Backtracking Algorithms

Dr. Jonathan Phillips

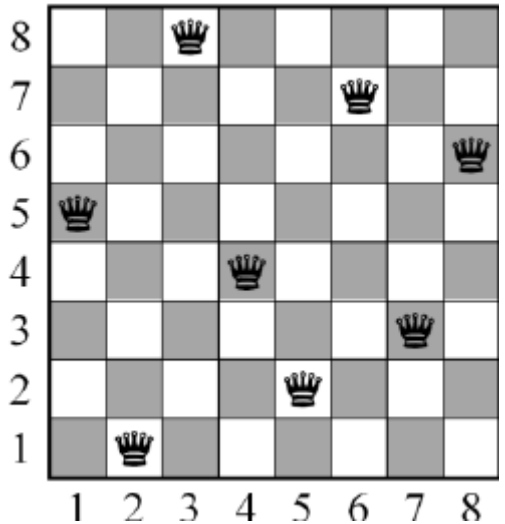
Utah State University

Backtracking Algorithm

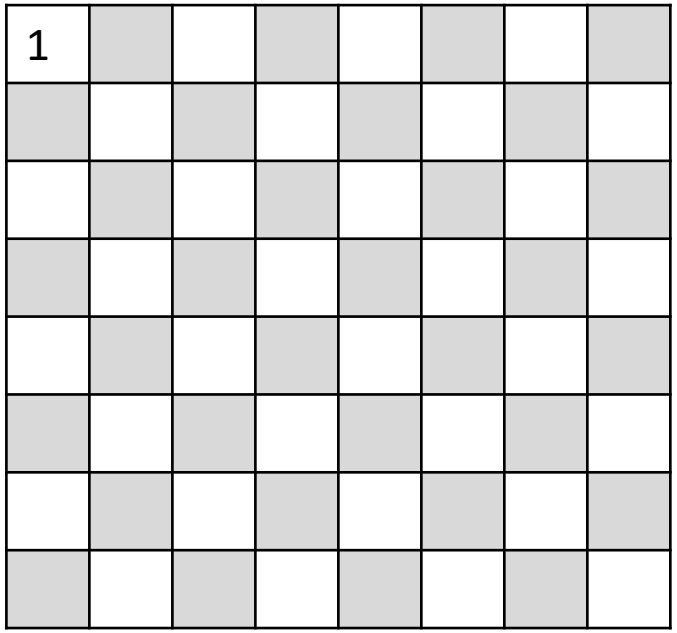
- Steps
  - Incrementally build a solution candidate
  - Abandon partial candidate (AKA backtrack) as soon as deemed invalid
- More efficient than “brute force” method
- Gives us an ordered method to build a solution

Example: 8 Queens

- Problem: Place 8 queens on a standard chessboard, such that no queen is attacking any other
- Brute Force solution
  - Try all combinations of positioning 8 queens until one is discovered
  - There are C(64, 8) unique positions: 4,426,165,396



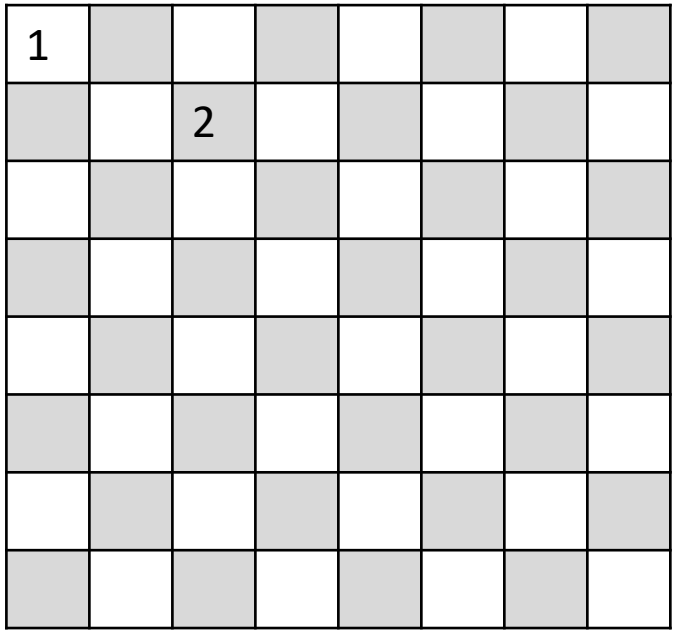
Example: 8 Queens



- Backtracking approach:

```
Begin
        place queen 1 at grid[1,1]
        n = 2, (x, y) = (1, 1)
        Loop until n = 9
                place queen n at grid[x, y]
                If no violations
                Then
                        n = n + 1, (x, y) + 1
                Else
                        (x, y) + 1
                        If no next square
                                n = n -1, (x, y) = (position of queen n) + 1
                        Endif
                End If
        EndLoop
End
```

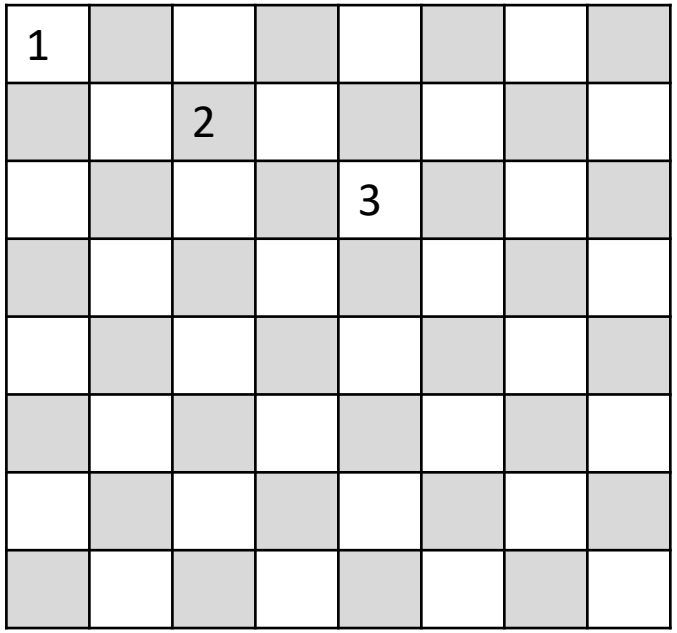
Example: 8 Queens



- Backtracking approach:

```
Begin
        place queen 1 at grid[1,1]
        n = 2, (x, y) = (1, 1)
        Loop until n = 9
                place queen n at grid[x, y]
                If no violations
                Then
                        n = n + 1, (x, y) + 1
                Else
                        (x, y) + 1
                        If no next square
                                n = n -1, (x, y) = (position of queen n) + 1
                        Endif
                End If
        EndLoop
End
```

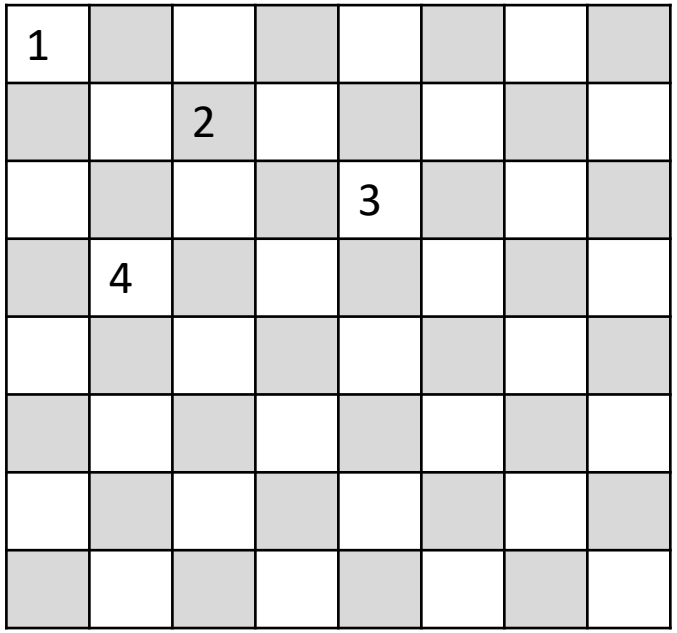
Example: 8 Queens



- Backtracking approach:

```
Begin
        place queen 1 at grid[1,1]
        n = 2, (x, y) = (1, 1)
        Loop until n = 9
                place queen n at grid[x, y]
                If no violations
                Then
                        n = n + 1, (x, y) + 1
                Else
                        (x, y) + 1
                        If no next square
                                n = n -1, (x, y) = (position of queen n) + 1
                        Endif
                End If
        EndLoop
End
```

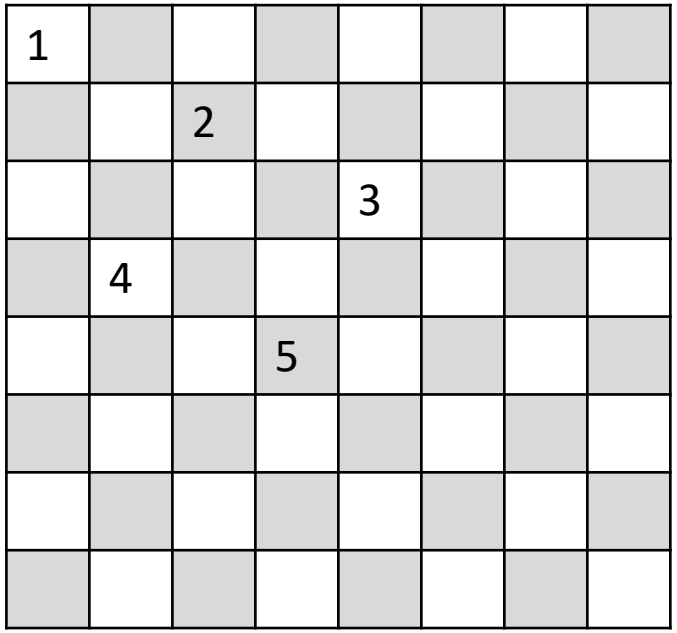
Example: 8 Queens



- Backtracking approach:

```
Begin
        place queen 1 at grid[1,1]
        n = 2, (x, y) = (1, 1)
        Loop until n = 9
                place queen n at grid[x, y]
                If no violations
                Then
                        n = n + 1, (x, y) + 1
                Else
                        (x, y) + 1
                        If no next square
                                n = n -1, (x, y) = (position of queen n) + 1
                        Endif
                End If
        EndLoop
End
```

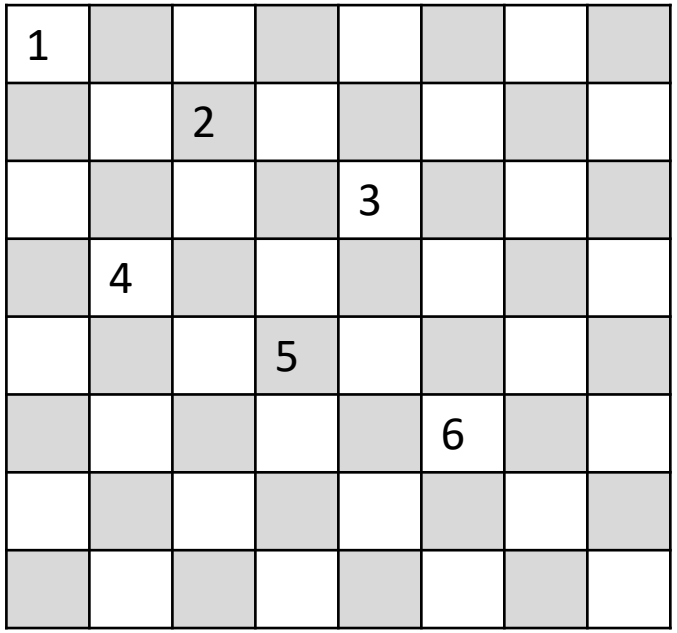
Example: 8 Queens



- Backtracking approach:

```
Begin
        place queen 1 at grid[1,1]
        n = 2, (x, y) = (1, 1)
        Loop until n = 9
                place queen n at grid[x, y]
                If no violations
                Then
                        n = n + 1, (x, y) + 1
                Else
                        (x, y) + 1
                        If no next square
                                n = n -1, (x, y) = (position of queen n) + 1
                        Endif
                End If
        EndLoop
End
```

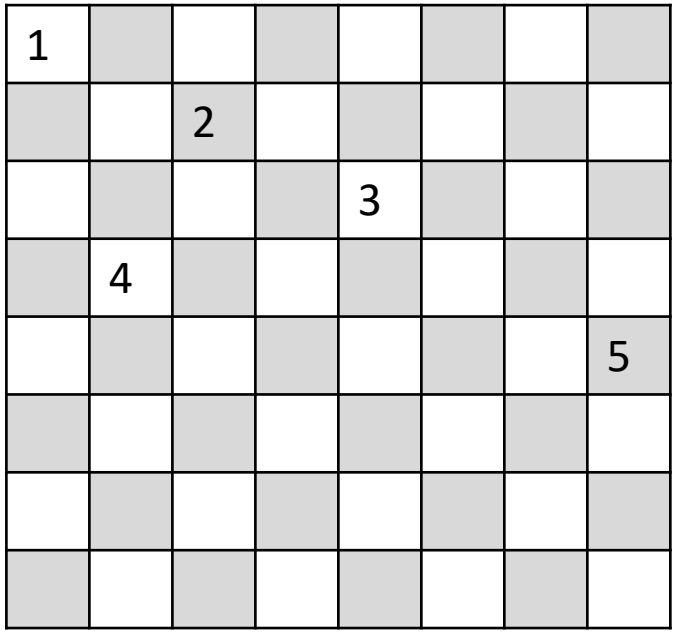
Example: 8 Queens



- Backtracking approach:

```
Begin
        place queen 1 at grid[1,1]
        n = 2, (x, y) = (1, 1)
        Loop until n = 9
                place queen n at grid[x, y]
                If no violations
                Then
                        n = n + 1, (x, y) + 1
                Else
                        (x, y) + 1
                        If no next square
                                n = n -1, (x, y) = (position of queen n) + 1
                        Endif
                End If
        EndLoop
End
```

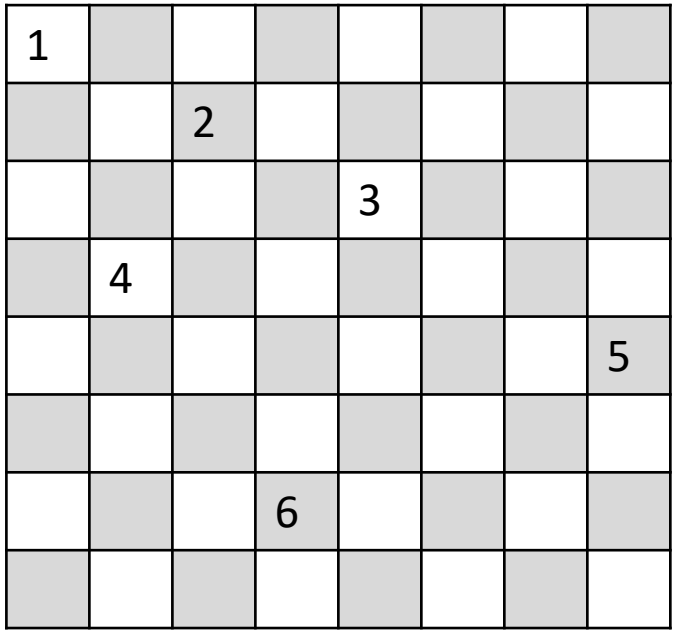
Example: 8 Queens



- Backtracking approach:

```
Begin
        place queen 1 at grid[1,1]
        n = 2, (x, y) = (1, 1)
        Loop until n = 9
                place queen n at grid[x, y]
                If no violations
                Then
                        n = n + 1, (x, y) + 1
                Else
                        (x, y) + 1
                        If no next square
                                n = n -1, (x, y) = (position of queen n) + 1
                        Endif
                End If
        EndLoop
End
```

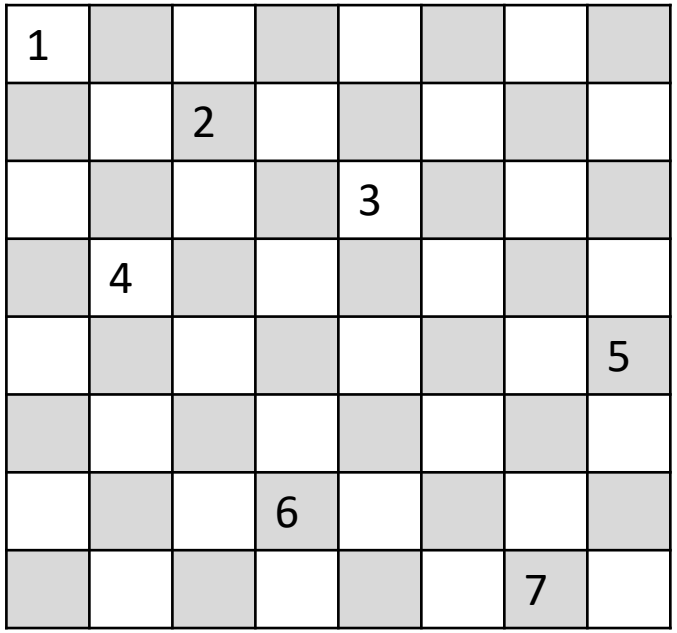
Example: 8 Queens



- Backtracking approach:

```
Begin
        place queen 1 at grid[1,1]
        n = 2, (x, y) = (1, 1)
        Loop until n = 9
                place queen n at grid[x, y]
                If no violations
                Then
                        n = n + 1, (x, y) + 1
                Else
                        (x, y) + 1
                        If no next square
                                n = n -1, (x, y) = (position of queen n) + 1
                        Endif
                End If
        EndLoop
End
```

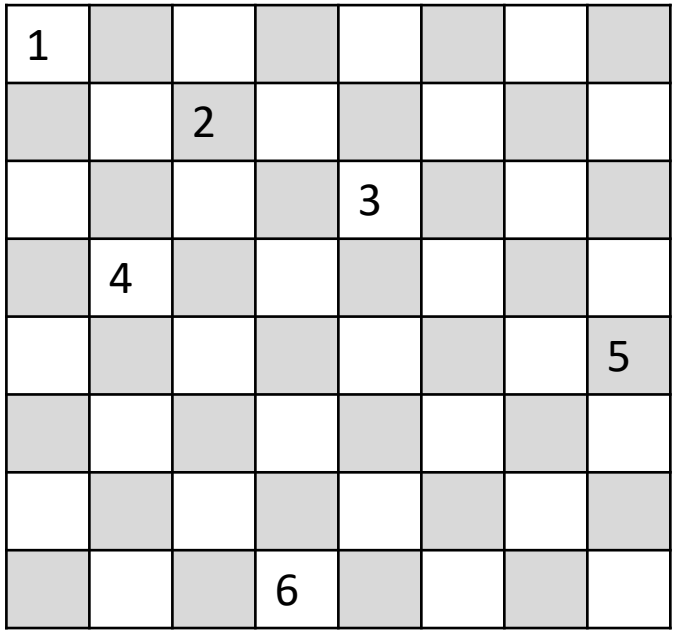
Example: 8 Queens



- Backtracking approach:

```
Begin
    place queen 1 at grid[1,1]
    n = 2, (x, y) = (1, 1)
    Loop until n = 9
        place queen n at grid[x, y]
        If no violations
        Then
            n = n + 1, (x, y) + 1
        Else
            (x, y) + 1
            If no next square
                n = n -1, (x, y) = (position of queen n) + 1
            Endif
        End If
    EndLoop
End
```

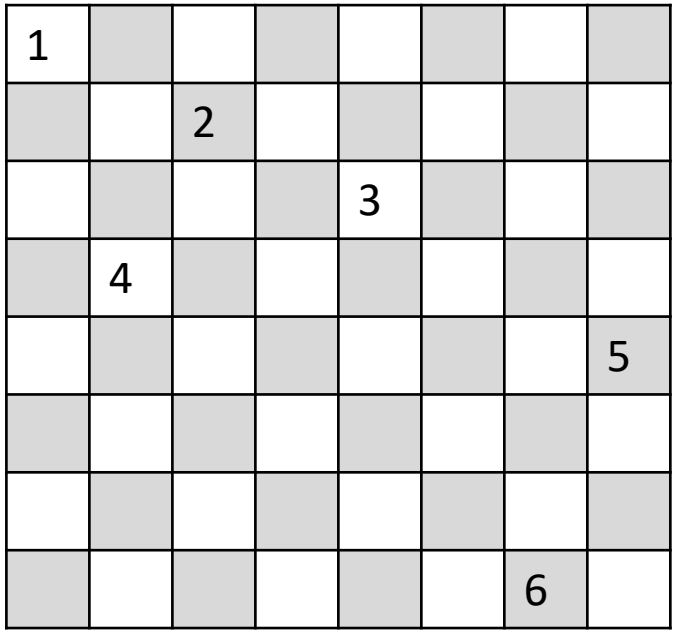
Example: 8 Queens



- Backtracking approach:

```
Begin
        place queen 1 at grid[1,1]
        n = 2, (x, y) = (1, 1)
        Loop until n = 9
                place queen n at grid[x, y]
                If no violations
                Then
                        n = n + 1, (x, y) + 1
                Else
                        (x, y) + 1
                        If no next square
                                n = n -1, (x, y) = (position of queen n) + 1
                        Endif
                End If
        EndLoop
End
```

Example: 8 Queens



- Backtracking approach:

```
Begin
        place queen 1 at grid[1,1]
        n = 2, (x, y) = (1, 1)
        Loop until n = 9
                place queen n at grid[x, y]
                If no violations
                Then
                        n = n + 1, (x, y) + 1
                Else
                        (x, y) + 1
                        If no next square
                                n = n -1, (x, y) = (position of queen n) + 1
                        Endif
                End If
        EndLoop
End
```

Can we apply Backtracking to Sudoku?

- Yes!!!

Algorithm

```
initialize grid to all zero
n=1, (x,y)=firstSquareInPlay
Loop
   place n at grid[x,y]
   If no violations
   Then
      n=1, (x,y)=nextSquareInPlay
      If off grid
      Then
         SUCCESS
      EndIf
   Else
      n=n+1
      If n is 10
      Then
         place 0 at grid[x,y]
         (x,y)=prevSquareInPlay
         If off grid
         Then
            FAIL
         Else
            n=grid[x,y]+1
         EndIf
      EndIf
   EndIf
EndLoop
```

Violations:

- n already exists in row
- n already exists in column
- n already exists in 3x3 box

Algorithm

```
initialize grid to all zero
n=1, (x,y)=firstSquareInPlay
Loop
   place n at grid[x,y]
   If no violations
   Then
      n=1, (x,y)=nextSquareInPlay
      If off grid
      Then
         SUCCESS
      EndIf
   Else
      n=n+1
      If n is 10
      Then
         place 0 at grid[x,y]
         (x,y)=prevSquareInPlay
         If off grid
         Then
            FAIL
         Else
            n=grid[x,y]+1
         EndIf
      EndIf
   EndIf
EndLoop
```

5	3			7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

Algorithm

```
initialize grid to all zero
n=1, (x,y)=firstSquareInPlay
Loop
   place n at grid[x,y]
   If no violations
   Then
      n=1, (x,y)=nextSquareInPlay
      If off grid
      Then
         SUCCESS
      EndIf
   Else
      n=n+1
      If n is 10
      Then
         place 0 at grid[x,y]
         (x,y)=prevSquareInPlay
         If off grid
         Then
            FAIL
         Else
            n=grid[x,y]+1
         EndIf
      EndIf
   EndIf
EndLoop
```

5	3	1		7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

Algorithm

```
initialize grid to all zero
n=1, (x,y)=firstSquareInPlay
Loop
   place n at grid[x,y]
   If no violations
   Then
      n=1, (x,y)=nextSquareInPlay
      If off grid
      Then
         SUCCESS
      EndIf
   Else
      n=n+1
      If n is 10
      Then
         place 0 at grid[x,y]
         (x,y)=prevSquareInPlay
         If off grid
         Then
            FAIL
         Else
            n=grid[x,y]+1
         EndIf
      EndIf
   EndIf
EndLoop
```

5	3	1	2	7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

Algorithm

```
initialize grid to all zero
n=1, (x,y)=firstSquareInPlay
Loop
   place n at grid[x,y]
   If no violations
   Then
      n=1, (x,y)=nextSquareInPlay
      If off grid
      Then
         SUCCESS
      EndIf
   Else
      n=n+1
      If n is 10
      Then
         place 0 at grid[x,y]
         (x,y)=prevSquareInPlay
         If off grid
         Then
            FAIL
         Else
            n=grid[x,y]+1
         EndIf
      EndIf
   EndIf
EndLoop
```

5	3	1	2	7	4			
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

Algorithm

```
initialize grid to all zero
n=1, (x,y)=firstSquareInPlay
Loop
   place n at grid[x,y]
   If no violations
   Then
      n=1, (x,y)=nextSquareInPlay
      If off grid
      Then
         SUCCESS
      EndIf
   Else
      n=n+1
      If n is 10
      Then
         place 0 at grid[x,y]
         (x,y)=prevSquareInPlay
         If off grid
         Then
            FAIL
         Else
            n=grid[x,y]+1
         EndIf
      EndIf
   EndIf
EndLoop
```

5	3	1	2	7	4	8		
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

Algorithm

```
initialize grid to all zero
n=1, (x,y)=firstSquareInPlay
Loop
   place n at grid[x,y]
   If no violations
   Then
      n=1, (x,y)=nextSquareInPlay
      If off grid
      Then
         SUCCESS
      EndIf
   Else
      n=n+1
      If n is 10
      Then
         place 0 at grid[x,y]
         (x,y)=prevSquareInPlay
         If off grid
         Then
            FAIL
         Else
            n=grid[x,y]+1
         EndIf
      EndIf
   EndIf
EndLoop
```

5	3	1	2	7	4	8	9	
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

Algorithm

```
initialize grid to all zero
n=1, (x,y)=firstSquareInPlay
Loop
   place n at grid[x,y]
   If no violations
   Then
      n=1, (x,y)=nextSquareInPlay
      If off grid
      Then
         SUCCESS
      EndIf
   Else
      n=n+1
      If n is 10
      Then
         place 0 at grid[x,y]
         (x,y)=prevSquareInPlay
         If off grid
         Then
            FAIL
         Else
            n=grid[x,y]+1
         EndIf
      EndIf
   EndIf
EndLoop
```

5	3	1	2	7	4	9		
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

Algorithm

```
initialize grid to all zero
n=1, (x,y)=firstSquareInPlay
Loop
   place n at grid[x,y]
   If no violations
   Then
      n=1, (x,y)=nextSquareInPlay
      If off grid
      Then
         SUCCESS
      EndIf
   Else
      n=n+1
      If n is 10
      Then
         place 0 at grid[x,y]
         (x,y)=prevSquareInPlay
         If off grid
         Then
            FAIL
         Else
            n=grid[x,y]+1
         EndIf
      EndIf
   EndIf
EndLoop
```

5	3	1	2	7	6			
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

Algorithm

```
initialize grid to all zero
n=1, (x,y)=firstSquareInPlay
Loop
   place n at grid[x,y]
   If no violations
   Then
      n=1, (x,y)=nextSquareInPlay
      If off grid
      Then
         SUCCESS
      EndIf
   Else
      n=n+1
      If n is 10
      Then
         place 0 at grid[x,y]
         (x,y)=prevSquareInPlay
         If off grid
         Then
            FAIL
         Else
            n=grid[x,y]+1
         EndIf
      EndIf
   EndIf
EndLoop
```

5	3	1	2	7	6	4		
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

Algorithm

```
initialize grid to all zero
n=1, (x,y)=firstSquareInPlay
Loop
   place n at grid[x,y]
   If no violations
   Then
      n=1, (x,y)=nextSquareInPlay
      If off grid
      Then
         SUCCESS
      EndIf
   Else
      n=n+1
      If n is 10
      Then
         place 0 at grid[x,y]
         (x,y)=prevSquareInPlay
         If off grid
         Then
            FAIL
         Else
            n=grid[x,y]+1
         EndIf
      EndIf
   EndIf
EndLoop
```

5	3	1	2	7	6	4	9	
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

Algorithm

```
initialize grid to all zero
n=1, (x,y)=firstSquareInPlay
Loop
   place n at grid[x,y]
   If no violations
   Then
      n=1, (x,y)=nextSquareInPlay
      If off grid
      Then
         SUCCESS
      EndIf
   Else
      n=n+1
      If n is 10
      Then
         place 0 at grid[x,y]
         (x,y)=prevSquareInPlay
         If off grid
         Then
            FAIL
         Else
            n=grid[x,y]+1
         EndIf
      EndIf
   EndIf
EndLoop
```

5	3	1	2	7	6	4	9	8
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

Algorithm

```
initialize grid to all zero
n=1, (x,y)=firstSquareInPlay
Loop
   place n at grid[x,y]
   If no violations
   Then
      n=1, (x,y)=nextSquareInPlay
      If off grid
      Then
         SUCCESS
      EndIf
   Else
      n=n+1
      If n is 10
      Then
         place 0 at grid[x,y]
         (x,y)=prevSquareInPlay
         If off grid
         Then
            FAIL
         Else
            n=grid[x,y]+1
         EndIf
      EndIf
   EndIf
EndLoop
```

5	3	1	2	7	6	4	9	8
6	2		1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

Algorithm

```
initialize grid to all zero
n=1, (x,y)=firstSquareInPlay
Loop
   place n at grid[x,y]
   If no violations
   Then
      n=1, (x,y)=nextSquareInPlay
      If off grid
      Then
         SUCCESS
      EndIf
   Else
      n=n+1
      If n is 10
      Then
         place 0 at grid[x,y]
         (x,y)=prevSquareInPlay
         If off grid
         Then
            FAIL
         Else
            n=grid[x,y]+1
         EndIf
      EndIf
   EndIf
EndLoop
```

5	3	1	2	7	6	4	9	8
6	2	4	1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

Algorithm

```
initialize grid to all zero
n=1, (x,y)=firstSquareInPlay
Loop
   place n at grid[x,y]
   If no violations
   Then
      n=1, (x,y)=nextSquareInPlay
      If off grid
      Then
         SUCCESS
      EndIf
   Else
      n=n+1
      If n is 10
      Then
         place 0 at grid[x,y]
         (x,y)=prevSquareInPlay
         If off grid
         Then
            FAIL
         Else
            n=grid[x,y]+1
         EndIf
      EndIf
   EndIf
EndLoop
```

5	3	1	2	7	6	4	9	8
6	2	4	1	9	5	3		
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

Algorithm

```
initialize grid to all zero
n=1, (x,y)=firstSquareInPlay
Loop
   place n at grid[x,y]
   If no violations
   Then
      n=1, (x,y)=nextSquareInPlay
      If off grid
      Then
         SUCCESS
      EndIf
   Else
      n=n+1
      If n is 10
      Then
         place 0 at grid[x,y]
         (x,y)=prevSquareInPlay
         If off grid
         Then
            FAIL
         Else
            n=grid[x,y]+1
         EndIf
      EndIf
   EndIf
EndLoop
```

5	3	1	2	7	6	4	9	8
6	2	4	1	9	5	7		
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

Algorithm

```
initialize grid to all zero
n=1, (x,y)=firstSquareInPlay
Loop
   place n at grid[x,y]
   If no violations
   Then
      n=1, (x,y)=nextSquareInPlay
      If off grid
      Then
         SUCCESS
      EndIf
   Else
      n=n+1
      If n is 10
      Then
         place 0 at grid[x,y]
         (x,y)=prevSquareInPlay
         If off grid
         Then
            FAIL
         Else
            n=grid[x,y]+1
         EndIf
      EndIf
   EndIf
EndLoop
```

5	3	1	2	7	6	4	9	8
6	2	7	1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

Algorithm

```
initialize grid to all zero
n=1, (x,y)=firstSquareInPlay
Loop
   place n at grid[x,y]
   If no violations
   Then
      n=1, (x,y)=nextSquareInPlay
      If off grid
      Then
         SUCCESS
      EndIf
   Else
      n=n+1
      If n is 10
      Then
         place 0 at grid[x,y]
         (x,y)=prevSquareInPlay
         If off grid
         Then
            FAIL
         Else
            n=grid[x,y]+1
         EndIf
      EndIf
   EndIf
EndLoop
```

5	3	1	2	7	6	4	9	8
6	2	7	1	9	5	3		
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

Algorithm

```
initialize grid to all zero
n=1, (x,y)=firstSquareInPlay
Loop
   place n at grid[x,y]
   If no violations
   Then
      n=1, (x,y)=nextSquareInPlay
      If off grid
      Then
         SUCCESS
      EndIf
   Else
      n=n+1
      If n is 10
      Then
         place 0 at grid[x,y]
         (x,y)=prevSquareInPlay
         If off grid
         Then
            FAIL
         Else
            n=grid[x,y]+1
         EndIf
      EndIf
   EndIf
EndLoop
```

5	3	1	2	7	6	4	9	8
6	4		1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

Algorithm

```
initialize grid to all zero
n=1, (x,y)=firstSquareInPlay
Loop
   place n at grid[x,y]
   If no violations
   Then
      n=1, (x,y)=nextSquareInPlay
      If off grid
      Then
         SUCCESS
      EndIf
   Else
      n=n+1
      If n is 10
      Then
         place 0 at grid[x,y]
         (x,y)=prevSquareInPlay
         If off grid
         Then
            FAIL
         Else
            n=grid[x,y]+1
         EndIf
      EndIf
   EndIf
EndLoop
```

5	3	1	2	7	6	4	9	8
6	4	7	1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

Algorithm

```
initialize grid to all zero
n=1, (x,y)=firstSquareInPlay
Loop
   place n at grid[x,y]
   If no violations
   Then
      n=1, (x,y)=nextSquareInPlay
      If off grid
      Then
         SUCCESS
      EndIf
   Else
      n=n+1
      If n is 10
      Then
         place 0 at grid[x,y]
         (x,y)=prevSquareInPlay
         If off grid
         Then
            FAIL
         Else
            n=grid[x,y]+1
         EndIf
      EndIf
   EndIf
EndLoop
```

5	3	1	2	7	6	4	9	8
6	4	7	1	9	5	3		
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9