KOCAELİ ÜNİVERSİTESİ

BİLGİSAYAR MÜHENDİSLİĞİ BÖLÜMÜ

Yazılım Laboratuvarı I- II. Proje

Multithread Kullanarak Samurai Sudoku Çözme

Gökçe Yılmaz yilmaz.gokce.tr@gmail.com

Özet ve problem tanımı

Bu projedeki amaç multithread yapısını kullanarak verilen samurai sudoku üzerinden çözüm bulmaktır. Beş ve on adet thread kullanılan iki çözüm istenmektedir. Bu sayede threadlerin çözüme ve çözüm hızına katkısının görülmesi beklenmektedir. Farkı görsel olarak da görebilmek için grafiksel bir çıktı istenmektedir.

I. Giriş

Projede Python programlama dili ve PyCharm 2021.2.3 geliştirme ortamı kullanılmıştır. Arayüz tasarımı için Python'un bir kütüphanesi olan *pygame*'den yararlanılmıştır.

Proje *main.py* üzerinden yürütülmektedir. İlk açıldığında sudokunun çözümsüz hali ve çözüm butonlarının görsel yönetimi ve genel yönetim *app.py* üzerinden yapılmaktadır. Sudokunun çözüm işlemleri ve konsolda yazdırılması *samurai.py* üzerinden yapılmaktadır. Çözülen sudokunun kontrolü *kontrol.py* üzerinden yapılmaktadır. Görsel ayarlar ve dosya okuma ayarları *settings.py* üzerinde yazılmıştır. Son olarak ise çözülen sudoku görsel olarak *solution.py* aracılığıyla kullanıcıya gösterilmektedir.

II. Yapılan araştırmalar

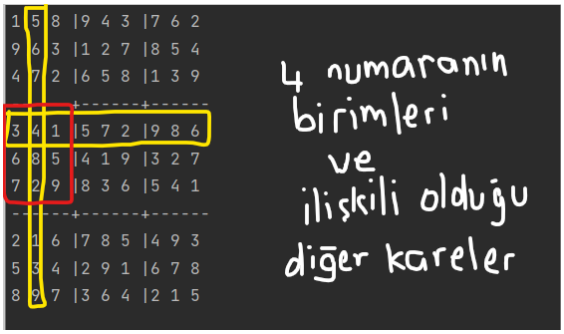
Öncelikle sudoku çözmek için normal hayatta kullanılan yöntemler ele alındı. Samurai sudokunun özellikleri araştırıldı. Kullanılabilecek en optimum algoritma araştırıldı ve Depth-first Search algoritması seçildi. Bu algoritma "Derinlik öncelikli arama" şeklinde Türkçeye çevrilmektedir.

Depth-first Search algoritması bizim belirlediğimiz bir root (kök) node'dan başlıyor ve herhangi bir çocuğunu seçiyor. Daha sonra bu çocuktan daha önce gezmediğimiz herhangi bir node'a gidiliyor. Bu seçimlerdeki iki kural var birincisi seçilen node'a direkt yol olmalı diğeri ise o node daha önce gezilmemiş olmalı. Bu kuralları uygulayarak rekürsif bir şekilde geziliyor.

Ancak öyle bir node'a geliniyor ki gidilecek yer kalmıyor. Çünkü direkt bağlantının olduğu bütün node'lar daha önceden gezilmiş. Böyle bir durumda ise bu node'dan önce en son gezilen node'a geri dönülüyor ve başka tercihler var mı diye bakılıyor. Algoritma rekürsif olarak çalıştığı için her takılınan noktada bir üste çıkmaya olanak sağlıyor. Algoritma başlanılan yere gelindiğinde ve hali hazırda bütün çocuklar gezildiğinde bitiyor.

III. Yöntem ve genel yapı

Bu bölümde sudoku çözümünde kullanılan yöntem açıklanacaktır. Samuria sudokunun içindeki her sudoku kendi içinde 81 adet kareden oluşmaktadır. Bu kareler 9 satır ve 9 sütunu oluşturmaktadırlar. İşlem kolaylığı açısından satırlar A'dan I'ya, sütunlar 1'den 9'a ayrıştırılabilirler. Ve her dokuz küçük karenin oluşturduğu kümeye ise birim diyebiliriz. Birimler sağdan sola, yukarından aşağıya ve kare oluşturan bir küme şeklinde olabilir. Her bir küçük kare birimi paylaştığı diğer karelerle ilişkilidir. Buradan çıkarımla her bir küçük karenin 3 birimi ve ilişkili olduğu 20 küçük kare olduğu söylenebilir.



Resim 1

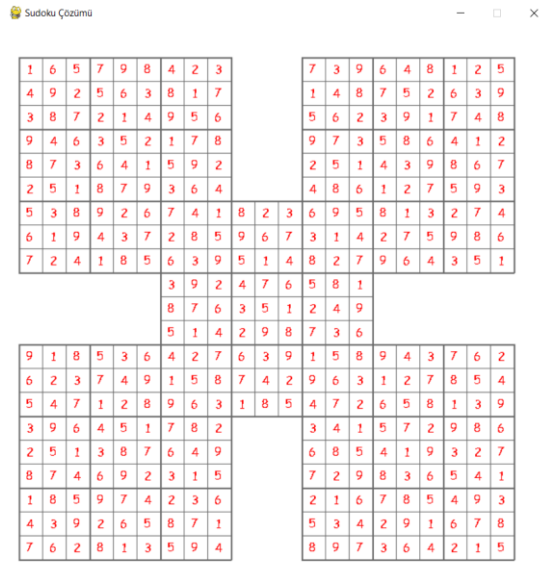
Bu şekilde vektörel fonksiyonunda vektörel çarpım yapılarak satırlar ve sütunlar "A1,A2,…,C3" şeklinde sıralanmıştır. Samurai sudokuda beş adet sudoku bulunduğundan dolayı her bir sudokuyu tanımlamak için ayrıca harfler

kullanılmıştır. Sol üst için “a”, sağ üst için “b”, sol alt için “c”, sağ alt için “d” ve orta için “+” karakterleri kullanılmış ve vektörel fonksiyonu buna göre güncellenmiştir. Sonuçta “A1a,A2a,…,A1b” biçiminde her karenin hangi sudokuya ait olduğunu gösteren bir isimlendirme yöntemi kullanılmıştır.

Karelerin değerleri bulunurken dept first search algoritmasından da yararlanarak eleme yöntemi izlenilmiştir. İlk başta bir kare bütün rakamları alabilir olarak farzedilmiştir ve eleyerek alınabilecek rakamlar azaltılmış en son bire düşene kadar devam edilmiştir.

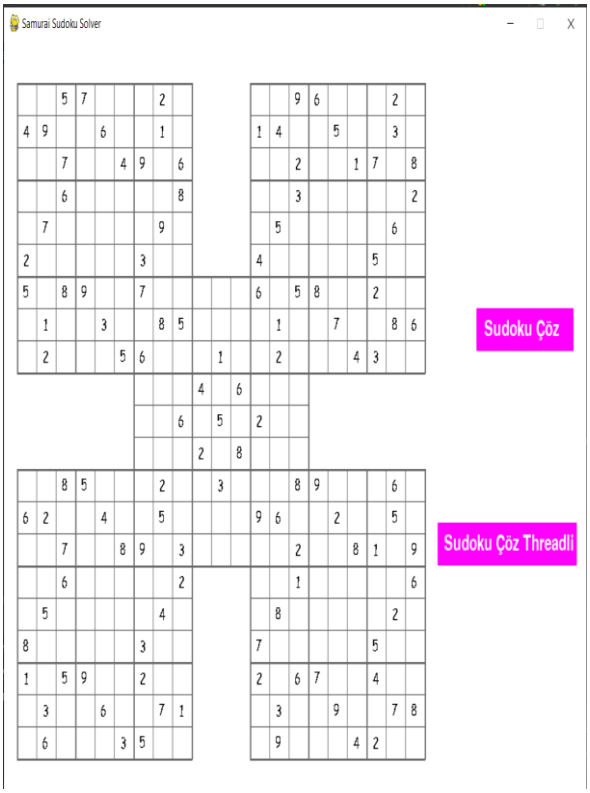
Bulunan her bir sudoku hem konsolda yazdırılmış hem de görsel arayüz olarak ekranda gösterilmiştir.

Threadli çözümde thread aşamaları *threadadimlai.txt* dosyasına yazdırılmış ve süre bilgisi eklenmiştir.

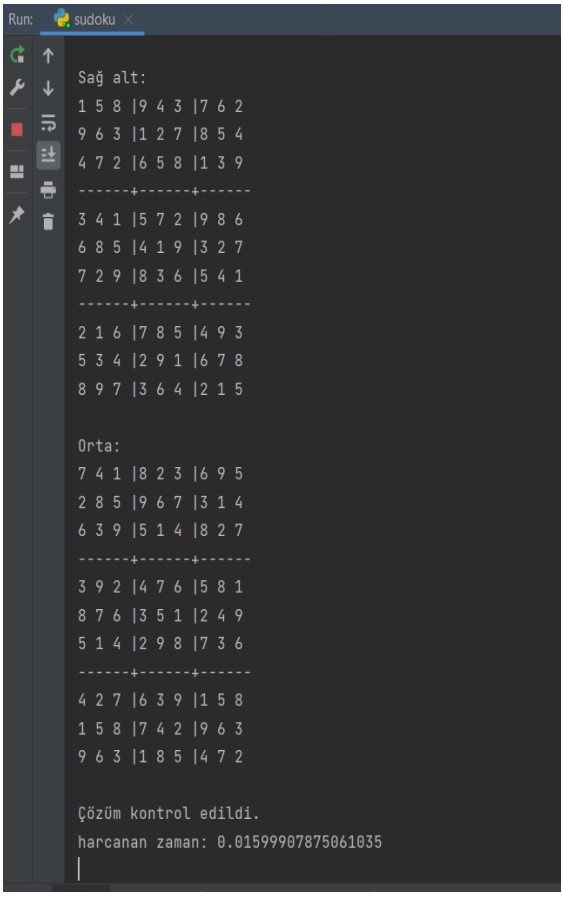


Resim 3: Çözüm ekranı

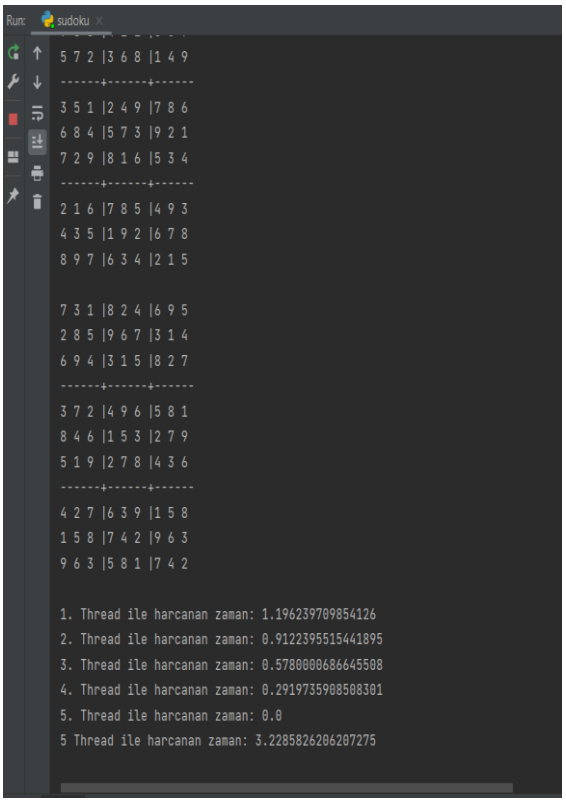
IV. Tasarım



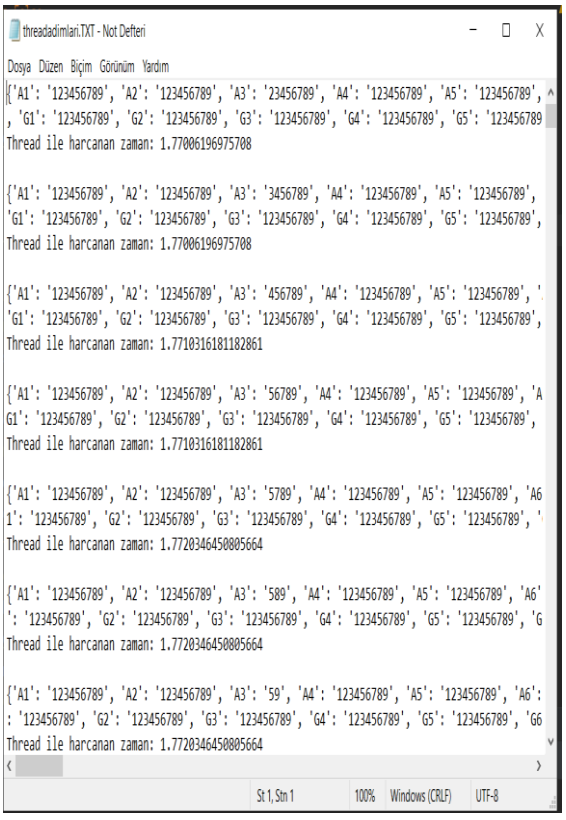
Resim 2: Açılış ekranı



Resim 4: Threadsiz çözüm konsolda gösterim

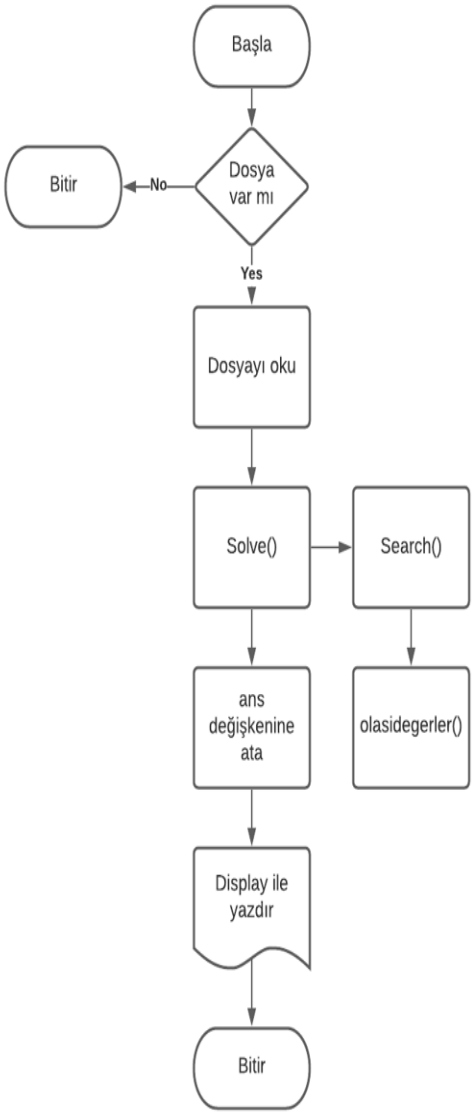


Resim 5: Threadli çözüm konsolda gösterim



Resim 6: Thread adımları txt uzantılı belge

V. Akış Diyagramı



VI. Yazılım mimarisi

Projenin genel yapısı verilen sudoku değerlerindeki boşlukları eleme yöntemi ile bulmaya dayanıyor. Bu yapı akış diyagramında da açıklanmıştır.

VII. Sonuç

Sudoku çözümü için algoritma araştırılmış ve uygulanmıştır. Thread kavramı anlaşılmış ve projeye uygulanması sağlanmaya çalışılmıştır.

VIII. Kaynakça

https://www.geeksforgeeks.org/how-to-create-buttons-in-a-game-using-pygame/

https://www.pygame.org/docs/ref/surface.html

https://ichi.pro/tr/yeni-baslayanlar-icin-pygame-38816435127663

https://en.wikipedia.org/wiki/Depth-first_search

https://www.codeproject.com/Articles/227435/Solving-Sudoku

https://oguzcelikarslan.medium.com/python-enumerate-kullan%C4%B1m%C4%B1-61e5c6997e25

https://www.mobilhanem.com/graph-teorisi-dfs-algoritmasi/

https://bilgisayarkavramlari.com/2008/11/13/derin-oncelikli-arama-depth-first-search/

https://python-ogren.readthedocs.io/en/latest/file_write.html

https://python-istihza.yazbel.com/temel_dosya_islemleri.html

https://www.halildurmus.com/2020/12/10/derin-oncelikli-arama-depth-first-search/