```
###
#    Projeto 2 de Fundamentos da ProgramaÃ§Ã£o
#    TomÃ¡s SimÃµes ist1102416
###
from functools import reduce


###
#   TAD posicao
#
#   O TAD posicao Ã© usado para representar uma posiÃ§Ã£o (x; y) de um prado
arbitrariamente
#   grande, sendo x e y dois valores inteiros nÃ£o negativos.
#
#   RepresentaÃ§Ã£o interna: posicao = (x,y) -> Lista de dois ints
###

class posicao:
    def __init__(self, x, y) -> None:
        if isinstance(x, int) and isinstance(y, int):
            if x >= 0 and y >= 0:
                self.x = x
                self.y = y

    def __repr__(self) -> str:
        return str((self.x, self.y))

    def __hash__(self):
        return hash((self.x, self.y))

    def __eq__(self, other) :
        return self.__dict__ == other.__dict__


###
#    Construtores
###

def cria_posicao(x: int, y: int) -> posicao:
    '''
        Recebe os valores correspondentes Ã s coordenadas de uma
        posiÃ§Ã£o e devolve a posiÃ§Ã£o correspondente. O construtor verifica a validade
        dos seus argumentos, gerando um ValueError.
    '''
    if isinstance(x, int) and isinstance(y, int):
        if x >= 0 and y >= 0:
            return posicao(x, y)

    raise ValueError("cria_posicao: argumentos invalidos")

def cria_copia_posicao(p: posicao) -> posicao:
    '''
        Recebe uma posiÃ§Ã£o p e devolve uma cÃ³pia nova da posiÃ§Ã£o.
    '''
    return posicao(p.x, p.y)


###
#   Seletores
###

def obter_pos_x(p: posicao) -> int:
    '''
        Devolve a componente x da posiÃ§Ã£o p
    '''
    return p.x

def obter_pos_y(p: posicao) -> int:
    '''
```

```
        Devolve a componente y da posiÃ§Ã£o p
    '''
    return p.y


###
#   Reconhecedor
###

def eh_posicao(arg) -> bool:
    '''
        Devolve True caso o seu argumento seja um TAD posicao e
        False caso contrÃ¡rio.
    '''
    return isinstance(arg, posicao)


###
#   Teste
###

def posicoes_iguais(p1: posicao, p2: posicao) -> bool:
    '''
        devolve True apenas se p1 e p2 sÃ£o posiÃ§Ãµes e sÃ£o
        iguais.
    '''
    if p1 == p2:
        return True

    return False


###
#   Transformador
###
def posicao_para_str(p: posicao) -> str:
    '''
        Devolve a cadeia de caracteres `(x, y)' que representa o
        seu argumento, sendo os valores x e y as coordenadas de p.
    '''
    return str(p)


###
#   FunÃ§Ãµes de alto nÃvel
###

def obter_posicoes_adjacentes(p: posicao) -> tuple:
    '''
        devolve um tuplo com as posiÃ§~oes adjacentes Ã  posiÃ§Ã£o
        p, comeÃ§ando pela posiÃ§Ã£o acima de p e seguindo no sentido horÃ¡rio.
    '''
    res = []
    x = obter_pos_x(p)
    y = obter_pos_y(p)
    if y-1 > 0:
        res.append(cria_posicao(x, y-1))
    res.extend([cria_posicao(x+1, y), cria_posicao(x, y+1)])
    if x-1 >0:
        res.append(cria_posicao(x-1, y))

    return tuple(res)

def ordenar_posicoes(t: tuple) -> tuple:
    '''
        devolve um tuplo contendo as mesmas posiÃ§Ãµes do tuplo fornecido
        como argumento, ordenadas de acordo com a ordem de leitura do prado.
    '''
    return tuple(sorted(list(t), key=lambda x: (obter_pos_y(x), obter_pos_x(x))))
```

```
###
#   TAD animal
#
#   O TAD animal Ã© usado para representar os animais do simulador de ecossistemas que
#    habitam o prado, existindo de dois tipos: predadores e presas. Os predadores s~ao
caracterizados
#    pela espÃ©cie, idade, frequ^encia de reproduÃ§Ã£o, fome e frequ^encia de
alimentaÃ§Ã£o.
#    As presas s~ao apenas caracterizadas pela espÃ©cie, idade e frequ^encia de
reproduÃ§Ã£o.
#
#   RepresentaÃ§Ã£o interna: posicao = (x,y) -> Lista de dois ints
###

class animal:
    def __init__(self, tipo, especie, freq_reprod, freq_alim, idade=0, fome=0) -> None:
        self.tipo = tipo
        self.especie = especie
        self.idade = idade
        self.freq_reprod = freq_reprod
        self.fome = fome
        self.freq_alim = freq_alim

    def __repr__(self) -> str:
        return str(
            {
            "tipo" : self.tipo,
            "especie" : self.especie,
            "idade": self.idade,
            "freq_reprod" : self.freq_reprod,
            "fome" : self.fome,
            "freq_alim" : self.freq_alim
            }
        )

    def __eq__(self, other) :
        return self.__dict__ == other.__dict__


###
#   Construtor
###

def cria_animal(s: str, r:int, a:int) -> animal:
    '''
        cria_animal(s, r, a) recebe uma string nÃ£o vazia s
        s = espÃ©cie
        r = freq reprod
        a = freq alim

        TODO
    '''
    if isinstance(s, str) and s:
        if isinstance(r, int) and r > 0:
            if isinstance(a, int) and a >= 0:
                if a > 0:
                    return animal("predador", especie=s, freq_reprod=r, freq_alim=a)
                return animal("presa", especie=s, freq_reprod=r, freq_alim=0)

    raise ValueError("cria_animal: argumentos invalidos")

def cria_copia_animal(a: animal) -> animal:
    '''
        TODO : DescriÃ§Ã£o
        Garantimos que estamos a criar deep copy do objeto
    '''
    return animal(a.tipo, a.especie, a.freq_reprod, a.freq_alim, idade=a.idade,
```

```
    fome=a.fome)


    ###
    #   Seletores
    ###

    def obter_especie(a: animal) -> str:
        '''
            TODO : descriÃ§Ã£o
        '''
        return str(a.especie)

    def obter_freq_reproducao(a: animal) -> int:
        '''
            obter_freq_reproducao(a) devolve a freq_reprod do animal a.
        '''
        return int(a.freq_reprod)

    def obter_freq_alimentacao(a: animal) -> int:
        '''
            obter_freq_alimentacao(a) devolve a freq_aliment do animal a.
        '''
        if a.tipo == "presa":
            return 0

        return int(a.freq_alim)

    def obter_idade(a: animal) -> int:
        '''
            obter_idade(a) devolve a idade do animal a.
        '''
        return int(a.idade)

    def obter_fome(a: animal) -> int:
        '''
            obter_fome(a) devolve a fome do animal a.
        '''
        if a.tipo == "presa":
            return 0

        return int(a.fome)


    ###
    #   Modificadores
    ###

    def aumenta_idade(a: animal) -> animal:
        '''
            aumenta idade(a) modifica destrutivamente o animal a incrementando o valor
            da sua idade em uma unidade, e devolve o prÃ³prio animal.
        '''
        a.idade += 1
        return a

    def reset_idade(a: animal) -> animal:
        '''
            reset_idade(a) modifica destrutivamente o animal a definindo o valor da sua
            idade igual a 0, e devolve o prÃ³prio animal
        '''
        a.idade = 0
        return a

    def aumenta_fome(a: animal) -> animal:
        '''
            aumenta fome(a) modifica destrutivamente o animal predador a incremen-
            tando o valor da sua fome em uma unidade, e devolve o pr oprio animal. Esta
            operacao nao modifica os animais presa.
```

```
    '''
    if eh_presa(a):
        return a
    a.fome += 1
    return a

def reset_fome(a: animal) -> animal:
    '''
        reset_fome(a) modifiuca destrutivamente o animal predador a definindo
        o valor da sua fome igual a 0, e devolve o proprio animal. Esta
        operaÃ§Ã£o nÃ£o modifica os animais presa.
    '''
    a.fome = 0
    return a


###
#   Reconhecedor
###

def eh_animal(arg) -> bool:
    '''
        eh_animal(arg) devolve True caso o seu argumento seja um TAD animal e
        Falso caso contrÃ¡rio.
    '''
    if isinstance(arg, animal):
        return True

    return False

def eh_predador(arg) -> bool:
    '''
        eh_predador(arg) devolve True caso o seu argumento seja um TAD animal
        do tipo predadr e False caso contrÃ¡rio
    '''
    if eh_animal(arg):
        if arg.tipo == "predador":
            return True

    return False

def eh_presa(arg) -> bool:
    '''
        eh_presa(arg) devolve True caso o seu argumento seja um TAD animal do
        tipo presa e False caso contrÃ¡rio
    '''
    if eh_animal(arg):
        if arg.tipo == "presa":
            return True

    return False


###
#   Teste
###

def animais_iguais(a1: animal, a2: animal) -> bool:
    '''
        animais_iguais(a1, a2) devolve True apenas de a1 e a2 sÃ£o animais e
        sÃ£o iguais
    '''
    if eh_animal(a1) and eh_animal(a2):
        if a1 == a2:
            return True

    return False
```

```
###
#   Transformadores
###

def animal_para_char(a: animal) -> str:
    '''
        animal para char(a) devolve a cadeia de caracteres dum Ãºnico elemento
        correspondente ao primeiro carÃ¡cter da espÃ©cie do animal passada por
        argumento, em maiÃºscula para animais predadores e em minÃºscula para
        animais presa.
    '''
    res = str(a.especie)[0]

    if a.tipo == "predador":
        return res.upper()
    else:
        return res.lower()

def animal_para_str(a: animal) -> str:
    '''
        animal_para_str(a) devolve a cadeia de caracteres que representa o
        animal.
    '''
    if eh_predador(a):
        return f"{a.especie} [{a.idade}/{a.freq_reprod};{a.fome}/{a.freq_alim}]"

    return f"{a.especie} [{a.idade}/{a.freq_reprod}]"


###
#   FunÃ§Ãµes Alto NÃvel
###

def eh_animal_fertil(a: animal) -> bool:
    '''
        eh_animal_fertil(a) devolve True caso o animal a tenha atingido a
        idade de reproduÃ§Ã£o e False caso contrÃ¡rio
    '''
    if obter_idade(a) >= obter_freq_reproducao(a):
        return True

    return False

def eh_animal_faminto(a: animal) -> bool:
    '''
        eh animal faminto(a) devolve True caso o animal a tenha atingindo um valor de
        fome igual ou superior Ã  sua frequÃªncia de alimentaÃ§Ã£o e False caso
contrÃ¡rio. As
        presas devolvem sempre False.

    '''
    if eh_presa(a):
        return False
    if obter_fome(a) >= obter_freq_alimentacao(a):
        return True

    return False

def reproduz_animal(a: animal) -> animal:
    '''
        reproduz animal(a) recebe um animal a devolvendo um novo animal da mesma
        espÃ©cie com idade e fome igual a 0, e modificando destrutivamente o animal
passado
        como argumento a alterando a sua idade para 0.

    '''
    reset_idade(a)
    return reset_fome(reset_idade(cria_copia_animal(a)))
```

```
###
#   TAD prado
#
#   O TAD prado representa o mapa do ecossistema e os animais que se encontram
#   lÃ¡ dentro.
#
#
###

class prado:
    '''
        TODO
    '''
    def __init__(self, cantInfDirPos, obs, animais, posAnimais) -> None:
        '''
         x,y = n colunas, n linhas do prado. (x,y) representa a pos do canto
               inferior direito do prado

        e.g:
            P = (11,4)
            +----------+
            |..........|
            |..........|
            |.........P|
            +----------+

            A representaÃ§Ã£o interna do prado (agora) nÃ£o ignora a moldura.

        '''
        # Conservamos o input original no caso de ser Ãºtil
        self.input_cantInfDirPos = cantInfDirPos
        self.input_obs = obs
        self.input_animais = animais
        self.input_posAnimais = posAnimais


        # RepresentaÃ§Ã£o interna
        self.colunas = obter_pos_x(cantInfDirPos) - 1
        self.linhas = obter_pos_y(cantInfDirPos) - 1
        self.obs = list(obs)
        self.animais = dict(zip(list(map(lambda x: (obter_pos_x(x), obter_pos_y(x)),
posAnimais)), animais))
        pass

    def generate_repr_interna(self):
        '''
            Gera a representaÃ§Ã£o em forma de matrix do prado
        '''
        self.repr = []

        for i in range(self.linhas):
            linha = []
            for j in range(self.colunas):
                linha.append(".")
            self.repr.append(linha)

        for o in self.obs:
                self.repr[obter_pos_y(o)-1][obter_pos_x(o)-1] = "@"

        for key in self.animais:
            self.repr[key[1]-1][key[0]-1] = animal_para_char(self.animais[key])

    def __repr__(self) -> str:
        return str(self.repr)

    def __eq__(self, other):
        if (self.colunas, self.linhas, self.obs, self.animais) == (other.colunas,
other.linhas, other.obs, other.animais):
```

```
            return True
        return False


###
#   Construtor
###

def cria_prado(d: posicao, r: tuple, a:tuple, p:tuple) -> prado:
    '''
        cria_prado(d, r, a, p) recebe uma posiÃ§Ã£o d que
        ocupa a montanha do canto inferior direito do prado,
        um tuplo r de 0 ou mais pos correspondentes aos rochedos,
        um tuplo a de 1 ou mais animais, e um tuplo p da mesma dimensÃ£o do a
        com as pos correspondentes dos animais; devolve o prado que representa
        internamente o mapa e animais presentes. Verifica a validade dos seus
        argumentos gerando um ValueError caso os argumentos nÃ£o sejam
        vÃ¡lidos.
    '''
    try:
        if eh_posicao(d) and isinstance(r, tuple) and isinstance(a, tuple) and
isinstance(p, tuple):
            if len(a) > 0 and len(p) > 0:
                if all(eh_animal(i) for i in a) and all(eh_posicao(i) for i in p):
                    if len(p) == len(a):
                        if len(r) == 0 or all(eh_posicao(i) for i in r):
                            if (
                                all(obter_pos_x(x) in range(1, obter_pos_x(d)) for x in r)
and
                                all(obter_pos_y(x) in range(1, obter_pos_y(d)) for x in r)
and
                                all(obter_pos_x(x) in range(1, obter_pos_x(d)) for x in p)
and
                                all(obter_pos_y(x) in range(1, obter_pos_y(d)) for x in p)
                            ):
                                pRes = prado(d, r, a, p)
                                return pRes
    except:
        raise ValueError('cria_prado: argumentos invalidos')

    raise ValueError('cria_prado: argumentos invalidos')

def cria_copia_prado(m: prado) -> prado:
    '''
        cria_copia_prado(m) recebe um prado e devolve uma nova cÃ³pia do mesmo.
    '''
    return cria_prado(m.input_cantInfDirPos, m.input_obs, m.input_animais,
m.input_posAnimais)


###
#   Seletores
###

def obter_tamanho_x(m: prado) -> int:
    '''
        obter_tamanho_x(m) devolve o valor inteiro que corresponde Ã  dimensÃ£o x
        do prado.
    '''
    return obter_pos_x(m.input_cantInfDirPos) + 1

def obter_tamanho_y(m: prado) -> int:
    '''
        obter_tamanho_y(m) devolve o valor inteiro que corresponde Ã  dimensÃ£o y
        do prado.
    '''
    return obter_pos_y(m.input_cantInfDirPos) + 1

def obter_numero_predadores(m: prado) -> int:
```

```
    '''
        obter_numero_predador(m) devolve o nÃºmero de animais predadores no prado.
    '''
    return int(reduce(lambda x,y: x+1 if (eh_predador(y)) else x,
list(m.animais.values()), 0))

def obter_numero_presas(m: prado) -> int:
    '''
        obter_numero_presas(m) devolve o nÃºmero de animais presa no prado.
    '''
    return int(reduce(lambda x,y: x+1 if (eh_presa(y)) else x, list(m.animais.values()),
0))

def obter_posicao_animais(m: prado) -> tuple:
    '''
        obter_posicoes_animais(m) devolve um tuplo contendo as posiÃ§Ãµes do prado
ocupadas
        por animais, ordenadas de acordo com a ordem de leitura.
    '''
    if m.animais:
        return tuple(map(lambda x: cria_posicao(x[0], x[1]),
sorted(list(m.animais.keys()), key=lambda x: (x[1], x[0]))))

    return ()

def obter_animal(m: prado, p: posicao) -> animal:
    '''
        obter_animal(m, p) devolve o animal que se encontra na pos p.
    '''
    return m.animais[(obter_pos_x(p), obter_pos_y(p))]


###
#    Modificadores
###

def eliminar_animal(m: prado, p: posicao) -> prado:
    '''
        eliminar_animal(m, p) modifica destrutivamente o prado m eliminado
        o animal da posiÃ§Ã£o p deixando-a livre. Devolve o prÃ³prio prado.
    '''
    del m.animais[(obter_pos_x(p), obter_pos_y(p))]
    return m

def mover_animal(m: prado, p1: posicao, p2: posicao) -> prado:
    '''
        mover_animal(m, p1, p2) modifica destrutivamente o prado m movimentando
        o animal da posiÃ§Ã£o p1 para a nova posiÃ§Ã£o p2, deixando livre a posiÃ§Ã£o onde
        se encontrava. Devolve o prÃ³prio prado.
    '''
    if p1 != p2:
        m.animais[(obter_pos_x(p2), obter_pos_y(p2))] = m.animais[(obter_pos_x(p1),
obter_pos_y(p1))]
        del m.animais[(obter_pos_x(p1), obter_pos_y(p1))]
    return m

def inserir_animal(m: prado, a: animal, p: posicao) -> prado:
    '''
        inserir_animal(m, a, p) modifica destrutivamente o prado m acrescentando na
        posiÃ§Ã£o p do prado o animal a passado com argumento. Devolve o prÃ³prio
        prado.
    '''
    m.animais[(obter_pos_x(p), obter_pos_y(p))] = a
    return m


###
#    Reconhecedores
###
```

```
def eh_prado(arg) -> bool:
    '''
        eh_prado(arg) devolve True caso o seu argumento seja um TAD prado e
        False caso contrÃ¡rio.
    '''
    if isinstance(arg, prado):
        return True

    return False

def eh_posicao_animal(m: prado, p: posicao) -> bool:
    '''
        eh_posiÃ§Ã£o_animal(m, p) devolve True apenas no caso da posiÃ§Ã£o p do prado
        estar ocupada por um animal.
    '''
    if any(posicoes_iguais(cria_posicao(x[0], x[1]), p) for x in m.animais):
        return True

    return False

def eh_posicao_obstaculo(m: prado, p: posicao) -> bool:
    '''
        eh_posicao_obstaculo(m, p) devolve True apenas no caso da posiÃ§Ã£o p do prado
        estar ocupada por montanha ou rochedo.
    '''
    if  (
        any(posicoes_iguais(x, p) for x in m.obs) or obter_pos_x(p) == 0 or obter_pos_y(p)
== 0 or
        obter_pos_x(p) >= obter_tamanho_x(m)-1 or obter_pos_y(p) >= obter_tamanho_y(m)-1
        ):
        return True
    return False

def eh_posicao_livre(m: prado, p: posicao) -> bool:
    '''
        eh_posicao_livre(m, p) devolve True apenas no caso da posiÃ§Ã£o p do prado
corresponder
        a um espaÃ§o livre (sem animais, nem obstÃ¡culos.)
    '''
    if (not (any(posicoes_iguais(cria_posicao(x[0], x[1]), p) for x in m.animais))) and
(not (any(posicoes_iguais(x, p) for x in m.obs))):
        if obter_pos_x(p) != 0 and obter_pos_y != 0:
            if obter_pos_x(p) < obter_tamanho_x(m) - 1 and obter_pos_y(p) <
obter_tamanho_y(m) - 1:
                return True

    return False


###
#    Teste
###

def prados_iguais(p1: prado, p2: prado) -> bool:
    '''
        prados_iguais(p1, p2) devolve True apenas de p1 e p2 forem prados e
        forem iguais.
    '''
    return p1 == p2


###
#    Transformador
###

def prado_para_str(m: prado) -> str:
    '''
        prado_para_str(m) devolve uma cadeira de caracteres que representa
```

```
        o prado.
    '''
    m.generate_repr_interna()

    res = ""

    res += "+"
    for i in range(m.colunas):
        res += "-"
    res += "+\n"

    for l in m.repr:
        res += "|"
        res += ''.join(l)
        res += "|"
        res += "\n"

    res += "+"
    for i in range(m.colunas):
        res += "-"
    res += "+"

    return res


###
#    FunÃ§Ãµes Alto NÃvel
###

def obter_valor_numerico(m: prado, p: posicao) -> int:
    '''
        obter_valor_numerico(m, p) devolve a posiÃ§Ã£o seguinte do animal na
        posiÃ§Ã£o p dentro do prado m de acordo com as regras de movimento dos
        animais no prado.
    '''
    return (obter_tamanho_x(m)) * obter_pos_y(p) + obter_pos_x(p)

def obter_movimento(m: prado, p: posicao) -> posicao:
    '''
        obter_movimento(m, p) devolve a posiÃ§Ã£o seguinte do animal na posiÃ§Ã£o p
        dentro do prado m de acordo com as regras de movimento dos animais
        no prado.
    '''
    n = 0
    j = 0
    posDict = {}
    presasDict = {}
    for pos in obter_posicoes_adjacentes(p):
        if eh_posicao_livre(m, pos):
            posDict[n] = pos
            n += 1
        if eh_posicao_animal(m, pos):
            if eh_predador(obter_animal(m, p)):
                if eh_presa(obter_animal(m, pos)):
                    presasDict[j] = pos
                    j += 1

    N = obter_valor_numerico(m, p)

    if presasDict:
        return presasDict[N%j]
    if posDict:
        return posDict[N%n]
    return p


###
#    FunÃ§Ãµes Adicionais
###
```

```
def geracao(m: prado) -> prado:
    '''
        geracao(m) Ã© a funÃ§Ã£o auxiliar que modifica o prado m de acordo
        com a evoluÃ§Ã£o correspondente a uma geraÃ§Ã£o inteira, e devolve
        o prÃ³prio prado.
    '''
    posicoesExcluidas = []

    for pos in list(obter_posicao_animais(m)):
        if (not posicoesExcluidas) or (not any(posicoes_iguais(x, pos) for x in
posicoesExcluidas)): # Se esta posiÃ§Ã£o nÃ£o estiver excluÃda
            a = obter_animal(m, pos)
            ### Incrementar
            aumenta_idade(a)

            ### MovimentaÃ§Ã£o
            novaPos = obter_movimento(m, pos)

            if eh_presa(a):
                if novaPos != pos:
                    mover_animal(m, pos, obter_movimento(m, pos))
                    if eh_animal_fertil(a):
                        reset_idade(a)
                        inserir_animal(m, reproduz_animal(a), pos)

            if eh_predador(a):
                aumenta_fome(a)
                if eh_posicao_animal(m, novaPos):
                    ### COMER
                    reset_fome(a)
                    eliminar_animal(m, novaPos)
                    mover_animal(m, pos, novaPos)
                    posicoesExcluidas.append(novaPos)
                    if eh_animal_fertil(a):
                        reset_idade(a)
                        inserir_animal(m, reproduz_animal(a), pos)
                else:
                    if obter_movimento(m, pos) != pos:
                        mover_animal(m, pos, novaPos)
                        if eh_animal_fertil(a):
                            reset_idade(a)
                            inserir_animal(m, reproduz_animal(a), pos)

                if eh_animal_faminto(a):
                    eliminar_animal(m, novaPos)

    return m

def simula_ecossistema(f: str, g :int, v: bool) -> tuple:
    '''
        simula_ecossistema(f, g, v) Ã© a funÃ§Ã£o principal que permite simular o
ecossistema de
        um prado. A funÃ§Ã£o recebe uma cadeira de caracteres f, um valor inteiro g e um
        booleano v e devolve o tuplo de dois elementos correspondente ao nÃºmero de
predadores
        e presas no fim da simulaÃ§Ã£o. O argumento v ativa o modo verboso.
    '''
    with open(f, "r") as file:
        dimTuple = eval(file.readline())
        dim = cria_posicao(dimTuple[0], dimTuple[1])

        evaledObs = eval(file.readline())
        obs = ()
        for o in evaledObs:
            obs += (cria_posicao(o[0], o[1]), )


        an = ()
```

```
        anPosics = ()

        for line in file:
            evaledLine = eval(line)
            an += (cria_animal(evaledLine[0], evaledLine[1], evaledLine[2]), )
            anPosics += (cria_posicao(evaledLine[3][0], evaledLine[3][1]), )

        m = cria_prado(dim, obs, an, anPosics)

        # Gen 0
        print(f'Predadores: {obter_numero_predadores(m)} vs Presas:
{obter_numero_presas(m)} (Gen. 0)')
        print(prado_para_str(m))

        prevPredador = obter_numero_predadores(m)
        prevPresas = obter_numero_presas(m)

        for i in range(g):
            m = geracao(m)
            if v:
                if prevPredador != obter_numero_predadores(m) or prevPresas !=
obter_numero_presas(m):
                    print(f'Predadores: {obter_numero_predadores(m)} vs Presas:
{obter_numero_presas(m)} (Gen. {i+1})')
                    print(prado_para_str(m))
            prevPredador = obter_numero_predadores(m)
            prevPresas = obter_numero_presas(m)

        if not v:
            print(f'Predadores: {obter_numero_predadores(m)} vs Presas:
{obter_numero_presas(m)} (Gen. {g})')
            print(prado_para_str(m))

    return (obter_numero_predadores(m), obter_numero_presas(m))


### FunÃ§Ã£o extra para o PradoGUI

def parse_config(f: str) -> prado:
    '''
        FunÃ§Ã£o extra requirida para garantir funcionalidade com
        o PradoGUI.
    '''
    with open(f, "r") as file:
        dimTuple = tuple(map(int, file.readline().strip('(').strip(')\n').split(', ')))
        dim = cria_posicao(dimTuple[0], dimTuple[1])

        obs = ()
        for o in eval(file.readline()):
            obs += (cria_posicao(o[0], o[1]), )


        an = ()
        anPosics = ()

        for line in file:
            evaledLine = eval(line)
            an += (cria_animal(evaledLine[0], evaledLine[1], evaledLine[2]), )
            anPosics += (cria_posicao(evaledLine[3][0], evaledLine[3][1]), )

        m = cria_prado(dim, obs, an, anPosics)

    return m
```