N-Reinas. Algoritmo genético

July 17, 2022

[2]:
```
# Paquetes a utilizar
import random
import numpy as np


# Clase Poblacion
class Poblacion():
    """ Clase que representa una población de tableros de n x n con n reinas␣
 ↪distribuidas en él.

    ...

    Atributos
    ----------

    casilleros: int
        El número de casilleros horizontales/verticales y/o reinas distribuidas␣
 ↪en el tablero.


    Métodos
    ----------

    PobInicial (cantidad):
        Crea una población inicial.

    SetPob ():
        Setea la población.

    GetPob ():
        Devuelve la población contenida en el objeto.


    Score ():
        Retorna el número de reinas bien colocadas para cada integrante de la␣
 ↪población.
```

```
    """

    def __init__(self,casilleros):
        """
        Constructor de la clase.

        Parámetros
        ------------

        casilleros: int
            El número de casilleros horizontales/verticales y/o reinas␣
↪distribuidas en el tablero.

            """
        self.__casilleros = casilleros
        self.__Pob = []


    def PobInicial(self,cantidad):
        """
        Crea una población inicial aleatoria.

        Parámetros
        ------------

        cantidad: int
               El número de integrantes que tendrá la población inicial."""

        Aux = []
        for i in range(cantidad):
            Aux2 = []
            for j in range(self.__casilleros):
                Aux2.append(random.randint(1,self.__casilleros))
            Aux.append(Aux2)
        self.__Pob = Aux


    def SetPob(self,Pob):
        """
        Setea la población.

        Parámetros
        ------------

        Pob: list
```

```
            La población que desea ser ingresada.

        """
        self.__Pob = Pob


    def GetPob(self):
        """
        Devuelve la población contenida en el objeto.

        Return
        -------

        list
            Una lista que contiene a todos los integrantes de la población.

        """
        return self.__Pob


    def Score (self):
        """
        Retorna el número de reinas bien colocadas para cada integrante de la␣
↪población.

        Return
        -------

        list
            Una lista que contiene el número de reinas bien colocadas para cada␣
↪integrante de la población.

        """

        AuxLista = []

        # Para cada individuo de la población

        for i in range(len(self.__Pob)):
            Aux = 0

            # Si la reina está en la misma fila y/o diagonal que otra reina, se␣
↪suma 1 a Aux2

            for j in range(self.__casilleros ):
                if self.__Pob[i].count(self.__Pob[i][j]) == 1:
                    Aux2 = 0
```

```
                    for k in range (self.__casilleros):
                        if self.__Pob[i][j] + j == self.__Pob[i][k] + k or self.
↪__Pob[i][j] - j == self.__Pob[i][k] - k:
                            Aux2 = Aux2 + 1

                    # Si solo está en la misma fila y/o diagonal que si misma,␣
↪se le suma 1 a la cantidad de reinas
                    # bien colocadas en ese individuo

                    if Aux2 == 1:
                        Aux = Aux + 1

            AuxLista.append(Aux)
        return AuxLista


# Clase AlgoritmoGenetico

class AlgoritmoGenetico (Poblacion):

    """ Clase que contiene y ejecuta un algoritmo genético enfocado en resolver␣
↪el problema de N-Reinas.

    ...

    Atributos
    ----------

    casilleros: int
        El número de casilleros horizontales/verticales y/o reinas distribuidas␣
↪en el tablero.


    Métodos
    ----------

    PobInicial (cantidad):
        Crea una población inicial.

    SetPob ():
        Setea la población.

    GetPob ():
        Devuelve la población contenida en el objeto.


    Score ():
```

```
        Retorna el número de reinas bien colocadas para cada integrante de la␣
↪población.

    GetSolucion ():
        Retorna un lista que contiene a un individuo que resuelve el problema␣
↪de las N-Reinas (Si es que dicho individuo fue
        encontrado).

    Iterar(n_iter):
        Realiza n_iter veces el ciclo genético y en el caso de encontrar un␣
↪individuo que sea solución del problema lo devuelve.




    """

    def __init__(self,casilleros):

        """
        Constructor de la clase.

        Parámetros
        ------------

        casilleros: int
            El número de casilleros horizontales/verticales y/o reinas␣
↪distribuidas en el tablero.

            """


        super().__init__(casilleros)
        self.__Solucion = []

        # Como solo los tableros de n x n  con n mayor a 3 tienen solución, en␣
↪caso de ingresar un número de casilleros menor
        # se produce un error.

        if casilleros < 4:
            raise ValueError("El número de casilleros debe ser mayor a 3.␣
↪Tableros mas pequeños no tienen solución.")


    def GetSolucion(self):
```

```
        """
        Retorna un lista que contiene a un individuo que resuelve el problema de las N-Reinas (Si es que dicho individuo fue
        encontrado).
        En caso de no haber encontrado todavía una solución, devuelve un error.

        Return
        -------

        list
            Una lista que contiene a un individuo que resuelve el problema de las N-Reinas.

        """

        if len(self.Solucion) > 0:
            return self.__Solucion
        else:
            raise ValueError("La solución aún no ha sido encontrada. Intente usar el método Iterar.")


    def Iterar(self,n_iter = 10000):
        """
        Realiza n_iter veces el ciclo genético y en el caso de encontrar un individuo que sea solución del problema lo devuelve.
        En caso de no encontrarlo, devuelve un error.

        Parámetros
        ------------

        n_iter: int
            Cantidad máxima de ciclos genéticos.

        Return
        -------

        list
            Una lista que contiene a un individuo que sea solución del problema.
        """

        n = 0
        while n < n_iter:
            Scores = self.Score()

            #Si encuentra la solución óptima, el ciclo se detiene.
```

```
            if max(Scores) == self._Poblacion__casilleros:
                self.__Solucion = self._Poblacion__Pob[Scores.
↪index(max(Scores))]
                return self._Poblacion__Pob[Scores.index(max(Scores))]
                break

            AuxPob = self._Poblacion__Pob
            Aux = []

            # Selección. Se selecciona la mitad que mas Score obtuvo.

            for i in range(int(len(AuxPob) / 2)):
                Aux.append(AuxPob[Scores.index(max(Scores))])
                del AuxPob[Scores.index(max(Scores))]
                del Scores[Scores.index(max(Scores))]

            # Cruzamiento. Se mezclan las segundas mitades de los individuos␣
↪antes seleccionados y se los integra en una lista
            # con los individuos seleccionados sin mezclar

            A = Aux
            B = Aux
            Aux = []
            Aux2 = []
            for i in range(len(B)):
                Aux.append(B[i])
            for i in range(len(A)):
                for j in range(int(len(A[i]) / 2),len(A[i])):
                    A[i][j] , A[i-1][j] = A[i-1][j] , A[i][j]
            for j in range(int(len(A[i]) / 2) , len(A[i])):
                A[len(A)-1][j] , A[0][j] = A[0][j] , A[len(A)-1][j]
            for i in range(len(A)):
                Aux.append(A[i])

            # Mutación. Se elige un índice al azar y se reemplaza ese índice de␣
↪todos los individuos por un número al azar.


            indice_random = random.randint(0,self._Poblacion__casilleros - 1)
            numeros_random = []
            for i in range(len(Aux)):
                numeros_random.append(random.choice(range(1,self.
↪_Poblacion__casilleros +1 )))
            for i in range(len(Aux)):
                Aux3 = []
                for j in range(len(Aux[i])):
```

```
                    if j == indice_random:
                        Aux3.append(numeros_random[i])
                    else:
                        Aux3.append(Aux[i][j])
                Aux2.append(Aux3)


            n = n + 1
            self._Poblacion__Pob = Aux2

        # Si la solución no se encontró, devuelve un error.

        if len(self.__Solucion) == 0:
            raise ValueError("La solución aún no ha sido encontrada. Intente␣
↪ampliando el número de iteraciones")



# Clase Tablero

class Tablero():

    """
    Clase que simula posiciones en un tablero de ajedrez.

    ...

    Atributos
    ----------

    Damas: str
        El símbolo por el cual se van a representar las reinas en el tablero.␣
↪Por defecto las reinas serán representadas
        con una X.


    Métodos
    ----------

    ImprimirTablero(Array):
        Imprime la posición contenida en el Array.

    """

    def __init__(self,Damas = "X"):
        """
        Constructor de la clase.
```

```
        Parámetros
        ------------

        Damas: str
            El símbolo por el cual se van a representar las reinas en el␣
↪tablero. Por defecto las reinas serán representadas
            con una X.



            """
        self.__Damas = Damas



    def ImprimirTablero(self,Array):

        """
        Imprime un tablero de ajedréz en donde los - representan escaques␣
↪vacios y las damas son representadas por el símbolo
        elegido en los parámetros de la clase.

        Parámetros
        ----------

        Array: list
            Lista que representa la posición de las damas en el tablero

        Return
        -------

        Imprime una representación de las damas en el tablero.
        """

        A = Array
        AuxLista = []
        for i in range(len(A)):
            Aux = []
            for j in range(len(A)):
                if A[i] == len(A)-j:
                    Aux.append(self.__Damas)
                else:
                    Aux.append("-")
            AuxLista.append(Aux)
```

```
        for i in range(len(AuxLista)):
            for x in AuxLista:
                print(x[i], end =' ')
            print()



# Clase N_Reinas (Siguiendo el patrón de diseño Facade). La razón para crear␣
 ↪esta clase y seguir el patrón Facade es que con
# N_Reinas proporciona una interfaz simple a un subsistema complejo que␣
 ↪contiene muchas partes móviles. Incluye las partes
# de las clases anteriores cuya interacción sirve solo para resolver lo mínimo␣
 ↪del problema.


class N_Reinas():

    """
    Clase (Facade) que resuelve el problema de N-Reinas usando un algorítmo␣
 ↪genético.

    ...

    Atributos
    ----------

    N_Reinas: int
        El número de casilleros horizontales/verticales y/o reinas distribuidas␣
 ↪en el tablero. Por defecto, 8.

    Iteraciones: int
        Cantidad máxima de ciclos genéticos. Por defecto, un millón.

    PobInicial: int
        El número de integrantes que tendrá la población inicial.

    Damas: str
        El símbolo por el cual se van a representar las reinas en el tablero.␣
 ↪Por defecto las reinas serán representadas
        con una X.


    Métodos
    ----------
```

```
   TableroSolucion ():
       Imprime una solución del problema de N-Reinas. En caso de no␣
↪encontrarla, retornará un error.

   ArraySolucion ():
       Devuelve una lista que contiene una solución del problema de N-Reinas.␣
↪En caso de no encontrarla en la cantidad de iteraciones indicadas, retornará␣
↪un error.

   TableroPobInicial ():
       En el caso de que no haya sido creada, crea una población inicial␣
↪aleatoria e imprime todos los tableros que están contenidos en ella.
       Si la población ya fue creada, imprime los tableros que pertenecen a la␣
↪población inicial.

   ArrayPobAleatoria ():
       Crea una población inicial aleatoria y retorna un array con todos los␣
↪tableros en ella.

   GetPobInicial ():
       Devuelve la polación inicial

   GetSolucion ():
       Devuelve la solución.




   """

   def __init__(self, N_Reinas = 8 , Damas = "X"):

       """
       Constructor de la clase.
       En este constructor se crean dos objetos (Uno de la clase␣
↪AlgoritmoGenetico y otro de Tablero) y se los almacena
       en el objeto instanciado de la clase N-Reinas.

       Parámetros
       ----------

       N_Reinas: int
           El número de casilleros horizontales/verticales y/o reinas␣
↪distribuidas en el tablero. Por defecto, 8.

       Iteraciones: int
```

```
            Cantidad máxima de ciclos genéticos. Por defecto, un millón.

        PobInicial: int
            El número de integrantes que tendrá la población inicial.

        Damas: str
            El símbolo por el cual se van a representar las reinas en el␣
↪tablero. Por defecto las reinas serán representadas
            con una X.

            """

        self.__subsistema1 = AlgoritmoGenetico(N_Reinas)
        self.__subsistema2 = Tablero(Damas)
        self.__Solucion = []
        self.__PobInicial = []



    def TableroSolucion(self,Iteraciones = 1000000):

        """
        Imprime una solución del problema de N-Reinas. En caso de no␣
↪encontrarla, retornará un error.
        Una vez encontrada, se almacena dicha solución para no volver a iterar.

        En caso de que no exista una población inicial generada, se generará␣
↪una con 30 individuos.

        Return
        -------

        Imprime una representación de la solución de N-Reinas.

        """

        # Si la solución ya existe, solo se imprime el tablero.

        if len(self.__Solucion) > 0:
            Aux = self.__subsistema2
            Aux.ImprimirTablero(self.__Solucion)

        # En caso contrario se realizan las iteraciones correspondientes

        else:
            try:
                Aux = self.__subsistema1
```

```
                # Si ya existe una población inicial no se genera una nueva

                if len(self.__PobInicial) > 0:
                    Aux.SetPob(self.__PobInicial)

                # Si no existe una población inicial, se genera una con 30 individuos.

                else:
                    Aux.PobInicial(30)

                    # Guarda la población inicial


                    self.__PobInicial = Aux._Poblacion__Pob

                Solucion = Aux.Iterar(Iteraciones)
                Aux = self.__subsistema2

                # Se imprime el tablero

                Aux.ImprimirTablero(Solucion)

                # Se almacena la solución en la clase

                self.__Solucion = Solucion

            # Si no se encuentra la solución, ocurre un error

            except TypeError:
                raise ValueError ( "No se encontró solución para este número de iteraciones. Pruebe aumentando dicho número")

    def ArraySolucion(self,Iteraciones = 1000000):

        """
        Devuelve una lista que contiene una solución del problema de N-Reinas. En caso de no encontrarla, retornará un error.

        En caso de que no exista una población inicial generada, se generará una con 30 individuos.

        Return
        -------

        list
```

```
            Una lista que contiene a un individuo que sea solución del problema.

        """

        # Si la solución ya existe, solo se devuelve la solución.

        if len(self.__Solucion) > 0:
            return self.__Solucion

        # En caso contrario se realizan las iteraciones correspondientes

        else:
            Aux = self.__subsistema1

            # Si ya existe una población inicial no se genera una nueva

            if len(self.__PobInicial) > 0:
                    Aux.SetPob(self.__PobInicial)

            # Si no existe una población inicial, se genera una con 30 individuos.

            else:
                    Aux.PobInicial(30)
            Solucion = Aux.Iterar(Iteraciones)

            # Guarda la población inicial

            self.__PobInicial = Aux._Poblacion__Pob

            # Si no se encuentra la solución se devuelve un error.

            if Solucion is None :
                raise ValueError ("No se encontró solución para este número de iteraciones. Pruebe aumentando dicho número")

            # Si se encuentra la solución se almacena y se retorna.

            else:
                self.__Solucion = Solucion
                return Solucion

    def TableroPobInicial(self,N_Pob = 30):

        """
        En el caso de que no haya sido creada,
```

```
        crea una población inicial aleatoria e imprime todos los tableros que␣
↪están contenidos en ella.
        Dicha población es almacenada para ser usada por otros métodos en caso␣
↪de ser necesario.

        Si la población ya fue creada, imprime los tableros que pertenecen a la␣
↪población inicial.


        Return
        -------

        Imprime todos los tableros de la población inicial

        """

        Aux = self.__subsistema1

        if len(self.__PobInicial) == 0:
            Aux.PobInicial(N_Pob)
            self.__PobInicial = Aux._Poblacion__Pob

        Aux2 = self.__subsistema2

        # Se imprime el tablero de cada uno de los individuos de la población␣
↪creada.

        for i in range(len(Aux._Poblacion__Pob)):
            print("Tablero número {}. \n".format(i+1))
            Aux2.ImprimirTablero(Aux._Poblacion__Pob[i])
            print("\n")


    def ArrayPobAleatoria(self,N_Pob = 30):

        """
        Crea una población inicial aleatoria y retorna un array con todos los␣
↪tableros en ella.
        Dicha población es almacenada para ser usada por otros métodos en caso␣
↪de ser necesario.

        Return
        -------

        list
```

```
            Una lista que contiene a todos los individuos que estén contenidos␣
↪en la población inicial.

        """

        Aux = self.__subsistema1
        Aux.PobInicial(N_Pob)
        self.__PobInicial = Aux._Poblacion__Pob
        return self.__PobInicial

    def GetPobInicial (self):
        """
        Devuelve la polación inicial

        Return
        ------

        list
            Lista que contiene a todos los individuos que estén contenidos en␣
↪la población inicial.
        """

        if len(self.__PobInicial) == 0:
            raise Exception ("La población inicial aún no ha sido creada")
        else:
            return self.__PobInicial

    def GetSolucion (self):
        """
        Devuelve la solución

        Return
        ------

        list
            Lista que contiene a un individuo que es solución del problema de␣
↪N-Reinas.
        """

        if len(self.__Solucion) == 0:
            raise Exception ("La solución aún no ha sido encontrada. Intente␣
↪usar el método ArraySolucion.")
        else:
            return self.__Solucion
```

[4]:
```
A = N_Reinas(Damas ="D",N_Reinas = 8)

A.TableroSolucion()
```

```
- D - - - - - -
- - - D - - - -
- - - - - D - -
- - - - - - - D
- - D - - - - -
D - - - - - - -
- - - - - - D -
- - - - D - - -
```

[6]:
```
B = N_Reinas(N_Reinas = 10)
print(B.ArraySolucion())
print()

B.TableroSolucion()
```

```
[5, 8, 1, 9, 7, 2, 10, 3, 6, 4]

- - - - - - X - - -
- - - X - - - - - -
- X - - - - - - - -
- - - - X - - - - -
- - - - - - - - X -
X - - - - - - - - -
- - - - - - - - - X
- - - - - - - X - -
- - - - - X - - - -
- - X - - - - - - -
```

[8]:
```
C= N_Reinas()

print(C.ArrayPobAleatoria())
print()
C.TableroPobInicial()
```

```
[[8, 2, 6, 3, 5, 6, 4, 2], [8, 5, 3, 8, 3, 6, 7, 6], [8, 2, 1, 2, 7, 3, 8, 3],
[1, 3, 2, 1, 8, 2, 2, 1], [2, 5, 2, 7, 2, 1, 7, 7], [7, 5, 3, 5, 1, 1, 6, 4],
[4, 2, 8, 8, 2, 6, 1, 3], [7, 8, 6, 4, 5, 1, 6, 2], [7, 2, 7, 6, 4, 1, 3, 3],
[7, 2, 7, 4, 4, 5, 5, 6], [1, 5, 8, 1, 6, 1, 2, 7], [5, 3, 1, 3, 1, 8, 1, 4],
[7, 5, 7, 5, 4, 5, 8, 2], [6, 2, 6, 7, 7, 1, 5, 1], [7, 7, 7, 6, 6, 1, 3, 5],
[7, 8, 6, 2, 1, 1, 6, 4], [3, 7, 6, 6, 2, 1, 8, 1], [8, 4, 3, 2, 4, 3, 4, 4],
```

```
[8, 6, 3, 4, 4, 2, 7, 4], [4, 3, 2, 6, 8, 3, 8, 8], [2, 3, 1, 4, 8, 2, 4, 1],
[3, 2, 3, 7, 7, 6, 8, 3], [2, 4, 1, 3, 4, 2, 7, 3], [7, 5, 1, 6, 7, 7, 7, 2],
[1, 1, 6, 2, 7, 7, 3, 8], [5, 1, 1, 8, 1, 7, 6, 2], [2, 2, 8, 7, 8, 8, 3, 3],
[4, 7, 1, 2, 7, 6, 2, 7], [5, 5, 2, 6, 7, 3, 1, 2], [1, 1, 3, 4, 8, 2, 1, 2]]

Tablero número 1.

X - - - - - - -
- - - - - - - -
- - X - - X - -
- - - - X - - -
- - - - - - X -
- - - X - - - -
- X - - - - - X
- - - - - - - -


Tablero número 2.

X - - X - - - -
- - - - - - X -
- - - - - X - X
- X - - - - - -
- - - - - - - -
- - X - X - - -
- - - - - - - -
- - - - - - - -


Tablero número 3.

X - - - - - X -
- - - - X - - -
- - - - - - - -
- - - - - - - -
- - - - - - - -
- - - - - X - X
- X - X - - - -
- - X - - - - -


Tablero número 4.

- - - - X - - -
- - - - - - - -
- - - - - - - -
- - - - - - - -
- - - - - - - -
```

```
- X - - - - - -
- - X - - X X -
X - - X - - - X
```

Tablero número 5.

```
- - - - - - - -
- - - X - - X X
- - - - - - - -
- X - - - - - -
- - - - - - - -
- - - - - - - -
X - X - X - - -
- - - - - X - -
```

Tablero número 6.

```
- - - - - - - -
X - - - - - - -
- - - - - - X -
- X - X - - - -
- - - - - - - X
- - X - - - - -
- - - - - - - -
- - - - X X - -
```

Tablero número 7.

```
- - X X - - - -
- - - - - - - -
- - - - - X - -
- - - - - - - -
X - - - - - - -
- - - - - - - X
- X - - X - - -
- - - - - - X -
```

Tablero número 8.

```
- X - - - - - -
X - - - - - - -
- - X - - - X -
- - - - X - - -
- - - X - - - -
```

```
- - - - - - - -
- - - - - - - X
- - - - - X - -
```

Tablero número 9.

```
- - - - - - - -
X - X - - - - -
- - - X - - - -
- - - - - - - -
- - - - X - - -
- - - - - - X X
- X - - - - - -
- - - - - X - -
```

Tablero número 10.

```
- - - - - - - -
X - X - - - - -
- - - - - - - X
- - - - - X X -
- - - X X - - -
- - - - - - - -
- X - - - - - -
- - - - - - - -
```

Tablero número 11.

```
- - X - - - - -
- - - - - - - X
- - - - X - - -
- X - - - - - -
- - - - - - - -
- - - - - - - -
- - - - - - X -
X - - X - X - -
```

Tablero número 12.

```
- - - - - X - -
- - - - - - - -
- - - - - - - -
X - - - - - - -
- - - - - - - X
```

```
- X - X - - - -
- - - - - - - -
- - X - X - X -
```

Tablero número 13.

```
- - - - - - X -
X - X - - - - -
- - - - - - - -
- X - X - X - -
- - - - X - - -
- - - - - - - -
- - - - - - - X
- - - - - - - -
```

Tablero número 14.

```
- - - - - - - -
- - - X X - - -
X - X - - - - -
- - - - - - X -
- - - - - - - -
- - - - - - - -
- X - - - - - -
- - - - - X - X
```

Tablero número 15.

```
- - - - - - - -
X X X - - - - -
- - - X X - - -
- - - - - - - X
- - - - - - - -
- - - - - - X -
- - - - - - - -
- - - - - X - -
```

Tablero número 16.

```
- X - - - - - -
X - - - - - - -
- - X - - - X -
- - - - - - - -
- - - - - - - X
```

```
- - - - - - - -
- - - X - - - -
- - - - X X - -
```

Tablero número 17.

```
- - - - - - X -
- X - - - - - -
- - X X - - - -
- - - - - - - -
- - - - - - - -
X - - - - - - -
- - - - X - - -
- - - - - X - X
```

Tablero número 18.

```
X - - - - - - -
- - - - - - - -
- - - - - - - -
- - - - - - - -
- X - - X - X X
- - X - - X - -
- - - X - - - -
- - - - - - - -
```

Tablero número 19.

```
X - - - - - - -
- - - - - - X -
- X - - - - - -
- - - - - - - -
- - - X X - - X
- - X - - - - -
- - - - - X - -
- - - - - - - -
```

Tablero número 20.

```
- - - - X - X X
- - - - - - - -
- - - X - - - -
- - - - - - - -
X - - - - - - -
```

```
- X - - - X - -
- - X - - - - -
- - - - - - - -
```

Tablero número 21.

```
- - - - X - - -
- - - - - - - -
- - - - - - - -
- - - - - - - -
- - - X - - X -
- X - - - - - -
X - - - - X - -
- - X - - - - X
```

Tablero número 22.

```
- - - - - - X -
- - - X X - - -
- - - - - X - -
- - - - - - - -
- - - - - - - -
X - X - - - - X
- X - - - - - -
- - - - - - - -
```

Tablero número 23.

```
- - - - - - - -
- - - - - - X -
- - - - - - - -
- - - - - - - -
- X - - X - - -
- - - X - - - X
X - - - - X - -
- - X - - - - -
```

Tablero número 24.

```
- - - - - - - -
X - - - X X X -
- - - X - - - -
- X - - - - - -
- - - - - - - -
```

```
- - - - - - - -
- - - - - - - X
- - X - - - - -
```

Tablero número 25.

```
- - - - - - - X
- - - - X X - -
- - X - - - - -
- - - - - - - -
- - - - - - - -
- - - - - - X -
- - - X - - - -
X X - - - - - -
```

Tablero número 26.

```
- - - X - - - -
- - - - - X - -
- - - - - - X -
X - - - - - - -
- - - - - - - -
- - - - - - - -
- - - - - - - X
- X X - X - - -
```

Tablero número 27.

```
- - X - X X - -
- - - X - - - -
- - - - - - - -
- - - - - - - -
- - - - - - - -
- - - - - - X X
X X - - - - - -
- - - - - - - -
```

Tablero número 28.

```
- - - - - - - -
- X - - X - - X
- - - - - X - -
- - - - - - - -
X - - - - - - -
```

```
- - - - - - - -
- - - X - - X -
- - X - - - - -
```

```
Tablero número 29.
```

```
- - - - - - - -
- - - - X - - -
- - - X - - - -
X X - - - - - -
- - - - - - - -
- - - - - X - -
- - X - - - - X
- - - - - - X -
```

```
Tablero número 30.
```

```
- - - - X - - -
- - - - - - - -
- - - - - - - -
- - - - - - - -
- - - X - - - -
- - X - - - - -
- - - - - X - X
X X - - - - X -
```

[ ]: