SQUARE PUZZLE GAME

Mehmet Oğuz Aktaş (150201170)

[oguzaktass@gmail.com](mailto:oguzaktass@gmail.com)

Kocaeli Üniversitesi

Mühendislik Fakültesi

Bilgisayar Mühendisliği Bölümü

Özet

*Projede istenen özellikleri bulunduran basit bir arayüz tasarladıktan sonra Resim Seç butonu ile JFileChooser classını kullanarak jpeg, png, gif, bmp uzantılı resim dosyalarının kullanıcı tarafından seçilebilmesini sağladım. Resim dosyasını yüklemek için BufferedImage tipinde loadImage() ve resmi yeniden boyutlandırmak için aynı tipteki resizeImage() metotlarını yazdım. BuildPuzzle() metodu içinde GridLayout classı, Collections.shuffle() ve createImage() hazır metotlarını kullanarak resmi bölüp her bir parçayı rastgele dağıtıp gösterme işlemlerini tamamladım. Projeye başlarken resim üzerinde yapılacak işlemler (resize, split) için Image ve BufferedImage tiplerinde değişkenler, resim boyutları ve puzzle parçalarının kontrolü için final değişkenler ve puzzle’ın bitirilip bitirilmediğini kontrol etmek için de boolean tipinde isSolved değişkenini kullandım.* ***Puzzle parçalarının doğru yerde olup olmadığını kontrol etmek için compareImages() metodu ile resimlerin pixel by pixel RGB değerlerini karşılaştırdım.*** *getHighestScore(), setScore() ve printScore() metotları ile de skor hesaplama ve yazdırma işlemlerini tamamladım. Her hamlede butonların yerlerini değiştirmek ve gerekli işlemleri yapabilmek için de ClickAction classını kullandım. Projede gereksiz kod tekrarları yapmamaya ve gereksiz yere bellekten alan ayırmamaya dikkat ettim.*

# Giriş

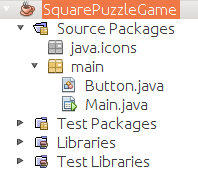
Bu projede, GUI ile resim dosyalarını seçtikten sonra resmi 16 parçaya bölüp parçaların her birini buton şeklinde rastgele dağıtarak bir puzzle oyunu tasarlamamız ve puzzle’ı çözerek her aşamada skor güncellemesi yapmamız istenmektedir. Ayrıca en az bir puzzle parçası doğru yere gelene kadar Karıştır butonuna basılması ve her hamlede skorun hesaplanarak puzzle tamamlandığında enyuksekskor.txt dosyasına yazılması da isteniyor. Puzzle parçalarının doğru yerde olup olmadığının kontrolü için ID karşılaştırması ve Point classı gibi yöntemlerle koordinat bilgilerinin kullanılması istenilmediği için öncelikle puzzle parçalarını nasıl karşılaştıracağım konusunu araştırdım. Geçen dönem Yazılım Lab I Proje 3’te uğraştığımız YUV Decoder projesinde resim dosyalarını tanımlamak için RGB değerlerini kullandığımız için aklıma ilk olarak RGB değerleriyle karşılaştırma işlemi geldi. Daha sonra projedeki diğer isterleri kısa sürede tamamlayıp bu aşamaya geldiğimde <https://stackoverflow.com/questions/15305037/java-compare-one-bufferedimage-to-another/15305092> adresindekine benzer comparaImages() isminde bir metot yazarak puzzle parçalarını karşılaştırma işlemini tamamladım. Projeyi geliştirirken en çok zamanımı alan kısım, her hamlede yani her tıklamada puzzle’ın tamamlanıp tamamlanmadığını kontrol edip butonların yerini değiştirmek ve gerekli skor bilgilerini güncellemek oldu. Bu işlemler için AbstractAction classından extend edilen ClickAction classını oluşturup, bu class içinde actionPerformed() metodunu override ederek her tıklamada yine bu class içinde tanımladığım changeButtons() metodunun çalışmasını sağladım. Tüm isterleri karşıladıktan sonra test işlemleri ve exception handling’ler ile son düzeltmeleri yaparak projeyi eksiksiz bir biçimde tamamladım.

# Temel Bilgiler

Uygulamayı geliştirmek yazmak için Java SE Development Kit 8u201 sürümüyle NetBeans 8.2 IDE’yi ve GUI için de Swing ve AWT API’lerini kullandım.

# Yazılım Mimarisi

Uygulamada Main isminde bir JFrame formu ve Button isminde bir Java classı olmak üzere 2 tane class kullandım.



*Şekil 1: Uygulamada bulunan classlar ve kütüphaneler*

Button classını buton işlemlerini daha düzenli yapabilmek için oluşturdum. Bu class içinde, butonların başlangıç özelliklerini ve her tıklamadaki border renklerini düzenlemek için oluşturduğum default Constructor içinde çalışan (yani her buton nesnesi oluştuğunda çalışacak olan) init() metodu ile buton resimleri için get ve set metotları bulunuyor.

Main classı içinde kullandığımız metotlardan bazıları şunlar;

* loadImage(): BufferedImage tipinde ve parametre almayan bu metot ile Resim Seç butonu ve JFileChooser ile alınan resim dosyası yolunun (filePath) BufferedImage ile resim dosyası olarak yüklenmesini sağladım.
* resizeImage(): BufferedImage tipinde olan ve 4 parametre (BufferedImage, width, height, type) alan bu metot ile width ve height değerlerine göre yeniden boyutlandırılmasını sağladım.
* buildPuzzle(): Uygulamadaki en önemli metot olan bu metotta yüklenen resmi GridLayout ile 16 parçaya bölüp her birini karıştırdıktan sonra buton şekline JPanel’e ekleme, butonları en az bir puzzle parçası doğru yerde olarak karıştırma ve bu adımlara göre skoru belirleme işlemlerini gerçekleştirdim.
* getHighestScore(): Bu metot ile enyuksekskor.txt dosyasını okuyup “Skor: {Sayi}” şeklindeki satırlardan en yüksek sayı değerini bularak GUI’deki highestScore ismindeki labele yazma işlemini yaptım.
* comparaImages(): Projenin en önemli amacı olan puzzle parçalarını RGB değerleri ile karşılaştırma işlemini boolean tipindeki bu metot ile sağladım. BufferedImage tipinde 2 resmi parametre olarak alarak width ve height değerlerine göre iç içe for döngüleri içinde resimlerin pixel by pixel RGB değerlerini karşılaştırdım.
* setScore(): Hamle başına -1 puan verilen ve her doğru yere gelen puzzle parçası için +6 puan verilen bir puan hesaplama sistemi oluşturup, bu metot ile her hamlede skoru hesapladıktan sonra score labeline yazma işlemini gerçekleştirdim. Ayrıca resim ilk yüklendiğinde tüm puzzle parçaları doğru yerde olursa 100 puan verilmesi ve ekrana JOptionPane mesajı yazılmasını bu metot ile sağladım. En az bir puzzle parçası doğru yere gelene kadar Karıştır butonuna basılmasının zorunlu tutulması işlemlerini de bu metot içinde score değişkenini kontrol ifadesi ile kullanarak yaptım.
* printScore(): Bu metot ile FileWriter classını kullanarak puzzle tamamlandıktan sonra score labelinden alınan skor değerinin “Skor: {Sayı}” biçiminde enyuksekskor.txt dosyasına yazılmasını sağladım.
* setPanelEnabled(): JPanel ve boolean tiplerinde 2 parametre alan bu metot ile JPanel’in ve içindeki buton şeklindeki puzzle parçalarının aktif edilip edilmeyeceğini ayarladım. Resim ilk yüklendiğinde eğer hiçbir puzzle parçası doğru yerde değilse JPanel’in aktifliğini kapatarak Karıştır butonuna basılmasını istendiği gibi zorunlu tuttum, bu işlemi setScore() metodu içinde sağladım. Ayrıca puzzle tamamlandıktan sonra da JPanel’in aktifliğini kapatma işlemini de bu metotu, changeButtons() metodu içinde isSolved değişkeni ile kullanarak sağladım.

Main classı içinde alt class olarak bulunan AbstractAction classından extend edilen ClickAction classı içindeki metotlar şunlar;

* actionPerformed(): ActionEvent parametresi alan bu hazır metodu override ederek her tıklamada yine bu metot içinde tanımladığım changeButtons() metodunun çalışmasını ve butonların yer değiştirmesini sağladım.
* changeButtons(): ActionEvent parametresi alan bu metodu yazarak, son tıklanan 2 butonun yer değiştirmesini, compareImages() metodu ile yer değiştirilen parçaların doğru yere gelip gelmediğinin kontrol edilmesini, doğru yere gelen her parça için +6 puan eklenmesini, her hamle başına -1 puan silinmesini ve her hamlede score labelinin güncellenmesi işlemlerini sağladım. Birçok işi tek bir metoda yüklemiş olsam da yapılan bu işlerin hepsi birbiriyle bağlantılı olduğu için ve sıralamasına dikkat edilmesi gerektiği için tüm bu işlemlerin tek bir metot içinde yapılmasını sağladım.
* updateButtons(): JPanel’deki butonların yenilenmesini ve yer değiştirmeler sonucunda görünümün güncellenmesini bu metot ile sağladım.

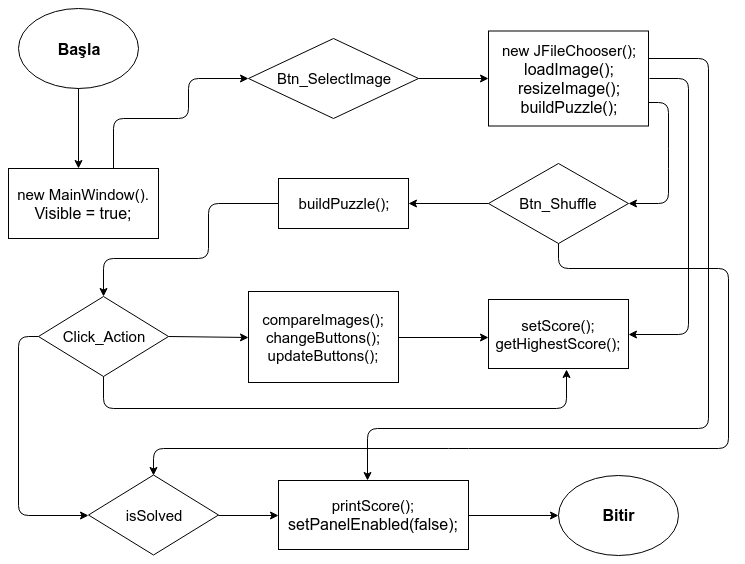
Main classı içinde kullandığım diğer metotlar şunlar;

* btn\_shuffleActionPerformed(): Bu hazır metot ile Karıştır butonuna (btn\_shuffle) basıldığında hangi kodların çalışacağını belirledim. Karıştır butonuna her basıldığında buildPuzzle() metodu çalışıyor.
* btn\_selectImageActionPerformed(): Bu hazır metot ile Resim Seç butonuna her basıldığında JFileChooser nesnesi oluşturup jpg, jpeg, png, gif, bmp uzantılı resim dosyalarının seçilmesi için dosya seçme panelinin açılmasını ve dosya seçme başarılıysa Karıştır butonunun aktif edilip score labeline 0 yazarak oyuna başlanmasını sağladım.

Butonları daha düzenli hale getirip kullanabilmek amacıyla oluşturduğum Button classı içindeki metotlar şunlar;

* init(): Her Button nesnesi oluşturulduğunda default Constructor metodu içinde çalışan bu metot ile butonların default border renklerini (gray), tıklandığındaki (red) ve mouse’un üzerine getirildiğindeki (yellow) border renklerini belirledim. Bu işlemleri mouseEntered(), mouseExited() ve mouseClicked() hazır metotlarını kullanarak sağladım.
* getImage(): BufferedImage tipindeki bu metot ile butondaki resme (puzzle parçasına) ulaşılmasını sağladım.
* setImage(): BufferedImage tipinde parametre alan bu metot ile butonun resminin belirlenmesini sağladım.

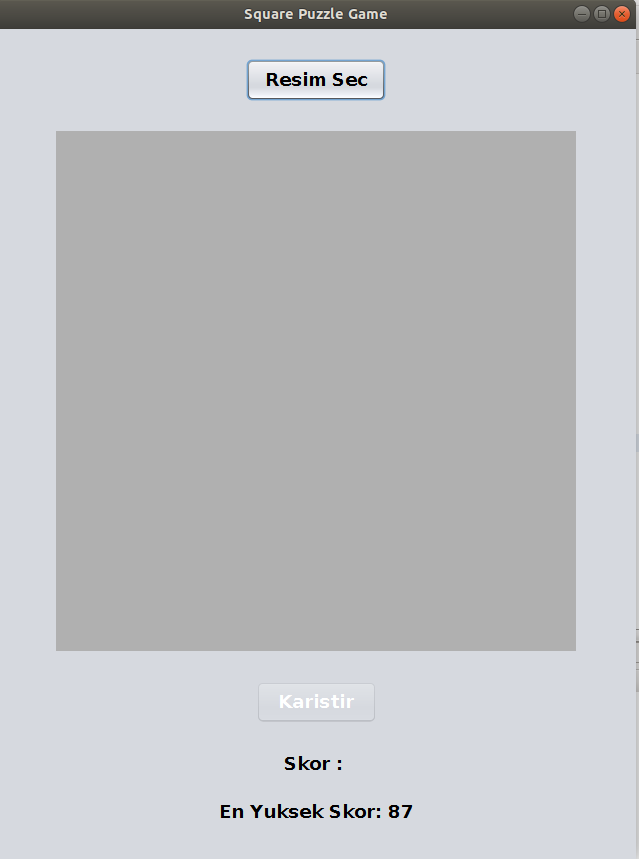
# Akış Şeması



*Şekil 2: Akış şeması*

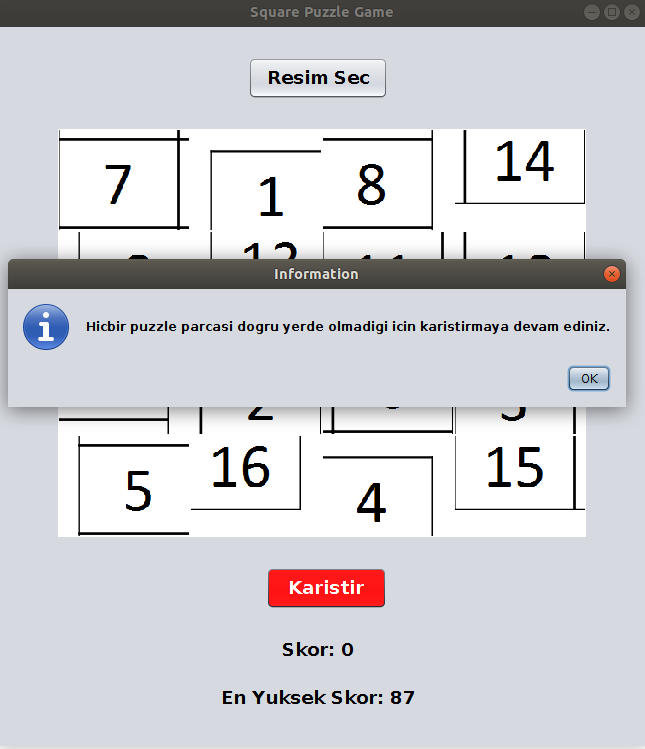
# Kullanıcı Kataloğu

Uygulamayı çalıştırınca ilk olarak Resim Seç butonunun ve boş bir JPanel ile aktif olmayan Karıştır butonunun butonunun bulunduğu ana ekran karşımıza geliyor.



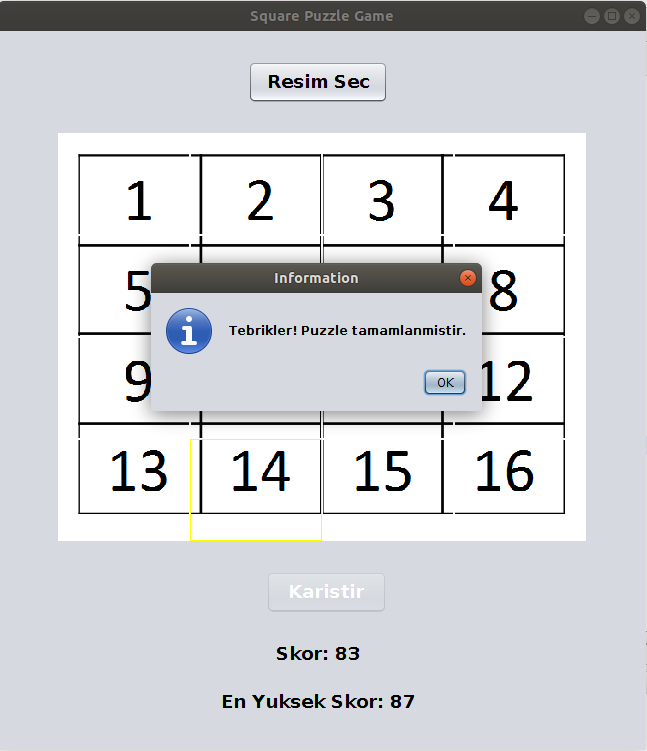
*Şekil 3: Programın ilk çalıştırılması ve resim seçme*

Resim seçildikten sonra Karıştır butonu aktif hale gelir fakat resmin tam halinin ekranda sadece puzzle tamamlandıktan sonra gösterilmesi istendiği için JPanel boş kalır. Karıştır butonuna basıldıktan sonra alttaki ekran gelir.



*Şekil 4: Resmin seçilmesi ve karıştırma işlemi*

En az bir puzzle parçası doğru yere gelene kadar karıştırılır. Daha sonra butonların üstüne tıklanıp hamle hamle yer değiştirmeleri sağlanır. En son puzzle tamamlandığında şu ekran karşınıza gelir;



Bilgi mesajını kapattıktan sonra JPanel ve butonların aktifliği kapanır. Daha sonra tekrar resim seçip yeni oyuna başlanabilir.

# Sonuçlar

Projeyi geliştirirken gereksiz kod tekrarları yapmamaya ve gereksiz yere bellekten alan ayırmamaya dikkat ettim. Her aşamada kod yazarken algoritma karmaşıklığını ve programın hızlı çalışmasını hesaba kattığım için yaptığım projenin yeterince özgün olduğunu ve istenenleri fazlasıyla karşıladığını söyleyebilirim. Puzzle parçalarının doğru yerde olup olmadığını, pixel by pixel RGB değerlerini karşılaştırarak kontrol ettim. Daha efektif ve hızlı sonuç verecek farklı görüntü işleme yöntemleri olup olmadığını pek araştırmadım, aklımda soru işareti oluşturan tek nokta bu oldu.

# Kaynakça

[1] <https://stackoverflow.com/>

[2] <https://docs.oracle.com/javase/8/docs/api/index.html?java/awt/image/BufferedImage.html> (Erişim tarihi: 15.03.2019)

[3] <http://zetcode.com/tutorials/javagamestutorial/puzzle/> (Erişim tarihi: 15.03.2019)

[4] <https://alvinalexander.com/blog/post/java/getting-rgb-values-for-each-pixel-in-image-using-java-bufferedi> (Erişim tarihi: 15.03.2019)

[5] <https://www.programcreek.com/java-api-examples/?class=java.awt.image.BufferedImage&method=TYPE_INT_ARGB> (Erişim tarihi: 15.03.2019)

[6] <https://www.javainuse.com/java/ImageEquals> (Erişim tarihi: 15.03.2019)

[7] <https://www.geeksforgeeks.org/image-processing-java-set-5-colored-red-green-blue-image-conversion/> (Erişim tarihi: 15.03.2019)

[8] <https://www.dyclassroom.com/image-processing-project/how-to-get-and-set-pixel-value-in-java> (Erişim tarihi: 15.03.2019)

[9] <https://www.journaldev.com/881/java-append-to-file> (Erişim tarihi: 16.03.2019)

[10] <https://www.geeksforgeeks.org/swapping-items-list-java-collections-swap/> (Erişim tarihi: 16.03.2019)

[11] <https://www.programcreek.com/java-api-examples/?class=javax.swing.JComponent&method=getClientProperty> (Erişim tarihi: 16.03.2019)