I. osnovna šola Celje
Vrunčeva ulica 13
3000 Celje

# Tic-tac-code

Področje računalništvo

RAZISKOVALNA NALOGA

Avtor:
Rudi Rodič

Mentor:
Žan Močivnik, prof. mat. in rač.

Celje, april 2024

### *Zahvala*:

Zahvaljujem se vsem, ki so pripomogli k nastanku te raziskovalne naloge. Pri raziskovalni nalogi mi je bil v največjo pomoč mentor Žan Močivnik, ki mi je je bil vedno na voljo za morebitna vprašanja, za kar sem mu zelo hvaležen. Zahvaljujem se tudi vodstvu I. osnovne šole Celje, ki mi je omogočilo raziskovanje.

## *POVZETEK*

V nalogi sem združil področje računalništva in umetne inteligence ter sam in s pomočjo umetne inteligence sprogramiral igro križci in krožci v programu Python. Namen raziskovalne naloge je bil raziskati, ali je umetna inteligenca sposobna zgolj enostavnejšega programiranja ali pa je sposobna tudi težjega programiranja in razmišljanja ter ustvarjanja zmagovalnih strategiji in potez. V teoretičnem delu sem pisal o sami zgodovini Pythona in računalniških iger, o razvoju lastnega programa, o pristopu umetne inteligence k programiranju, nato sem še analiziral, katera je bila pri nalogi uspešnejša/učinkovitejša. Celotna raziskovalna naloga je bila narejena s pomočjo Pythona, ki je dostopen na Windowsih. Najtežji del naloge pa je bilo analiziranje učinkovitosti programov obeh umetnih inteligenc.

**Ključne besede:** Python, križci in krožci, terminal, knjižica, umetna inteligenca, računalniška igra.

## *Kazalo vsebine*

## *Kazalo slik*

# 1 Uvod

## 1.1 Opis raziskovalnega problema in namen

Namen raziskovalne naloge je bil raziskati, ali je umetna inteligenca sposobna zgolj enostavnejšega programiranja ali pa je sposobna tudi težjega programiranja in razmišljanja ter ustvarjanja zmagovalnih strategiji in potez.

## 1.2 Raziskovalno vprašanje

Umetna inteligenca, ki je prosto dostopna, je sposobna enostavnejšega programiranja in generiranja programov za igre, a se pri težjih nalogah brez dodatnih navodil zmede, saj še ni sposobna samostojnega, težjega programiranja in generiranja kode, ki bi vsebovala zmagovalni algoritem.

## 1.3 Raziskovalne metode

Pri izdelavi raziskovalne naloge sem uporabljal naslednje metode dela:

- sistematični pregled literature,
- metoda možganske nevihte,
- eksperiment,
- metoda študija primera.

Sistematični pregled literature – metoda, s katero sistematično preverimo raziskovalno področje in primerjamo svoje ugotovitve. Pri tej metodi sem preveril zgodovino umetnih inteligenc, video iger in igre križci in krožci.

Metoda možganske nevihte – pripomogla je k raziskovanju različnih problemov in idej, kako igro pretvoriti v računalniški program. Pri tej metodi sem želel igro križcev in krožcev pretvoriti v računalniški jezik.

Eksperiment – metoda, kjer zapišemo posamezni del programa in ga testiramo. Pri tem ugotavljamo vse možnosti in iščemo napake. Ta metoda je bila zelo pomembna, saj je potrebno pri razvijanju programa veliko testiranja. Prav tako vedno ne pridemo do rešitve v prvem koraku, zato lahko rečem, da je programiranje neke vrste eksperimentiranje.

Metoda študija primera – je metoda, kjer se analizira posamezen košček kode in poskuša razumeti, kako vpliva na delovanje programa. Po analizi programa, ki preverja za zmago, sem ugotovil, da lahko kodo za preverjanje, ali so znaki diagonalno med seboj enaki, uporabim tudi pri programih za preverjanje, ali so znaki enaki navpično in vodoravno.

## 1.4 Orodja in pripomočki

Za svoje raziskovanje sem potreboval sledeče programe.

Oblak 365 povezuje Arnes AAI s storitvami Microsoft 365. Učencem, učiteljem in drugim delavcem v izobraževalnih organizacijah omogoča hiter dostop do različnih storitev, skrbnikom pa enostavnejše upravljanje s temi dostopi in licencami.

Office 365 je okolje za urejanje dokumentov, ki ga je razvilo podjetje Microsoft. Paket vsebuje urejevalnik besedil, planer, izdelavo elektronskih prosojnic, shrambo dokumentov in drugo. Pripomoček sem uporabil za pisanje raziskovalne naloge in odlaganje dokumentov.

OneDrive je Microsoftova storitev v oblaku, ki vas poveže z vsemi vašimi datotekami. Omogoča shranjevanje in zaščito vaših datotek, njihovo skupno rabo z drugimi in dostop do njih od koder koli v vseh vaših napravah.

Word je orodje za ustvarjanje in urejanje dokumentov, ki omogoča enostavno skupno rabo in sodelovanje pri dokumentih ter del paketa Oblak 365.

Google Messages je Googlova digitalna aplikacija za pošiljanje in prejemanje besedilnih sporočil, fotografij, glasovnih sporočil in videov.

XMind je programska oprema za miselne zemljevide in možgansko nevihto. Razvilo jo je podjetje XMIND LTD, registrirano v Hong Kongu, leta 2007. Aplikacijo je mogoče uporabiti za vizualizacijo idej, razjasnitev misli, upravljanje kompleksnih informacij in spodbujanje timskega sodelovanja.

Visual Studio Code je poenostavljen urejevalnik kode s podporo za razvojne operacije, kot so odpravljanje napak, izvajanje opravil in nadzor različic.

Python je računalniški programski jezik, ki se pogosto uporablja za izdelavo spletnih mest in programske opreme, avtomatizacijo opravil in izvajanje analize podatkov.

# 2 Teoretični del

V teoretičnem delu sem se posvetil predvsem sami zgodovini programskih jezikov, računalniških iger, igri križci in krožci ter opisal postopek razvijanja lastnega programa in programa dveh AI-jev in analiziral, kateri je bil učinkovitejši, hitrejši, predvsem pa najzanesljivejši.

## 2.1 Zgodovina

### 2.1.1 Zgodovina računalniških iger

Računalniška igra je računalniški program, ki ga uporabljamo za razvedrilo, učenje in zabavo. Na voljo je veliko raznovrstnih računalniško nadzorovanih iger, pri katerih je računalnik lahko nasprotnik človeškemu igralcu. Računalniške igre običajno uporabljajo hitro animirano grafiko na zaslonu in sintetiziran zvok.

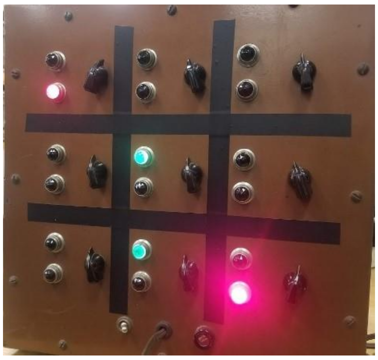
Video igre lahko razdelimo v več kategorij, te so:

- Strategija
- Avantura
- Simulacija
- Arkada

*Slika 1: Primer strateške igre.*

*Slika 2: Primer avanturistične igre.*

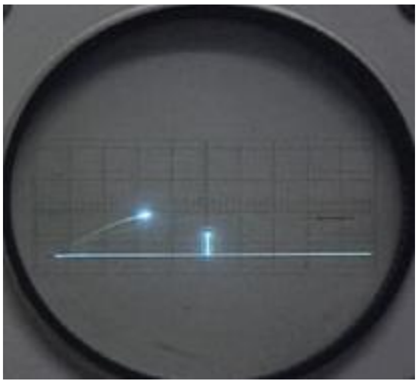
*Slika 3: Primer simulaciske igre.*



*Slika 4: Prva Igra: "Tic-tac-toe".*

Sam začetek video iger sega že v 19. stoletje, ko je angleški filozof, matematik in računalnikar Charles Babbage trdil, da lahko njegov mehanski računalnik igra igro križcev in krožcev. Zamisel se je uresničila šele čez slabo stoletje okoli leta 1952, na Illiacu, enemu prvih elektronskih računalnikov. Dejstvo, da je tehnologija prišla do takšne razvitosti, da lahko v igri premaga človeka, je povzročilo večje zanimanje in navdušuje za računalnike.

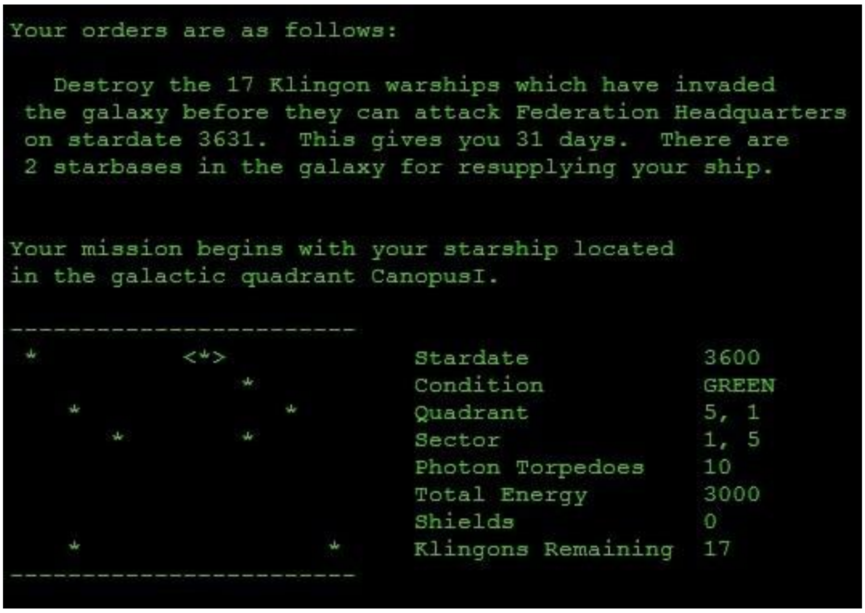
Veliko ljudi pa še vedno trdi, da je bila prva »računalniška« igra razvita leta 1958 v Brookhavenskem narodnem laboratoriju, ameriškem inštitutu za atomske raziskave, kjer je William Higinbotham izdelal igro Tenis for Two. To mu je uspelo tako, da je na osciloskop (naprava, ki prikazuje časovni potek električne napetosti) priključil analogni računalnik Donner in ustvaril računalniški tenis.



*Slika 5: igra: "Tenis for two".*

Tri leta pozneje so Wayne Wutanen, Martin Graetz in Stephen R. Russel izdelali prvo vesoljsko strelsko igro za dva igralca, imenovano Spacewar!. Igralec je moral pri igri le udarjati po tipkah; pozneje je bila igra izboljšana z oteževanjem manevriranja vesoljskih raket. Igro Vesoljska vojna so javno predstavili maja 1962, ob dnevu odprtih vrat na Tehnološkem inštitutu Massachusettsa. Seveda pa se tu proizvodnja in razvoj računalniških iger šele začne in v kasnejših letih nastane še veliko »klasičnih« video igric, ena izmed njih je Spacewar!, ki je bila popularna video igra, a sredi šestdesetih let prejšnjega stoletja marsikateri računalnik še vedno ni imel monitorja, temveč le tiskalniški terminal. Zaradi tega so nastale številne tako imenovane tekstovne igre, prva med njimi pa je bila vesoljska strelska igra, nekakšen Spacewar! prelit v svet tiskalniškega izpisa; z nekaj spremembami seveda in poimenovana po televizijski nadaljevanki Zvezdne steze (Star Trek), ki je bila med programerji zelo priljubljena. Igralec je vodil vesoljsko ladjo Enterprise v vesolju, sestavljenem iz 8 x 8 "kvadrantov", v njem je moral odkriti klingonsko floto in jo uničiti.

Star Trek je nastal sredi šestdesetih let, verjetno leta 1966 ali 1967, in se hitro širil med takratnimi računalniki. Prilagodili so ga različnim modelom, marsikateri programer pa ga je mimogrede še predelal, dopolnil ali prilagodil svojemu okusu. Deset let po nastanku je kot Super Star Trek postal zelo priljubljen tudi na prvih osebnih računalnikih.



```
Your orders are as follows:

   Destroy the 17 Klingon warships which have invaded
 the galaxy before they can attack Federation Headquarters
 on stardate 3631.  This gives you 31 days.  There are
 2 starbases in the galaxy for resupplying your ship.


Your mission begins with your starship located
in the galactic quadrant CanopusI.

------------------------
 *         <*>           Stardate            3600
              *          Condition           GREEN
   *             *       Quadrant            5, 1
      *       *          Sector              1, 5
                         Photon Torpedoes    10
                         Total Energy        3000
                         Shields             0
   *                *    Klingons Remaining  17
------------------------
```

*Slika 6: Slika klasične video igrice SpaceWar!.*

### 2.1.2 Zgodovina umetnih inteligenc

Ideja o ustvarjanju umetne inteligence (AI) sega že v antiko. Aristotel je že v 4. stoletju pred našim štetjem razmišljal o tem, ali bi lahko stroji razmišljali in se obnašali kot ljudje. V 17. stoletju je René Descartes predlagal, da bi lahko stroji delovali tako, da bi sledili pravilom logike.

***Zgodnji začetki (1950–1960)***

Prvi dejanski poskusi ustvarjanja AI so se začeli v 20. stoletju. Leta 1950 je Alan Turing objavil članek z naslovom "Izračunljivost strojev in inteligenca", v katerem je postavil Turingov test, ki je še danes eden od temeljev raziskav AI.

V tem obdobju so bili postavljeni temelji za raziskovanje AI. Raziskovalci so začeli razvijati nove metode in tehnologije, ki so omogočile AI, da doseže prve uspehe. Med temi uspehi so bili:

- Razvoj šahovskih programov, ki so bili sposobni premagati človeške šahovske velemojstre.
- Razvoj programov za prepoznavanje obraza, ki so bili sposobni prepoznati obraze ljudi z visoko natančnostjo.
- Razvoj programov za strojno prevajanje, ki so bili sposobni prevajati besedila iz enega jezika v drugega.

***Zima AI (1970–1980)***

Leta 1970 je prišlo do obdobja stagnacije v raziskovanju AI. Raziskovalci so se soočali z različnimi izzivi, zaradi katerih so se njihova pričakovanja o AI zmanjšala. Med temi izzivi so bili:

- Pomanjkanje močnega računalniškega strojnega znanja.
- Težava pri ustvarjanju programov, ki bi lahko reševali resnične probleme iz resničnega sveta.

***Obdobje ponovne oživitve (1980–2000)***

Leta 1980 se je raziskovanje AI ponovno začelo razvijati. Raziskovalci so začeli uporabljati nove metode, kot so strojno učenje in nevronske mreže. V tem obdobju so bili doseženi nekateri novi dosežki, med njimi:

- Razvoj avtonomnih vozil, ki so sposobna voziti brez človeškega nadzora.
- Razvoj glasovnih asistentov, ki lahko razumejo in se odzivajo na človeški jezik.
- Razvoj velikih jezikovnih modelov, ki lahko ustvarjajo besedilo, prevajajo jezike in odgovarjajo na vprašanja.

***Moderna doba (2000–danes)***

V zadnjih letih se je raziskovanje AI še bolj pospešilo. Raziskovalci se osredotočajo na razvoj novih metod in tehnologij, ki bodo omogočile AI, da doseže še večje sposobnosti. Med temi novimi metodami in tehnologijami so:

- Umetne nevronske mreže, ki so sposobne učenja iz podatkov in so postale zelo uspešne pri številnih nalogah, kot so prepoznavanje obraza, strojno prevajanje in igranje iger.
- Veliki jezikovni modeli, ki so sposobni ustvarjati besedilo, prevajati jezike in odgovarjati na vprašanja.

- Umetna inteligenca za samovozeče avtomobile, ki je sposobna voziti brez človeškega nadzora.

*Slika 7: Logotip ChatGPT-ja.*

*Slika 8 Logotip Google Barda.*

## 2.1.3 Križci in krožci

### *2.1.3.1 Opis igre*

Tic-Tac-Toe ali nam bolj znani križci in krožci je klasična igra, ki se običajno igra med majhnimi otroki in je znana po svojih preprostih pravilih.

### *2.1.3.2 Pravila igre*

1. Igrata jo dve osebi, ki izmenično označujeta X ali O na mreži 3x3.
2. Prva oseba, ki je vodoravno, navpično ali diagonalno označila tri svoje znake, je zmagovalka.

## 2.1.4 Zgodovina

Zgodovina igre Tic-Tac-Toe sega v antiko. Najstarejši dokazi o igri so bili najdeni v Egiptu, kjer so jo igrali že pred 5000 leti. Igra je bila znana tudi v drugih starodavnih civilizacijah, kot so Rimljani, Kitajci in Indijci.

Ime “Tic-Tac-Toe” ni bilo uporabljeno do poznega osemnajstega stoletja v Angliji, kjer še danes ugotavljajo, da je bilo ime prvič uporabljeno za opis popolnoma drugačne igre, pred tem pa je igra imela veliko različnih imen, kot sta "kralj polj" in "trikotniki".

Poleg tega to ni bila le igra s papirjem in svinčnikom, ampak je bila tudi ena prvih video iger, ki jo je posebej razvil Charles Babbage leta 1952.

# 3 Osrednji del

## 3.1 Okolje Python

Python je računalniški programski jezik na visoki ravni, cenjen zaradi podobnosti z angleškim jezikom in zmogljivih vgrajenih funkcij in knjižnic za analizo podatkov.

Nizozemski programer Guido van Rossum je leta 1991 razvil Python, potem ko je izrazil razočaranje nad omejitvami programskega jezika ABC. Python, ki ga je poimenoval po britanski televizijski seriji Leteči cirkus Montyja Pythona, je bil javno izdan leta 1994. Čeprav je Van Rossum ostal globoko vpleten v razvoj Pythona do leta 2018, je igra močno prispevala tudi k skupnosti drugih razvijalcev.

Pythonova odprtokodna, nelastniška narava je imela ključno vlogo pri njegovem hitrem razvoju in naraščajoči priljubljenosti. Ogromna skupnost programerjev prostovoljcev je skupaj razvila na tisoče modulov, ki so močno razširili zmogljivosti Pythona.

Doživel je tri primarne evolucije: Python 1.0, izdan leta 1994; Python 2.0, leta 2000; Python 3.0, leta 2008. Predvsem Python 3.0 ni združljiv s starejšimi različicami. Do začetka leta 2020 je Python postal eden najpogosteje uporabljanih programskih jezikov po vsem svetu.

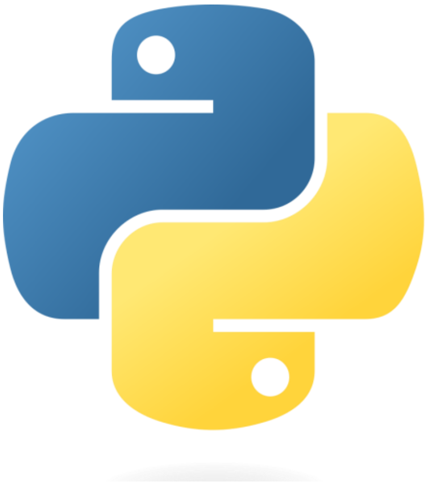
Python Software Foundation opisuje Python kot "interpretiran, objektno usmerjen programski jezik na visoki ravni z dinamično semantiko." Za razliko od jezikov, kot je Java, je Python interpretiran jezik, kar pomeni, da je njegovo izvorno kodo mogoče neposredno uporabiti in izvesti brez potrebe po prevajalniku. Python je tudi objektno usmerjen jezik, v nasprotju s funkcionalnimi programskimi jeziki, kot je C. Objektno usmerjeni jeziki oblikujejo programsko opremo okoli predmetov, ki so lahko entitete iz resničnega sveta, kot so avtomobili, ali abstraktni koncepti, kot so številke. Poleg tega je Python opredeljen kot visokonivojski programski jezik (v nasprotju z nizkonivojskimi jeziki, kot je sestavljanje, npr. Scratch).

Priljubljenost Pythona se je povečala deloma zaradi njegove jasne in jedrnate sintakse, ki izboljša berljivost. Python poudarja jasnost kode in spodbuja občutek elegance. Čeprav pa je Python predvsem objektno usmerjen jezik, ga je mogoče uporabiti tudi za proceduralne ali funkcionalne aplikacije.

Python kot jezik, ki uporablja več platform, omogoča izvajanje aplikacij v različnih računalniških sistemih, kot so Windows, macOS in Linux, ne da bi jih bilo treba prevesti. Podobno, čeprav imajo knjižnice Python drugih proizvajalcev nekatere funkcionalnosti, specifične za platformo, na splošno ostanejo uporabne na različnih platformah. Pythonova standardna knjižnica omogoča širok spekter aplikacij, od prenosa datotek z interneta do nastavitve internetnega strežnika posameznika, brez uporabe knjižnic tretjih oseb. Hkrati je raznolikost razpoložljivih knjižnic tretjih oseb Python uvedla v več domen, od spletnega razvoja do znanstvenih raziskav.

### 3.1.1 Zgradba Pythona

Program je sestavljen iz zaporedja ukazov. Navadno pišemo en ukaz v eno vrstico. Ukaz se mora praviloma začeti v prvi koloni. Izjeme so opisane v nadaljevanju. Konec vrstice kaže, da je ukaza konec. V eno vrstico lahko pišemo več ukazov, če jih ločimo s podpičjem (;). Dolg ukaz lahko pišemo v več zaporednih vrstic, če na koncu vsake vrstice pišemo poševnico (\). Nekateri ukazi so sestavljeni iz glave in telesa. Glava je ukaz, ki se konča z dvopičjem (:). Telo je zaporedje ukazov, ki so vsi zamaknjeni v desno za isto število presledkov, najbolje za štiri. Sestavljeni ukazi lahko vsebujejo druge sestavljene ukaze. Ti so zamaknjeni še globlje. Vse, kar v vrstici sledi znaku #, Python ignorira. To je komentar za vzdrževalce programa.



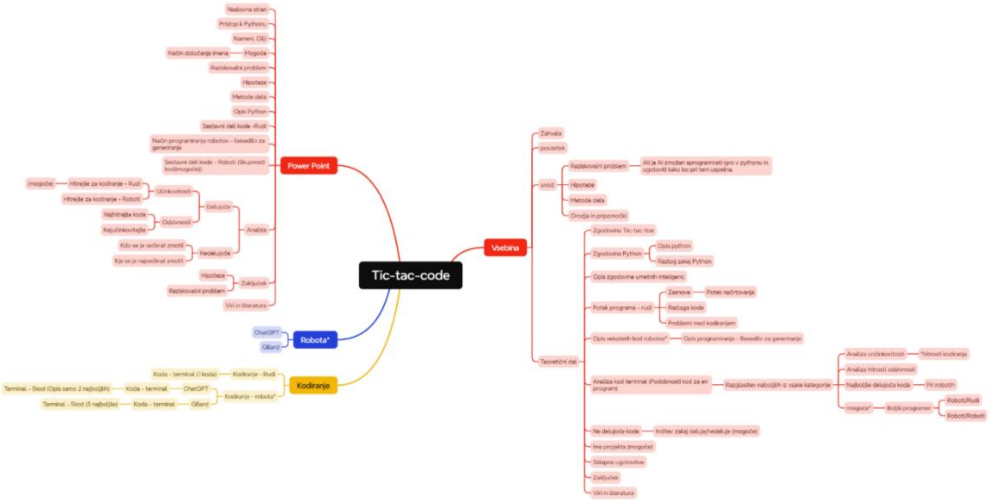
Slika 9: Logotip Pythona.

Slika 10: Igriv logotip Pythona.

## 3.2 Razvoj

### 3.2.1 Zasnova ideje

Preden razložim postopek razvoja ideje, bi bilo treba povedati, da se je raziskovalna naloga spremenila, kar trikrat, saj sem znova in znova ugotavljal, da je tema, ki sem jo izbral, zelo obsežna in sem jo moral spremeniti, vključno z raziskovalnim problemom. Sama ideja za to raziskovalno nalogo se je začela z nenadnim navdušenjem nad dvema umetnima inteligencama (ChatGPT in Google Bard). Medtem ko sem poslušal različne pogovore o različnih temah, sta na plano vedno znova prišli ti dve umetni inteligenci, zato sem se začel zanimati, katera izmed njiju je bolj razvita oziroma katera je boljši (hitrejši/učinkovitejši) pomočnik za programiranje in katera je namenjena le generiranju spisov, odgovarjanju na vprašanja in pomoči pri delanju domačih nalog. Na to vprašanje sem želel odgovoriti s to raziskovalno nalogo, ker pa sem se med letom začel zanimati za programski jezik Python, sem se odločil, da bom na raziskovalni problem odgovoril prav s pomočjo Pythona. Ostal je le še en problem – kako bom preizkusil, ali sta dve najznamenitejši umetni inteligenci sposobni programirati. Da bi dobil odgovor, sem se odločil, da bom obema inteligencama podal enak poziv oziroma enake informacije za razvoj kode, vse kode analiziral, preveril učinkovitost in odzivnost in jih nato primerjal z lastnim programom. Vsi programi so bili narejeni oz. preverjeni v programu VS Code.



Slika 11: Zamisel sestave RN.

Z določenim raziskovalnim vprašanjem sta pred mano stali še dve izbiri, in sicer kateri program bom uporabil in kako oziroma s čim ga bom oblikoval. Za začetek bi lahko uporabil veliko različnih programskih jezikov, kot so Java/JavaScript, C/C+/C++ in Python, a mi je med vsemi različnimi jeziki izstopal predvsem Python, saj je eden izmed enostavnejših jezikov, ki utrdi znanje za nadgradnjo sposobnosti programiranja, je uporabljen na več različnih področjih (razvoj spletnih mest in programske opreme, avtomatizacija opravil, analiza podatkov in vizualizacija podatkov) in za veliko različnih projektov v večjih podjetjih, kot so Google, Microsoft in Meta. Z izbranim programskim jezikom pa je bilo treba le še izbrati, kako bom oblikoval kodo. Na izbiro sem imel dve različni opciji, in sicer ali bo program deloval s pomočjo ene od veliko različnih knjižic ali drugače bolj strokovno pravilno imenovanih modulov (v tem primeru bi uporabil knjižico Turtle*), ki bi omogočili vizualno privlačnejši končni izdelek (s tem tudi slabše pregledno kodo), višji standard znanja o programiranju, težjo analizo, počasnejši odzivni čas in večjo možnost za napake pri generiranju oziroma izvajanju kode, ali pa na drugi strani kodo, ki ne bi uporabila knjižic (modulov) in bi delovala v konzoli, to bi pomenilo manj privlačen končni izdelek, a s preglednejšo kodo, ki omogoča lažjo analizo, hitrejši odzivni čas in lažje generiranje z manjšo možnostjo za napake.

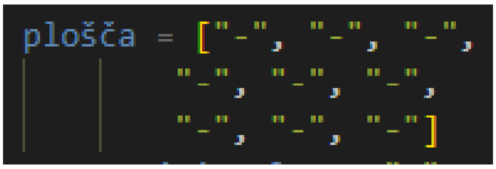
### 3.3 Zasnova ideje programa

Delovanje programa križci in krožci sem si zamislil tako, da bi igralec pritisnil eno od številk od 1 do 9, odvisno od številke bi se na tistem polju pojavil X, ki bi bil oznaka igralca s prvo potezo. Ko bi igralec določil svojo prvo potezo, bi naslednjo določila umetna inteligenca, ki bi se z algoritmom, ki bi ji ga dodal oz. določil s programiranjem, odločila za svojo potezo. Tako bi se igra nadaljevala, dokler ne bi igralec ali umetna inteligenca dobila tri v vrsto, torej vodoravno, navpično ali postrani. Ko bi se ta pogoj izpolnil, bi program določil zmagovalca, ga oznanil in zaključil igro.

Za vsako igro seveda stoji koda, ki predstavlja ključni del raziskovalne naloge in bi mi kasneje omogočila primerjavo med kodami umetnih inteligenc (AI). Torej sam program bi bil sestavljen iz petih glavnih delov (kategorij), ki bi jih kasneje »razdrobil« na manjše delce programa (podkategorije). Glavne kategorije bi bile uvoz knjižic*, definiranje funkcij, definiranje polja, pregled nad zmago/izenačenjem, sprejemanje igralčevih vnosov, menjava potez in umetna inteligenca.
(*V tem primeru knjižica nima enakega pomena kot že omenjena in služi le kot del umetne inteligence.)

## 4 Začetek programiranja

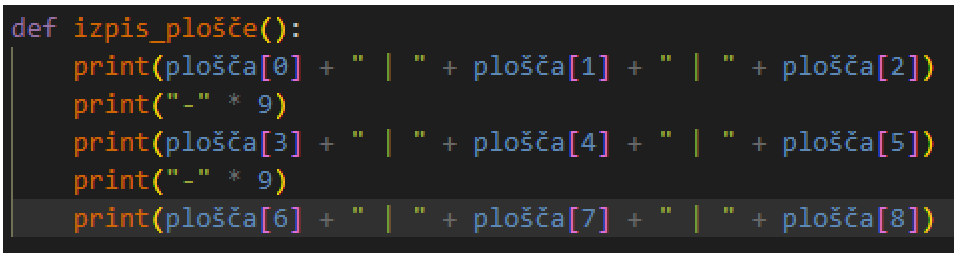
Ker sem že imel zamisel, kako bo izgledal program, sem se odločil, da s kodiranjem začnem pri ustvarjanju in definiranju funkcij. Prva funkcija, ki sem jo definiral, je bila funkcija *plošča*, ki ne bi nikoli spremenila »vrednosti«. Ta funkcija služi kot celotno polje igre. Ker na polja še nismo postavili X oziroma O, jih označi z znaki, da pa bi bila prazna polja bolje vidna, sem uporabil znak »-«. Medtem ko igra napreduje, se polja zapolnijo, dokler ne pride do zmage ali izenačenja. Ker pa je ta funkcija le seznam, potrebujem še kodo, ki bo narisala polje in vanj vstavljala polja iz te funkcije.



```
plošča = ["-", "-", "-",
          "-", "-", "-",
          "-", "-", "-"]
```

*Slika 12: Definiranje polja.*

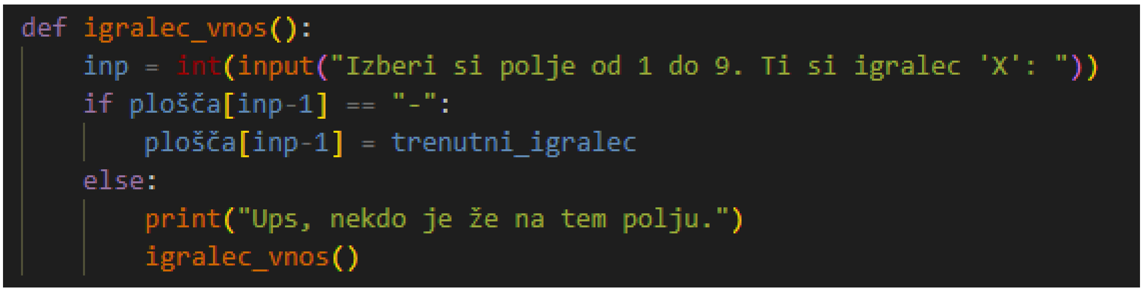
Kot omenjeno, funkcija *plošča* ne more delovati brez dodatne funkcije, ki bi narisala celotno podlago in vanjo vstavila polja, zato sem se odločil, da bom naredil funkcijo, ki bo naredila prav to. Delovanje igre v konzoli je predstavljalo manjšo težavo, saj bi moral program po vsaki potezi narisati novo igralno polje, kar bi program lahko upočasnilo, zato sem se odločil, da bom funkcijo *polje* klical le, po tem ko potezo izvede igralec in računalniški nasprotnik. Program sem začel z definiranjem funkcije *nariši polje*, nato sem vanjo vstavil kodo, ki nariše celotno igralno podlago in jo zapolni s polji (začne s prvim poljem, nato ga loči z drugim in nato vstavi drugo polje itd.) iz funkcije *polje* (pri polnjenju polj z znaki iz funkcije *polje* je bilo treba dodati številko, saj sem drugače naletel na napako opisano v inženirskem dnevniku). Pri definiranju tabele sem ponovno naletel na problem. Ker je program deloval v konzoli, ga nisem mogel oblikovati po lastni volji, ampak sem moral uporabiti znaka, kot sta |, da sem ločil stolpce in -, da sem ločil vrstice.



```python
def izpis_plošče():
    print(plošča[0] + " | " + plošča[1] + " | " + plošča[2])
    print("-" * 9)
    print(plošča[3] + " | " + plošča[4] + " | " + plošča[5])
    print("-" * 9)
    print(plošča[6] + " | " + plošča[7] + " | " + plošča[8])
```

*Slika 13: Koda za risanje igralne plošče.*

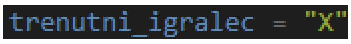
Ker sem imel narejeno funkcijo za risanje tabele, sem se odločil, da bom naredil kodo, ki bo sprejemala igralčeve napotke, kam naj postavi X (med programiranjem sem se odločil, da bo vedno prvi na vrsti igralec, ki bo igral z znakom X), nato naredila novo igralno polje in na mesto, ki se ujema z igralčevim vnosom (številko), vnesla X. Kodo za sprejemanje sem začel z definiranjem funkcije *igralec_vnos*, to sem razdelil na dve možni izvedbi. Prvo možno izvedbo sem naredil tako, da sem igralca najprej povabil, naj si izbere eno od številk od 1 do 9, nato sem dal pregledati, kam bi igra postavila igralčev vnos-1 na polju (»-1« je v tej funkciji nujno potreben, saj takoj ko Python zazna, da je funkcija sestavljena iz več delov, zabeleži prvi del z nič in ne ena, zato bi se v primeru, da bi bil igralčev vnos enak devet, ves program sesul, saj ne bi vedel, kaj naj naredi, ko sprejme številko 9). V primeru, da bi bilo polje prazno, bi program postavil X in zamenjala potezo na računalnik. Drugo izvedbo sem namenil preverjanju možnosti, ali bi bil igralčev vnos enak polju, v katerem je že postavljen X oziroma O. Če bi do tega prišlo, bi igra igralcu povedala, da je to polje zasedeno in ta funkcija bi se ponovno izvedla.



```python
def igralec_vnos():
    inp = int(input("Izberi si polje od 1 do 9. Ti si igralec 'X': "))
    if plošča[inp-1] == "-":
        plošča[inp-1] = trenutni_igralec
    else:
        print("Ups, nekdo je že na tem polju.")
        igralec_vnos()
```

*Slika 14: Koda za funkcijo igralec_vnos.*

Med programiranjem kode za sprejemanje igralčevih vnosov sem opazil, da potrebujem še funkcijo, ki bo igri povedala, čigava je prva poteza, in program za menjavanje poteze med igralcem in računalnikom, zato sem se odločil najprej vrniti k definiranju funkcij. Novo funkcijo sem se odločil dodati na vrh k funkciji *polje*, jo poimenoval *trenutni_igralec* in ji določil »vrednost« X, ki preostalim delom kode sporoča, da je prvi na vrsti igralec.

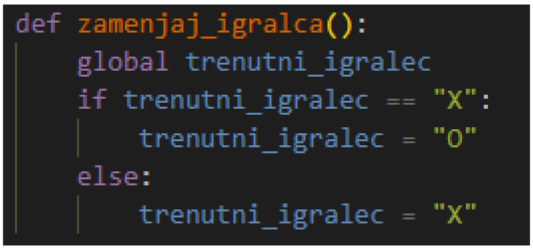


```
trenutni_igralec = "X"
```

*Slika 15: Trenutni igralec vstavlja vrednost X.*

Pri programiranju sem se že na samem začetku odločil narediti program, kjer je prvi na vrsti igralec in ne računalnik. Tako je igra bolj dinamična in od igralca zahteva, da med igro razmišlja strateško, omogoča pa tudi raznolikost med različnimi igralnimi potezami, potekom igre in končnim izidom. Med drugim to igralcu omogoča seznanitev s samim igralnim poljem in mehaniko igre, preden se sooči z računalniškim nasprotnikom. To, da je bolje, če igro začne igralec, potrjuje tudi primer, ki so ga poimenovali inteligentni papir (vir: http://vidra.si/inteligentni-papir/Papir.pdf). Iz algoritma je razvidno, da v primeru začetka z umetnim nasprotnikom nikoli ne pride do zmage igralca, saj program med pregledom polja izpolnjuje pogoje, ki odločijo oziroma določijo najboljšo potezo, ki bi nasprotniku omogočila zmago. Inteligentni papir igro nasprotnika opisuje tako, da s prvo potezo postavi krožec v kot, nato pa preveri, ali lahko v naslednji potezi postavi krožec v njemu nasproten kot (če to ni mogoče, ga postavi v kakšen drug kot). Po prvih dveh potezah računalniški nasprotnik preveri, ali lahko zmaga oziroma ali lahko blokira igralca. Če lahko zmaga, postavi krožec v to polje, sicer blokira igralca. V predzadnji potezi postavi krožec v polje, s katerim lahko zmaga ali blokira, v zadnji potezi (če do te poteze sploh pride) zapolni še zadnje polje. Tako inteligenten papir dokazuje, zakaj je smiselno, da z igro začne igralec. Ker pa je ta program še ne tako optimiziran, je vseeno, ali začne igralec ali računalnik. V naprednejših programih bi bila prva poteza nujno v »lasti igralca«, saj drugače igra ne bi bila zanimiva, ker bi vedno zmagal ali izenačil nasprotnik.

Po poimenovanju funkcije *trenutni_igralec* sem naredil kodo za menjavo potez med igralcem in računalnikom. To kodo sem začel z definiranjem nove funkcije, in sicer *zamenjaj_igralca*, znotraj funkcije sem naredil kodo, ki se izvede, ko igralec ali računalnik konča s potezo, takrat preveri, kdo je trenutno na vrsti in glede na ta podatek poda potezo nasprotniku (če sprejme informacijo, da je bila trenutna poteza igralčeva, poda potezo računalniku).

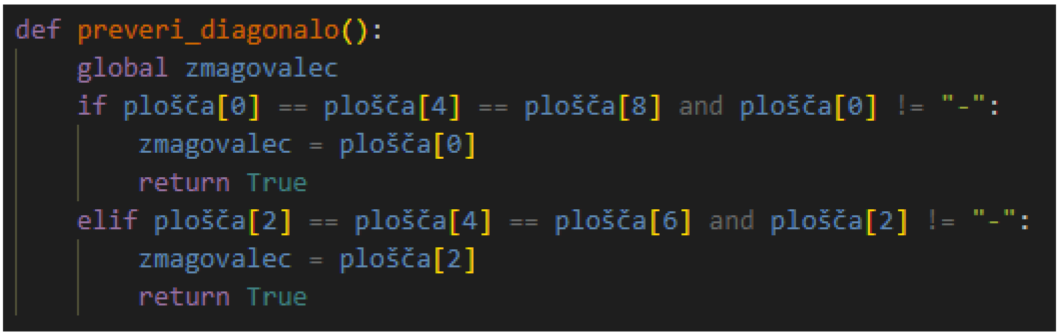


```
def zamenjaj_igralca():
    global trenutni_igralec
    if trenutni_igralec == "X":
        trenutni_igralec = "O"
    else:
        trenutni_igralec = "X"
```

*Slika 16: Menjava igralca v programu.*

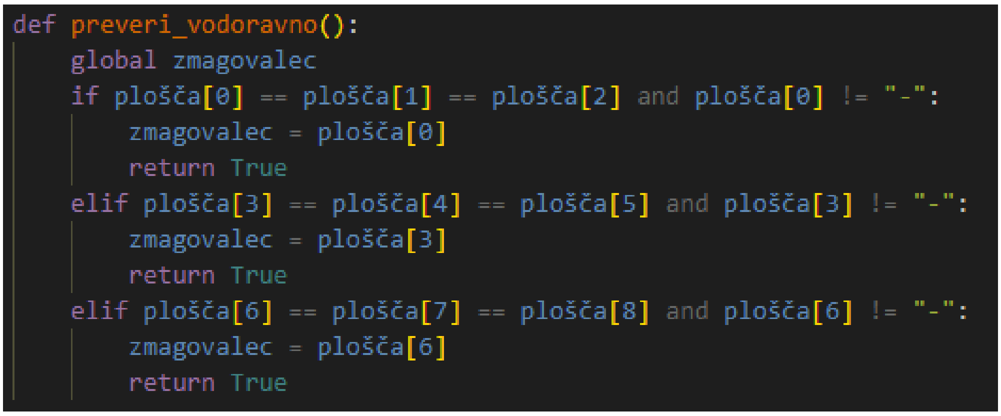
Ker sem imel narejen program za sprejemanje igralčevih vnosov, sem se odločil narediti program za ugotovitev, ali je igralec zmagal. Kodo za ugotavljanje zmage sem razdelil na eno glavno kategorijo (preveri_zmago_ali_izenačenje), nato pa glavno kategorijo na tri podkategorije preveri_diagonalo, preveri_vodoravno in preveri_navpično. S kodo sem začel pri kodah podkategorij. Prva podkategorija, ki sem jo začel kodirati, je bila preveri_diagonalo (razložil bom samo kodo za preveri_diagonalno, saj ostali podkategoriji delujeta na podoben način).

Kodo sem začel z ustvarjanjem funkcije, ki sem jo poimenoval preveri_diagonalno, po ustvarjanju funkcije sem umestil kodo, ki preveri, ali so vsa polja diagonalno zapolnjena in ali jih je zapolnila ista oseba (igralec ali računalnik). V primeru, da so vsi pogoji izpolnjeni, se funkcija zmagovalec spremeni na križec oziroma krožec (odvisno od znaka, ki ga je imel zmagovalec) ter izzove izvedbo glave kategorije preveri_zmago_ali_izenačenje in pove programu, da se je ta program zaključil.



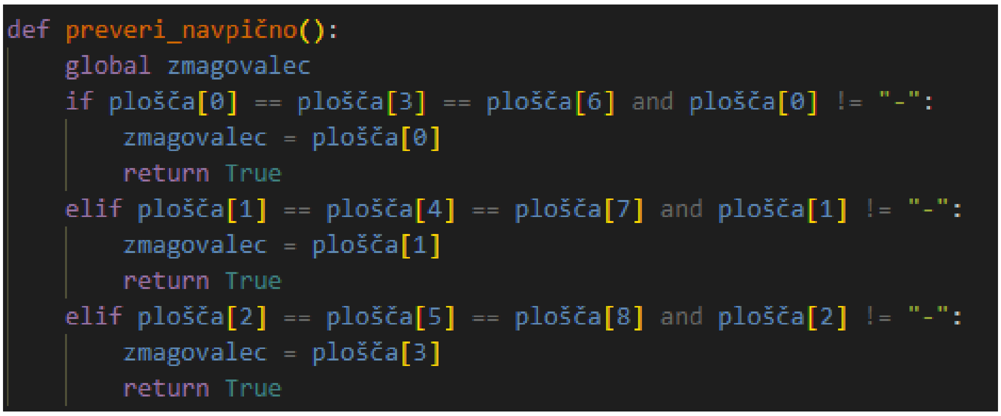
```
def preveri_diagonalo():
    global zmagovalec
    if plošča[0] == plošča[4] == plošča[8] and plošča[0] != "-":
        zmagovalec = plošča[0]
        return True
    elif plošča[2] == plošča[4] == plošča[6] and plošča[2] != "-":
        zmagovalec = plošča[2]
        return True
```

*Slika 17: Preveri zmagovalca po diagonali.*



```
def preveri_vodoravno():
    global zmagovalec
    if plošča[0] == plošča[1] == plošča[2] and plošča[0] != "-":
        zmagovalec = plošča[0]
        return True
    elif plošča[3] == plošča[4] == plošča[5] and plošča[3] != "-":
        zmagovalec = plošča[3]
        return True
    elif plošča[6] == plošča[7] == plošča[8] and plošča[6] != "-":
        zmagovalec = plošča[6]
        return True
```

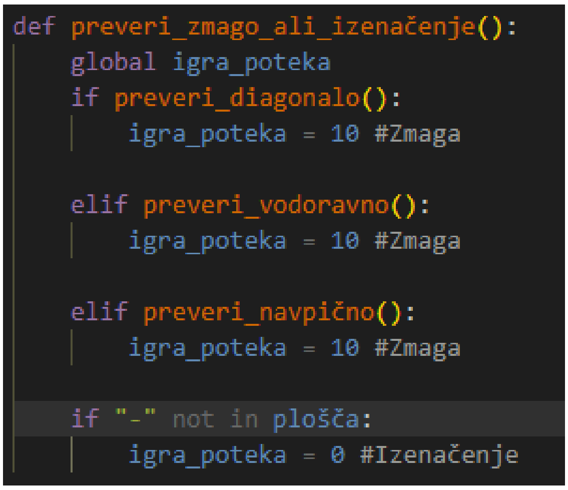
*Slika 18: Preveri zmagovalca vodoravno.*



```
def preveri_navpično():
    global zmagovalec
    if plošča[0] == plošča[3] == plošča[6] and plošča[0] != "-":
        zmagovalec = plošča[0]
        return True
    elif plošča[1] == plošča[4] == plošča[7] and plošča[1] != "-":
        zmagovalec = plošča[1]
        return True
    elif plošča[2] == plošča[5] == plošča[8] and plošča[2] != "-":
        zmagovalec = plošča[3]
        return True
```

*Slika 19: Preveri zmagovalca navpično.*

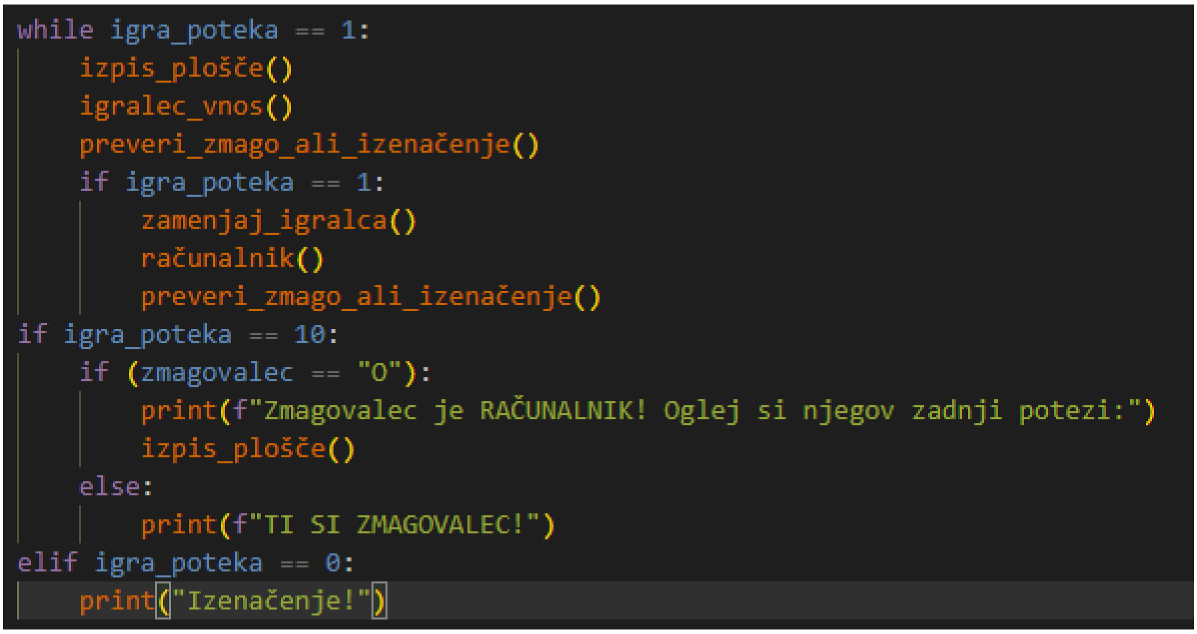
Izvedbe programov preveri_diagonalno/vodoravno/navpično izzovejo izvedbo programa preveri_zmago_ali_izenačenje. Koda za ta program je zelo enostavna, najprej definiram funkcijo preveri_zmago_ali_izenačenje, preverim, kateri izmed programov za preverjanje se je izvedel in sporočil programu, da je prišlo do zmage. Pri programiranju sem se odločil, da ne bo razlike med zmago vodoravno in zmago navpično; v primeru zmage koda vedno napiše samo »Zmagovalec je:«. V primeru, da pride do izenačenja, koda sporoči programu, da je prišlo do izenačenja.



```
def preveri_zmago_ali_izenačenje():
    global igra_poteka
    if preveri_diagonalo():
        igra_poteka = 10 #Zmaga

    elif preveri_vodoravno():
        igra_poteka = 10 #Zmaga

    elif preveri_navpično():
        igra_poteka = 10 #Zmaga

    if "-" not in plošča:
        igra_poteka = 0 #Izenačenje
```

*Slika 20: Program za preverjanje zmage oz. izenačenja.*

Po preverjanju, ali je nekdo zmagal ali je prišlo do izenačenja, koda preveri_zmago_ali_izenačenje in pošlje to informacijo naslednjemu programu. Tokrat sem se odločil, da to kodo ne bom umestil v posebno funkcijo, ampak sem jo vključil v kodo, ki skrbi, da se vsi programi izvajajo. To kodo sem začel s pogojem, ki preveri, ali se igra izvaja (to sem naredil z ustvarjanjem funkcije, ki na začetku igre sporoči temu programu, da se je igra začela). Ko je ta pogoj izveden, zažene prvi program, nato drugega in tako dalje, dokler ne pride do funkcije preveri_zmago_ali_izenačenje, tam dobi informacijo, da je oseba zmagala ali pa ni zmagala. V primeru, da je oseba zmagala, zažene kodo, ki napiše, da je oseba zmagala in nariše igralno polje (samo v primeru, da je zmagal računalnik). V primeru, da je prišlo do izenačenja, se zažene drugačen program, ki oznani izenačenje in igro zaključi. V primeru, da zmagovalec ni bila oseba, koda nadaljuje z izvajanjem programov, a tokrat preveri, ali je zmagal računalnik. Če zmaga računalnik, požene kodo, ki napiše, da je računalnik zmagal in nariše igralno polje. V primeru, da nihče ni zmagal in ni prišlo do izenačenja, se igra nadaljuje.



```
while igra_poteka == 1:
    izpis_plošče()
    igralec_vnos()
    preveri_zmago_ali_izenačenje()
    if igra_poteka == 1:
        zamenjaj_igralca()
        računalnik()
        preveri_zmago_ali_izenačenje()
if igra_poteka == 10:
    if (zmagovalec == "O"):
        print(f"Zmagovalec je RAČUNALNIK! Oglej si njegov zadnji potezi:")
        izpis_plošče()
    else:
        print(f"TI SI ZMAGOVALEC!")
elif igra_poteka == 0:
    print("Izenačenje!")
```

*Slika 21: Preverjanje zmagovalca.*

Po kodiranju programa za izvajanje programov sem se odločil narediti kodo računalniškega nasprotnika, a sem moral pred kodiranjem uvoziti knjižico oziroma modul random, ki predstavlja najpomembnejši del programa za računalniškega nasprotnika, saj se s to knjižico odloča, na katero polje bo postavil krožec.

```
import random
```

*Slika 22: Knjižica random.*

Po uvažanju knjižice random sem začel kodirati kodo računalnika, ki bo igral proti igralcu. Kodo računalniškega nasprotnika sem začel kakor ostale, naredil sem novo funkcijo in jo poimenoval računalnik. Naprej sem začel z ustvarjanje še ene funkcije, ki sem jo poimenoval odločitev in ji določil vrednost 99, nato sem naredil kodo, ki preleti polja in preveri, ali so prosta (če zazna, da je polje zasedeno, ga preskoči) ter vanj umesti O in preveri, ali bi lahko s to potezo zmagala. Če lahko z znano potezo zmaga in dobi informacijo, da lahko zmaga, tja postavi krožec in sporoči, da je zmagala, v primeru, da zazna, da s poljem ne more zmagati, ga spremeni nazaj v prazno. Ko izvede kodo za preverjanje, ali lahko zmaga, in od te kode ne dobi informacije, da lahko zmaga, zažene naslednji del kode, ki preverja, ali lahko zablokira igralca. Preverjanje tega, ali lahko zmaga računalniški nasprotnik, poteka podobno kot preverjanje, ali lahko zmaga, samo da tokrat ne umešča krožcev ampak križce in preverja, ali bi lahko s to potezo zmagal igralec. Če je to res, potem to polje zapolni s krožcem in prepreči, da bi zmagal igralec. Če koda računalniškega nasprotnika zazna, da igralec ali računalnik ne more zmagati ali blokirati, potem zažene zadnji del kode, ki služi le v sili in s pomočjo uvožene knjižice izbere naključno prazno polje in vanj vstavi krožec.

*Slika 23: Vstavljanje naključne pozicije.*

```
def računalnik():
    global igra_poteka
    odločitev = 99
    # Preveri, ali je možen zmagovalni potezi
    for i in range(9):
        if plošča[i] == "-":
            plošča[i] = "O"
            preveri_zmago_ali_izenačenje()
            if igra_poteka == 10:
                plošča[i] = "-"
                odločitev = i
            igra_poteka = 1
            plošča[i] = "-"

    # Preveri, ali je potrebno blokirati potezo nasprotnika
    if odločitev == 99:
        for i in range(9):
            if plošča[i] == "-":
                plošča[i] = "X"
                preveri_zmago_ali_izenačenje()
                if igra_poteka == 10:
                    plošča[i] = "-"
                    odločitev = i
                igra_poteka = 1
                plošča[i] = "-"

    # Sicer, izberi naključno potezo
    if odločitev == 99:
        odločitev = random.randint(0, 8)
        while plošča[odločitev] != "-":
            odločitev = random.randint(0, 8)

    plošča[odločitev] = "O"
    zamenjaj_igralca()
```

## 4.1 Razvoj programa umetnih inteligenc (AI)

V zaključni fazi programiranja sem se odločil svoje raziskovanje nadaljevati z umetno inteligenco. Naročil sem ji, naj mi generira 4 različne programe za igro križci in krožci, da pa bi bila raziskava pravična in bi imel natančnejše podatke, sem se za generiranje programov odločil vprašati dve različni najznamenitejši umetni inteligenci, Google Barda (Gemini) in ChatGPT. Pri razvoju umetnih inteligenc sem odločil, da bom obe umetni inteligenci prosil, naj generirata dva različna programa, za katera ne bi dodal veliko opisa programa, nato pa bi ju ponovno prosil za generiranje še dveh programov, a bi tokrat dodal podrobnejši opis stvari, pri katerih sta se zmotila (oba AI bom prosil za generiranje programov z enakim opisom). Analiziral bom velikost datoteke s kodo programov, dolžino programa, kako je umetna inteligenca optimizirala program, ali je AI pri generiranju programov dodal še manjše stvari oziroma delce kode, ki bi program izboljšale, ali bo ponavljala napake, ki bodo izpostavljene, kako pameten bo računalniški nasprotnik, kako hitro bo svojo potezo zaključil in ali bo umetna inteligenca sledila napotkom (pozivu) in popravila napake ali pa izpostavljenih napak niti ne bo spremenila oziroma dodala novih.

## 4.2 Google Bard (Gemini)

Preden začnem z generiranjem programa, je pomembno omeniti, da se je verzija Google Barda, ki sem jo uporabil med raziskovanjem in generiranjem programov, spremenila (2-krat) in se preimenovala v Gemini (kljub temu bom umetno inteligenco Gemini omenjal kot Bard oziroma Google Bard).

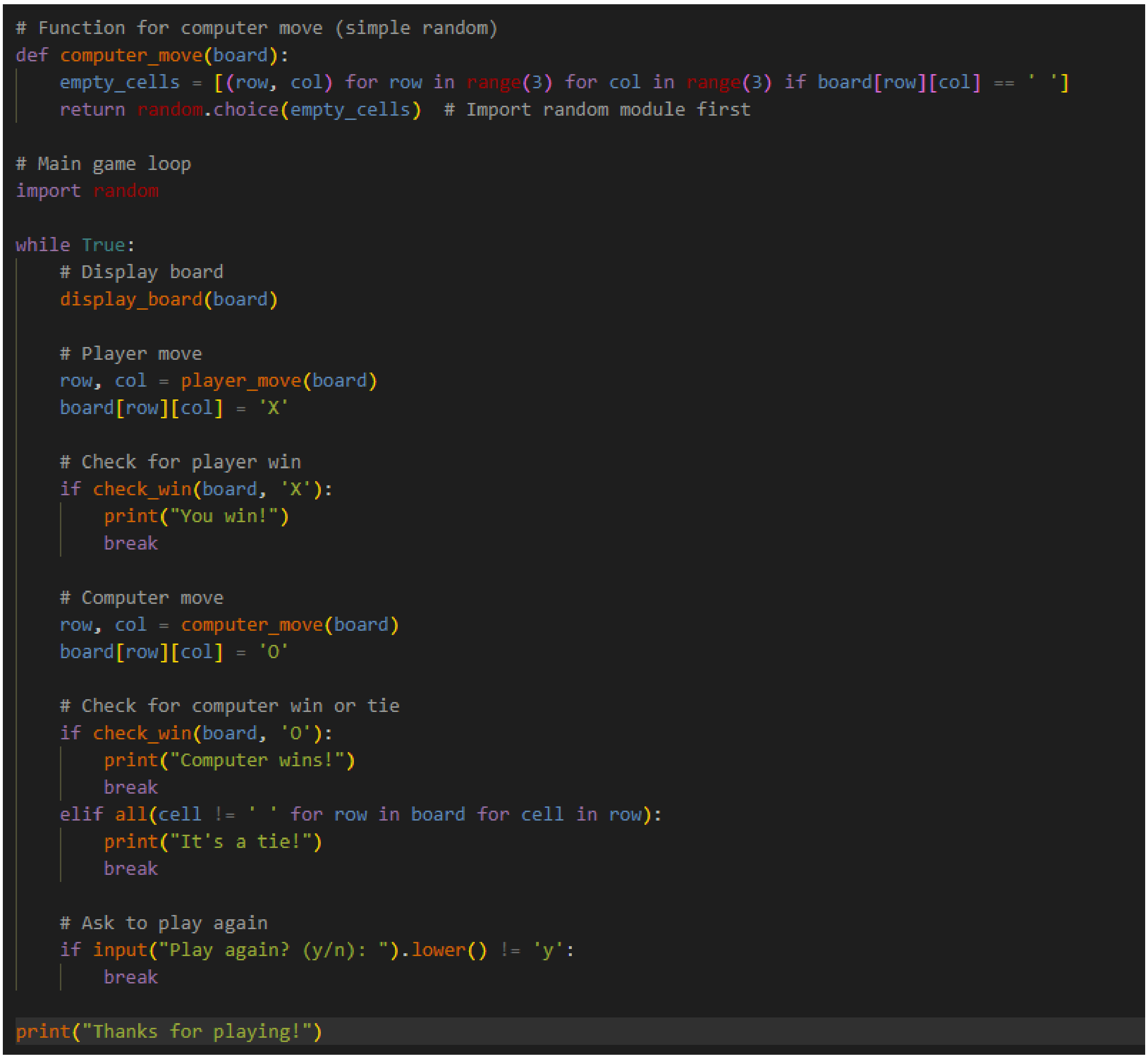
H glavnemu delu raziskovalne naloge sem se odločil pristopiti z umetno inteligenco (AI) Gemini (gemini.google.com). Za začetek sem obiskal njeno spletno stran in začel nov pogovor. V pogovorno okence sem začel pisati poziv, ki ga bom uporabil pri generiranju prvih dveh programov. Prva dva poziva sem se odločil narediti poenostavljena, v prvem bom umetni inteligenci povedal, kaj hočem in ji drugič le izpostavil večje napake (primer večje napake: polje ni vidno / nedelujoči računalniški nasprotnik). Poziv, ki sem ga vnesel, sem naredil v angleščini, saj sem od umetne inteligence hotel boljše rezultate (obe umetni inteligenci izhajata iz angleško govorečih držav in zato bolje razumeta angleški kot slovenski jezik). Poziv, ki sem ga napisal se ga odločil uporabiti za prvi poziv generiranja prvega programa, je naslednji:

*»Generate me a simple an easy Pyhon code for a game of tic-tac-toe that will run in console. Include things like computer "enemy" you will play against and system for getting player inputs.«*

Po vnosu poziva je Gemini potreboval nekaj trenutkov, a se hitro odzval in začel z generiranjem programa. Med generiranjem programa sem opazil že nekaj razlik in podobnosti med lastnim programom in programom Geminija. Program, ki ga je umetna inteligenca napisala je naslednji:

```
board = [[' ', ' ', ' '],
         [' ', ' ', ' '],
         [' ', ' ', ' ']]

# Define winning conditions
win_conditions = [
    [[0, 0], [0, 1], [0, 2]],
    [[1, 0], [1, 1], [1, 2]],
    [[2, 0], [2, 1], [2, 2]],
    [[0, 0], [1, 0], [2, 0]],
    [[0, 1], [1, 1], [2, 1]],
    [[0, 2], [1, 2], [2, 2]],
    [[0, 0], [1, 1], [2, 2]],
    [[0, 2], [1, 1], [2, 0]]
]

# Function to display the board
def display_board(board):
    for row in board:
        print("|", end="")
        for cell in row:
            print(f" {cell} |", end="")
        print("")
    print("---+---+---")

# Function to check if a player has won
def check_win(board, player):
    for condition in win_conditions:
        if all(board[i][j] == player for i, j in condition):
            return True
    return False

# Function for player input
def player_move(board):
    while True:
        row = int(input("Enter row (1-3): ")) - 1
        col = int(input("Enter column (1-3): ")) - 1
        if 0 <= row <= 2 and 0 <= col <= 2 and board[row][col] == ' ':
            return row, col
        else:
            print("Invalid move. Try again.")
```

*Slika 24: Kreiranje programa v okolju Bard (Gemini) – 1. del.*



```
# Function for computer move (simple random)
def computer_move(board):
    empty_cells = [(row, col) for row in range(3) for col in range(3) if board[row][col] == ' ']
    return random.choice(empty_cells)  # Import random module first

# Main game loop
import random

while True:
    # Display board
    display_board(board)

    # Player move
    row, col = player_move(board)
    board[row][col] = 'X'

    # Check for player win
    if check_win(board, 'X'):
        print("You win!")
        break

    # Computer move
    row, col = computer_move(board)
    board[row][col] = 'O'

    # Check for computer win or tie
    if check_win(board, 'O'):
        print("Computer wins!")
        break
    elif all(cell != ' ' for row in board for cell in row):
        print("It's a tie!")
        break

    # Ask to play again
    if input("Play again? (y/n): ").lower() != 'y':
        break

print("Thanks for playing!")
```

*Slika 25: Kreiranje programa v okolju Bard (Gemini) – 2. del.*

Program, ki ga je generiral Gemini, me je zelo presenetil, saj ni vseboval veliko večjih napak, koda pa je bila skoraj popolna za tako enostaven poziv. Če na kratko povzamem kodo:

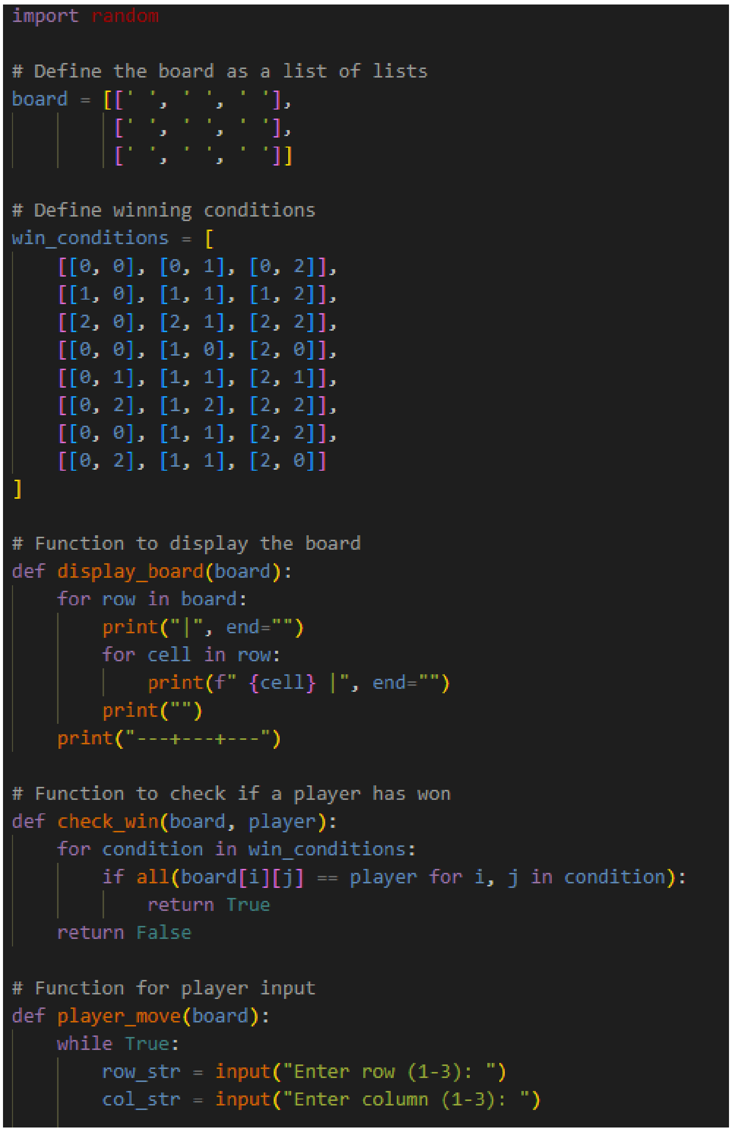
1. Program najprej naredi tablo za igro. Tablo predstavlja podlaga 3x3, kjer je vsaka celica na začetku prazna in se z vsako potezo napolni z križci in krožci.
2. Program nato pozove igralca, naj vnese svojo potezo. Igralec vnese vrstico in stolpec, kamor želi postaviti križ.
3. Program preveri, ali je igralčeva poteza veljavna. Če je poteza veljavna, program postavi križ na vneseno mesto.
4. Program nato preveri, ali je igralec zmagal. Če je igralec zmagal, program izpiše sporočilo o zmagi in se konča.
5. Če igralec ni zmagal, program izračuna potezo za računalnik. Računalnik izbere naključno prazno celico in vanjo postavi krožec.
6. Program nato preveri, ali je računalnik zmagal. Če je računalnik zmagal, program izpiše sporočilo o zmagi in se konča.
7. Če noben igralec ni zmagal, se program vrne na 2. korak.

Pri tem programu me je presenetila le ena stvar, in sicer to, da je Gemini naredil posebej kodo za naključno izbiro postavitve poteze računalniškega nasprotnika, namesto da bi uporabil modul random, ki ga je uvozil šele kasneje zato, da bi ga kasneje uporabil v programu za vprašanje igralcu, ali hoče še igrati. V celotnem programu se je pojavila le ena večja težava, in sicer da je igra igralca pred vsako potezo vprašala, ali hoče nadaljevati igro ali jo zaključiti.

Ko je Gemini končal z generiranjem prve kode, sem mu dodal nov vnos, da generira še eno kodo, a sem tokrat k pozivu dodal še nekaj napotkov za popravek kode. Nov poziv (v angleščini) sem naredil tako, da sem k staremu pozivu dodal le še nekaj napotkov za popravke večje težave, ki sem jo odkril, a pri tem nisem uničil preprostosti, ki jo je imel prvotni poziv:

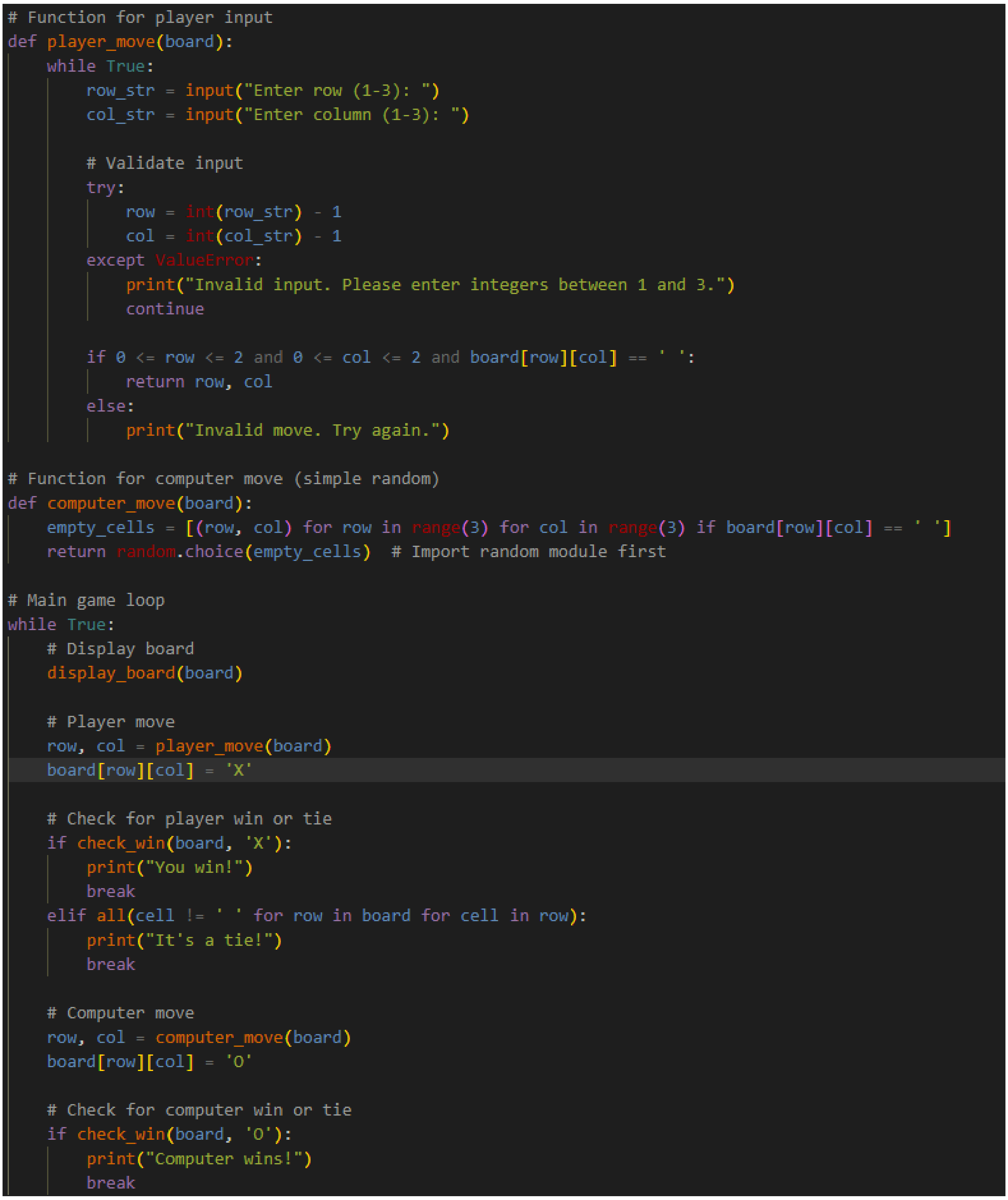
*»Generate me a simple an easy Pyhon code for a game of tic-tac-toe that will run in console. Include things like computer "enemy" you will play against and system for getting player inputs. But this time don't include a question do you want to continue the game?«*

Po vnosu tega poziva je Gemini potreboval očitno več časa za generiranje kode, a me kljub temu zelo presenetil s kodo, ki jo je generiral. Koda je bila hkrati podobna in hkrati različna od starejše (sliko kode bom umestil, a je ne bom razložil, saj so ključni deli kode enaki starejši).



```
import random

# Define the board as a list of lists
board = [[' ', ' ', ' '],
         [' ', ' ', ' '],
         [' ', ' ', ' ']]

# Define winning conditions
win_conditions = [
    [[0, 0], [0, 1], [0, 2]],
    [[1, 0], [1, 1], [1, 2]],
    [[2, 0], [2, 1], [2, 2]],
    [[0, 0], [1, 0], [2, 0]],
    [[0, 1], [1, 1], [2, 1]],
    [[0, 2], [1, 2], [2, 2]],
    [[0, 0], [1, 1], [2, 2]],
    [[0, 2], [1, 1], [2, 0]]
]

# Function to display the board
def display_board(board):
    for row in board:
        print("|", end="")
        for cell in row:
            print(f" {cell} |", end="")
        print("")
    print("---+---+---")

# Function to check if a player has won
def check_win(board, player):
    for condition in win_conditions:
        if all(board[i][j] == player for i, j in condition):
            return True
    return False

# Function for player input
def player_move(board):
    while True:
        row_str = input("Enter row (1-3): ")
        col_str = input("Enter column (1-3): ")
```

*Slika 26: Popravek kode po natančnejšem vnosu parametrov – 1. del.*



```python
# Function for player input
def player_move(board):
    while True:
        row_str = input("Enter row (1-3): ")
        col_str = input("Enter column (1-3): ")

        # Validate input
        try:
            row = int(row_str) - 1
            col = int(col_str) - 1
        except ValueError:
            print("Invalid input. Please enter integers between 1 and 3.")
            continue

        if 0 <= row <= 2 and 0 <= col <= 2 and board[row][col] == ' ':
            return row, col
        else:
            print("Invalid move. Try again.")

# Function for computer move (simple random)
def computer_move(board):
    empty_cells = [(row, col) for row in range(3) for col in range(3) if board[row][col] == ' ']
    return random.choice(empty_cells)  # Import random module first

# Main game loop
while True:
    # Display board
    display_board(board)

    # Player move
    row, col = player_move(board)
    board[row][col] = 'X'

    # Check for player win or tie
    if check_win(board, 'X'):
        print("You win!")
        break
    elif all(cell != ' ' for row in board for cell in row):
        print("It's a tie!")
        break

    # Computer move
    row, col = computer_move(board)
    board[row][col] = 'O'

    # Check for computer win or tie
    if check_win(board, 'O'):
        print("Computer wins!")
        break
```

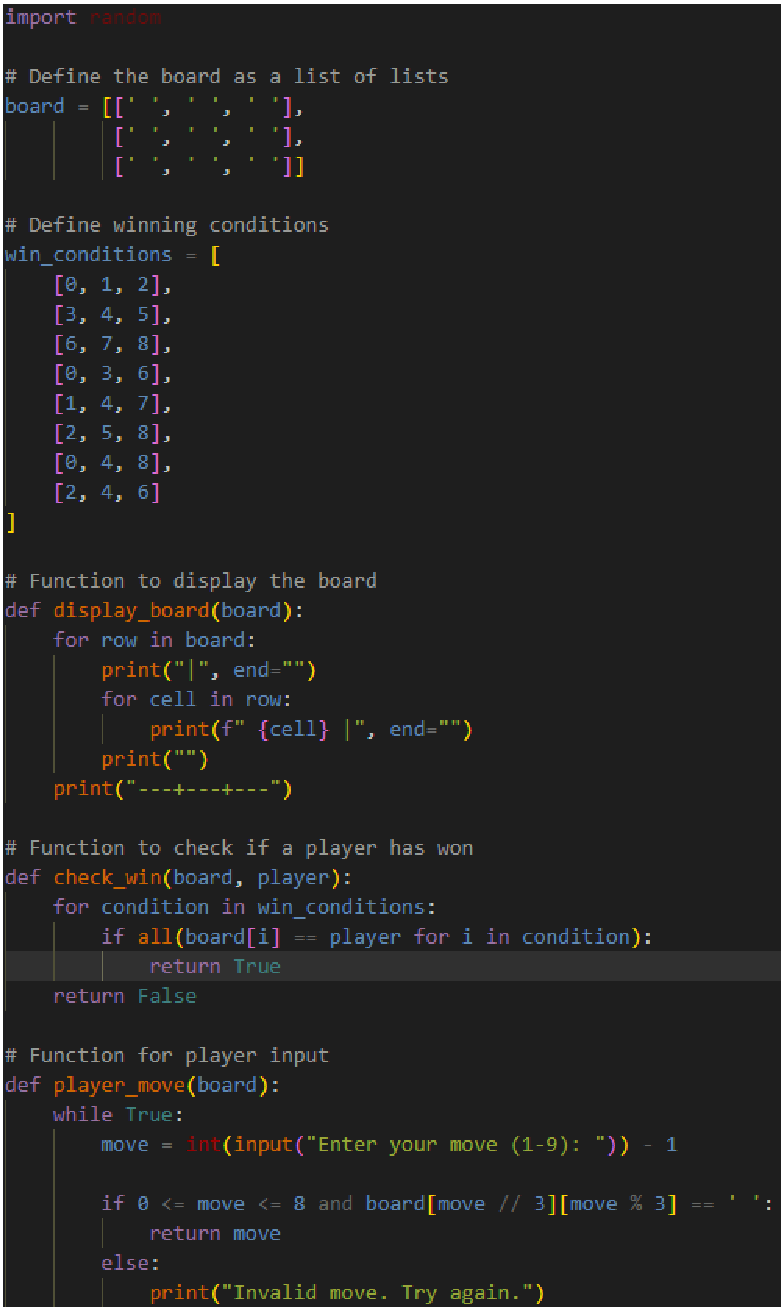
*Slika 27: Popravek kode po natančnejšem vnosu parametrov – 2. del.*

Koda, ki jo je generiral, je bila zelo podobna starejši, razlika je bila v tem, da je odstranila vprašanje, ali hočeš nadaljevati, in prestavila uvoz modula random na začetek. Opazil sem tudi, da je Gemini uporabil podoben način sprejemanja igralčevih vnosov kakor jaz, v programu je vprašal za številko od ena do devet in ji nato odštel ena, kar sem od njega tudi hotel, a nisem dodal v poziv, saj se mi je to zdelo prepodrobno za splošen vpis.

Ko sem generiral prvi dve kodi, sem Gemini vprašal po dveh novih kodah, le da sem tokrat vključil podrobnejši izgled igre in natančneje napisal, kako naj popravi napake (vse napake, ne le večje), po tem ko je generiral prvo kodo. Kot omenjeno, sem tokrat v poziv vključil podrobnejše stvari in do pike napisal, kaj od njega zahtevam. Poziv, ki sem ga uporabil, je naslednji:

*»Please generate a simple and easy Python code for a game of tic-tac-toe. Ensure the game runs in the console. When implementing the player input system, require the player to press a number from 1 to 9 to select the space on the board where they want to place their 'X'. Additionally, include a computer opponent that you will play against. Ensure the computer opponent is intelligent and doesn't make random moves.«*

Gemini se pri tej kodi ni preveč obremenil in jo je generiral kar hitro. Ko je s kodo končal, sem bil ponovno presenečen, saj je upošteval vse moje napotke, a tokrat sam h kodi ni prispeval, ampak je samo dodal stvari, za katere sem ga prosil. Ena izmed stvari, ki me je najbolj presenetila, je, da Gemini ni ustvarjal nove kode, ampak je le popravljal staro. Koda, ki jo je generiral, je izgledala tako:



```
import random

# Define the board as a list of lists
board = [[' ', ' ', ' '],
         [' ', ' ', ' '],
         [' ', ' ', ' ']]

# Define winning conditions
win_conditions = [
    [0, 1, 2],
    [3, 4, 5],
    [6, 7, 8],
    [0, 3, 6],
    [1, 4, 7],
    [2, 5, 8],
    [0, 4, 8],
    [2, 4, 6]
]

# Function to display the board
def display_board(board):
    for row in board:
        print("|", end="")
        for cell in row:
            print(f" {cell} |", end="")
        print("")
    print("---+---+---")

# Function to check if a player has won
def check_win(board, player):
    for condition in win_conditions:
        if all(board[i] == player for i in condition):
            return True
    return False

# Function for player input
def player_move(board):
    while True:
        move = int(input("Enter your move (1-9): ")) - 1

        if 0 <= move <= 8 and board[move // 3][move % 3] == ' ':
            return move
        else:
            print("Invalid move. Try again.")
```

*Slika 28: Tretji popravek kode – 1. del.*

```python
# Function for player input
def player_move(board):
    while True:
        move = int(input("Enter your move (1-9): ")) - 1

        if 0 <= move <= 8 and board[move // 3][move % 3] == ' ':
            return move
        else:
            print("Invalid move. Try again.")

# Function for computer move (minimax algorithm)
def minimax(board, player):
    if check_win(board, player):
        return 1
    elif check_win(board, opponent(player)):
        return -1
    elif all(cell != ' ' for row in board for cell in row):
        return 0

    max_score = -2
    best_move = None
    for i in range(9):
        if board[i // 3][i % 3] == ' ':
            board[i // 3][i % 3] = player
            score = minimax(board, opponent(player))
            board[i // 3][i % 3] = ' '
            if score > max_score:
                max_score = score
                best_move = i
    return max_score if player == 'X' else -max_score

def computer_move(board):
    best_score = -2
    best_move = None
    for i in range(9):
        if board[i // 3][i % 3] == ' ':
            board[i // 3][i % 3] = 'O'
            score = minimax(board, 'X')
            board[i // 3][i % 3] = ' '
            if score > best_score:
                best_score = score
                best_move = i
    return best_move

# Function to determine the opponent
def opponent(player):
    return 'X' if player == 'O' else 'O'
```

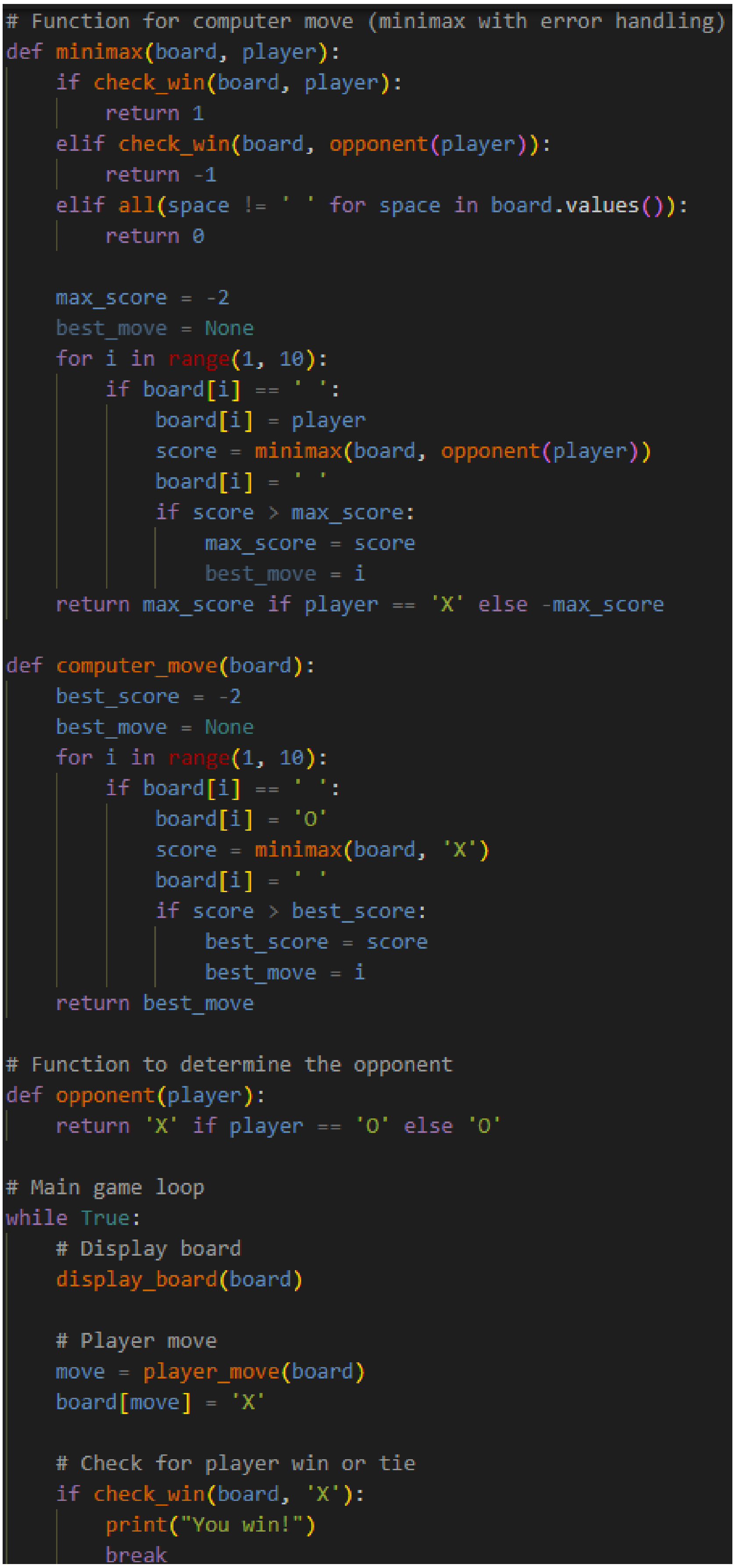
*Slika 29: Tretji popravek kode – 2. del.*

Ker je s kodo končal, sem jo vnesel v VS Code, kjer sem jo imel namen preveriti, a ko sem odigral prvo igro in hotel odigrati še eno, se je koda sesula, kar je pomenilo, da se je nekje v kodi zmotil, zato sem k novemu pozivu dodal opis, kako naj kodo popravi. Pri dodajanju natančnejših podatkov o popravi napake sem dodal še manjši napotek za preureditev nekaterih delov kode (način sprejemanja igralčevih vnosov).

*»Please generate a simple and easy Python code for a game of tic-tac-toe. Ensure the game runs in the console. When implementing the player input system, require the player to press a number from 1 to 9 to select the space on the board where they want to place their 'X'. Additionally, include a computer opponent that you will play against. Search for any mistakes and correct them, and remake the program so that it won't give an error list index out of range.«*

```
import random

# Define the board as a dictionary for numbered selection
board = {1: ' ', 2: ' ', 3: ' ',
         4: ' ', 5: ' ', 6: ' ',
         7: ' ', 8: ' ', 9: ' '}

# Define winning conditions
win_conditions = [
    [1, 2, 3],
    [4, 5, 6],
    [7, 8, 9],
    [1, 4, 7],
    [2, 5, 8],
    [3, 6, 9],
    [1, 5, 9],
    [3, 5, 7]
]

# Function to display the board
def display_board(board):
    for i in range(1, 10):
        print(board[i], end="")
        if i % 3 == 0:
            print("")

# Function to check if a player has won
def check_win(board, player):
    for condition in win_conditions:
        if all(board[i] == player for i in condition):
            return True
    return False

# Function for player input
def player_move(board):
    while True:
        move = int(input("Enter your move (1-9): "))
        if 1 <= move <= 9 and board[move] == ' ':
            return move
        else:
            print("Invalid move. Try again.")
```

*Slika 30: Zadnji popravek kode glede na parametre – 1. del.*



```
# Function for computer move (minimax with error handling)
def minimax(board, player):
    if check_win(board, player):
        return 1
    elif check_win(board, opponent(player)):
        return -1
    elif all(space != ' ' for space in board.values()):
        return 0

    max_score = -2
    best_move = None
    for i in range(1, 10):
        if board[i] == ' ':
            board[i] = player
            score = minimax(board, opponent(player))
            board[i] = ' '
            if score > max_score:
                max_score = score
                best_move = i
    return max_score if player == 'X' else -max_score

def computer_move(board):
    best_score = -2
    best_move = None
    for i in range(1, 10):
        if board[i] == ' ':
            board[i] = 'O'
            score = minimax(board, 'X')
            board[i] = ' '
            if score > best_score:
                best_score = score
                best_move = i
    return best_move

# Function to determine the opponent
def opponent(player):
    return 'X' if player == 'O' else 'O'

# Main game loop
while True:
    # Display board
    display_board(board)

    # Player move
    move = player_move(board)
    board[move] = 'X'

    # Check for player win or tie
    if check_win(board, 'X'):
        print("You win!")
        break
```

*Slika 31: Zadnji popravek kode glede na parametre – 2. del.*

Ker je bila to zadnja koda, sem bil prepričan, da bo popolna, a sem še vedno zasledil nekaj napak, predvsem je izstopalo to, da koda ni narisala polja, le križce in krožce. Poleg te večje napake nisem zasledil drugih večjih napak. Koda, ki jo je generiral, je bila zelo napredna, kljub ne zelo optimiziranemu programu. Pri kodah sem opazil, da je Gemini dodal veliko več stvari in programiral veliko bolj samostojno pri programih, kjer poziv ni bil razširjen. Največji problem, ki sem ga opazil, pa je bil, da je Gemini bil sposoben narediti računalniškega nasprotnika, za katerega bi lahko rekli, da je bil inteligenten šele, ko je za to dobil natančnejše napotke.

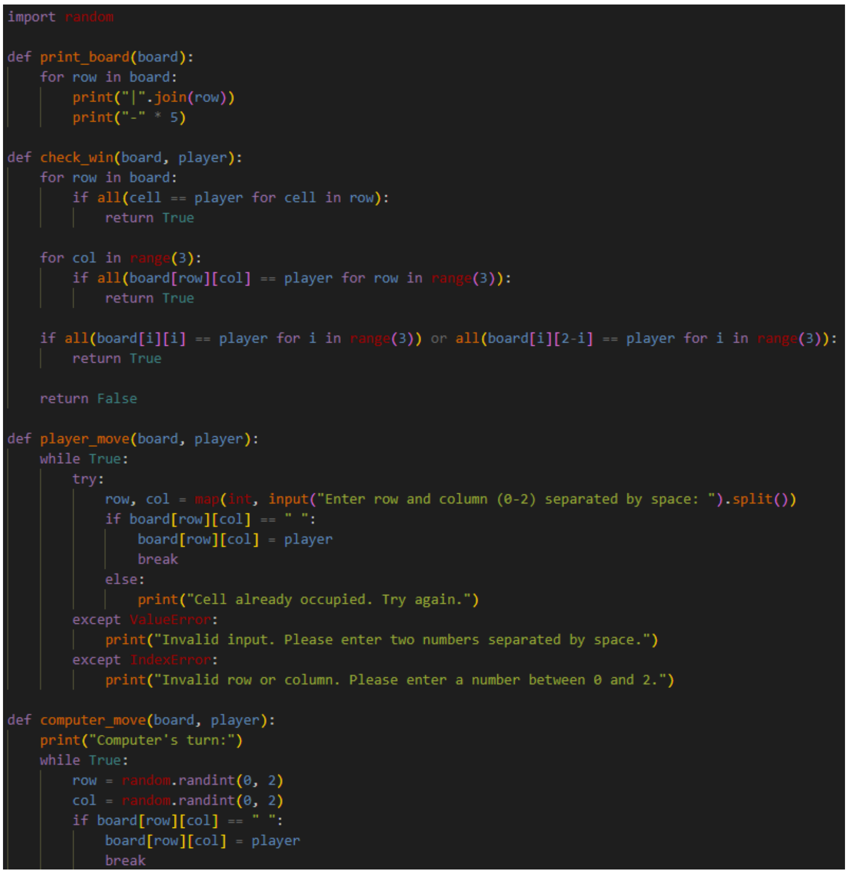
## 4.3 ChatGPT

Za generiranje programa sem uporabil verzijo ChatGPT-ja, ki ni bila plačljiva in deluje na verziji 3.5.

Z generiranjem kode umetne inteligence Gemini sem končal, zato sem se odločil za generiranje programa za križce in krožce, ter podal pozive še za ChatGPT. Ker sem se odločil uporabiti enake začetne kode, mi je to olajšalo delo, saj sem moral dodatno napisati/izpopolniti le pozive za popravljanje napak, ki jih bo ChatGPT naredil. Začel sem na enak način kot pri umetni inteligenci Gemini, in sicer z obiskom spletne strani ChatGPT-ja, ustvarjanjem novega pogovora in pisanjem poziva v pogovorno okence. Uporabil sem enak poziv kot pri Gemini-ju:

*»Generate me a simple an easy Pyhon code for a game of tic-tac-toe that will run in console. Include things like computer "enemy" you will play against and system for getting player inputs.«*

Program, ki ga je generiral ChatGPT, me je zelo presenetil, saj je bil po enostavnem pozivu zelo napreden in optimiziran.



```python
import random

def print_board(board):
    for row in board:
        print("|".join(row))
        print("-" * 5)

def check_win(board, player):
    for row in board:
        if all(cell == player for cell in row):
            return True

    for col in range(3):
        if all(board[row][col] == player for row in range(3)):
            return True

    if all(board[i][i] == player for i in range(3)) or all(board[i][2-i] == player for i in range(3)):
        return True

    return False

def player_move(board, player):
    while True:
        try:
            row, col = map(int, input("Enter row and column (0-2) separated by space: ").split())
            if board[row][col] == " ":
                board[row][col] = player
                break
            else:
                print("Cell already occupied. Try again.")
        except ValueError:
            print("Invalid input. Please enter two numbers separated by space.")
        except IndexError:
            print("Invalid row or column. Please enter a number between 0 and 2.")

def computer_move(board, player):
    print("Computer's turn:")
    while True:
        row = random.randint(0, 2)
        col = random.randint(0, 2)
        if board[row][col] == " ":
            board[row][col] = player
            break
```

*Slika 32: Generiranje prve kode s ChatGPT – 1. del.*

```
def main():
    board = [[" "]*3 for _ in range(3)]
    players = ['X', 'O']
    turn = 0

    print("Welcome to Tic-Tac-Toe!")
    print_board(board)

    while True:
        player = players[turn % 2]

        if player == 'X':
            player_move(board, player)
        else:
            computer_move(board, player)

        print_board(board)

        if check_win(board, player):
            print(f"{player} wins!")
            break

        if all(all(cell != " " for cell in row) for row in board):
            print("It's a tie!")
            break

        turn += 1

if __name__ == "__main__":
    main()
```

*Slika 33: Generiranje prve kode s ChatGPT – 2. del.*

Koda ChatGPT-ja me je torej zelo presenetila, saj je bila za enostaven poziv zelo napredna in optimizirana, ampak je imela malo več napak. Med napakami ni bilo večjih, le nepregledna tabela in čuden način sprejemanja igralčevih vnosov, ki je povzročil veliko zmede med analiziranjem programa.

Drugi program sem začel enako kot prvega, le da sem dodal manjši dodatek za olepšanje igralnega polja. Pri tem sem vnesel poziv:

*»Generate me a simple an easy Pyhon code for a game of tic-tac-toe that will run in console. Include things like computer "enemy" you will play against and system for getting player inputs and make more transparent board.«*

Pri novem programu me je ChatGPT presenetil, saj je bila koda enaka in v njej ni spremenil skoraj ničesar, razen tega, da je preuredil način sprejemanja igralčevih vnosov, da igralca ni zmedlo. Kljub temu vse skupaj še ni bilo urejeno, kot bi hotel, zato sem ChatGPT vprašal za novo kodo. Nova koda je izgledala tako:

```
import random

def print_board(board):
    for row in board:
        print("   ".join(row))
        print("-" * 11)

def check_win(board, player):
    for row in board:
        if all(cell == player for cell in row):
            return True

    for col in range(3):
        if all(board[row][col] == player for row in range(3)):
            return True

    if all(board[i][i] == player for i in range(3)) or all(board[i][2-i] == player for i in range(3)):
        return True

    return False

def player_move(board, player):
    while True:
        try:
            position = int(input("Enter position (1-9): "))
            row, col = (position - 1) // 3, (position - 1) % 3
            if board[row][col] == " ":
                board[row][col] = player
                break
            else:
                print("Cell already occupied. Try again.")
        except ValueError:
            print("Invalid input. Please enter a number between 1 and 9.")
        except IndexError:
            print("Invalid position. Please enter a number between 1 and 9.")

def computer_move(board, player):
    print("Computer's turn:")
    while True:
        row = random.randint(0, 2)
        col = random.randint(0, 2)
        if board[row][col] == " ":
            board[row][col] = player
            break
```

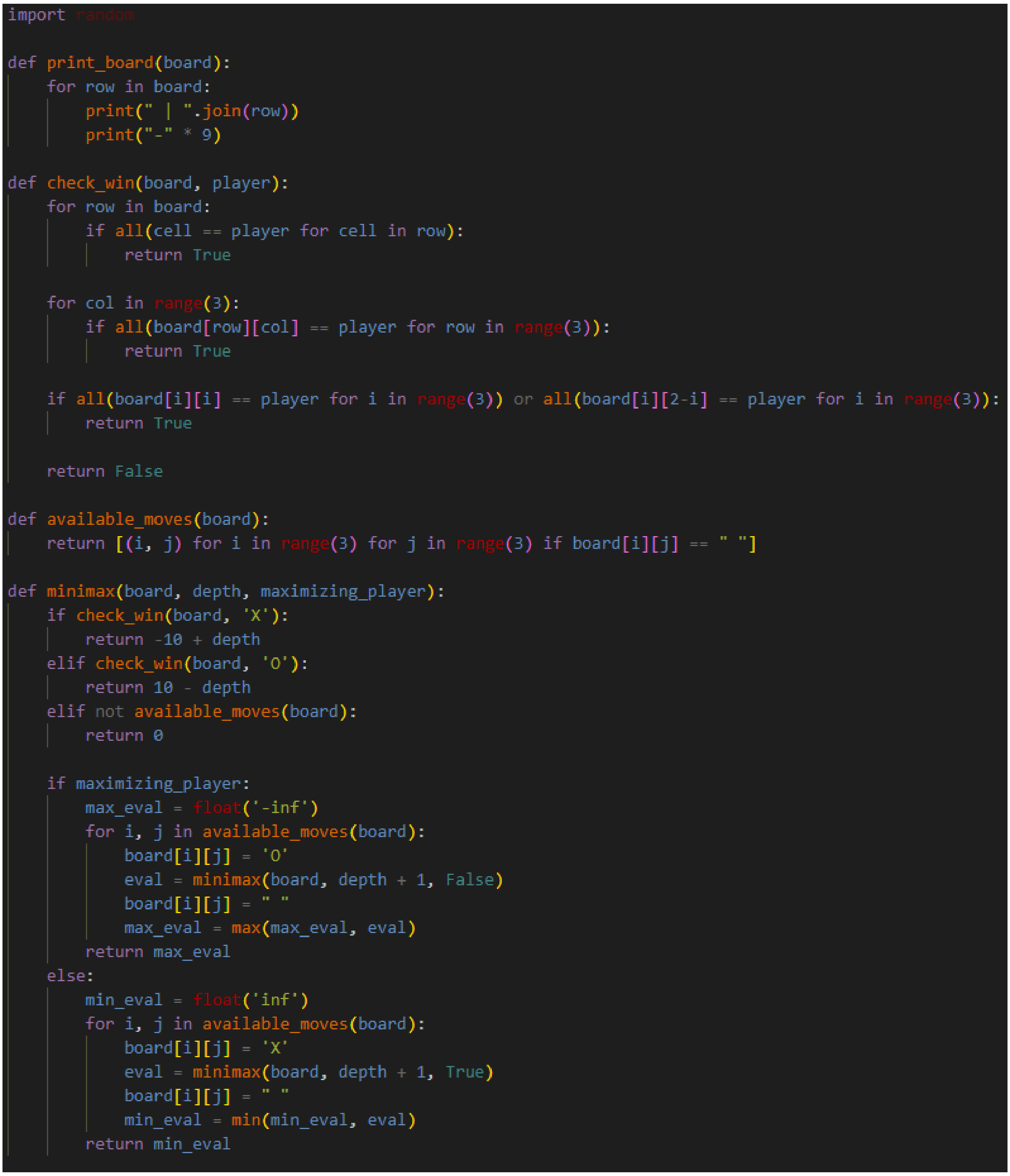
*Slika 34: Popravek kode v ChatGPT-ju – 1. del.*

```
def main():
    board = [[" "]*3 for _ in range(3)]
    players = ['X', 'O']
    turn = 0

    print("Welcome to Tic-Tac-Toe!")
    print_board(board)

    while True:
        player = players[turn % 2]

        if player == 'X':
            player_move(board, player)
        else:
            computer_move(board, player)

        print_board(board)

        if check_win(board, player):
            print(f"{player} wins!")
            break

        if all(all(cell != " " for cell in row) for row in board):
            print("It's a tie!")
            break

        turn += 1

if __name__ == "__main__":
    main()
```

*Slika 35: Popravek kode v ChatGPT-ju – 2. del.*

Pri tej kodi je ChatGPT upošteval dodatek, ki sem ga dodal pozivu, in naredil igralno polje preglednejše in brez dodatnih navodil izpopolnil sprejemanje igralčevih vnosov. Pri tem programu me je presenetilo to, da je ChatGPT potreboval dva poskusa, a vseeno brez dodatne spodbude samostojno izpopolnil program.

Ker sem imel narejena programa z enostavnim pozivom, sem nadaljeval z generiranjem programov z podrobnejšim pozivom. Ker sem imel že narejen poziv, sem ga samo napisal v pogovorno okence.



```
import random

def print_board(board):
    for row in board:
        print(" | ".join(row))
        print("-" * 9)

def check_win(board, player):
    for row in board:
        if all(cell == player for cell in row):
            return True

    for col in range(3):
        if all(board[row][col] == player for row in range(3)):
            return True

    if all(board[i][i] == player for i in range(3)) or all(board[i][2-i] == player for i in range(3)):
        return True

    return False

def available_moves(board):
    return [(i, j) for i in range(3) for j in range(3) if board[i][j] == " "]

def minimax(board, depth, maximizing_player):
    if check_win(board, 'X'):
        return -10 + depth
    elif check_win(board, 'O'):
        return 10 - depth
    elif not available_moves(board):
        return 0

    if maximizing_player:
        max_eval = float('-inf')
        for i, j in available_moves(board):
            board[i][j] = 'O'
            eval = minimax(board, depth + 1, False)
            board[i][j] = " "
            max_eval = max(max_eval, eval)
        return max_eval
    else:
        min_eval = float('inf')
        for i, j in available_moves(board):
            board[i][j] = 'X'
            eval = minimax(board, depth + 1, True)
            board[i][j] = " "
            min_eval = min(min_eval, eval)
        return min_eval
```

*Slika 36: Zadnji popravek kode v CharGPT-ju – 1. del.*

```
def computer_move(board):
    best_move = None
    for i, j in available_moves(board):
        board[i][j] = 'O'
        score = minimax(board, 0, False)
        board[i][j] = " "
        if score > best_score:
            best_score = score
            best_move = (i, j)
    board[best_move[0]][best_move[1]] = 'O'

def main():
    board = [[" "]*3 for _ in range(3)]
    players = ['X', 'O']
    turn = 0

    print("Welcome to Tic-Tac-Toe!")
    print_board(board)

    while True:
        player = players[turn % 2]

        if player == 'X':
            while True:
                try:
                    position = int(input("Enter position (1-9): "))
                    row, col = (position - 1) // 3, (position - 1) % 3
                    if board[row][col] == " ":
                        board[row][col] = player
                        break
                    else:
                        print("Cell already occupied. Try again.")
                except ValueError:
                    print("Invalid input. Please enter a number between 1 and 9.")
                except IndexError:
                    print("Invalid position. Please enter a number between 1 and 9.")
        else:
            computer_move(board)

        print_board(board)

        if check_win(board, player):
            print(f"{player} wins!")
            break

        if all(all(cell != " " for cell in row) for row in board):
            print("It's a tie!")
            break

        turn += 1
```

*Slika 37: Zadnji popravek kode v CharGPT-ju – 2. del.*

Med generiranjem kode sem takoj opazil, da je bila koda vidno daljša, kar je bila posledica zelo inteligentnega računalniškega igralca, ki ga je ChatGPT naredil tako pametnega, da se ga sploh ni dalo premagati. Med to večjo spremembo sem opazil tudi, da je ohranil način zbiranja igralčevih vnosov, največja razlika pa je bila hitrost odzivnosti programa, saj je program rabil skoraj sekundo, da se je odzval.

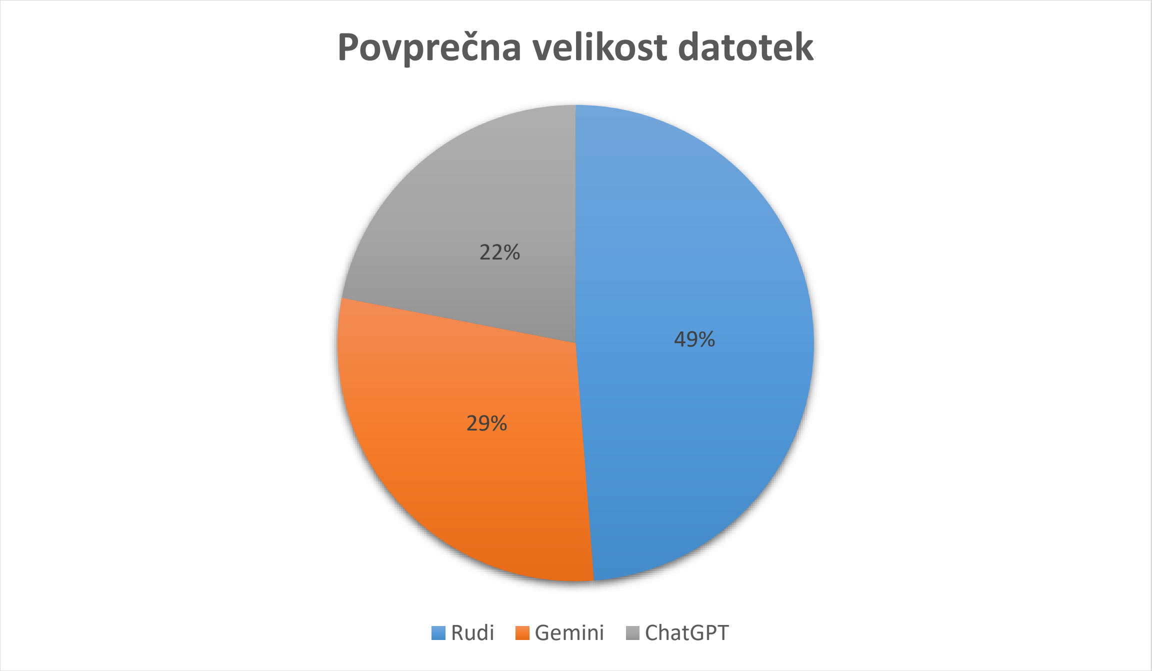
Za konec sem ChatGPT prosil za še zadnjo kodo. Pri tej kodi nisem imeli veliko za dopolniti, saj je bila koda popolna do pike, kar me je zelo presenetilo, saj sem sklepal, da bo v programiranju uspešnejša umetna inteligenca Gemini.

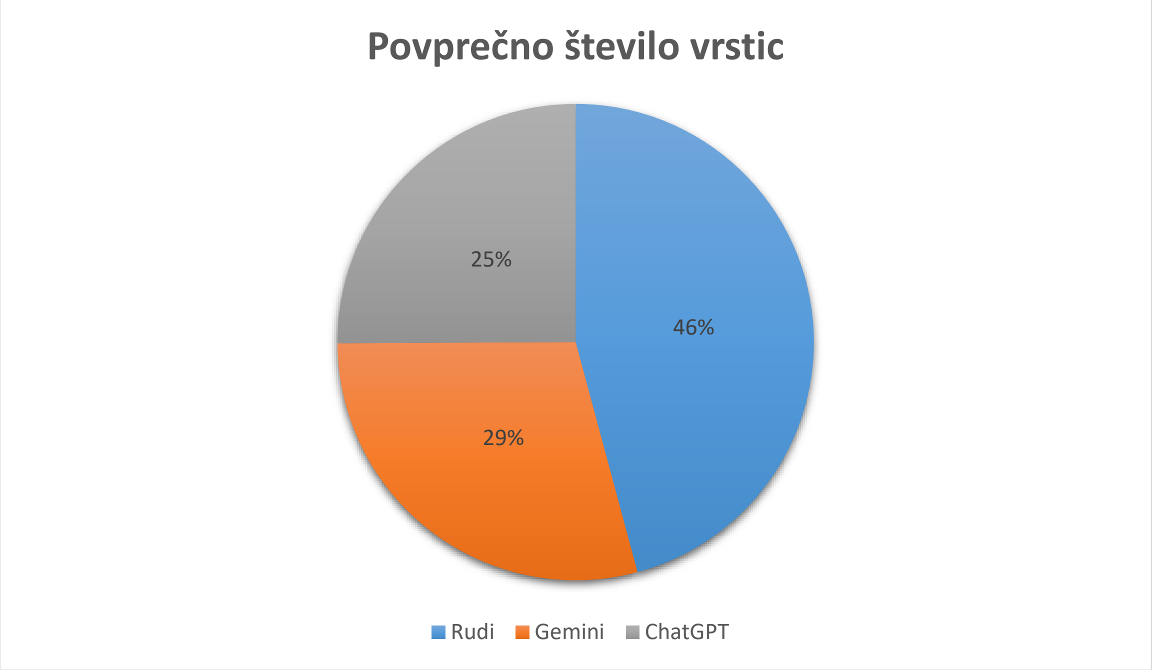
# 5 Analiza

S končanimi oziroma generiranimi programi sem se za konec odločil vse programe analizirati in svoje ugotovitve predstaviti v tabelah / tortnih prikazih. Analiziral sem velikost datoteke s kodo programov, dolžino programa, kako je umetna inteligenca optimizirala program, ali je AI pri generiranju programov dodal še manjše stvari oziroma delce kode, ki bi program izboljšali, ali bo ponavljala napake, ki bodo izpostavljene, kako pameten bo računalniški nasprotnik, kako hitro bo svojo potezo zaključil in ali bo umetna inteligenca sledila napotkom (pozivu) in popravila napake ali pa izpostavljenih napak niti ne bo spremenila oziroma dodala novih. Pri analizi sem izračunal povprečje (povprečno število neuspešnih programov/velikost datotek …) in to povprečje zapisal v tabelo spodaj in nato rezultate prikazal še v tortnih prikazih, ostale ugotovitve sem zapisal pod tabelo.

| | Rudi | Gemini | ChatGPT |
|---|---|---|---|
| **Velikost datotek** | 5 KB | 3 KB | 2,25 KB |
| **Dolžina programa** | 148 vrstic | 94 vrstic | 81 vrstic |
| **Št. neuspešnih programov** | 1 | 0 | 1 |
| **Št. napak** | 7 | 5 | 3 |
| **Št. zmag nasprotnika** | 20 % | 0 % | 40 % |

*RČN je okrajšava za računalniški nasprotnik. *v je okrajšava za vrstice *KB je okrajšava za kilobajte





Med analizo programov sem ugotovil, da je bila pri programiranju najuspešnejša umetna inteligenca ChatGPT, ki je imela najmanjšo velikost datoteke s programom, hkrati pa tudi najbolj optimiziran program, saj zavzema v povprečju samo 81 vrstic in 2,25 KB, kar je mogoče samo z močno optimizacijo in uporabo naprednih in težjih tehnik in modulov. Pri programov sem opazil tudi velik problem – v programih umetne inteligence Gemini je bila koda generirana enostavneje, a je bila daljša, vendar je to omogočilo hitrejše delovanje, ki je na pogled skoraj takojšnje v primerjavi z najboljšim programom ChatGPT-ja, ki ima čas odziva skoraj eno sekundo. To, da je ChatGPT boljši pri programiranju, sem opazil že na samem začetku, saj je Gemini namesto uvažanja modula za naključno izbiranja mesta začel pisati lasten program, kar je povečalo velikost datoteke programa, seveda pa sem lahko to sklepal šele pri programu samega računalniškega nasprotnika pri težjem programiranju, kjer je v primerjavi z Gemini-jem ChatGPT naredil skoraj nepremagljivega nasprotnika, ki je bil sposoben pregledati celotno polje in s tem pridobiti dovolj informacij, da lahko zmaga oziroma izenači.

Z analizo sem opazil, da sta se umetni inteligenci učili iz starejših napak in samostojno pregledali program, ki sta ga že naredili in ga samo izpopolnili in izboljšali, namesto da bi vsakič generirali čisto drugačen program. Med analiziranjem sem prav tako opazil, da sta umetni inteligenci poleg stvari opisanih v pozivu dodali lastne dodatke, ki so sam program tudi izboljšali, hkrati pa nobena izmed inteligenc ni ponavljala napake. To se je seveda spremenilo pri težjem programiranju, kjer sta z vsakim dodatkom, ki jih je bilo malo, samo še povečali število napak.

# 6 Inženirski dnevnik

Vse spremembe svojih rešitev sem skrbno beležil v inženirskem dnevniku. Pri tem sem lahko sledil, na katerem koraku razvoja rešitve sem, kaj je potrebno spremeniti in kako sem težavo rešil.

| Začetek | Problem | Opis problema |
|---|---|---|
| **#0** | Definicija problema | Začeti programirati |

| # | Problem | Opis problema | Rešitev problema |
|---|---|---|---|
| **#1** | Sprejemanje vnosa | Polja se začnejo s številom nič, potreben pa je začetek z ena. | Pri sprejemu vnosa od številke, ki jo vnesem, odšteje ena. |
| **#2** | Preverjanje, ali je kdo na polju | Če nekdo vnese številko polja, kjer je nekdo že, potem se vse sesuje. | Pred postavitvijo preveri, ali je polje zasedeno, in to sporoči. |
| **#3** | Preverjanje, ali lahko RČN blokira | RČN izbira naključna polja. | Računalnik preleti polje in pregleda, ali lahko igralcu prepreči zmago. |
| **#4** | Zaganjanje progama | Program se izvede enkrat in se ustavi. | Na koncu sem dodal kodo, ki je ponovno zagnala novo kodo. |
| **#5** | Preverjanje za zmago | Če so vsa polja zapolnjena, potem se igra sesuje. | Dodal sem kodo, ki preveri, ali je ta polja zasedel isti igralec. |
| **#6** | Preveri za izenačenje | Če igralec zapolni vsa polja in nihče ne zmaga, se vse sesede. | Rešil sem tako, da sem dodal nov ukaz, da program ob zapolnjenih mestih sporoči izenačenje in zaključi igro. |
| **#7** | Zmagovalec | Ko nekdo zmaga, se igra zaključi in ne pokaže, kdo je zmagal in kako. | Dodal sem ukaz, da se ob zmagi ali porazu ponovno izriše polje in zraven napiše, kdo je zmagal. |

*Tabela 1: Inženirski dnevnik – sledenje spremembam.*

# 7 Sklepna ugotovitev

S to raziskovalno nalogo sem dokazal, da je prosto dostopna inteligenca sposobna samostojnega enostavnejšega programiranja in učenja iz napak, a pri težjih programerskih projektih ni sposobna programiranja z osnovnim pozivom. Ugotovil sem tudi, da je umetna inteligenca sposobna generiranja računalniškega nasprotnika, ki ima le delno izpopolnjeno zmagovalni algoritem, ki ga je še vedno mogoče premagati. Prav tako sem z analizo ugotovil, da je pri programiranju v veliko večjo pomoč umetna inteligenca ChatGPT, ki je pri projektih še kako priročna in je v primerjavi z umetno inteligenco Gemini veliko bolj razvita za programiranje.

# 8 Zaključek

Z raziskovalno nalogo sem ugotovil, da je umetna inteligenca sposobna programiranja na enostavnejši stopnji, a pri težjem programiranju še vedno ni čisto izpopolnjena.

Pri programiranju je zelo pomembno analitično razmišljanje, podobno kot pri šahu. Problem je treba zaznati in predvideti, nato pa najti pot do rešitve. To pot je treba znati razdrobiti na majhne korake in jih enega za drugim znati sprogramirati. Zadevo dodatno popestri dejstvo, da je možnih poti vedno vsaj nekaj. Vseskozi je treba izbirati in se odločati.

Ugotovil sem, da je pomembna vztrajnost ne glede na vse izzive in probleme, ki jih programiranje prinese.

Skozi izdelavo raziskovalne naloge sem spoznal različne vidike programiranja in uporabe Pythona-a. Podrobno sem se seznanil tudi z umetno inteligenco. V samem raziskovanju sem izredno užival. Spoznal sem nove načine razmišljanja in programiranja, kar mi bo v pomoč še v prihodnje.

# 9 Viri in literatura

Bard. Pridobljeno 25. 2. 2024 s https://bard.google.com/chat.

ChatGPT. Pridobljeno 25. 2. 2024 s https://chat.openai.com/g/g-jCYeXl5xh-new-gpt-5.

Choi, A. S. (2021). TIC-TAC-TOE. Pridobljeno 17. 10. 2023 s https://momath.org/wp-content/uploads/2021/08/Alyssa-Choi-Tic-Tac-Toe.pdf.

Pečenko, N. (2005). Kratka zgodovina računalniških iger. Pridobljeno 10. 10. 2023 s https://www.monitor.si/clanek/kratka-zgodovina-racunalniskihiger/122009/?xURL=301.

Programski jezik Python. Pridobljeno 27. 9. 2023 s https://www.diameter.si/clutter/python.pdf.

Python. Pridobljeno 24. 10. 2023 s https://www.python.org/.

Python. Pridobljeno 28. 9. 2023 s https://www.britannica.com/technology/Python-computer-language.

Tutorialspoint. Pridobljeno 11. 10. 2023 s https://www.tutorialspoint.com/python/index.htm.

Visual Studio Code. Pridobljeno 24. 10. 2023 s https://code.visualstudio.com/.