draw right leg
```

```
    pygame.draw.line(window, darkGray, (displayWidth/2, 240),
(displayWidth/2+25, 275), 8)
    displayText("Izgubili ste!", red, int(displayWidth/20),
displayWidth/2, displayHeight/1.8)
    displayText("Riječ: " + word, darkBlue, int(displayWidth/20),
displayWidth/2, displayHeight/1.5)
    displayText("Pritisnite space za novu igru", darkBlue,
int(displayWidth/24), displayWidth/2, displayHeight/1.3)

# Game loop
run = True
while run:

    # Events
    for event in pygame.event.get():
        # Quit
        if event.type == pygame.QUIT:
            run = False
        # Pressed any key
        if event.type == pygame.KEYDOWN:
            if event.key == pygame.K_SPACE:
                window.fill(white)
                # Lives left
                lives = 6
                # Reset game
                word = random.choice(words)
                length = len(word)
                print(word, length)
                # Is character opened
                opened = [False] * length
                # Display empty word
                text = ' '.join('_' * length)
                displayText(text, darkGray, int(displayWidth/16),
displayWidth/2, displayHeight-100)
                drawHangingPole()
            # Send character to check
            if event.unicode.isalpha():
                # Check character
                correct = word.count(event.unicode.lower())
                if not correct:
                    lives -= 1
                for i in range(0, length):
                    if word[i] == event.unicode.lower():
                        opened[i] = True
                # Update word
                window.fill(white)
                drawHangingPole()
                drawStick()
                text = ''
```

```
                for i in range(0, length):
                    if opened[i]:
                        text += word[i] + (' ' if i < length - 1
else '')
                    else:
                        text += '_' + (' ' if i < length - 1 else
'')
                displayText(text, darkGray, int(displayWidth/16),
displayWidth / 2, displayHeight - 100)
                # Check winner
                countOpened = 0
                for i in opened:
                    if i:
                        countOpened += 1
                if countOpened == length:
                    displayText("Pobjedili ste!", green,
int(displayWidth/16), displayWidth/2, displayHeight/1.7)
                    displayText("Pritisnite space za novu igru",
darkBlue, int(displayWidth/16), displayWidth/2, displayHeight/1.4)

    pygame.display.update()
    pygame.time.delay(5)

pygame.quit()
```

## Prilog D

```
import pygame
import random

# Colors
white = (255, 255, 255)
green = (0, 255, 213)
darkBlue = (0, 130, 196)
darkGray = (34, 39, 43)

# Display resolution
displayWidth = 1280
displayHeight = 900

# Initialize window
pygame.init()
window = pygame.display.set_mode((displayWidth, displayHeight))
window.fill(white)
pygame.display.set_caption("Igra vješala")

def displayText(message, color, fontSize, xOffset, yOffset, window):
    black = (0, 0, 0)
    font = pygame.font.Font("Roboto-Regular.ttf", fontSize)
    # Shadow
    text = font.render(message, True, black)
    window.blit(text, text.get_rect(center=(xOffset+1, yOffset+1)))
    # Text
    text = font.render(message, True, color)
    window.blit(text, text.get_rect(center=(xOffset, yOffset)))
    pygame.display.update()

def drawHangingPole(window, displayWidth):
    darkGray = (34, 39, 43)
    # Draw hanging pole
    pygame.draw.line(window, darkGray, (displayWidth/2, 80),
(displayWidth/2, 40), 6)
    pygame.draw.line(window, darkGray, (displayWidth/2-2, 40),
(displayWidth/2+120+3, 40), 6)
    pygame.draw.line(window, darkGray, (displayWidth/2+120, 40),
(displayWidth/2+120, 360), 6)
    pygame.draw.line(window, darkGray, (displayWidth/2+40, 360),
(displayWidth/2+160, 360), 6)

# Dictionary
words = {"otorinolaringologija"}
file = open("hrvatski_rjecnik.txt", "r", encoding="utf8")
for row in file:
    if row.split("\t")[3] == "imenica\n":
```

```
        words.add(row.split("\t")[1].lower())
words = list(filter(lambda x: True if x.isalpha() else False,
words))

# Lives left
lives = 6

# Pick a word
word = random.choice(words)
length = len(word)
print(word, length)

# Is character opened
opened = [False for i in range(0, length+1)]

# Display empty word
text = ' '.join('_' * length)
displayText(text, darkGray, int(displayWidth/16), displayWidth/2,
displayHeight-100, window)
drawHangingPole(window, displayWidth)


def drawStick(lives, window, displayWidth, displayHeight):
    red = (224, 67, 67)
    darkBlue = (0, 130, 196)
    darkGray = (34, 39, 43)
    if lives == 6: return
    # Draw head
    pygame.draw.arc(window, darkGray, (displayWidth / 2 - 29, 80,
61, 61), 0, 360, 6)
    if lives == 5: return
    # Draw body
    pygame.draw.line(window, darkGray, (displayWidth/2, 140),
(displayWidth/2, 240), 6)
    if lives == 4: return
    # Draw left arm
    pygame.draw.line(window, darkGray, (displayWidth/2-45, 150),
(displayWidth/2, 165), 6)
    if lives == 3: return
    # Draw right arm
    pygame.draw.line(window, darkGray, (displayWidth/2, 165),
(displayWidth/2+45, 150), 6)
    if lives == 2: return
    # Draw left leg
    pygame.draw.line(window, darkGray, (displayWidth/2-25, 275),
(displayWidth/2, 240), 8)
    if lives == 1: return
    # Draw right leg
```

```
    pygame.draw.line(window, darkGray, (displayWidth/2, 240),
(displayWidth/2+25, 275), 8)
    displayText("Izgubili ste!", red, int(displayWidth/20),
displayWidth/2, displayHeight/1.8, window)
    displayText("Riječ: " + word, darkBlue, int(displayWidth/20),
displayWidth/2, displayHeight/1.5, window)
    displayText("Pritisnite space za novu igru", darkBlue,
int(displayWidth/24), displayWidth/2, displayHeight/1.3, window)

# Game loop
run = True
while run:

    # Events
    for event in pygame.event.get():
        # Quit
        if event.type == pygame.QUIT:
            run = False
        # Pressed any key
        if event.type == pygame.KEYDOWN:
            if event.key == pygame.K_SPACE:
                window.fill(white)
                # Lives left
                lives = 6
                # Reset game
                word = random.choice(words)
                length = len(word)
                print(word, length)
                # Is character opened
                opened = [False] * length
                # Display empty word
                text = ' '.join('_' * length)
                displayText(text, darkGray, int(displayWidth/16),
displayWidth/2, displayHeight-100, window)
                drawHangingPole(window, displayWidth)
            # If key pressed is alphabet character
            if event.unicode.isalpha():
                # Check character
                correct = word.count(event.unicode.lower())
                if not correct:
                    lives -= 1
                for i in range(0, length):
                    if word[i] == event.unicode.lower():
                        opened[i] = True
                # Update word
                window.fill(white)
                drawHangingPole(window, displayWidth)
                drawStick(lives, window, displayWidth,
displayHeight)
```

```
                text = ' '.join(list(map(lambda x, i: x if opened[i]
else '_', word, range(0, length))))
                displayText(text, darkGray, int(displayWidth/16),
displayWidth/2, displayHeight-100, window)
                # Check winner
                countOpened = sum([1 if x else 0 for x in opened])
                if countOpened == length:
                    displayText("Pobjedili ste!", green,
int(displayWidth/20), displayWidth/2, displayHeight/1.7, window)
                    displayText("Pritisnite space za novu igru",
darkBlue, int(displayWidth/24), displayWidth/2, displayHeight/1.4,
window)

    pygame.display.update()
    pygame.time.delay(5)

pygame.quit()
```