# Kare Puzzle Oyunu

Kocaeli Üniversitesi, Bilişim Sistemleri Mühendisliği Bölümü, Yazılım Geliştirme Labaratuvarı II, 2022-2023 Bahar, Proje I

Alperen Kuzhan
Kocaeli Üniversitesi
Bilişim Sistemleri Mühendisliği
191307031
kuzhanalperen@gmail.com

Hayri Batuhan Aral
Kocaeli Üniversitesi
Bilişim Sistemleri Mühendisliği
191307033
batuhanaral3@gmail.com

Hüseyin Oğuz Çetinkaya
Kocaeli Üniversitesi
Bilişim Sistemleri Mühendisliği
191307003
huseyinoguzc@gmail.com

***Özet*—Bu proje kapsamında C# masaüstü uygulaması kullanılarak kullanıcın yüklediği resmi Bitmap formatında 16 parçaya bölerek bağlı listede ile saklandı. Sonrasında kullanıcının seçeceği iki parçanın yerleri tıklanan butonlar aracılığıyla değiştirildi, kullanıcıya doğru işlem yaptıysa +5 yanlış işlem yaptıysa -10 puan vererek skoru hesaplandı. Bu sayede kullanıcı oyundaki başarısını görüp kendini test edebilecektir.**

***Anahtar kelimeler—puzzle, oyun, csharp, form, linkedlist***

## I. GİRİŞ

Puzzle oyunları, yüzyıllardır insanların ilgisini çeken ve zamanla farklı formlarda geliştirilen zekâ oyunlarıdır. Günümüzde, dijital çağın hızla gelişmesiyle birlikte, masaüstü bilgisayarlar için de birçok puzzle oyunu geliştirilmiştir. Bu oyunlar, kullanıcıların beyin jimnastiği yapmasını sağlayan ve aynı zamanda keyifli vakit geçirmesini sağlayan bir türdür.

Masaüstü puzzle oyunları, bilgisayarın donanım ve yazılım özelliklerini kullanarak farklı zorluk seviyelerinde puzzleları kullanıcılara sunmaktadır. Oyunlar genellikle tek kişilik oynanmakta ve kullanıcıların strateji geliştirmelerine, zekâlarını kullanmalarına ve problem çözme becerilerini geliştirmelerine yardımcı olmaktadır. Ayrıca, puzzle oyunları, farklı temalar ve görsel efektler kullanarak kullanıcıların oyun deneyimini zenginleştirmekte ve daha eğlenceli hale getirmektedir.

Bu proje kapsamında, bir puzzle oyunu geliştirilmesi hedeflenmektedir. Oyun, kullanıcıların seçmiş olduğu resmi puzzle olarak sunacak, kullanıcıların zekâlarını ve problem çözme becerilerini geliştirmelerine yardımcı olacak ve keyifli vakit geçirmelerini sağlayacaktır. Ayrıca, oyunun görsel tasarımı ve kullanıcı arayüzü de önemli bir konudur. Kullanıcıların oyunu kolaylıkla kullanmaları ve deneyimlerinin daha keyifli hale gelmesi için görsel tasarımın da dikkatli bir şekilde düşünülmesi gerekmektedir.

## II. PROBLEM TANIMI

Bu proje, kullanıcıların bir görseli 16 parçaya bölüp, bu parçaları doğru yerlerine yerleştirerek tamamlamaları amaçlanmaktadır. Kullanıcılar, bir resim dosyası yükleyerek veya URL üzerinden resim seçerek başlayacaklardır. Daha sonra, resim 4 satır ve 4 sütun olacak şekilde 16 parçaya ayrılacaktır.

Proje, C# form kullanılarak geliştirilmiştir ve geliştirme aşamalarında GitHub platformu kullanılmıştır. Projenin GUI'sine Karıştır butonu eklenerek, kullanıcılar parçaları karıştırabileceklerdir. Karıştırma işlemi devam edecektir, ta ki en az bir puzzle parçası doğru yerine yerleştirilene kadar.

Kullanıcı tarafından doğru yerleştirilen parçaların yer değiştirmesini engellemek için bu butonlar kilitlenecektir. Parçaların doğru yerde olup olmadığının kontrolünde, ID veya koordinat bilgileri kullanılmayacaktır. Bunun yerine, 0-15 puzzle parça numaraları için bir bağlı liste oluşturulacak ve parçaların konumu bu bağlı liste kullanılarak kontrol edilecektir.

Proje, en yüksek skor bilgisi içeren bir arayüzle birlikte gelir. Kullanıcılar, her doğru hamle için 5 puan kazanacak ve her yanlış hamle için 10 puan kaybedeceklerdir. Her hamleden sonra, kullanıcı puan durumunu arayüz üzerinden görebilir. Ayrıca, en yüksek skor bilgisi 'enyuksekskor.txt' belgesine yazılacaktır. Bu belgede, kullanıcı adı, hamle sayısı ve puan bilgileri yer alacaktır. Oyun boyunca kullanıcının hamle sayısı kaydedilecek ve her oyun açıldığında, tasarlanan arayüzün bir kısmında en yüksek puanlar azalan sırada gösterilecektir.

## III. GELİŞTİRME ORTAMI

Bu uygulama geliştirilirken C# programlama dili ve Visual Studio geliştirme ortamı kullanılacaktır.

C#, Microsoft tarafından geliştirilmiş, nesne yönelimli, modern bir programlama dilidir. C#, hem Windows uygulamaları hem de web uygulamaları geliştirmek için kullanılan yaygın bir dildir. Ayrıca, C# ile birlikte Visual Studio gibi gelişmiş bir entegre geliştirme ortamı kullanarak hızlı bir şekilde uygulama geliştirebiliriz.

Masaüstü oyunları gibi performans gerektiren uygulamaları geliştirmek için de C# tercih edilebilir. C#, Java gibi bir sanal makine üzerinde çalışır ve sıkı bir derleme zamanı kontrolüne sahip olduğu için performansı da oldukça iyidir. Ayrıca, C#'ın zengin standart kütüphaneleri sayesinde, kullanıcı arayüzü oluşturma, veri tabanı bağlantısı kurma gibi işlemler de kolayca yapılabilir.

Bu nedenlerle, biz de C# dili ve Visual Studio geliştirme ortamını kullanarak masaüstü puzzle oyunumuzu geliştirdik.

## IV. UYGULAMA GELİŞTİRME AŞAMASI

Kullanıcının seçtiği fotoğraf Open File Dialog nesnesi kullanarak yüklenecektir. Yüklenen bu fotoğraf bitmap fomatında saklanacaktır. Kullanıcı tarafından seçilen bu fotoğraf daha sonra 16 parçaya bölünecektir. Bölünen her bir parça daha önce tanımlamış olduğumuz Image sınıfı türünden bağlı listede tutulacaktır ardından fotoğraf parçalarından en az bir tanesi doğru yerde olana kadar kullanıcı karıştırma işlemi yapacaktır. Fotoğraflardan en az bir tanesi doğru yerde olduğunda karıştır butonu disable olup bir daha karıştırılmayacaktır.

Kullanıcının hamlelerini yapması için fare işlemleri kullanılacaktır. Kullanıcının bir parçayı seçmesi için fare tıklama işlemi, seçili olan parçanın yer değiştirmesi için ilk tıklanan parça ile ikinci tıklayacağı parçanın yer değiştirme işlemi kullanılacaktır. Parça yerleştirildikten sonra doğru yerleştirme kontrolü yapılacak ve puanlama işlemi gerçekleştirilecektir. Yapılan puanlamalar ise başka kullanıcılar ile kıyaslanıp uygulamada gösterilecektir. Aynı zamanda kullanıcı skorları .txt dosyasında saklanıp kullanılacaktır.

*A. Fotoğraf Yüklenmesi*

C# masaüstü uygulamalarında OpenFileDialog sınıfı, kullanıcının dosya seçmesine olanak tanıyan bir iletişim kutusu sağlar. Bu iletişim kutusunu kullanarak kullanıcının seçtiği dosya yolu, uygulama tarafından kullanılabilir.

Bu uygulamaya bir PictureBox nesnesi ekleyerek ve OpenFileDialog sınıfını kullanarak resim yükleme işlemi gerçekleştiriyoruz. Kullanıcı PictureBox nesnesine tıkladığında, OpenFileDialog penceresi açılır ve kullanıcı bir resim dosyası seçer. Seçilen dosya, PictureBox kontrolüne yüklenir ve masaüstü uygulamasında görüntülenir.

Bu yöntem, uygulamanın daha interaktif hale getirilmesine yardımcı olabilir ve kullanıcıların uygulamaya kendi resimlerini yüklemesine olanak tanır.

*B. Parçalama Fonksiyonu*

Bu kısım, bir resmi parçalara ayırmak ve bu parçaları 16 adet butona atamak için kullanılıyor. Kod, öncelikle "original" adlı bir nesnenin görüntüsünün var olup olmadığını kontrol ediyor. Eğer varsa, "linkedList" adlı bağlı liste öğesini temizliyor.

Daha sonra, orijinal görüntü "resim" adlı bir Bitmap nesnesine kopyalanıyor ve 4x4 parçalara ayrılıyor. Bu işlem, resmin boyutunu 400x400 piksel olarak belirliyor ve her parça 100x100 piksel boyutunda olacak şekilde oluşturuluyor.

Bağlı listede, her bir parça "linkedList" adlı değişkenin sonuna ekleniyor. Bu işlem, listedeki parçaların sıralı olarak saklanmasını sağlar.

Son olarak, "current" adlı bir "LinkedListNode<Image>" değişkeni, bağlı listenin ilk düğümüne atanıyor. Daha sonra, her butonun Image özelliği, listedeki sıradaki parçanın Image değeri ile değiştiriliyor. Bu işlem, butonların sırayla düzenlenmesini sağlar.

Bu fonksiyon, bir görüntünün parçalara ayrılmasını sağlamak için oldukça kullanışlıdır ve grafiksel kullanıcı arayüzleri geliştirmek için sıklıkla kullanılır.

*C. Karıştırma Fonksiyonu*

Uygulamanın bu kısmında puzzle oyununu karıştırmak için bir metod oluşturuldu. Metodun adı "karistir"dir ve işlevi, resim parçalarını karıştırarak oyuncuya rastgele bir sırayla sunmaktır.

Metod, öncelikle "original" adlı bir resim nesnesinin yüklenip yüklenmediğini kontrol eder. Eğer resim yüklü ise, "mixedLinkedList" adlı bir bağlı listede yer alan tüm elemanları temizler. Bu bağlı listede her bir eleman, resmin bir parçasını temsil eder.

Daha sonra, "copyLinkedList" adlı yeni bir bağlı liste oluşturulur ve "linkedList" adlı önceden tanımlanmış bağlı listenin tüm elemanları buraya kopyalanır. Bu işlem, parçaların karıştırılması sırasında orijinal listedeki elemanların bozulmasını engeller.

Metod, "for" döngüsü aracılığıyla 16 parça olduğu için 16 kez çalışır. Her bir döngü adımında 0 ile “copyLinkedList” listesinin eleman sayısı arasında rastgele bir sayı oluşturulur ve bu sayı, "copyLinkedList" listesindeki bir elemanın indeksini belirler. Daha sonra, bu eleman, "mixedLinkedList" listesine eklenir ve "copyLinkedList" listesinden çıkarılır. Bu sayede, her bir parça yalnızca bir kez karıştırılır.

Karıştırma işlemi tamamlandıktan sonra, "while" döngüsü aracılığıyla "mixedLinkedList" ve "linkedList" listeleri karşılaştırılır. Eğer bu iki liste arasında en az bir eleman aynıysa, karıştırma ve resim yükleme butonları devre dışı bırakılır ve her bir parça için oluşturulan butonlar aktif hale gelir.

Son olarak, "for" döngüsü aracılığıyla her bir butona yeni bir resim parçası atanır ve karıştırma işlemi tamamlanmış olur ve oyun kullanıcının oynayabileceği hale gelmiş olur.

*D. Yer Değiştirme İşlemi*

Oyuncunun resmin parçalarını orijinal resimdeki doğru yerlerine yerleştirmesi gerekmektedir.

FirstImage değişkeni, ilk seçilen butonun resim değerini, firstButton değişkeni ise ilk seçilen butonun kendisini saklar. secondButton değişkeni, ikinci seçilen butonun kendisini saklar. hamleSayac değişkeni, oyuncunun yaptığı hamle sayısını, skorSayac değişkeni ise oyuncunun skorunu tutar. eslesmeSayac değişkeni, doğru yerdeki parçaların sayısını saklar.

button1_Click fonksiyonu, bir butona tıklandığında çalışır. Bu buton, oyun sırasında tıklanan butonlardan biridir. Fonksiyonun amacı, tıklanan butonun işlevselliğini yerine getirmek ve gerekli kontrolleri yapmaktır.

İlk olarak, currentButton değişkenine, tıklanan butonun kendisi atanır. firstImage değişkeni boşsa, yani ilk buton seçiliyorsa, firstImage değişkenine tıklanan butonun resim değeri atanır ve firstButton değişkenine tıklanan butonun kendisi atanır.

Eğer firstImage değişkeni doluysa, yani ikinci buton seçiliyorsa, hamleSayac değişkeni bir arttırılır ve hamle sayacı etiketindeki yazı güncellenir. Daha sonra secondButton değişkenine tıklanan buton atanır.

Karışık listenin node1 ve node2 adlı değişkenlerle bulunması için mixedLinkedList listesinde bulunan düğümler arasında Find() fonksiyonu kullanılır. Find() fonksiyonu, belirtilen değere sahip olan ilk düğümü bulur ve bu düğümü döndürür. Swap() fonksiyonu, bulunan düğümlerin yerlerini değiştirir.

Karışık listenin düğümleri, current adlı yeni bir düğüme atılır ve for döngüsü içinde bu düğümler, resimlerini içeren butonlara atanır.

Daha sonra düğümler "mixedLinkedList" ve "linkedList" karşılaştırılarak tek tek bulunur ve doğru yerdeki düğümlerin butonları pasif hale getirilir. Bu işlem doğru hamle yapıldığında gerçekleşir ve "dogruHamleMi" değişkeni true olarak ayarlanır.

Eğer yanlış bir hamle yapılmışsa, "dogruHamleMi" değişkeni false olarak kalacaktır ve skorSayac değişkeninden 10 puan çıkarılacaktır.

Son olarak, tüm düğümler doğru yerlerine yerleştirilmişse, oyun biter ve bir mesaj kutusu görüntülenir. Bu mesaj kutusu, kullanıcının kaç hamle yaptığını ve kaç

puan kazandığını gösterir. Ayrıca, tüm butonlar tekrar etkin hale getirilir ve kullanıcının yeniden oynaması için bir resim seçmesi istenir.

*E. Swap Fonksiyonu*

Bu kısım, LinkedList sınıfı ile oluşturulan bağlı listedeki iki düğümün yerlerini değiştirmek için kullanılır.

Öncelikle, Swap metodu, üç parametre alır: bir tane LinkedList<Image> nesnesi ve iki tane LinkedListNode<Image> düğümü. Eğer herhangi bir parametre null ise, metot hiçbir işlem yapmadan geri döner.

Daha sonra, metot, parametreden gelen düğüm1 ve düğüm2 değerlerinin aynı olup olmadığını kontrol eder. Eğer aynıysalar, hiçbir işlem yapmadan geri döner.

Ardından, düğüm1 ve düğüm2 değerlerini geçici değişkenlerde depolar. Daha sonra, düğüm1'in değerini düğüm2'nin değeri ile değiştirir ve düğüm2'nin değerini de geçici değişkendeki değerle değiştirir. Bu işlem, iki düğümün değerlerini değiştirmiş olur.

Bu şekilde, Swap metodu, bağlı listedeki iki düğümün yerlerini değiştirmek için kullanılır.

*F. Dosya İşlemleri*

*1) Dosya Oluşturma:*

Bu kısım, belirtilen dosya yolunda bir dosya yoksa, yeni bir dosya oluşturarak içerisine belirtilen ilk satırı yazdırır.

İlk olarak, File.Exists() metodunun kullanımı ile belirtilen dosya yolu üzerinde bir dosya olup olmadığı kontrol edilir. Eğer dosya yolu üzerinde bir dosya yoksa, kod bloğu if koşulu altında çalışır.

"using" bloğu içerisinde StreamWriter sınıfından bir nesne oluşturulur. Bu nesne, belirtilen dosya yolunda yeni bir dosya oluşturmak için kullanılır. File.CreateText() metodu, belirtilen dosya yolu üzerinde yeni bir dosya oluşturur ve StreamWriter sınıfından bir nesne döndürür.

StreamWriter nesnesi, dosyaya yazmak için WriteLine() metodu kullanarak belirtilen ilk satırı yazdırır. Bu örnekte, "Ad,Hamle,Puan" değeri dosyaya yazılmaktadır.

"using" bloğu sona erdiğinde, StreamWriter nesnesi kapatılır ve kaynaklar serbest bırakılır. Bu şekilde, dosya işlemi güvenli bir şekilde tamamlanmış olur.

Bu kod parçası, dosya oluşturma ve dosyaya yazma işlemlerinde hata almamak için oldukça kullanışlıdır. Eğer belirtilen dosya yolunda bir dosya yoksa, yeni bir dosya oluşturarak içerisine belirtilen ilk satırı yazdırmak suretiyle, dosya işlemleri sırasında olası hataların önüne geçilmiş olur.

*2) Dosyaya Yazma:*

Bu kısım, dosyaya yazma işlemi yapmak için kullanılır. hamleSayac ve skorSayac gibi iki parametre alır ve bu parametrelerle birlikte kullanıcı adı değerini kullanarak bir satır oluşturur. Oluşturulan bu satır daha sonra dosyaya yazılır.

İlk olarak, string tipinde satır değişkeni tanımlanır ve kullaniciAdi, hamleSayac ve skorSayac değişkenlerinin değerleri kullanılarak bir metin oluşturulur. Bu metin, virgülle ayrılmış üç alan içerir: kullanıcının adı, hamle sayısı ve skor sayısıdır. Bu üç alan dosyadaki sütün adlarımıza karşılık gelmektedir.

Daha sonra, StreamWriter sınıfından bir nesne oluşturulur ve File.AppendText() metodu kullanılarak dosya açılır. Bu metot, belirtilen dosya yolunda mevcut dosyanın sonuna ekleme yapmak için kullanılır. StreamWriter nesnesi, dosyaya yazmak için WriteLine() metodu kullanarak satır değişkeninde oluşturulan metni yazdırır.

Bu kısım, bir dosya içerisine kullanıcının adı, hamle sayısı ve skor sayısı gibi verilerin yazdırılması işlemini gerçekleştirir. Bu veriler, sonrasında dosyadan okunarak istatistiksel işlemler yapılabilir veya kaydedilmiş kullanıcı verileri gösterilebilir.

*3) Dosyadaki Kayıtları Listeleme:*

Bu kısım, bir dosyanın içindeki verileri okuyarak belirli bir sütuna göre sıralayarak listelemeyi ve sonrasında sadece belirli bir sayıda en yüksek puanlı verileri listelemeyi sağlar.

İlk olarak, listBox1 öğesi için Clear() metodunun kullanımı, önceki listeleme işleminde listbox öğesine eklenmiş verileri temizler.

Daha sonra, bir List<string> nesnesi olan kayıtlar tanımlanır, burada tutulan veriler listelenecek.

StreamReader sınıfından bir nesne oluşturularak dosya okunur. Daha sonra, dosyadaki her satır ReadLine() metodu ile okunur ve satır değişkenine atanır.

Okunan her satır, virgüllerle ayrılmış sütunlara bölünür ve bu sütunlar sütun adlı bir string dizisine atanır. Ardından, if koşulu kullanılarak başlık sütunlarını (Ad, Hamle, Puan) almamak için ilk satır kontrol edilir ve kayıtlar listesine sadece isim ve puan eklenir.

Kayıtlar listesi, Sort metodu ile puan sütuna göre küçükten büyüğe sıralanır ve Reverse metodu kullanılarak büyükten küçüğe çevrilir. Bu işlemler sonrasında, listedeki en yüksek puanlara sahip verilerin en üstte olduğu bir liste oluşur.

foreach döngüsü ile kayıtlar listesi taranır ve sadece en yüksek 10 puanlı veriler listbox öğesine eklenir. sayaç değişkeni, listbox öğesine eklenecek veri sayısını belirlemek için kullanılır.

Bu kod parçası, bir dosyadaki verileri okuyarak, istenilen sütuna göre sıralayarak, en yüksek puanlı verileri listelemek için kullanılır. Listelenen veriler, örneğin oyun skorları veya yarışma sonuçları gibi, herhangi bir ölçüt veya değere göre sıralanabilir ve daha sonra listelenen veriler farklı yöntemlerle kullanılabilir.

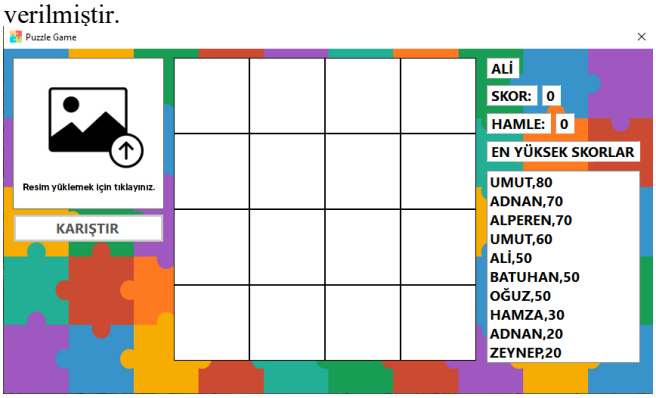
## V. UYGULAMANIN TANITIMI VE KULLANIMI

Uygulamaya ilk girildiğinde kullanıcıdan skoru ve hamle sayısıyla birlikte kaydedilmek üzere bir kullanıcı adı (Şekil 1) istenir. Burada kullanıcı boş bir karakter giremez. Bunun kontrolü yapılmıştır. Kullanıcı adı girilip butona veya "enter" tuşuna tıklandığı anda kullanıcıyı oyun ekranı (Şekil 2)

karşılayacaktır.

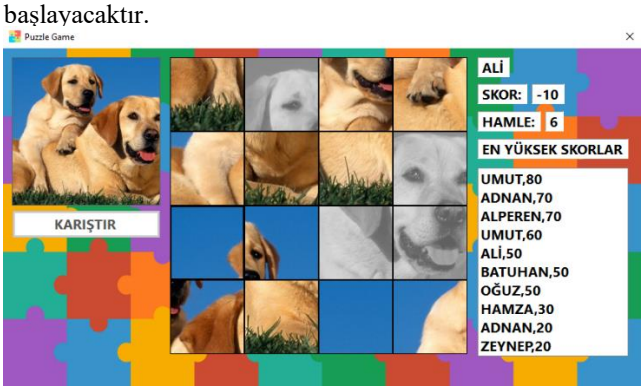

*Şekil 1*

Kullanıcı adı girişi yapılıp butona tıklandıktan sonra kullanıcıyı karşılayacak olan ekranda (Şekil 2) puzzle oyununu oynamak istediği resmi seçeceği bir alan bulunmaktadır. Resim yüklendiği anda aktif olacak olan ve oyuna başlamak için gerekli olan karıştırma işlemini en az bir parça doğru yerde olana dek sürdürecek olan bir karıştır butonu bulunmaktadır. Karıştırma işlemi bittikten sonra buton pasif hale gelecektir. Orta kısımda henüz resim yüklenmediği için puzzle parçaları boş bir şekilde bulunmaktadır. Kullanıcının resim yüklemesi halinde bu butonlarda ilgili resmin parçaları sıralı bir şekilde bulunacaktır. En sağ tarafta ise kullanıcının giriş yapmış olduğu kullanıcı adı, oyun süresince elde ettiği skor ve bu skoru elde ederken yaptığı hamle sayısı güncellenerek gösterilmektedir. Bu kısmın hemen altında ise oyunda en yüksek skoru elde etmiş ilk 10 oyuncunun kullanıcı isimleri ve skorları azalan bir sırada verilmiştir.



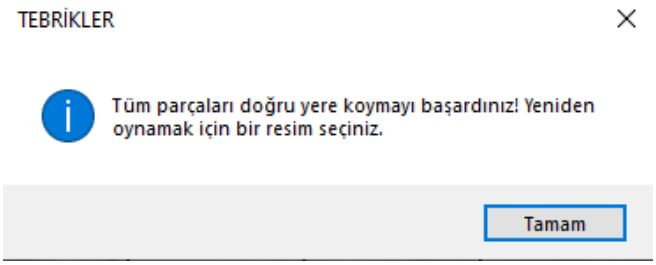

*Şekil 2*

Oyuncunun resim yükleme alanına tıklayıp bir resim seçmesinin ardından karıştır butonu aktif olacak ve en az bir parça doğru yerde olana kadar aktif olmayı sürdürecek, oyun başlamayacaktır. En az bir parça doğru yerde olduktan sonra puzzle parça butonları aktif olacak (Şekil 3) ve oyun başlayacaktır.




*Şekil 3*

Tüm parçalar doğru yerde olduktan sonra oyuncuya bir mesaj kutusu gelir (Şekil 4) ve oyuncuyu bilgilendirir.




*Şekil 4*

## VI. KARŞILAŞTIĞIMIZ ZORLUKLAR VE BİZE KATTIKLARI

Uygulamayı geliştirmeye ilk başladığımızda isterleri ve problemi iyi bir şekilde kavramaya çalıştık. Ardından problemleri parçalara bölüp önce bağlı listeler ile swap işleminin nasıl yapılacağı, formda butonların yerlerinin nasıl değiştirileceği, bir fotoğrafın nasıl bölüneceği, dosyaya yazdırma ve dosyadan okuma vb. işlemleri ayrı ayrı gerçekleştirdik. Yapmamız gereken işlemleri ayrı ayrı gerçekleştirip anladıktan sonra uygulamamızın son halini alacak şeklini geliştirmeye başladık.

Yapacağımız uygulamayı önce bağlı liste veri yapısı ile entegre etmeye çalıştık. Bölünen her bir parçayı bağlı liste veri yapısının her bir düğümüne veri olarak gömdük. Bunu yaparken bağlı liste veri yapısını Image sınıfı türünden üretip Image sınıfı türünden bir bağlı liste veri yapısı ile ilk defa çalışmış olduk. Daha önce veri yapıları ile sadece basit uygulamalar yapıp veri olarak düğümlerin içinde sadece sayı saklamıştık fakat resim verisi saklamak gibi komplike bir uygulamayı ilk defa deneyimledik.

Karıştırma işlemine kadar sorunsuz bir şekilde ilerlerken uygulama karmaşıklaşmaya ve isterleri uygulamaya çalışmaya başladıkça zorlanmaya başladık. Karıştırma işlemi yaparken kontrol için kullanılacak olan sıralı bağlı listeyi silmek zorunda kalıyorduk. Bu uygulama için bağlı liste veri yapılarını sadece bir kez değil birden fazla kullanmamız gerektiğini anladık. Sıralı olan bağlı listeyi kopyalayıp karıştırma işlemi yaparken bu listenin düğümlerini sildik ve parçaların sıralı değil de karmaşık olarak bulunacağı yeni bir karmaşık listeye aktardık. Uygulama boyunca kullanıcının üzerinde değişiklik yapacağı karmaşık bir liste ve konum doğruluğunun tespiti için bir tane de düzenli liste elimizde olacak. Sistem genel olarak bu iki liste üzerine kurulu olacak.

Oyun içinde çok sefer fotoğraf seçilmesine rağmen her zaman ilk seçilen fotoğraf kalması problemi ile karşılaştık, bunun sebebi yeni eklenen resim parçalarının sürekli bağlı listenin sonuna eklemeye devam edilmesiydi. Bu sayede Clear metodunu ve listeyi her bir yeni seçimde temizlememiz gerektiğinin farkına vardık. Bu sayede proje süresince Clear fonksiyonunu pek çok kez kullandık.

Uygulamanın pek çok kısmı tamamlandıktan sonra sıra yerleri değiştirilen parçaların konum doğruluğunu kontrol etmeye geldiğinde öncelikle bu kontrolü butonların index numarasına göre gerçekleştirdik. Uygulama her ne kadar doğru bir şekilde gerçekleşse bile pek çok kontrol yapmak zorunda kaldık. Özellikle yeri değiştirilen iki parça da doğru yerdeyse bunu iki kez kontrol etmemiz gerekiyordu. Her bir butonu tek tek gezip sıralı listede doğru yerde bulunan düğüm ile eşleşiyorsa ilgili butonu pasif hale getirdik ancak proje

isterlerine göre projede index, ID veya parçaların koordinat bilgileri parçaların konum doğruluğunu tespit etmekte kullanılmayacağı için uzunca bir süre bunu farklı bir şekilde nasıl yapacağımız üzerine düşündük.

Elimizde karışık ve sıralı olmak üzere iki liste bulunduğu için swap fonksiyonu ile kullanıcının butonlar üzerinden seçtiği her bir resim değerini düğümler içinden Find metodu ile bulup karışık listede kullanıcının yerlerini değiştirmek istediği düğümleri yer değiştirdik ardından kullanıcının düzenlemiş olduğu karışık listeyi yine butonlara tek tek atadık. Bu sırada da karışık listenin düğümleri ile sıralı listenin düğümlerini tek tek karşılaştırdık. Oyuncu her bir hamle sırasında karışık listeyi düzenleyecektir. Oyun sonunda karışık liste ile sıralı liste birbirine eşit olana kadar oyunu devam ettirdik ve doğru yerde bulunan butonları pasif hale getirdik. Oyunumuzu bu mantık üzerinde inşa ettik ve parçaların konum doğruluğunu tamamen bağlı listeleri kullanarak gerçekleştirdik.

Karışık listedeki düğümlerin yerlerini değiştirmek için Swap fonksiyonu oluşturduk. Bun yaparken C# LinkedList sınıfının dahili olarak bir Swap fonksiyonu olmadığı için Swap kodunu kendimiz yazdık. Öncelikle LinkedList sınıfını genişletmeyi düşündük fakat bunu başaramadığımız için oluşturulan listenin düğümlerinin gönderileceği uygulamayı geliştirmiş olduğumuz ana sınıfın içerisine bir Swap fonksiyonu oluşturduk. Uzunca bir süre Swap fonksiyonu üzerinde ve karışık listenin güncellenmesi üzerinde duruldu.

Konum doğruluğu kontrolü bağlı listeler ile yapıldıktan sonra bize pek çok kolaylık sağladı. Sadece tek bir düzgün çalışan kod yazdıktan sonra tüm parçaların konumunun kontrolünü tek seferde yapabilmiş olduk. Oysa ki butonların index kontrolü yapılırken birden fazla kez kontrol yapmamız gerekiyordu. Örneğin tek hamlede iki parça için de doğru yer bulununca buton index ile yapılan kontrol yetersiz kalıyordu ve ekstra kod yazmamız gerekiyordu. Bağlı listelerin sağladığı kolaylık kısmında karşımıza bu gibi örnekler çıkıyor.

Puan artırma ve azaltma işlemlerinde tek seferde listelerin her bir düğümü kontrol edildiği için tek hamlede çok kez puan verme ve puan kırma gibi bir sorunla karşılaşıyorduk. Puan işlemlerinin kontrolü için pek çok yöntem denedik. En sonunda yanlış işlemler için bayrak kullandık. Bu sayede her bir yanlış hamle için sadece bir kez puan kırmıştık olduk. Puan verirken ise her bir doğru yerde olan düğüm için tekrar puan veriyorduk. Oysa istenen her bir doğru hamlede puan vermektir. Bunun sorunun önüne butonları kullanarak geçmiş olduk. Her bir butonun pasif olup olmadığını kontrol edip ekstra bir koşul bloğu daha yazdık. Bu sayede her bir pasif olan buton için tekrar puan vermemiş olduk.

Genel olarak karşılaştığımız zorluklar ve çözümleri bu şekilde sıralanabilir. Bağlı listeler ile böyle komplike bir uygulamayı ilk kez gerçekleştirmek bize pek çok şey kattı. Özellikle bağlı listelerin kontrol mekanizmalarında kullanmanın

- Veri erişiminde hız: Bağlı listeler, elemanların belirli bir sırayla tutulmadığı bir veri yapısıdır. Bu nedenle, belirli bir elemana erişmek için tüm listenin taranması gerekmez. Bunun yerine, sadece hedef elemanın yerini bilmek yeterlidir. Bu, veri erişim hızını artırır.
- Veri ekleme ve silmede etkinlik: Bağlı listeler, eleman ekleme ve silme işlemleri için ideal bir veri yapısıdır. Yeni bir eleman eklemek için sadece o elemanın önceki elemana bağlanması yeterlidir. Eleman silmek için de sadece önceki elemanın bağlantısının kırılması yeterlidir. Bu nedenle, bağlı listeler veri ekleme ve silme işlemlerinde daha etkili ve verimlidir.
- Esneklik: Bağlı listeler, herhangi bir veri tipi için kullanılabilir ve boyutları dinamik olarak ayarlanabilir. Bu, verileri farklı şekillerde organize etmek ve depolamak için esneklik sağlar.
- Veri yapıları arası uyumluluk: Bağlı listeler, diğer veri yapıları ile birleştirilebilir veya diğer veri yapılarından türetilmiş olabilir. Bu, veri yapıları arasında veri aktarımı ve paylaşımı için uygun bir yoldur.
- Düşük bellek kullanımı: Bağlı listeler, elemanları bir arada tutmak için tek bir bellek bloğu kullanmak yerine, elemanların her biri için ayrı bir bellek bloğu kullanır. Bu, bellek kullanımını optimize eder ve düşük bellek kapasitesine sahip cihazlar için idealdir.

gibi faydalarını öğrenmiş olduk.

## KAYNAKLAR


[1] stackoverflow.com

[2] btkakademi.gov.tr/portal/course/algoritma-ve-veri-yapilari-ileri-seviye-17824

[3] youtube.com/@ErkanUzunK

[4] geeksforgeeks.org

[5] learn.microsoft.com/tr-tr/dotnet/csharp/

[6] github.com

[7] youtube.com/@BarsAslan