MULTITHREAD KULLANARAK SAMURAI SUDOKU ÇÖZME

AKIN SALİM ERDEM-190201072

Bilgisayar Mühendisliği Bölümü

Akiinerdem@gmail.com

1-)Projenin Tanımı

Bu kısım sadece projenin açıklamasını okuyup edindiğimiz ön bilgiye göre yazılmıştır.

Projenin açıklamasını dikkatli bir şekilde okuduğumuzda bizden multithread yapısını kullanarak verilecek samurai sudoku üzerinden çözüm bulmamız istenmektedir.Threadleri yaparken 5 noktadan başlatmamız istenmektedir. Sudoku değerlerini .txt dosyasından almamız istenmektedir. Ayrıca threadlerin yaptığı çözümleri de bir .txt ya da veritabanında tutmamız istenmektedir.

2-)Yapılan Araştırmalar ve Karşılaşılan Sıkıntılar

Bu kısım proje öncesi ve sonrası araştırmaları ve de projenin yapım aşamasındaki sıkıntıları ve çözümlerini içermektedir.

İlk önce bu projeyi yaparken hangi IDE ve programlama dilini kullanacağımı karar vermek için araştırmalar yaptım. Araştırmalar sonucu Intellij IDE'si ve java dilinde daha rahat olacağımı anladım ve o yönde çalışmalara başladım.

Projeyi yaparken multithread kullanmamız isteniyordu bundan dolayı birtakım multithread kullanım örneklerini inceledim.

Sudokuyu txtden veri çekerek oluşturmamız gerekiyordu. Txtden veri çekmeyi bilgisayar üstünden dosya seçme olarak yaptım. Txtden veri çekme işini samurai sudoku formatında yapamadım bu yüzden 9x9 luk normal sudoku üzerinden projeme devam ettim.

Threadlerin yaptığı çözümleri veritabanında tutmaktansa txtye yazdırmayı daha mantıklı bir çözüm olarak buldum ve projeme bu yönde devam ettim. Threadlerin tüm hareketlerini txtye yazdırdım.

3.1-)Kod Kısmı

Koda öncelikle .txt belgesi üzerinden veri çekme ve sudoku tablosunu ona göre oluşturarak başladım. Yazdığım kodda sizden bilgisayar üstünde bir txt belgesi seçmenizi istiyor. Yapılan seçimden sonra txt belgesindeki satır satır verileri çekiyor. Verileri * ve rakam olarak ayırıyor . Satır satır okurken aralara çizgi çekip verileri yerleştirerek son olarak sudoku tablosunu oluşturuyor.

Move sınıfında satır, sütun ve değer verileri bulunuyor. İçindeki constructor satır, sütun ve değer verilerine değer veriyor.

Thread sınıfında threadleri çalıştıran run fonksiyonu threadleri random yerden başlatıyor. Solve fonksiyonu sudokudaki boşlukları uygun bir biçimde çözerek dolduruyor. PrintMoves fonksiyonu çözülen sudoku kısımlarını sudoku ile birlikte yazdırıyor. Bekle fonksiyonu threadlerin arasında karmaşıklık olmaması için her çözümden sonra bir süre fonksiyonu bekletiyor. Böylece çözüm yaparken rahat oluyor ve hata riski azalıyor. CheckFinished fonksiyonu sudokunun bitip bitmediğini kontrol ediyor. Check fonksiyonu çözüm yaparken değerlerin doğru boşluğa gelip gelmediğini kontrol ediyor. Aynı rakam herhangi bir satırda var mı. Aynı rakam herhangi bir sütunda var mı. Aynı rakam herhangi bir 3x3 lük alanda var mı. Tüm kontrolleri sağlarsa true tüm kontrolleri sağlamazsa false döndürüyor. PrintSudoku sudokuyu yazdırıyor. PrintTime sudoku çözülürkenki zamanı tutup kaç milisaniyede sudokuyu çözmeyi tamamladığını ve hangi çözücü tarafından çözdüğünü yazıyor. PrintMovesToTxt threadlerin yaptığı bütün işlemleri txtye yazıyor.

Run sınıfı tüm işlemlerin olduğu main sınıf. Threadler oluşturuluyor. Txt belgesi seçiliyor. Threadlerin bitip bitmediği kontrol ediliyor. Sudoku tahtasını kopyalayarak her threade farklı tahtaları veriyor. En sonda da sudoku tahtasını yazdırma fonksiyonu bulunuyor.

3.2-)Kod Bilgisi

Bu kısımda kodun fiziksel özellikleri yazmaktadır:

Programın tamamı kullanılan kütüphaneler, kendi yazdığımız fonksiyonlar ve main fonksiyonun içindeki kod kısımları da dahil olmak üzere toplam 427 satırdan oluşmaktadır.

Programda 1 main ve 2 tane de class bulunmakta.

Programa dahil edilmiş 13 kütüphane bulunmakta.

4-)Kütüphaneler

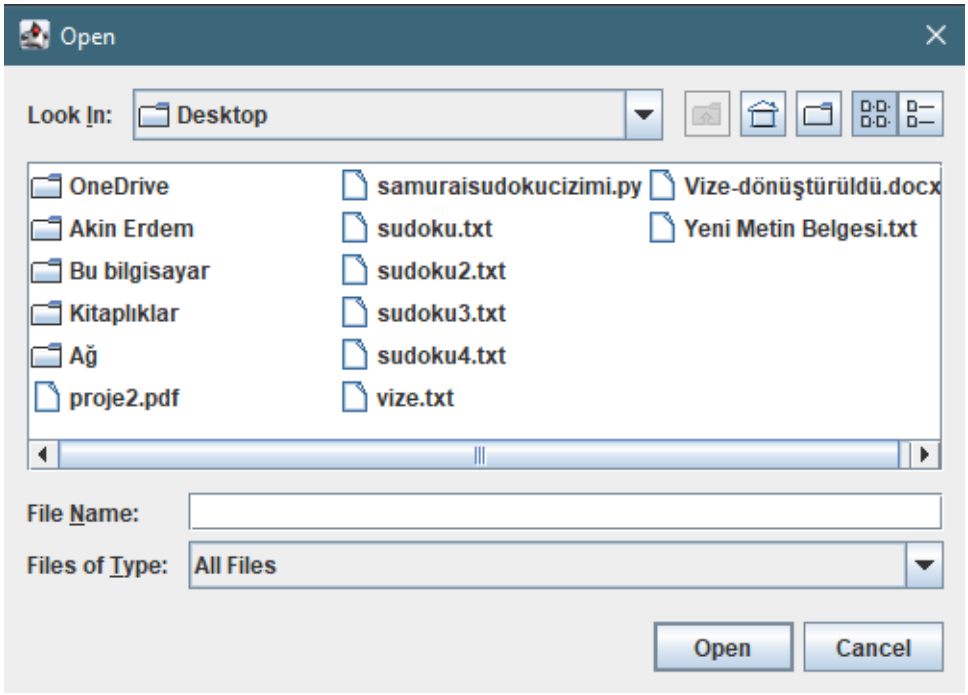
javax.swing.*
javax.swing.filechooser.FileSystemView
java.io.BufferedWriter
java.io.File
java.io.FileWriter
java.io.IOException
java.nio.file.Files
java.nio.file.Paths
java.util.stream.Stream

Java.util.ArrayList

Java.util.List

Java.util.Random

5-) Örnek Çıktılar




```
Sudoku To Solve:
 ----------------------
| * * * | * 4 * | * * * |
| * * 2 | 6 * 7 | 1 * * |
| 8 7 1 | * * * | 6 9 4 |
 ----------------------
| * 6 * | * * * | * 4 * |
| 2 * 5 | 9 * 6 | 7 * 8 |
| * 8 * | * * * | * 2 * |
 ----------------------
| 6 5 8 | * * * | 4 7 1 |
| * * 9 | 4 * 8 | 5 * * |
| * * * | * 7 * | * * * |
 ----------------------
```

```
Solved by Solver4 in 892 millisecond.
 -----------------------
| 5 3 6 | 1 4 9 | 2 8 7 |
| 4 9 2 | 6 8 7 | 1 5 3 |
| 8 7 1 | 3 2 5 | 6 9 4 |
 -----------------------
| 9 6 7 | 8 1 2 | 3 4 5 |
| 2 4 5 | 9 3 6 | 7 1 8 |
| 1 8 3 | 7 5 4 | 9 2 6 |
 -----------------------
| 6 5 8 | 2 9 3 | 4 7 1 |
| 7 1 9 | 4 6 8 | 5 3 2 |
| 3 2 4 | 5 7 1 | 8 6 9 |
 -----------------------
```

```
Solver2 Moves

[3][0] = 1  [5][0] = 3  [7][0] = 7  [8][0] = 4  [0][1] = 3  [1][1] = 4  [1][1] = 9  [4][1] = 4  [7][1] = 1  [8][1] = 2
[0][2] = 6  [3][2] = 7  [7][1] = 2  [8][1] = 1  [0][2] = 6  [3][2] = 7  [0][1] = 9  [1][1] = 3  [4][1] = 4  [7][1] = 1
[8][1] = 2  [0][2] = 6  [3][2] = 7  [7][1] = 2  [8][1] = 1  [0][2] = 6  [3][2] = 7  [1][1] = 4  [5][0] = 4  [7][0] = 3
[7][0] = 7  [8][0] = 3  [0][1] = 3  [1][1] = 4  [1][1] = 9  [0][1] = 9  [1][1] = 3  [1][1] = 4  [4][1] = 3  [7][1] = 1
[8][1] = 2  [0][2] = 3  [3][2] = 7  [0][2] = 6  [3][2] = 7  [7][1] = 2  [8][1] = 1  [0][2] = 3  [3][2] = 7  [0][2] = 6
[3][2] = 7  [5][0] = 7  [7][0] = 3  [8][0] = 4  [0][1] = 3  [1][1] = 4  [1][1] = 9  [4][1] = 4  [7][1] = 1  [8][1] = 2
[0][2] = 6  [3][2] = 3  [7][1] = 2  [8][1] = 1  [0][2] = 6  [3][2] = 3  [0][1] = 9  [1][1] = 3  [4][1] = 4  [7][1] = 1
[8][1] = 2  [0][2] = 6  [3][2] = 3  [7][1] = 2  [8][1] = 1  [0][2] = 6  [3][2] = 3  [1][1] = 4  [4][1] = 3  [7][1] = 1
[8][1] = 2  [0][2] = 3  [0][2] = 6  [7][1] = 2  [8][1] = 1  [0][2] = 3  [0][2] = 6  [5][0] = 9  [7][0] = 3  [8][0] = 4
[0][1] = 3  [1][1] = 4  [1][1] = 9  [4][1] = 4  [7][1] = 1  [8][1] = 2  [0][2] = 6  [3][2] = 3  [5][2] = 7  [3][2] = 7
[5][2] = 3  [7][1] = 2  [8][1] = 1  [0][2] = 6  [3][2] = 3  [5][2] = 7  [3][2] = 7  [5][2] = 3  [0][1] = 9  [1][1] = 3
[4][1] = 4  [7][1] = 1  [8][1] = 2  [0][2] = 6  [3][2] = 3  [5][2] = 7  [3][2] = 7  [5][2] = 3  [7][1] = 2  [8][1] = 1
[0][2] = 6  [3][2] = 3  [5][2] = 7  [3][2] = 7  [5][2] = 3  [1][1] = 4  [4][1] = 3  [7][1] = 1  [8][1] = 2  [0][2] = 3
[3][2] = 7  [5][2] = 4  [0][2] = 6  [3][2] = 7  [5][2] = 4  [7][1] = 2  [8][1] = 1  [0][2] = 3  [3][2] = 7  [5][2] = 4
[0][2] = 6  [3][2] = 7  [5][2] = 4  [7][0] = 7  [8][0] = 3  [0][1] = 3  [1][1] = 4  [1][1] = 9  [4][1] = 4  [7][1] = 1
[8][1] = 2  [0][2] = 6  [3][2] = 3  [5][2] = 7  [8][2] = 4  [0][3] = 1  [2][3] = 2  [3][3] = 5  [5][3] = 3  [3][3] = 7
[5][3] = 3  [5][3] = 5
Latest status of sudoku:
 -----------------------
| * 3 6 | 1 4 * | * * * |
| * 9 2 | 6 * 7 | 1 * * |
| 8 7 1 | 2 * * | 6 9 4 |
 -----------------------
| 1 6 3 | 7 * * | * 4 * |
| 2 4 5 | 9 * 6 | 7 * 8 |
| 9 8 7 | 5 * * | * 2 * |
 -----------------------
| 6 5 8 | * * * | 4 7 1 |
| 7 1 9 | 4 * 8 | 5 * * |
| 3 2 4 | * 7 * | * * * |
```

6-)Referanslar

- https://stackoverflow.com/questions/48882654/sudoku-solver-using-multi-threading
- https://en.wikipedia.org/wiki/Sudoku_solving_algorithms
- https://dev.to/aspittel/how-i-finally-wrote-a-sudoku-solver-177g
- http://pi.math.cornell.edu/~mec/Summer2009/meerkamp/Site/Solving_any_Sudoku_II.html