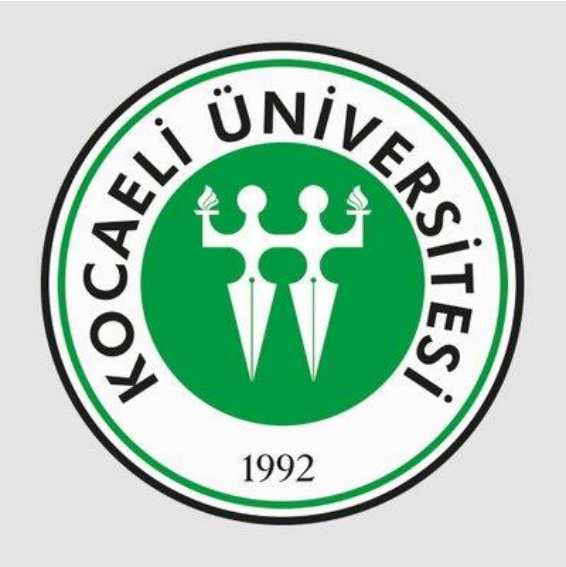
KOCAELİ UNİVERSİTESİ

BİLGİSAYAR MUHENDİSLİĞİ



YAZILIM LABORATUVARI I

PROJE 2

-MULTİTHREAD ile SAMURAI SUDOKU ÇÖZÜCÜ –

numan.mercan24@gmail.com

200202098 Muhammed Numan MERCAN

ÖZET

Projede bizden ,bir samurai sudokuyu multithread yapısı kullanılarak çözmemiz istenmektedir. Bu sudoku değerleri .txt dosyasından dinamik olarak çekilmek istenmiştir.

- Verilen 5 sudoku icin her birinde ayrı olmak üzere 5 thread ile çözüme ulaşılması istenmiştir.
- Verilen 5 sudoku icin her birinde ayrı 2 şer nokta olmak üzere 10 thread ile çözüm istenmiştir.
- Bu 2 problemin zaman ve çözüm karesi arasındaki ilişkinin grafiği çizdirilecektir.

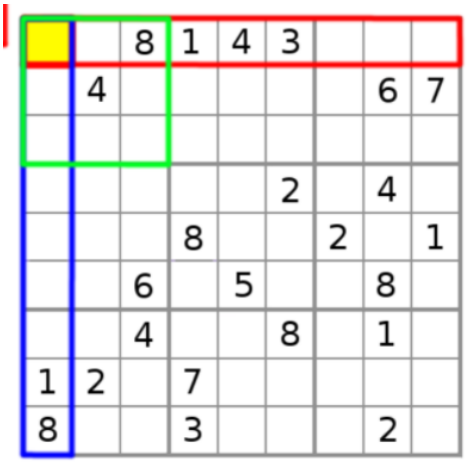
Not : Çözüm için hazır kütüphane kullanımı olmamalıdır çözüm bireysel yöntemler ile bulunmalıdır.

GİRİŞ ve YÖNTEM

İşe sudoku sonra samurai sudoku mantığını ve nasıl çözüldüğünü öğrenerek başladım. İnternette örnek sudoku algoritmalarını inceledim en temel mantık ve koşulları öğrendikten sonra algoritma taslağını oluşturmuştum . Ardından yeni öğreniyor olduğum multithread yapısını hem dil bağımsız hemde Java ile anlamaya çalıştım. Kullanımı , mantığı ve algoritması hakkında bilgi edindikten sonra taslak olarak yazacağım kodu nasıl multithread yapısı ile kuracağımı düşündüm. Yeni olmam sebebiyle multithread kullanımını önce tek thread kullanarak klasik yöntemle oluşturdum ardından multithread yapısını koda entegre etmeye çalıştım. Bu şekilde algoritmayı programladım ve birkaç hata ile çalışır hale getirdim.

İLERLEYİŞ

Proje isterlerine göre elde ettiğim samurai sudoku kaynaklarını toparladım ve çizerek ve çözerek yazacağım algoritmayı içselleştirdim. Yapmamız gereken satır , sütun ve 3x3 lük yerel box kontrolüydü bu kontrollerin tümünü karşılayan bir değer sudokuya yazdırılıyordu. Ortadaki hariç tüm sudokular bu şekilde çözüme kavuştu. Orta sudoku diğerlerine bağımlı bir çözüme sahip olması gerektiği için ona ayrı bir algoritma oluşturmam gerekti. Fakat bu algoritmayı oluşturup yazacak vakte sahip olmadığım için bu projede maalesef eksik kaldı.

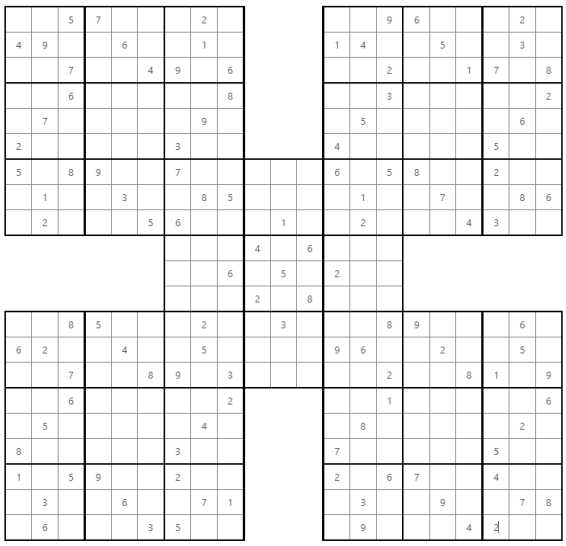


Sekil 1- (Klasik sudoku kontrolü)

Bu kontroller sonucunda tüm sudokular çözülü şekilde yazdırılıyor. Tek iplikle gerçekleşen bu uygulamayı çok iplikli bir yapıya taşıma işine ancak tek iplikli halini tamamıyla çözdükten sonra koyuldum.

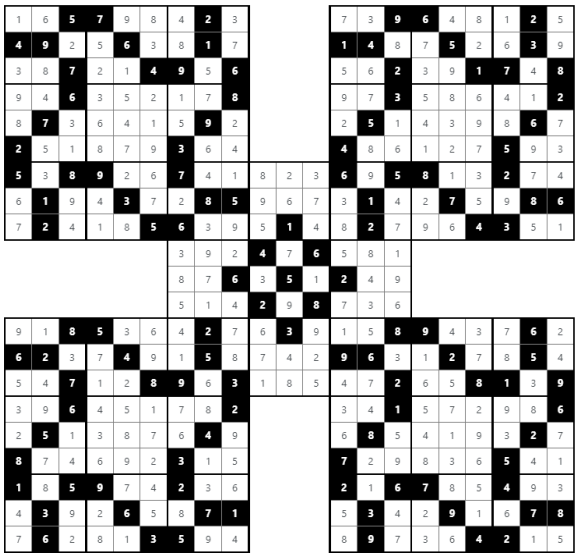
Bu şekilde algoritmayı multithread yapıya taşıdım ve çözdürmeyi başardım. Fakat çalışan threadlerin çözümü beklediğimden çok daha uzun sürüyordu. Bu sorunu multithread yapısına tam hakim olamdagımdan dolayı çözemedim , fakat gözden kaçan bir şeyin olduğunun ama onu anlayacak tecrübede olmadıgımın farkındayım.

Çözülen sudokuların doğruluğunu test etmek icin bir çözdürücü programı kullandım.



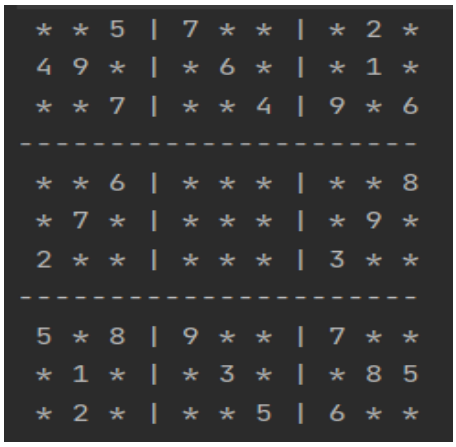
Sekil 2 – (Sudoku cozulmemis hali)

Sudokuyu çözdürerek kendi algoritmam üzerinde kontrolleri sağladım.



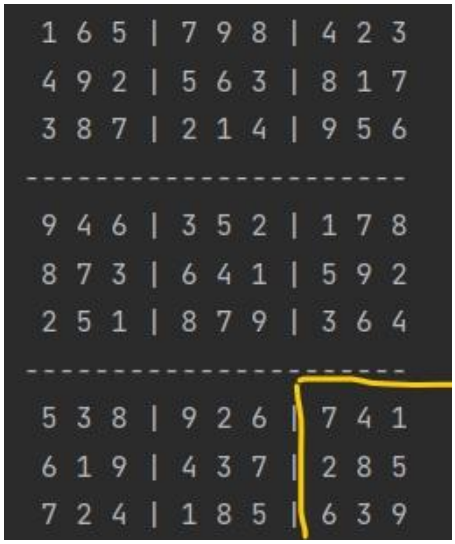
Sekil 3- Çözülmüş Samurai Sudoku

Bu çözümleri algoritmam ile terminale yazdırdım ve kontrollerim sonucu doğru bir çözüm yaptığını doğrulamış oldum.



Sekil 4 – Board 1 çözülmemiş

Multithread kullanılarak çözdüğüm yapıyıda aşağıda paylaşıyor olacağım.



Sekil 5- Board 1 çözülmüş

Karşılaşılan Problemler

Yaşadığım en büyük problem multithread yapısını nerede koymamı bilememdi , bu yüzden bu yapı uzun süren bir işlem sürecine dönüştü , bu süreci kısaltmaya calıstım fakat çözemedim. 2. Problemim grafik arayüze aktaramamak oldu fakat bunun sebebi kısıtlı vakitte bunu yapamayacağımı anlayıp odağımı projenin thread isterlerine yöneltmemdi.

Deneysel Sonuclar ve Gözlem

Programı çalıştırıp denemelerimiz sonucu thredi hangi fonksiyonda calistirmam gerektiğini az çok anladım fakat o fonksiyona implement edecek süreye sahip olmadığım icin bu haliyle bıraktım. Bu multithread yapısıyla program yaklaşık 4 dk kada sudokuyu çözüme kavuşturuyor.

```
Sudoku -1- basari ile cozuldu!
Board 1 icin gecen sure : 0.30111668 dk
Sudoku -2- basari ile cozuldu!
Board 2 icin gecen sure : 2.4486334 dk
Sudoku -3- basari ile cozuldu!
Board 3 icin gecen sure : 3.3314333 dk
Sudoku -4- basari ile cozuldu!
Board 4 icin gecen sure : 3.3895333 dk
Sudoku -mid- basari ile cozuldu!
Board mid icin gecen sure : 4.7785 dk
```

Sekil 6 Çözüm Süreleri

Burada yanlış anlaşılmaması icin bir şey söylemem gerekiyor. Bu sürelerin ardarda olması multithread yapısına aykırı gözükebilir fakat multithread asıl olarak her puzzle ın kendi içindeki çözümde kullanılmıştır.

Gecen total süre:

```
Cozumde gecen sure : 4.7785335 dk
```

Olası Çözüm Algoritması

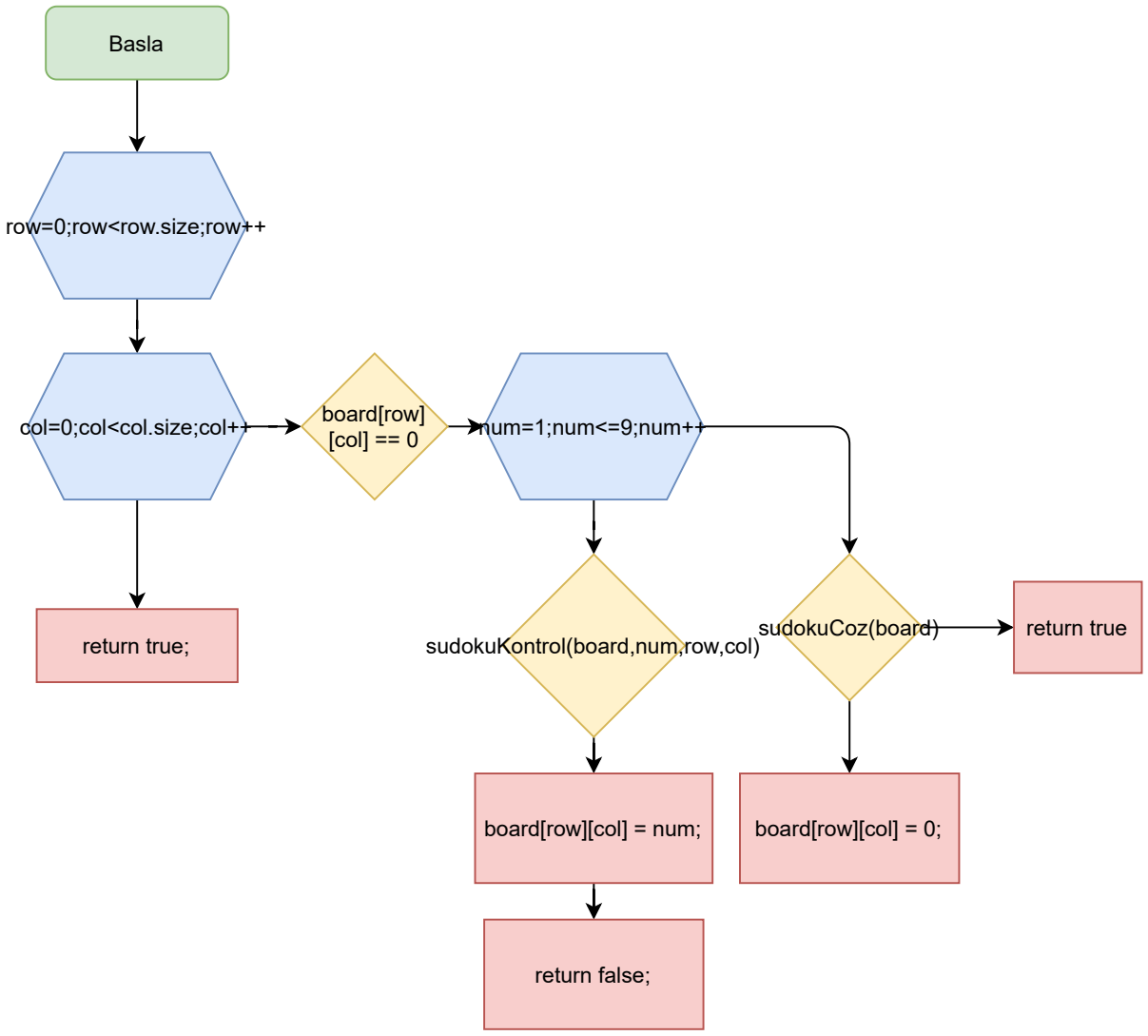
Eger vaktim olsaydı thread yapısını hem her boarda ayrı hem içlerine ayrı şekilde uygulardım. Samurai sudoku içinde kosedeki ortak matrisleri ayrı bir 3x3 lük matrise atar o sekilde kontrol ve atamaları sağlardım. Bu şekilde iç sudoku da diğerlerine bağlı olarak bulunmuş olurdu.

SONUÇ

Bu proje thread kavramını biraz olsun kavramamı ve onları nasıl kullanacağım hakkında yardımcı fikirler edinmemi sağladı. Güzel bir algoritma alıştırması olan sudokuyu hem normal şekilde hemde algoritmasını oluşturma şeklinde öğrenmemi ve farklı trickler ile basit ve güçlü fonksiyonlar oluşturmayı öğrenmemi sağladı. Bir problemin kompleks olmasına bakmaksızın , sağlam bir anlayış ve algoritma ile basit bir şekilde çözülebileceğini görmüş oldum.

Proje ne yazık ki tam anlamıyla bitmedi ve oldukça eksiğe sahip. Fakat kısıtlı sürede bana birçok şey katmayı başarmış ve oldukça verimli bir süreç geçirmemi saglamıstır.

Projeye ait akış şeması ve yalanncı kodu aşağıda bulabilirsiniz. Aynı şekilde proje süresince yararlandığım tüm kaynakları direk linkleri ile birlikte “REFERANS” kısmında bulabilirsiniz.



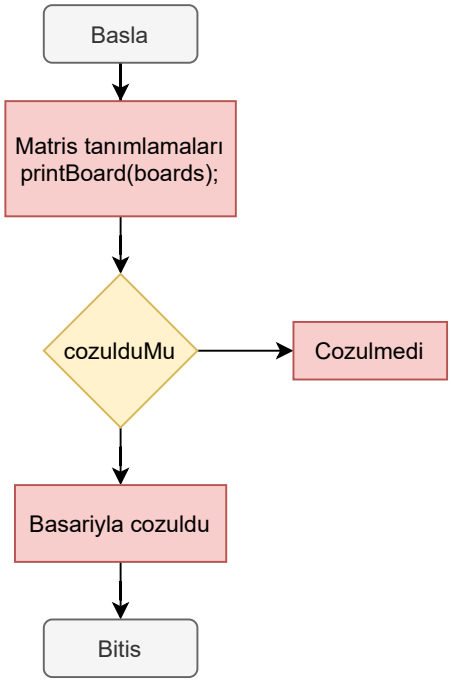
Yalanci Kod

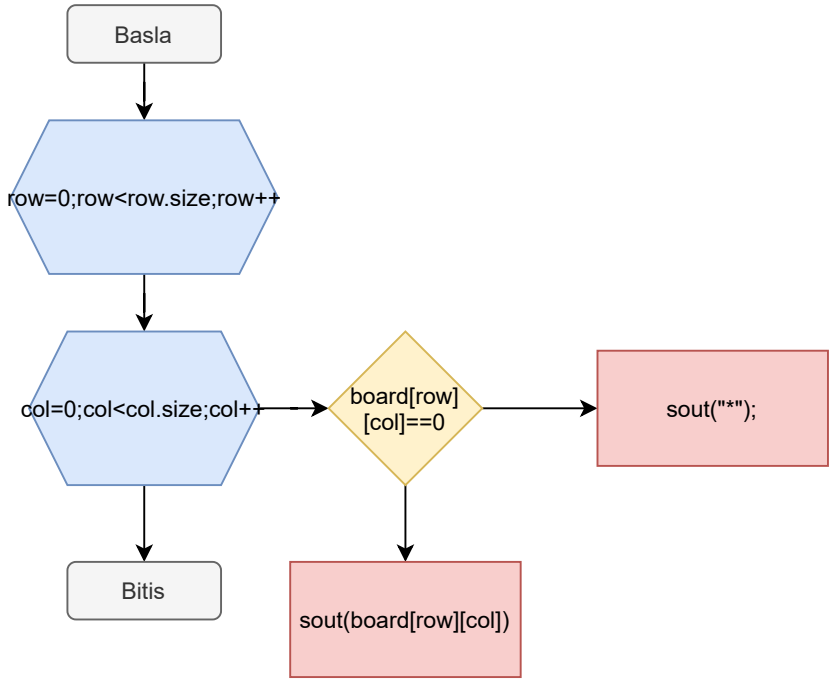
```
main(){
   dizi tanımlamaları
   printBoard(boards);// çözülmemis hali yazdırılıyor
   if(solveBoards) sout("Basariyla çözüldü");
   else sout("Cozulemedi");

   printBoard(boards); // sudokunun çözülen hali yazdiriliyor
}


printBoard(int [][] board){
      for(row=0;row<row.size;row++){
        for(col=0;col<col.size;col++){
           if (col%3==0 ve col != 0) sout("| ");
           if (board[row][col]==0) sout("*");//deger 0 sa bos demektir
           else sout(board[row][col])
         }
       }
}

satirNumaraKontrol(int[][] board, int number, int row){
          for(i=0;i<row.size;i++)
             if board[row][i] == number return true

           return false;
       }
```

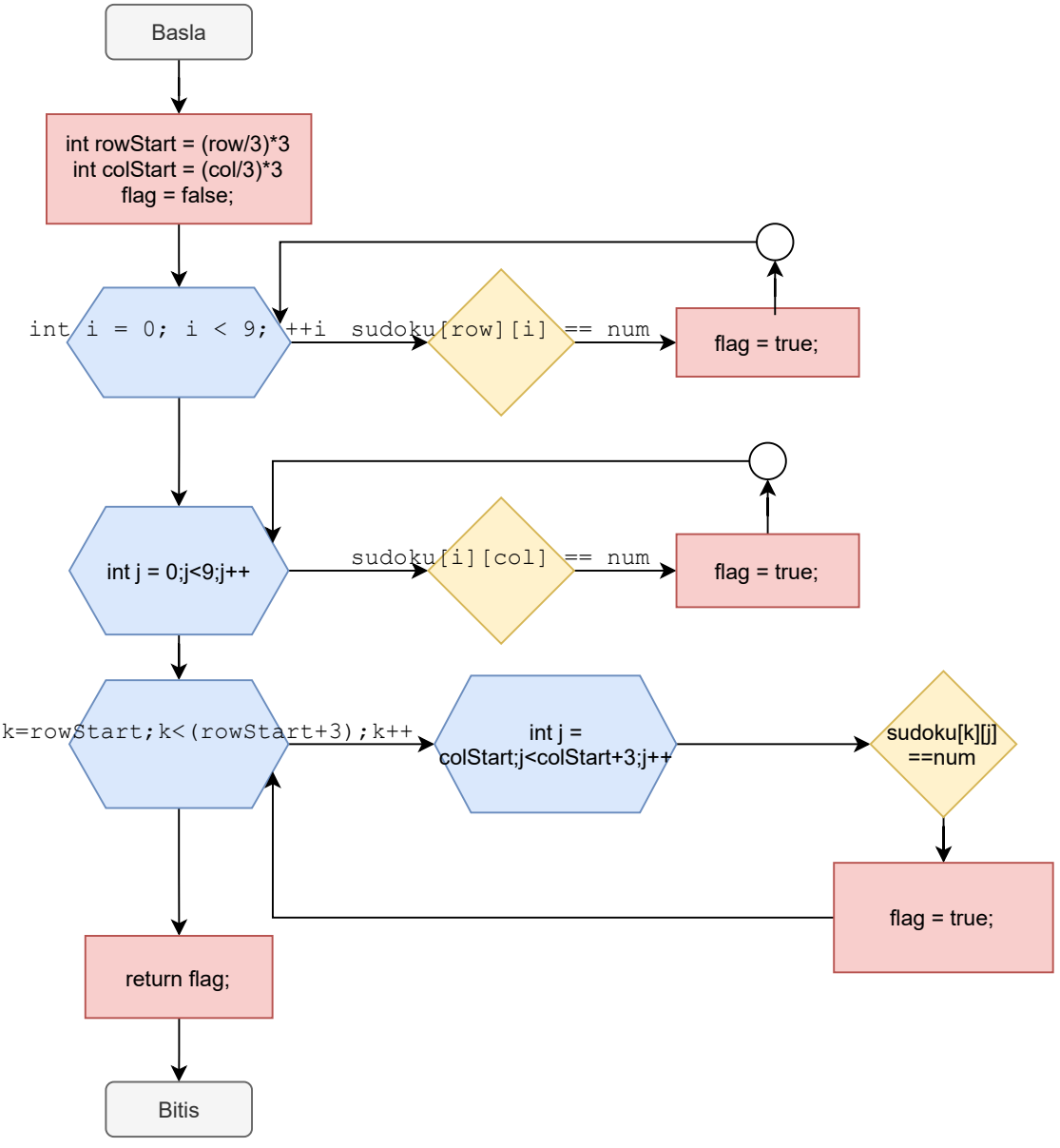
1

```
kutuNumaraKontrol(int[][] board, int number, int row, int column){
          int kutuSatir = row - row%3;
          int kutuSutun = col - col%3;

               for(i=kutuSatir;i<kutuSatir+3;i++)
                   for(j=kutuSutun;j<kutuSutun+3;j++)
                       if(board[i][j] == number) return true;

              return false;
       }

sudokuKontrol(int[][] board, int number, int row, int column) {
       return !satirKontrol(board,number,row) &&
              !sutunKontrol(board,number,col) &&
              !kutuKontrol(board, number, row, column);
    //tum kontrolleri karsilayip karsilamadigini burda kontrol ettik
  }

sudokuCoz(int[][] board){
       for (row = 0; row < row.size; row++) {
           for (col = 0; col < col.size; col++) {
             if (board[row][column] == 0) {
               for (sayi = 1; sayi<= GRID_SIZE; sayi++) {
                if sudokuKontrol(board, sayi,row,col)) {
                  board[row][column] = sayi;

                  if(sudokuCoz(board)) return true;
                  else board[row][col] = 0;

              }
             return false;
 }
       return true;
```

2


Basla
Matris tanımlamaları
printBoard(boards);
cozulduMu
Cozulmedi
Basariyla cozuldu
Bitis


Basla
row=0;row<row.size;row++
col=0;col<col.size;col++
board[row]
[col]==0
sout("*");
Bitis
sout(board[row][col])


Basla
int rowStart = (row/3)*3
int colStart = (col/3)*3
flag = false;
int i = 0; i < 9; ++i
sudoku[row][i] == num
flag = true;
int j = 0;j<9;j++
sudoku[i][col] == num
flag = true;
k=rowStart;k<(rowStart+3);k++
int j =
colStart;j<colStart+3;j++
sudoku[k][j]
==num
flag = true;
return flag;
Bitis

REFERANSLAR

Articles and Tutorials

- https://stackoverflow.com/questions/48882654/sudoku-solver-using-multi-threading

- https://github.com/lpelczar/Multithreaded-Sudoku-Solver

- https://github.com/billthefarmer/samurai-sudoku

- https://www.geeksforgeeks.org/program-sudoku-generator/

- https://github.com/pocketjoso/sudokuJS

- https://www.codeproject.com/Articles/237372/Informed-search-algorithms-to-solve-Sudoku-Samurai
  -Hosein Fereidooni , Article (11/08/2011)

Video

- https://www.youtube.com/watch?v=mcXc8Mva2bA&ab_channel=CodingwithJohn -> Sudoku App Tutorial

Helper Services

- https://www.samurai-sudoku.com/