DEĞİŞKEN EKRAN ÇÖZÜNÜRLÜKLERİ İÇİN GELİŞMİŞ BİR YERLEŞTİRME YÖNETİCİSİ

AN ADVANCED LAYOUT MANAGER FOR VARIABLE USER INTERFACES

GÖZDE BUSE ÖCAL

DOÇ. DR. BURKAY GENÇ

Tez Danışmanı

Hacettepe Üniversitesi Bilişim Enstitüsü

Bilgisayar Grafiği

Anabilim Dalı için Öngördüğü

Yüksek Lisans Tezi olarak hazırlanmıştır.

Mayıs 2022

ÖZET

DEĞİŞKEN EKRAN ÇÖZÜNÜRLÜKLERİ İÇİN GELİŞMİŞ BİR YERLEŞTİRME YÖNETİCİSİ

Gözde Buse ÖCAL

Yüksek Lisans, Bilgisayar Animasyonu ve Oyun Teknolojileri Bölümü

Tez Danışmanı: Doç. Dr. Burkay GENÇ

Mayıs 2022, 64 sayfa

Günümüzde bir web sitesi, bir uygulama veya bir oyun farklı cihazlarda ve ekran boyutlarında çalışabilecek şekilde geliştirilmektedir. Geliştirme süresi bakımından bu konu ele alındığında, bu durumun hem arayüz tasarımcılarına hem de arayüz yazılımcılarına ekstra zamana mal olduğu görülmektedir. Literatürde bu durumu önlemek için çalışmalar olmasına karşın bu çözümlerden bazıları halihazırdaki sorunu tam olarak ortadan kaldırmayıp, hafifletmekte, bazıları ise henüz çalışma esnasında uygulanabilecek seviyede olmamaktadır.

Bu tezde gün geçtikçe artan yeni cihaz sayısı ve buna bağlı olarak çeşitlenen ekran en-boy oranı, boyutu ve çözünürlüğüne uygun arayüzleri hem tasarım hem yazılım yönünden geliştirmek için harcanan süreyi kısaltmaya yönelik çözümler aranmış ve bu problemi çözmekte temel kabul edilebilecek bir algoritma önerilmiştir.

Anahtar Kelimeler: akıllı arayüz yerleştirmesi, kullanıcı arayüzü, arayüz estetiği, çözünürlüğe duyarlı arayüz

ABSTRACT

AN ADVANCED LAYOUT MANAGER FOR VARIABLE USER INTERFACES

Gözde Buse ÖCAL

Master of Science, Department of Computer Animation and Game Technologies

Supervisor: Assoc. Prof. Burkay GENÇ

May 2022, 64 pages

Today, a website, an application or a game is developed to run on different devices and screen sizes. Considering this issue in terms of development time, it seems that this situation costs both interface designers and interface developers extra time. Although there are studies in the literature to prevent this situation, some of these solutions do not completely eliminate the current problem, but reduce it, and some are not yet at a level that can be applied during the run-time.

In this thesis, solutions have been sought to shorten the time spent on developing interfaces, both in terms of design and software, suitable for the increasing number of new devices and the screen aspect ratio, size and resolution, which are increasing day by day and an algorithm that can be considered as the basis for solving this problem has been proposed.

Keywords: smart interface layout, user interface, interface aesthetics, resolution sensitive interface

TEŞEKKÜR

Bu çalışmanın yürütülmesi esnasında bilgi birikimiyle bana yol gösteren, yardıma ihtiyacım olduğunda bana destek olan ve beni cesaretlendiren değerli danışmanım Doç. Dr. Burkay Genç'e sonsuz teşekkürlerimi sunarım.

Tezimin değerlendirmesinde destek olan ve çalışmamı gözden geçiren tez değerlendirme jüri üyeleri Prof. Dr. Haşmet Gürçay ve Dr. Öğr. Üyesi Fatih Sağlam'a teşekkürü borç bilirim.

Son olarak, bana bu yolda hiçbir zaman desteklerini eksik etmeyen annem Semra Çakmak, kız kardeşim Hilal Ebru Öcal, ablam Yeliz Çiçek ve teyzem Sevgi Yalçın'a sonsuz teşekkür ederim.

İÇİNDEKİLER

ŞEKİLLER DİZİNİ

TABLOLAR DİZİNİ

ALGORİTMALAR DİZİNİ

1. GİRİŞ

1.1. Problemin Tanımı

Bir oyun, web sitesi veya uygulamanın arayüzü kullanıcının arkada çalışan sistemle iletişime geçmesini sağlayan yapıdır. Arayüz tasarımının kullanıcı deneyimine uygun bir şekilde tasarlanması, kullanımı kolaylaştıracak ve tercih sebebi olacaktır. Arayüz tasarımına bağlı kullanıcı deneyimi farklı cihazlara göre değişebilmektedir. Bu duruma mobil cihazlardaki uygulamalarda sıkça karşılaşılan ve sağ altta bulunan "+" butonu örnek olarak gösterilebilir. Uygulamanın sunduğu hizmete göre yeni bir şeyler oluşturmaya yarayan bu buton, bir web sayfasında kullanılırsa kafa karışıklığına sebep olacaktır. Çünkü web sayfalarında sağ altta bulunan benzer yapıdaki buton, genellikle yardım butonu veya sayfa başına dönme butonu olmaktadır. Bir web sayfasında bu konuma "+" butonu koymak, kullanıcının alışık olduğu kullanıma aykırı olduğu için kafa karışıklığına sebep olacak ve kullanıcı deneyimini olumsuz etkileyecektir. Kullanıcı deneyimine etki eden faktörleri detaylı bir şekilde sınıflandırmak fazlasıyla zordur. Yine de literatürde arayüz estetiğini ölçmeye yarayan bazı metrikler ve formüller mevcuttur [8].

Kullanıcı deneyimine etki eden faktörler, sadece arayüz elemanlarını alışılagelmiş konumlara yerleştirmek değil, aynı zamanda bu elemanların boyutlarına ve ekran içindeki dizilim şekline de bağlıdır. Hemen her cihazda bir internet tarayıcısı olduğunu göz önüne aldığımız zaman, özellikle web siteleri gibi yapılar, aynı içeriği çok çeşitli ekran boyutlarında göstermek zorundadır. Üstelik her geçen gün çeşitlenen cihaz türleri sebebiyle, bu cihazların boyutları, en-boy oranları ve çözünürlükleri de çeşitlenmeye başlamıştır. Bu sebeplerden ötürü, bir uygulamanın, oyunun veya web sitesinin arayüzü geliştirilirken farklı cihazların ekranlarında gösterilebilecek ve kullanıcı deneyimini en üst düzeyde tutacak şekilde tasarlanır. Dolayısıyla arayüz tasarımcıları ve ön yüz geliştiricileri çeşitli ekran boyutlarına, oranlarına ve çözünürlüklerine uygun tasarımlar yapmak ve kodlamak zorundadır. Bu süreci kolaylaştırmak için uygulanan yöntemler mevcuttur. Ekranı eş boyutlu sütunlara bölerek tasarım yapmak, ekran boyutları için bazı sınır değerler belirleyerek değişen tasarım sayısını aza indirmek veya sektörde öncü şirketlerin belirlediği tasarım kalıplarını kullanmak, arayüz geliştirme sürecini kolaylaştıracak yöntemler arasında sayılabilir. Yine de bu yöntemler çeşitli ekranlar için farklı arayüz tasarlamanın önüne geçememiş sadece var olan problemi azaltabilmişlerdir.

Farklı cihazlara ait farklı ekran oranları ve boyutları ortaya çıkmaya devam ettikçe, arayüzlerin geliştirilme süreleri de uzamaya devam edecektir. Geliştirme sürecinde tasarımcılardan beklenen farklı ekranlara uyumlu tasarımlar, yazılım kısmında da uygun bir şekilde kodlanmak zorunda olup zaman almaktadır. Üstelik geliştirme süreci bitse bile yeni çıkan ve/veya tasarımcıların öngöremediği, bahsedilen kolaylaştırma yöntemlerindeki kurallara uymayan ekran boyutları, oranları ve çözünürlükleri için arayüzün uyumlu hale getirilmesi gerekecek ve tekrar tasarlanacaktır. Bu durumda geliştirme süreci biraz daha uzayacak ve kullanıcı deneyimi de olumsuz etkilenecektir.

1.2. Çözüm Önerisi

Arayüzün farklı ekran boyutlarına uyabilmesi için yeniden tasarlanıp yazılması sorunu sektörde sıkça karşılaşılan bir problemdir ve henüz pratikte kullanılabilecek kadar gelişmiş bir çözümü bulunmamaktadır. Halihazırda kullanılan yöntemler de problemi tam olarak çözmeyip, basitleştirmek üzerine dayanmaktadır. Arayüzün en basit haliyle tasarımcıdan alınıp, minimum bilgiler ile istenilen ekran boyutlarına dinamik bir şekilde ve çalışma esnasında yerleşmesini sağlayacak bir yöntem, şüphesiz sadece arayüz tasarımcılarının işini kolaylaştırmayacak, aynı zamanda geliştirme sürecini de büyük ölçüde hızlandıracaktır.

Bu çalışmada literatürde var olan arayüzde bulunan bileşenleri topolojik bir ağaç yapısında göstererek, verilen ekran boyutlarına uygun olarak çizdirmeye çalışan bir algoritma [4] kullanılmıştır. Kullanılan algoritma tasarımcıdan basitçe bir arayüz bileşeninin minimum ve maksimum, genişlik ve yükseklik değerlerini alarak, yine kullanıcıdan alınan ekran genişlik ve yüksekliğine tam sığan yerleşim dizilimlerini dinamik olarak bulmaktadır. Lakin bu algoritma estetik olarak yetersiz kalmakla birlikte, arayüz bileşenlerinin, girilen ekran boyutuna tam sığan bir yerleşim dizilimi bulamadığı durumlarda çalışmamaktadır.

Bahsi geçen problemler 3 ana başlık altında incelenmiştir. Bunlar; verilen arayüzün ekranı doldurmadığı durumlar, verilen arayüzün ekrana sığmadığı durumlar ve estetik ölçümündeki engellerdir.

Arayüzün ekranı doldurmadığı durumlarda uygun bir yerleştirme sunabilmek için tasarımcı tarafından verilen topolojik ağaç yapısına boş elemanlar eklenmiş ve bunların ekranda doldurucu görevi görmesi hedeflenmiştir. Kullanıcıya daha fazla seçim hakkı verebilmek için ayrıca yatay ve dikey, başlangıç, orta ve son hizalama seçenekleri konulması uygun görülmüştür.

Arayüzün ekrana sığmadığı durumlarda ise basitçe ekrana yatay ve dikey kaydırma eklenmiş, yerleştirme işlemi ekran boyutlarına göre değil, arayüzün sığacağı en küçük çerçeveye göre yapılmıştır.

Estetik ölçümü iyileştirmek için ise öncelikle kullanılan algoritmada [4], çalışmaya uygun olmaması durumuyla kullanılamayan formüller eklenmiş, estetik ölçümdeki verimi arttırmak için de metrikler önem durumuna göre ağırlıklar kullanılarak hesaplanmıştır.

Yapılan çalışmanın akıllı arayüz yerleştirme alanında ileriki çalışmalara temel olacak nitelikte olması ve arayüz geliştirme sürecinde hem arayüz tasarımcılarının hem de önyüz geliştiricilerinin üzerindeki yükü alması hedeflenmiştir.

1.3. Tezin Bölümleri

2. bölümde, arayüz geliştirme sürecini kolaylaştırmak için yapılan çalışmalardan bahsedilmiş ve bu çalışmada temel olarak kullanılan algoritmalar ve estetik ölçüm metriklerine değinilmiştir.

3. bölümde, kullanılan algoritmada [4] var olan ve içeriği değiştirilen sınıflar ile yeni eklenen sınıflara değinilmiştir. Bahsedilen sınıf ve yapıların içeriği paylaşılmış ve neden gerekli oldukları ve çalışma şekilleri açıklanmıştır.

4. bölümde, tasarımcı tarafından verilen ağaca uygun yerleşim dizilimlerini bulan `GetRanges` algoritması paylaşılmıştır. Yerleşim dizilimleri için sınıflandırma yapılmış ve önceliğe göre uygun bulunmayan dizilimler elenmiştir. Bulunan uygun yerleşim dizilimleri `EliminateLayouts` algoritması ile kullanıcı tarafından girilen kırpma değerlerine göre tekrar elemeye tabi tutulmuştur.

5. bölümde, `GetFinalLayout` algoritması kullanılarak bulunan son uygun dizilim, kullanıcı tarafından girilen boyutlardaki ekrana yerleştirilmiştir. Son uygun dizilimi yerleştirmek dışında bu algoritmanın neden çalıştığı açıklanmış ve estetik ölçüm için kullanılma biçiminden bahsedilmiştir.

6. bölümde, son yerleşim dizilimini bulan estetik ölçüm metrikleri anlatılmış ve bu metrikleri hesaplarken kullanılan formüller verilmiştir. Ağırlıklı ölçüm yaparken kullanılan katsayılar ve neden bu katsayıların seçilmesinde etkili olan çalışmalar paylaşılmıştır.

7. bölümde, algoritma, hız, ağırlıklı ve ağızlıksız estetik ölçüm farkı ve hizalamanın estetiğe etkisini ölçmek için üç farklı test yapılmış ve sonuçlar tartışılmıştır. Testleri yaparken kullanılan ağaç yapıları açıklanmış ve görsel olarak verilmiştir.

8. bölümde ise geliştirilen algoritma yorumlanmış, algoritmanın çalışmasını olumsuz etkileyen durumlara değinilmiştir. Bu hususta gelecek çalışmalarda yardımcı olabilecek konular tartışılmıştır.

2. LİTERATÜR ÖZETİ

Literatürde gerçek zamanlı arayüz adaptasyonu sağlamaya yönelik çalışmalar mevcuttur. Gajos ve Weld, [1] yaptığı çalışmada, arayüzün dinamik uyum sağlaması sorununu optimizasyon problemi olarak tanımlamıştır. Çalışmada, ekran sınırlarına uygun ve kullanıcının kullanım desenlerine bağlı olarak aksiyon alma işlemini kolaylaştıran Supple ismini verdikleri sistem ile arayüz oluşturma algoritması ortaya koymuşlardır. Yöntemlerinin uygunluğunu gösterebilmek için testler yapmış olsalar da bu testler sadece basit yapıdaki arayüzler için yapılmış olup, kompleks arayüzlerde nasıl sonuç vereceği bilinmemektedir. Sottet vd. [2] tasarım zamanı ve gerçek zaman süreçlerini bir araya getirmek ve gerçek zamanda da tasarım yapılabilmesi amacıyla model-tabanlı yazılım yöntemlerini incelemiştir. Arayüz deneyimleri ile model-tabanlı yazılım deneyimleri arasındaki bağı bularak, gerçek zamanlı arayüz oluşturulmasında model-tabanlı yazılım yöntemlerini kullanmayı önermiştir. Roscher vd. [3] arayüzü otomatik bir şekilde dağıtabilen ve öngörülemeyen durumlara uyum sağlayabilen gerçek zamanlı değişiklik gösterebilen model-tabanlı bir yaklaşım önermiştir. Bu yaklaşımda, şekillendirilebilirlik, dağılım, çok modluluk, paylaşılabilirlik, birleştirilebilirlik arayüzlerin sıkça görülen özellikleri olarak kabul edilmiştir. Gerçek zamanlı uyum sağlama tasarımcının belirlediği arayüz elemanları ve elemanların sınırlarıyla sağlanmaktadır. Buna ek olarak, kullanıcılara kendi ihtiyaçlarına göre kişiselleştirebilmeleri için meta kullanıcı arayüzü imkanı sunulmuştur. Son olarak Çelik vd. [4] arayüz elemanlarının ve bu elemanlara ait sınırların tasarımcı tarafından belirlendiği ve topolojik bir ağaç yapısında tutulduğu bir dinamik arayüz oluşturma algoritması önermiştir. Bu yöntem ve buna ek olarak önceden hesaplanmış değerlerin hafızada tutulması sayesinde, algoritma çok kompleks arayüzler için bile çok hızlı sonuçlar verebilmektedir. Ayrıca Çelik vd. bu çalışmasında, arayüzün estetik olarak uygunluğunu da göz önünde bulundurmuş ve Ngo vd. [7, 8] tarafından önerilen metrik hesaplamalara göre birden fazla bulunan arayüzler arasında puanlama yaparak algoritmanın en uygun arayüzü seçmesini sağlamıştır. Lakin kullandığı estetik ölçümleri yeterli gelmemiştir. Üstelik verilen topolojik ağaç yapısının, eldeki ekran çözünürlüğüne tam olarak uymaması durumuna bir çözüm önermemiştir.

Arayüzlerin estetiğini ve kullanılabilirliğini ölçmek için yapılan çalışmalar [12, 13, 14] sadece elemanların yerleştirilme düzenini değil, renk, yapısal sıralamalar vb. bu çalışmada ele alınmayan, tasarımcının karar verdiği özellikleri de incelemektedir. Estetiğin ölçülebilmesi için, literatürde bulunan çalışmaların birçoğu Gestalt prensiplerine [5]

dayanmaktadır. Gestalt prensipleri, anlamlı bir algıya sahip olmanın temelde hangi kurallara dayandığını formüle etmeye çalışır. Todorovic, [5] Gestalt prensiplerini incelediği çalışmasında, yakınlık, benzerlik, kapatma, ortak kader, iyi gestalt, geçmiş deneyim ilkelerini incelemiştir. Zeidler vd. [6] arayüz geliştirirken, elde olan boşluğu, göze en iyi hitap edecek şekilde arayüz elemanlarına dağıtmak için kullanılabilecek stratejileri tanımlamış ve karşılaştırmıştır. Bunun için group layout, grid layout, flow layout ve constraint-based layout karşılaştırmaları yapmış ve Gestalt prensiplerinin arayüz yerleştirme estetiğinde, elemanların nasıl yerleştirileceğini önemli ölçüde etkilediğini ancak bunu uygulamak için henüz uygun bir yöntem bulunmadığını ortaya koymuştur. Ayrıca çalışmasında ağırlıklı dağıtımın daha çok basit tasarımlarda, eşit dağıtımın ise kompleks tasarımlarda etkili olduğunu ortaya koymuştur. Ngo vd. [7] yaptığı çalışmada grafiksel görünümü 14 estetik ölçüm metriği kullanarak ölçmeye çalışmış ve her bir metrik için 0 ile 1 arasında değerler veren formüller geliştirmiştir. Başka bir çalışmada Ngo vd. [8] Gestalt prensiplerini incelemiş ve arayüz estetiğinin ölçülebilmesi için hesaplamalı teoriler öne sürmüştür. Çalışmasında daha çok yapısal konseptlerin algısal ölçümü üzerine odaklanmıştır. Altaboli ve Lin, [9] yaptığı çalışmada Ngo'nun öne sürdüğü bu ölçümlerin hepsinin eşit önemde olmadığını öne sürmüş ve denge, birlik ve sıralama ölçümlerinin önemli ölçüde arayüz estetiğini etkilediğini göstermiştir. Zen vd. [10] Ngo'nun estetik ölçümlerini kullanarak, var olan arayüzleri basitleştirip, arayüz ölçümlerini yapan QUESTIM web servisini geliştirmiştir. Bu ölçüm yöntemi sadece arayüz tamamen ortaya çıktığında kullanılabilirdir. Son olarak, Zen vd. [11] denge, simetri, oran ve sadeliği 4 ana metrik olarak seçmiş ve 10 GUI üzerinden yarı-otomatik hesaplama ile QUESTIM'i test etmiştir. Bu çalışmasında yatay denge ve sadelik için hesaplanan sonuçların iyi tahmin yaptığı lakin simetri ve orantı hesaplamalarının güvenilirliğinin düşük olduğu ve bunlar için yeni formüller geliştirilmesi gerektiği sonucuna varmıştır.

Arayüz estetiğini veya kullanılabilirliğini ölçmek ve test etmek için yapılan çalışmalar bunlarla sınırlı değildir. Çalışmalarda Ngo'nun estetik ölçüm metrikleri kullanarak test etmenin yanı sıra son kullanıcıyla yapılmış testler de mevcuttur. Hartmann vd. [13] arayüz estetiği ve bunun diğer niteliklerle etkileşimi üzerine kullanıcı muhakemesine dayanan çalışmalar yapmıştır. Riegler ve Holzman [15] mobil arayüz tasarımında kullanıcı testlerine gerek kalmadan, arayüzün karmaşıklığını ve kullanılabilirliğini ölçmek için metrik ölçümler kullanmıştır. Purchase vd. [16] Ngo.'nun öne sürdüğü metrikleri test etmiş ve bu metrikleri ayrıca tüm bir yerleşime ve tekil bileşenlere etki eden olarak gruplandırmıştır. Bulduğu

sonuçta metriklere ağırlık verilmesinin gerekli olduğunu öne sürmüştür. Son olarak Bourguet [17] Ngo.'nun metriklerini kullanarak yapılan çalışmaları incelemiş ve farklı segmentasyon yöntemlerinin metrikleri hesaplama üzerindeki etkilerini gösteren deneysel bir çalışma yapmıştır.

3. KULLANILAN SINIF VE YAPILAR

3.1. Yerleşim Ağacı ve Layoutable Sınıfları

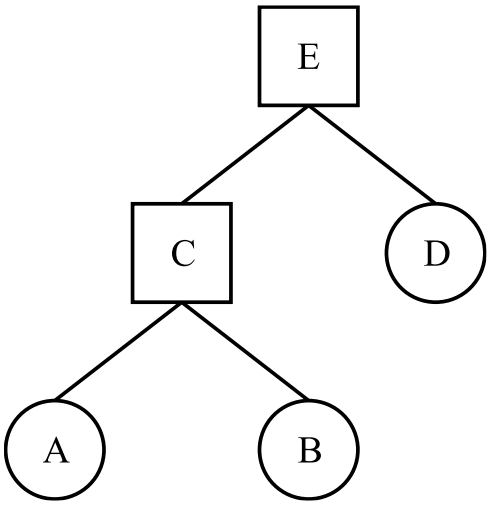
Çelik vd.[4]'nin de çalışmasında kullandığı gibi, bu çalışmada da her bir Java Swing elemanı ilişkisel bir ağaç yapısı içerisinde gösterilmiştir. Oluşturulan ağaç yapısına göre tanımlanacak her bir `JComponent` objesi için `LayoutComponent` sınıfı, bu sınıfın objelerini birbirine bağlayan her bir ağaç düğümü için de `LayoutContainer` sınıfı oluşturulmuştur. Bu sınıfların her ikisi de kendilerine atanan konum değerleri, yükseklik ve genişliklerini tutmaktadır. Bunların yanı sıra her iki sınıfta da `LayoutComponent` sınıfında tek obje, `LayoutContainer` sınıfında genişletilmiş dizi olacak şekilde alabilecekleri maksimum ve minimum yükseklik ve genişlikleri tutacak değişkenler ve `LayoutContainer` sınıfında ayrıca ebeveyni olduğu `LayoutComponent` objelerinin (bundan sonra bileşen olarak geçecektir) veya `LayoutContainer` objelerinin (bundan sonra kapsayıcı olarak geçecektir) listesi ve atanmış yatay veya dikey sıralama bilgileri tutulmaktadır. Burada atanmış değerlerdeki kasıt uygun yerleşim dizilimi bulunduktan sonra dizilimin yerleştirme işleminde ağacın her bir elemanına verilen değerlerdir.

```
LayoutComponent

JComponent jcomponent;
WidthHeightRange range;

int assignedX;
int assignedY;
int assignedWidth;
int assignedHeight;
```

Şekil 3.1. LayoutComponent sınıfında tutulan değişkenler.

```
LayoutContainer

Layoutable[] children;
ExtendedArray<WidthHeightRange>
memory;

int assignedX;
int assignedY;
int assignedWidth;
int assignedHeight;
OrientationEnum
assignedOrientation;
```

Şekil 3.2. LayoutContainer sınıfında tutulan değişkenler.



Şekil 3.3. Örnek bir ağaç yapısı (A, B, D bileşen, C, E kapsayıcı olmak üzere).

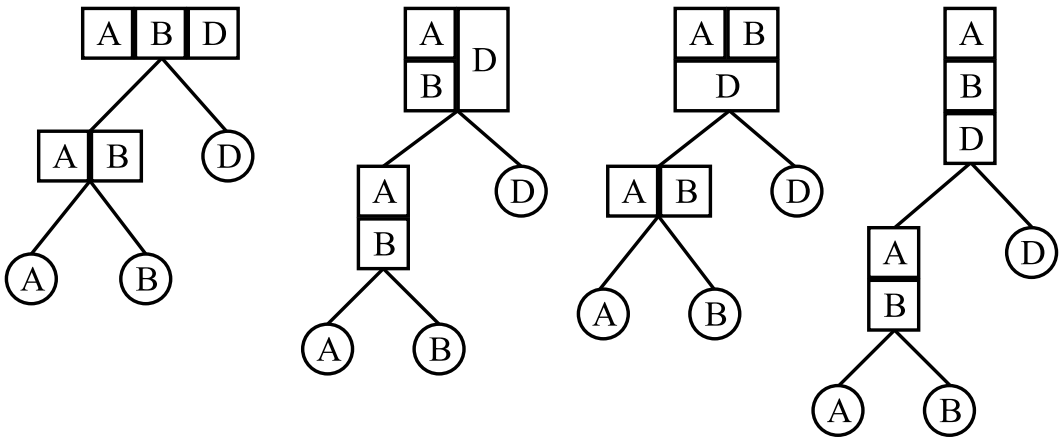
3.2. WidthHeightRange Sınıfı

`WidthHeightRange` sınıfı, basitçe bir bileşen veya kapsayıcının alabileceği minimum ve maksimum genişlik ve minimum ve maksimum yükseklik değerlerini tutar. Bu değerler, ağaç yapısı oluşturulmadan önce arayüz tasarımcısı tarafından girilecek değerlerdir. Bu sınıfın sahip olduğu diğer değişkenler ise yatay, dikey veya tekil sıralama bilgisi, boşluk, yatay kaydırma veya dikey kaydırmaya ihtiyacı olup olmadığı bilgisidir. Sıralama değeri, kapsayıcı objeler için "yatay" veya "dikey" olabilirken, bileşen objeler için "tekil" değerini alabilir. Bu değerler, 4. bölümde gösterileceği üzere olası yerleşimler hesaplanırken sıralamanın hangi yöne göre olacağına karar vermek içindir. Boşluk, yatay kaydırma ve dikey kaydırma gerekliliğini tutan değerler ise kullanıcı tarafından girilen genişlik ve yükseklik değerlerine göre hesaplanan değerlerdir. Bu değerlere göre hesaplanan yerleşim dizilimleri sınıflara ayrılır ve gerekli elemeler yapılır.

`WidthHeightRange` sınıfında tutulan *subRanges* isimli dizi, bileşenlerde *null* olacak şekilde, kapsayıcılarda ise çocuklara ait aralık değerlerini tutacak şekilde bulunur. Benzer şekilde minimum ve maksimum genişlik ve yükseklik dizileri de bu alt aralıkların minimum ve maksimum değerlerini tutar.

```
WidthHeightRange

int minWidth, maxWidth;
int minHeight, maxHeight;

boolean hasFiller;
boolean hasVerticalScroll;
boolean hasHorizontalScroll;

OrientationEnum orientation;

WidthHeightRange[] subRanges;
int[] minWidthValues, maxWidthValues;
int[] minHeightValues, maxHeightValues;
```

Şekil 3.4. WidthHeightRange sınıfında tutulan değişkenler.



Şekil 3.5. *Şekil 3.3*'te verilen ağaç yapısına ait alt aralık dizisinin gösterimi.

3.3. ExtendedArray ve ExtendedArrayList Sınıfları

4. bölümümde de daha detaylı bahsedileceği üzere bu çalışmada, yerleşim dizilimlerinde elemeler yapılırken bütün ihtimaller ele alınarak eleme yapılmıştır. Algoritma çalışırken önce uygun yerleşim dizilimleri olup olmadığına bakmakta, bulamadığı durumda sırayla boşluk, dikey kaydırma, yatay kaydırma ve en sonda kalan bütün yerleşimlere bakmaktadır. Bu hedefi gerçekleştirebilmek için yerleşim dizilimlerini bulma algoritması tekrar tekrar çalışmaktadır. Bu yükü azaltmak için her bir `LayoutContainer` sınıfı kendi içinde daha önce hesaplanmış alt çocukların yerleşim dizilim bilgisini tutmaktadır. Bu noktada hangi

çocuğun yerleşim dizilim bilgisini hesaplayıp hangisini hesaplamadığı bilgisini tutmak büyük önem taşır. Hesaplanmayan bu çocukların bilgileri *null* olarak tutulur. Lakin normal bir dizinin ve `ArrayList`in tutabileceği eleman sayısı, *integer* bir değerin alabileceği maksimum değer olan $2^{31} - 1$ sayısından fazla olamaz. Bu algoritma ile oluşturulan yerleşim ağacından ortaya çıkabilecek bütün dizilimler de çok kolay bir şekilde bu sayıyı geçebilmektedir.

Bahsi geçen problemi çözmek için `ExtendedArray` ve `ExtendedArrayList` sınıfları sırayla genişletilmiş dizi ve genişletilmiş `ArrayList` gibi davranacak şekilde oluşturulmuştur. Sınıflarda tek boyutlu gibi davranan iki boyutlu diziler ve `ArrayList`ler kullanılmıştır. Bu sınıflara tek boyutlu denilmesinin sebebi, yapılarının normal iki boyutlu yapıların aksine satır ve sütun olarak değil, her satırın birbirini takip edecek şekilde oluşturulmasıdır. İlgili hesaplamalar yapılırken bu duruma uygun olarak matematiksel işlemler kullanılmıştır.

	Sütun1	Sütun2	Sütun3
Satır1	[0][0]	[0][1]	[0][2]
Satır2	[1][0]	[1][1]	[1][2]
Satır3	[2][0]	[2][1]	[2][2]

2 boyutlu dizi

	Sütun1	Sütun2	Sütun3
Satır1	[0]	[1]	[2]
Satır2	[3]	[4]	[5]
Satır3	[6]	[7]	[8]

ExtendedArray

Şekil 3.6. *2* boyutlu dizi ve ExtendedArray sınıfı ile oluşturulan dizilerin yapısı.

3.4. ComponentData ve QuadrantData Sınıfları

Estetik ölçüm hesaplaması yapılırken ağaçtaki her bir yaprağın, yani arayüz tasarımcısı tarafından girilmiş bileşenlerin, uygulamanın boyutuna ve oluşturulan yerleşim diziliminin boyutuna göre bazı değerleri hesaplanır. Seçilen herhangi bir dizilim için yerleştirme işlemi yapılırken her bileşen için bu sınıflar da oluşturulur ve ilgili değerler hesaplanır.

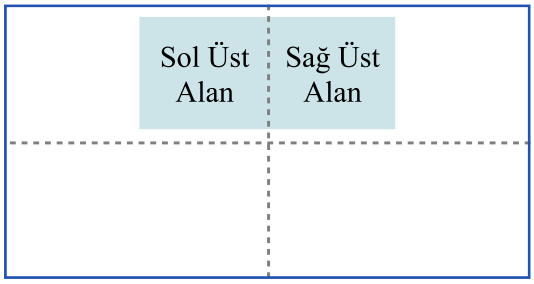
```
QuadrantData

QuadrantEnum
quadrantPosition;

float area;

float width;
float height;

float centerX;
float centerY;
```

Şekil 3.7. QuadrantData sınıfında tutulan değişkenler.

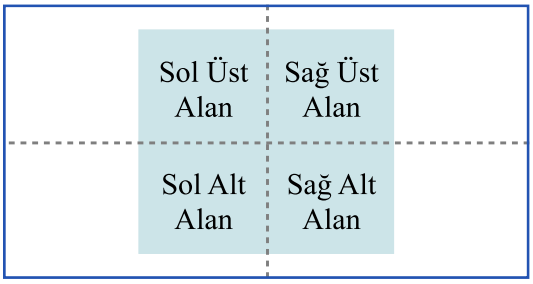
```
ComponentData

LayoutComponent component;

float totalArea;
float leftArea, rightArea;
float topArea, bottomArea;

float distance;
float verticalDistance;
float horizontalDistance;
float leftDistance;
float rightDistance;
float topDistance;
float bottomDistance;

float centerX, centerY;

ArrayList<QuadrantData> quadrants;
```

Şekil 3.8. ComponentData sınıfında tutulan değişkenler.

`ComponentData` sınıfı, o bileşene ait alan bilgilerini, `QuadrantData` tipinde bir listede koordinat bilgilerini, bileşenin merkezini ve içinde bulunduğu en büyük `JFrame`'in merkezine olan uzaklık bilgilerini tutmaktadır. `QuadrantData` sınıfı ise o bileşenin koordinat düzlemindeki durum bilgilerini tutan sınıftır ve her bileşende sol-üst, sağ-üst, sol-alt, sağ-alt olmak üzere 4 tane bulunmaktadır. Koordinat bilgileri bileşenin hangi bölgede olduğunun yanı sıra, o bölgede ne kadar alanı olduğu, o bölgedeki merkezi, yüksekliği ve genişliğini de tutmaktadır.



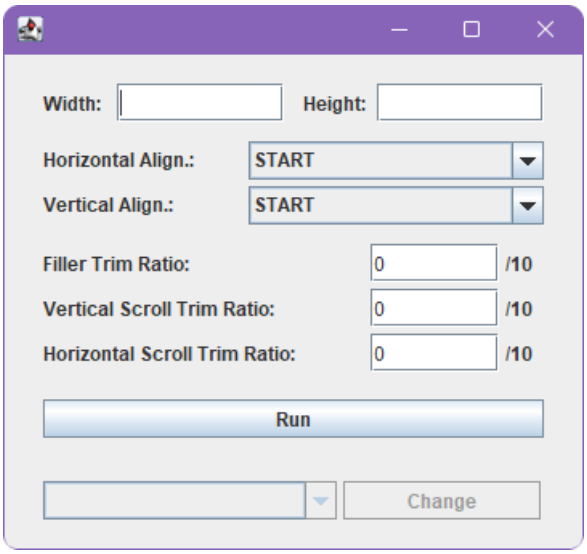
Şekil 3.9.(a) *2* farklı koordinat bölgesinde bölünmüş bir bileşen.



Şekil 3.9.(b) *4* farklı koordinat bölgesinde bölünmüş bir bileşen.

3.5. Controller Sınıfı

Bu sınıf algoritmanın başlangıcında gerekli girdilerin alınması ve eleme kontrolünün sağlaması için oluşturulmuştur. Kullanıcı tarafından girilecek ve bu tezde farklı yöntemlerin test edilmesini sağlayacak yatay-dikey hizalama ve eleme yöntemlerini seçmek için bir arayüz oluşturur.



Şekil 3.10. Controller penceresinin ekran görüntüsü.

Kullanıcı tarafından seçilebilen, yatay hizalama ve dikey hizalama, baş, orta, son olmak üzere 3 farklı seçenek içerir. Özellikle boşluklarla çizdirilebilen yerleşim dizilimlerinde, yerleştirme işlemi yapılırken, algoritmanın 9 farklı yerleştirme yapması sağlanmaktadır.

Boşluk kırpma oranı, çizdirilmek istenilen yükseklik ve genişlik değeriyle çarpılarak kabul edilebilir minimum yükseklik ve genişlik değerinin bulunmasını sağlar. Maksimum yükseklik veya genişlik değeri, sırayla kabul edilebilir minimum yükseklik ve genişlik değerinin altında kalan yerleşim dizilimleri elenir. Eğer kullanıcı tarafından genişlik 1000, boşluk kırpma oranı 7 olarak girilirse ve algoritmanın bulduğu yerleşim dizilimlerinden birinin minimum ve maksimum genişlik bilgisi (300-400) olursa;

1000 x 0.7 = 700, kabul edilebilecek en düşük genişlik ve

700 > 400, olduğu için bulunan yerleşim dizilimi elenecektir.

Eğer kullanıcı tarafından genişlik 1000, boşluk kırpma oranı 7 olarak girilirse ve algoritmanın bulduğu yerleşim dizilimlerinden birinin minimum ve maksimum genişlik bilgisi (800-900) olursa;

1000 x 0.7 = 700, kabul edilebilecek en düşük genişlik ve

700 < 900, olduğu için bulunan yerleşim dizilimi elenmeyecektir.

Dikey kaydırma kırpma oranı, çizdirilmek istenilen yükseklik değeriyle çarpılır ve istenilen yükseklik değeri ile toplanarak kabul edilebilir minimum yükseklik değerinin bulunmasını sağlar. Maksimum yükseklik değeri, kabul edilebilir minimum yükseklik değerinin altında kalan yerleşim dizilimleri elenir. Eğer kullanıcı tarafından yükseklik 1000, dikey kaydırma kırpma oranı 3 olarak girilirse ve algoritmanın bulduğu minimum ve maksimum yükseklik bilgisi (1100-1200) olursa;

1000 x 0.3 = 300, 300 + 1000 = 1300, kabul edilebilecek en düşük yükseklik ve

1300 > 1200, olduğu için bulunan yerleşim dizilimi elenecektir.

Eğer kullanıcı tarafından yükseklik 1000, dikey kaydırma kırpma oranı 3 olarak girilirse ve algoritmanın bulduğu minimum ve maksimum yükseklik bilgisi (1100-1400) olursa;

1000 x 0.3 = 300, 300 + 1000 = 1300, kabul edilebilecek en düşük yükseklik ve

1300 < 1400, olduğu için bulunan yerleşim dizilimi elenmeyecektir.

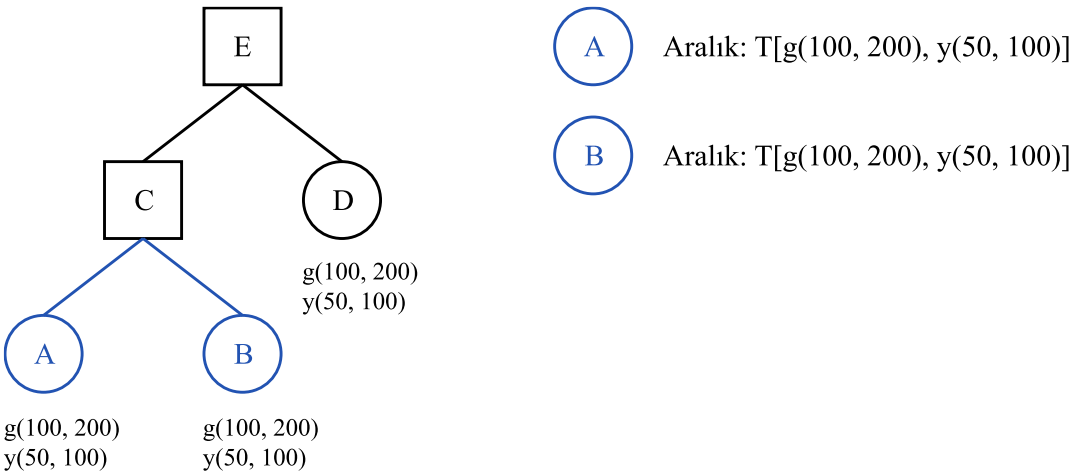
Benzer şekilde yatay kaydırma kırpma oranı ile genişlik üzerine eleme yapılır. Boşluk kırpma oranı, dikey ve yatay kaydırma kırpma oranı 0 girildiğinde herhangi bir eleme yapılmamaktadır.

4. UYGUN YERLEŞİM DİZİLİMLERİNİN HESAPLANMASI

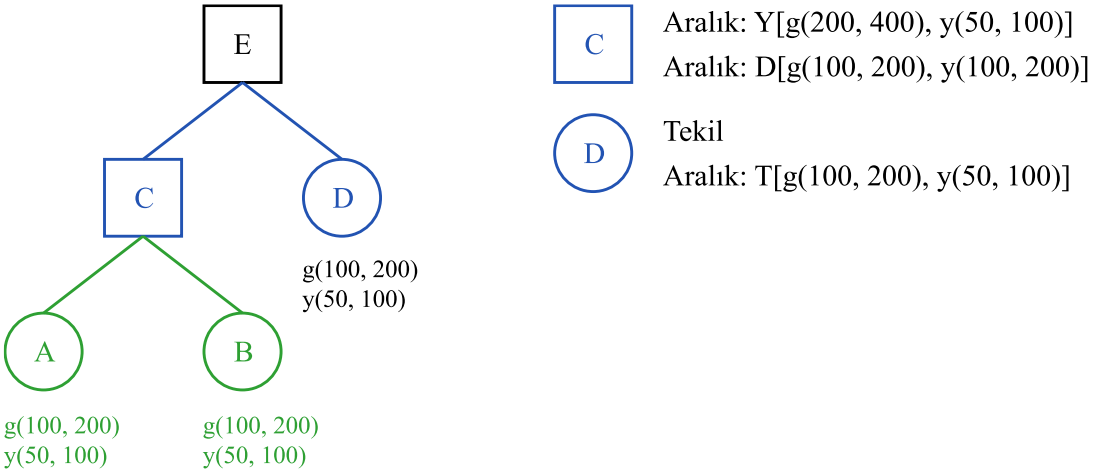
4.1. Olası Yerleşimler

Bu çalışmada algoritmanın en az bir tane çizdirilebilir yerleşim dizilimi bulması hedeflenmiştir. Bunun için kullanıcı tarafından girilen yükseklik ve genişlik değerlerine tam oturup oturmamasına bakmaksızın bütün yerleşim değerlerinin kontrol edilmesi ihtimali düşünülmüştür.

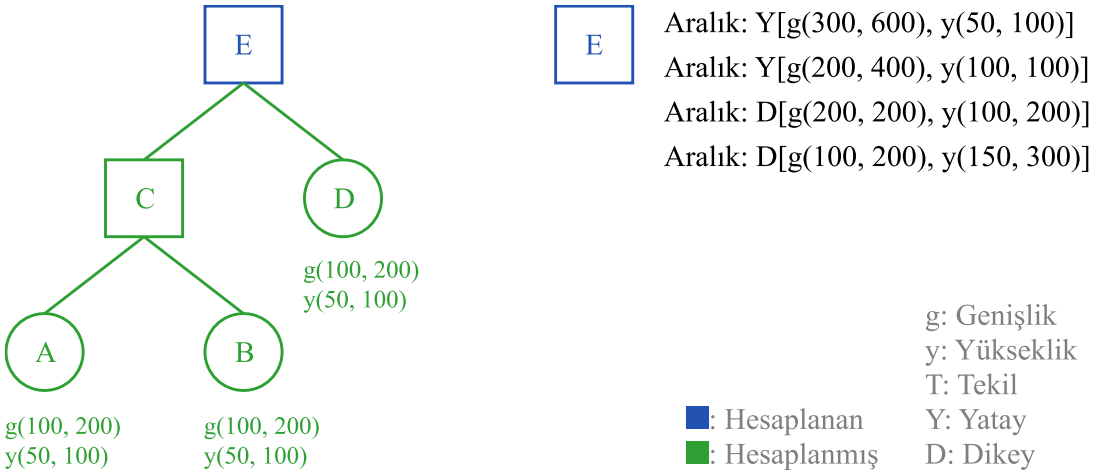
Yerleşim dizilimleri oluşurken bir kapsayıcı içinde bulundurduğu bütün kapsayıcı ve bileşen çocukları yatay veya dikey hizalamaya göre sıralar. Hesaplama yerleşim ağacının köküne kadar yapılır ve en sonunda arayüz tasarımcısı tarafından verilen bileşenlere ait minimum ve maksimum genişlik ve yükseklik değerlerinden, yerleşim ağacının tamamının çizdirilebileceği minimum ve maksimum genişlik ve yükseklik değerleri elde edilir.



Şekil 4.1.(a) Örnek yerleşim ağacının aralık değerlerinin hesaplanması, adım 1.

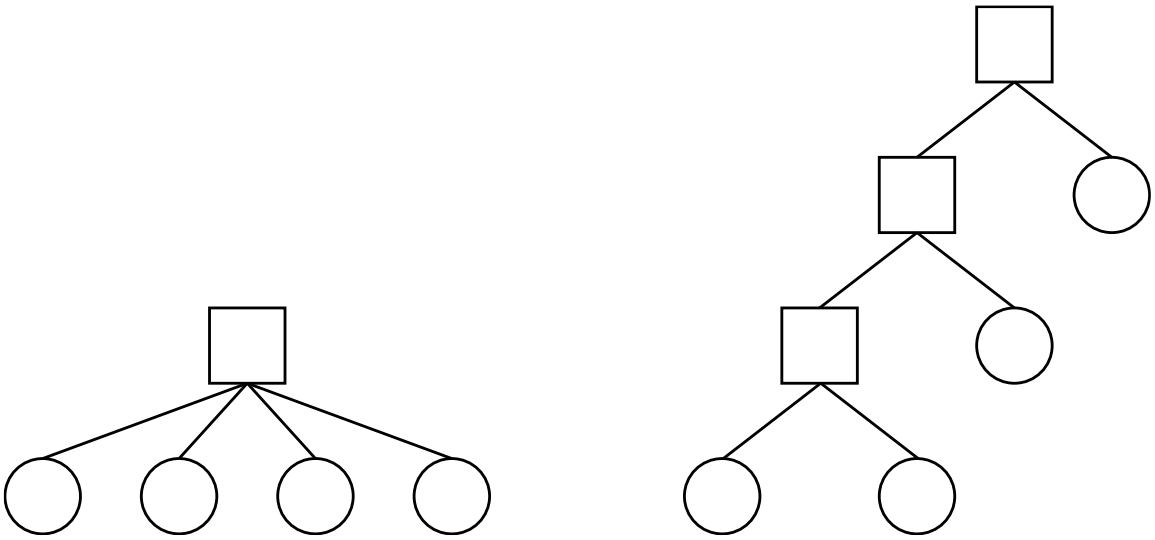


Şekil 4.1.(b) Örnek yerleşim ağacının aralık değerlerinin hesaplanması, adım 2.



Şekil 4.1.(c) Örnek yerleşim ağacının aralık değerlerinin hesaplanması, adım 3.

Bir ağaç için hesaplanabilecek minimum yerleşim dizilimi, bütün bileşenlerin tek bir kapsayıcının çocuğu olma durumda ortaya çıkar. Bütün bileşenler birbirleriyle ya yatay ya da dikey olarak sıralanabileceği için bileşenlerin sayısı ne olursa olsun olası minimum yerleşim sayısı 2 olacaktır.



Şekil 4.2. Soldaki minimum ve sağdaki maksimum olmak üzere olası yerleşim dizilimine sahip ağaç örnekleri.

Bir ağaç için hesaplanabilecek maksimum yerleşim dizilimi ise, ağaçta bulunan her bir kapsayıcının 2 çocuğa sahip olması ve bu kapsayıcılardan 1 tanesinin 2 bileşen çocuğa, kalanlarının ise 1 kapsayıcı, 1 bileşen çocuğa sahip olması durumunda ortaya çıkar. Bu durumda ortaya çıkacak bütün yerleşim dizilimlerinin sayısı 2^{n-1} olacaktır. Ortaya çıkacak

bu büyük miktardaki yerleşim dizilimlerinin her biri için işlem yapmak masraflı olacağı için bazı elemeler yapılması gerekmektedir. Öncelikle yerleşim dizilimleri bulunurken tam sığanlar, boşluklular, dikey kaydırılabilenler, yatay kaydırılabilenler ve hem dikey hem yatay kaldırılabilenler sınıflandırmalarına göre aranırlar. Uygun dizlimler bulunduğunda, en başta kullanıcıdan alınan kırpma değerlerine göre elenirler. Kalan yerleşim dizilimleri 5. ve 6. bölümde bahsedilecek yerleştirme ve estetik ölçüme tabi tutulur ve en uygun sonuç bulunur.

4.2. Sınıflandırma ile Yerleşim Dizilimlerinin Bulunması

Bu aşamada yerleşim ağacının kök düğümü, sırasıyla tam sığanlar, boşluklular, dikey kaydırılabilenler, yatay kaydırılabilenler ve hem dikey hem yatay kaldırılabilenler sınıflandırmaları için `GetRanges` algoritmasını çalıştırır.

4.2.1. GetRanges Algoritmasının Genel Çalışma Prensibi

`GetRanges` algoritması çalışırken önce bütün çocukların aralık değerlerinin hesaplanıp hesaplanmadığına bakar. Bütün çocuklar hesaplandıysa bu değeri döndürür. Hesaplanmadıysa her bir çocuk için `GetRanges` işlemine devam eder ve bunu yaparken sadece aralık değeri *null* olan yani hesaplanmamış çocuklar için işlem yapar. Algoritmanın sonunda bulunan bütün aralıklar kapsayıcının hafızasına atılır ve bir üst kapsayıcıya gönderilir. Eğer hiçbir aralık bulunamamışsa *null* gönderilir.

Bu algoritma bütün sınıflandırmalar için çalışır. Herhangi bir sınıflandırma için en az bir tane uygun yerleşim dizilimi bulunduğu anda algoritma kalanlara bakmaz ve bulunan dizilimler ile işleme devam edilir. Tekrar yapılan hesaplamalardan kaçınmak için kapsayıcı sınıfları daha önceki hesapları *memory* yani hafıza isimli bir değişken içinde tutar.

Algoritma 1: Kapsayıcılar için yerleşim dizilimlerini bulan GETRANGES algoritması.

Output: WidthHeightRange tipinde ExtendedArray objesi
if *isMemoryFull* **then return** *memory*
rangeSize ← 1
allNull ← true
childRanges ← ∅ WidthHeightRange tipinde ExtendedArray
rangeH ← ∅ WidthHeightRange tipinde ExtendedArray
rangeV ← ∅ WidthHeightRange tipinde ExtendedArray
foreach *child in children* **do**
  if *child*.GETRANGES = ∅ **then return** ∅

```
    childRanges ← child.GETRANGES elemanını ekle
    rangeSize ← rangeSize * child.GETRANGES.LENGTH
isMemoryFull ← true
index ← 0
for index < rangeSize do
    if memory≠ ∅ and memory.GET(index)≠ ∅ and memory.GET(index + rangeSize)≠∅ then
        rangeH[index] ← memory.GET(index)
        rangeV[index] ← memory.GET(index + rangeSize)
        allNull ← false
        break
    else
        minWidthH ← maxWidthH ← minHeightH ← 0
        minWidthV ← maxHeightV ← minHeightV ← 0
        maxHeightH ← maxWidthV ← sonsuz
        tempRangeH ← ∅ WidthHeightRange
        tempRangeV ← ∅ WidthHeightRange
        foreach childRange in childrenRanges do
            if childRange = ∅ then
                isMemoryFull ← false
                break
            minWidthH ← minWidthH + childRange.GETMINWIDTH
            maxWidthH ← maxWidthH + childRange.GETMAXWIDTH
            minHeightV ← minHeightV  + childRange.GETMINHEIGHT
            maxHeightV ← maxHeightV + childRange. GETMAXHEIGHT
            minHeightH ← MAX(minHeightH, childRange.GETMINHEIGHT)
            maxHeightH ← MIN(maxHeightH, childRange.GETMAXHEIGHT)
            minWidthV ← MAX(minWidthV, childRange. GETMINWIDTH)
            maxWidthV ← MIN(maxWidthV, childRange. GETMAXWIDTH)
            tempRangeH ← minWidthH, maxWidthH, minHeightH, maxHeightH değerleriyle
            oluşturulan WidthHeightRange objesi
            tempRangeH ← minWidthH, maxWidthH, minHeightH, maxHeightH değerleriyle
            oluşturulan WidthHeightRange objesi
            if maxHeightH < minHeightH then
                maxHeightH ← minHeightH
                tempRangeH.SETHASFILLER ← true
```

```
            if maxWidthV < minWidthV then
                maxWidthV ← minWidthV
                tempRangeV.SETHASFILLER ← true
            if childRange.GETHASFILLER then
                tempRangeH.SETHASFILLER ← true
                tempRangeV.SETHASFILLER ← true
        if tempRangeH ≠ ∅ then
            rangeH ← tempRangeH
            if rangeH algoritmanın çalıştığı o anki sınıflandırmaya uygunsa then
                rangesH[index] ← rangeH
                allNull ← false
            else
                rangesH[index] ← ∅
                isMemoryFull ← false
        else
            rangesH[index] ← ∅
        if tempRangeV ≠ ∅ then
            rangeV ← tempRangeV
            if rangeV algoritmanın çalıştığı o anki sınıflandırmaya uygunsa then
                rangesV[index] ← rangeV
                allNull ← false
            else
                rangesV[index] ← ∅
                isMemoryFull ← false
        else
            rangesV[index] ← ∅
            index ← index + 1
if allNull then
    isMemoryFull ← false
    return ∅
memory ← rangesH + rangesV
return memory
```

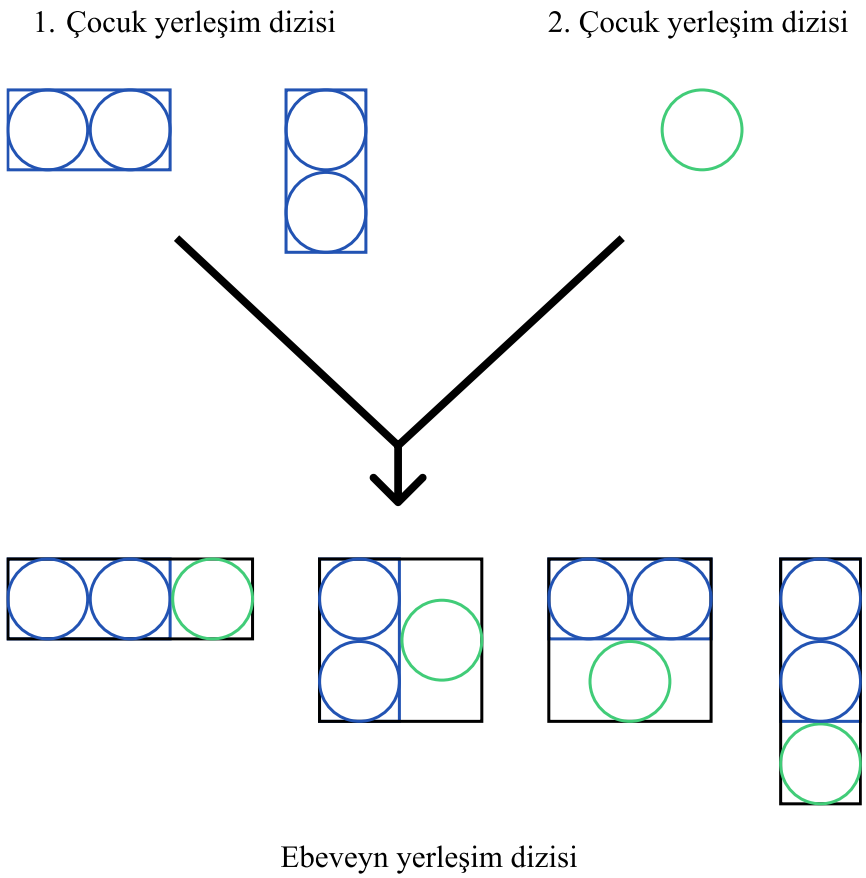
4.2.2. Aralık Dizisinin Oluşturulması

Aralık hesaplama işlemlerine başladıktan sonra önce kaç tane alt aralık tutulacağının sayısı hesaplanır. Bunun için bütün çocuklara ait alt aralıkların sayısı çarpılarak toplam alt aralık sayısının yarısı bulunur. Yarısı olmasının sebebi bütün bu dizilimlerin hem yatay hem de dikey için yapılacak olmasıdır. Örnek olarak; bir kapsayıcının `GetRanges` hesaplama algoritması ele alınabilir. Bu kapsayıcının 2 tane çocuğu varsa ve çocukların `GetRanges` algoritması ile sırasıyla dönen aralıklar;

1. çocuk için [Yatay[Tekil, Tekil], Dikey[Tekil, Tekil]] olmak üzere 2 uzunluğunda bir dizi,

2. çocuk için [Tekil] olmak üzere 1 uzunluğunda bir dizi olursa, kapsayıcının hesaplanan aralıkları;

[Yatay[Yatay[Tekil, Tekil], Tekil], Yatay[Dikey[Tekil, Tekil], Tekil], Dikey[Yatay[Tekil, Tekil], Tekil], Dikey[Dikey[Tekil, Tekil], Tekil]] olmak üzere 4 uzunluğunda bir dizi olacaktır ve bu sonuç 2 x 1 x 2 = 4 hesaplamasından ortaya çıkmaktadır. Bu örneğin görseli *Şekil 4.3*'te gösterilmektedir.



Şekil 4.3. Kapsayıcının alt aralıklarından kendi aralık değerlerini hesaplaması.

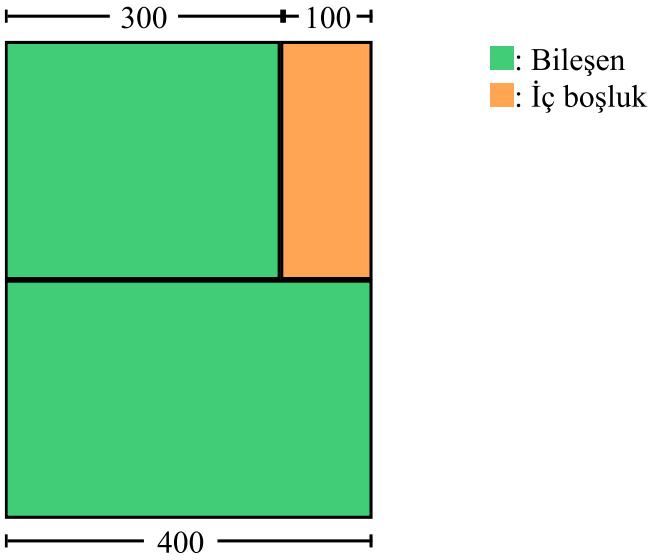
4.2.3. Aralık Dizisinin Değerlerinin Hesaplanması

Kapsayıcı, çocuklara ait aralıkları ve kendi aralık dizisinin uzunluğunu aldıktan sonra her bir çocuk aralığı sırayla kardeş aralıklarla toplayarak kendi aralıklarını hesaplar. Yatay sıralama için çocukların genişlik değerleri toplanır. Minimum yükseklik değeri için çocuklardan maksimum yüksekliğe sahip olanı, maksimum yükseklik değeri içinse çocuklardan minimum yüksekliğe sahip olanının değeri alır. Benzer hesaplama genişlik yükseklik, yükseklik genişlik olacak şekilde dikey sıralama için de yapılır. *Şekil 4.4*'de aralıklar bu hesaplamaya göre yapılmıştır.

Bazı durumlarda çocuklardan alınan minimum değer, maksimum değerden büyük olabilir. İki çocuktan birinin yükseklik aralığı (200-300), diğerininse (400-500) ise ve bu çocuklar yatay sıralanmak isteniyorsa, kapsayıcının yükseklik aralığı;

max(200, 400) = 400, minimum yükseklik değeri olacak şekilde ve

min(300, 500) = 300, maksimum yükseklik değeri olacak şekilde bulunur. Böyle bir durumda 400 olan minimum değer, 300 olan maksimum değeri geçmiştir. Bu durumu çözmek için kapsayıcının maksimum değeri minimum değere atanır ve yeterli olmayan çocuğa boşluk eklenir. Yani son durumda kapsayıcının yükseklik aralığı (400-400) olurken hem kapsayıcı hem de yeterli olmayan çocuk boşluğa ihtiyacı var olarak işaretlenir. Bu tarz bir hesaplamayla ortaya çıkan boşluğa iç boşluk denir.



Şekil 4.4. Yükseklik aralıkları (200-300) ve (400-500) olan 2 bileşenin iç boşluk alarak yatay olarak dizilimi.

4.2.4. Sınıflandırma Kısıtlamaları

`GetRanges` algoritması hesaplanan her bir aralığı kontrole tabi tutar. Bu kontrollerde bulunan aralığın sınıflandırmaya uygun olup olmadığına bakılır. Sınıflandırma elemesi, bu aşamada tam sığan yerleşim dizilimleri ve boşluklu yerleşim dizilimlerini ayırt edemez. Bunun sebebi bulunan aralıkların son ana kadar kullanıcı tarafından istenilen yükseklik ve genişlik değerlerinden küçük olarak ilerlemesidir. Bu da son noktada tam sığan olabilecek bir yerleşim diziliminin henüz alt aralıklar hesaplanırken elenmesine sebep olacaktır. Dolayısıyla bu aşamada boşluklu dizilim ve tam sığan dizilim ayrımı sadece boşluğa sahip olduğu kesin olan iç boşluklar için yapılır. Bu eleme işlemi sonra bahsedilecek olan bulunan yerleşim dizilimlerini eleme aşamasında yapılacaktır.

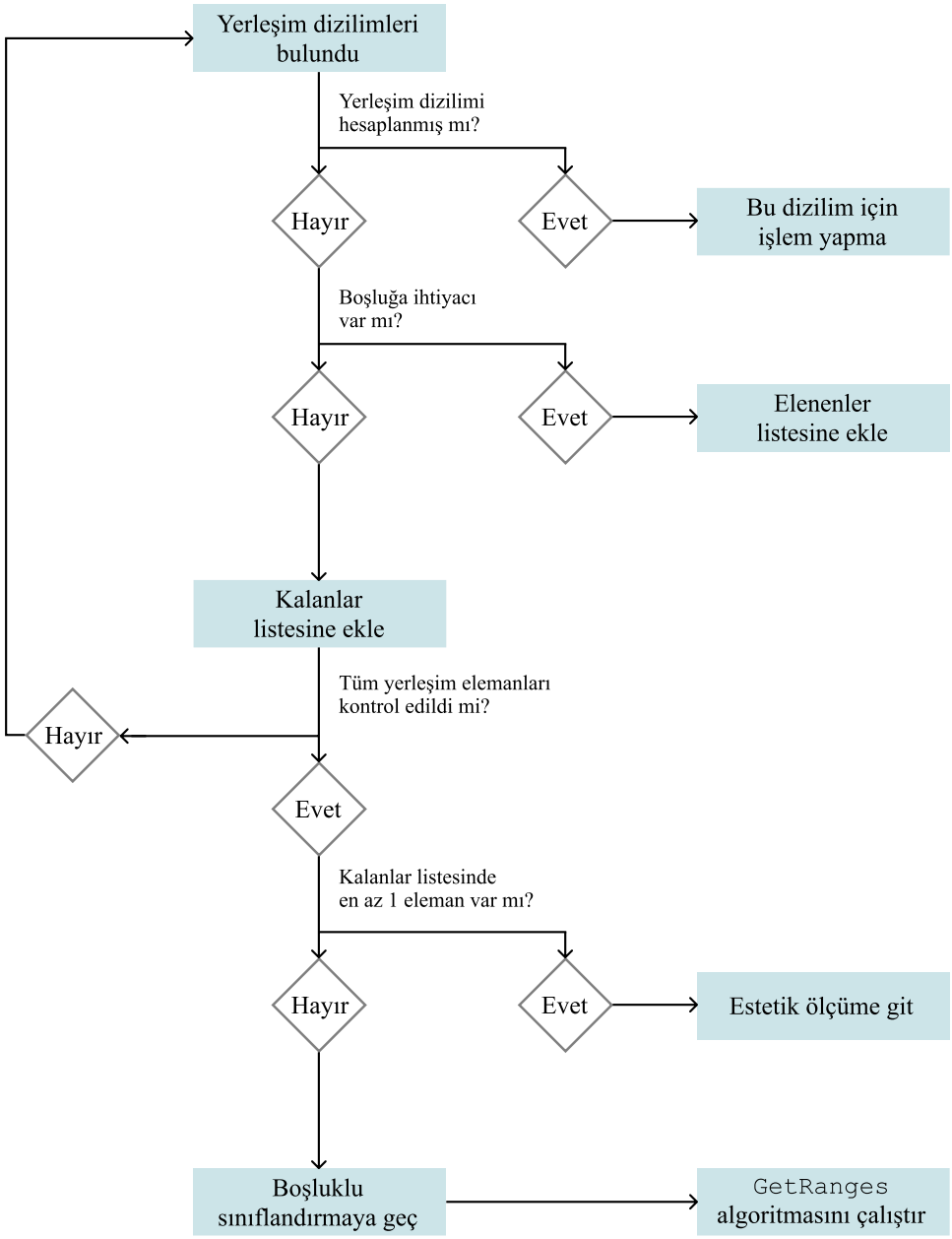
Kalan sınıflandırma elemanları için ise boşluklu dizilimler aranırken yatay ve dikey kaydırma ihtiyacı olmayana bakılarak, dikey kaydırma dizilimi aranırken yatay kaydırma ihtiyacı olmayana bakılarak, yatay kaydırma dizilimi aranırken dikey kaydırma ihtiyacı olmayana bakılarak ve her iki kaydırma olan dizilim aranırken herhangi bir elemeden geçirmeksizin kontrol edilir. Kontrol kurallarından da anlaşılacağı üzere kaydırmaya sahip olan sınıflandırmalar aynı zaman da boşluk da içerebilir.

4.3. Yerleşim Dizilimlerinin Elenmesi

4.3.1. Genel Bakış

`GetRanges` algoritması ile bulunan yerleşim dizilimleri, `EliminateLayouts` algoritmasına tabi tutulur. Algoritma bulunan bütün yerleşim dizilimlerini gözden geçirerek, hesaplanmamış dizilimleri, kullanıcı tarafından girilen kırpma değerleri dışında kalan dizilimleri ve sınıflandırmanın tam sığan yerleştirmelere baktığı durumlarda boşluklu dizilimleri eler. Algoritma, sınıflandırmanın tam sığanları kontrol ettiği durumlar ve diğer durumlar için farklı çalışmaktadır.

Tam sığanlar kontrol edilirken, hesaplanmış yani *null* olmayan yerleşim dizilimlerinin boşluğa ihtiyacı olma durumu kontrol edilir. Boşluğa ihtiyacı olan dizilimler elenir ve kalan dizilimler estetik ölçümüne tabi tutulur. Kırpma elemesi yapılmaz. Eleme işlemi sonunda tam sığan hiçbir yerleşim dizilimi bulunmazsa, sınıflandırma boşluklu dizilimleri arayacak şekilde değiştirilir ve `GetRanges` algoritması tekrar çalıştırılır. `GetRanges` algoritmasını tekrar çalıştırmadaki amaç, önceden elenmiş iç boşluğa sahip yerleşim dizilimlerini de hesaplamaya dahil etmektir.

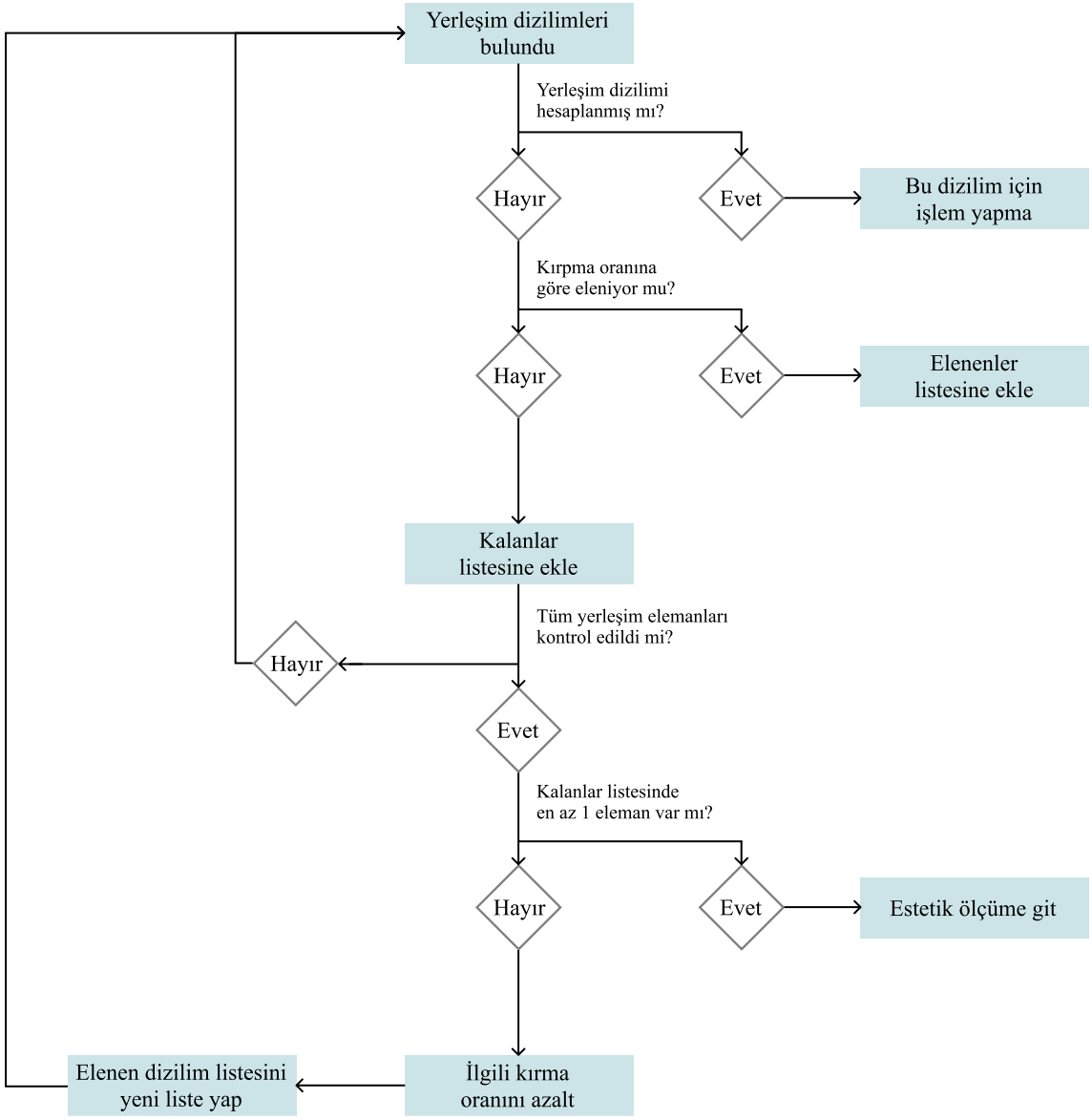


Şekil 4.5. Tam sığan sınıflandırması için bulunan yerleşim dizilimlerini eleme algoritmasının çalışma şekli.

4.3.2. Kırpma Oranları ile Eleme

Kırpma oranları 3. bölümde de bahsedildiği üzere, gerekli işlemler yapılarak kabul edilebilir minimum genişlik ve yükseklik değerlerinin bulunması ve bu değerlerin altında kalan aralık değerlerinin elenmesi ile gerçekleştirilir. Elenen yerleşim dizilimleri elenmişler isimli bir listede tutulurken kalan dizilimler kalanlar isimli bir listeye aktarılır. En az 1 tane kalan yerleşim dizilimi bulunması durumunda, son yerleşimler isimli diziye aktarılarak estetik ölçüme tabi tutulur.

Kalan yerleşim dizilimi listesinde eleman bulunmaması durumunda, kırpma oranları azaltılarak elenmişler listesi tekrar kontrol edilir ve en az 1 tane kalan yerleşim dizilimi bulana kadar bu adımlar tekrarlanır. Sınıflandırmanın boşluklu olduğu durumlarda, boşluk kırpma oranı üzerinden yükseklik ve genişliğe göre; sınıflandırmanın dikey kaydırma olduğu durumlarda, dikey kaydırma kırpma oranı üzerinden yüksekliğe göre; sınıflandırmanın yatay kaydırma olduğu durumlarda, yatay kaydırma kırpma oranı üzerinden genişliğe göre; sınıflandırmanın hem yatay hem dikey kaydırma olduğu durumlarda ise hem yatay kaydırma hem dikey kaydırma kırpma oranı üzerinden genişlik ve yüksekliğe göre yapılır.



Şekil 4.6. Kırpma oranı ile yerleşim dizilimlerini eleme algoritmasının çalışma şekli.

Kırpma elemesi, uygun yerleşim dizilimleri bulunduktan sonra yapılmak yerine, `GetRanges` algoritması içerisinde yapılabilseydi şüphesiz hesaplanan birçok alt aralık değerinin elenmesini sağlayarak, bulunacak yerleşim dizilim sayısını, algoritmanın önceki aşamalarında azaltmış olurdu. Lakin bu elemenin `GetRanges` algoritması içerisinde yapılamamasının sebebi, daha önce de bahsedilen tam sığan ve boşluklu sınıflandırmaya ait yerleşim dizilimlerinin ayırt edilememesiyle aynıdır.

Algoritma 2: Bulunan yerleşim dizilimlerini eleyen ELIMINATELAYOUTS algoritması.

Input: Bulunan yerleşim dizilimlerini tutan WidthHeightRange tipinde ExtendedArray *finalLayouts* değişkeni

Output: Eleme yapıldıktan sonra kalan yerleşim dizilimleri

remainingLayouts ← ∅ WidthHeightRange tipinde ExtendedArrayList

eliminatedLayouts ← ∅ WidthHeightRange tipinde ExtendedArrayList

while *remainingLayouts*.LENGTH = 0 **do**

  foreach *layout in finalLayouts* **do**

    if *layout* ≠ ∅ **then**

      if tam sığanlar sınıflandırması **and** *layout* tam sığan kurallarını sağlıyor **then**

        layout objesini *remainingLayouts*'a ekle

      else

        if *layout* kırpma değerlerine göre elenmiyor **then**

          layout objesini *remainingLayouts*'a ekle

        else

          layout objesini *eliminatedLayouts*'a ekle

  if *remainingLayouts*.LENGTH = 0 **then**

    if tam sığanlar sınıflandırması **then**

      sınıflandırmayı boşluklu olarak değiştir

      eliminatedLayouts ← ∅

      finalLayouts ← GETRANGES

    else

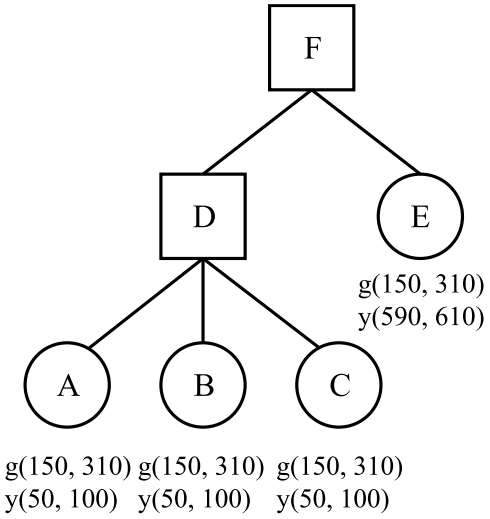
      sınıflandırmaya göre kırpma oranını azalt

      finalLayouts ← *eliminatedLayouts*

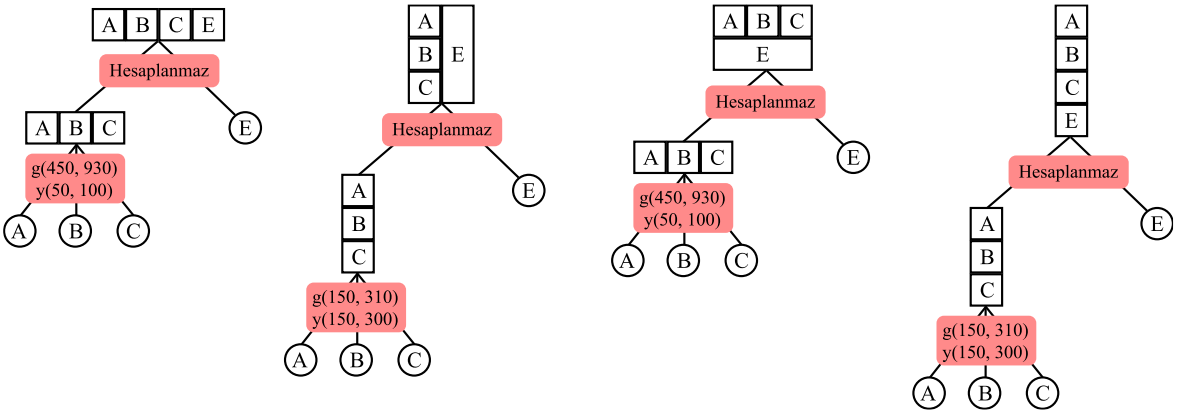
return *remainingLayouts*

`GetRanges` algoritması çalışırken, yerleşim ağacının kök düğümünden başlar ve bu düğümü hesaplayabilmek için yapraklara yani bileşenlere kadar iner. Bileşenlerden ve çocuk kapsayıcılardan gelen alt aralık değerlerinin, özellikle yerleşim ağacının yapraklarına yakın olan düğümlerinde, kabul edilebilir yükseklik ve genişlik değerlerinden daha küçük olması muhtemeldir. Eğer kırpma elemesi bu algoritmanın içine konulursa hesaplanması gereken birçok alt aralık hatalı olarak elenecektir.

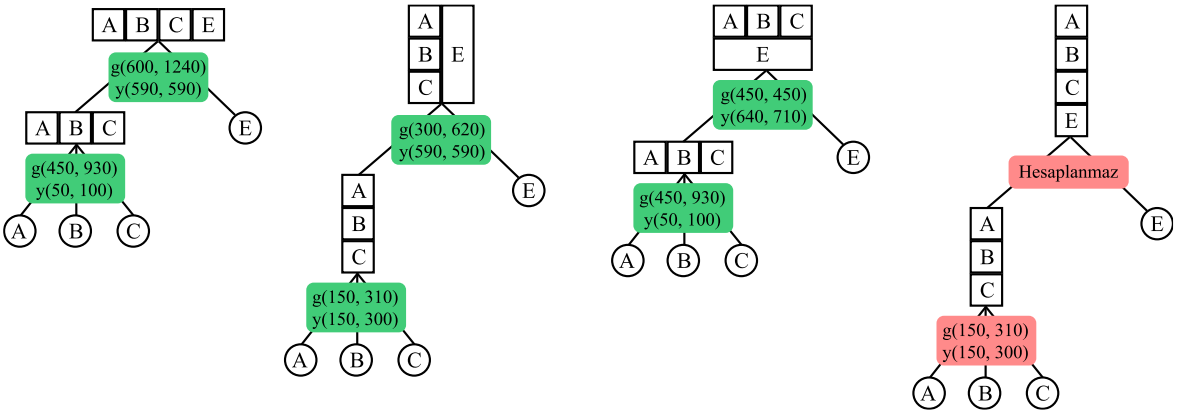
Şekil 4.7(a)'daki ağaç ele alındığında, hesaplanacak yerleşim dizilimleri 700x700 boyutlarında bir ekrana çizdirilmek istenirse ve boşluk kırpma oranı 5 olarak girilirse, 700x0.5 işleminden minimum genişlik ve yükseklik değeri 350 olarak hesaplanacaktır. Ağacın alt aralık değerlerine bakıldığı zaman D kapsayıcısının aralık değerlerinin tamamı bu değerin altında kalıp elenmektedir (*Şekil 4.7(b)*). Oysa hesaplama bir üst kapsayıcı olan E kapsayıcısına aktarıldığı zaman aralık değerlerinden 3 tanesinin kırpma değerlerine uygun olduğu ve elenmemesi gerektiği görülmektedir (*Şekil 4.7(c)*). Bu hususlarda ortaya çıkan hatalı elemeyi önlemek için kırpma elemesi `GetRanges` algoritmasından sonra çalışır.



Şekil 4.7.(a) Hatalı kırpma örneğinde kullanılan yerleşim ağacının yapısı.



Şekil 4.7.(b) *Şekil 4.7(a)*'daki ağaca göre, 700x700 boyutunda bir ekrana yerleştirilirken, GetRanges algoritması içinde kırpma elemesi yapılınca hesaplanan hatalı sonuç.



Şekil 4.7.(c) *Şekil 4.7(a)*'daki ağacın, 700x700 boyutunda bir ekrana yerleştirilirken, GetRanges algoritması dışında çalışınca hesapladığı sonuçlar.

5. BULUNAN DİZİLİMLERİNİN YERLEŞTİRİLMESİ

5.1. Konum, Boyut ve Sıralama Bilgilerini Atama

Uygun yerleşim dizilimleri arasından kullanıcıya gösterilecek son yerleşim dizilimi bulunur. Son yerleşim diziliminin bulunma yöntemine 6. bölümde değinilecektir. Bulunan son yerleşim diziliminin yerleştirmesi yapılmaktadır. Yerleştirme işleminde ağaç içerisinde bulunan her bir kapsayıcı ve bileşen düğüm için atanmış x, y, genişlik, yükseklik değerleri ve kapsayıcılar için ayrıca atanmış sıralama değeri bulunur. x ve y değerleri o kapsayıcı veya bileşenin sol üst köşesinin çizdirilecek pencere içindeki konumunu, genişlik ve yükseklik değeri genişliğini ve yüksekliğini, sıralama değeri ise kapsayıcının yatay ya da dikey olduğu bilgisini tutar. Arayüzü ekrana çizme aşamasında kapsayıcılar gösterilmeyeceği için yerleştirme işlemi bittiğinde sadece bileşenlerin atanmış değerleri önem arz etmektedir.

Yerleştirme işlemi `GetFinalLayout` algoritması ile yapılır. Algoritma yerleşim ağacının kök düğümünden başlar ve yapraklara yani bileşen düğümlere kadar her bir kapsayıcı ve bileşen için hesaplanarak ilerler. Kök düğümü için atanmış x ve y değerleri 0'dan başlar. Bunun anlamı kök düğümün sol üst köşesinin pencere içinde (0, 0) noktasına denk geldiğidir. Kalan düğümlere gönderilecek x, y değerleri kök düğüm ve sonraki çocuk kapsayıcılarda yapılacak dağıtım hesaplamalarına göre karar verilir.

Algoritma 3: Kapsayıcılar için son dizilimi yerleştiren GETFINALLAYOUT algoritması.

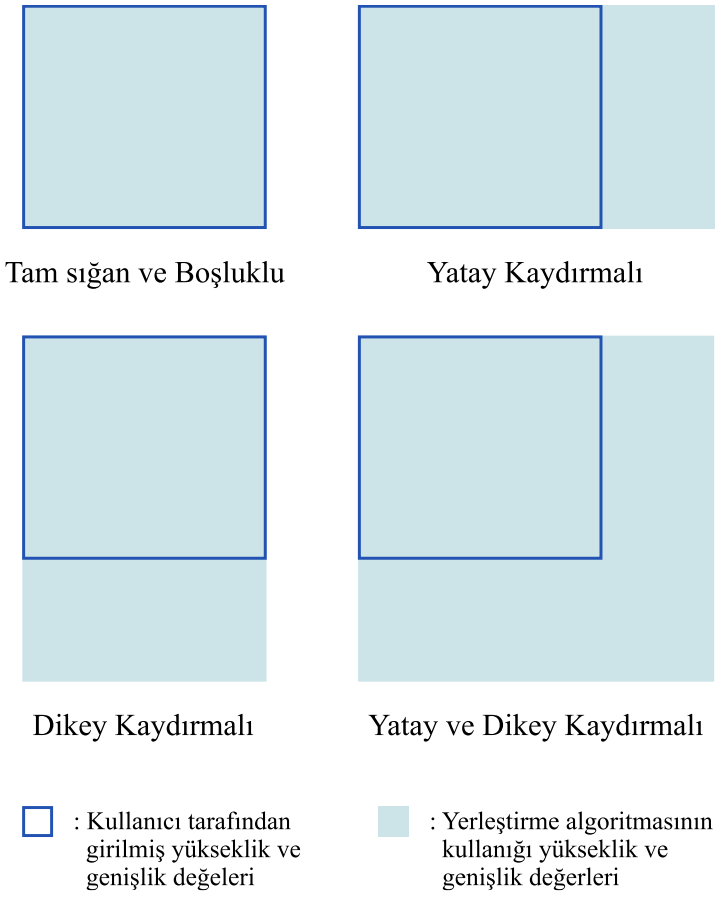
Input: *x*, sol üst köşenin yatay koordinatı, *y*, sol üst köşenin dikey koordinatı
w, genişlik, *h*, yükseklik, *range*, alt aralıkları da tutan WidthHeightRange nesnesi

subRanges ← *ranges*.GETSUBRANGES
if yatay sıralama **then**
    remaining ← *w*
    minValues ← *range*.GETMINWIDTHVALUES
    maxValues ← *range*.GETMAXWIDTHVALUES
else if dikey sıralama **then**
    remaining ← *h*
    minValues ← *range*.GETMINHEIGHTVALUES
    maxValues ← *range*.GETMAXHEIGHTVALUES
distribution ← STRATEGYBALANCE(*remaining, minValues, maxValues*)
if yatay sıralama **then**
    fillerWidth ← *w* – *range*.GETMAXWIDTH

```
    fillerUnit ← fillerWidth / children.LENGTH
else if dikey sıralama then
    fillerHeight ← h – range.GETMAXHEIGHT
    fillerUnit ← fillerHeight / children.LENGTH
distance ← 0
index ← 0
foreach child in children do
    if yatay sıralama then
        child.GETFINALLAYOUT(x + distance, y, distribution[index], h, subRanges[index])
    else if dikey sıralama then
        child.GETFINALLAYOUT(x, y + distance, w, distribution[index],subRanges[index])
    distance ← distance + distribution[index]
    index ← index + 1
```

Ağacın kök düğümüne atanacak genişlik ve yükseklik değeri yapılan sınıflandırmaya göre değişiklik gösterebilir. Sınıflandırmanın tam sığan ve boşluklu olduğu durumlarda kullanıcı tarafından girilen genişlik ve yükseklik değerleri aynen alınır. Dikey kaydırma olduğu durumda son yerleşim diziliminin yükseklik değeri, yatay kaydırma olduğu durumda ise son yerleşim diziliminin genişlik değeri alınır.

Kapsayıcıların atanmış sıralama bilgisi, bulunan son yerleşim diziliminden gelir. Yerleşim dizilimi, 3. bölümde de gösterildiği gibi `WidthHeightRange` sınıfına ait bir objedir ve yerleştirme ağacına ait kök düğümün ve diğer kapsayıcı düğümlerin sıralama bilgisi bu objenin ve alt aralıklarının sıralama bilgisine göre atanır.



Şekil 5.1. Farklı sınıflandırma türlerine göre kullanıcı tarafından girilen boyut değerleri ve yerleştirme algoritmasının kullandığı boyut değerlerinin karşılaştırılması.

5.2. Çocuk Düğümlere Dağıtım

İlgili değerler atandıktan sonra alt aralıklar ve her bir çocuğun maksimum ve minimum değerleri alınır. Maksimum ve minimum değerleri yatay sıralama için genişlik aralığının değerleri, dikey sıralama için yükseklik aralığının değerleridir. Çocuk düğümlere gerekli atamaların yapılabilmesi için yatay sıralama için kapsayıcının atanmış genişliği, dikey sıralama için kapsayıcının atanmış yüksekliği çocuklara dağıtılır. Dağıtılacak bu değere kalan değer denir ve Çelik vd.[4]'nin çalışmasında yaptığı gibi dağıtım "denge stratejisi" ne göre yapılır.

Denge stratejisine göre bütün çocuklar önce kendilerinin alabilecekleri en küçük minimum değerleri alırlar. Bu değerler kapsayıcının sıralamasına göre genişlik veya yükseklik olabilir. Her bir çocuğun aldığı değer kalan değerden çıkarılır. Kalan değer çocuk sayısına bölünerek her bir çocuğa eklenmesi gereken ek değerler bulunur. Ek değerlerin herhangi bir çocuğun maksimum değerini geçmesi durumunda aradaki fark diğer çocuklara dağıtılır. Bu işlem

kalan değer kalmayana veya bütün çocuklar maksimum değere ulaşana kadar devam eder. Çocuklarının genişlik aralıklarının (50-100), (150-250), (100-300) olduğu bir kapsayıcı 540 atanmış genişlik değerine sahipse, kalan değer en başta 550 olacak şekilde çocukların minimum değerleri atanmaya başlar;

Adım 1. Birinci çocuk 50, ikinci çocuk 150, üçüncü çocuk 100, kalan değer 240 olarak güncellenir.

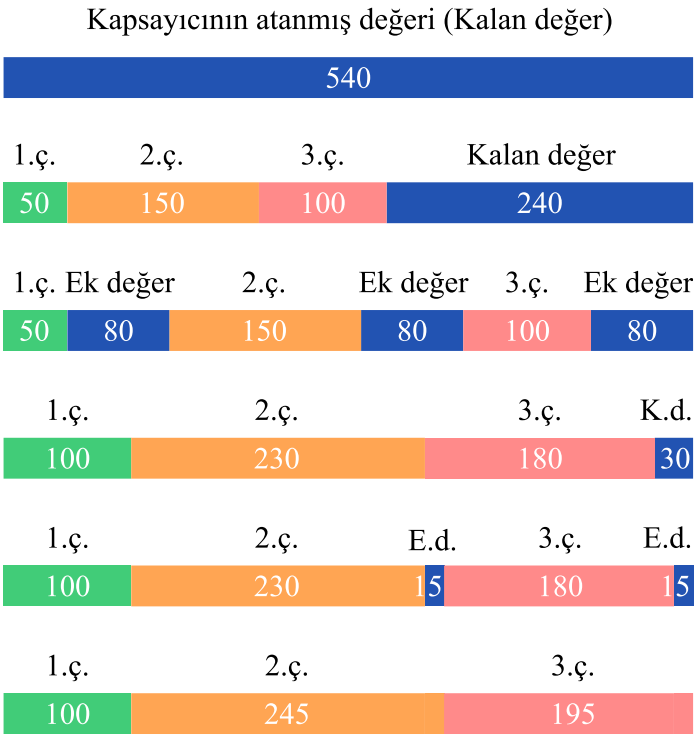
Adım 2. Kalan 240 değeri her bir çocuğa 240 / 3 = 80 ek değer verecek şekilde paylaştırılır. Birinci çocuk alabileceği maksimum değer 100 olduğu için 80 ek değerin yalnızca 50'sini alır ve kalan 30'u kalan değere aktadır. Son durumda birinci çocuk 100, ikinci çocuk 230, üçüncü çocuk 180 olarak güncellenir.

Adım 3. Kalan 30 değeri her bir çocuğa 30 / 2 = 15 ek değer verecek şekilde paylaştırılır. İkinci çocuk 245, üçüncü çocuk 195, kalan değer 0 olarak güncellenir.

Son durumda kalan değer 0 ve birinci çocuk 100, ikinci çocuk 245 ve üçüncü çocuk 195 değerlerini alır. Aynı kapsayıcı 1000 atanmış genişlik değerine sahip olsaydı, çocuklar maksimum değerlerini, yani birinci çocuk 100, ikinci çocuk 250 ve üçüncü çocuk 300 değerini alacak ve kalan değer olan 350 de boşluk değeri olarak alt çocuklara aktarılacaktı.

Dağılım algoritması yapılırken kapsayıcının atanmış değeri, çocukların maksimum değerinin toplamından büyük olabilse de, ki bahsedildiği üzere bu boşlukların olma durumudur, çocukların minimum değerlerinden küçük olamaz. Bu olası durum kaydırma olan yerleşim dizilimlerinde ortaya çıkabilecek bir durum iken `GetFinalLayout` algoritmasına başlarken kök düğümün atanmış değerleri kullanıcıdan alınan değerler yerine o yerleşim diziliminin minimum değerlerine göre alındığı için bu ihtimalin önüne geçilmiştir.

Denge stratejisine göre dağıtım algoritmasında Çelik vd.[4]'nin çalışmasından farklı olarak kalan değerin 0 olmasına gerek yoktur. Kalan değerin 0 olmadığı durumlar o kapsayıcıda boşlukların olduğunu gösterir. Dağılım bittikten sonra kalan değer, yani boşluk değeri bütün çocuklara eşit olacak şekilde paylaştırılır ve boşluk değerleri bileşenlere kadar ilerler. Boşlukların nasıl konumlandırılacağı bileşenlerin atanan değerleri hesaplanırken bulunur.



Şekil 5.2. Denge stratejisi ile kapsayıcının atanmış değerlerinin çocuklarına dağıtılması örnek adımları.

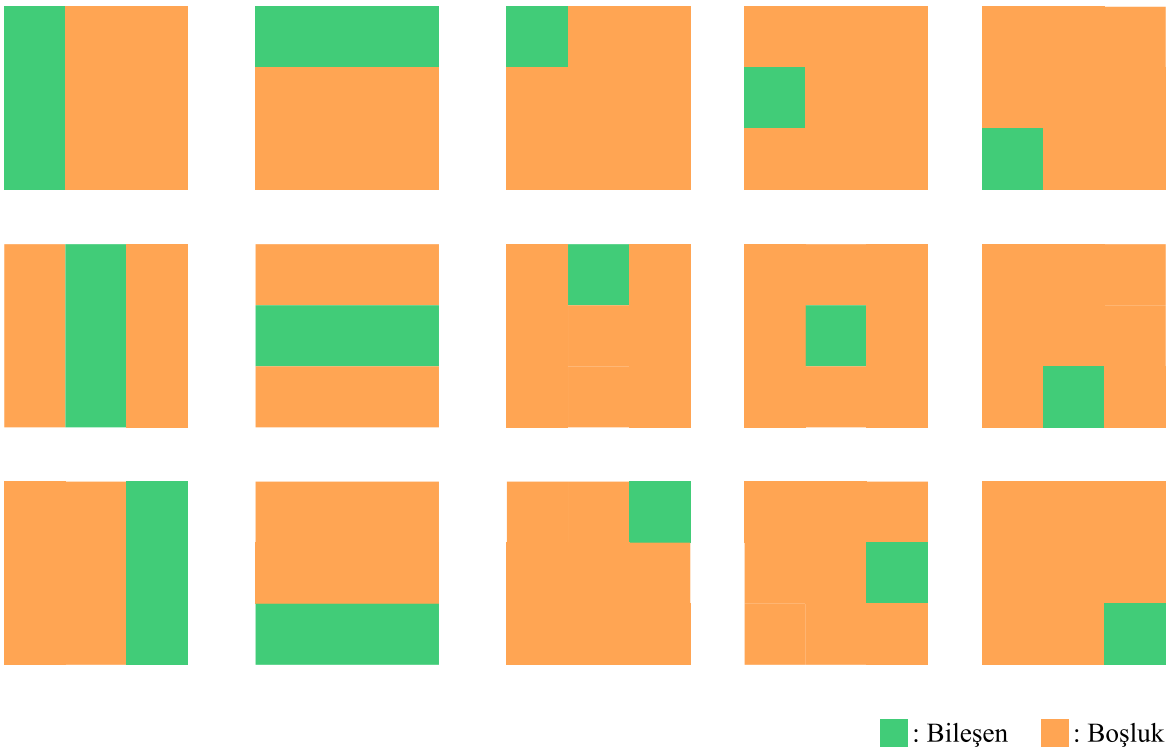
5.3. Bileşenleri Yerleştirme Algoritması

5.3.1. Bileşenlerin Atanan Değerlerinin Hesaplanması

`GetFinalLayout` algoritması bileşenler için çalışırken önce boşluğa ihtiyaç olup olmadığı kontrol eder. Boşluğa ihtiyaç olmadığı durumlarda içinde bulunduğu kapsayıcıdan aldığı x, y, genişlik ve yükseklik değerini direkt alarak estetik veri hesaplamalarına geçer.

Boşluğa ihtiyacı kontrolü bileşenin içinde bulunduğu kapsayıcıdan gelen genişlik ve yükseklik değerleri, bileşenin alabileceği maksimum genişlik ve yükseklik değerleri ile karşılaştırılarak bulunur. Kapsayıcıdan gelen değerler, maksimum değerlerden büyükse boşluğa ihtiyaç vardır. Boşluk objesinin değerlerini bulmak için kapsayıcıdan gelen değerlerle maksimum değerlerin farkı alınır. Kapsayıcıdan sadece yatay veya sadece dikey boşluk geliyorsa, oluşturulacak boşluğun sırayla dikey ve yatay değerleri bileşen ile aynıdır.

Kullanıcıdan alınan yatay ve dikey hizalama bilgilerine göre, boşluklar bölünebilir ve yerleri değişebilir. Eğer kullanıcıdan hizalama bilgisi olarak başlangıç alınmışsa, bileşen başa, boşluk sona; son alınmışsa bileşen sona, boşluk başa; orta alınmışsa bileşen ortaya, boşluk da iki parçaya bölünerek başa ve sona eklenir.

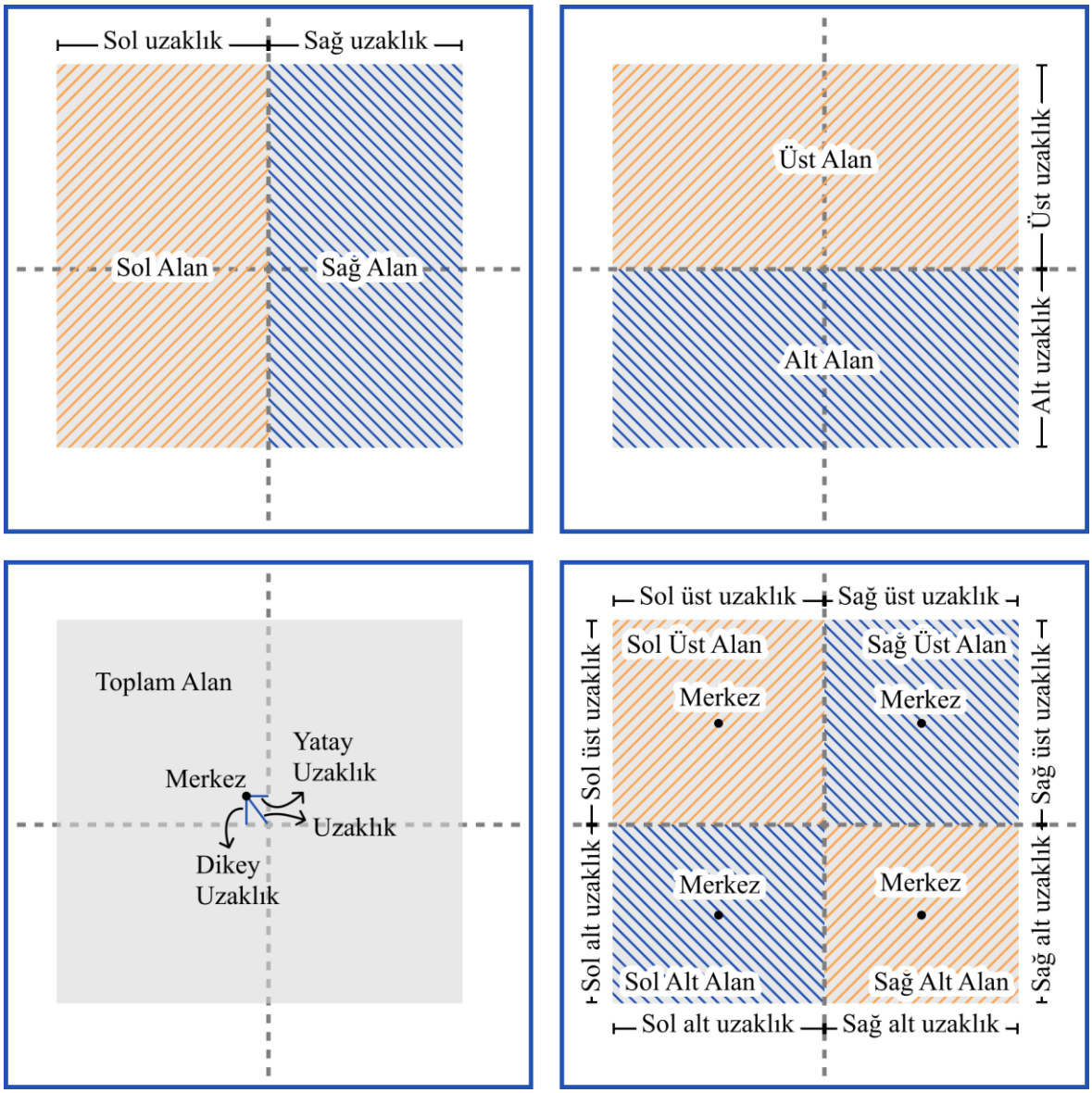


Şekil 5.3. Farklı hizalama seçenekleri ile ortaya çıkan farklı dizilimler.

Boşluklar `JComponent` objesi olmayan bileşenler olarak oluşturulup, boşluklar isimli bir listede tutulurlar. Boşluk bileşenlerini oluşturmanın ve tutmanın amacı estetik hesaplamalarda kullanılacak olmalarıdır.

5.3.2. Estetik Ölçüme Hazırlık

Sınıflandırmalar ve kırpma oranları ile kullanıcı tarafından girilen yükseklik ve genişlik değerine uyan çizdirilebilir yerleşim dizilimleri bulunduktan sonra bunlar arasından hangisinin kullanıcıya gösterilecek son arayüz yerleşimi olduğuna karar vermek gerekir. Bütün yerleşim dizilimleri, estetik ölçüm denilen bir sistemle puanlanırlar. Bu ölçüm sisteminin detaylarına 6. bölümde değinilecektir. Yerleşim dizilimlerinin estetik ölçüme tabi tutulabilmesi için önce o dizilim için bileşenlerin belirli verilerine sahip olmak gerekir. Bu belirli verileri hesaplamak için bulunan her bir uygun yerleşim dizilimi için `GetFinalLayouts` algoritmasını çalıştırılır. Algoritma bileşenler için hesaplamaya geldiğinde gerekli estetik ölçüm verilerini de hesaplar. Bu şekilde kontrol edilen yerleştirme algoritması için Estetik ölçüm yapılırken tekrar bu verileri hesaplamakla uğraşılmaz. Bileşenlerin her biri için 1 `ComponentData` objesi ve 4 `QuadrantData` objesi hesaplanır.



Şekil 5.4. Estetik ölçüm için hesaplanan bileşen verileri.

6. ESTETİK ÖLÇÜM

Algoritma, bulduğu birçok yerleştirme dizilimlerinden hangisini kullanıcıya göstereceğine karar vermek için bütün uygun dizilimleri estetik olarak ölçer. Estetik ölçümde Ngo vd. [8] tarafından önerilen metrikler ve formülleri ağırlıklar verilerek kullanılmıştır.

6.1. Denge

Denge, bir ekrandaki bileşenlerin şekil, renk ve alan bakımından dağılımını gösterir. Ngo vd.[8]'ne göre koyu renkler, karmaşık şekiller ve geniş alanlar daha ağır; açık renkler, basit şekiller ve küçük alanlar daha hafiftir. Bu çalışmada sadece alan hesaplaması kullanılmıştır. Denge, bileşenlerin yatay ve dikey denge ağırlıklarının aritmetik ortalaması alınarak bulunur. Denge formülü, aşağıda belirtildiği gibi hesaplanır [8]:

L, R, T, B sırayla sol, sağ, üst ve alt, a_{ij} i bileşeninin j bölgesindeki alanı, d_{ij} bileşenin ve çerçevenin merkez çizgileri arasındaki uzaklık, n_j j bölgesindeki toplam bileşen sayısı olmak üzere;

$$BM = 1 - \frac{\left|BM_{dikey}\right| + \left|BM_{yatay}\right|}{2} \in [0,1] \tag{1}$$

$$BM_{dikey} = \frac{w_L - w_R}{maks(|w_L|, |w_R|)} \tag{2}$$

$$BM_{yatay} = \frac{w_T - w_B}{maks(|w_T|, |w_B|)} \tag{3}$$

$$w_j = \sum_{n_j}^{i} d_{ij} \frac{a_{ij}}{a_{maks}}, \qquad j = L, R, T, B \tag{4}$$

$$a_{maks} = maks(a_{ij}, i = 1, 2, \dots, n_j \quad j = L, R, T, B) \tag{5}$$

6.2. Eşitlik

Eşitlik, ekranda bulunan bileşenleri kapsayan alanın merkezinin, ekranın merkeziyle olan durumunu ifade eder. Bu çalışmada ekran ölçüleri kullanıcı tarafından girilen genişlik ve yükseklik değeri olarak kabul edilmiştir. Her bir bileşenin yatay ve dikey ağırlık merkezlerine göre hesaplanan eşitliğinin aritmetik ortalaması alınarak bulunur. Eşitlik formülü, aşağıda belirtildiği gibi hesaplanır [8]:

(x_i, y_i) ve (x_c, y_c) sırayla i bileşeninin ve çerçevenin merkez koordinatları, a_i i bileşeninin alanı, $b_{çerçeve}$ ve $h_{çerçeve}$ çerçevenin genişlik ve yüksekliği ve *n* çerçevedeki bileşen sayısı olmak üzere;

$$EM = 1 - \frac{|EM_x| - |EM_y|}{2} \in [0, 1] \tag{6}$$

$$EM_x = \frac{2\sum_i^n a_i(x_i - x_c)}{nb_{çerçeve}\sum_i^n a_i} \tag{7}$$

$$EM_y = \frac{2\sum_i^n a_i(y_i - y_c)}{nh_{çerçeve}\sum_i^n a_i} \tag{8}$$

6.3. Simetri

Simetri, ekran merkezi çizgiler ile bölündüğünde bir alanın diğer alanın birebir yansıması olma durumudur. Yatay, dikey ve çapraz olmak üzere 3 farklı yönden hesaplanan simetrinin aritmetik ortalaması alınır. Simetri formülü, aşağıda belirtildiği gibi hesaplanır [8]:

UL, UR, LL, LR sırayla sol-üst, sağ-üst, sol-alt, sağ-alt, (x_{ij}, y_{ij}) ve (x_c, y_c) sırayla j bölgesindeki i bileşeninin ve çerçevenin merkez koordinatları, b_{ij} ve h_{ij} bileşenin j bölgesindeki genişlik ve yüksekliği ve n_j j bölgesindeki toplam bileşen sayısı olmak üzere;

$$SYM = 1 - \frac{|SYM_{dikey}| + |SYM_{yatay}| + |SYM_{çapraz}|}{3} \in [0, 1] \tag{9}$$

X_j', Y_j', H_j', B_j', θ_j' ve R_j' sırayla X_j, Y_j, H_j, B_j, θ_j ve R_j'nin normalleştirilmiş değerleri olmak üzere;

$$SYM_{dikey} = \frac{\begin{matrix}|X'_{UL} - X'_{UR}| + |X'_{LL} - X'_{LR}| + |Y'_{UL} - Y'_{UR}| + |Y'_{LL} - Y'_{LR}| + \\ |H'_{UL} - H'_{UR}| + |H'_{LL} - H'_{LR}| + |B'_{UL} - B'_{UR}| + |B'_{LL} - B'_{LR}| + \\ |\theta'_{UL} - \theta'_{UR}| + |\theta'_{LL} - \theta'_{LR}| + |R'_{UL} - R'_{UR}| + |R'_{LL} - R'_{LR}|\end{matrix}}{12} \tag{10}$$

$$SYM_{yatay} = \frac{\begin{matrix}|X'_{UL} - X'_{LL}| + |X'_{UR} - X'_{LR}| + |Y'_{UL} - Y'_{LL}| + |Y'_{UR} - Y'_{LR}| + \\ |H'_{UL} - H'_{LL}| + |H'_{UR} - H'_{LR}| + |B'_{UL} - B'_{LL}| + |B'_{UR} - B'_{LR}| + \\ |\theta'_{UL} - \theta'_{LL}| + |\theta'_{UR} - \theta'_{LR}| + |R'_{UL} - R'_{LL}| + |R'_{UR} - R'_{LR}|\end{matrix}}{12} \tag{11}$$

$$SYM_{çapraz} = \frac{\begin{matrix}|X'_{UL} - X'_{LR}| + |X'_{UR} - X'_{LL}| + |Y'_{UL} - Y'_{LR}| + |Y'_{UR} - Y'_{LL}| + \\ |H'_{UL} - H'_{LR}| + |H'_{UR} - H'_{LL}| + |B'_{UL} - B'_{LR}| + |B'_{UR} - B'_{LL}| + \\ |\theta'_{UL} - \theta'_{LR}| + |\theta'_{UR} - \theta'_{LL}| + |R'_{UL} - R'_{LR}| + |R'_{UR} - R'_{LL}|\end{matrix}}{12} \tag{12}$$

$$X_j = \sum_i^{n_j} \left| x_{ij} - x_c \right| , \qquad j = UL, UR, LL, LR \tag{13}$$

$$Y_j = \sum_i^{n_j} \left| y_{ij} - y_c \right| , \qquad j = UL, UR, LL, LR \tag{14}$$

$$H_j = \sum_i^{n_j} h_{ij} , \qquad j = UL, UR, LL, LR \tag{15}$$

$$B_j = \sum_i^{n_j} b_{ij} , \qquad j = UL, UR, LL, LR \tag{16}$$

$$\theta_j = \sum_i^{n_j} \left| \frac{y_{ij} - y_c}{x_{ij} - x_c} \right| , \qquad j = UL, UR, LL, LR \tag{17}$$

$$R_j = \sum_i^{n_j} \sqrt{(x_{ij} - x_c)^2 + (y_{ij} - y_c)^2} , \qquad j = UL, UR, LL, LR \tag{18}$$

6.4. Sıralanma

Kişilere bir ekran gösterildiğinde bazı metriklere göre, dikkatini çeken ilk yer ve son yer değişebilir. Araştırmalar insan gözünün büyük nesnelerden küçüklere, canlı renklerden soluklara, siyahtan beyaza ve düzensiz şekillerden düzenli şekillere hareket ettiğini göstermektedir [8]. Sıralanma, gözün bu hareketini kolaylaştıran yerleştirmeleri ölçen bir metriktir. Sıralanma formülü, aşağıda belirtildiği gibi hesaplanır [8]:

UL, *UR*, *LL*, *LR* sırayla sol-üst, sağ-üst, sol-alt, sağ-alt, a_{ij} i bileşeninin j bölgesindeki alanı olmak üzere;

$$SQM = 1 - \frac{\sum_{j=UL,UR,LL,LR} \left| q_j - v_j \right|}{8} \in [0, 1] \tag{19}$$

$$\{q_{UL}, q_{UR}, q_{LL}, q_{LR}\} = \{4, 3, 2, 1\} \tag{20}$$

$$v_j = \begin{cases} 4 & , w_j, w \text{ içindeki en büyük ise} \\ 3 & , w_j, w \text{ içindeki ikinci en büyük ise} \\ 2 & , w_j, w \text{ içindeki üçüncü en büyük ise} \\ 1 & , w_j, w \text{ içindeki en küçük ise} \end{cases} \qquad j = UL, UR, LL, LR \tag{21}$$

$$w_j = q_j \sum_{i}^{n_j} a_{ij} \tag{22}$$

6.5. Bağıntı

Bağıntı, bileşenler, bileşenleri kapsayan alan ve ekranın genişlik-yükseklik oranları arasındaki benzerliğe göre ölçülür. Boyuna uzun olan bir ekran ile boyuna uzun olan bir bileşen arasında daha yüksek bir bağıntı ölçülürken, boyuna uzun olan bir ekran ile enine geniş bir bileşen arasında daha düşük bir bağıntı ölçülecektir. Fakat pratikte bakıldığı zaman bu formül bazı istisnalarla karşılaşır. Bir telefon ekranı ele alındığında, telefon ekranı boyuna uzun iken, ekranda görünen bir butonun enine geniş olması yaygın görünen bir durumdur ve kötü bir tasarım değildir. Aksine boyuna uzun bir buton kullanıcı deneyimi için kötü bile olabilir. Bu sebeple bu metriğin katsayısı düşük tutulmuştur. Bağıntı formülü, aşağıda belirtildiği gibi hesaplanır [8]:

b_i ve h_i , $b_{yerleşim}$ ve $h_{yerleşim}$, $b_{çerçeve}$ ve $h_{çerçeve}$ sırayla i bileşeninin, bileşenleri kapsayan en küçük dörtgenin ve çerçevenin genişlik ve yüksekliği ve *n* çerçevedeki bileşen sayısı olmak üzere;

$$CM = \frac{|CM_{fl}| + |CM_{lo}|}{2} \in [0,1] \tag{23}$$

$$CM_{fl} = \begin{cases} t_{fl} & , t_{fl} \leq 1 \; ise \\ \dfrac{1}{t_{fl}} & , diğer \end{cases} \tag{24}$$

$$CM_{lo} = \frac{\sum_i^n f_i}{n} \tag{25}$$

$$t_{fl} = \frac{h_{yerleşim} / b_{yerleşim}}{h_{çerçeve} / b_{çerçeve}} \tag{26}$$

$$f_i = \begin{cases} t_i & , t_i \leq 1 \; ise \\ \dfrac{1}{t_i} & , diğer \end{cases} \tag{27}$$

$$t_i = \frac{h_i / b_i}{h_{yerleşim} / b_{yerleşim}} \tag{28}$$

6.6. Birlik

Birlik, bileşenlerin bir arada bulunup tek parça olma halini ölçen metriktir. Bileşenlerin arasındaki boşluğun bileşenlerle çerçeve arasındaki boşluktan fazla olması durumunda düşük, az olması durumunda yüksek ölçülür. Bu çalışmada bileşenlerin kendi arasında ve çerçeve ile aralarında hiçbir boşluk bulunmaması durumunda *1* olarak ölçülür. Bu durum sadece tam uygun yerleşim dizilimlerinin bulunduğu sınıflandırmada ortaya çıkar. Birlik formülü, aşağıda belirtildiği gibi hesaplanır [8]:

a_i , $a_{yerleşim}$, $a_{çerçeve}$ sırayla i bileşeninin, bileşenleri kapsayan en küçük dörtgenin ve çerçevenin alanı, n_{boyut} boyutların sayısı ve *n* çerçevedeki bileşen sayısı olmak üzere;

$$UM = \frac{|UM_{form}| + |UM_{boşluk}|}{2} \in [0,1] \tag{29}$$

$$UM_{form} = 1 - \frac{n_{boyut} - 1}{n} \tag{30}$$

$$UM_{boşluk} = \begin{cases} 1 & , a_{çerçeve} = A\ ise \\ 1 - \dfrac{a_{yerleşim} - A}{a_{çerçeve} - A} & , diğer \end{cases} \tag{31}$$

$$A = \sum_{i}^{n} a_i \tag{32}$$

6.7. Orantı

Orantı, bileşenlerin boyutları ile insan gözüne hitap eden oranların arasındaki karşılaştırmalı ilişkidir. Ngo vd. [8]'ne göre göze hitap eden oranlar; *1:1, 1:1.414, 1:1.618, 1:1.732, 1:2* olarak alınmıştır. Orantı formülü, aşağıda belirtildiği gibi hesaplanır [8]:

b_i ve h_i , $b_{yerleşim}$ ve $h_{yerleşim}$ sırayla i bileşeninin ve bileşenleri kapsayan en küçük dörtgenin genişlik ve yüksekliği, p_j j şeklinin orantısı olmak üzere;

$$PM = \frac{|PM_{bileşen}| + |PM_{yerleşim}|}{2} \in [0,1] \tag{33}$$

$$PM_{bileşen} = \frac{1}{n}\sum_{i}^{n}\left(1 - \frac{min(|p_j - p_i|,\ j = sq, r2, gr, r3, ds)}{0.5}\right) \tag{34}$$

$$PM_{yerleşim} = 1 - \frac{min(|p_j - p_{yerleşim}|,\ j = sq, r2, gr, r3, ds)}{0.5} \tag{35}$$

$$\{p_{sq}, p_{r2}, p_{gr}, p_{r3}, p_{ds}\} = \left\{\frac{1}{1}, \frac{1}{1.414}, \frac{1}{1.618}, \frac{1}{1.732}, \frac{1}{2}\right\} \quad (36)$$

$$p_i = \begin{cases} r_i & , r_i \leq 1 \; ise \\ \dfrac{1}{r_i} & , diğer \end{cases} \quad (37)$$

$$p_{yerleşim} = \begin{cases} r_{yerleşim} & , r_{yerleşim} \leq 1 \; ise \\ \dfrac{1}{r_{yerleşim}} & , diğer \end{cases} \quad (38)$$

$$r_i = \frac{h_i}{b_i} \quad (39)$$

$$r_{yerleşim} = \frac{h_{yerleşim}}{b_{yerleşim}} \quad (40)$$

6.8. Sadelik

Sadelik, ekranda bulunan bileşenlerin basit ve yalın bir biçiminde dizilimini ifade eder. Ekranda bulunan bileşenlerin sayısı, yatay ve dikey hizalama noktalarının az olması durumunda yüksek, aksi durumda düşük ölçülür. Sadelik formülü, aşağıda belirtildiği gibi hesaplanır [8]:

n_{vap} ve n_{hap} yatay ve dikey hizalama noktalarının sayısı ve n çerçevedeki bileşen sayısı olmak üzere;

$$SMM = \frac{3}{n_{vap} + n_{hap} + n} \in [0, 1] \quad (41)$$

6.9. Yoğunluk

Yoğunluk, bileşenlerin ekranı ne dereceye kadar kapsadığını ölçen metriktir. Bileşenlerin alanları ve çizdirilen çerçevenin alanları oranlanarak bulunur. Yoğunluk formülü, aşağıda belirtildiği gibi hesaplanır [8]:

a_i ve $a_{çerçeve}$ i bileşeninin ve çerçevenin alanı olmak üzere;

$$DM = 1 - \frac{\sum_i^n a_i}{a_{çerçeve}} \in [0, 1] \quad (42)$$

6.10. Düzenlilik

Düzenlilik, bileşenler arasındaki boşlukların ölçüsü ve yatay ve dikey hizalama nokta sayısındaki değişmezlik ölçülerek bulunur. Diğer bir değişle bu parametreler arasındaki tutarlılığı ele alır. Belirtilen parametreler arasındaki çeşitlilik arttıkça düzen formülü ile ölçülen değer azalır. Düzenlilik formülü, aşağıda belirtildiği gibi hesaplanır [8]:

n_{vap} ve n_{hap} yatay ve dikey hizalama noktalarının sayısı, $n_{boşluk}$ sütun ve satırların başlangıç noktaları arasındaki uzaklık sayısı ve *n* çerçevedeki bileşen sayısı olmak üzere;

$$RM = \frac{|RM_{hizalama}| + |RM_{boşluk}|}{2} \in [0,1] \tag{43}$$

$$RM_{hizalama} = 1 - \frac{n_{vap} + n_{hap}}{2n} \tag{44}$$

$$RM_{boşluk} = \begin{cases} 1, & n = 1\ ise \\ 1 - \dfrac{n_{boşluk} - 1}{2(n-1)}, & diğer \end{cases} \tag{45}$$

6.11. Ekonomi

Ekonomi, şekil, renk ve boyut bakımından bileşenlerin sayısının ne kadar olduğu ile ilişkilidir. Renk, şekil ve boyut sayısı fazla olan yerleştirmeler daha düşük sonuç verirken, az olanlar daha yüksek sonuç verir. Bu çalışmada sadece boyut sayısı ele alınmıştır. Ekonomi formülü, aşağıda belirtildiği gibi hesaplanır [8]:

n_{boyut} boyutların sayısı olmak üzere;

$$ECM = \frac{1}{n_{boyut}} \in [0,1] \tag{46}$$

6.12. Homojenlik

Ekran koordinat düzlemine göre sol-üst, sağ-üst, sol-alt, sağ-alt olmak üzere dört parçaya ayrılmıştır. Homojenlik, bahsedilen bu dört koordinat parçasına bileşenlerin ne kadar eşit oranda dağıldığını ölçer. Homojenlik formülü, aşağıda belirtildiği gibi hesaplanır [8]:

n_{UL} , n_{UR} , n_{LL} , n_{LR} sırayla, sol-üst, sağ-üst, sol-alt, sağ-alt bölgelerindeki bileşenlerin sayısı ve *n* çerçevedeki bileşen sayısı olmak üzere;

$$HM = \frac{W}{W_{maks}} \in [0,1] \tag{47}$$

$$W = \frac{n!}{\prod_{j\,=\,UL,UR,LL,LR} n_j} = \frac{n!}{n_{UL}!\,n_{UR}!\,n_{LL}!\,n_{LR}!} \tag{48}$$

$$W_{maks} = \frac{n!}{\frac{n}{4}!\frac{n}{4}!\frac{n}{4}!\frac{n}{4}!} = \frac{n!}{\left(\frac{n}{4}!\right)^4} \tag{49}$$

6.13. Ritim

Ritim, bileşenlerin sistematik olarak ne kadar düzenli olduğunu ölçen metriktir. Yatay, dikey ve alan ritimleri hesaplanıp aritmetik ortalamaları alınarak bulunur. Ritim formülü, aşağıda belirtildiği gibi hesaplanır [8]:

UL, UR, LL, LR sırayla sol-üst, sağ-üst, sol-alt, sağ-alt, (x_{ij}, y_{ij}) ve (x_c, y_c) sırayla j bölgesindeki i bileşeninin ve çerçevenin merkez koordinatları, a_{ij} i bileşeninin j bölgesindeki alanı ve n_j j bölgesindeki toplam bileşen sayısı olmak üzere;

$$RHM = 1 - \frac{|RHM_x| - |RHM_y| - |RHM_{alan}|}{3} \in [0, 1] \tag{50}$$

X_j', Y_j' ve A_j' sırayla X_j, Y_j ve A_j'nin normalleştirilmiş değerleri olmak üzere;

$$RHM_x = \frac{\begin{matrix}|X'_{UL} - X'_{UR}| + |X'_{UL} - X'_{LR}| + \\ |X'_{UL} - X'_{LL}| + |X'_{UR} - X'_{LR}| + \\ |X'_{UR} - X'_{LL}| + |X'_{LR} - X'_{LL}|\end{matrix}}{6} \tag{51}$$

$$RHM_y = \frac{\begin{matrix}|Y'_{UL} - Y'_{UR}| + |Y'_{UL} - Y'_{LR}| + \\ |Y'_{UL} - Y'_{LL}| + |Y'_{UR} - Y'_{LR}| + \\ |Y'_{UR} - Y'_{LL}| + |Y'_{LR} - Y'_{LL}|\end{matrix}}{6} \tag{52}$$

$$RHM_{alan} = \frac{\begin{matrix}|A'_{UL} - A'_{UR}| + |A'_{UL} - A'_{LR}| + \\ |A'_{UL} - A'_{LL}| + |A'_{UR} - A'_{LR}| + \\ |A'_{UR} - A'_{LL}| + |A'_{LR} - A'_{LL}|\end{matrix}}{6} \tag{53}$$

$$X_j = \sum_i^{n_j} \left|x_{ij} - x_c\right| , \qquad j = UL, UR, LL, LR \tag{54}$$

$$Y_j = \sum_i^{n_j} \left|y_{ij} - y_c\right| , \qquad j = UL, UR, LL, LR \tag{55}$$

$$A_j = \sum_{i}^{n_j} a_{ij} \ , \qquad j = UL, UR, LL, LR \tag{56}$$

6.14. Düzen ve Karmaşıklık

Ngo vd. [8]'nin yaptığı çalışmaya göre düzen ve karmaşıklık, hesaplanan diğer metriklerin aritmetik ortalamaları alınarak hesaplanır. Fakat zaman içinde bu metriklerden bazıları önemini yitirmiş, bazıları ise daha önemli hale gelmiştir. Dolayısıyla yapılan bu çalışmada metriklere ağırlıklar verilmiştir. Ağırlıklar; denge için *1.0*, eşitlik için *0.8*, simetri için *0.6*, sıralanma için *0.2*, bağıntı için *0.5*, birlik için *0.4*, orantı için *0.4*, sadelik için *0.6*, yoğunluk için *0.4*, düzenlilik için *0.1*, ekonomi için *0.5*, homojenlik için *0.1*, ritim için *0.2* olarak verilmiştir. Bu katsayılar literatürde incelenen çalışmalar ve metriklerin çalışmalarda ele alınma sıklığına göre hesaplanmıştır [9, 10, 11, 15, 16, 17]. Düzen ve karmaşıklık formülü, aşağıda belirtildiği gibi hesaplanır [8]:

$$OM = \frac{\sum_i^{13} k_i M_i}{\sum_i^{13} k_i} \in [0,1], \qquad 0 \leq k_i \leq 1 \tag{57}$$

$$\begin{aligned} M &= \{M_1, M_2, M_3, M_4, M_5, M_6, M_7, M_8, M_9, M_{10}, M_{11}, M_{12}, M_{13}\} \\ &= \{BM, EM, SYM, SQM, CM, UM, PM, SMM, DM, RM, ECM, HM, RHM\} \end{aligned} \tag{58}$$

$$\begin{aligned} k &= \{k_1, k_2, k_3, k_4, k_5, k_6, k_7, k_8, k_9, k_{10}, k_{11}, k_{12}, k_{13}\} \\ &= \{1.0, 0.8, 0.6, 0.2, 0.5, 0.4, 0.4, 0.6, 0.4, 0.1, 0.5, 0.1, 0.2\} \end{aligned} \tag{59}$$

7. TESTLER VE SONUÇLAR

7.1. Hız Testi

Hız testinde çalışmanın temel algoritmaları olan `GetRanges`, `EliminateLayouts`, `GetFinalLayouts` ve estetik ölçüm yapan `MeasureAesthetics` algoritmalarının çalışma hızları ölçülmüştür. `GetFinalLayouts` estetik ölçümü hesaplamak için bulunan her yerleşim dizilimi için çalıştırılmıştır. Ayrıca `GetRanges` algoritması tam sığan sınıflandırması için önceki bölümlerde de belirtiği üzere uygun dizilim bulamadığında tekrar çalışmıştır. Verilen süreler bu değerlerin toplam değeridir. Yapılan ölçümlerde `EliminateLayouts` algoritması çalışırken ilk etapta hiçbir kırpma oranı almadan çalışmış, ikinci bölümde boşluk kırpma oranı verilerek test edilemeyen senaryolar ele alınmıştır. Her senaryoda hizalama yatay ve dikey için başlangıç olarak alarak alınmıştır.

Testlerin yapıldığı bilgisayarın özellikleri;

İşletim Sistemi: Windows 11

Sistem Türü: 64 bit işletim sistemi

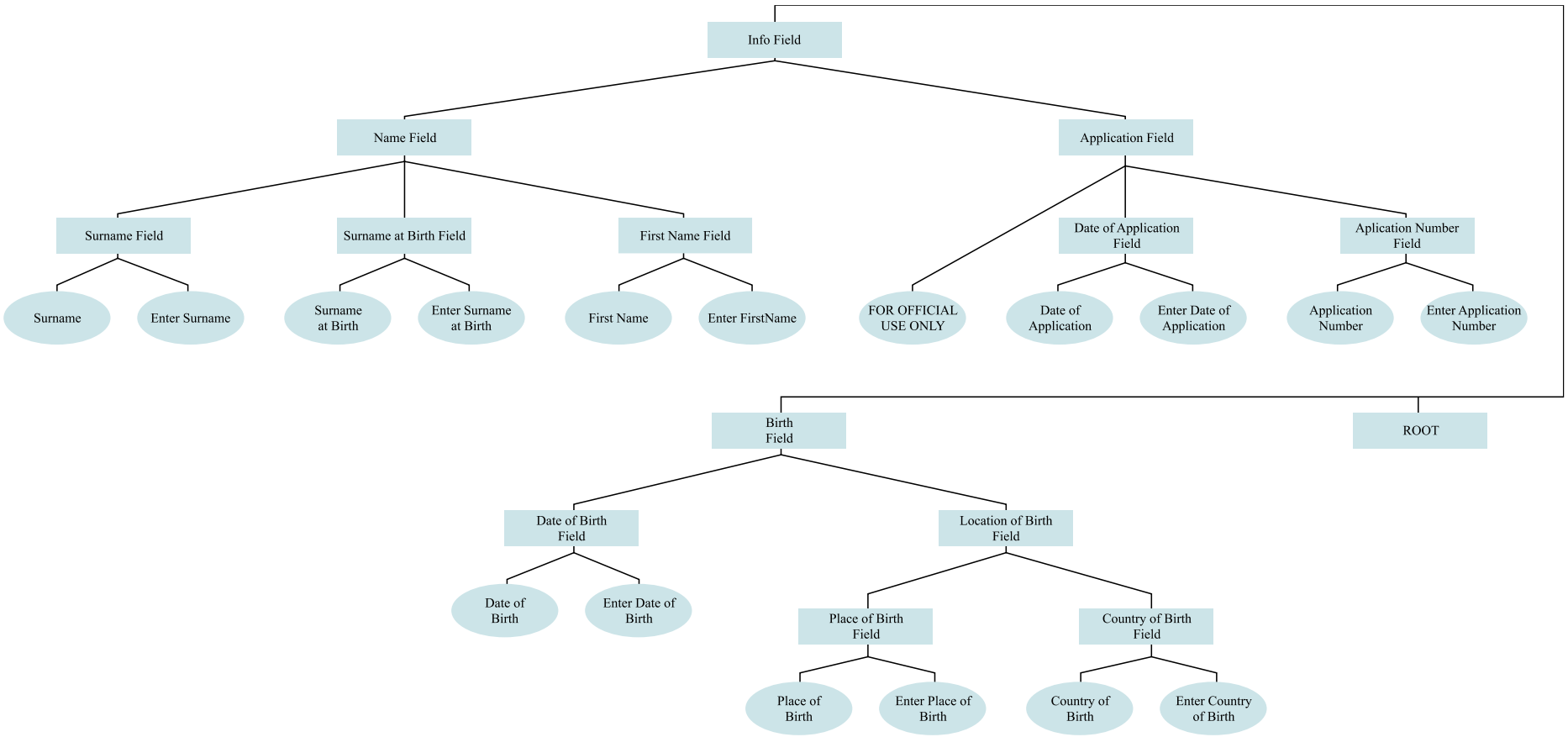
İşlemci: AMD Ryzen 5 1600 Six-Core Processor, 3,20 GHz

Ram: 16 GB

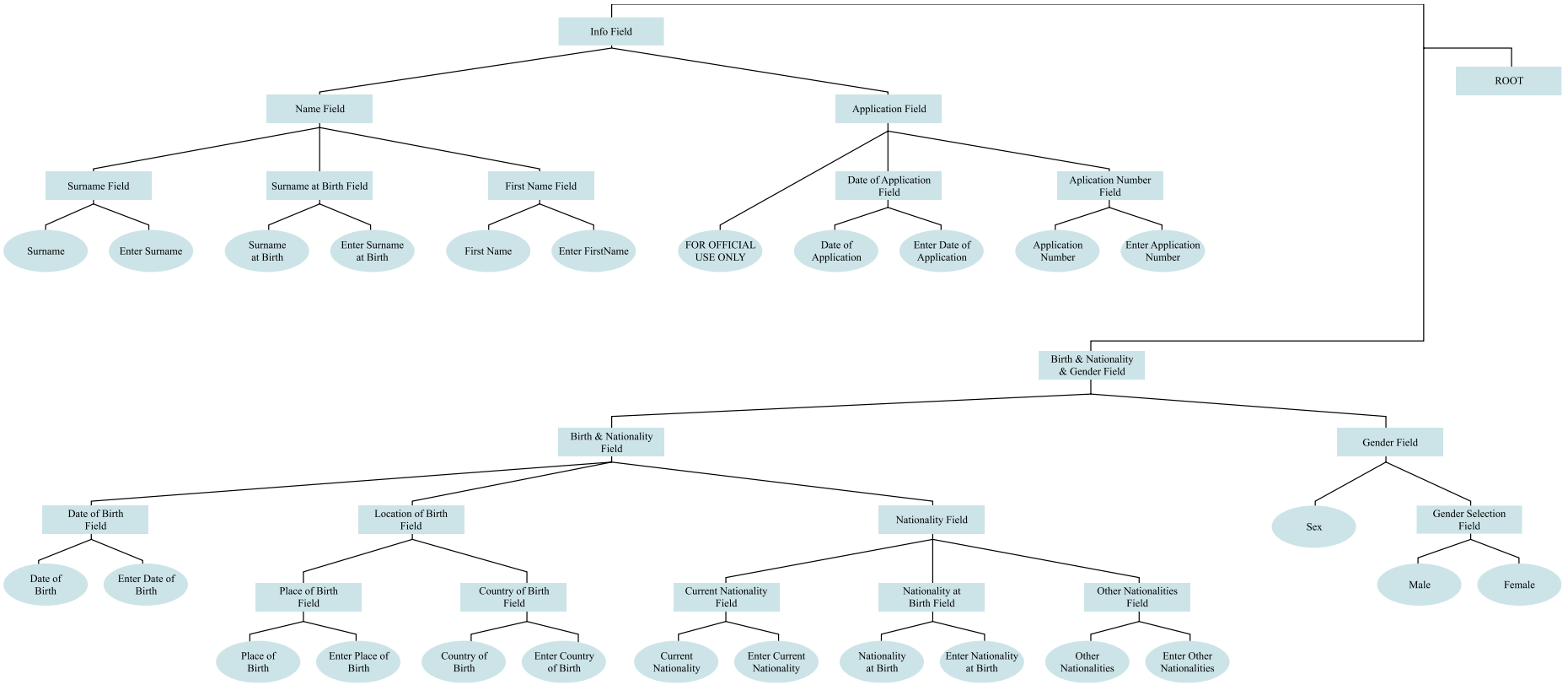
Ekran Kartı: NVIDIA GeForce GT 1030

7.1.1. Yerleşim Ağaçlarının Yapıları

Ölçümler, birincisi 17 bileşen ve 14 kapsayıcıya, ikincisi 26 bileşen ve 21 kapsayıcıya sahip olmak üzere 2 farklı yerleşim ağacı üzerinde yapılmıştır. Ağaçların yapısı *Şekil 7.1* ve *Şekil 7.2'de* detaylı bir şekilde gösterilmektedir. Şekillerde dörtgenler kapsayıcıları, elipsler bileşenleri temsil etmektedir. Her iki ağaç da 150x700, 360x600, 500x500 ve 1280x720 boyutlarında 4 farklı ekrana çizdirilmiştir. Sonuçlar *Tablo 7.1* ve *Tablo 7.2*'de gösterilmektedir.



Şekil 7.1 Yerleşim ağacı yapısı *(a)*.



Şekil 7.2 Yerleşim ağacı yapısı *(b)*.

7.1.2. Sonuçlar

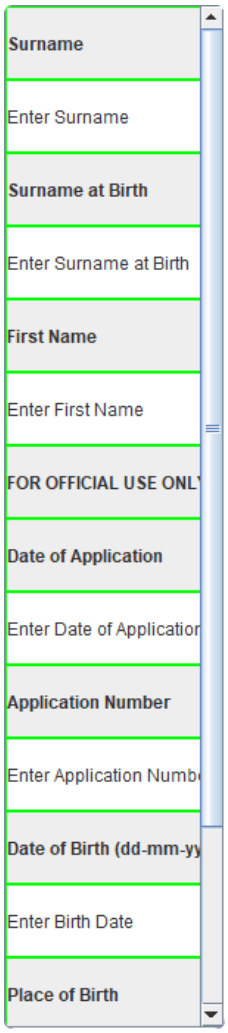
Önceki bölümde de belirtildiği gibi verilen her iki ağaç, 4 farklı ekran boyutunda denenmiş ve algoritma süreleri µs cinsinden verilmiştir. Tablolarda ayrıca algoritmanın en son hangi sınıflandırma için ve kaç tane yerleşim dizilimi bulduğu da paylaşılmıştır.

	150x700	**360x600**	**500x500**	**1280x720**
GETRANGES (µs)	23915	13113.3	14645.9	36584.3
ELIMINATELAYOUTS (µs)	884.9	1111.8	1297.9	27977.2
GETFINALLAYOUTS (µs)	1500.5	20608	12684.5	2837658.5
MEASUREAESTHETICS (µs)	32932.6	105065.1	58037.4	508338.9
Sınıflandırma	Dikey Kaydırma	Tam Sığan	Tam Sığan	Boşluk
Bulunan Yerleşim Dizilimleri	1	316	54	16300

Tablo 7.1. *Şekil 7.1*'deki yerleşim ağacının ekran boyutlarında çizdirilme ölçüleri.

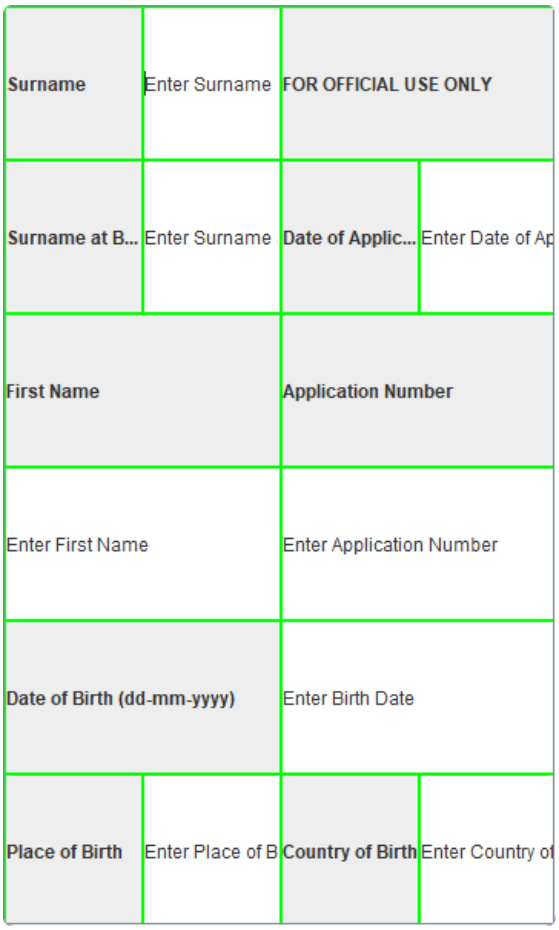
Tablo 7.1'de de görüldüğü gibi bulunan yerleşim dizilimi sayısı az iken `MeasureAesthetics` algoritması, `GetFinalLayouts` algoritmasına göre daha uzun zamanda tamamlanmaktır. Fakat yerleşim dizilimi sayısı arttıkça her iki algoritmanın çalışma süresinde de artış görülmekle birlikte `GetFinalLayouts` algoritmasının artış oranı daha fazla olmuş hatta saniyenin üzerine çıkmıştır. Bu durum, daha karmaşık yapıdaki yerleşim ağaçları için bulunan yerleşim dizilimi sayısı daha fazla olacağından olumsuz bir sonuç teşkil etmektedir.

Bulunan sonuçlara ait şekiller incelendiğinde tabloda verilen sınıflandırmalarla çizdirilen yerleşim dizilimlerinin uyumlu olduğu görülmektedir. Verilen yerleşim ağacı için 360x600 ve 500x500 ekran ölçüleri için tam sığan dizilimler bulunabilirken, 150x700 için dikey kaydırmalı ve 1280x720 için boşluklu dizilimler bulunmuştur. Şekillerde görülen yeşil çerçeve bileşenleri, turuncu çerçeve ise boşlukları ifade eder.

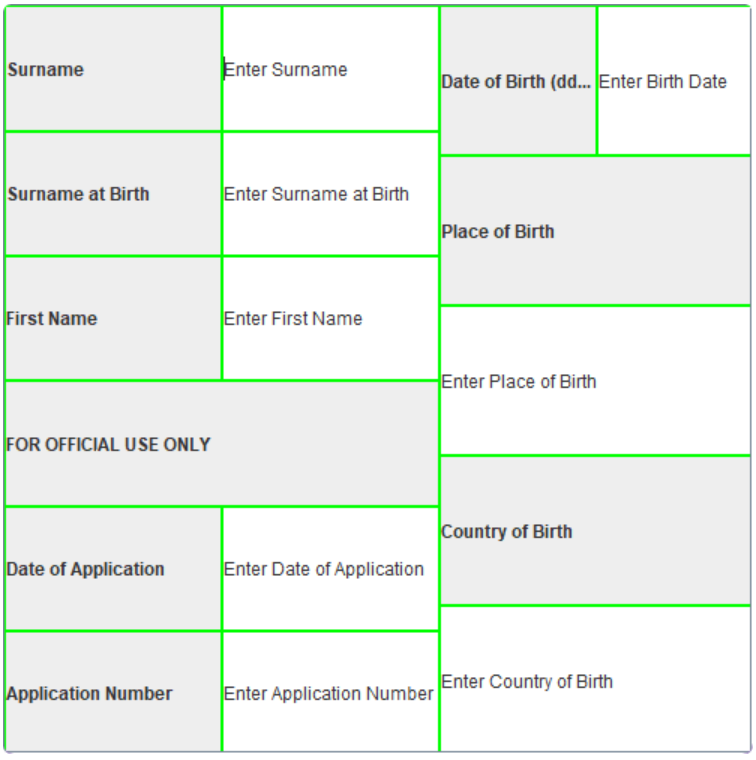


Şekil 7.3. *Şekil 7.1*'deki yerleşim ağacının 150x700 boyutundaki ekrana çizdirilmesi.

Şekil 7.4. *Şekil 7.1*'deki yerleşim ağacının 1280x 720 boyutundaki ekrana çizdirilmesi.



Şekil 7.5. *Şekil 7.1*'deki yerleşim ağacının 360x720 boyutundaki ekrana çizdirilmesi.



Şekil 7.6. *Şekil 7.1*'deki yerleşim ağacının 500x 500 boyutundaki ekrana çizdirilmesi.

	150x700	360x600	500x500	1280x720
GETRANGES (µs)	1364852.2	668657.1	649816.8	2416015.5
ELIMINATELAYOUTS (µs)	16907.5	64857.5	35037.4	2090857.7
GETFINALLAYOUTS (µs)	4778.9	68764.1	95969.2	Ölçülemedi
MEASUREAESTHETICS (µs)	3965.0	198862.4	303672.2	Ölçülemedi
Sınıflandırma	Dikey Kaydırma	Tam Sığan	Tam Sığan	Boşluk
Bulunan Yerleşim Dizilimleri	9	1971	5216	6174415

Tablo 7.2. *Şekil 7.2*'deki yerleşim ağacının farklı ekran boyutlarında çizdirilme ölçüleri.

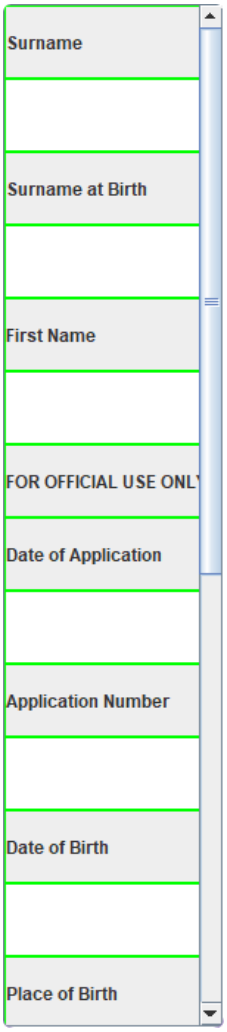
*Tablo 7.2'*ye göre bulunan yerleşim sayısı düşük olmasına rağmen 150x700 ekran boyutlarında `GetRanges` algoritması, saniyenin üzerine çıkmış, bulunan yerleşim sayısı düşük olan 360x600 ve 500x500 ekran boyutlarında ise saniyenin altında kalmıştır. Aynı zamanda sonuçlar Tablo *7.1* ile kıyaslandığında bütün ekran boyutları için `GetRanges` algoritmasının çalışma süresinin arttığı söylenebilir. Buradan `GetRanges` algoritmasının çalışma süresinin yerleşim ağacının karmaşıklığına bağlı olmakla beraber, süreyi tek etkileyen faktör olmadığı görülebilir. Sınıflandırma değişimi arttıkça algoritmanın çalışma süresi de artmıştır.

1280x720 ekran boyutlarına çizdirilmeye çalışılan yerleşim dizilimleri sayılarının çok fazla olması dolayısıyla hesaplanması tamamlanamamış ve ölçülememiştir. `GetRanges` ve `EliminateLayouts` gibi algoritmaların daha kuvvetli bir eleme yapması gereklidir, sonucu çıkarılabilir. Bu ekran ölçüsü her iki ağaç için de bir sonraki bölümde kırpma oranı girilerek tekrar test edilmiştir.

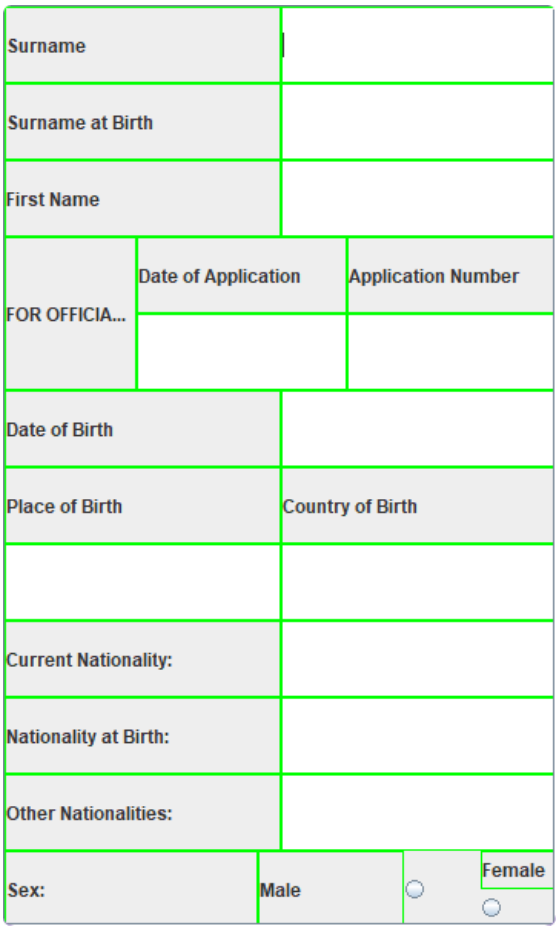
İki tablo karşılaştırıldığında sadece algoritma sürelerinde artış değil, karmaşıklık arttıkça bulunan yerleşim dizilimlerinde de artış görülmektedir. Karmaşıklığın artması özellikle tam sığan sınıflandırmadaki dizilimlerin sayısında azalmaya sebep olmamıştır.

Her iki yerleşim ağacına ait ekran görüntüleri incelendiğinde, aynı ebeveynin çocuğu olan kapsayıcıların, değerlerinin eş olmasına rağmen, kimilerinde sıralama yatay, kimilerinde ise dikey olduğu görülmektedir. *Şekil 7.9* incelendiğinde aynı kapsayıcının çocuğu olan *male* ve *female* kapsayıcıları (*Şekil 7.9'*da sağ altta görülmektedir) aynı kapsayıcının çocukları olmasına rağmen *male* kapsayıcısı yatay, *female* kapsayıcısı ise dikey hizalanmıştır. Bu

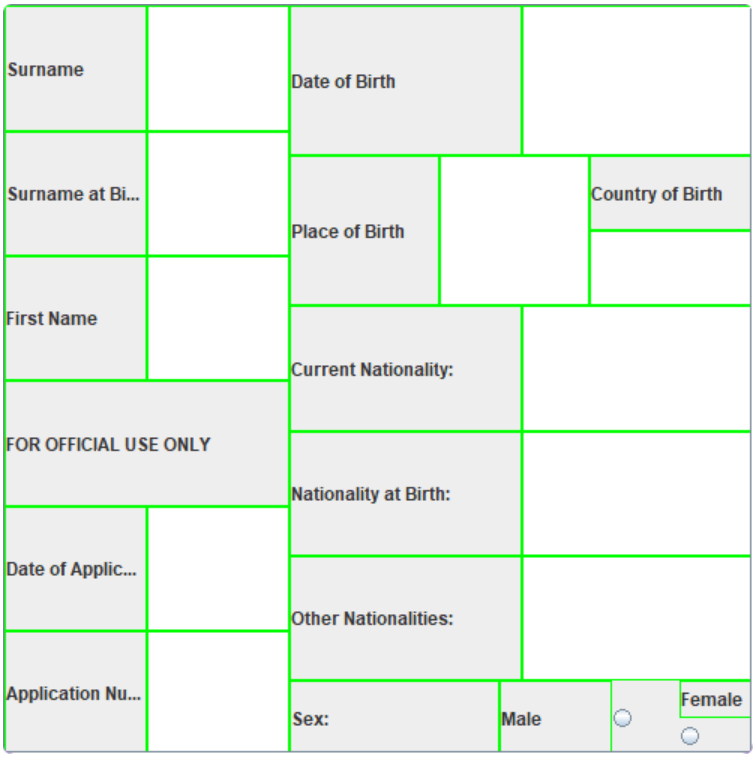
durum, estetik ölçümünün sadece bütün bir yerleşim dizilimi için değil, iç kapsayıcılar için de yapılması gerekliliğini ortaya koymuştur.



Şekil 7.7. *Şekil 7.2*'deki yerleşim ağacının 150x700 boyutundaki ekrana çizdirilmesi.



Şekil 7.8. *Şekil 7.2*'deki yerleşim ağacının 360x600 boyutundaki ekrana çizdirilmesi.



Şekil 7.9. *Şekil 7.2*'deki yerleşim ağacının 500x500 boyutundaki ekrana çizdirilmesi.

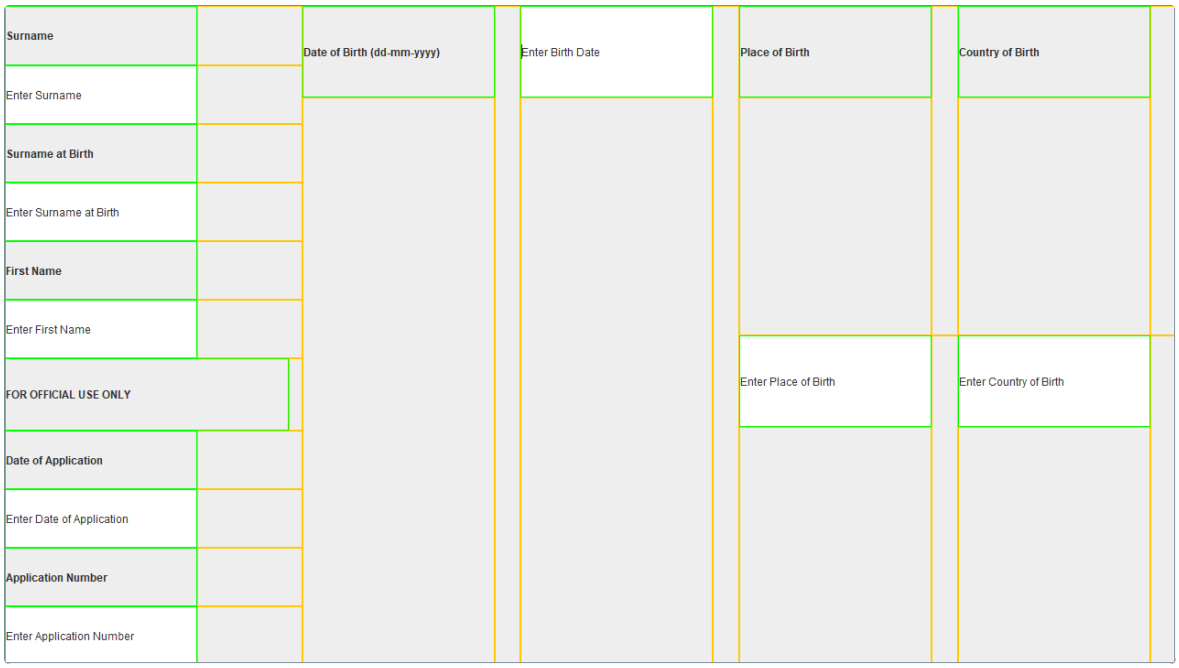
7.1.3. Kırpma Algoritması ile Sonuçlar

Tablo 7.2.'de de görüldüğü üzere bileşen ve kapsayıcı sayısı arttıkça bulunan yerleşim dizilimlerinin sayısı artmış bu da `GetFinalLayout` ve `MeasureAesthetic` algoritmalarının 1280x720 boyutlarındaki ekran için tamamlanamamasına sebep olmuştur. Dolayısıyla aynı testler boşluk kırpma oranı girilerek tekrar yapılmış ve sonuçlar *Tablo 7.3.*'de gösterilmiştir. Yapılan testlerde boşluk kırpma oranı 9 olarak alınmıştır. Bütün durumlarda sınıflandırıma bilgisi boşluklu olduğu için tabloya eklenmemiştir. Bunun yerine tabloya her son çizdirilen yerleşim dizilimi için hesaplanan estetik ölçümü eklenmiştir.

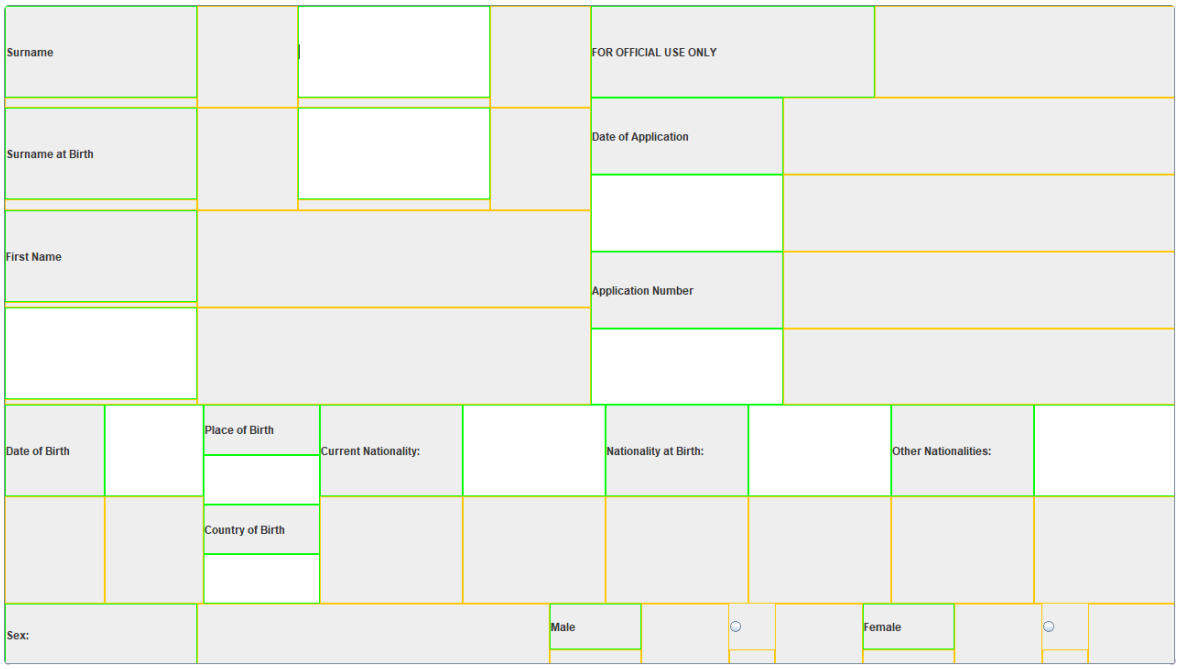
	***Şekil 7.1*'de verilen yerleşim ağacı**		***Şekil 7.2*'de verilen yerleşim ağacı**	
	Kırpma oranı yok	**Kırpma oranı: 9**	**Kırpma oranı yok**	**Kırpma oranı: 9**
GETRANGES (µs)	36584.3	36463.3	2416015.5	2290657
ELIMINATELAYOUTS (µs)	27977.2	43197.9	2090857.7	2056591.1
GETFINALLAYOUTS (µs)	2837658.5	24207.9	Ölçülemedi	13120937
MEASUREAESTHETICS (µs)	508338.9	50345.8	Ölçülemedi	1962456
Bulunan Yerleşim Dizilimleri	16300	24	6174415	78692
Estetik Ölçüm	0.72510004	0.5930564	Ölçülemedi	0.6472422

Tablo 7.3. *Şekil 7.1 ve Şekil 7.2*'de verilen yerleşim ağaçlarının boşluk kırpma oranı ile 1280x720 boyutlarındaki ekranda ölçüleri.

`EliminateLayouts` algoritmasına uygulanan boşluk kırpma oranı ile test aşamasının ilk etabında çizdirilemeyen yerleşim dizilimi için uygun bir ekran görüntüsü alınabilmiştir. Fakat bulunan yerleşim dizilimlerinde göz ardı edilemeyecek kadar büyük bir düşüş olmasına rağmen, algoritmaların çalışma hızı saniyenin altına düşememiştir. Üstelik tabloda ve *Şekil 7.10*'da görüldüğü üzere arayüz estetiğinde gözle görünür bir düşüş olmuştur.



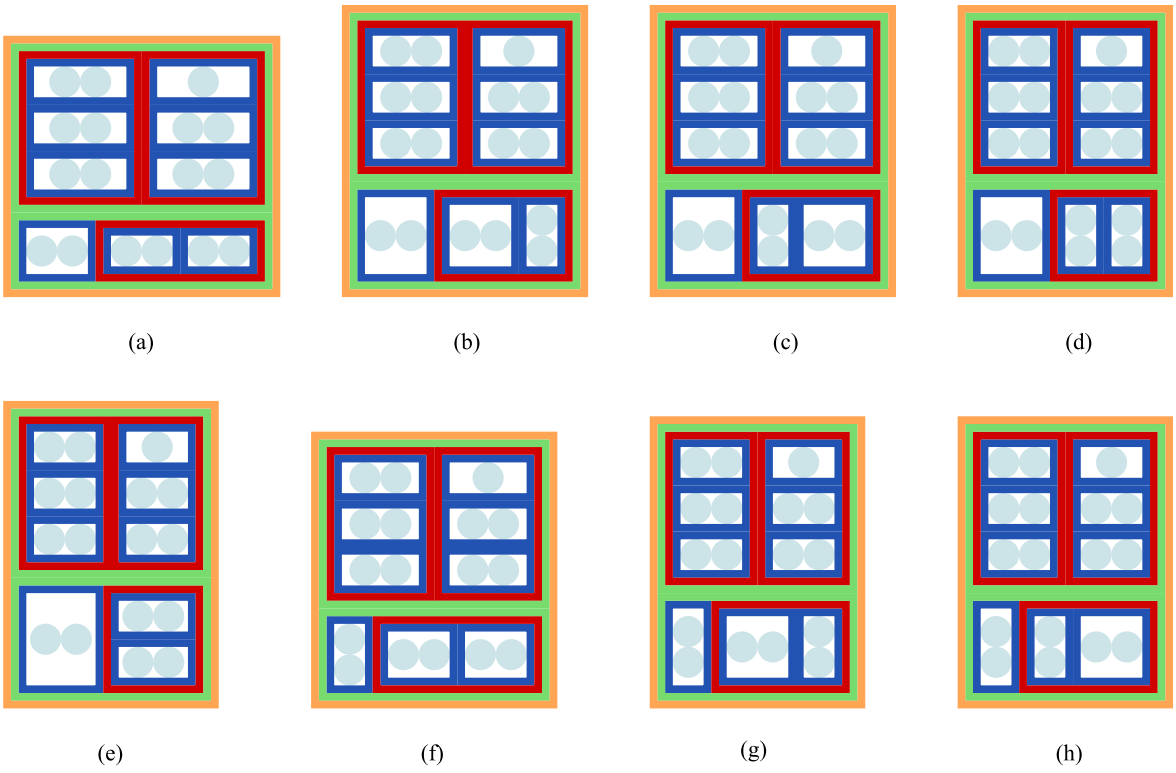
Şekil 7.10. *Şekil 7.1*'deki yerleşim ağacının boşluk kırpma oranı ile 1280x720 boyutlarındaki ekrana çizdirilmesi.



Şekil 7.11. *Şekil 7.2*'deki yerleşim ağacının boşluk kırpma oranı ile 1280x720 boyutlarındaki ekrana çizdirilmesi.

7.2. Ağırlıklı / Ağırlıksız Estetik Ölçüm Karşılaştırması

6. bölümde anlatıldığı üzere estetik ölçümün son metriği düzen ve karmaşıklık, her bir metriğin ağırlıklı olarak ortalamalarının alınması ile bulunur. Bu test senaryosunda Şekil 7.1'de bulunan yerleşim ağacı, 720x400 boyutlarındaki bir ekrana 2 farklı ağırlık katsayı grubu ile çizdirilmiştir. İlk test senaryosunda 6. bölümde düzen ve karmaşıklık formülünde de verilen denge için *1.0*, eşitlik için *0.8*, simetri için *0.6*, sıralanma için *0.2*, bağıntı için *0.5*, birlik için *0.4*, orantı için *0.4*, sadelik için *0.6*, yoğunluk için *0.4*, düzenlilik için *0.1*, ekonomi için *0.5*, homojenlik için *0.1*, ritim için *0.2*, ikinci senaryoda ise her bir katsayı *1* olacak şekilde alınmıştır. İlk test senaryosuna ağırlıklı, ikinci test senaryosuna ise ağırlıksız estetik ölçüm denmektedir. İki test senaryosunda da bulunan yerleşim dizilimi sayısı 8'dir ve bu sonuç estetik ölçüme bağlı olarak değişmemektedir. *Şekil 7.13*'te bulunan 8 yerleşim diziliminin sıralama yapısı gösterilmektedir.



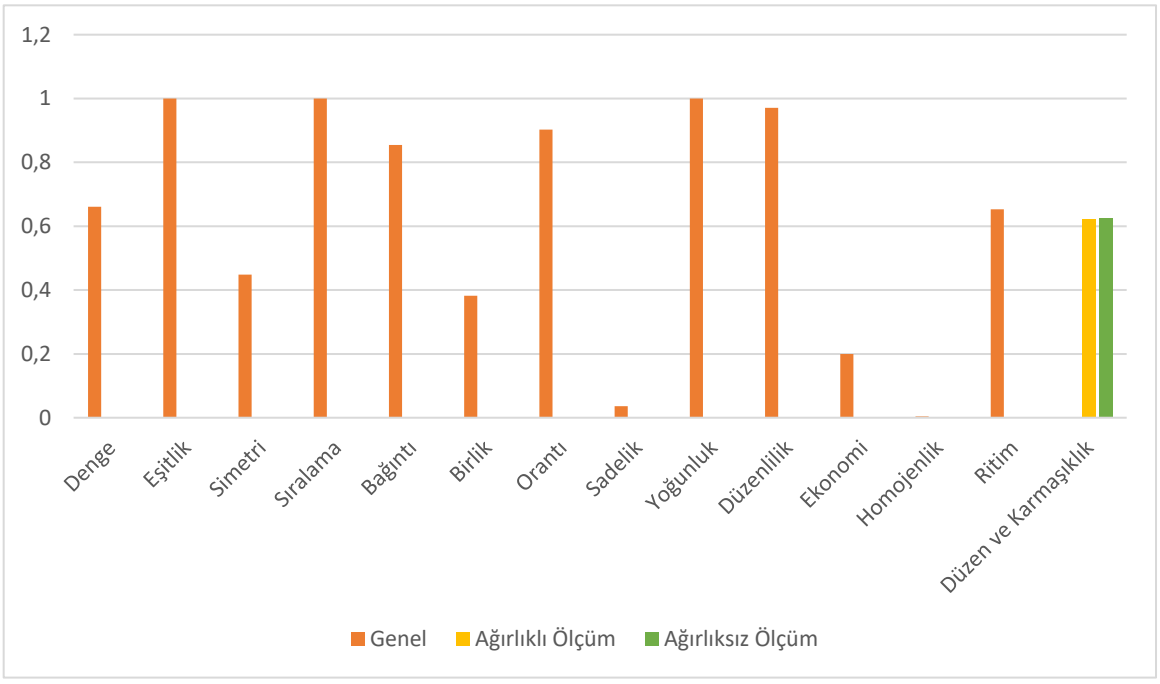
Şekil 7.12. *Şekil 7.1*'deki yerleşim ağacının 720x400 boyutundaki ekrana çizdirilmesi ile bulunan tam sığan 8 dizilimin hizalama yapısı.

Test sonunda ağırlıklı ve ağırlıksız estetik ölçümün karar verdiği son dizilim aynıdır (*Şekil 7.13, Şekil 7.14*). 8 yerleşim diziliminin her biri için hesaplanan estetik ölçümler *Tablo 7.4*'te gösterilmiştir.

	Dizilim							
	a	**b**	**c**	**d**	**e**	**f**	**g**	**h**
Test (1)	0.6174	0.6111	0.6066	0.6202	0.6203	0.6041	0.6041	0.6123
Test (2)	0.6192	0.6149	0.6097	0.6241	0.6241	0.6092	0.6079	0.6156

Tablo 7.4. *Şekil 7.1 Test (1),* ağırlıklı estetik ölçüm *ve Test (2),* ağırlıksız estetik ölçüme göre sonuçları.

Yapılan testler *Şekil 7.1*'de bulunan yapının karmaşıklığında bir yerleşim ağacı için ağırlıklı estetik ölçüm sonuçlarının büyük ölçüde değişiklik göstermediğini ortaya koymuştur. Bunun sebebi, karmaşıklık arttıkça bulunan dizilimlerin estetik ölçümünün azalması ve farklı dizilimler arasında çok küçük ölçüm farklılıkları gözlenmesi olarak gösterilebilir. Bu sonuçta bulunan yerleşim dizilimlerinin benzer yapıda olmasının da etkisi vardır.



Şekil 7.13. *Estetik ölçüm detaylı sonuçlar.*

Surname	Enter Surname	**FOR OFFICIAL USE ONLY**	
Surname at Birth	Enter Surname at Birth	**Date of Application**	Enter Date of Application
First Name	Enter First Name	**Application Number**	Enter Application Number
Date of Birth (dd-mm-yyyy)	Enter Birth Date	**Place of Birth**	**Country of Birth**
		Enter Place of Birth	Enter Country of Birth

Şekil 7.14. *Şekil 7.1*'deki yerleşim ağacının ağırlıklı estetik ölçüm ile 720x400 boyutundaki ekrana çizdirilmesi.

Surname	Enter Surname	**FOR OFFICIAL USE ONLY**	
Surname at Birth	Enter Surname at Birth	**Date of Application**	Enter Date of Application
First Name	Enter First Name	**Application Number**	Enter Application Number
Date of Birth (dd-mm-yyyy)	Enter Birth Date	**Place of Birth**	**Country of Birth**
		Enter Place of Birth	Enter Country of Birth

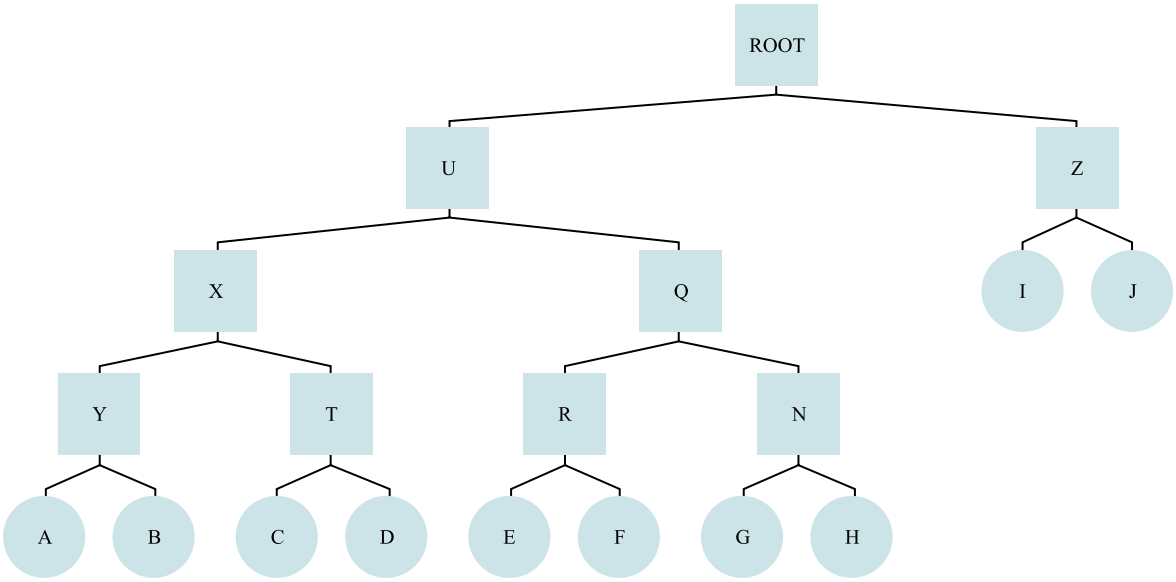
Şekil 7.15. *Şekil 7.1*'deki yerleşim ağacının ağırlıksız estetik ölçüm ile 720x400 boyutundaki ekrana çizdirilmesi.

7.3. Hizalama Testi

Hizalama, bileşenlerin yatayda ve dikeyde sağa yaslı, sola yaslı ve ortalanma durumuna denir. Testte *Şekil 7.16*'da verilen yerleşim ağacının yatay-dikey hizalamaları, sırasıyla; başlangıç-başlangıç, başlangıç-orta, başlangıç-son, orta-başlangıç, orta-orta, orta-son, son-başlangıç, son-orta, son-son olmak üzere dokuz farklı hizalamaya göre elde edilen sonuçlar alınmış ve her bir sonucun hesapladığı estetik ölçümler Tablo 7.5'te paylaşılmıştır. Test bulunan yerleşim dizilimlerinde boşluk ihtiyacını ortaya çıkarmak için 640x640 boyutlarında ekran için yapılmıştır.

7.3.1. Kullanılan Yerleşim Ağacı Yapısı

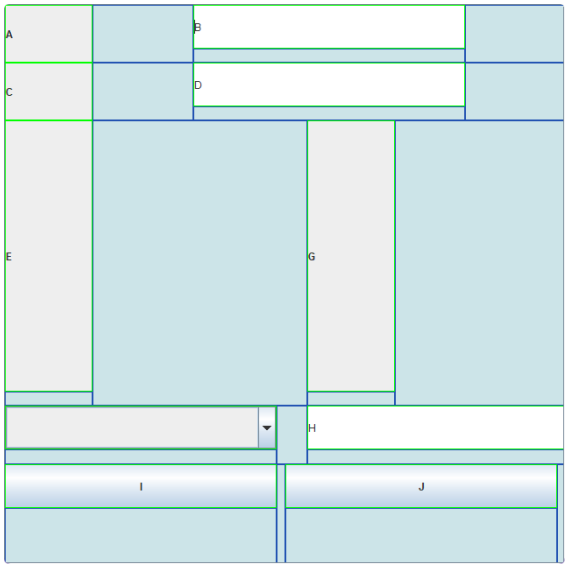
Ölçümler, 10 bileşen ve 9 kapsayıcıya sahip bir yerleşim ağacı kullanılmıştır. Ağacın yapısı *Şekil 7.16*'da gösterilmektedir. Şekillerde dörtgenler kapsayıcıları, elipsler bileşenleri temsil etmektedir. Yerleşim ağacı 640x640 boyutlarında bir ekrana çizdirilmiş ve bu ölçüler için `GetRanges` algoritmasından boşluklu sınıflandırma için 512 farklı yerleşim dizilimi bulunmuştur.



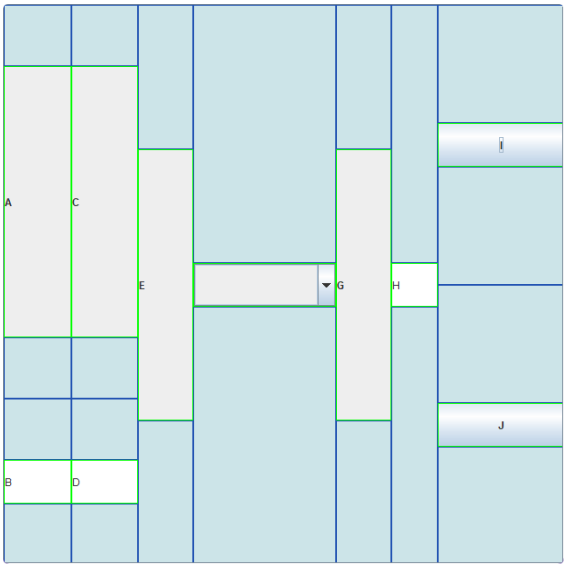
Şekil 7.16. Hizalama testi yerleşim ağacı yapısı.

7.3.2. Sonuçlar

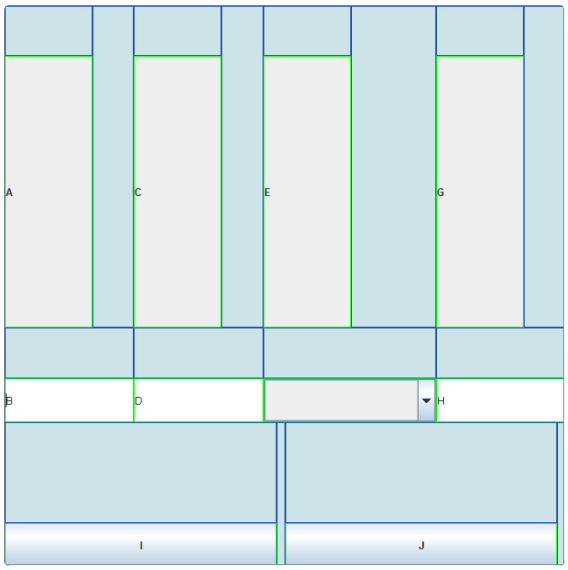
Yapılan testte elde edilen yerleşimlerin ekran görüntüleri *Şekil 7.17(a, b, c, d, e, f, g, h, i)*'de gösterilmektedir. Algoritmanın bu yerleşimlerde ölçtüğü estetik ölçümler *Tablo 7.5*'te verilmiştir.



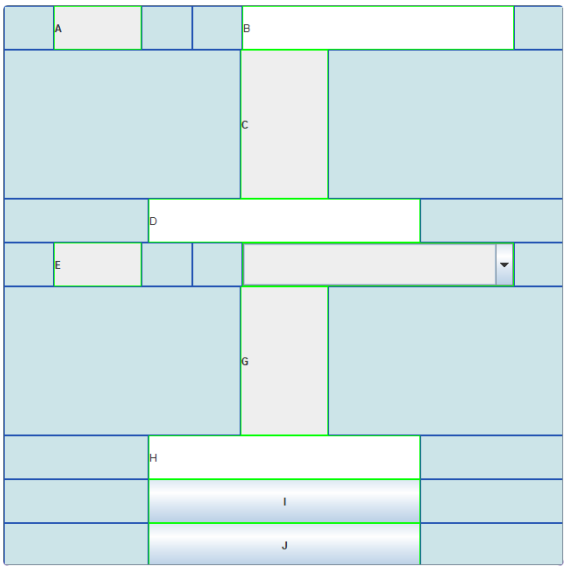
Şekil 7.17.(a) Yatay hizalama başlangıç, dikey hizalama başlangıç olan dizilim.



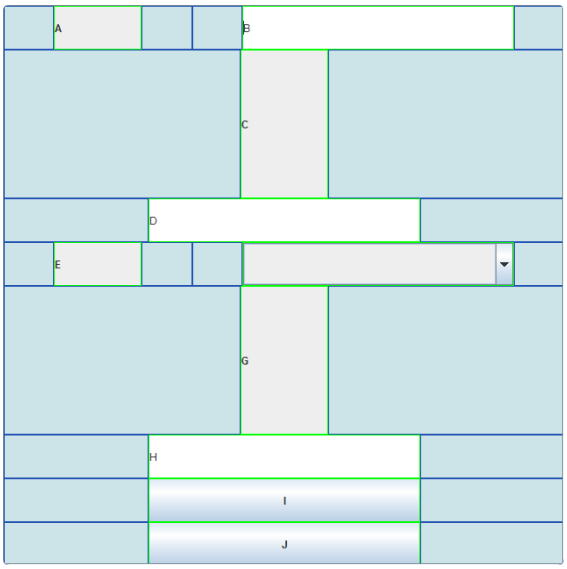
Şekil 7.17.(b) Yatay hizalama başlangıç, dikey hizalama orta olan dizilim.



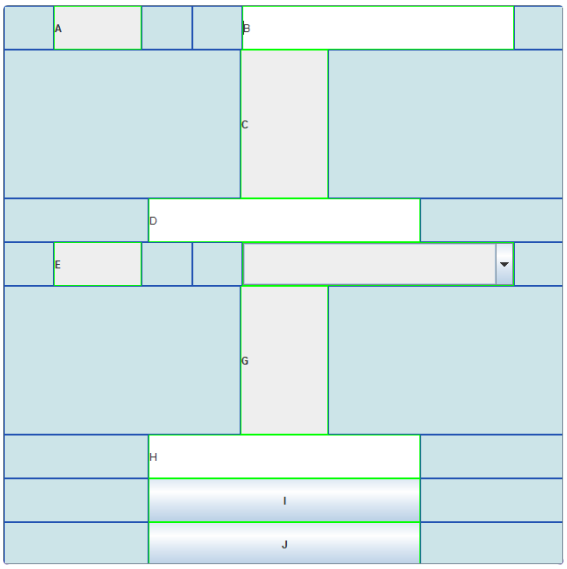
Şekil 7.17.(c) Yatay hizalama başlangıç, dikey hizalama son olan dizilim.



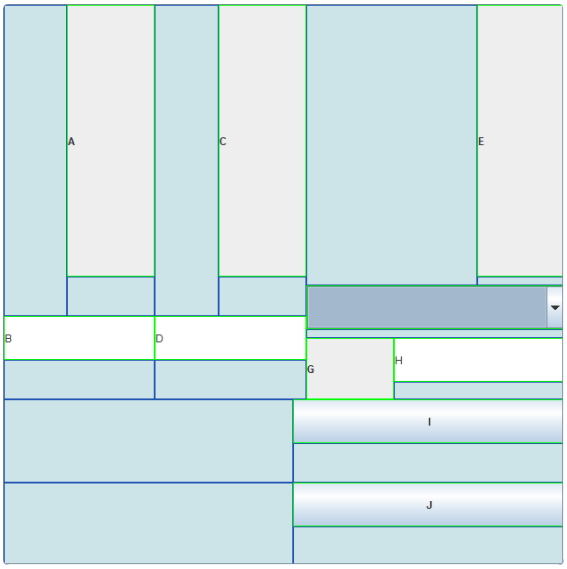
Şekil 7.17.(d) Yatay hizalama orta, dikey hizalama başlangıç olan dizilim.



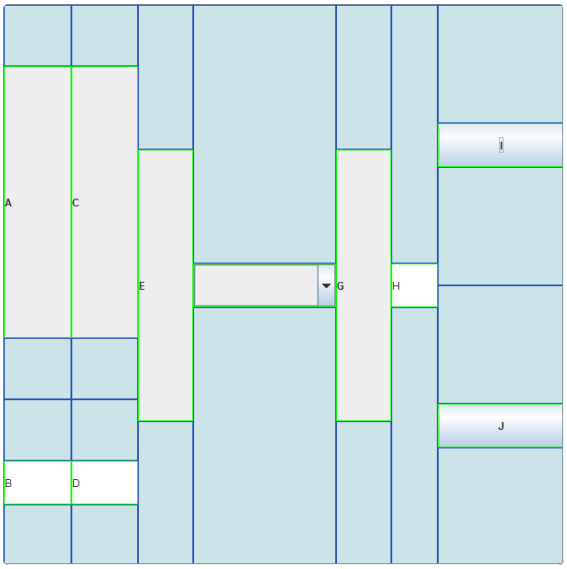
Şekil 7.17.(e) Yatay hizalama orta, dikey hizalama orta olan dizilim.



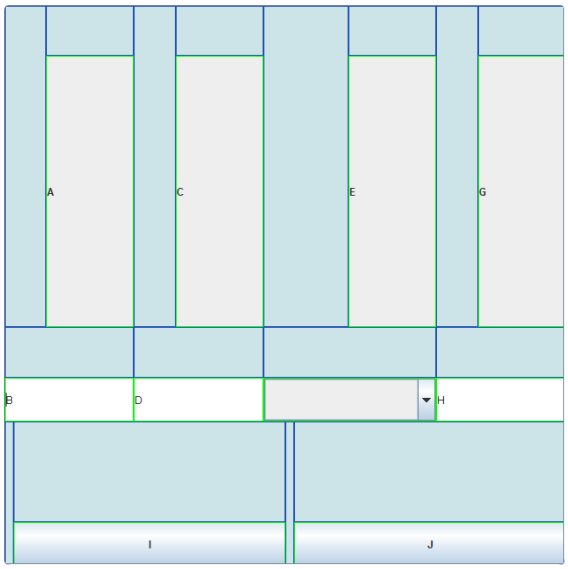
Şekil 7.17.(f) Yatay hizalama orta, dikey hizalama son olan dizilim.



Şekil 7.17.(g) Yatay hizalama son, dikey hizalama başlangıç olan dizilim.



Şekil 7.17.(h) Yatay hizalama son, dikey hizalama orta olan dizilim.



Şekil 7.17.(i) Yatay hizalama son, dikey hizalama son olan dizilim.

Sonuçlar incelendiğinde algoritmanın estetik ölçüm olarak en yüksek değerini yatay hizalama orta ve en yüksek ikinci değerini dikey hizalama orta yerleşim dizilimlerinde aldığı görünmektedir. *(d)*, *(e)* ve *(f)* yerleşim dizilimleri birbirinin aynısı olmakla birlikte, estetik ölçümleri 0.7167 olmuştur. Benzer şekilde, *(b)* ve *(h)* yerleşim dizilimleri aynı ve estetik ölçümleri 0.7114 olmuştur. En yüksek ölçüm hizalama orta olan dizilimlerde yapılmasına rağmen diğer dizilimlerle aralarında büyük farklar gözlenmemiştir.

B-B(a)	**B-O(b)**	**B-S(c)**	**O-B(d)**	**O-O(e)**	**O-S(f)**	**S-B(g)**	**S-O(h)**	**S-S(i)**
0.6145	0.7114	0.6264	0.7167	0.7167	0.7167	0.6291	0.7114	0.6371

Tablo 7.5. *Şekil7.17(a, b, c, d, e, f, g, h, i)*'de verilen yerleşimlerin estetik ölçüleri.

8. YORUM

Yapılan testler sonucunda görülmüştür ki, her ne kadar kırpma oranları ve sınıflandırmalarla bulunan yerleşim oranı sayısı azaltılsa da bileşen sayısı arttıkça ve yerleşim ağacının yapısı karmaşıklaştıkça bu azalma yeterli olmamıştır. Hesaplama sürelerinin saniyenin üstüne çıktığı durumlarda çalışma anında kullanıcının beklemesine sebep olacak ve kullanıcı deneyimi olumsuz eklenecektir. Dolayısıyla algoritmayı daha verimli kullanabilmek için daha hassas kırpmalar yapılması gereklidir. Ayrıca `GetRanges` algoritması çalışırken sınıflandırma dışında farklı bir eleme yöntemi eklenmesi uygun olur. Özellikle ağırlıklı ve ağırlıksız estetik ölçüm testinde ortaya çıkan birbirine çok yakın olan bazı yerleşim dizilimlerini elemek çözüm olabilir.

Estetik ölçüm testi sonuçlarından anlaşıldığı üzere, ölçüm metriklerine sadece ağırlık vermek sağlıklı bir seçim yapmada yeterli olmamıştır. Sonuçlarda çok ufak oynamalar olmakla birlikte genel olarak karmaşık yapılı yerleşim ağaçlarında birbirine benzeyen çok fazla yerleşim dizilimi olduğu için birbirine yakın estetik ölçümler saptanmıştır. `GetRanges` algoritması çalışırken bileşenlere yönelik metrikler kullanılarak elemeler yapılması bu durumun önüne geçebilir.

Estetik ölçümde karşılaşılan diğer bir sorun ise bu metrik ve formüllerin genel olarak tek sayfa web sitesi tasarımlarına yönelik olmasıdır. Mobil veya tablet gibi cihazlar için güncel metriklere ihtiyaç duyulmaktadır.

Bunların yanı sıra `GetFinalLayouts` algoritmasının pahalı olması ve her bir bulunan yerleşim diziliminde estetik ölçüm için çalışmak zorunda olması, bu çalışmadaki en büyük yadsınamaz sorundur. Yapılan testlerde bu problem son arayüze ulaşmayı engellemiştir. `GetFinalLayouts` algoritması ve estetik ölçümü birbirinden ayırabilecek bir yöntem bulunması soruna çözüm olabilir.

Son olarak, tüm bunların yanı sıra testlerde var olmayan bir problem olan `ExtendedArray` ve `ExtendedArrayList` sınıflarının karmaşıklık arttıkça boyutlarının üstel olarak artması sebebiyle hafıza çok fazla yer kaplanmasına sebep olmaktadır. Bu durumu engellemek için hem hafızada hangi alt aralıkların hesaplandığını tutabilecek hem de kabul edilen yerleşim dizilimlerini bir liste halinde tutabilecek bir yöntem geliştirilmelidir.

9. KAYNAKLAR

[1] Krzysztof Gajos and Daniel S Weld. Supple: automatically generating user interfaces. Proceedings of the 9th international conference on Intelligent user interfaces, Sayfa 93–100. **2004**.

[2] Jean-S´ebastien Sottet, Ga¨elle Calvary, and Jean-Marie Favre. Models at runtime for sustaining user interface plasticity. In Models@ run. time workshop (in conjunction with MoDELS/UML 2006 conference). **2006**.

[3] Dirk Roscher, Grzegorz Lehmann, Veit Schwartze, Marco Blumendorf, and Sahin Albayrak. Dynamic distribution and layouting of model-based user interfaces in smart environments. In Model-Driven Development of Advanced User Interfaces, Sayfa 171–197. Springer, **2011**.

[4] Barış Çelik, Burkay Genç. A Screen Resolution Sensitive Intelligent Interface Layout Approach. Mühendislik Bilimleri ve Tasarım Dergisi 8(5), Sayfa 113-125, **2020**

[5] Dejan Todorovic. Gestalt principles. Scholarpedia, 3(12):5345, **2008**.

[6] Clemens Zeidler, Christof Lutteroth, and Gerald Weber. Constraint solving for beautiful user interfaces: how solving strategies support layout aesthetics. In Proceedings of the 13th International Conference of the NZ Chapter of the ACM's Special Interest Group on Human-Computer Interaction, Sayfa 72–79. **2012**.

[7] David Chek Ling Ngo, Lian Seng Teo, and John G Byrne. A mathematical theory of interface aesthetics. In Visual mathematics, Cilt 2. Mathematical Institute SASA, **2000**.

[8] David Chek Ling Ngo and John G Byrne. Another look at a model for evaluating interface aesthetics. International Journal of Applied Mathematics and Computer Science, 11:515–535, **2001**.

[9] Altaboli, A. and Y. Lin (2011, January). Investigating effects of screen layout elements on interface and screen design aesthetics. Adv. in Hum.-Comp. Int., 5:1–5:10. **2011**

[10] Mathieu Zen and Jean Vanderdonckt. Towards an Evaluation of Graphical User Interfaces Aesthetics based on Metrics. IEEE Eighth International Conference on Research Challenges in Information Science (RCIS). **2014**.

[11] Mathieu Zen and Jean Vanderdonckt. Assessing user interface aesthetics based on the

inter-subjectivity of judgment. In Proceedings of the 30th International BCS Human Computer Interaction Conference: Fusion!, sayfa 25. BCS Learning & Development Ltd., **2016**.

[12] Fiora T. W. Au, Simon Baker, Ian Warren, Gillian Dobbie. Automated usability testing framework. Proceedings of the ninth conference on Australasian user interface - Cilt 76, Sayfa 55–64. Ocak **2008**.

[13] Jan Hartmann, Alistair Sutcliffe, Antonella De Angeli. Towards a theory of user judgment of aesthetics and user interface quality. ACM Transactions on Computer-Human Interaction Cilt 15, Sayı 4, Makale No.: 15, Sayfa 1–30. Kasım **2008**.

[14] Mário Simões-Marques, Isabel L. Nunes. Usability of Interfaces. Ergonomics - A Systems Approach, Chapter: Usability of Interfaces. **2012**.

[15] Andreas Riegler, Clemens Holzmann. Measuring Visual User Interface Complexity of Mobile Applications With Metrics. Interacting with Computers. 30. 207-223. Mayıs **2018**.

[16] Helen C. Purchase, John Hamer, Adrian Jamieson, Oran Ryan. Investigating Objective Measures of Web Page Aesthetics and Usability. Conferences in Research and Practice in Information Technology Series. 117. 19-28. **2011**.

[17] Marie-Luce Bourguet. Metrics-Based Evaluation of Graphical User Interface Aesthetics: The Segmentation Problem. 31-38. Kasım **2018**.