In [ ]:

```python
from easyAI import TwoPlayersGame, Human_Player, AI_Player, Negamax
from tkinter import Tk, Button
from tkinter.font import Font
from copy import deepcopy


class Board:

    def __init__(self, other=None):
        self.player = 'X'
        self.opponent = 'O'
        self.empty = '.'
        self.size = 3
        self.fields = {}
        for y in range(self.size):
            for x in range(self.size):
                self.fields[x, y] = self.empty
        # copy constructor
        if other:
            self.__dict__ = deepcopy(other.__dict__)

    def move(self, x, y):
        board = Board(self)
        board.fields[x, y] = board.player
        (board.player, board.opponent) = (board.opponent, board.player)
        return board

    def __minimax(self, player):
        if self.won():
            if player:
                return (-1, None)
            else:
                return (+1, None)
        elif self.tied():
            return (0, None)
        elif player:
            best = (-2, None)
            for x, y in self.fields:
                if self.fields[x, y] == self.empty:
                    value = self.move(x, y).__minimax(not player)[0]
                    if value > best[0]:
                        best = (value, (x, y))
            return best
        else:
            best = (+2, None)
            for x, y in self.fields:
                if self.fields[x, y] == self.empty:
                    value = self.move(x, y).__minimax(not player)[0]
                    if value < best[0]:
                        best = (value, (x, y))
            return best

    def best(self):
        return self.__minimax(True)[1]

    def tied(self):
        for (x, y) in self.fields:
            if self.fields[x, y] == self.empty:
                return False
        return True

    def won(self):
        # horizontal
        for y in range(self.size):
            winning = []
            for x in range(self.size):
                if self.fields[x, y] == self.opponent:
                    winning.append((x, y))
            if len(winning) == self.size:
                return winning
        # vertical
        for x in range(self.size):
            winning = []
            for y in range(self.size):
                if self.fields[x, y] == self.opponent:
                    winning.append((x, y))
            if len(winning) == self.size:
                return winning
        # diagonal
        winning = []
        for y in range(self.size):
            x = y
            if self.fields[x, y] == self.opponent:
                winning.append((x, y))
        if len(winning) == self.size:
            return winning
        # other diagonal
        winning = []
        for y in range(self.size):
            x = self.size-1-y
            if self.fields[x, y] == self.opponent:
                winning.append((x, y))
        if len(winning) == self.size:
            return winning
        # default
        return None

    def __str__(self):
        string = ''
        for y in range(self.size):
            for x in range(self.size):
                string += self.fields[x, y]
            string += "\n"
        return string


class GUI:

    def __init__(self):
        self.app = Tk()
        self.app.title('TicTacToe')
        self.app.resizable(width=200, height=200)
        self.board = Board()
        self.font = Font(family="Helvetica", size=32)
        self.buttons = {}
        for x, y in self.board.fields:
            def handler(x=x, y=y): return self.move(x, y)
            button = Button(self.app, command=handler,
                            font=self.font, width=2, height=1)
            button.grid(row=y, column=x)
            self.buttons[x, y] = button

        def handler(): return self.reset()
        button = Button(self.app, text='Reiniciar', command=handler)
        button.grid(row=self.board.size+1, column=0,
                    columnspan=self.board.size, sticky="WE")
        self.update()

    def reset(self):
        self.board = Board()
        self.update()

    def move(self, x, y):
        self.app.config(cursor="watch")
        self.app.update()
        self.board = self.board.move(x, y)
        self.update()
        move = self.board.best()
        if move:
            self.board = self.board.move(*move)
            self.update()
        self.app.config(cursor="")

    def update(self):
        for (x, y) in self.board.fields:
            text = self.board.fields[x, y]
            self.buttons[x, y]['text'] = text
            self.buttons[x, y]['disabledforeground'] = 'black'
            if text == self.board.empty:
                self.buttons[x, y]['state'] = 'normal'
            else:
                self.buttons[x, y]['state'] = 'disabled'
        winning = self.board.won()
        if winning:
            for x, y in winning:
                self.buttons[x, y]['disabledforeground'] = 'red'
            for x, y in self.buttons:
                self.buttons[x, y]['state'] = 'disabled'
        for (x, y) in self.board.fields:
            self.buttons[x, y].update()

```

In [ ]:

```
from tkinter import *
from tkinter import ttk
from tkinter import messagebox

raiz = Tk()
raiz.geometry('300x100')  # anchura x altura

raiz.title('Examen IA')
Label(raiz, text="Examen IA").place(x=113, y=0)

Label(raiz, text="Nombre del Jugador").place(x=100, y=25)

entry = ttk.Entry(raiz)
entry.place(x=90, y=50)


ttk.Button(raiz, text='Juego',command=GUI).place(x=50, y=75)
ttk.Button(raiz, text='Recomendacion').place(x=173, y=75)

raiz.mainloop()
```

In [2]:

```python
from neo4j import GraphDatabase


class Neo4jService(object):

    def __init__(self, uri, user, password):
        self._driver = GraphDatabase.driver(uri, auth=(user, password))

    def close(self):
        self._driver.close()

    def crear_nodo(self, tx, nombre,v1,v2,v3,v4,v5,v6,v7,v8,v9):
        tx.run("MERGE (jugador:Lugar {name:$nombre})"
        "SET jugador.embedding = [$v1, $v2, $v3, $v4, $v5, $v6, $v7, $v8, $v9]",nombre=nombre
        ,v1=v1,v2=v2,v3=v3,v4=v4,v5=v5,v6=v6,v7=v7,v8=v8,v9=v9)

    def recomendacion(self,tx):
        result = tx.run("MATCH (m:Lugar)\n"
                        "WITH {item:id(m), weights: m.embedding} AS userData\n"
                        "WITH collect(userData) AS data\n"
                        "CALL gds.alpha.similarity.pearson.stream({\n"
                        "data: data,\n"
                        "skipValue: null\n"
                        "})\n"
                        "YIELD item1, item2, similarity\n"
                        "RETURN gds.util.asNode(item1).name AS from, gds.util.asNode(item2).name AS to,
                        "ORDER BY similarity DESC")
        for record in result:
            r1=(record["from"])
            r2=(record["to"])
            r3=(record["similarity"])
            if r1 ==nombre.get() and r3>=0.80:
                resultado.insert(tk.END, "\n"+r2)
```

In [3]:

```
from tkinter import *
from tkinter import ttk
from tkinter import messagebox
import tkinter as tk

raiz1 = Tk()

def clearTextInput():
    resultado.delete("1.0","end")

def comenzar():
    neo4j = Neo4jService('bolt://localhost:7687', 'neo4j', 'examenia')
    with neo4j._driver.session() as session:
        session.write_transaction(neo4j.crear_nodo , nombre.get(),float(tfa1.get()), float(tfa2.get()),

def buscar():
    neo4j = Neo4jService('bolt://localhost:7687', 'neo4j', 'examenia')
    with neo4j._driver.session() as session:
        session.read_transaction(neo4j.recomendacion)



raiz1.geometry('400x500')  # anchura x altura

raiz1.title('Examen IA')
Label(raiz1, text="Examen IA").place(x=155, y=0)

Label(raiz1, text="Que tanto te gusta el clima caliente.").place(x=55, y=25)
tfa1 = Spinbox(raiz1, from_=1, to=3, width=5, increment=1)
tfa1.place(x=300, y=25)

Label(raiz1, text="Que tanto te gusta la playa.").place(x=55, y=50)
tfa2 = Spinbox(raiz1, from_=1, to=3, width=5, increment=1)
tfa2.place(x=300, y=50)

Label(raiz1, text="Te gusta los platos preparados con marisco.").place(x=55, y=75)
tfa3 = Spinbox(raiz1, from_=1, to=3, width=5, increment=1)
tfa3.place(x=300, y=75)

Label(raiz1, text="Que tanto te guta el clima frio.").place(x=55, y=100)
tfa4 = Spinbox(raiz1, from_=1, to=3, width=5, increment=1)
tfa4.place(x=300, y=100)
```

```python

Label(raiz1, text="Que tanto te gusta el paramo.").place(x=55, y=125)
tfa5 = Spinbox(raiz1, from_=1, to=3, width=5, increment=1)
tfa5.place(x=300, y=125)

Label(raiz1, text="Te gusta los paltos preparados con carne..").place(x=55, y=150)
tfa6 = Spinbox(raiz1, from_=1, to=3, width=5, increment=1)
tfa6.place(x=300, y=150)

Label(raiz1, text="Que tanto te guta el clima humedo").place(x=55, y=175)
tfa7 = Spinbox(raiz1, from_=1, to=3, width=5, increment=1)
tfa7.place(x=300, y=175)

Label(raiz1, text="Que tanto te gusta el bosque").place(x=55, y=200)
tfa8 = Spinbox(raiz1, from_=1, to=3, width=5, increment=1)
tfa8.place(x=300, y=200)

Label(raiz1, text="Te gusta la comida exotica.").place(x=55, y=225)
tfa9 = Spinbox(raiz1, from_=1, to=3, width=5, increment=1)
tfa9.place(x=300, y=225)

ttk.Button(raiz1, text='Guardar Datos', command=comenzar).place(x=75, y=250)
ttk.Button(raiz1, text='Recomendar', command=buscar).place(x=250, y=250)
ttk.Button(raiz1, text="Limpiar",command=clearTextInput).place(x=165, y=250)

Label(raiz1, text="Se recomienda visitar estos lugares.").place(x=55, y=300)

resultado = Text(raiz1)
resultado.place(x = 55, y=325, width=300, height=100)

Label(raiz1, text="Nombre:").place(x=100, y=450)

nombre = ttk.Entry(raiz1)
nombre.place(x = 200, y=450)

raiz1.mainloop()
```

In [ ]: