**Veštačka inteligencija**

projekat 1 – faza 1

U ovoj fazi implementirali smo način predstavljanja problema, tablu, poziciju zidova i broj peostalih zidova. Sve vrednosti (dimenzije table, položaj kućica figura, broj zidova) moguće je uneti sa tastature ili koristiti defaulte vrednosti. Takođe se ovom fazom implementiraju funkcije za proveru kraja igre kao i mogućnost ponovnog pocetka.

Globalne promenljive koje se koriste u kodu su:

tabla=[]-lista listi koja predstavlja tablu za igru

x1,x2,o1,o2 – tuple u kome se nalaze koordinate kućice igrača x1,x2,o1,o2

cijiPotez="x" – promenljiva koja pamti čiji je trenutno potez

xZidovi, oZidovi –broj zidova igrača x,o

pozicije={}- dictionary koji cuva trenutne pozicije figura

listaStates- lista mogućih stanja

**def endGame()-fja koja proverava da li je došlo do kraja igre**

def endGame()-

    global pozicije

    global x1

    global x2

    global o1

    global o2

    px1i=[x1[0],x1[1]]

    px2i=[x2[0],x2[1]]

    po1i=[o1[0],o1[1]]

    po2i=[o2[0],o2[1]]

    if (pozicije["px1"]==po1i or pozicije["px1"]==po2i or pozicije["px2"]==po1i or pozicije["px2"]==po2i ):

        print("Korisnik X je pobedio")

        return True

    if (pozicije["po1"]==px1i or pozicije["po1"]==px2i or pozicije["po2"]==px1i or pozicije["po2"]==px2i ):

        print("Korisnik O je pobedio")

        return True

    return False

**def inputT()-unos svih početnih parametara igre, ukoliko se unese 0 prilikom unosa parametra koristi se defaultna vrednost**

global n

    global m

    global x1

    global x2

    global o1

    global o2

    print("Unesite n da bude neparan broj. Ukoliko ne zelite unesite 0")

    n=(int)(input())

    while (n%2 == 0 or n<11 or n>22) and n!=0:

        print("Unesite n da bude neparan broj. Ukoliko ne zelite unesite 0")

        n=(int)(input())

    if(n == 0):

       n=11

    n=2\*n

    print("Unesite m da bude paran broj. Ukoliko ne zelite unesite 0")

    m=(int)(input())

    while (m%2!=0 or m<14 or m>28) and m!=0:

        print("Unesite m da bude paran broj. Ukoliko ne zelite unesite 0")

        m=(int)((int)(input()))

    if(m == 0):

       m=14

    m=2\*m

    print()

    print("Unesite pocetna polja prve figure, ukoliko ne zelite unesite 0 0")

    a, b = input().split()

    if(int(a)==1 or int(a)==n or int(b)==1 or int(b)==m):

        print("Unesite pocetna polja prve figure, nije moguce postaviti figure na ivicama table, ukoliko ne zelite unesite 0 0")

        a, b = input().split()

    if(int(a)==0 or int(b)==0):

       a=4

       b=4

    x1=(2\*(int(a)-1),2\*(int(b)-1))

    print("Unesite pocetna polja druge figure, ukoliko ne zelite unesite 0 0")

    a, b = input().split()

    if(int(a)==1 or int(a)==n or int(b)==1 or int(b)==m):

        print("Unesite pocetna polja prve figure, nije moguce postaviti figure na ivicama table, ukoliko ne zelite unesite 0 0")

        a, b = input().split()

    if(int(a)==0 or int(b)==0):

       a=8

       b=4

    x2=(2\*(int(a)-1),2\*(int(b)-1))

    o1=x1

    o2=x2

    brojZidova()

    XorO()

    Mirror()

**Def Mirror()- u odnosu na početne vrednosti jednog igrača postavlja vrednosti kućica drugog igrača**

def Mirror():

    global x1

    global x2

    global o1

    global o2

    global pozicije

    if(x1==(6,6)):

        o1=(6,20)

    else:

        pom1=int(((n-x1[0])-4))

        pom2=int(((m-x1[1])-4))

        o1=(pom1,pom2)

    if(x2==(14,6)):

        o2=(14,20)

    else:

        pom3=int(((n-x2[0])-4))

        pom4=int(((m-x2[1])-4))

        o2=(pom3,pom4)

    if(not player1):

        pom=x1

        x1=o1

        o1=pom

        pom=x2

        x2=o2

        o2=pom

    px1i=[x1[0],x1[1]]

    px2i=[x2[0],x2[1]]

    po1i=[o1[0],o1[1]]

    po2i=[o2[0],o2[1]]

    pozicije['px1']=px1i

    pozicije['px2']=px2i

    pozicije['po1']=po1i

    pozicije['po2']=po2i

**Def Zid()- postavljaju se zidovi u odnosu na izbor vrste zida (horizontalni ili vertikalni) i unose se koordinate zida (prilikom unosa koordinata proverava se da li je moguće postaviti zid na tu koordinatu)**

def zid():

global xZidovi

    global oZidovi

    global cijiPotez

    if (cijiPotez == 'x'):

        if (xZidovi == 0 ):

            print("Nemate vise zidova za postavljanje")

            return

        print("Unesite p ukoliko zelite da postavite plavi zid (horizontalni) ili z ukoliko zelite zeleni (vertikalni)")

        slovo = input()

        while (slovo != 'p' and slovo != 'z'):

            print("Nevalidan unos, unesite ponovo")

            slovo = input()

        if (slovo == 'p'):

            print("Plavi zid:")

            plavi()

            xZidovi -= 1

        elif (slovo == 'z'):

            print("Zeleni zid:")

            zeleni()

            xZidovi -= 1

    else:

        if (oZidovi == 0 ):

            print("Nemate vise zidova za postavljanje")

            return

        print("Unesite p ukoliko zelite da postavite plavi zid ili z ukoliko zelite zeleni")

        slovo = input()

        while (slovo != 'p' and slovo != 'z'):

            print("Nevalidan unos, unesite ponovo")

            slovo = input()

        if (slovo == 'p'):

            print("Plavi zid:")

            plavi()

            oZidovi -= 1

            print(checkWall())

        elif (slovo == 'z'):

            print("Zeleni zid:")

            zeleni()

            oZidovi -= 1

**def move()-fja koja proverava da li je moguće kretanje u tom smeru (proverava da li je taj potez izvan table, da li se neko nalazi na toj poziciji, da li se u tom smeru nalazi zid). Mogući smerovi su u-up, d-down, l-left, r-right, ul-up left, ur-up right, dl- down left, dr-down right**

def move():

global tabla

    global pozicije

    global n

    global m

    global x1

    global x2

    global o1

    global o2

    global cijiPotez

    global moved

    print("Unesite figuru (px1, px2, po1, po2) i smer pomeranja figure (u, d, l, r, ur, ul, dr, dl)")

    try:

        p, where = input().split()

        while (p not in pozicije):

                print("Ime figure nije korektno")

                print("Unesite figuru (px1, px2, po1, po2) i smer pomeranja figure (u, d, l, r, ur, ul, dr, dl)")

                p, where = input().split()

        while (cijiPotez not in p):

                print("Ime figure nije korektno")

                print("Unesite figuru (px1, px2, po1, po2) i smer pomeranja figure (u, d, l, r, ur, ul, dr, dl)")

                p, where = input().split()

        pom=pozicije[p]

        if (where=="u" and  (pom[0]==0 or pom[0]==2 or pom[0]==1 or tabla[pom[0]-3][pom[1]]=="===" or tabla[pom[0]-1][pom[1]]=="===" )):

                print("Nije moguce pomeranje figure u tom smeru, promenite smer kretanja")

        elif(where=="d" and  (pom[0]==n-2 or pom[0]==n-4 or tabla[pom[0]+3][pom[1]]=="===" or tabla[pom[0]+1][pom[1]]=="===" )):

                print("Nije moguce pomeranje figure u tom smeru, promenite smer kretanja")

        elif (where=="l" and  (pom[1]==0 or pom[1]==2 or pom[1]==2 or tabla[pom[0]][pom[1]-3]==" ǁ " or tabla[pom[0]][pom[1]-1]==" ǁ " )):

                print("Nije moguce pomeranje figure u tom smeru, promenite smer kretanja")

        elif (where=="r" and  (pom[1]==m-2 or pom[1]==m-4 or tabla[pom[0]][pom[1]+3]==" ǁ " or tabla[pom[0]][pom[1]+1]==" ǁ " )):

                print("Nije moguce pomeranje figure u tom smeru, promenite smer kretanja")

        elif (where=="ur" and  ((pom[0]==0) or (pom[1]==m-2 or pom[1]==m-4)

                    or (tabla[pom[0]-1][pom[1]]=="===" and tabla[pom[0]-1][pom[1]+2]=="===")

                    or (tabla[pom[0]-2][pom[1]+1]==" ǁ " and tabla[pom[0]][pom[1]+1]==" ǁ ")

                    or (tabla[pom[0]-1][pom[1]+2]=="===" and tabla[pom[0]-2][pom[1]+1]==" ǁ ")

                    or (tabla[pom[0]-1][pom[1]]=="===" and tabla[pom[0]][pom[1]+1]==" ǁ ")

                    )):

                print("Nije moguce pomeranje figure u tom smeru, promenite smer kretanja")

        elif (where=="ul" and  ((pom[0]==0 ) or (pom[1]==0 or pom[1]==1)

                    or (tabla[pom[0]-1][pom[1]-2]=="===" and tabla[pom[0]-1][pom[1]]=="===")

                    or (tabla[pom[0]][pom[1]-1]==" ǁ " and tabla[pom[0]-2][pom[1]-1]==" ǁ ")

                    or (tabla[pom[0]-1][pom[1]-2]=="===" and tabla[pom[0]-2][pom[1]-1]==" ǁ ")

                    or (tabla[pom[0]-1][pom[1]]=="===" and tabla[pom[0]][pom[1]-1]==" ǁ "))):

                print("Nije moguce pomeranje figure u tom smeru, promenite smer kretanja")

        elif(where=="dr" and  ((pom[0]==n-2  ) or(pom[1]==m-2 or pom[1]==m-4)

                    or (tabla[pom[0]+1][pom[1]]=="===" and tabla[pom[0]+1][pom[1]+2]=="===" )

                    or (tabla[pom[0]][pom[1]+1]==" ǁ " and tabla[pom[0]+2][pom[1]+1]==" ǁ " )

                    or (tabla[pom[0]+1][pom[1]+2]=="===" and tabla[pom[0]+2][pom[1]+1]==" ǁ ")

                    or (tabla[pom[0]+1][pom[1]]=="===" and tabla[pom[0]][pom[1]+1]==" ǁ "))):

                print("Nije moguce pomeranje figure u tom smeru, promenite smer kretanja")

        elif(where=="dl" and  ((pom[0]==n-2 ) or (pom[1]==0 or pom[1]==1)

                    or (tabla[pom[0]+1][pom[1]]=="===" and tabla[pom[0]+1][pom[1]-2]=="===" )

                    or (tabla[pom[0]][pom[1]-1]==" ǁ " and tabla[pom[0]+2][pom[1]-1]==" ǁ " )

                    or (tabla[pom[0]+1][pom[1]]=="===" and tabla[pom[0]][pom[1]-1]==" ǁ ")

                    or (tabla[pom[0]+1][pom[1]-2]=="===" and tabla[pom[0]+2][pom[1]-1]==" ǁ ")

                    )):

                print("Nije moguce pomeranje figure u tom smeru, promenite smer kretanja")

        else:

            moved=True

            Moving(p,where)

    except ValueError:

        print("Nevalidan unos")

        move()

    return

**def Moving()-Vrši pomeranje figure u unetom smeru, ukoliko se na poziciji na kojoj treba da se pomeri nalazi drugi igrač, figurica se pomera za jedno mesto**

def Moving(p, where):

global tabla

    global pozicije

    global n

    global m

    global x1

    global x2

    global o1

    global o2

    global cijiPotez

    pom=pozicije[p]

    lista = []

    lista2 = []

    lista3=[]

    lista3.append([x1[0],x1[1]])

    lista3.append([x2[0],x2[1]])

    lista3.append([o1[0],o1[1]])

    lista3.append([o2[0],o2[1]])

    lista2.append(" O ")

    lista2.append(" X ")

    for key in pozicije.keys():

            lista.append(key)

    if (where=="u"):

        p1=int(pom[0]-4)

        p2=int(pom[0]-2)

        if([p1,pom[1]] in lista3):

            tabla[pom[0]][pom[1]]="   "

            pom[0]-=4

        elif((tabla[p1][pom[1]] in lista) or (tabla[p2][pom[0]] in lista2)):

            tabla[pom[0]][pom[1]]="   "

            pom[0]-=2

        else:

            tabla[pom[0]][pom[1]]="   "

            pom[0]-=4

    elif (where=="d"):

        p1=int(pom[0]+4)

        p2=int(pom[0]+2)

        if([p1,pom[1]] in lista3):

            tabla[pom[0]][pom[1]]="   "

            pom[0]+=4

        elif((tabla[p1][pom[1]] in lista) or (tabla[p2][pom[0]] in lista2)):

            tabla[pom[0]][pom[1]]="   "

            pom[0]+=2

        else:

            tabla[pom[0]][pom[1]]="   "

            pom[0]+=4

    elif (where=="l"):

        p1=int(pom[1]-4)

        p2=int(pom[1]-2)

        if([pom[0],p1] in lista3):

            tabla[pom[0]][pom[1]]="   "

            pom[1]-=4

        elif((tabla[pom[0]][p1] in lista) or (tabla[pom[0]][p2] in lista2)):

            tabla[pom[0]][pom[1]]="   "

            pom[1]-=2

        else:

            tabla[pom[0]][pom[1]]="   "

            pom[1]-=4

    elif (where=="r"):

        p1=int(pom[1]+4)

        p2=int(pom[1]+2)

        if([pom[0],p1] in lista3):

            tabla[pom[0]][pom[1]]="   "

            pom[1]+=4

        elif((tabla[pom[0]][p1] in lista) or (tabla[pom[0]][p2] in lista2)):

            tabla[pom[0]][pom[1]]="   "

            pom[1]+=2

        else:

            tabla[pom[0]][pom[1]]="   "

            pom[1]+=4

    elif (where=="ur"):

            tabla[pom[0]][pom[1]]="   "

            pom[0]-=2

            pom[1]+=2

    elif (where=="ul"):

            tabla[pom[0]][pom[1]]="   "

            pom[0]-=2

            pom[1]-=2

    elif (where=="dr"):

            tabla[pom[0]][pom[1]]="   "

            pom[0]+=2

            pom[1]+=2

    elif (where=="dl"):

            tabla[pom[0]][pom[1]]="   "

            pom[0]+=2

            pom[1]-=2

    startPos()

    if cijiPotez=="o":

        cijiPotez="x"

    elif  cijiPotez=="x":

        cijiPotez="o"

    return

**def update()-vrši update prikaza tabele**

def update():

     global tabla

    global pozicije

    global x1

    global x2

    global o1

    global o2

    global n

    global m

    tabla[x1[0]][x1[1]]=" X "

    tabla[x2[0]][x2[1]]=" X "

    tabla[o1[0]][o1[1]]=" O "

    tabla[o2[0]][o2[1]]=" O "

    for key in pozicije.keys():

        pom=pozicije[key]

        tabla[pom[0]][pom[1]]=key

    a = np.array(tabla)

    for line in a:

       print ('  '.join(map(str, line)))

**def game()- vrši logiku igre, poziva fje za unos, za unos zidova i pomeranje figura**

def game():

  global tabla

    global x1

    global x2

    global o1

    global o2

    global pozicije

    global n

    global m

    global xZidovi

    global oZidovi

    global moved

    inputT()

    tabla=Tabla(n,m,x1,x2,o1,o2)

    statesOfPlayer(" X ")

    while (not endGame()):

        print("Korisnik " + cijiPotez +" je na potezu")

        while(not moved):

            move()

        moved=False

        update()

        zid()

        update()

**def main()- vrši logiku za mogućnost ponovnog igranja igre nakon jedne partije**

def main():

    game()

    print("Da li zelite da igrate ponovo?")

    p=input()

    while ("da" in p) or ("Da" in p) :

        game()

        print("Da li zelite da igrate ponovo?")

        p=input()

main()

**def findPath()-pronalazi put između startne i krajnje pozicije. Vraća put ukoliko postoji.**

def findPath(start, end):

    global n

    global m

    def h(x):

        return abs(end[0]-x[0])+abs(end[1]-x[1])

    found\_end = False

    open\_set = set()

    open\_set.add(start)

    pq = PriorityQueue()

    pq.put((0, start))

    closed\_set = set()

    g = {}

    prev\_nodes = {}

    g[start] = 0

    prev\_nodes[start] = None

    while len(open\_set) > 0 and (not found\_end):

        node = pq.get()[1]

        if node not in open\_set:

            continue

        if node == end:

            found\_end = True

            break

        for dx, dy in zip([-2, 2, 0, 0, -2, -2, 2, 2], [0, 0, -2, 2, 2, -2, 2, -2]):

            c = (node[0]+dx, node[1]+dy)

            if c[0] >= 0 and c[1] >= 0 and c[0] <= n-1 and c[1] <= m-1 and isValid(c,open\_set,closed\_set,dx,dy):

                f = g[node] + 1 + h(c)

                if c not in open\_set and c not in closed\_set:

                    open\_set.add(c)

                    prev\_nodes[c] = node

                    g[c] = g[node] + 1

                    pq.put((f,c))

                else:

                    if g[c] > g[node] + 1 :

                        g[c] = g[node] + 1

                        prev\_nodes[c] = node

                        if c in closed\_set:

                                closed\_set.remove(c)

                                open\_set.add(c)

                        pq.put((f,c))

        open\_set.remove(node)

        closed\_set.add(node)

    path = []

    if found\_end:

        prev = end

        while prev\_nodes[prev] is not None:

            path.append(prev)

            prev = prev\_nodes[prev]

        path.append(start)

        path.reverse()

    return path

**def isValid()- Proverava da li je određeni smer putanje validan**

def isValid(c,open\_set,closed\_set,dx,dy):

        global tabla

        global n

        global m

        if(( c[0]+1>n-1) and dx==2 and dy==0):

            return False

        else:

            if(tabla[c[0]+1][c[1]]=="===" and dx==-2 and dy==0): #u

                return False

        if(( c[1]+1>m-1) and dx==0 and dy==2):

            return False

        else:

            if(tabla[c[0]][c[1]+1]==" ǁ "and dx==0 and dy==-2):#l

                    return False

        if(( c[0]-1<0 ) and (dx==-2 and dy==0)):

            return False

        else:

            if(tabla[c[0]-1][c[1]]=="==="and dx==2 and dy==0): #d

                return False

        if(( c[1]-1<0 ) and dx==0 and dy==-2):

            return False

        else:

            if ( tabla[c[0]][c[1]-1]==" ǁ " and dx==0 and dy==2): # r

                return False

        if(dx==-2 and dy==2):

            if( c[0]-2<0 or c[0]+2>n-1 or c[1]-2<0 or c[1]+2>m-1 or c[0]-1<0 or c[0]+1>n-1 or c[1]-1<0 or c[1]+1>m-1):

                return False

            else:

                if (dx==-2 and dy==2 and (tabla[c[0]+1][c[1]]=="===" and tabla[c[0]+1][c[1]-2]=="===")

                        or (tabla[c[0]+2][c[1]-1]==" ǁ " and tabla[c[0]][c[1]-1]==" ǁ ")

                        or (tabla[c[0]+1][c[1]-2]=="===" and tabla[c[0]+2][c[1]-1]==" ǁ ")

                        or (tabla[c[0]+1][c[1]]=="===" and tabla[c[0]][c[1]-1]==" ǁ ")): #ur

                        return False

        if(dx==-2 and dy==-2):

            if( c[0]-2<0 or c[0]+2>n-1 or c[1]-2<0 or c[1]+2>m-1 or c[0]-1<0 or c[0]+1>n-1 or c[1]-1<0 or c[1]+1>m-1):

                return False

            else:

                if (dx==-2 and dy==-2 and (tabla[c[0]+1][c[1]+2]=="===" and tabla[c[0]+1][c[1]]=="===")

                            or (tabla[c[0]][c[1]+1]==" ǁ " and tabla[c[0]+2][c[1]+1]==" ǁ ")

                            or (tabla[c[0]+1][c[1]+2]=="===" and tabla[c[0]+2][c[1]+1]==" ǁ ")

                            or (tabla[c[0]+1][c[1]]=="===" and tabla[c[0]][c[1]+1]==" ǁ ")): # ul

                            return False

        if(dx==+2 and dy==+2):

            if( c[0]-2<0 or c[0]+2>n-1 or c[1]-2<0 or c[1]+2>m-1 or c[0]-1<0 or c[0]+1>n-1 or c[1]-1<0 or c[1]+1>m-1):

                return False

            else:

                if (dx==+2 and dy==+2 and (tabla[c[0]-1][c[1]]=="===" and tabla[c[0]-1][c[1]-2]=="===" )

                        or (tabla[c[0]][c[1]-1]==" ǁ " and tabla[c[0]-2][c[1]-1]==" ǁ " )

                        or (tabla[c[0]-1][c[1]-2]=="===" and tabla[c[0]-2][c[1]-1]==" ǁ ")

                        or (tabla[c[0]-1][c[1]]=="===" and tabla[c[0]][c[1]-1]==" ǁ ")): # dr

                        return False

        if(dx==+2 and dy==-2):

            if( c[0]-2<0 or c[0]+2>n-1 or c[1]-2<0 or c[1]+2>m-1 or c[0]-1<0 or c[0]+1>n-1 or c[1]-1<0 or c[1]+1>m-1):

                return False

            else:

                    if (dx==+2 and dy==-2 and (tabla[c[0]-1][c[1]]=="===" and tabla[c[0]+1][c[1]-2]=="===" )

                            or (tabla[c[0]][c[1]+1]==" ǁ " and tabla[c[0]-2][c[1]+1]==" ǁ " )

                            or (tabla[c[0]-1][c[1]]=="===" and tabla[c[0]][c[1]+1]==" ǁ ")

                            or (tabla[c[0]-1][c[1]+2]=="===" and tabla[c[0]-2][c[1]+1]==" ǁ ")): # dl

                            return False

        return True

**def checkWall()- Proverava da li se dodavanjem zida, ograđuje neka figura/ da li postoje putanje iz figura do ciljnih pozicija.**

def checkWall():

    global pozicije

    global x1

    global x2

    global o1

    global o2

    lista=[]

    for key in pozicije.keys():

            lista.append(key)

    for l in lista:

        if(findPath((pozicije[l][0],pozicije[l][1]),x1)==[]):

            return False

        if(findPath((pozicije[l][0],pozicije[l][1]),x2)==[]):

            return False

        if(findPath((pozicije[l][0],pozicije[l][1]),o1)==[]):

            return False

        if(findPath((pozicije[l][0],pozicije[l][1]),o2)==[]):

            return False

    return True

**def statesOfPlayer()- Vraća sva moguća sledeća stanja određenog igrača**

def statesOfPlayer(koIgra):

    if(koIgra==" X "):

        states('px2')

        states('px1')

        print(len(listaStates))

        update()

    else:

        states('po1')

        states('po2')

        print(len(listaStates))

**def states()- Vraća sva moguća sledeća stanja određene figure igrača, proverava sve moguće pravce kretanja figure igrača i poziva funkciju za proveru postavljanja zidova**

def states(koIgra):

    global xZidovi

    global oZidovi

    zidovi=0

    if(koIgra=='px1' or koIgra=='px2'):

        zidovi=xZidovi

    else: zidovi=oZidovi

    global pozicije

    pom=pozicije[koIgra]

    global tabla

    global listaStates

    if(pom[0]!=0 and pom[0]!=2 and pom[0]!=1 and tabla[pom[0]-3][pom[1]]!="===" and tabla[pom[0]-1][pom[1]]!="===" ): #u

        potez="u" #koji je potez

        if(zidovi>0):

            zidStates(potez,koIgra)

        else:

            listaStates.append(makeNewState(koIgra,[0,0],"",potez))

    if(pom[0]!=n-2 and pom[0]!=n-4 and tabla[pom[0]+3][pom[1]]!="===" and tabla[pom[0]+1][pom[1]]!="===" ): #d

        potez="d"

        if(zidovi>0):

            zidStates(potez,koIgra)

        else: listaStates.append(makeNewState(koIgra,[0,0],"",potez))

    if(pom[1]!=0 and pom[1]!=2 and pom[1]!=1 and tabla[pom[0]][pom[1]-3]!=" ǁ " and tabla[pom[0]][pom[1]-1]!=" ǁ " ): #l

        potez="l"

        if(zidovi>0):

            zidStates(potez,koIgra)

        else: listaStates.append(makeNewState(koIgra,[0,0],"",potez))

    if(pom[1]!=m-2 and pom[1]!=m-4 and tabla[pom[0]][pom[1]+3]!=" ǁ " and tabla[pom[0]][pom[1]+1]!=" ǁ " ): #r

        potez="r"

        if(zidovi>0):

            zidStates(potez,koIgra)

        else: listaStates.append(makeNewState(koIgra,[0,0],"",potez))

    if((pom[0]!=0) and (pom[1]!=m-2 and pom[1]!=m-4)

                    and (tabla[pom[0]-1][pom[1]]!="===" or tabla[pom[0]-1][pom[1]+2]!="===")

                    and (tabla[pom[0]-2][pom[1]+1]!=" ǁ " or tabla[pom[0]][pom[1]+1]!=" ǁ ")

                    and (tabla[pom[0]-1][pom[1]+2]!="===" or tabla[pom[0]-2][pom[1]+1]!=" ǁ ")

                    and (tabla[pom[0]-1][pom[1]]!="===" or tabla[pom[0]][pom[1]+1]!=" ǁ ")): #ur

        potez="ur"

        if(zidovi>0):

            zidStates(potez,koIgra)

        else: listaStates.append(makeNewState(koIgra,[0,0],"",potez))

    if((pom[0]!=0) and (pom[1]!=0 and pom[1]!=1)

                    and (tabla[pom[0]-1][pom[1]-2]!="===" or tabla[pom[0]-1][pom[1]]!="===")

                    and (tabla[pom[0]][pom[1]-1]!=" ǁ " or tabla[pom[0]-2][pom[1]-1]!=" ǁ ")

                    and (tabla[pom[0]-1][pom[1]-2]!="===" or tabla[pom[0]-2][pom[1]-1]!=" ǁ ")

                    and (tabla[pom[0]-1][pom[1]]!="===" or tabla[pom[0]][pom[1]-1]!=" ǁ ")): #ul

        potez="ul"

        if(zidovi>0):

            zidStates(potez,koIgra)

        else: listaStates.append(makeNewState(koIgra,[0,0],"",potez))

    if((pom[0]!=n-2) and(pom[1]!=m-2 and pom[1]!=m-4)

                    and (tabla[pom[0]+1][pom[1]]!="===" or tabla[pom[0]+1][pom[1]+2]!="===" )

                    and (tabla[pom[0]][pom[1]+1]!=" ǁ " or tabla[pom[0]+2][pom[1]+1]!=" ǁ " )

                    and (tabla[pom[0]+1][pom[1]+2]!="===" or tabla[pom[0]+2][pom[1]+1]!=" ǁ ")

                    and (tabla[pom[0]+1][pom[1]]!="===" or tabla[pom[0]][pom[1]+1]!=" ǁ ")): #dr

        potez="dr"

        if(zidovi>0):

            zidStates(potez,koIgra)

        else: listaStates.append(makeNewState(koIgra,[0,0],"",potez))

    if((pom[0]!=n-2) and (pom[1]!=0 and pom[1]!=1)

                    and (tabla[pom[0]+1][pom[1]]!="===" or tabla[pom[0]+1][pom[1]-2]!="===" )

                    and (tabla[pom[0]][pom[1]-1]!=" ǁ " or tabla[pom[0]+2][pom[1]-1]!=" ǁ " )

                    and (tabla[pom[0]+1][pom[1]]!="===" or tabla[pom[0]][pom[1]-1]!=" ǁ ")

                    and (tabla[pom[0]+1][pom[1]-2]!="===" or tabla[pom[0]+2][pom[1]-1]!=" ǁ ")): #dl

        potez="dl"

        if(zidovi>0):

            zidStates(potez,koIgra)

        else: listaStates.append(makeNewState(koIgra,[0,0],"",potez))

    return len(listaStates)

**def zidStates()- Ukoliko mogu da se postave zidovi u sledećem potezu, proverava sve moguće lokacije postavljanja zidova.**

def zidStates(potez,koIgra):

    global tabla

    global n

    global m

    global pozicije

    global listaStates

    p=[]

    z=[]

    for i in range(n-2):

        for j in range(m-2):

            if i%2==1 and j%2==0:

                if(i<n-1 and j<m-1 and i > 0 and j > 0 and tabla[i-1][j-2] != "===" and tabla[i-1][j] != "===" and tabla[i-2][j-1] != " ǁ " or tabla[i][j-1] != " ǁ "):

                    p=[i,j]

                    if(checkWall()):

                        makeNewState(koIgra,p,"plavi",potez)

            if j%2==1 and i%2==0:

                if(i<n-1 and j<m-1 and i > 0 and j > 0 and tabla[i-1][j-2] != "===" or tabla[i-1][j] != "===" and tabla[i-2][j-1] != " ǁ " and tabla[i][j-1] != " ǁ "):

                    z=[i,j]

                    if(checkWall()):

                        makeNewState(koIgra,z,"zeleni",potez)

**def makeNewStates()- Pravi novo stanje iz prethodnog, na njemu odigrava potez i čuva to stanje u listu svih mogućih sledećih stanja.**

def makeNewState(koIgra,zid,vrstaZida,potez):

    global pozicije

    global x1

    global x2

    global o1

    global o2

    pom=pozicije[koIgra]

    ispis=""

    tablaDup = [ [" "  for i in range(m)] for j in range(n) ]

    for i in range (n-1):

        for j in range(m):

            tablaDup[i][j]=tabla[i][j]

    if(koIgra=='px1' or koIgra=='px2'):

        ispis=" X "

    else: ispis=" O "

    lista = []

    lista2 = []

    lista3=[]

    lista3.append([x1[0],x1[1]])

    lista3.append([x2[0],x2[1]])

    lista3.append([o1[0],o1[1]])

    lista3.append([o2[0],o2[1]])

    lista2.append(" O ")

    lista2.append(" X ")

    for key in pozicije.keys():

            lista.append(key)

    if (potez=="u"):

        p1=int(pom[0]-4)

        p2=int(pom[0]-2)

        if([p1,pom[1]] in lista3):

            tablaDup[pom[0]][pom[1]]="   "

            tablaDup[pom[0]-4][pom[1]]=ispis

        elif((tablaDup[p1][pom[1]] in lista) or (tablaDup[p2][pom[0]] in lista2)):

            tablaDup[pom[0]][pom[1]]="   "

            tablaDup[pom[0]-2][pom[1]]=ispis

        else:

            tablaDup[pom[0]][pom[1]]="   "

            tablaDup[pom[0]-4][pom[1]]=ispis

    elif (potez=="d"):

        p1=int(pom[0]+4)

        p2=int(pom[0]+2)

        if([p1,pom[1]] in lista3):

            tablaDup[pom[0]][pom[1]]="   "

            tablaDup[pom[0]+4][pom[1]]=ispis

        elif((tablaDup[p1][pom[1]] in lista) or (tablaDup[p2][pom[0]] in lista2)):

            tablaDup[pom[0]][pom[1]]="   "

            tablaDup[pom[0]+2][pom[1]]=ispis

        else:

            tablaDup[pom[0]][pom[1]]="   "

            tablaDup[pom[0]+4][pom[1]]=ispis

    elif (potez=="l"):

        p1=int(pom[1]-4)

        p2=int(pom[1]-2)

        if([pom[0],p1] in lista3):

            tablaDup[pom[0]][pom[1]]="   "

            tablaDup[pom[0]][pom[1]-4]=ispis

        elif((tablaDup[pom[0]][p1] in lista) or (tablaDup[pom[0]][p2] in lista2)):

            tablaDup[pom[0]][pom[1]]="   "

            tablaDup[pom[0]][pom[1]-2]=ispis

        else:

            tablaDup[pom[0]][pom[1]]="   "

            tablaDup[pom[0]][pom[1]-4]=ispis

    elif (potez=="r"):

        p1=int(pom[1]+4)

        p2=int(pom[1]+2)

        if([pom[0],p1] in lista3):

            tablaDup[pom[0]][pom[1]]="   "

            tablaDup[pom[0]][pom[1]+4]=ispis

        elif((tablaDup[pom[0]][p1] in lista) or (tablaDup[pom[0]][p2] in lista2)):

            tabla[pom[0]][pom[1]]="   "

            tablaDup[pom[0]][pom[1]+2]=ispis

        else:

            tablaDup[pom[0]][pom[1]]="   "

            tablaDup[pom[0]][pom[1]+4]=ispis

    elif (potez=="ur"):

            tablaDup[pom[0]][pom[1]]="   "

            tablaDup[pom[0]-2][pom[1]+2]=ispis

    elif (potez=="ul"):

            tablaDup[pom[0]][pom[1]]="   "

            tablaDup[pom[0]-2][pom[1]-2]=ispis

    elif (potez=="dr"):

            tablaDup[pom[0]][pom[1]]="   "

            tablaDup[pom[0]+2][pom[1]+2]=ispis

    elif (potez=="dl"):

            tablaDup[pom[0]][pom[1]]="   "

            tablaDup[pom[0]+2][pom[1]-2]=ispis

    if(vrstaZida=="plavi"):

        tablaDup[zid[0]][zid[1]+2] = "==="

        tablaDup[zid[0]][zid[1]] = "==="

    else:

        if(vrstaZida=="zeleni"):

            tablaDup[zid[0]+2][zid[1]] = " ǁ

            tablaDup[zid[0]][zid[1]] = " ǁ "

    listaStates.append(tablaDup)

    return tablaDup