Lab 4

1. $X_{ij} = variable\ retaining\ value\ from\ cell(i,j)$

$$D_{ij} = domain\ of\ variable\ X_{ij} = \begin{cases} \{1,2,..9\}, & if\ the\ cell\ is\ an\ empty\ white\ cell \\ \{2,4,6,8\}, & if\ the\ cell\ is\ an\ empty\ grey\ cell \\ val, & if\ the\ cell\ already\ contains\ a\ value \end{cases}$$

Obs: Instead of considering the domain {2,4,6,8} for empty grey cells, one can work with a domain of {1,2,…,9} and create the constraints that those cells should contain an even number

$R = \{ X_{ij} \mathrel{!}= X_{ik}\ for\ k \mathrel{!}= j,\ \ 1 \le i,j,k \le 9$

$X_{ij} \mathrel{!}= X_{kj}\ for\ k \mathrel{!}= i,\ \ 1 \le i,j,k \le 9$

$X_{i1j1} \mathrel{!}= X_{i2j2}\ \forall\ (i1,j1) \mathrel{!}= (i2,j2)\ \&\&\ (i1,j1),(i2,j2) \in\ square$

$[\ X_{ij}\ \%\ 2 == 0, for\ (i,j)\ empty\ grey\ cell\ ]$

$\}$

2. **Backtracking for CSP**:

```
def BKT(assignment):
    if (isComplete(assignment)):
        return assignment

    var = next_unassigned_variable(assignment)
    for value in Domain(var):
        if consistent(assignment, var, value):
           new_assignment = assignment ∪ {var = value}
           res = BKT(new_assignment)
           if res is not None:
              return res
    return None
```

Forward checking:

```
Def BKT_with_FC(assignment, domains):
    if (isComplete(assignment)):
        return assignment

    var = next_unassigned_variable(assignment)
    for value in Domain(var):
        if consistent(assignment, var, value):
           new_assignment = assignment ∪ {var = value}
           new_domains = update_domains_FC(domains, var, value)
           if (no new_domain of an unassigned variable is empty):
                 res = BKT_with_FC(new_assignment, new_domains)
                 if res is not None:
                    return res
    return None
```

When attributing a value v to a variable A, we eliminate the values w from the domains of the other variables B if assigning $A = v \text{ and } B = w$ breaks any of the restrictions.

If at one point a domain of an unassigned variable gets empty, this means there will be no solution in the future! Therefore, there is no need to go further in the Backtracking.

3. **Minimum remaining values (MRV)**
   MRV heuristic = choose the variable with the smallest number of values remaining in its domain

```
Def BKT_with_FC_MRV(assignment, domains):
    if (isComplete(assignment)):
        return assignment

    var = next_unassigned_variable_MRV(assignment, domains)
    for value in Domain(var):
        if consistent(assignment, var, value):
           new_assignment = assignment ∪ {var = value}
           new_domains = update_domains_FC(domains, var, value)
           if (no new_domain of an unassigned variable is empty):
                 res = BKT_with_FC_MRV(new_assignment, new_domains)
                 if res is not None:
                    return res
    return None
```

4. Bonus: **Arc Consistency**

$X \rightarrow Y$ consistent iff $\forall\, x \in Domain(X)\ \exists\, y \in Domain(Y)$ *such that no constraints are broken*

Put in Q all pairs of variables (X,Y), such that X and Y are linked via at least a constraint.

```
While (!Q.empty()):
   (X,Y) = Q.pop()
   ok = True
   for each value x ∈ Domain(X):
       if ∄ y ∈ Domain(Y) such that X=x, Y=y doesn't break constraints:
          ok = False
          delete x from Domain(X)
   if(ok == 0):
      For Z in neighbors(X):
          Q.push(Z, X)
```

Final Result: Updated domains of variables.
Arc Consistency can be used as a preprocessing step before Backtracking, or inside Backtracking.
What is the complexity of arc consistency? Why? (hint: think about the maximum number of times you can insert a pair in the queue)