Ece Alptekin
24156
Assignment 2: Solving Aquarium Puzzle using CSP

CSP representation of Aquarium puzzle

Variables are defined as a cell of the Aquarium puzzle. The variables are initialized in a nested loop. They are defined by their coordinates in the puzzle. Each variable has the domain {0,1}. If there is a water in the cell, the value “1” is assigned to the corresponding variable, which has i as the row number and j as the column number. If there is no water in the cell, the value “0” is picked from the domain of the variable and assigned to the variable.

```
# Variables
water = {}
for i in range(size):
    for j in range(size):
        water[i, j] = model.NewBoolVar('%ij%i' % (i,j))
```

The constraint below should be satisfied. For each row, the number of the filled cells should be equal to the clue, which is located outside the grid and corresponds this row. In the implementation, I store clues of the rows to use while defining this constraint.

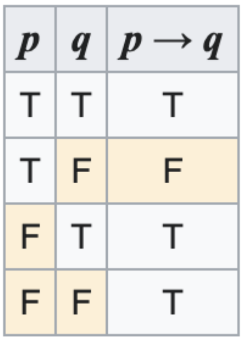
```
rowb = Board.getRow()
for i in range(size):
    row_constraint = rowb[i]
    model.Add(sum(water[i, j] for j in range(size)) == row_constraint)
```

The constraint below should be satisfied. For each column, the number of the filled cells should be equal to the clue, which is located outside the grid and corresponds this column. In the implementation, I store clues of the columns to use while defining this constraint.

```
columnb = Board.getColumn()
for j in range(size):
    column_constraint = columnb[j]
    model.Add(sum(water[i, j] for i in range(size)) == column_constraint)
```

In the implementation, the board object stores the aquarium information. Different aquariums are represented with different digits in the input.

The constraint below satisfies the condition: If two cells are in the same aquarium and if there is a water cell on the top of the cell (same column, row-1), then this cell should be filled with water. Otherwise, if there is no water on the top of the cell, then this cell can contain water or not. This condition is represented with the implication operator in the implementation. I use the compound $water[i-1,j] \rightarrow water[i,j]$ to implement this constraint. The p corresponds $water[i-1,j]$, which is the cell on the top of this cell and q corresponds $water[i,j]$, which is the current cell. If p is true, meaning that $water[i-1,j]$ is filled with water and q is false, meaning that $water[i,j]$ is not filled, the compound $p \rightarrow q$ is false. thisaquarium is the aquarium of the current cell, upperaquarium is the aquarium of the top cell.



p	q	$p \rightarrow q$
T	T	T
T	F	F
F	T	T
F	F	T

```
for i in range(1, size):
    for j in range (size):

        thisaquarium = Board.getAquarium(i,j)
        upperaquarium = Board.getAquarium(i-1,j)

        if(thisaquarium == upperaquarium):
            model.AddImplication(water[i-1,j], water[i,j])
```

The constraint below satisfies the condition: If two cells are in the same aquarium and if there is a water in the cell, then its left cell (same row, column-1) should be filled with water. Otherwise, if there is no water in the cell, then its left cell has no water. I declare an intermediate boolean variable b to store the value of the cell. If the cell has a water, then b is true and if the cell has no water, then b is false. If b is true, then the left cell has water and if b is false, then the left cell has no water. thisaquarium is the aquarium of the current cell, leftaquarium is the aquarium of the left cell.

```python
#If the aquarium level is the same with the left, the value should be same.
for i in range (size):
    for j in range(1, size):

        thisaquarium = Board.getAquarium(i,j)
        leftaquarium = Board.getAquarium(i,j-1)

        if(thisaquarium == leftaquarium):

            # Declare our intermediate boolean variable.
            b = model.NewBoolVar('b')


            model.Add(water[i,j] == 1).OnlyEnforceIf(b)
            model.Add(water[i,j] == 0).OnlyEnforceIf(b.Not())


            model.Add(water[i, j-1] == 1).OnlyEnforceIf(b)
            model.Add(water[i, j-1] == 0).OnlyEnforceIf(b.Not())
```

The constraint below satisfies the condition: If two cells are in the same aquarium and if there is a water in the cell, then its right cell (same row, column+1) should be filled with water. Otherwise, if there is no water in the cell, then its right cell has no water. I declare an intermediate boolean variable b to store the value of the cell. If the cell has a water, then b is true and if the cell has no water, then b is false. If b is true, then the right cell has water and if b is false, then the right cell has no water. thisaquarium is the aquarium of the current cell, rightaquarium is the aquarium of the right cell.

```python
#If the aquarium level is the same with the right, the value should be same.
for i in range (size):
    for j in range (size-1):

        thisaquarium = Board.getAquarium(i,j)
        rightaquarium = Board.getAquarium(i,j+1)

        if(thisaquarium == rightaquarium):

            # Declare our intermediate boolean variable.
            b = model.NewBoolVar('b')


            model.Add(water[i,j] == 1).OnlyEnforceIf(b)
            model.Add(water[i,j] == 0).OnlyEnforceIf(b.Not())


            model.Add(water[i, j+1] == 1).OnlyEnforceIf(b)
            model.Add(water[i, j+1] == 0).OnlyEnforceIf(b.Not())
```

A discussion on whether A* or CSP is more appropriate for solving this puzzle

CSP aims to find a model, which is an assignment of values to all of its variables that satisfies all of its constraints, or determine whether or not a model exists. For solving this puzzle, the purpose is to complete assignments of values to variables that satisfy all constraints.

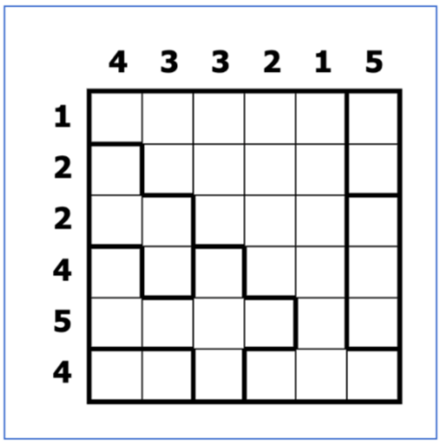
In A* states are black box, but in CSP they are assignments of values to a subset of the variables, which means we have more information about the states and we can make use of it. In CSP goal structure is exploited by the algorithm, therefore the goal state cannot be a black box.

The path to a goal isn't important in CSP, only the solution is. A* is mostly used to store the path. The path is not considered for solving this puzzle.

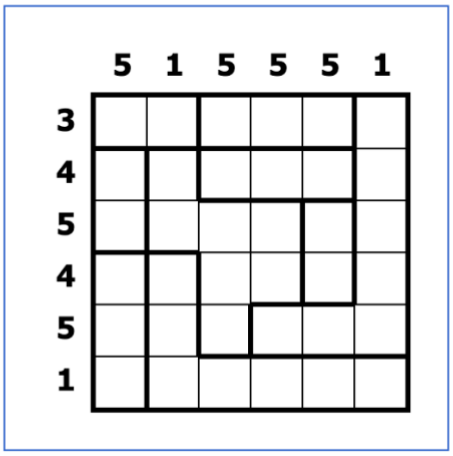
A heuristic function as estimating of the distance to the goal is not used for solving this puzzle. There is no cost function and weights. A* is generally used in weighted graphs to find a path to the given goal node having the smallest cost. Hence, A* is not suitable for solving this puzzle.

Ece Alptekin
24156
Assignment 2: Solving Aquarium Puzzle using CSP

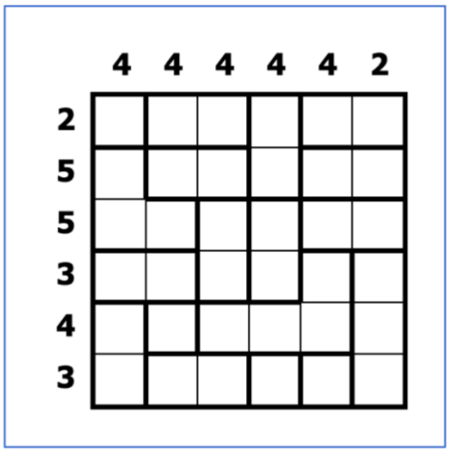
Aquarium puzzle



6x6 Easy Puzzle ID: 8,763,407

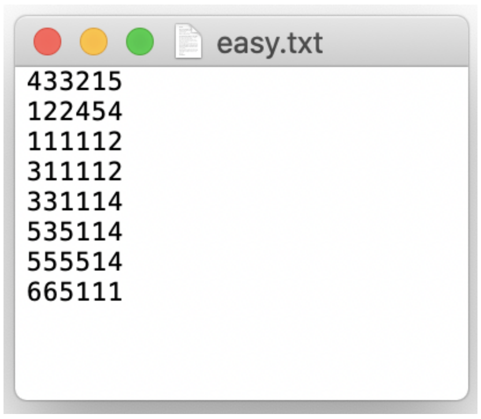


6x6 Normal Puzzle ID: 8,243,029

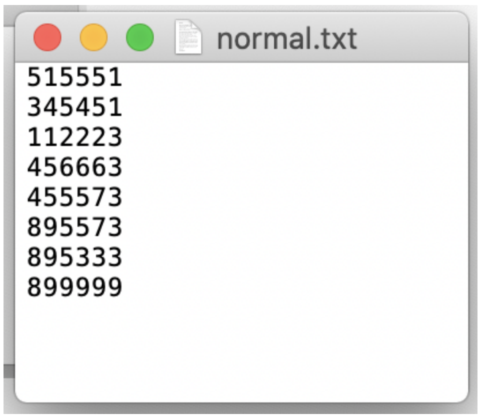


6x6 Hard Puzzle ID: 10,404,647

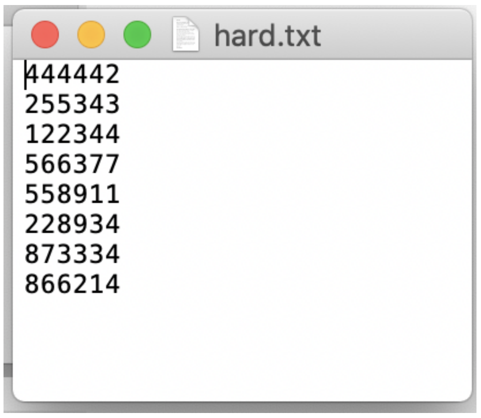
The puzzles presented to the CSP solver



```
433215
122454
111112
311112
331114
535114
555514
665111
```



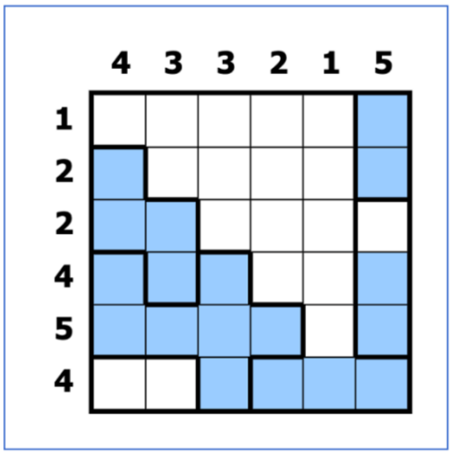
```
515551
345451
112223
456663
455573
895573
895333
899999
```



```
444442
255343
122344
566377
558911
228934
873334
866214
```

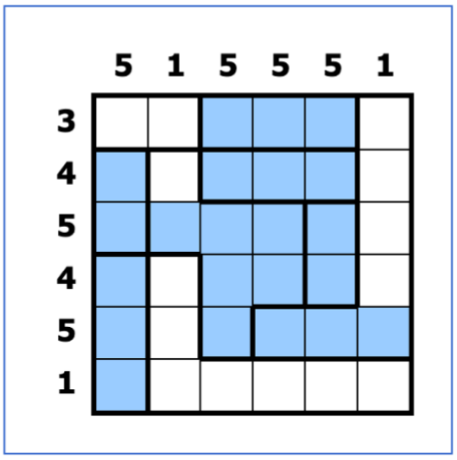
Ece Alptekin
24156
Assignment 2: Solving Aquarium Puzzle using CSP

The solutions of the puzzles computed by the CSP solver



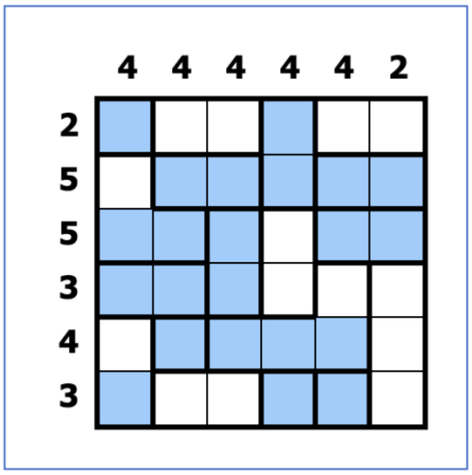
6x6 Easy Puzzle ID: 8,763,407

```
_ _ _ _ _ W
W _ _ _ _ W
W W _ _ _ _
W W W _ _ W
W W W W _ W
_ _ W W W W
```



6x6 Normal Puzzle ID: 8,243,029

```
_ _ W W W _
W _ W W W _
W W W W W _
W _ W W W _
W _ W W W W
W _ _ _ _ _
```



6x6 Hard Puzzle ID: 10,404,647

```
W _ _ W _ _
_ W W W W W
W W W _ W W
W W W _ _ _
_ W W W W _
W _ _ W W _
```