Z3 Assignments, week 2
Group 2

8 Queens Assignment 3
Problem description: 3
Solution explanation: 3
Source code: 5

7 Medicines in 7 test rounds assignment: 9
Problem Description: 9
Solution explanation: 9
Source code: 11

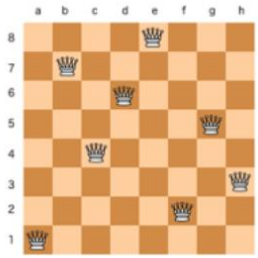
8 Queens Assignment

Problem description:

The problem states the following constraints:

Wk 2 Assignments (I)

Let Z3 solve the famous Eight Queens Puzzle:
https://en.wikipedia.org/wiki/Eight_queens_puzzle



Solution explanation:

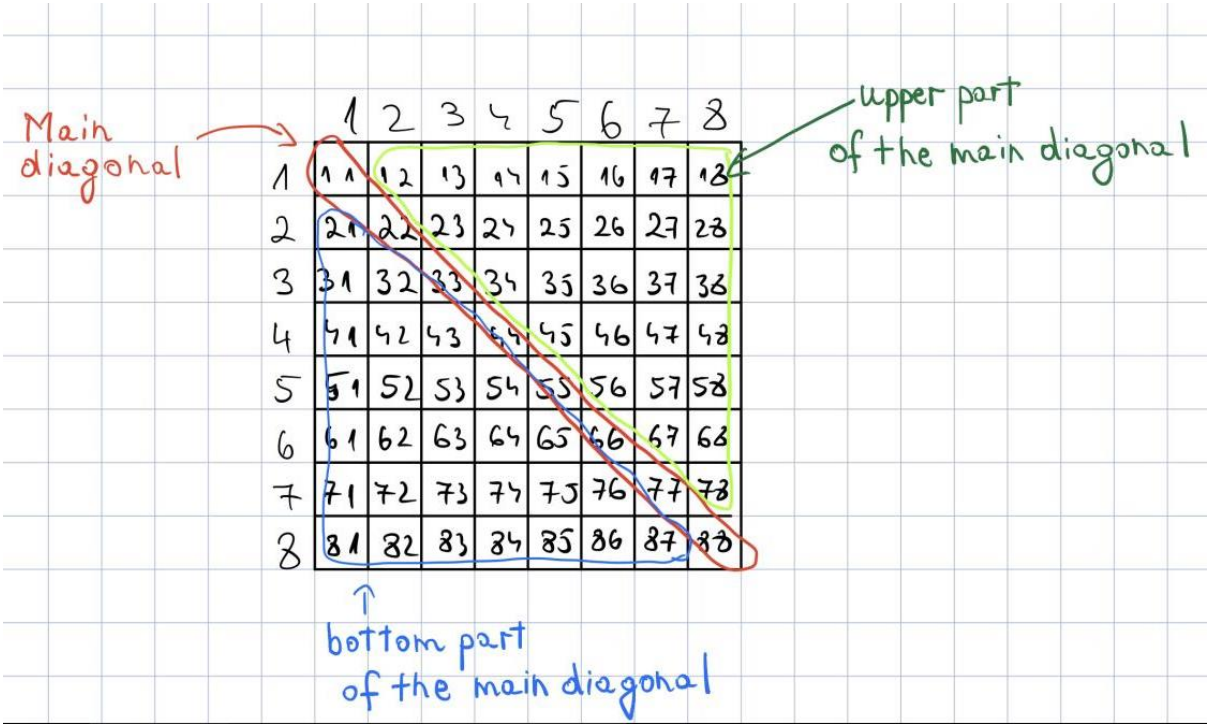
Solution

- To solve this assignment, we came up with an idea to create a grid which represents a 8x8 chess board.
- When a queen is placed, we need to check all the possible combinations for the rows, columns, and diagonals.

Summary

- To create the board, we declare a function that takes in two integers as parameters and returns a boolean value. This is done so that we can map exactly where a queen can be, by mapping the first integer as the row and the second for the column.

- We then declare a function that covers the constraint of having only 1 queen per row (OnePerRow) which takes as a parameter the row and we iterate through it. (Same logic applies to OnePerColumn)
- A function to initialize all the rows (AllRows/AllColumns) is created.
- Since a queen can move diagonally, we need to check all the possible combinations where we could put the queen. (A picture to explain how we approach the diagonal can be found below). Therefore, we first check the main diagonal and then we go both above and under it. We do the same thing for the second diagonal as well.



Source code:

```
(declare-fun grid(Int Int) Bool)
(declare-fun Row (Int) Int)
(declare-fun Column (Int) Int)

(define-fun OnePerRow ((r Int)) Bool
    (and (= (Row r) (+
            (ite (grid r 1) 1 0)
            (ite (grid r 2) 1 0)
            (ite (grid r 3) 1 0)
            (ite (grid r 4) 1 0)
            (ite (grid r 5) 1 0)
            (ite (grid r 6) 1 0)
            (ite (grid r 7) 1 0)
            (ite (grid r 8) 1 0)

        )
    )
    (= (Row r) 1))
)

(define-fun AllRows() Bool
    (and
        (OnePerRow 1)
        (OnePerRow 2)
        (OnePerRow 3)
        (OnePerRow 4)
        (OnePerRow 5)
        (OnePerRow 6)
        (OnePerRow 7)
        (OnePerRow 8)
    )
)

(define-fun OnePerCol ((c Int)) Bool
    (and (= (Column c) (+
            (ite (grid 1 c) 1 0)
            (ite (grid 2 c) 1 0)
            (ite (grid 3 c) 1 0)
            (ite (grid 4 c) 1 0)
            (ite (grid 5 c) 1 0)
            (ite (grid 6 c) 1 0)
            (ite (grid 7 c) 1 0)
            (ite (grid 8 c) 1 0)
```

```
        )
    )
    (= (Column c) 1))
)

(define-fun AllCols() Bool
    (and
        (OnePerCol 1)
        (OnePerCol 2)
        (OnePerCol 3)
        (OnePerCol 4)
        (OnePerCol 5)
        (OnePerCol 6)
        (OnePerCol 7)
        (OnePerCol 8)
    )
)

(define-fun Diagonal ((v1 Bool) (v2 Bool) (v3 Bool) (v4 Bool) (v5 Bool)
(v6 Bool) (v7 Bool) (v8 Bool)) Bool
      (not (or
                  (and v1 v2) (and v1 v3) (and v1 v4) (and v1 v5) (and v1
v6) (and v1 v7) (and v1 v8)
                  (and v2 v3) (and v2 v4) (and v2 v5) (and v2 v6) (and v2
v7) (and v2 v8)
                (and v3 v4) (and v3 v5) (and v3 v6) (and v3 v7) (and v3
v8)
                (and v4 v5) (and v4 v6) (and v4 v7) (and v4 v8)
                  (and v5 v6) (and v5 v7) (and v5 v8)
                (and v6 v7) (and v6 v8)
                (and v7 v8))))

(define-fun Diagonal1 ((v1 Bool) (v2 Bool) (v3 Bool) (v4 Bool) (v5 Bool)
(v6 Bool) (v7 Bool)) Bool
      (not (or
                  (and v1 v2) (and v1 v3) (and v1 v4) (and v1 v5) (and v1
v6) (and v1 v7)
                  (and v2 v3) (and v2 v4) (and v2 v5) (and v2 v6) (and v2
v7)
                (and v3 v4) (and v3 v5) (and v3 v6) (and v3 v7)
                (and v4 v5) (and v4 v6) (and v4 v7)
                  (and v5 v6) (and v5 v7)
                (and v6 v7))))

(define-fun Diagonal2 ((v1 Bool) (v2 Bool) (v3 Bool) (v4 Bool) (v5 Bool)
(v6 Bool)) Bool
```

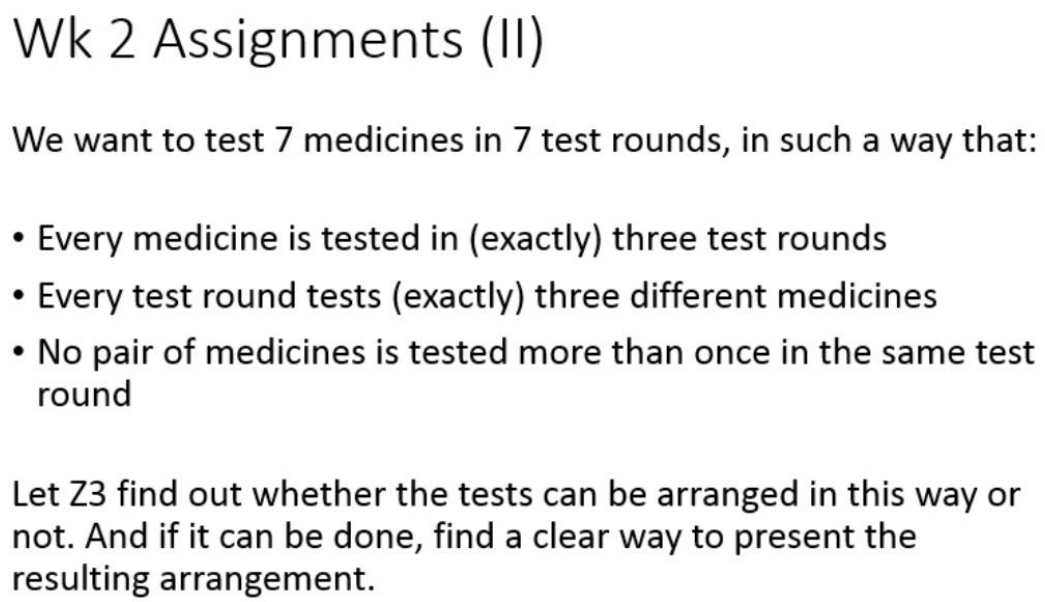
```
      (not (or
                  (and v1 v2) (and v1 v3) (and v1 v4) (and v1 v5) (and v1
v6)
                  (and v2 v3) (and v2 v4) (and v2 v5) (and v2 v6)
                (and v3 v4) (and v3 v5) (and v3 v6)
                (and v4 v5) (and v4 v6)
                  (and v5 v6) )))

(define-fun Diagonal3 ((v1 Bool) (v2 Bool) (v3 Bool) (v4 Bool) (v5
Bool)) Bool
      (not (or
                  (and v1 v2) (and v1 v3) (and v1 v4) (and v1 v5)
                  (and v2 v3) (and v2 v4) (and v2 v5)
                (and v3 v4) (and v3 v5)
                (and v4 v5))))

(define-fun Diagonal4 ((v1 Bool) (v2 Bool) (v3 Bool) (v4 Bool)) Bool
      (not (or
                  (and v1 v2) (and v1 v3) (and v1 v4)
                  (and v2 v3) (and v2 v4)
                (and v3 v4))))

(define-fun Diagonal5 ((v1 Bool) (v2 Bool) (v3 Bool)) Bool
      (not (or
                  (and v1 v2) (and v1 v3)
                  (and v2 v3))))

(define-fun Diagonal6 ((v1 Bool) (v2 Bool)) Bool
      (not (or
                  (and v1 v2) )))


(assert (and
        AllCols
        AllRows

))

(assert (and

        (Diagonal   (grid 1 1) (grid 2 2) (grid 3 3) (grid 4 4) (grid 5
5) (grid 6 6) (grid 7 7) (grid 8 8))
            (Diagonal1  (grid 2 1) (grid 3 2) (grid 4 3) (grid 5 4) (grid
6 5) (grid 7 6) (grid 8 7))
            (Diagonal2  (grid 3 1) (grid 4 2) (grid 5 3) (grid 6 4) (grid
7 5) (grid 8 6))
```

```
            (Diagonal3  (grid 4 1) (grid 5 2) (grid 6 3) (grid 7 4) (grid
8 5))
            (Diagonal4  (grid 5 1) (grid 6 2) (grid 7 3) (grid 8 4))
            (Diagonal5  (grid 6 1) (grid 7 2) (grid 8 3))
            (Diagonal6  (grid 7 1) (grid 8 2))

        (Diagonal1  (grid 1 2) (grid 2 3) (grid 3 4) (grid 4 5) (grid 5
6) (grid 6 7) (grid 7 8))
            (Diagonal2  (grid 1 3) (grid 2 4) (grid 3 5) (grid 4 6) (grid
5 7) (grid 6 8))
            (Diagonal3  (grid 1 4) (grid 2 5) (grid 3 6) (grid 4 7) (grid
5 8))
            (Diagonal4  (grid 1 5) (grid 2 6) (grid 3 7) (grid 4 8))
            (Diagonal5  (grid 1 6) (grid 2 7) (grid 3 8))
            (Diagonal6  (grid 1 7) (grid 2 8))


        (Diagonal   (grid 1 8) (grid 2 7) (grid 3 6) (grid 4 5) (grid 5
4) (grid 6 3) (grid 7 2) (grid 8 1))
            (Diagonal1  (grid 1 7) (grid 2 6) (grid 3 5) (grid 4 4) (grid
5 3) (grid 6 2) (grid 7 1))
            (Diagonal2  (grid 1 6) (grid 2 5) (grid 3 4) (grid 4 3) (grid
5 2) (grid 6 1))
            (Diagonal3  (grid 1 5) (grid 2 4) (grid 3 3) (grid 4 2) (grid
5 1))
            (Diagonal4  (grid 1 4) (grid 2 3) (grid 3 2) (grid 4 1))
            (Diagonal5  (grid 1 3) (grid 2 2) (grid 3 1))
            (Diagonal6  (grid 1 2) (grid 2 1))


        (Diagonal1  (grid 2 8) (grid 3 7) (grid 4 6) (grid 5 5) (grid 6
4) (grid 7 3) (grid 8 2))
            (Diagonal2  (grid 3 8) (grid 4 7) (grid 5 6) (grid 6 5) (grid
7 4) (grid 8 3))
            (Diagonal3  (grid 4 8) (grid 5 7) (grid 6 6) (grid 7 5) (grid
8 4))
            (Diagonal4  (grid 5 8) (grid 6 7) (grid 7 6) (grid 8 5))
            (Diagonal5  (grid 6 8) (grid 7 7) (grid 8 6))
            (Diagonal6  (grid 7 8) (grid 8 7))

))

(check-sat)
(get-model)
```

---

7 Medicines in 7 test rounds assignment:

Problem Description:



Solution explanation:

Solution

- To solve this assignment, we came up with an idea to map each of the cells where a medicine test could occur.
- This consists of a table containing 7 rows and 7 columns.
- The rows are the medicines, and the columns are the test rounds.

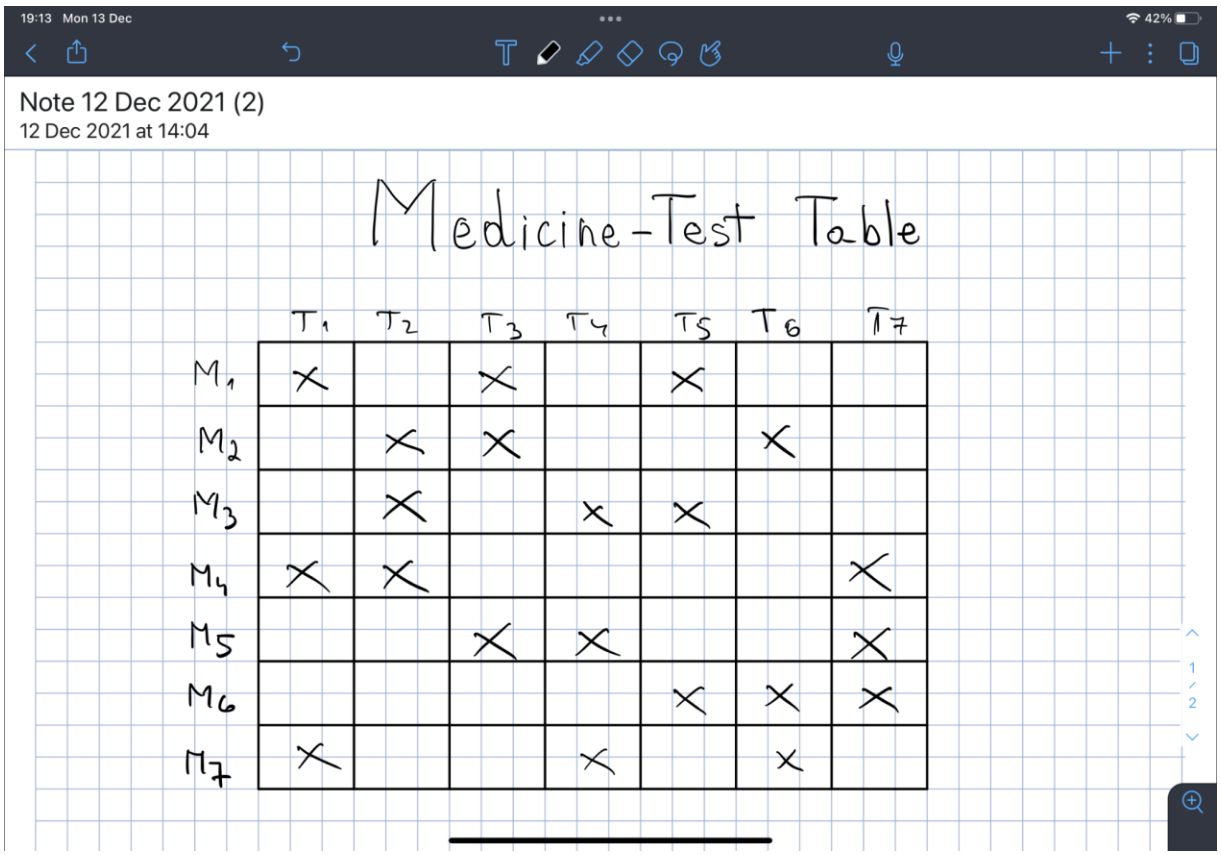
Summary

- To create the table, we declare a function that takes in two integers as parameters and returns a boolean value. This is done so that we could map exactly where a medicine test could occur by mapping the first integer as the row and the second for the column. (`ThreeRoundsPerMedicine`)

- We then declare a function that covers the first constraint of having only 3 cells (medicines) to exist per round. We do this by initializing “arrays” for each row and column, where ***r*** is the variable that we replace later in the function ***AllRounds().*** (Same logic applies to ***AllMeds()*** function)

- To take care of the second constraint of only having exactly 3 different medicines and their completion in 3 rounds, we have the same logic applied, but this time having a variable ***r*** that we later replace in the next method call.

- To make sure that no pair of the same medicines can be tested again, we have the final declaration of the function ***NoMedPairsPerRound(),*** where we compare each column (test round) with each row (medicine).



Source code:

```
;M is the "Matrix" or positions of the medicines in the table
(declare-fun M (Int Int) Bool)
(declare-fun Pairs (Int Int) Int)
;array of rounds
(declare-fun Rounds (Int) Int)

;array of meds
(declare-fun Meds (Int) Int)

;We check if the cells of the array are 3
;Makes sure that only 3 cells exist per round
(define-fun ThreeRoundsPerMedicine ((r Int)) Bool
    (and  (= (Rounds r) (+
            (ite (M r 1) 1 0)
            (ite (M r 2) 1 0)
            (ite (M r 3) 1 0)
            (ite (M r 4) 1 0)
            (ite (M r 5) 1 0)
            (ite (M r 6) 1 0)
            (ite (M r 7) 1 0)

        )
    )
    (= (Rounds r) 3))
)

;Sets the position for all instances
(define-fun AllRounds() Bool
    (and
        (ThreeRoundsPerMedicine 1)
        (ThreeRoundsPerMedicine 2)
        (ThreeRoundsPerMedicine 3)
        (ThreeRoundsPerMedicine 4)
        (ThreeRoundsPerMedicine 5)
        (ThreeRoundsPerMedicine 6)
        (ThreeRoundsPerMedicine 7)
    )
)

;Makes sure that the medicine is completed in exactly 3 rounds
;We check if the cells of the array are 3
```

```
(define-fun ThreeMedicinesPerRound ((r Int)) Bool
    (and (= (Meds r) (+
            (ite (M 1 r) 1 0)
            (ite (M 2 r) 1 0)
            (ite (M 3 r) 1 0)
            (ite (M 4 r) 1 0)
            (ite (M 5 r) 1 0)
            (ite (M 6 r) 1 0)
            (ite (M 7 r) 1 0)

        )
    )
    (= (Meds r) 3))
)

;Sets the position for all instances
(define-fun AllMeds() Bool
    (and
        (ThreeMedicinesPerRound 1)
        (ThreeMedicinesPerRound 2)
        (ThreeMedicinesPerRound 3)
        (ThreeMedicinesPerRound 4)
        (ThreeMedicinesPerRound 5)
        (ThreeMedicinesPerRound 6)
        (ThreeMedicinesPerRound 7)
    )
)

; Based on the row and column, we make sure that no pair
; of medicines can occur twice in the table
(define-fun NoMedPairsPerRound ((r Int) (c Int)) Bool
    (and (= (Pairs r c) (+
            (ite (and (M 1 r) (M 1 c)) 1 0)
            (ite (and (M 2 r) (M 2 c)) 1 0)
            (ite (and (M 3 r) (M 3 c)) 1 0)
            (ite (and (M 4 r) (M 4 c)) 1 0)
            (ite (and (M 5 r) (M 5 c)) 1 0)
            (ite (and (M 6 r) (M 6 c)) 1 0)
            (ite (and (M 7 r) (M 7 c)) 1 0)
        )
    )
    (not (> (Pairs r c) 1)))
)

;Sets the position for all instances
(define-fun NoPairs() Bool
```

```
    (and
        (NoMedPairsPerRound 1 2)
        (NoMedPairsPerRound 1 3)
        (NoMedPairsPerRound 1 4)
        (NoMedPairsPerRound 1 5)
        (NoMedPairsPerRound 1 6)
        (NoMedPairsPerRound 1 7)

        (NoMedPairsPerRound 2 3)
        (NoMedPairsPerRound 2 4)
        (NoMedPairsPerRound 2 5)
        (NoMedPairsPerRound 2 6)
        (NoMedPairsPerRound 2 7)

        (NoMedPairsPerRound 3 4)
        (NoMedPairsPerRound 3 5)
        (NoMedPairsPerRound 3 6)
        (NoMedPairsPerRound 3 7)

        (NoMedPairsPerRound 4 5)
        (NoMedPairsPerRound 4 6)
        (NoMedPairsPerRound 4 7)

        (NoMedPairsPerRound 5 6)
        (NoMedPairsPerRound 5 7)

        (NoMedPairsPerRound 6 7)
    )
)

;Sets the position for all instances
;Asserts the functions (calls the functions)
(assert (and
    AllRounds
    AllMeds
    NoPairs


))

(check-sat)
(get-model)
```